KASKBOOK

Uczenie maszynowe i systemy rozproszone

Praca zbiorowa pod redakcją
dr. hab. inż. Juliana Szymańskiego

Gdańsk 2021

PRZEWODNICZĄCY KOMITETU REDAKCYJNEGO
WYDAWNICTWA POLITECHNIKI GDAŃSKIEJ
Dariusz Mikielewicz

REDAKTOR PUBLIKACJI NAUKOWYCH
Michał Szydłowski

RECENZENCI
Jacek Rak
Marcin Woźniak

SKŁAD I PROJEKT OKŁADKI
Szymon Olewniczak

Wydano za zgodą
Rektora Politechniki Gdańskiej

Oferta wydawnicza Politechniki Gdańskiej jest dostępna pod adresem
https://www.sklep.pg.edu.pl

ISBN 978-83-7348-847-2

WYDAWNICTWO POLITECHNIKI GDAŃSKIEJ

Wyd. I. Ark. wyd. 18,3, ark. druku 19,25, 254/1156

Przedmowa

Jedenasta edycja monografii naukowej KASKBOOK Katedry Architektury Systemów Komputerowych, Wydziału Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej poświęcona jest zagadnieniom uczenia maszynowego i systemom rozproszonym. Zawiera ona opis zagadnień teoretycznych, jak i wyniki badań zrealizowanych przez pracowników i współpracowników Katedry. Znalazły się tu przede wszystkim aplikacje nowoczesnych architektur sieci neuronowych oraz tradycyjnych podejść do uczenia maszyn w bardzo szerokim spektrum zastosowań: od przetwarzania tekstu, dźwięku i obrazu do teoretycznych rozważań nad możliwością uczenia środowisk logicznych. W monografii opisano również metody stosowane w programowaniu środowisk współbieżnych, a także zastosowania nowoczesnych systemów rozproszonych. W książce zamieszczono również ogólne rozważania nad jakością wytwarzania i licencjonowaniem aplikacji informatycznych.

W tym miejscu chciałbym podziękować osobom, dzięki którym powstała ta monografia: mgr. inż. Szymonowi Olewniczakowi za skład tekstu i Mateuszowi Olszewskiemu za projekt okładki. Szczególne podziękowania składam dr. hab. inż. Marcinowi Woźniakowi z Politechniki Śląskiej i dr. hab. inż. Jackowi Rakowi z Politechniki Gdańskiej za recenzje i uwagi, dzięki którym udało zachować się wysoki poziom redakcyjny i merytoryczny tego wydania.

dr hab. inż. Julian Szymański

Spis treści

1. Neuronowe modele z atencją w przetwarzaniu języka naturalnego

Szymon Olewniczak

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
szymon.olewniczak@pg.edu.pl

Streszczenie
Celem niniejszego rozdziału jest wprowadzenie w tematykę sieci neuronowych z atencją oraz ich zastosowań w przetwarzaniu języka naturalnego. Rozdział skupia się w szczególności na dokładnym omówieniu architektury modelu Transformer, wykorzystującego atencję jako podstawowy mechanizm swojego działania.

Słowa kluczowe: przetwarzanie języka naturalnego, rekurencyjne sieci neuronowe, atencja w sieciach neuronowych, Transformer

1.1 Wprowadzenie

Przetwarzanie języka naturalnego (ang. *natural language processing*) stanowi bardzo szeroką dziedzinę informatyki. W jej skład wchodzi wiele różnorodnych zagadnień, od stosunkowo prostych takich jak np. korekta literówek, czy wyszukiwanie dokładne, po bardzo złożone jak np. tłumaczenie maszynowe, automatyczne streszczenia, czy odpowiadanie na pytania.

Obecnie za każdym razem kiedy wpisujemy jakieś hasło w wyszukiwarkę internetową, odbieramy pocztę elektroniczną, czy logujemy się do portali społecznościowych, korzystamy z różnych rozwiązań wchodzących w skład przetwarzania języka naturalnego. Co więcej, systemy reklamy spersonalizowanej, stanowiące podstawową formę zarabiania dla firm takich jak Google czy Facebook, również opierają się na różnych technikach i algorytmach wchodzących w skład omawianej dziedziny. Nie bez przesady można więc stwierdzić, że NLP stanowi obecnie jedną z ważniejszych, jeśli nie najważniejszą dziedzinę informatyki.

W rozwoju przetwarzania języka naturalnego można wyróżnić na trzy ważne etapy. Historycznie pierwszy stanowią systemy bazujące na regułach (ang. *rule-based*), w których starano się rozwiązać problemy z zakresu NLP przy wykorzystaniu ręcznie definiowanych reguł przetwarzania, co jednak nie dawało zadowalających rezultatów. Drugi etap rozwoju NLP stanowi uczenie płytkie (ang. *shallow machine learning*), w którym ręczne reguły zastąpiono modelami podejmującymi decyzje na podstawie wiedzy płynącej z przykładów. Podejście to

pozwoliło osiągnąć znacznie lepsze rezultaty, jednak prawdziwą rewolucją w dziedzinie okazało się wykorzystanie głębokich sieci neuronowych (ang. *deep neural networks*), których pojawienie się rozpoczęło trzeci etap rozwoju systemów NLP.

Do niedawna największą popularnością w przetwarzaniu języka naturalnego przy wykorzystaniu głębokich sieci neuronowych cieszyły się sieci rekurencyjne (ang. *recurrent neural networks*). Pomimo swoich licznych zalet, posiadały jedną zasadniczą wadę – niemożliwość zrównoleglenia ich treningu. W efekcie przekładało się to na ograniczenie wielkości sieci i możliwych do przetworzenia przykładów treningowych. Jednak wraz z pojawieniem się architektury sieci Transformer, która posiadała zalety sieci rekurencyjnych, jednocześnie umożliwiając trening równoległy, sytuacja ta uległa zasadniczej zmianie.

W niniejszym opracowaniu chciałbym pokrótce przedstawić koncepcję wykorzystania sieci neuronowych z atencją w przetwarzaniu języka naturalnego, ze szczególnym uwzględnieniem architektury Transformer. Modele neuronowe z atencją stanowią obecnie najlepsze znane rozwiązanie dla bardzo wielu problemów NLP oraz są nieustannie udoskonalane przez wiele zespołów badawczych na całym świecie. Praca ma na celu wyjaśnienie podstaw działania modeli neuronowych z atencją, aby umożliwić czytelnikowi wykorzystywanie ich w praktyce oraz stworzyć podstawę dla dalszych własnych poszukiwań.

1.2 Przetwarzanie języka naturalnego

Zanim przejdziemy do omówienia wybranych architektur sieci neuronowych służących przetwarzaniu języka naturalnego, chciałbym pokrótce przedstawić najważniejsze zagadnienia wchodzące w zakres tej dziedziny.

Na początku warto wspomnieć, że w ramach NLP zajmujemy się nie tylko przetwarzaniem wejścia tekstowego, ale również zagadnieniami takimi jak przetwarzanie mowy [16], pisma odręcznego [20] i optycznego rozpoznawania znaków [3]. W wielu rozwiązaniach działających na innym typie danych wejściowych niż tekst, pierwszym krokiem jest zamiana wejścia na ciąg znaków, jednak należy pamiętać, że może to prowadzić do utraty części informacji. Na przykład w przetwarzaniu mowy ważna jest nie tylko sama treść, ale także emocjonalne nacechowanie wypowiedzi. W dalszej części tego opracowania będziemy się zajmować wyłącznie przetwarzaniem języka naturalnego dla wejścia tekstowego, ale warto pamiętać, że dziedzina ta obejmuje także inne zagadnienia.

Ważną osią klasyfikacji zagadnień zagadnień związanych z przetwarzaniem tekstu jest podział na analizę oraz syntezę treści. W pierwszym przypadku celem jest zrozumienie wejściowej informacji, w drugim wygenerowanie treści, która będzie przekazywała określone znaczenie. Oba zagadnienia realizowane są przy wykorzystaniu modeli językowych [13]. Najprościej rzecz ujmując, model językowy stanowi zbiór zależności pomiędzy słowami w języku naturalnym. Czym lepiej te zależności przedstawimy, tym lepszej analizy i syntezy dokonamy. Modele językowe konstruuje się obecnie przy wykorzystaniu dużych zbiorów danych i różnych technik uczenia maszynowego, choć w przeszłości popularne było podejście ręcznej konstrukcji modeli przez ekspertów.

Do najważniejszych tematów związanych z przetwarzaniem tekstu możemy zaliczyć [6]:

1. Klasyfikacja tekstów – w zadaniach tego typu celem jest przypisanie odpowiedniej kategorii dla zadanego tekstu, np. głównego tematu dla wiadomości prasowych. Szczególny przypadek klasyfikacji tekstu stanowi analiza sentymentu, gdzie celem jest ustalenie wydźwięku analizowanej wypowiedzi, decydując, czy wypowiedź jest pozytywna, czy negatywna.
2. Wyszukiwanie informacji (ang. *information retrieval*) – szeroka dziedzina wchodząca w skład przetwarzania języka naturalnego. Podstawowym zadaniem stojącym przed systemami IR jest wyszukiwanie oraz rankingowanie dokumentów z określonego zbioru danych, na podstawie zapytań kierowanych przez użytkowników systemu.
3. Ekstrakcja informacji (ang. *information extraction*) – głównym celem w tej klasie zadań jest ekstrakcja interesujących informacji z zadanego tekstu. Dwoma popularnymi podproblemami zaliczanymi do tego zadania są wykrywanie nazw własnych (ang. *named entity recognition*) oraz linkowanie nazw (ang. *entity linking*), gdzie naszym celem jest połączenie fraz w tekście wejściowym z zewnętrzną bazą wiedzy definiującą ich znaczenie.
4. Odpowiadanie na pytania (ang. *question answering*) – w problemach tej kategorii, celem jest udzielanie odpowiedzi na pytania kierowane w języku naturalnym przez użytkowników systemu.
5. Tłumaczenie maszynowe (ang. *machine translation*) – tutaj celem jest przetłumaczenie tekstu z jednego języka naturalnego na drugi.
6. Systemy dialogowe (ang. *dialogue system*) – systemy tego typu służą prowadzeniu konwersacji z użytkownikiem, tak jakby prowadził on konwersację z innym człowiekiem.
7. Automatyczne streszczenia (ang. *automatic summarization*) – zadaniem stojącym przed algorytmami automatycznego streszczenia jest wygenerowanie skróconej wersji tekstu wejściowego, tak aby zachować najważniejsze zawarte w nim informacje.

1.3 Reprezentacja tekstu w sieciach neuronowych

W celu przetwarzania tekstu przez sieć neuronową, musimy w pierwszej kolejności dokonać jego zamiany na format zrozumiały dla modelu. Pierwszym etapem tej konwersji jest zamiana tekstu na sekwencję klas wejściowych.

Klasy wejściowe wykorzystywane przez sieć muszą zostać z góry zdefiniowane w słowniku wejściowym, tworzonym przed rozpoczęciem treningu. Istnieje kilka możliwości zamiany tekstu na sekwencję klas wejściowych, które pokrótce przedstawię.

Pierwszą możliwością jest zamiana tekstu wejściowego na sekwencję kolejnych wyrazów. W tym przypadku słownik wejściowy składa się ze słów wykorzystywanych w przetwarzanym języku. Wykorzystując ten typ reprezentacji, zanim dokonamy podziału na kolejne wyrazy, zazwyczaj dokonujemy wstępnego przetworzenia tekstu, ujednolicając wielkość liter, stosując stemming i podobne

techniki. Na przykład dla tekstu: „Ala ma żółwia" i słownika: Dict=[ala, asia, ma, psa, żółwia], sekwencja wejściowa mogłaby wyglądać następująco: [0,2,4].

Inną możliwością zamiany tekstu na sekwencję jest podział wejścia na kolejne znaki. W tej sytuacji słownik wejściowy zawiera wszystkie znaki które występują w przetwarzanym języku (czasami taki słownik nazywany jest alfabetem). Na przykład dla tekstu: „Ala ma żółwia" i słownika: Dict=[' ', a, ą, b, c, ć, d, e, ę, f, g, h, i, j, k, l, ł, m, n, ń, o, ó, p, r, s, ś, t, u, y, z, ź, ż], otrzymalibyśmy następującą sekwencję wejściową: [1, 15, 1, 0, 17, 1, 0, 32, 21, 16, 28, 12, 1].

Dwie powyższe metody stanowią najpopularniejszy sposób reprezentacji tekstu w sieciach neuronowych, jednak istnieje też kilka mniej popularnych, ale również ciekawych rozwiązań. Jednym z nich są bajtowe sieci neuronowe, w których reprezentujemy tekst wejściowy jako sekwencję kolejnych bajtów w wybranym kodowaniu (np. w UTF-8) [10]. Zaletą tego rozwiązania jest całkowita rezygnacja ze wstępnego przetwarzania tekstu i konstrukcji słownika wejściowego, co może stanowić dużą zaletę dla niektórych rozwiązań (np. w wielojęzycznych sieci neuronowych).

Różne reprezentacje tekstu sprawdzają się z różną skutecznością, w zależności od realizowanego zadania. Ogólnie rzecz biorąc, w przypadku zadań wysokopoziomowych, jak np. odpowiadanie na pytania, streszczanie testu, itp. lepiej sprawdzają się sieci wyrazowe, natomiast w problemach niskopoziomowych, jak np. NER, wykrywanie części mowy itp. sieci znakowe i bajtowe mogą sprawdzić się lepiej.

Po zdefiniowaniu słownika wejściowego i sposobu zamiany tekstu wejściowego na sekwencję klas wejściowych, kolejny etap stanowi zamiana sekwencji klas na sekwencję wektorów wejściowych. Najprostszą stosowaną tu techniką jest kodowanie z gorącą jedynką (ang. *one-hot encoding*), w którym każdy element sekwencji zamieniany jest na wektor o rozmiarze równym wielkości słownika, z jedynką w miejscu reprezentującą dany element i zerami w pozostałych miejscach.

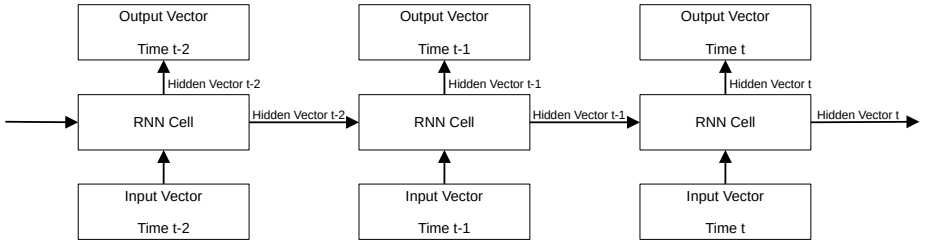
Inną często wykorzystywaną techniką zamiany sekwencji wejściowej na wektory wejściowe, jest osadzanie (ang. *embed*). Dokonywane jest ono w sieci neuronowej przez specjalną warstwę osadzającą (ang. *embedding layer*), która zamienia wejściową sekwencję klas na sekwencję gęstych wektorów (np. 100, albo 200 wymiarowych). Idea osadzeń polega na tworzeniu wektorów, które będą w podobny sposób reprezentować semantycznie podobne do siebie wyrazy. Warstwa osadzająca może być trenowana równolegle z pozostałymi elementami sieci, albo wykorzystać wcześniej przetrenowane osadzenia, takie jak np. word2vec [14], gloVe [18] czy FastText [15].

1.4 Sieci rekurencyjne w przetwarzaniu tekstu

Zanim przejdziemy do omówienia mechanizmu atencji w sieciach neuronowych, chciałbym pokrótce przedstawić zastosowanie sieci rekurencyjnych w przetwarzaniu tekstu. Jest to dla nas ważne, ponieważ mechanizm atencji zaproponowano właśnie w celu przezwyciężenia występujących w tych sieciach ograniczeń.

Podstawową cechą rekurencyjnych sieci neuronowych jest możliwość przetwarzania sekwencji wejściowych o zmiennej długości, co nie jest możliwe w sieciach jednokierunkowych (ang. *feed-forward*). Główny element każdej sieci rekurencyjnej stanowi komórka rekurencyjna (ang. *RNN cell*). Podczas przetwarzania sekwencji wektorów wejściowych przez sieć, każdy kolejny wektor jest przekazywany na wejście komórki rekurencyjnej, która następnie łączy zawarte w nim informacje z wektorem ukrytym (ang. *hidden vector*), wyliczonym w poprzednim kroku sieci (Rysunek 1.1). Wektor ukryty dla pierwszego kroku sieci stanowi jeden z jej hiperparametrów (zwykle jest to po prostu wektor zerowy).

Wektor ukryty w sieciach rekurencyjnych przechowuje informacje o aktualnym stanie sieci. Sposób w jaki sieć łączy informacje z wektora wejściowego i ukrytego zależy od konkretnej architektury komórki (najpopularniejszymi architekturami są: ElmanRNN, LSTM [11] i GRU [5]). Warto zaznaczyć, że komórka rekurencyjna posiada te same parametry dla każdego kroku czasowego. Parametry te są aktualizowane przez algorytm spadku gradientowego dopiero po przetworzeniu całej sekwencji (lub wsadu sekwencji). Proces ten nazywany jest propagacją wsteczną w czasie (ang. *back propagation through time*).

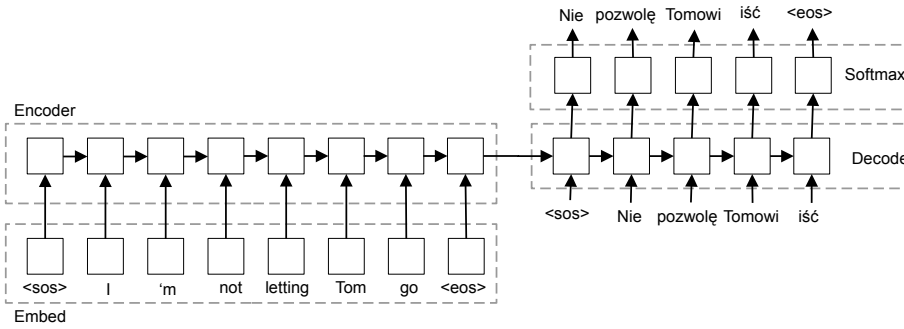


Rysunek 1.1. Architektura rekurencyjnej sieci neuronowej

1.5 Sieci typu ciąg-ciąg

Sieci rekurencyjne dobrze nadają się do zadań związanych z klasyfikacją tekstu, jednak wykorzystanie ich do zadań polegających na przetworzeniu sekwencji wejściowej na inną sekwencję wyjściową, takich jak np. tłumaczenia maszynowe, czy automatyczne streszczanie jest dość problematyczne. Problem polega na tym, że długość wyjścia sieci rekurencyjnej musi być identyczna z długością jej wejścia – każdy kolejny krok sieci generuje dokładnie jeden wektor wyjściowy. Rozwiązaniem tego problemu stanowią sieci ciąg-ciąg (ang. *sequence-to-sequence*) [17].

Sieci ciąg-ciąg składają się zasadniczo z dwóch osobnych sieci neuronowych, zwanych: enkoderem (ang. *encoder*) i dekoderem (ang. *decoder*) (Rysunek 1.2). Zadaniem enkodera jest przetworzenie wejściowej sekwencji i „skompresowaniem" jej znaczenia do wektora stanu (ang. *state vector*), przekazywanego następnie jako pierwszy wektor ukryty dla dekodera. Za wektor stanu przyjmuje się zazwyczaj ostatni stan ukryty sieci enkodera.



Rysunek 1.2. Przykładowa architektura sieci ciąg-ciąg dla problemu tłumaczenia maszynowego.

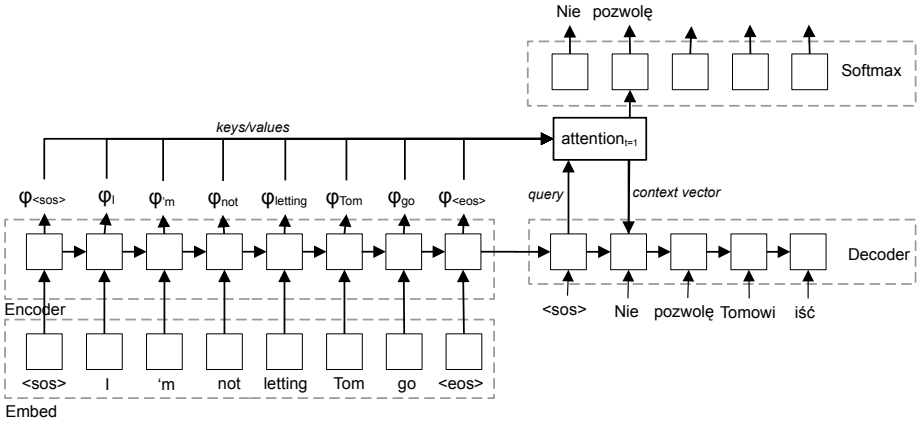
Na architekturę sieci ciąg-ciąg składają się również dwa słowniki: wejściowy oraz wyjściowy, które odwzorowują odpowiednio tekst wejściowy na klasy wejściowe i klasy wyjściowe na właściwe wyjście sieci. Słowniki te, poza standardowymi elementami, posiadają także dwa specjalne elementy oznaczające odpowiednio początek sekwencji (ang. *start of sequence*): <sos> i koniec sekwencji (ang. *end of sequence*): <eos>. Token informujący nas o końcu sekwencji wyjściowej jest dla nas bardzo istotny, ponieważ nie jesteśmy w stanie z góry przewidzieć jej długości.

Istotną kwestią dla sieci ciąg-ciąg jest też sposób dostarczania kolejnych elementów sekwencji wejściowej dla dekodera (Rysunek 1.2). Na etapie działania wytrenowanej sieci jako kolejne wejścia sieci dekodera są przekazywane jego poprzednie wyjścia (jako pierwsze wejście jest przekazywany specjalny element <sos>). Podczas treningu natomiast mamy dwie możliwości. Pierwszą jest wykorzystanie wyjść dekodera (tak jak podczas działania sieci), drugą natomiast użycie właściwej sekwencji wejściowej ze zbioru treningowego. Drugie podejście jest nazywane wymuszaniem przez nauczyciela (ang. *teacher forcing*) i umożliwia przyspieszyć trening sieci, choć może również prowadzić do pewnych problemów podczas właściwego działania modelu. W praktyce wymuszanie przez nauczyciela jest często wykorzystywane na początku treningu sieci, która później jest dotrenowywana przy wykorzystywaniu własnych wyjść.

Modele ciąg-ciąg całkiem nieźle sprawdzają się w praktyce, jednak wątpliwości może budzić tutaj pomysł na „kompresję" całej informacji o wejściowej sekwencji do postaci pojedynczego wektora. Być może udałoby się osiągnąć lepsze rezultaty, gdyby dekoder sieci mógł czerpać więcej informacji z sekwencji wejściowej? Właśnie ta intuicja doprowadziła do zaproponowania mechanizmu atencji w sieciach neuronowych, który teraz chciałbym przedstawić.

1.6 Mechanizm atencji

Mechanizm atencji w modelach ciąg-ciąg umożliwia sieci dekodera na czerpanie informacji nie tylko z wektora stanu, ale także dodatkowo z wyjść enkodera (rysunek 1.3). Mechanizm ten dla problemu tłumaczenia maszynowego został po raz pierwszy zaproponowany w 2015 roku i umożliwił na znaczną poprawę wydajności dotychczasowych modeli, szczególnie przy tłumaczeniu długich zdań [2].



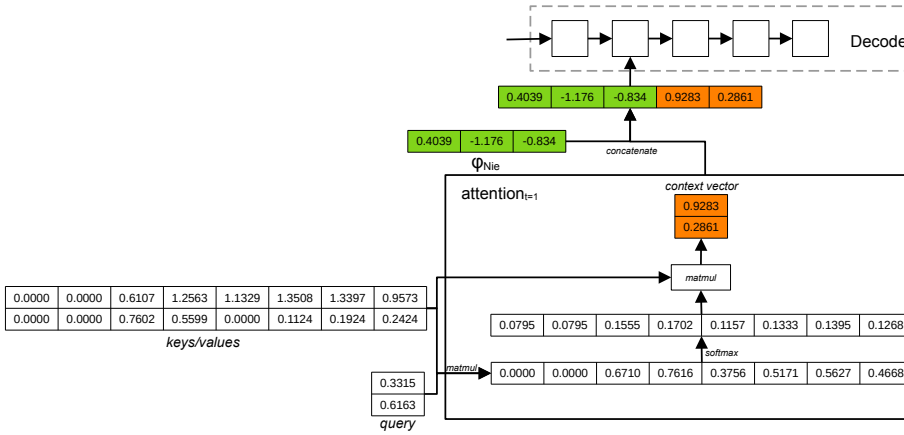
Rysunek 1.3. Mechanizm atencji w sieci ciąg-ciąg dla problemu tłumaczenia maszynowego

Istnieje klika sposobów implementacji atencji. Na potrzeby niniejszego opracowania, chciałbym się skupić na najpopularniejszym z nich, zwanym atencją z iloczynem skalarnym (ang. *dot-product attention*). Atencja z iloczynem skalarnym formalnie jest zdefiniowana w następujący sposób:

$$Attention(Q, K, V) = softmax(QK^T)V \tag{1.1}$$

W modelach ciąg-ciąg atencja jest obliczana dla każdego kroku sieci dekodera (Rysunek 1.4). Pierwszym etapem wyliczania atencji jest przemnożenie macierzy kolejnych wyjść enkodera (nazywanej macierzą kluczy (ang. *keys*) przez wektor reprezentujący poprzedni stan ukryty dekodera (nazywany wektorem zapytania (ang. *query*), tworząc wagi atencji (ang. *attention weights*). Długość wektora wag atencji jest identyczna z długością sekwencji wyjściowej dekodera. W kolejnym kroku wagi atencji są przekazywane na wejście funkcji *softmax*, co pozwala interpretować je jako stopień wpływu poszczególnych elementów sekwencji wejściowej na aktualny krok czasowy dekodera. Następnie wagi atencji są ponownie przemnażane przez macierz kolejnych wyjść enkodera (nazywanej w tym kontekście wartościami (ang. *values*) w celu wyliczenia wektora kontekstu (ang. *context vector*). Ostatecznie wektor kontekstu jest dołączany do wektora wejściowego sieci

dekodera. Zauważmy, że dla każdego kroku atencji wektor kluczy i wartości jest zawsze ten sam, a zmianie ulega jedynie wektor zapytania.



Rysunek 1.4. Pojedynczy krok atencji w sieci ciąg-ciąg. Zakładamy tutaj, że sieć enkodera zwraca na wyjściu sekwencję wektorów dwuelementowy, a sieć dekodera pobiera na wejściu wektory pięcioelementowe

Podczas treningu sieci dekodera, komórka rekurencyjna uczy się nie tylko wykorzystywać poprzedni stan ukryty, ale również czerpać informację z wektora wyjściowego sieci enkodera. W przypadku problemu tłumaczenia maszynowego, pozwala jej to kojarzyć poszczególne wyrazy z języka wejściowego z odpowiadającymi im wyrazami z języka wyjściowego, nawet jak znajdują się one na innej pozycji w zdaniu.

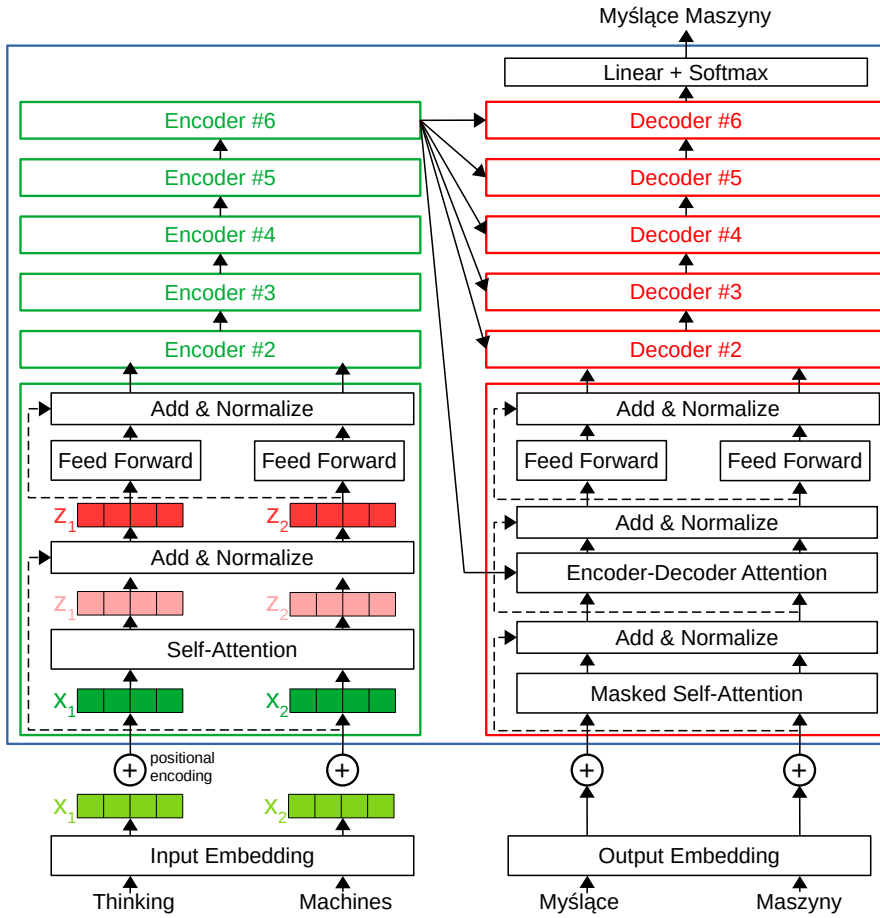
Mechanizm atencji w sieciach neuronowych okazał się bardzo trafionym pomysłem, a od momentu jego zaproponowania, pojawiło się bardzo wiele jego wariantów i modyfikacji. Jednak niewątpliwie najbardziej radykalny pomysł zastosowania tego mechanizmu stanowi sieć Transformer, zaproponowana przez zespół badawczy z Google w 2017 roku [21]. Zaproponowany model wychodzi z założenia, że atencja jest na tyle ekspresywnym mechanizmem, że na jej rzecz możemy całkowicie zrezygnować z rekurencyjnych sieci neuronowych. Pomysł ten okazał się niezwykle trafiony, a zaproponowany model otworzył zupełnie nowy rozdział w dziedzinie przetwarzania języka naturalnego.

1.7 Transformer

Jedną z podstawowych wad rekurencyjnych sieci ciąg-ciąg jest ich sekwencyjna natura. Ponieważ każdy kolejny krok czasowy wymaga informacji o kroku poprzednim, niemożliwe jest zrównoleglenie treningu sieci, co wymusza ograniczenie wolumenu danych treningowych oraz utrudnia testowanie nowych architektur. W obliczu tego ograniczenia podejmowano różne próby jego przezwyciężenia,

na przykład testowano zastosowanie sieci konwolucyjnych zamiast rekurencyjnych [9], jednak dopiero sieć Transformer okazała się prawdziwym przełomem w tej kwestii.

Sieć Transformer stanowi przykład sieci ciąg-ciąg z wyróżnionymi modułami enkodera i dekodera (Rysunek 1.5). Cechą charakterystyczną sieci jest możliwość trenowania jej w sposób równoległy – całą sekwencję wejściową oraz wyjściową można przetworzyć równocześnie, co znacznie przyspiesza trening.



Rysunek 1.5. Ogólna architektura sieci Transformer z rozpisanymi blokami enkodera i dekodera. Każdy blok enkodera i dekodera posiada identyczną architekturę, ale różne wagi

Pierwszym etapem działania sieci Transformer, jest zamiana tokenów wejściowych na wektory osadzeń. Dodatkowo każdy wektor wejściowy jest sumowany

z wektorem kodowania pozycyjnego, informującym o pozycji danego elementu w sekwencji.

Na moduł enkodera i dekodera sieci Transformer składa się po sześć następujących po sobie bloków. Każdy blok enkodera składa się z warstwy samoatencji (ang. *self-attention*) oraz sieci jednokierunkowej (ang. *feed forward neural network*). Pomiędzy tymi warstwami występują dodatkowe warstwy normalizacyjne oraz połączenia rezydualne.

Blok dekodera składa się natomiast z trzech zasadniczych warstw. Pierwszą stanowi *masked self-attention*, następną atencja pomiędzy enkoderem a dekoderem (ang. *encoder-decoder attention*), a ostatnią sieć jednokierunkowa. Tak jak w przypadku enkodera pomiędzy tymi warstwami występują dodatkowe warstwy normalizacyjne oraz połączenia rezydualne. Po ostatnim bloku dekodera znajduje się dodatkowa warstwa liniowa oraz *softmax*, służąca przeprowadzeniu ostatecznej predykcji tokenu wyjściowego.

W następnych podrozdziałach postaram się pokrótce opisać poszczególne moduły składające się na sieć oraz wytłumaczyć zasadę ich działania.

1.7.1 Kodowanie pozycyjne

W sieciach rekurencyjnych informacja o pozycji danego wektora w sekwencji wejściowej jest przekazywana poprzez sekwencyjne wywoływanie komórki rekurencyjnej. W modelu Transformer jednak, gdzie cała sekwencja jest przetwarzana w sposób równoległy, nie mamy bezpośredniego dostępu do tej informacji.

W celu rozwiązania tego problemu, twórcy Transformera zaproponowali specjalną technikę, zwaną kodowaniem pozycyjnym (ang. *positional encoding*). Pomysł ten polega na dodaniu dodatkowego wektora do osadzeń poszczególnych słów sekwencji wejściowych, który informowałby model o pozycji danego wektora w sekwencji:

$$\psi'(w_t) = \psi(w_t) + \vec{p_t}, \tag{1.2}$$

gdzie $w_{1\ldots n}$ to kolejne elementy sekwencji wejściowej, $\psi(\cdot)$ to funkcja osadzająca, $\vec{p_t}$ to wektor kodowania pozycyjnego dla t-tego elementu sekwencji, a $\psi'(\cdot)$ to docelowy wektor przekazywany na wejście modelu Transformer. Jak widzimy zarówno wektor $\psi(\cdot)$ jak i $\vec{p_t}$ muszą posiadać jednakową długość. W modelu Transformer została ona ustalona na $d = 512$.

Twórcy modelu uznali, że aby wektor kodowania pozycyjnego dobrze spełniał swoje zadanie, musi posiadać następujące cechy:

- unikatowość – każdy krok czasowy musi być zakodowany w sposób unikatowy;
- zachowanie dystansu – względny dystans pomiędzy dwoma krokami czasowymi musi być zawsze taki sam;
- skalowalność – kodowanie pozycyjne powinno dać się zastosować dla sekwencji o dowolnej długości;
- determinizm – wektor kodowania pozycyjnego powinien być deterministyczny.

Biorąc pod uwagę te wymagania, twórcy Transformera zdecydowali się na zastosowanie wektora kodowania pozycyjnego, składającego się z par funkcji sinus i cosinus:

$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1.t) \\ \cos(\omega_1.t) \\ \\ \sin(\omega_2.t) \\ \cos(\omega_2.t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2}.t) \\ \cos(\omega_{d/2}.t) \end{bmatrix}_{d\times 1} \quad \text{gdzie } \omega_k = \frac{1}{10000^{2k/d}} \tag{1.3}$$

Jak widzimy, wektor $\vec{p_t}$ składa się z kolejnych funkcji sinusoidalnych o okresie rosnącym od 2π dla ω_1 do $10000 \cdot 2\pi$ dla $\omega_{d/2}$. Wektor ten na pewno jest deterministyczny, co bezpośrednio wynika z definicji. Musimy się natomiast trochę głębiej zastanowić nad tym, w jaki sposób zapewnia on unikatowość, skalowalność oraz zachowanie dystansu.

Unikatowość oraz skalowalność Zacznijmy od pewnej intuicji. Przypomnijmy sobie w jaki sposób zmieniają się kolejne bity w kolejnych liczbach zakodowanych w systemie dwójkowym (Rysunek 1.6).

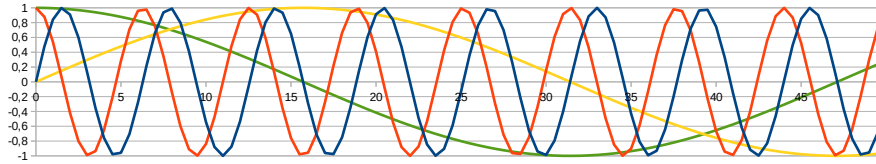
0:	0	0	0	0	8:	1	0	0	0
1:	0	0	0	1	9:	1	0	0	1
2:	0	0	1	0	10:	1	0	1	0
3:	0	0	1	1	11:	1	0	1	1
4:	0	1	0	0	12:	1	1	0	0
5:	0	1	0	1	13:	1	1	0	1
6:	0	1	1	0	14:	1	1	1	0
7:	0	1	1	1	15:	1	1	1	1

Rysunek 1.6. Liczby od 0 do 15 zakodowane w systemie dwójkowym

Jak widzimy wszystkie zmiany następują tutaj w sposób okresowy, jednak wraz ze wzrostem starszeństwa bitów okres ten jest coraz dłuższy – dla najmłodszego bitu zmiana następuje co jedną liczbę, dla starszego co dwie, itd. Podobna sytuacja ma miejsce w kodowaniu pozycyjnym, tylko tutaj zamiast zmieniających się bitów, występują okresowe funkcje sinus oraz cosinus (Rysunek 1.7).

Jak widzimy pierwszy i drugi element wektora $\vec{p_t}$ zmienia się co ok. 5 kroków czasowych, element trzeci i czwarty co ok. 60 itd. Wektor kodowania pozycyjnego, dla każdego kolejnego kroku czasowego, składa się więc z kombinacji ko-

lejnych funkcji sinus i cosinus o coraz dłuższych okresach, podobnie do tego jak kodowanie liczb w systemie dwójkowym, składa się z okresowo zmieniających się bitów na kolejnych pozycjach.



t \ $\vec{p}_t$	$\sin(\omega_1 \cdot t)$	$\cos(\omega_1 \cdot t)$	$\sin(\omega_2 \cdot t)$	$\cos(\omega_2 \cdot t)$
0	sin(0/1) = 0,00	cos(0/1) = 1,00	sin(0/10) = 0,00	cos(0/10) = 1,00
1	sin(1/1) = 0,84	cos(1/1) = 0,54	sin(1/10) = 0,10	cos(1/10) = 0,99
2	sin(2/1) = 0,91	cos(2/1) = -0,42	sin(2/10) = 0,20	cos(2/10) = 0,98
3	sin(3/1) = 0,14	cos(3/1) = -1,00	sin(3/10) = 0,30	cos(3/10) = 0,96
4	sin(4/1) = -0,75	cos(4/1) = -0,65	sin(4/10) = 0,39	cos(4/10) = 0,92
5	sin(5/1) = -0,96	cos(5/1) = 0,28	sin(5/10) = 0,48	cos(5/10) = 0,88
6	sin(6/1) = -0,28	cos(6/1) = 0,96	sin(6/10) = 0,56	cos(6/10) = 0,83
7	sin(7/1) = 0,65	cos(7/1) = 0,75	sin(7/10) = 0,64	cos(7/10) = 0,76

Rysunek 1.7. Pierwsze cztery elementy wektora kodowania pozycyjnego, przy założeniu $d = 8$

Mam nadzieję, że zapewnienie unikatowości przez wektor $\vec{p_t}$ jest teraz widoczne – dla każdej kolejnej wartości t dostajemy unikatowe kombinacje funkcji trygonometrycznych. Co dodatkowo istotne, wektor ten umożliwia nam na zakodowanie sekwencji o dowolnej długości, w szczególności sekwencji dłuższych od tych uwzględnianych w treningu. Teoretycznie funkcja o największym okresie „zapętli" się jeżeli długość sekwencji wejściowej przekroczy $t = 10,000$, co jednak nie stanowi problemu w praktyce (przynajmniej w przypadku tłumaczenia maszynowego, dla którego model Transformer był projektowany).

Dodatkową bardzo istotną cechą kodowania pozycyjnego jest zapisywanie informacji o pozycji tokenu w sekwencji na stosunkowo niewielkiej liczbie pierwszych elementów wektora wejściowego $\psi'(w_t)$. Wynika to tego, że kolejne funkcje trygonometryczne w wektorze mają coraz większe okresy i dla małych wartości t będą one pozostawać przy swoich wartościach początkowych (0 dla sinusa i 1 dla cosinusa). Ponieważ osadzenia w modelu Transformer są trenowane równolegle z całym modelem, umożliwia to modelowi na zapisywanie semantyki poszczególnych wyrazów na dalszych elementach wektora wejściowego i pozostawienie jego pierwszych elementów na informację o pozycji danego elementu w sekwencji.

Zachowanie dystansu Zastanówmy się teraz nad następującą kwestią: dlaczego w wektorze kodowania pozycyjnego występują naprzemiennie funkcje sinus oraz cosinus, skoro w celu zapewnienia unikatowości oraz skalowalności, wystarczyłaby tylko jedna z nich? Odpowiedź stanowi tutaj potrzeba zachowania dystansu pomiędzy poszczególnymi elementami sekwencji.

Problem ten możemy sformułować również w następujący sposób: dla dowolnej ustalonej wartości i, musi istnieć liniowe przekształcenie wektora: $\vec{p_t}$ w wektor: $\vec{p_{t+i}}$, niezależne od wartości t. W celu udowodnienia hipotezy, musimy wykazać, że dla każdej pary: $\sin(w_k{+}t)$, $\cos(w_k{+}t)$ w wektorze, istnieje przekształcenie liniowe $M \in \mathbb{R}^{2\times 2}$, takie że:

$$M \cdot \begin{bmatrix} \sin(\omega_k.t) \\ \cos(\omega_k.t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k.(t+i)) \\ \cos(\omega_k.(t+i)) \end{bmatrix} \tag{1.4}$$

W celu prześledzenia dowodu krok po kroku odsyłam do odpowiedniego źródła: [12]. Dla nas w tym momencie istotny jest fakt, że takie przekształcenie faktycznie istnieje i dla dowolnych wartości i i k ma ono następującą postać:

$$M_{i,k} = \begin{bmatrix} \cos(\omega_k \cdot i) & \sin(\omega_k \cdot i) \\ -\sin(\omega_k \cdot i) & \cos(\omega_k \cdot i) \end{bmatrix} \tag{1.5}$$

Przedstawiona własność wektora kodowania pozycyjnego sprawia, że względne odległości między poszczególnymi elementami w sekwencji wejściowej są sobie równe, niezależnie od konkretnych pozycji w sekwencji. W konsekwencji umożliwia to modelowi na wychwycenie zależności między poszczególnymi tokenami, niezależnie od ich bezwzględnej pozycji.

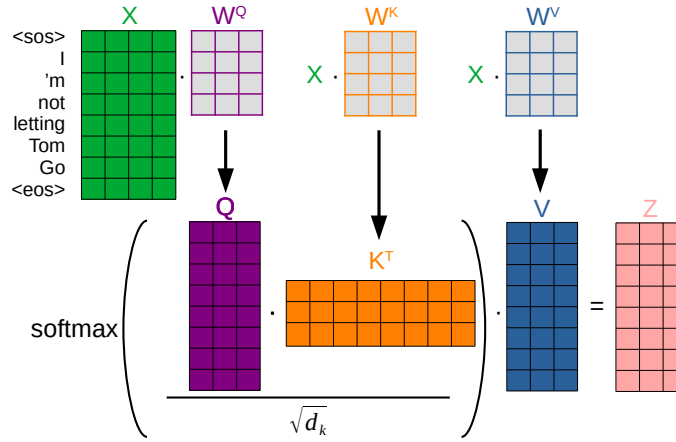
1.7.2 Self-attention

Ważnym blokiem występującym zarówno po stronie bloków enkodera jak i dekodera jest warstwa *self-attention*. Warstwa ta wykorzystuje znany nam mechanizm atencji, w celu wyliczenia jak poszczególne wyrazy w sekwencji wejściowej odnoszą się do siebie nawzajem. W celu zilustrowania idei *self-attention* rozważmy następujące zdania: „Daliśmy małpom banany, bo były głodne." oraz „Daliśmy małpom banany, bo były przejrzałe.". Oba zdania posiadają identyczną strukturę gramatyczną, jednak czasownik „były" w pierwszym odnosi się do małp, a w drugim do bananów. Celem mechanizmu *self-attention* jest wychwycenie tego typu zależności, umożliwiając tym samym na lepsze zrozumienie języka naturalnego przez model.

Mechanizm *self-attention* działa podobnie do mechanizmu atencji z iloczynem skalarnym w sieciach rekurencyjnych. Najważniejszą różnicą jest wykorzystanie tutaj podlegających treningowi macierzy: W^Q, W^K oraz W^V, w celu wyliczenia odpowiednio wektorów zapytań, kluczy i wartości (Rysunek 1.8). Dodatkowo każdy element wyniku iloczynu między macierzą zapytań i kluczy jest dzielony przez pierwiastek z d_k w celu regularyzacji. d_k stanowi hiperparametr modelu definiujący wielkości wektora zapytań, kluczy oraz wartości (w sieci Transformer $d_k = 64$).

W pierwszym kroku *self-attention*, dokonujemy wyliczenia macierzy: Q, K, V, poprzez wymnożenie macierzy wejściowej poprzez podlegające treningowi macierze: W^Q, W^K oraz W^V. Następnie wykorzystujemy wyliczone wektory w celu wyliczenia wektora Z, stanowiącego wyjście modułu *self-attention*. Na

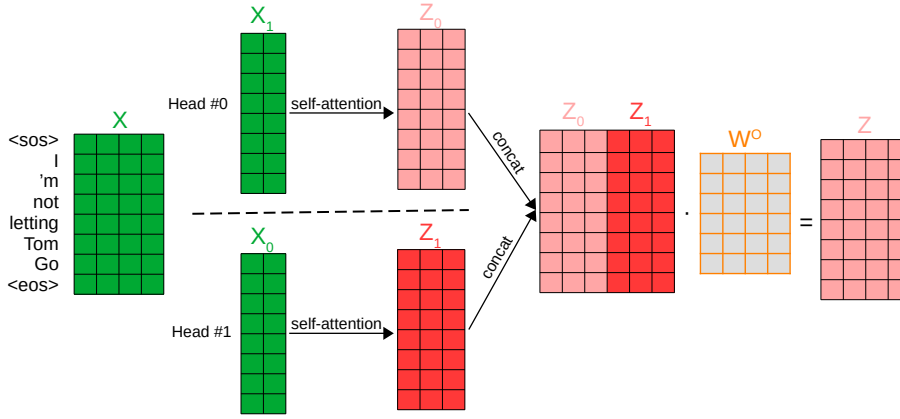
wejście pierwszego bloku enkodera przekazywane są osadzenia kolejnych elementów sekwencji wejściowej, natomiast na wejścia następnych bloków enkodera oraz bloków dekodera, wyjścia poprzednich bloków (Rysunek 1.5).



Rysunek 1.8. Działanie modułu *self-attention*. Na rysunku zakładamy, że na wejściu otrzymujemy czterowymiarowe wektory, a na wyjściu wektory trójwymiarowe

Mechanizm samo-atencji w sieci Transformer został dodatkowo rozszerzony do postaci tzw. *multi-head self-attention* (Rysunek 1.9). Jedyna różnica polega tu na tym, że przed wyliczeniem *self-attention* dla wejściowej macierzy, dzielimy ją na h równych części (wzdłuż osi osadzeń) i dla każdej z nich obliczamy osobną macierz Z_i (w modelu Transformer $h = 8$). Ostatecznie sklejamy wszystkie macierze $Z_{1...8}$ i przemnażamy je przez dodatkową macierz W^O w celu otrzymania ostatecznego wyjścia warstwy Z.

Na koniec warto jeszcze zaznaczyć, że moduł *self-attention* różni się nieco między blokiem enkodera, a blokiem dekodera. W przypadku enkodera każdy element sekwencji wejściowej może czerpać informację zarówno z elementów stojących przed nim jak i za nim. W przypadku dekodera każdy token może jedynie czerpać informację o elementach stojących za nim w sekwencji. Mechanizm ten został przez twórców modelu nazwany *masked self-attention*. To dodatkowe ograniczenie wynika z faktu, że gdybyśmy podczas treningu sieci „pokazywali" dekoderowi całą sekwencję docelową na raz, to zamiast tworzyć własne predykcje, wykorzystywałby po prostu informację z wejściowej sekwencji treningowej. *Masked self-attention* jest realizowana poprzez zastąpienie wszystkich maskowanych wejść deokdera wartością $-inf$, tuż przed wyliczeniem wartości funkcji *softmax*.



Rysunek 1.9. Działanie modułu *multi-head self-attention*. W modelu Transformer wejściowy wektor o szerokość 512, jest dzielony na 8 wektorów o szerokości 64 każdy. Pośrednie macierze Z_i dla każdej głowy również mają szerokość 64. Szerokość wyjściowej macierzy Z jest równa na powrót 512

1.7.3 Stos enkodera

W każdym bloku enkodera po warstwie *self-attention* znajduje się pierwsza warstwa normalizacyjna (ang. *normalize layer*) [1], której celem jest skrócenie czasu treningu modelu (Rysunek 1.5). Przed zastosowaniem normalizacji, wyjście modułu *self-attention* jest dodatkowo sumowane z macierzą wejściową bloku. Operacja sumowania tworzy połączenie rezydualne, które pozwala modelowi na lepsze radzenie sobie ze zjawiskiem zanikającego gradientu (ang. *vanishing gradient*).

Po pierwszej warstwie normalizacyjnej, każde z wyjść jest przepuszczane przez sieć jednokierunkową (wagi warstwy jednokierunkowej są identyczne dla każdego wektora w sekwencji). W modelu Transformer sieć ta składa się z dwóch warstw. Pierwsza mapuje 512-wymiarowy wektor Z na wektor o wymiarze 2048, a następnie przepuszcza go przez funkcję $ReLU$. Druga natomiast przekształca wektor z powrotem na 512 wymiarowy i nie posiada już dodatkowej funkcji aktywacji.

Na wejście pierwszego w stosie bloku enkodera trafia macierz osadzeń wyrazów, a na wejście kolejnych wyjścia warstw niższych. Wejścia i wyjścia bloków enkodera mają ten sam rozmiar $d = 512$.

1.7.4 Stos dekodera

Tak samo jak w przypadku rekurencyjnych sieci ciąg-ciąg, na wejście pierwszego bloku stosu dekodera trafia fraza docelowa rozpoczynająca się od specjalnego tokenu <sos>. Jeżeli trenujemy model w trybie wymuszania przez nauczyciela, możemy przetworzyć całą sekwencję docelową w sposób równoległy. Jeżeli natomiast trenujemy model w sposób standardowy lub uruchamiamy go na danych

docelowych, musimy uruchamiać moduł dekodera element po elemencie, aż do napotkania na wyjściu specjalnego tokenu <eos>. W tym drugim przypadku po każdym uruchomieniu uzyskujemy na wyjściu kolejny element sekwencji docelowej, który następnie dołączamy na koniec sekwencji wejściowej przed kolejnym uruchomieniem.

Tak samo jak w stosie enkodera, po module *masked self-attention* w dekoderze znajduje się warstwa normalizacyjna oraz dodatkowe połączenie rezydualne.

Następny element bloku dekodera stanowi warstwa atencji enkoder-dekoder (ang. *encoder-decoder attention*). Warstwa ta działa identycznie jak warstwa *self-attention*, z tą różnicą, że macierze K i V są wyliczane na podstawie macierzy wyjściowej pobranej z ostatniej warstwy enkodera. Tak samo jak po warstwie *self-attention*, po tej warstwie również znajduje się warstwa normalizacyjna oraz dodatkowe połączenie rezydualne.

Ostatnią warstwę bloku dekodera stanowi sieć jednokierunkowa o architekturze identycznej jak w przypadku bloku enkodera.

Bloki dekodera połączone są ze sobą analogicznie jak to ma miejsce w enkoderze. Po ostatnim bloku dekodera znajduje się pojedyncza warstwa liniowa, która mapuje wyjściowy wektor o rozmiarze $d = 512$ na wektor o rozmiarze równym wielkości słownika wyjściowego (kodowanie z gorącą jedynką). Po zastosowaniu funkcji *softmax*, token o największym prawdopodobieństwie stanowi wyjście kroku czasowego.

1.8 Podsumowanie

W niniejszym opracowaniu skupiliśmy się na przybliżeniu tematyki atencji w sieciach neuronowych oraz szczegółowym przeanalizowaniu architektury sieci Transformer. Pomysły zaczerpnięte z Transformera zostały w niedługim czasie wykorzystane przy projektowaniu modeli językowych nowej generacji, takich jak BERT [7], czy GPT-2 [19] i GPT-3 [4], wnoszących zupełnie nową jakość w wielu zagadnieniach związanych z przetwarzaniem języka naturalnego.

Na koniec warto wspomnieć, że zastosowania sieci opartych o architekturę Transformer wykraczają poza zagadnienia związane z przetwarzaniem języka naturalnego i zdobywają coraz większe uznanie w innych dziedzinach uczenia maszynowego. Szczególnie warto tutaj wspomnieć zastosowanie sieci Transformer jako alternatywy dla sieci konwolucyjnych w widzeniu komputerowym [8]. Stanowi to jeszcze jedne powód dla którego warto się dobrze z tą architekturą zapoznać.

Bibliografia

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization (2016).
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015).

3. Borovikov, E.: A survey of modern optical character recognition techniques (2014).
4. Brown, T.B., et al.: Language models are few-shot learners (2020).
5. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. pp. 1724–1734. ACL (2014). https://doi.org/10.3115/v1/d14-1179.
6. Deng, L., Liu, Y. (eds.): Deep Learning in Natural Language Processing. Springer, Berlin/Heidelberg (2018).
7. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). pp. 4171–4186. Association for Computational Linguistics (2019). https://doi.org/10.18653/v1/n19-1423.
8. Dosovitskiy, A., et al.: An image is worth 16x16 words: Transformers for image recognition at scale (2021).
9. Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N.: Convolutional sequence to sequence learning. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34$^{\text{th}}$ International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1243–1252. PMLR (06–11 Aug 2017).
10. Gillick, D., Brunk, C., Vinyals, O., Subramanya, A.: Multilingual language processing from bytes (2016).
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735.
12. Kazemnejad, A.: Transformer architecture: The positional encoding (Sep 2019), `https://kazemnejad.com/blog/transformer_architecture_positional_encoding/`, Accessed 21 May 2021.
13. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval, chap. 12. Cambridge University Press (2008).
14. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: Bengio, Y., LeCun, Y. (eds.) 1$^{\text{st}}$ International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings (2013).
15. Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A.: Advances in pre-training distributed word representations. In: Calzolari, N., et al. (eds.) Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018. European Language Resources Association (ELRA) (2018).
16. Nassif, A.B., Shahin, I., Attili, I., Azzeh, M., Shaalan, K.: Speech recognition using deep neural networks: A systematic review. IEEE Access **7**, 19143–19165 (2019). https://doi.org/10.1109/ACCESS.2019.2896880.

17. Neubig, G.: Neural machine translation and sequence-to-sequence models: A tutorial (2017).
18. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. pp. 1532–1543. ACL (2014). https://doi.org/10.3115/v1/d14-1162.
19. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019).
20. Remaida, A., Moumen, A., El Idrissi, Y.E.B., Sabri, Z.: Handwriting recognition with artificial neural networks a decade literature review. In: Proceedings of the 3rd International Conference on Networking, Information Systems & Security. NISS2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3386723.3387884.
21. Vaswani, A., et al.: Attention is all you need. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 5998–6008 (2017).

2. Generowanie tekstu z użyciem sieci typu Transformer

Michał Wilk,
Radosław Baziak,
Julian Szymański

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
michalwilk139@gmail.com
radek-baziak@wp.pl
julian.szymanski@eti.pg.edu.pl

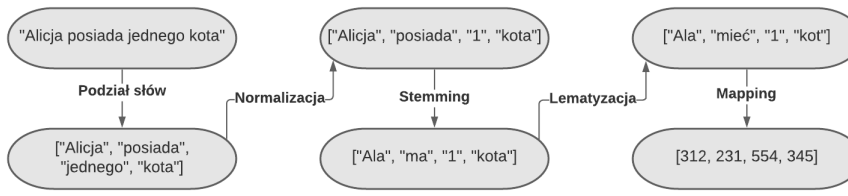
Streszczenie
Opisano działanie wybranych modeli uczenia maszynowego znajdujących zastosowanie w przetwarzaniu języka naturalnego w szczególności wykorzystywanych do generowania tekstu. Przedstawiono również model BERT i jego różne wersje, a także praktyczne wykorzystanie modeli typu Transformer. Przedstawiono ich działanie w aplikacji zmieniającej nastrój tekstu w sposób sekwencyjny.

Słowa kluczowe: Transformer, *Sentiment Analysis*, *Text Generation*, *Natural Language Processing*, *Deep Learning*

2.1 Wprowadzenie

Przetwarzanie języka naturalnego jest obecnie bardzo dynamicznie rozwijającą się dziedziną. W ciągu ostatnich lat metody uczenia maszynowego znalazły zastosowanie m.in. w: tłumaczeniu tekstu, *chatbotach*, oznaczaniu części mowy, wykrywaniu plagiatów, określaniu nastroju wypowiedzi, podpowiedziach wyrazów (*IntelliSense*), czy tworzeniu streszczeń [21].

Typowe etapy przetwarzania języka można przedstawić jako sekwencję działań pokazaną na Rysunku 2.1.



Rysunek 2.1. Etapy kodowania tekstu na strukturę danych

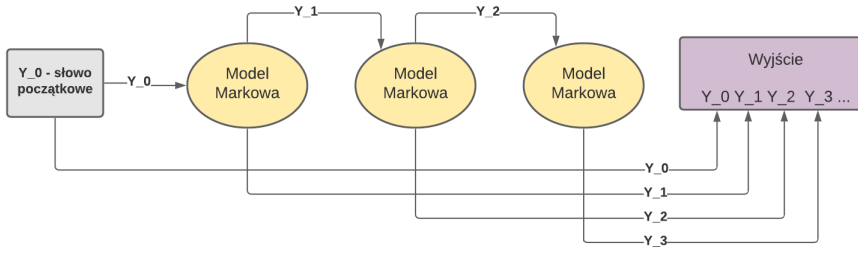
1. **Tokenizacja** - zamiana tekstu na listę atomicznych słów, inaczej też zwanych tokenami. Operacja ta przeprowadzana jest w kilku etapach. Pierwszym z nich jest rozdzielenie tekstu według wystąpień spacji czy znaków interpunkcyjnych. Następnie przeprowadza się normalizację, czyli zamianę słów o tym samym znaczeniu na jednolitą reprezentację. Dobrym przykładem są tutaj liczebniki. Na przykład wyrazy „dziesięć" oraz „10" są znaczeniowo identyczne, więc oba normalizujemy do tej samej postaci: „10". w kolejnych krokach przeprowadza się *stemming* lub lematyzacje. *Stemming* odpowiada za redukcję słowa do jego rdzenia. Aby to zrobić, usuwane są z niego wszelkie afiksy (prefiksy i sufiksy). Popularnymi algorytmami *stemmingowymi* są algorytm Portera (1979 r.) oraz algorytm Lancaster (1990 r.). Lematyzacja sprowadza słowo do jego formy podstawowej, na przykład dla czasowników będzie to sprowadzenie do formy bezokolicznikowej.
2. ***Mapping*** - to operacja odwzorowania wyszczególnionych tokenów do reprezentujących je indeksów w słowniku. Znacznie przy tym zmniejszamy rozmiar pamięci zajmowanej przez zadany tekst.
3. **Konstrukcja i ewaluacja modelu** - lista z indeksami (czyli pozycjami wyrazów w tablicy-słowniku) podana zostaje do modelu jako wejście modelu przetwarzania języka.

2.2 Modele analizy języka

Przedstawiono wybrane podejścia do zadania generowania tekstu.

2.2.1 Łańcuchy Markowa

Podejście oparte na idei procesów Markowa, czyli ciągów zdarzeń, w których prawdopodobieństwo zajścia kolejnego zdarzenia zależy tylko od zdarzenia go poprzedzającego. Kolejne etapy algorytmu generowania tekstu z użyciem tego podejścia zostały przedstawione na Rysunku 2.2. do zbudowania modelu wykorzystywana jest tablica słów i ich następników. do każdego pola tej tablicy przydzielona zostaje liczba wystąpień drugiego słowa.



Rysunek 2.2. Schemat generowania kolejnych słów w tekście z użyciem modelu Markowa

Załóżmy, że mamy zdanie „Ala ma kota i Ala ma psa". Zdanie to zmienione zostaje na postać stokenizowaną ‚Ala', ‚ma', ‚kot', ‚i', ‚Ala', ‚ma', ‚pies'.

Następnie budowana jest tablica ze słowami wraz z liczebnością ich następników. Przykład takiej tablicy podano w Tabeli 2.1.

Tablica 2.1. Tabela przejść między słowami

słowo \ następnik	Ala	ma	kot	i	pies
Ala	x	2	0	0	0
ma	0	x	1	0	1
kot	0	0	x	1	0
i	1	0	0	x	0
pies	0	0	0	0	x

Znając liczbę wystąpień słów będących następnikami danego słowa, model przewiduje kolejne słowo, wybierając te najbardziej liczne.

Każde kolejne słowo w procesie Markowa jest generowane według wzoru 2.1.

M — model Markowa

$$x_{n+1} = M(x_n) \tag{2.1}$$

Ograniczenie tego modelu pojawia się, gdy próbujemy rozszerzyć go o kolejne zapamiętane słowa. By to zrealizować, konieczne jest dodanie kolejnych kolumn do tabeli, co znacząco będzie zwiększać jej rozmiar. Jeśli potrzebne by było przechowywanie informacji o prawdopodobieństwach łączących więcej niż tylko pary słów do utworzenia łańcucha Markowa wyższego rzędu, zapotrzebowanie na pamięć byłoby jeszcze większe. Koszt pamięciowy (ZP — złożoność pamięciowa) każdego k ciągu słów w modelu Markowa można obliczyć w następujący sposób:

$$ZP = n^k \tag{2.2}$$

k — rząd łańcucha Markowa
n — liczba słów w słowniku

Podejście to jest szeroko wykorzystywane w modelowaniu procesów losowych. W porównaniu do innych metod generuje on nowy tekst znacznie bardziej wydajnie. By określić kolejne słowo, potrzebne jest jedynie przejrzenie odpowiedniej wartości w tablicy z wyrazami. Jednym z zastosowań tego modelu jest podpowiadanie kolejnych słów w niektórych telefonach komórkowych, gdzie zasoby obliczeniowe są bardzo ograniczone.

2.2.2 Rekurencyjne sieci neuronowe (RNN)

Model Markowa oparty został na podejściu, gdzie kolejne słowo generowane jest na podstawie prawdopodobieństwa uzyskiwanego z analizy statystycznej współwystępujących słów w tekście. Idea ta została rozwinięta w modelu rekurencyjnej

sieci neuronowej, która również może zostać wykorzystana do generowania tekstu [22]. Prosty schemat modelu został przedstawiony na Rysunku 2.3.

Oprócz prostej statystyki następników wewnątrz modelu tworzony jest wektor stanów ukrytych. Odpowiada on za przenoszenie informacji o kontekście analizowanego ciągu słów, zdania. Po wygenerowaniu tekstu jest on aktualizowany o nowo zdobyte informacje.

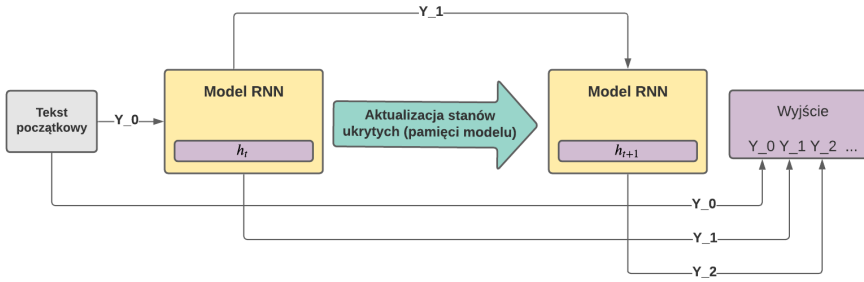
Kolejny stan wektora stanów ukrytych (h_t) w czasie t określony jest wzorem 2.3. Obrazuje on dodawanie kolejnego słowa x_t do wektora reprezentującego pamięć modelu.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{2.3}$$

Wygenerowane słowo (y_t) w czasie t określane jest formułą 2.4.

$$y_t = \sigma_y(W_y h_t + b_y) \tag{2.4}$$

Słowo jest generowane na podstawie parametrów wewnętrznych modelu (W,b) oraz stanu ukrytego (h_t), reprezentującego obecną wiedzę modelu o przetwarzanym tekście.
t : krok modelu
h_t : wektor stanów ukrytych w kroku t
x_t : dane wejściowe w kroku t
σ_h, σ_y : funkcje aktywacyjne, np. tanh
W, U, b : parametry modelu



Rysunek 2.3. Generowanie tekstu za pomocą zaktualizowanego modelu

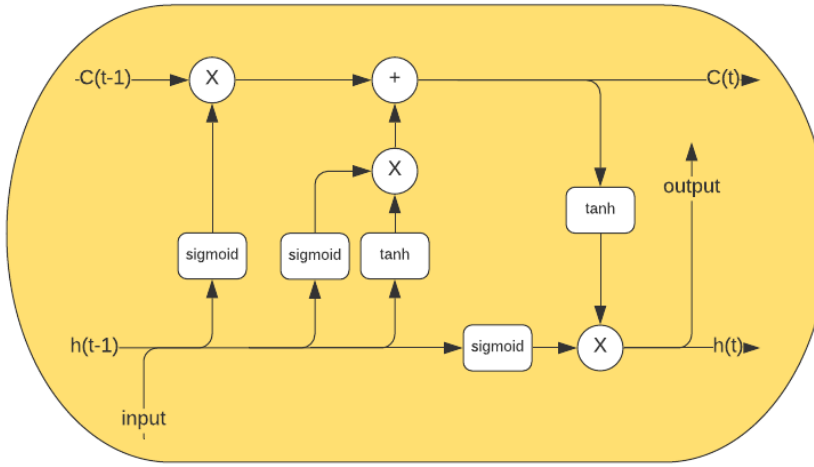
Problem, który w tym momencie się pojawia, to niewielka wiedza na temat semantyki zdania. By to poprawić, model RNN traktowany jest jako wielowarstwowy model sieci *FeedForward*, gdzie każda warstwa ukryta jest reprezentowana przez ten sam model RNN w czasach $t_1, t_2, ..t_n$. Dzięki temu parametry mogą być strojone z użyciem techniki zwanej propagacją wsteczną w czasie (*Backpropagation Through Time*) [7].

Teoretycznie model tego typu jest w stanie zapamiętać dowolną ilość informacji z kontekstu. Zapotrzebowanie pamięciowe takiego modelu jednak rośnie bardzo gwałtownie, co powoduje, że zastosowania praktyczne stają się mocno ograniczone. Jednym z rozwiązań, które się stosuje, jest podział rodzajów pamięci modelu RNN na krótkotrwałą, która skupia się na kilku poprzednich wyrazach oraz na długotrwałą, która przez długi czas przechowuje pojedyncze słowa, na które warto zwrócić uwagę. Stało się to podstawą do stworzenia rozwinięcia sieci rekurencyjnej, zawierającej bramki umożliwiające zapamiętywanie odległych od siebie podsekwencji.

2.2.3 Long short-term memory

Jest jednym z rekurencyjnych modeli przetwarzania tekstu, który umożliwia analizę odległych od siebie zależności. Podobnie jak RNN, model LSTM zapamiętuje poprzedni stan, ale dzięki specjalnej architekturze nie ma problemu z pamiętaniem oddalonych od siebie w czasie informacji.

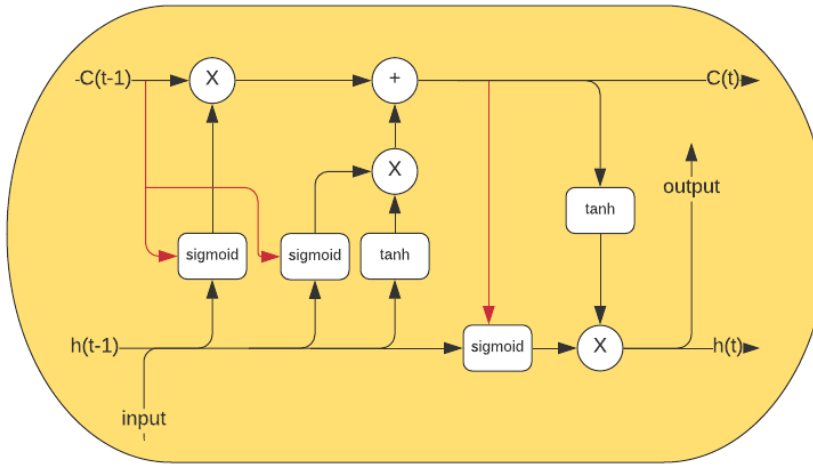
Standardowa budowa modułu w LSTM składa się z 4 warstw przedstawionych na Rysunku 2.4.



Rysunek 2.4. Schemat standardowego modelu LSTM

Jedną z najważniejszych części tego modelu jest tzw. komórka stanu długoterminowego (górna linia $C_{(t-1)} \rightarrow C_{(t)}$), która przechodzi najpierw przez *forget gate* (bramkę zapominania), gdzie usuwane są niepotrzebne informacje (operacja mnożenia), a następnie przechodzi przez *input gate* (bramkę wejściową), gdzie dodawane są nowe informacje (operacja dodawania), by w końcu udzielać się

przy określaniu wyniku aktualnej operacji i przekazać swój nowy stan kolejnemu modułowi. Bramka ta wskazuje wartość z danych wejściowych, która jest wykorzystywana do modyfikacji pamięci.



Rysunek 2.5. Schemat modułu standardowego LSTM z modyfikacją *Peephole Connection*

Forget gate bierze pod uwagę dane wejściowe oraz dane z poprzedniego stanu H(t-1) (stan krótkoterminowy) i kieruje je do sigmoidalnej funkcji aktywacyjnej, która zwraca wartości od 0 do 1, by określić, które informacje mają zostać usunięte z komórki stanu długoterminowego, a które mają zostać zachowane.

Druga bramka, *input gate*, bierze do siebie te same informacje, co *forget gate* i przepuszcza je równolegle przez funkcję sigmoidalną oraz funkcję tanh (tanges hiperboliczny). Wyniki tych funkcji są dodawane i przekazywane do komórki stanu długoterminowego.

Ostatnia bramka, *output gate*, składa się z danych wejściowych i poprzedniego stanu przekształconych funkcją sigmoidlaną oraz danych komórki stanu długoterminowego przekształconych funkcją tangesa hiperbolicznego. Wyniki tych funkcji są mnożone i przekazywane jako wyjście oraz stan krótkoterminowy następnego modułu.

Taka budowa modelu pozwala na jednoczesne istnienie pamięci długoterminowej i krótkoterminowej i wzajemne ich oddziaływanie na siebie.

Istnieje wiele modyfikacji powyższego modelu, jedną z nich jest przedstawiona na Rysunku 2.5 implementacja *peephole connection*, czyli dodanie do wybranych lub wszystkich bramek na wejściu danych z komórki stanu długoterminowego [5].

2.3 Model Transformer

W przeciwieństwie do modeli RNN sieci typu Transformer pozwalają na niesekwencyjną analizę danych wejściowych. W podejściu tym tekst wejściowy jest dodawany w jednym kroku, dzięki czemu większość wewnętrznych obliczeń może być wykonywana równolegle na urządzeniach takich, jak karty graficzne. Dlatego też trening modelu Transformer zajmuje znacznie mniej czasu.

Bazuje on na koncepcji tzw. uwagi lub skupienia (*attention*), początkowo wykorzystywanej w dziedzinie *computer vision*. Skupienie to służy do podkreślania istotnych dla modelu danych i maskowania tych mało istotnych. Dla zdjęć można to sobie zestawić z wyszukiwaniem rejonów zdjęcia, w których prawdopodobieństwo znalezienia interesującego nas elementu jest znacznie większe. Na przykład szukając drzwi na zdjęciu, model może się nauczyć, że powinien ich szukać w otoczeniu budynków, a nie, na przykład na niebie.

2.3.1 Reprezentacja wyrazów w modelu Transformer: *Word2vec*

Do wyznaczania wartości skupienia w modelu Transformer potrzebny jest sposób reprezentacji wyrazów, który umożliwia ich maszynowe przetwarzanie. Typowo do tego celu używane są wektory słów, np.: *Word2Vec* [6], które budowane są następująco:

1. Na początku tworzona jest macierz (rzadka) N na N, gdzie N jest liczbą wyrazów w słowniku. Każde słowo początkowo reprezentowane jest za pomocą *one hot encoding*: każde k-te słowo opisane jest jako wektor zawierający jedynkę na k-tej pozycji i zera na pozostałych.
   Model trenowany jest na podstawie relacji, między słowami podanymi w zbiorze treningowym, wykonywanych za pomocą prostych operacji matematycznych. w rezultacie, każde słowo otrzymuje pewien leksykalny zbiór cech, przedstawiony w swoim wektorze [11].
   - Paryż - Francja + Włochy = Rzym
   - wielki - większy + mały = mniejszy
2. W kolejnym kroku zmniejszony zostaje rozmiar wektorów. Jedną z wykorzystywanych technik jest algorytm rozkładu na wartości osobliwe (SVD) [14].

2.3.2 Skupienie (*Attention*)

Wartość skupienia (*attention*) odpowiada za określenie podobieństwa między wyrazami w tekście. Oblicza się ją, wykorzystując wzór analogiczny do średniej ważonej. do porównywania wyrazów naiwnym podejściem byłoby stosowanie metryk wyznaczających odległości edycyjne (np.: odległość Levenshteina), jednak dzięki wektorowej reprezentacji słów możliwe jest również wyznaczanie podobieństwa semantycznego.

Wzór 2.5, opisujący skupienie, określa, jak bardzo zadane słowo (K) pasuje do kontekstu (Q). Wynikiem jest wektor z prawdopodobieństwem wystąpienia każdego słowa ze słownika w zadanym kontekście.

$$Attention_j(Q, K, V) = softmax(S(Q_j, K_1) \cdot V_1, ..., S(Q_j, K_i) \cdot V_i) \tag{2.5}$$

Funkcja softmax pozwala zamienić punktację danego słowa na wartość prawdopodobieństwa od 0 do 1.

$$softmax(X) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.6}$$

Q (*query*) - słowo, dla którego szukamy podobnych wyrazów
K (*key*) - słowo, na którym obliczamy podobieństwo
V (*weight*) - waga danego połączenia słów. Tę wartość będziemy aktualizować za pomocą wstecznej propagacji

Funkcja S służy do obliczenia podobieństwa wyrazów, reprezentowanych przez wektory. Często wartość tej funkcji jest dzielona przez wartość $\sqrt{d}$, gdzie d jest wymiarem wektora słowa.

$$S(q, k) = q^T \cdot k \tag{2.7}$$

Podobieństwo wyrazów określamy za pomocą wartości odległości kosinusowej (czyli iloczynu skalarnego) wektorów q oraz k.

2.3.3 Budowa modelu Transformer

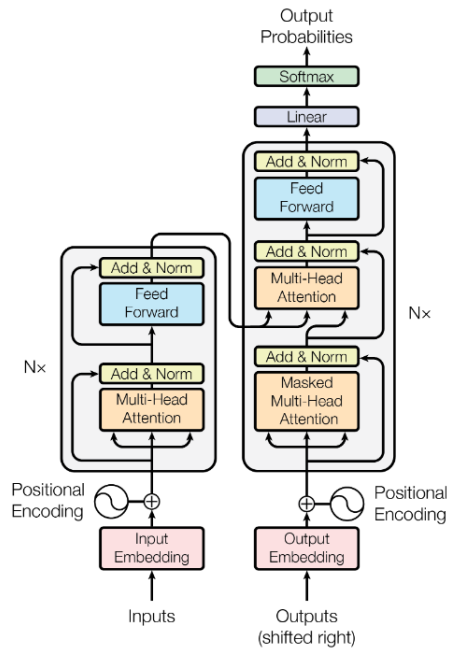
Oryginalny model Transformer jest podzielony na dwie części: *encoding* oraz *decoding*. Każda z nich zawiera kilka lub kilkanaście bloków enkodujących lub dekodujących (w publikacji obie części zawierają po 6 bloków [23]). Pierwotnie model Transfomer został zaprojektowany do tłumaczenia tekstu, wiec część *encoding* odpowiedzialna była za reprezentację tekstu wejściowego w oryginalnym języku. Część *decoding* służyła do predykcji kolejnych wyrazów według ich reprezentacji utworzonej przez enkoder.

Kodowanie pozycji wyrazów. We wzorze 2.5, opisującym skupienie, brakuje zawartej informacji o pozycji wyrazów w zdaniu. Jest ona bardzo istotna do właściwego przetwarzania tekstu.

W sieciach RNN problem ten nie występował, ponieważ dane były dodawane w sposób sekwencyjny. W modelu Transformer dane są przekazywane w jednej operacji, przez co potrzebny jest inny sposób przechowywania informacji o położeniu wyrazów w zdaniu.

Jednym z rozwiązań jest stworzenie wektora pozycji (*positional encoding*) o wymiarze równym podstawowemu wektorowi słowa (d). Informuje on, na jakiej pozycji w zdaniu znajduje się słowo. Poniżej podano wzór wyznaczający wartości

w wektorze pozycji na indeksach odpowiednio: parzystych (2.8) i nieparzystych (2.9).



Rysunek 2.6. Schemat modelu Transformer [23]

$$PE_{(pos,2i)} = \sin(pos(10000^{2i/d}) \tag{2.8}$$

$$PE_{(pos,2i+1)} = \cos(pos(10000^{2i/d}) \tag{2.9}$$

pos - pozycja słowa w zdaniu,
i - indeks w wektorze słowa [0,...,d],
d - rozmiar wektora reprezentującego słowo (*word2vec*) propagacji.

Powodem wyboru funkcji sinus/cosinus była ich nieliniowość, periodyczność oraz generowanie wartości w zakresie [0,1], przez co dane o pozycji nie ulegają rozmyciu. Kodowanie pozycji sprawia, że słowa, które znajdują się w podobnej odległości od siebie w tekście, mają zbliżone wektory pozycji, a te, które są od siebie oddalone, mają wartości różne na wielu różnych pozycjach.

Blok Enkoder. Blok enkoder (lewa połowa Rysunku 2.6) składa się z dwóch warstw:

- warstwa *self-attention* - służy do określenia stopnia ważności słowa w danym kontekście. Zbudowana jest ona z warstwy *Multi-Head Attention*. Jako że rezultatem tej operacji jest skalar, dodajemy jego wartość do początkowego wejścia warstwy i poddawany jest operacji normalizacji [1],
- warstwa *feed-forward* - służy do przetworzenia poprzednich wartości do formy lepiej zrozumiałej dla kolejnych bloków Transformera.

Wewnątrz warstwy *self-attention* wykorzystany został mechanizm tzw. *MultiHead Attention*. Odpowiada on za połączenie wartości skupienia dla h projekcji. Parametr h jest dobierany w sposób heurystyczny. Projekcje można sobie wyobrazić jako pewną formę filtrów, które maskują część tekstu, co znacząco poprawia jakość generowanych rezultatów.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.10}$$

$$head_i = Attention(QW_i^Q.KW_i^K, VW_i^V) \tag{2.11}$$

Wartości W_i (macierze) określają kolejne projekcje tekstu.

Warstwa *Feed-forward* składa się z dwóch pojedynczych transformacji liniowych połączonych funkcją aktywacyjną ReLU.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.12}$$

Blok Dekoder. Blok dekodera (prawa połowa Rysunku 2.6) składa się z trzech warstw:

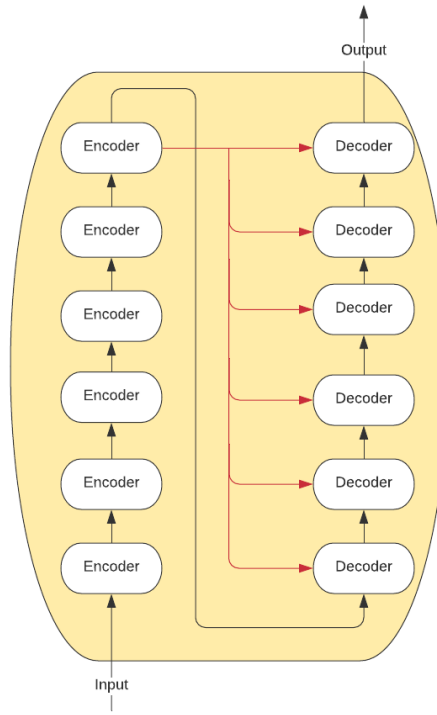
- warstwa *self-attention* - służy do tworzenia wektorów skupienia z pierwotnego tekstu,
- warstwa *encoder-decoder attention* - służy do łączenia obecnych wektorów skupienia z wektorami wygenerowanymi przez enkodery,
- warstwa *feed-forward* - analogicznie jak w enkoderze poprawia reprezentacje wektorów dla kolejnych bloków dekodera.

Ważną różnicą w porównaniu do bloku enkodującego jest maskowanie parametrów w macierzy z wartościami skupienia. Dla każdego wyrazu maskowane są wartości, do których nie może być dostępu, ponieważ model nie powinien znać tych słów, jako że występują przed nim. Operacja ta jest wykonywana poprzez wypełnienie części górnej trójkątnej bez wartości diagonalnych, wartościami $-\infty$. Ponieważ atencja jest liczona według funkcji softmax, to wartością skupienia dla takich słów będzie zero.

2.3.4 Uproszczony tok pracy

Dane wejściowe w postaci tekstu trafiają do pierwszego enkodera, który jako jedyny posiada możliwość zamiany tekstu w wektory. Następnie wektory przetwarzane są za pomocą kolejnych bloków enkodera. Dane z ostatniego enkodera

następnie trafiają do warstwy *Encoder-Decoder Attention* dekoderów. Przetwarzany tekst (w postaci wektora) trafia do dolnego dekodera, który w analogiczny sposób przetwarza wektory, przepuszczając je przez swoje warstwy i zwraca wynik do kolejnego bloku dekodera. Rysunek 2.7 przedstawia graficzną reprezentację powyższych przejść.



Rysunek 2.7. Schemat budowy modelu Transformer

Gdy wektory zostaną przetworzone przez wszystkie enkodery i dekodery, są one zwracane do ostatniej warstwy, zamieniającej wektor, z powrotem, w stokenizowany tekst.

2.3.5 Destylacja wiedzy modeli

Modele Transformer są nieporęczne: mają spore zapotrzebowanie na moc obliczeniową oraz pamięciową, przez co ich zastosowanie na słabszym sprzęcie jest utrudnione. Jest jednak sposób na przenoszenie tej wiedzy na mniejsze modele za pomocą techniki zwanej też jako *destylacja wiedzy*.

Współczesne modele językowe uczone są często na ogromnych zbiorach danych. Model GPT-2 został przetrenowany na zbiorze danych *WebText*, zwierającym 40 GB danych tekstowych pozyskanych z linków wychodzących z portalu *www.reddit.com*. Spora część tych danych posiada bardzo powtarzalne dane, które niewiele wnoszą do modelu. Ponadto dane są często *twardo* klasyfikowane. Oznacza to, że z danych uczących można wyciągnąć jedynie jedną odpowiedź, nie biorąc pod uwagę niepewności wyników.

Mniejszy model, w przeciwieństwie do większego odpowiednika, jest trenowany na danych z rozmytymi klasyfikacjami. Dzięki temu pozyskuje on zdecydowanie więcej informacji z tych samych danych wejściowych [8]. Różnice w wynikach twardej i rozmytej klasyfikacji pokazano poniżej:

Tekst wejściowy: Ala ma [...]
Predykcja twarda: Ala ma [kota]
Predykcja rozmyta: Ala ma [kota, psa, papugę]

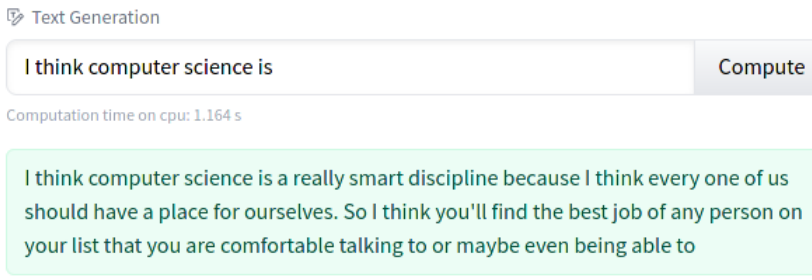
Dodatkowo, aby model mógł pozyskiwać jeszcze większą liczbę informacji z danych niejednoznacznych, modyfikuje się dodatkowo wewnętrzną funkcję softmax 2.6, obliczającą prawdopodobieństwa wyjść, dodając dodatkowy parametr T o nazwie *Temperatura*, który odpowiada za szerszy rozkład prawdopodobieństw określony wzorem 2.13.

$$softmaxDistilled(x) = softmax(x/T) \tag{2.13}$$

Temperatura sprawia, że model jest znacznie bardziej czuły na różnorodność w danych treningowych. Dzięki temu uczy się on szybciej, ale sprawia to, że wprowadzany jest szum i rezultaty uzyskiwane przez model mogą ulec pogorszeniu.

Dzięki tej technice możliwe jest stworzenie modelu takiego jak *DistilBert* [18], który posiadając jedynie 60% wewnętrznych parametrów jest w stanie zachować ok. 95% oryginalnej wiedzy.

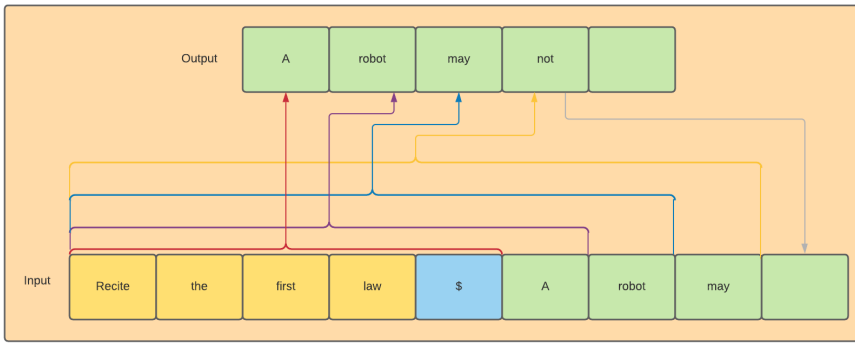
2.3.6 Model GPT-2



Rysunek 2.8. Wykorzystanie modelu GPT na platformie huggingface.

Generative Pre-trained Transformer [16, 21], został opublikowany przez *OpenAI* jako sukcesor modelu GPT. Stworzony został z myślą o predykcji kolejnego wyrazu po podanym tekście. Przykładowe działanie przedstawione zostało na Rysunku 2.8.

Pomimo architektury opartej na Transformerach model GPT-2 korzysta tylko z dekoderów (najmniejszy opublikowany GPT-2 small posiada ich 12). Model GPT-2 produkuje dane wyjściowe sekwencyjnie po jednym wyrazie, a w trakcie pracy automatycznie zbiera słowa z *output'u* do danych wejściowych. Mechanizm ten nazywany jest *auto-regresją* i został przedstawiony na Rysunku 2.9.



Rysunek 2.9. Schemat działania modelu GPT-2

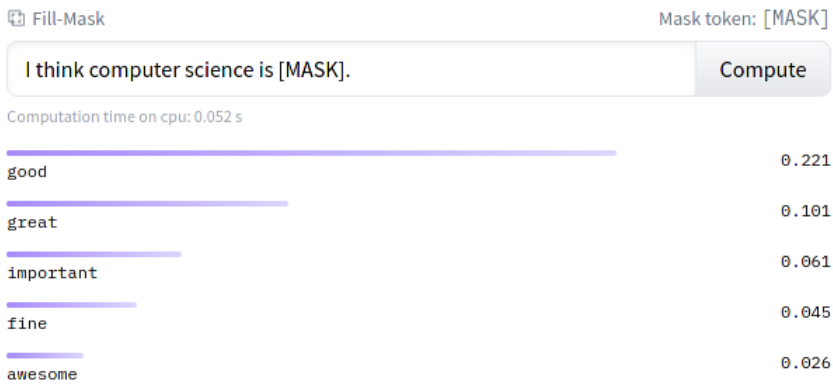
Budowa dekoderów w modelu GPT-2 różni się od standardowej architektury. Pozbyto się tu warstwy *encoder-decoder attention*, ponieważ bez enkoderów nie ma ona racji bytu. Zmianę przeszła również warstwa *self-attention* i stała się warstwą *masked self-attention*. Zasadniczo nie różni się ona od oryginału poza funkcją maskowania wszystkich wyrazów po prawej stronie aktualnie przetwarzanego tokenu.

Ze względu na swoją budowę model ten specjalizuje się w predykcji kolejnych wyrazów, mając podany na wejściu tekst bazowy. Doświadczalnie pokazano jego skuteczność w tym obszarze. Próby wykorzystania tego modelu do innych zadań, takich jak, uzupełnianie luk w zdaniach czy tłumaczenie tekstu przy odpowiednim treningu daje zadowalające wyniki, lecz odbiegają one od wydajności i poprawności innych modeli ogólnego użytku [15].

2.3.7 Model BERT

Model wykorzystywany jest m.in. w wyszukiwarce Google, by lepiej zrozumieć złożone i nieoczywiste zapytania, tworzone przez użytkowników [12]. Zbudowany został na podstawie modelu Transformer, na części odpowiedzialnej za *encoding*. Dzięki temu jesteśmy w posiadaniu pewnej wiedzy o najczęstszych poprzednikach oraz następnikach wszystkich wyrazów. Część dekodująca służy w tym

modelu do generowania predykcji pod określone zadania. Przykładowe działanie przedstawione zostało na Rysunku 2.10.



Rysunek 2.10. Wykorzystanie modelu BERT na platformie huggingface: `https://huggingface.co/bert-base-uncased`.

Jako że część enkodująca nie jest w stanie sama z siebie wygenerować tekstu, stosowane są tak zwane *tokeny specjalne*, które reprezentują między innymi: początek zdania, koniec zdania czy token maski, który reprezentuje brak słowa.

Potrafi on wyciągnąć zdecydowanie więcej informacji z tekstu w porównaniu do modeli korzystających z części dekodującej architektury Transformer (GPT). Model BERT posiada też znacznie mniej parametrów od innych modeli z rodziny Transformer. Dzięki temu zarówno rozmiar, jak i czas treningu ulega znacznemu zmniejszeniu.

Teoretycznie generowanie tekstu przez ten model jest wykonalne. Wystarczy bowiem dodać na koniec zdania tokeny masek. Niestety, wyniki takiej generacji tekstu zdecydowanie odstają w porównaniu do modeli takich jak GPT, ponieważ model BERT nie jest przygotowany na niedobór informacji o kolejnych wyrazach. Innym ograniczeniem jest obowiązek określenia z góry rozmiaru zdań wejściowych (wykorzystany tu model posiadał miejsce na 512 tokenów) [4]. Sprawia to, że model jest w stanie pracować tylko i wyłącznie na krótkim tekście. Dzieje się tak, ponieważ złożoność obliczeniowa rośnie kwadratowo względem liczby słów w zdaniu.

2.3.8 Model laMDA

[2](*Language Model for Dialogue Applications*) został stworzony przez Google, jest to jeden z nowszych (zaprezentowany 18 maja 2021), nieopublikowanych modeli korzystających z architektury Transformer. Działa na podobnej zasadzie co BERT czy GPT-2.

Model laMDA został stworzony z myślą o prowadzeniu długich konwersacji, w których często zmieniany jest temat lub nie jest zachowana ciągłość wypowiedzi poprzez wtrącenia. Powoduje to problem w szybkim rozpoznawaniu kontekstu zdań i tematyki wypowiedzi. By zaradzić tym zjawiskom, zaproponowano nowe rozwiązanie w architekturze Transformera.

Omawiany model stosuje innowacyjne podejście, zastępując całkowicie warstwę *self-attention* nową warstwą lambda. Ma ona tak jak oryginalna warstwa *self-attention*, na celu analizować tekst z wejścia i tworzyć dane mówiące o kontekście. Różnica polega na tym, że zamiast przekazywać te dane do kolejnych warstw, są one zmieniane w funkcję lambda, która 'nakładana' jest na dane wejściowe. Powoduje to uwzględnienie zarówno zawartości wyrazu, jego kontekstu jak i pozycji, nie zwiększając zbytnio wymagań pamięciowych.

Rozwiązanie to sprawdza się zarówno w analizie pełnych tekstów, jak i tych zawierających puste pola (*mask*) przy niskim zużyciu pamięci, nawet przy długich tekstach. W odróżnieniu od innych modeli ze względu na swoje przeznaczenie laMDA trenowany był prawie wyłącznie na zbiorze danych składającym się z dialogów.

2.3.9 Modyfikacje modelu BERT

Pojawiły się modele usprawniające model BERT pod względami takimi, jak szybkość przetwarzania danych czy jakość wyników.

RoBERTa [10] to wersja modelu BERT, która wprowadza nową technikę treningu, polegającą na dynamicznych tokenach *mask*, czyli w każdej pętli treningu dane zdanie posiada maski w innych, losowo wyznaczanych, miejscach. Dodatkowo model ten został przetrenowany na dziesięciokrotnie większej bazie danych. Efektem tych zabiegów jest poprawa wydajności BERT-a o średnio 10%.

AlBERT [9] model ten optymalizuje swoją pracę, dzieląc parametry między poszczególnymi warstwami enkoderów oraz faktoryzując swoje dane. Faktoryzacja danych polega na zamianie wektora danych w dwa mniejsze wektory, które po połączeniu dają wektor zbliżony do oryginalnego. Powoduje to ograniczenie wymaganej pamięci i parametrów kosztem utraty części danych [19]. W efekcie, zachowując znikomą różnicę w poprawności wyników, zmniejszona zostaje liczba parametrów o prawie 90%.

DistilBERT model BERT wykorzystuje mechanizm tzw. *Transfer Learning*, który polega na przeniesieniu części wiedzy z dużego modelu do jego mniejszego odpowiednika [18]. Wykorzystano lekko zmodyfikowaną technikę destylacji wiedzy (2.3.5). Udało się w ten sposób zredukować model $BERT_{LARGE}$ zawierający 340 milionów parametrów do modelu z 66 milionami parametrów. Model ten zachowuje przy tym 95% wiedzy i jest o 60% szybszy.

MegatronBERT [20] to model bardzo niepraktyczny, zaprojektowany przez firmę NVIDIA. Posiada 3.9 miliarda parametrów, trenowany równolegle na 512 GPU utrzymując 15.1 PetaFlopów. Aby skorzystać z modelu, potrzebny jest superkomputer, a poprawność wyników zwiększa się o niewiele punktów procentowych. Został on stworzony w celach eksperymentalnych, by odpowiedzieć na pytanie, czy zwiększenie liczby parametrów modelu poprawi jego rezultaty oraz aby sprawdzić, z jakimi problemami można się spotkać przy modelach tak ogromnej skali.

herBERT [17] to model BERT z niewieloma zmianami, stworzony jedynie w celu przetrenowania na polskim tekście i zbadania, jak modele przetwarzania tekstu naturalnego dają sobie radę z językiem polskim (i przy okazji innymi językami). Efekty pracy były dość zadowalające, jeśli chodzi o analizę tekstu i poprawność odpowiedzi.

2.4 Podsumowanie modeli Transformer

Przedstawione w 2017 r. pierwsze modele Transformerów zrewolucjonizowały środowisko NLP swoją wydajnością i metodą analizy danych. Od tamtego czasu stoją w samym centrum badań z przetwarzania języka naturalnego. Nowe pomysły pojawiają się praktycznie co miesiąc, starając się w różny sposób usprawnić i udoskonalić system. Przełomowe rozwiązania prezentowane są w krótkich odstępach czasu przez wielkie korporacje, jak Google czy Facebook. Podkreśla to tylko, jak bardzo praktyczna jest to dziedzina sztucznej inteligencji. Oglądając dotychczasowe tempo rozwoju, można zaryzykować stwierdzeniem, że Transformery zostaną jeszcze przez jakiś czas, wciąż ewoluując i zmieniając swoją formę. Jednocześnie nie można wykluczyć, że ta metoda nie zostanie niedługo wyparta przez kolejną innowację w tej dziedzinie. Warto zaznaczyć kilka obserwacji dotyczących dotychczasowych zastosowań Transformerów. Pomimo swoich zalet modele Transformer mają też szereg wad:

- *modele mają bardzo dużo parametrów* - każdy blok w modelu Transformer składa się z wielu mniejszych elementów. Podstawowy model zawiera aż 6 bloków enkodujących i 6 dekodujących. Każdy nich posiada własną sieć *feed-forward* oraz wagi służące do obliczania skupienia (2.5). Dla modeli małych jest to pewna bariera nie do przejścia. Mimo to modele Transformer są bardzo popularne z powodu swojej bardzo dobrej skalowalności, ponieważ złożoność obliczeniowa generowania tekstu jest liniowa dla parametru stanów ukrytych,
- *modele są bardzo wolne* - z powodu swojej sporej złożoności oraz obowiązku wykonania wielu dodatkowych operacji jak (2.5), czy (2.10), wygenerowanie nawet krótkiego tekstu zajmuje takim modelom bardzo dużo czasu,
- *modele uczą się bardzo długo* - z powodu ogromnej liczby parametrów wewnątrz modelu, wymagane jest dostarczenie bardzo dużej ilości danych,
- *modele zwracają uwagę na nieistotne elementy zdania* - model BERT najczęściej zwraca uwagę na nieistotne, z ludzkiego punktu widzenia, części zdania

[3] (głównie na token początkowy i końcowy, ale też sporo uwagi jest zwracana zaimkom oraz spójnikom),
– *duża liczba wewnętrznych elementów* - w porównaniu do modeli, takich jak RNN czy łańcuchy Markowa, w tym modelu znajdziemy zdecydowanie większą liczbę wewnętrznych elementów, dzięki czemu prawdopodobieństwo błędu powstałego podczas implementacji jest znacznie większe. Również modyfikacja wewnętrznych parametrów, w przypadku niesatysfakcjonujących wyników, jest znacznie utrudniona. Komplikuje to, a czasami wręcz uniemożliwia, wsteczne śledzenie modelu w poszukiwaniu przyczyn niepoprawnych rezultatów.

Modele o architekturze Transformer znajdują szereg praktycznych zastosowań w wielu dziedzinach. Jednymi z najpopularniejszych to: odpowiadanie na pytania, klasyfikacja tekstów, tłumaczenie tekstu i streszczanie tekstu. Wielu użytkowników nieświadomie wchodzi w interakcje z aplikacjami korzystającymi z modeli Transformer, np. podczas korzystania z translatora, rozmowy z *chatbotami.*

2.5 Przykładowe zastosowanie: transformacja nastroju tekstu

Do demonstracji możliwości sieci typu Transformer zbudowano aplikacje, której celem było wykonanie liniowej transformacji wprowadzonego tekstu. Pomysł ten nie jest nowy i występuje np. w powieści 1984 [13] autorstwa Georga Orwella, gdzie partia rządząca, za pomocą instrumentu Ministerstwa Prawdy, zmieniała nastrój tekstów w swoich archiwach wobec państw, co do których zmieniła swoje nastawienie. Aplikacja jest w stanie na małych danych i przy bardzo ograniczonych zasobach obliczeniowych zwracać satysfakcjonujące wyniki.

Dane do treningu modeli zostały pozyskane z bazy danych portalu IMDB i składają się one z ponad 80 MB danych tekstowych[1]. w aplikacji dokonano podziału według tej metryki na dwa zbiory danych testowych: negatywna i pozytywna. Ilość danych w każdej z grup jest bardzo zbliżona.

2.5.1 Wykorzystane biblioteki / narzędzia

– **Transformers**[2] - framework ułatwiający tworzenie, przechowywanie oraz użytkowanie modeli typu Transformer,
– **Pytorch**[3] - biblioteka służąca do budowania aplikacji wykorzystujących uczenie maszynowe,
– **Nltk**[4] – platforma dla aplikacji z dziedziny przetwarzania języka. Zawiera ona wiele przydatnych funkcji, takich jak tokenizacja czy określanie części mowy w zdaniu,

[1] `https://huggingface.co/datasets/imdb`
[2] `https://huggingface.co/transformers/`
[3] `https://pytorch.org/`
[4] `https://www.nltk.org/`

– **Ipywidgets**[5] - jest to biblioteka pozwalająca tworzyć interaktywne widgety w środowisku *Jupyter notebook*.

Użyto trzech modeli DistilBERT, poddanych operacji *fine-tune*. Dwa z nich posłużyły do predykcji wyrazów w zdaniu. Jeden z nich został poddany treningowi na pozytywnych recenzjach, drugi na negatywnych. Podczas treningu losowe wyrazy (ok. 15%) w tekście zostały zamaskowane i porównane z oryginalnymi wartościami. Trzeci model służy do klasyfikacji tekstu. Został on przetrenowany na tych samych danych i klasyfikuje recenzje do grupy pozytywnej lub negatywnej.

Użyty został model perceptron z modułu NLTK, który posłużył do przydzielenia do słów tak zwanych znaczników części mowy (*Part-of-speech*, inaczej *POS taggs*). Według niego określono kolejność zamiany słów w tekście.

Do tokenizacji zdań wykorzystano program Sacremoses, który jest wbudowany w biblioteki *nltk* oraz *Transformers* [6].

2.5.2 Działanie aplikacji

Aplikacja oczekuje na podanie przez użytkownika zdania, którego nastrój chcemy zmienić. Ze zdania wybierane są wyrazy, które mogą mieć wpływ na jego nastrój. Następnie są one sortowane według ich pozycji na liście ważności części mowy. Lista ta została stworzona według heurystycznie obranej kolejności: przymiotniki, przyimki, partykuły, czasowniki, rzeczowniki, spójniki, zaimki osobowe. Następnie oczekuje się od użytkownika decyzji odnośnie kierunku zmiany nastroju zdania. Tekst trafia do pętli, w której odpowiedni model (pozytywny lub negatywny) proponuje kolejne słowa do osiągnięcia pożądanej polaryzacji tekstu. w przypadku, gdy ocena jest na tyle wysoka lub niska, że dalsze zmiany polaryzacji są niemożliwe, aplikacja wychodzi z pętli przedwcześnie.

2.5.3 Wyniki

Stworzono dwa modele DistilBERT poddane operacji *fine-tune* [7]. Trzeci przetrenowany model został pobrany ze strony huggingface[8].

Aplikacja jest dostępna na platformie github[9]. Stworzone 2 modele, posiadające po ok. 250 mb, zostały udostępnione na platformie huggingface[10] [11].

[5] `https://ipywidgets.readthedocs.io/en/latest/`

[6] Tokenizer: `https://github.com/alvations/sacremoses`

[7] Kod tworzącego modele: `https://github.com/michalwilk123/huggingface-bert-finetune-example-pl`

[8] Użyty model: `https://huggingface.co/textattack/bert-base-uncased-imdb`

[9] `https://github.com/michalwilk123/nlp-transformer-app-pl`

[10] Model negatywny: `https://huggingface.co/michalwilk123/distilbert-imdb-negative`

[11] Model pozytywny: `https://huggingface.co/michalwilk123/distilbert-imdb-positive`

Poniżej zaprezentowano przykładowe wyniki uzyskane przez zastosowanie modelu.

Wynik początkowy
krok 0, nastrój: 0.99

i absolutely love this movie! i do think it is great. Watching this film was a great experience. The acting was also very well made.

Transformacja negatywna
krok 1, nastrój: 0.9898

i absolutely love this movie! i do think it is funny. Watching this film was a great experience. The acting was also very well made.

Transformacja negatywna
krok 2, nastrój: 0.9896

i absolutely love this movie! i do think it is funny. Watching this film was a good experience. The acting was also very well made.

Transformacja negatywna
krok 8, nastrój: 0.3038

i do see this movie! i do hope it is funny. Watching this film was a good experience. The acting was not even properly made.

Nastrój jest wartością zwracaną z modelu klasyfikatora BERT. Wartość **1** nastroju oznacza nastrój bardzo pozytywny, **0** oznacza nastrój bardzo negatywny. Podkreślone zostały dodatkowo słowa, które uległy zmianie.

Wynik początkowy
krok 0, nastrój: 0.0039

This movie is awful! i dislike every second of it!

Transformacja pozytywna
krok 1, nastrój: 0.9795

This movie is good ! i dislike every second of it!

Transformacja pozytywna
krok 2, nastrój: 0.9938

This movie is good! i enjoyed every second of it!

Wynik początkowy
krok 0, nastrój: 0.9696

Alice has a cat.

Transformacja negatywna
krok 1, nastrój: 0.9093

Alice was a cat.

2.5.4 Dyskusja uzyskanych wyników

Zaimplementowane modele uczone są na ograniczonej liczbie danych, co wyraźnie przekłada się na zwracane przez nie wyniki i szybki zanik powiązania między proponowanymi przez nie wyrazami.

Model klasyfikatora został stworzony za pomocą tak zwanej twardej klasyfikacji, przez co ma wyraźną tendencję do zwracania skrajnych ocen. Dlatego największa widoczna różnica przy zmianie polaryzacji zdania ma miejsce do około 10-20 kroków. Poza tym przedziałem zdania zaproponowane przez model pozytywny i negatywny szybko zbiegają się do pozytywnie lub negatywnie spolaryzowanej formy.

Do ograniczeń podejścia należy zaliczyć, widoczny w ostatnim przykładzie, brak mechanizmu wychwytującego *neutralność* tekstu.

Aplikacja może zostać rozwinięta w szeregu kierunków. Najprostszym sposobem na poprawę wyników byłby, na przykład, stworzenie modelu klasyfikatora zwracającego bardziej rozmyte wyniki. Kolejnym sposobem na poprawę wyników byłoby wykorzystanie modelu bardziej nastawionego na odnajdywaniu kontekstu w zamaskowanych wyrazach jak XLNET [24]. Dzięki temu aplikacja byłaby w stanie dodawać/usuwać odpowiednie słowa, które pasowałyby do ogólnego kontekstu zdania.

Aplikacja ta jest przykładem zastosowania sieci neuronowej do transformacji tekstu i stanowi ona dobry punkt wejścia do rozwoju metod generowania języka naturalnego. W przyszłości można ją rozwinąć o interaktywne sterowanie nastrojem arbitralnego tekstu, co z pewnością będzie interesującą funkcjonalnością, choć trudnym do realizacji zadaniem.

Bibliografia

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization (2016).
2. Bello, I.: Lambdanetworks: Modeling long-range interactions without attention (2021).
3. Clark, K., Khandelwal, U., Levy, O., Manning, C.D.: What does bert look at? an analysis of bert's attention (2019).
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2019).

5. Gers, F.A., Schraudolph, N.N., Schmidhuber, J.: Learning precise timing with lstm recurrent networks. Journal of machine learning research **3**(Aug), 115–143 (2002).
6. Goldberg, Y., Levy, O.: word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 (2014).
7. Guo, J.: Backpropagation through time. Unpubl. ms., Harbin Institute of Technology **40**, 1–6 (2013).
8. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network (2015).
9. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R.: Albert: A lite bert for self-supervised learning of language representations (2020).
10. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: RoBERTa: A robustly optimized BERT pretraining approach (2019).
11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013).
12. Nayak, P.: Understanding searches better than ever before, `https://blog.google/products/search/search-language-understanding-bert/`.
13. Orwell, G.: 1984 (1949).
14. Peter, R., Shivapratap, G., Divya, G., Soman, K.: Evaluation of svd and nmf methods for latent semantic analysis. International Journal of Recent Trends in Engineering **1**(3), 308 (2009).
15. Qu, Y., Liu, P., Song, W., Liu, L., Cheng, M.: A text generation and prediction system: Pre-training on new corpora using bert and gpt-2. In: 2020 IEEE 10th International Conference on Electronics Information and Emergency Communication (ICEIEC). pp. 323–326. IEEE (2020).
16. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2018), `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`.
17. Rybak, P., Mroczkowski, R., Tracz, J., Gawlik, I.: Klej: Comprehensive benchmark for polish language understanding (2020).
18. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter (2020).
19. Schmidhuber, Jürgen: Deep learning in neural networks: An overview. Neural Networks **61**, 85–117 (Jan 2015). https://doi.org/10.1016/j.neunet.2014.09.003, `http://dx.doi.org/10.1016/j.neunet.2014.09.003`.
20. Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., Catanzaro, B.: Megatron-lm: Training multi-billion parameter language models using model parallelism (2020).
21. Solaiman, I., Brundage, M., Clark, J., Askell, A., Herbert-Voss, A., Wu, J., Radford, A., Krueger, G., Kim, J.W., Kreps, S., McCain, M., Newhouse, A., Blazakis, J., McGuffie, K., Wang, J.: Release strategies and the social impacts of language models (2019).

22. Sutskever, I., Martens, J., Hinton, G.E.: Generating text with recurrent neural networks. In: ICML (2011).
23. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017).
24. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q.V.: Xlnet: Generalized autoregressive pretraining for language understanding (2020).

3. Modelowanie ciągów danych z użyciem sieci neuronowych

Adam Wawrzyński

wawrzynski.adam@protonmail.com

Streszczenie
Rozdział opisuje problematykę przetwarzania ciągów danych. Opisane zostały typy ciągów danych: dane sekwencyjne, sekwencje czasowe oraz przebiegi czasowe. Przedstawiona została architektura sieci rekurencyjnych, problem zanikającego i eksplodującego gradientu, a także architektury komórek rekurencyjnych LSTM oraz GRU, które ten problem rozwiązują. W dalszej części opisana została architektura typu Transformer wraz z przykładowymi wykorzystującymi ją modelami: BERT, GPT, T5. Następnie zaprezentowano działanie modeli autoregresyjnych. Na końcu opisano metody dekodowania używane w zadaniach generacyjnych.

Słowa kluczowe: przetwarzanie języka naturalnego, sieci rekurencyjne, GRU, LSTM, Transformer

3.1 Przetwarzanie ciągów danych

Początkowo architektury sieci neuronowych były dedykowane do przetwarzania danych macierzowych, takich jak obrazy lub zestawy sztucznie stworzonych cech reprezentujących obiekty. Przełomowe w ich przetwarzaniu okazały się architektury sieci splotowych [12], które pozwalały znacząco poprawić wyniki w problemach związanych z między innymi klasyfikacją obrazów. Ze względu na odmienną charakterystykę danych sekwencyjnych, konieczne było opracowanie nowego sposobu na modelowanie takiego typu danych, który pozwoliłby efektywnie wykorzystać ich cechy. W tym rozdziale opisane zostały cechy różnych typów ciągów danych oraz sposoby ich przetwarzania.

3.1.1 Typy danych

Ciągi danych stanowią ogólną grupę danych, dla których cechą wyróżniającą jest uzależnienie znaczenia ciągu od kolejności występowania w nim próbek. Wśród ciągów danych możemy wydzielić 3 podgrupy, które różnią się między sobą pewnymi cechami. Są to dane sekwencyjne, sekwencje czasowe oraz przebiegi czasowe.

Dla danych sekwencyjnych istotna jest kolejność próbek w ciągu danych, a także brak występowania wymiaru czasu. Kolejne symbole nie są uzależnione od siebie czasowo, lecz istnieją od niego niezależnie. Przykładem takich danych

są szyfrogramy, sekwencje DNA lub wyrazów, w których istotne są tylko kolejne znaki w ciągu.

Dla sekwencji czasowych istotna jest zarówno kolejność próbek w ciągu danych, jak również czas ich wystąpienia. Wymiar czasu jest jedną z cech ciągu. Przykładem takiego typu danych może być zestaw informacji opisujących profil klienta za pomocą jego ostatnich aktywności, odwiedzanych stron lub dokonywanych zakupów wraz z czasem ich wystąpienia.

Ostatnim typem ciągów danych są przebiegi czasowe, dla których istotna jest kolejność próbek, a odległość w wymiarze czasu między kolejnymi próbkami jest wartością stałą. Jako przebiegi czasowe możemy traktować wszelkie notowania giełdowe lub cykliczne pomiary uzyskane z czujników.

3.1.2 Metody przetwarzania

Do przetwarzania ciągów danych zwykle używane są metody statystyczne oraz metody oparte o sztuczne sieci neuronowe. Metody statystyczne wykorzystywane do modelowania ciągów danych skupiają się na mierzeniu i porównywaniu różnych parametrów, takich jak wariancja, odchylenie standardowe lub średnia krocząca. W odróżnieniu od nich, metody oparte o sztuczne sieci neuronowe wykorzystują odmienne podejście polegające na kontekstowym odwzorowaniu każdego elementu ciągu na wielowymiarową przestrzeń. Następnie każdy z tych punktów, reprezentujących element ciągu danych, jest agregowany do postaci punktu reprezentującego cały ciąg. Na podstawie zarówno reprezentacji każdego elementu, jak również całego ciągu danych, dokonywane są operacje mające na celu rozwiązanie zdefiniowanego zadania, na przykład klasyfikacji elementów ciągu, klasyfikacji ciągu danych lub przewidywania kolejnego elementu. Przykładami rzeczywistych zadań odpowiadających wymienionym wyżej przykładom mogą być: klasyfikacja części mowy dla każdego wyrazu w zdaniu, klasyfikacja sekwencji DNA oraz przewidywanie przyszłych wartości akcji w notowaniach giełdowych.

Ze względu na sukces uczenia głębokiego w modelowaniu tego typu danych w dalszej części pracy przedstawione zostaną kolejne generacje architektur sieci neuronowych.

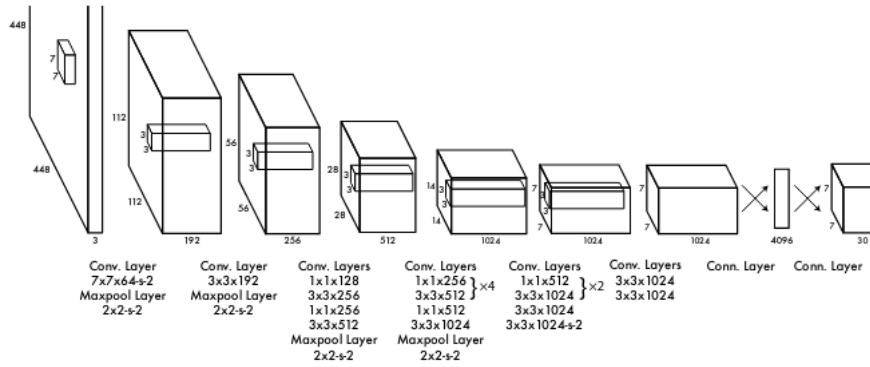
Na przestrzeni lat algorytmy odwzorowania elementów ciągu na wielowymiarową przestrzeń podlegały ewolucji. W części 3.2 przedstawione zostały kolejne generacje sieci neuronowych służących do reprezentacji ciągów danych: proste sieci rekurencyjne, sieci LSTM, sieci GRU, sieci typu Transformer, BERT, GPT oraz T5. Architektury te zostały stworzone głównie z myślą o przetwarzaniu języka naturalnego, ale mogą być z powodzeniem zastosowane do przetwarzania dowolnych ciągów danych.

3.2 Sieci rekurencyjne

Architekturą sieci neuronowych modelującą cechy charakterystyczne ciągów danych były sieci rekurencyjne. Dzięki zastosowaniu mechanizmu rekurencji moż-

liwe było modelowanie sekwencyjnej natury danych. W tej części omówiony został mechanizm rekurencji oraz dwie najpopularniejsze architektury komórek rekurencyjnych. Dokonane zostało porównanie przepływu informacji w sieciach rekurencyjnych oraz splotowych. W dalszej części pracy opisano działanie architektury typu Transformer oraz modele, które są na niej oparte. Na końcu tej części omówiono działanie modeli autoregresyjnych oraz metod dekodowania.

Przepływ danych w wielowarstwowych perceptronach oraz sieciach splotowych odbywa się w kierunku od wejścia do wyjścia sieci przechodząc przez kolejne warstwy sieci. Na rys. 3.1 przedstawiony przepływ odbywa się od lewej do prawej strony.

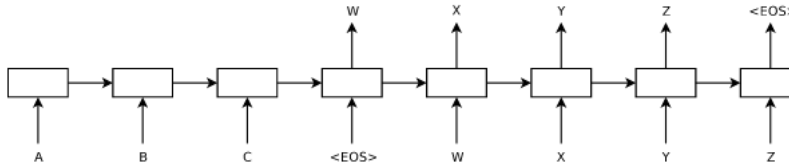


Rysunek 3.1. Architektura sieci splotowych [17]

Sieci rekurencyjne umożliwiły modelowanie ciągów przez przekazywanie informacji między sąsiednimi komórkami wewnątrz tej samej warstwy sieci, co zostało zaprezentowane na rys. 3.2. Ideą stojącą za tą architekturą było założenie, że kodowanie aktualnego elementu ciągu wraz z zakodowaną historią wszystkich poprzednich elementów pozwoli lepiej modelować ciągi danych. Dzięki zastosowaniu tego mechanizmu możliwe było modelowanie relacji pomiędzy odległymi od siebie elementami ciągów, przykładowo relacje między kolejnymi wyrazami w zdaniu. Zakodowany stan można traktować jako pamięć krótkotrwałą, analogiczną do tej występującej w ludzkim mózgu, a więc przechowującej informacje przez krótki czas. Porównanie to dobrze oddaje ograniczenia tych komórek, które wiążą się z problemami występującymi przy modelowaniu relacji w długich ciągach danych.

Implementacjami tej idei było wiele typów komórek rekurencyjnych, ale dwiema najczęściej wykorzystywanymi są LSTM (*Long-short Term Memory*) [8] oraz GRU (*Gated Recurrent Unit*) [4], które zostały szczegółowo opisane w dalszej części.

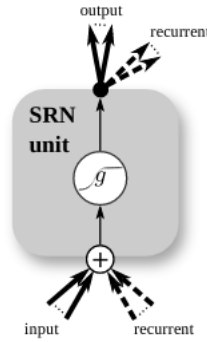
Architektury obu komórek odróżnia liczba i typ operacji, które są dokonywane na danych do niej wchodzących. Łączy je jednak mechanizm, który pozwala nauczyć komórki, które elementy ciągu są istotne z punktu widzenia pro-



Rysunek 3.2. Architektura sieci rekurencyjnych [19]

cesu uczenia, a które nie. Dzięki temu istotne informacje mogą być przechowane i wykorzystane przy obliczaniu ostatecznego wyniku sieci. Nieistotne informacje, czyli takie, które nie miały dużego wpływu na odpowiedź sieci, są pomijane i zapominane przez komórki rekurencyjne, dzięki czemu nie wprowadzają szumu informacyjnego do modelu.

Komórki rekurencyjne łączy podobny sposób działania, który polega na połączeniu wartości pamięci krótkotrwałej oraz aktualnie przetwarzanego elementu ciągu, wykonaniu na nich pewnej operacji, zwróceniu wartości wyjściowej oraz zaktualizowanej pamięci krótkotrwałej. Schemat budowy takiej komórki został przedstawiony na rys. 3.3.



Rysunek 3.3. *Simple Recurrent Network* [8]

Działanie sieci rekurencyjnej można opisać w kilku krokach. Najpierw elementy ciągu są zamieniane na wektory liczb, które stanowią punkt reprezentujący dany element w wielowymiarowej przestrzeni. Następnie wektory te są sekwencyjnie przetwarzane przez kolejne warstwy sieci neuronowej. Przy obliczaniu wartości komórek rekurencyjnych jednym ze źródeł informacji jest wartość obliczona przez poprzednią komórkę rekurencyjną. Wartość ta stanowi pamięć krótkotrwałą sieci neuronowej, w której przechowywane są informacje dotyczące poprzedniego elementu ciągu.

3.2.1 Problem eksplodującego i zanikającego gradientu

Sieci neuronowe są uczone z wykorzystaniem metody gradientowej oraz reguły łańcuchowej, co pozwala na aktualizację wag sieci w zależności od ich wpływu na poziomu błędu modelu. Stosowana metoda propagacji wstecznej (ang. *backpropagation*) pozwala rozłożyć sieć warstwa po warstwie do postaci łańcucha iloczynów pochodnych cząstkowych działań wykonywanych w ramach sieci. W przypadku sieci rekurencyjnych stosowana jest technika zwana propagacją wsteczną w czasie (ang. *backpropagation through time*), która rozwija rekurencyjne kroki sieci i przekształca ją w bardzo głęboką prostą siec neuronową i pozwala na obliczenie gradientu w każdym kroku obliczeń.

Problem eksplodującego i zanikającego gradientu [14] pojawia się dla konkretnych funkcji aktywacji w głębokich sieciach neuronowych oraz sieciach rekurencyjnych, gdzie sygnał przechodząc przez kolejne funkcje aktywacji osiąga wartości zbliżone do 1 lub do 0. Przykładowo dla funkcji aktywacji *sigmoid* pochodna dla obu krańców jest zbliżona do zera, co implikuje, że dla osiągniętych wartości aktywacji zbliżonych do 1 lub 0 pochodna również będzie zbliżona do zera. Mała wartość pochodnej powoduje, że zakumulowana pochodna będzie bliska zeru. W takich przypadkach uczenie sieci metodami gradientowymi jest nieefektywne. Z tego powodu sieci cierpiące z powodu zanikającego gradientu nie są w stanie poprawnie zaktualizować wag sieci, co powoduje ich niską skuteczność.

Rozwiązaniem problemu zanikającego gradientu jest zastosowanie funkcji aktywacji, które nie skutkują w małej pochodnej cząstkowej, takie jak ReLU (*rectified linear unit*) lub ELU (*exponential linear unit*), oraz zastosowanie połączeń szczątkowych [10]. Połączenia te są stosowane do przekazania oryginalnego sygnału pomiędzy odległymi warstwami sieci.

Analogicznie, problem eksplodującego gradientu pojawia się wtedy, gdy pochodne cząstkowe są duże, co powoduje eksponencjalny wzrost zakumulowanej pochodnej. W przypadku olbrzymiej pochodnej model nie jest w stanie efektywnie się uczyć i staje się niestabilny.

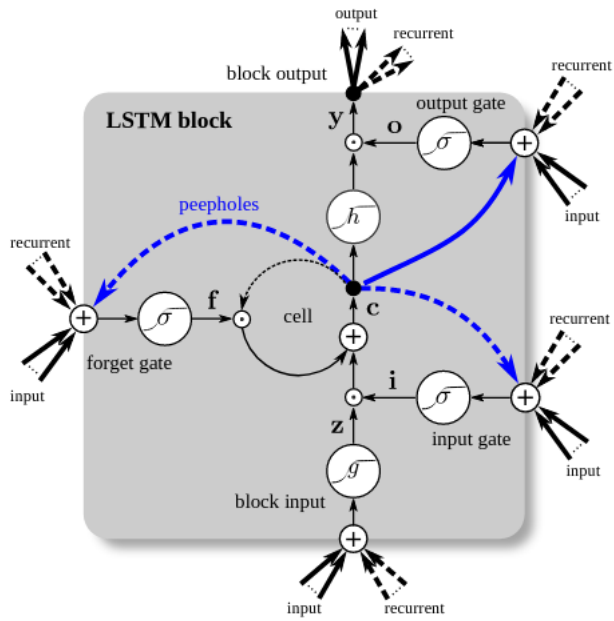
W celu zwalczenia problemu eksplodującego gradientu stosuje się specjalną inicjalizację wag [7], przycinanie lub normalizację gradientu [22]. Duże znaczenie dla procesu uczenia sieci ma rozkład z jakiego losowane są wartości do początkowej macierzy wag, dlatego należy przyjrzeć się odpowiednim funkcjom rozkładu. Normalizacja gradientu ta polega na skalowaniu pochodnych w przypadku, gdy ich wartość jest za duża. W przypadku normalizacji gradientu, jeśli norma wektora gradientu przekroczy pewien próg, wartości wektora są skalowane tak, żeby norma wektora była równa progowi. Natomiast w przypadku przycinania wartości gradientu jeśli norma wynosi 0.5, a wartość gradientu wyniosła mniej niż -0.5 to zostanie zastąpiona przez -0.5. Analogicznie jeśli wartość jest większa niż 0.5 zostanie zastąpiona przez 0.5.

Problem zanikającego i eksplodującego gradientu został rozwiązany w architekturze komórek LSTM oraz GRU. W prostych sieciach rekurencyjnych pochodna cząstkowa w bezpośredni sposób zależy od wyjścia poprzedzającej komórki. Dla ciągu komórek i sygnału aktywacji mniejszego od 1 pochodna będzie mniejsza od 1 co wraz ze wzrostem liczby komórek powoduje, że rośnie praw-

dopodobieństwo na to, że pochodna będzie maleć. W komórkach LSTM oraz GRU wartość sygnału zależy jednocześnie od sumy większej liczby parametrów, co powoduje, że aby pochodna wynosiła zero wszystkie one musiałyby być równe zero, co jest mniej prawdopodobne.

3.2.2 *Long-short term memory*

Komórka LSTM, przedstawiona na rys. 3.4, składa się z czterech bramek: *input*, *output*, *block* oraz *forget*. Rdzeniem komórki jest magistrala przechodząca przez wszystkie komórki w ramach danej warstwy, która przekazuje pamięć krótkotrwałą między komórkami i modyfikuje ją. Ta ścieżka jest reprezentowana przez połączenie bramki *forget* oraz wyjścia sieci ze wszystkimi operacjami, które na tej drodze się znajdują. Wraz z podróżą sygnału pamięci przez kolejne komórki informacje są do niej dodawane i odejmowane przez wspomniane wcześniej bramki, które decydują, jakie informacje są zachowywane, a które zapominane.



Rysunek 3.4. Komórka LSTM [8]

Bramki są wyposażone w dwa rodzaje funkcji aktywacji: *tanh* oraz *sigmoid*. Funkcja *tanh* odwzorowuje wartości na liczby z przedziału od -1 do 1, co zapewnia normalizację wyników. Z kolei funkcja *sigmoid* dokonuje transformacji liczb na wartości z przedziału od 0 do 1, dzięki czemu możliwe jest nauczenie komórek, które wartości należy zapominać przez ich wyzerowanie.

Pierwszą bramką, która zostanie opisana, jest bramka *forget*. Stanowi ona pierwszy element, jaki napotyka podróżujący po magistrali łączącej komórki sygnał. Zadaniem tej bramki jest decyzja, w jakim stopniu należy zachować sygnał z przeszłości. Zakodowana wartość poprzednich stanów oraz reprezentacja aktualnego elementu ciągu są do siebie dodawane, a na ich sumie wykonywana jest funkcja *sigmoid*, która określa, w jakim stopniu informację należy zachować w pamięci. Wynikiem tej operacji, na którym dodatkowo dokonano operacji iloczynu z poprzednim stanem komórki, jest zmienna f.

Bramka *input* służy do aktualizacji stanu komórki. Suma sygnału pamięci oraz reprezentacji aktualnego elementu ciągu są przekształcane przez funkcję *sigmoid*, a na wyniku tej operacji oraz stanie komórki dokonywane jest mnożenie, którego wynik oznaczamy i. Po raz kolejny funkcja aktywacji typu *sigmoid* decyduje o tym w jakim stopniu należy zachować sygnał przetwarzanego elementu ciągu. Przekształceniem funkcją *tanh* sumy sygnału pamięci oraz aktualnego elementu ciągu jest zmienna z.

Kolejnym krokiem jest obliczenie stanu komórki, oznaczanego c, przez przemnożenie elementów z oraz i i zsumowanie ich z iloczynem c oraz f. W wyniku tych operacji otrzymujemy zaktualizowany stan komórki. Innymi słowy, aktualizujemy stan komórki przez obliczenie jego iloczynu z zachowanym sygnałem przetwarzanego elementu ciągu oraz dodanie do niego iloczynu zachowanego sygnału pamięci i zachowanego sygnału przetwarzanego elementu. Zaktualizowany stan komórki oznaczamy symbolem $\overline{c}$.

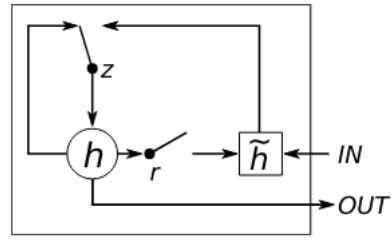
Ostatnim krokiem jest obliczenie wartości bramki *output*, która decyduje o sygnale pamięci, który zostanie przekazany do kolejnej komórki. Suma elementu ciągu oraz zaktualizowanego stanu komórki $\overline{c}$ jest przekształcana funkcją *sigmoid*, w wyniku czego otrzymujemy zmienną o. Następnie zaktualizowany stan komórki $\overline{c}$ przekształcamy funkcją *tanh* oraz mnożymy z obliczonym stanem o, w wyniku czego otrzymujemy nowy stan komórki oraz sygnał pamięci przesyłane do kolejnej komórki.

3.2.3 *Gated recurrent unit*

W tym podrozdziale przyjrzyjmy się kolejnemu typowi komórek rekurencyjnych, czyli GRU. W stosunku do LSTM budowa komórki jest prostsza i składa się tylko z dwóch bramek: *update* i *reset*.

Pierwsza z nich zachowuje się podobnie do bramek *forget* oraz *input*. Decyduje, które informacje dodać, a które zapomnieć. Bramka *reset*, oznaczana na rysunku r, decyduje ile informacji z przeszłości należy zachować, natomiast bramka *update*, oznaczana z, określa w jakim stopniu stan komórki jest aktualizowany. Symbole h oraz $\overline{h}$ oznaczają odpowiednio aktywację wychodzącą z komórki oraz aktywację przetwarzanego elementu.

Aktywacja h_t w czasie t jest liniową interpolacją między poprzednim stanem h_{t-1} oraz aktywacją elementu $\overline{h}$. Bramka z jest obliczana jako funkcja aktywacji *sigmoid* z sumy iloczynów sygnału wejściowego x oraz aktywacji h_{t-1} wraz z odpowiadającymi im macierzami wag. Sygnał bramki *reset* r jest obliczany jako *sigmoid* z sumy iloczynów sygnałów x oraz h_{t-1} wraz z odpowiadającymi



(b) Gated Recurrent Unit

Rysunek 3.5. Komórka GRU [4]

im macierzami wag. Ostatecznie sygnał aktywacji kandydata $\overline{h}_t$ obliczany jest jako funkcja *tanh* z sumy iloczynów sygnału x oraz g wraz z odpowiadającymi im wagami, gdzie g jest określony jako $r \odot h_{t-1}$.

Ze względu na mniejszą liczbę operacji wykonywanych w komórkach GRU uzyskiwane jest przyspieszenie czasu obliczeń oraz zmniejszenie zapotrzebowania na pamięć w stosunku do komórek LSTM. W kwestiach skuteczności nie ma jednoznacznie lepszej architektury.

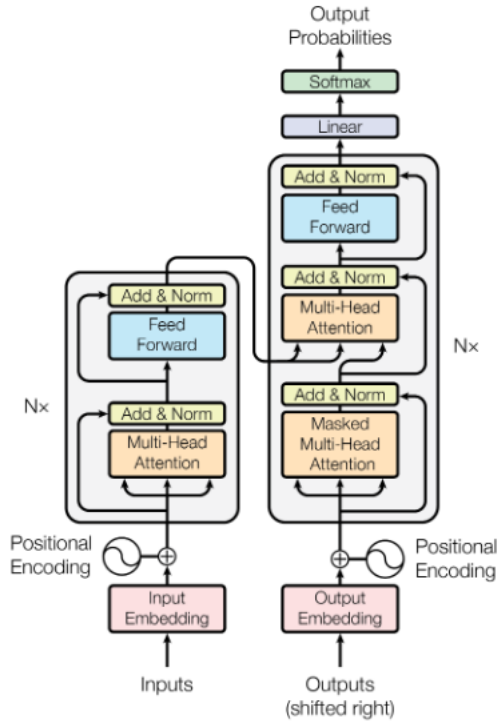
3.2.4 Transformer

Przełomem w rozwoju głębokich sieci neuronowych okazała się architektura Transformer [20]. Warstwa Transformer (rys. 3.6) jest oparta o architekturę koder-dekoder lecz, w odróżnieniu od sieci rekurencyjnych, używa prostych sieci neuronowych wraz z innowacyjnym mechanizmem *multi-head self-attention*. Dzięki niemu możliwe jest równoległe obliczenie wartości zależności pomiędzy każdym elementem ciągu, co pozwala lepiej niż w przypadku sieci rekurencyjnych modelować relacje, a także skraca czas obliczeń. Dodatkowo obliczenia te są realizowane w przestrzeni o większej liczbie wymiarów, co pozwala zakodować w niej więcej informacji.

Koder składa się ze stosu sześciu identycznych warstw. Każda z nich zawiera dwie podwarstwy: mechanizm *multi-head self-attention* oraz *element-wise fully-connected feed forward network*. W każdej z podwarstw zastosowano sieci typu Highway [18], po których następuje normalizacja warstwy. Każda z podwarstw posiada projekcję do 512-wymiarowej reprezentacji.

Dekoder również składa się ze stosu sześciu identycznych warstw. Oprócz dwóch podwarstw obecnych w koderze dodana została podwarstwa, która dokonuje operacji *masked multi-head attention* na reprezentacjach uzyskanych ze stosu koderów. W tej warstwie, podobnie jak w koderze, zastosowane zostały połączenia sieciami typu Highway wraz z normalizacją warstwy.

Model uwagi może zostać opisany jako odwzorowanie każdego z n tokenów na wektor n wartości, które reprezentują wzajemną istotność tokenów między sobą. Blok *multi-head attention*, zaprezentowany na rys. 3.7, składa się z ośmiu

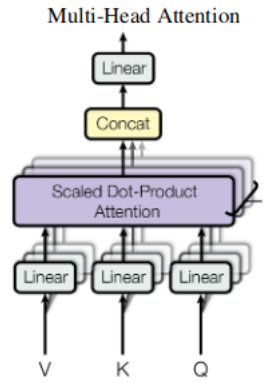


Rysunek 3.6. Architektura modelu Transformer [20]

równoległych bloków. Pierwszą warstwą jest warstwa projekcji Q, K i V do 64-wymiarowej reprezentacji, po którym występuje funkcja *scaled dot-product attention*. Uzyskane wektory są następnie sklejane oraz dokonywane jest ich mnożenie przez macierz wag.

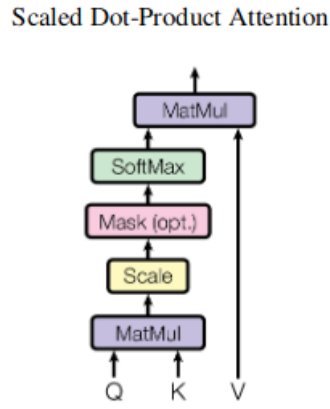
Liniowa projekcja wektorów tokenów wejściowych jest iloczynem wektora wejściowego z wektorem wag. W przypadku omawianego mechanizmu *multi-head attention* projekcja jest dokonywana między wektorem tokenu wejściowego oraz trzema wektorami wag, czego wynikiem są wektory: *Query*, *Key* oraz *Value*, oznaczane odpowiednio Q, K oraz V.

Funkcja *scaled dot-product attention*, pokazana na rys. 3.8, obliczana jest dla każdego z wektorów tokenów wejściowych. Wartość uwagi jest obliczana następująco: dla każdego tokenu obliczany jest wynik mnożenia macierzowego między wektorem Q_i oraz macierzą wektorów K poszczególnych tokenów wejściowych, łącznie z wartością uwagi dla aktualnie obliczanego tokenu. Uzyskana wartość jest następnie skalowana przez podzielenie jej przez pierwiastek drugiego stopnia z wymiaru wektora K. Następnie obliczana jest wartość funkcji *softmax* z uzyskanego wektora wartości dla wartości uwagi i-tego tokenu. Kolejnym krokiem jest obliczenie iloczynu uzyskanych wartości z wektorami V wszystkich tokenów.



Rysunek 3.7. Mechanizm *multi-head attention* [20]

Ostatecznie suma uzyskanych wektorów reprezentuje uwagę dla i-tego tokenu. W ten sposób uwaga jest obliczana dla wszystkich tokenów. Wymiar wag służących do uzyskania wektorów Q oraz K musi być jednakowy, natomiast wymiar wag wektora V może być dowolny.



Rysunek 3.8. *Scaled dot-product attention* [20]

Scaled dot-product attention jest funkcją opisaną wzorem:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_K}})V, \tag{3.1}$$

gdzie QK^T jest iloczynem skalarnym wektorów, d_K jest wymiarem wektora K.

Architektura *Transfromer* pozwala na uzyskanie wartości uwagi pomiędzy elementami w przetwarzanym ciągu, co umożliwia wyodrębnienie istotnych fragmentów, a także relacji między nimi. Dzięki tej cesze modele budowane w oparciu o architekturę *Transformer* cechuje wysoka skuteczność w modelowaniu języka, modelowaniu akustycznym, a w ostatnim czasie również w przetwarzaniu obrazów [2]. Główną wadą tej architektury jest rozmiar sieci oraz kwadratowy wzrost zapotrzebowania na pamięć wraz z wydłużeniem przetwarzanego ciągu danych. Powstało wiele modyfikacji tej architektury, które starają się minimalizować te wady, jak choćby *Longformer* [1], *Linformer* [21] oraz *Performer* [3].

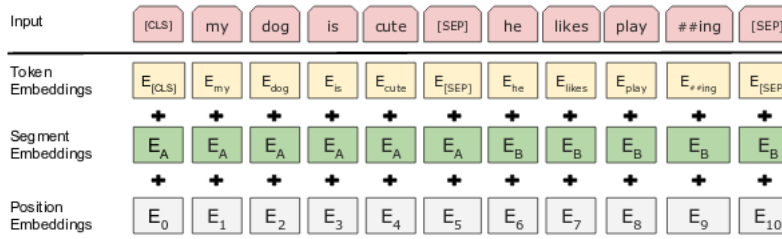
Ze względu na cechy kodera i dekodera bloki te są stosowane do rozwiązywania różnych problemów. Koder ze względu na przetwarzanie dwukierunkowego kontekstu nie może zostać zastosowany do zadań autoregresyjnych (3.3), natomiast jest dobrym kandydatem w przypadku zadań klasyfikacyjnych, w których pełni rolę ekstraktora cech. Dekoder, w odróżnieniu od kodera, przetwarza tylko lewostronny kontekst. Innymi słowy, w trakcie obliczeń ma dostęp tylko do już przetworzonego ciągu danych. Dzięki tej cesze może być z powodzeniem zastosowany w zadaniach generacyjnych. Zastosowanie architektury koder-dekoder pozwala wykorzystać mocne strony obu architektur i stworzyć model generacyjny, który jest w stanie wykonywać szersze spektrum zadań. Przykładem może być model T5 [16], który przy odpowiednim zmodyfikowaniu zadania, jest w stanie zarówno generować teksty, jak i dokonywać klasyfikacji elementów lub całych ciągów.

3.2.5 BERT

Modelem opartym na architekturze Transformer, który był przełomowy w przetwarzaniu języka naturalnego, jest BERT (*Bidirectional Encoder Representations from Transformers*) [5]. Model składa się ze stosu bloków kodera, po których występuje głowa klasyfikacyjna, najczęściej będąca warstwą projekcji liniowej oraz funkcji *softmax*. Ze względu na wykorzystanie zarówno lewego jak i prawego kontekstu nie jest przystosowany do generowania tekstu. Zastosowanie jedynie warstw kodera pozwoliło zbudować wysokiej jakości ekstraktor cech, co potwierdzają wyniki tego modelu w zadaniach klasyfikacyjnych.

Poza kodowaniem tokenów zastosowano w nim dodatkowe kodowanie segmentów, co pozwoliło odróżniać te same tokeny w różnych miejscach w ciągu danych. Rysunek 3.9 prezentuje sposób kodowania tokenów w przetwarzanym ciągu danych. W tym modelu kodowanie segmentu dotyczyło rozróżnienia dwóch zdań w ciągu wejściowym, a kodowanie pozycyjne określało pozycję tokenu w ciągu. Dla każdego tokenu w ciągu wejściowym, jego wejściowa reprezentacja jest konstruowana jako suma jego reprezentacji, segmentu oraz pozycji.

Jako strategie kodowania stosowane są następujące techniki [20]: absolutna pozycja, kodowanie względne oraz sinusoidalne.



Rysunek 3.9. Reprezentacja tokenów wejściowych w modelu BERT [5]

3.2.6 GPT

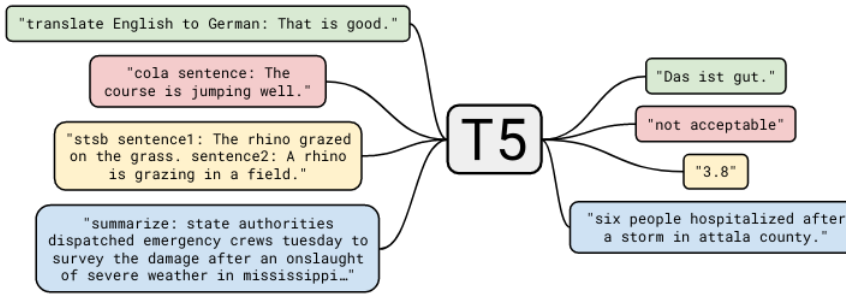
Modelem wykorzystującym stos dekoderów typu Transformer jest model GPT (*Generative Pre-Training*) [15], którego trzecia generacja zasłynęła ze zdolności do generowania tekstów jakością dorównujących tekstom napisanym przez ludzi. Zostało to osiągnięte przez wykorzystanie jedynie dekoderów korzystających z lewostronnego kontekstu oraz autoregresyjnego działania modelu opisanego szerzej w rozdziale 3.3. Model jest w stanie dokonywać dowolnego zadania tekstowego, zarówno generowania streszczeń, odpowiedzi na pytania, jak również klasyfikacji i określania podobieństwa. Ze względu na wykorzystanie lewostronnego kontekstu osiąga gorsze wyniki w zadaniach wymagających ekstrakcji cech.

3.2.7 T5

Modelem w architekturze koder-dekoder wykorzystujący bloki typu Transformer jest model T5 (*Text-to-Text Transfer Transformer*) [16]. Dzięki zastosowaniu obu typu bloków typu koder jest w stanie w wydobywać istotne cechy tekstów wejściowych, a bloki dekodera pozwalają na generowanie tekstów w sposób autoregresyjny (część 3.3). Dzięki temu jest w stanie rozwiązywać zarówno zadnia klasyfikacyjne, jak i generacyjne. Negatywnym aspektem zastosowania obu bloków jest rozmiar sieci, który przy tej samej liczbie warstw, jest dwukrotnie większy w stosunku do modelu BERT.

Innowacyjne w tym modelu było zastosowanie zunifikowanego sposobu reprezentacji danych do reprezentacji różnych zadań przy wykorzystaniu komendy sterujące. Przykładowe komendy zostały zaprezentowane na rys. 3.10.

Koncepcja zastosowania słów sterujących została rozszerzona i wykorzystana do stworzenia modeli, których proces generowania tekstów można kontrolować.

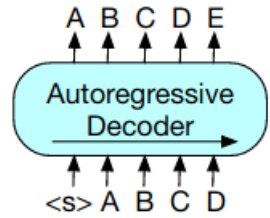


Rysunek 3.10. Komendy sterujące reprezentujące różne typy zadań tekstowych [16]

Jednym z nich jest CTRLsum [9], w którym zastosowano generowanie streszczeń warunkowane wykrytymi słowami kluczowymi.

3.3 Modele autoregresyjne

W statystyce modele autoregresyjne służą do reprezentacji zmieniającego się w czasie procesu, który zależy od swoich poprzednich wartości oraz zmiennej stochastycznej. W przypadku sieci neuronowych modelami autoregresyjnymi nazywamy takie sieci, które w każdym kolejnym kroku generują ciągi symboli [19]. Na każdym etapie model otrzymuje ciąg symboli wejściowych wraz z dotychczas wygenerowanymi symbolami i na ich podstawie generuje jeden nowy symbol wyjściowy. Proces ten trwa do momentu napotkania specjalnego symbolu określającego koniec generowanej sekwencji lub osiągnięcie progu długości generowanych ciągów symboli. Modele autoregresyjne mają zastosowanie przy generowaniu ciągów symboli, takich jak generowanie tekstu, sygnałów dźwiękowych lub sekwencji DNA.



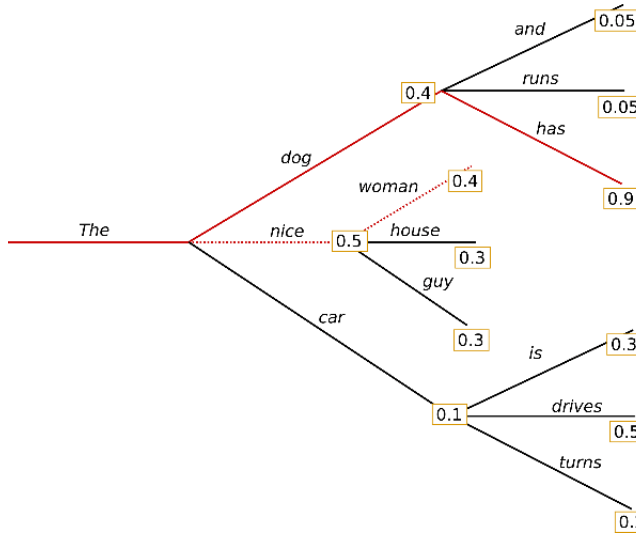
Rysunek 3.11. Sposób działania modelu autoregresyjnego [13]

3.3.1 Metody dekodowania

W celu poprawienia jakości generowanych ciągów symboli opracowany został szereg metod generowania optymalnych sekwencji. Na problem dekodowania symboli możemy patrzeć jak na wyszukiwanie najkrótszej ścieżki w drzewie, którego ścieżkami są generowane symbole, a ich wagami są prawdopodobieństwa. Problem ten należy do klasy problemów *NP*-zupełnych, ale istnieją algorytmy heurystyczne, które mają mniejszą złożoność obliczeniową.

Jedną ze stosowanych technik dekodowania używaną w modelach generacyjnych jest algorytm zachłanny (ang. *greedy search*). W ogólności polega on na tym, że w każdym kroku wybierany jest symbol, którego prawdopodobieństwo wygenerowania jest największe. Algorytm ten jest prosty i szybki w obliczeniach, jednak jego wadą jest brak różnorodności generowanych tekstów. Dodatkowo w przypadku tej metody często dochodzi do sytuacji, w której model zapętla się i generuje w kółko ten sam ciąg symboli.

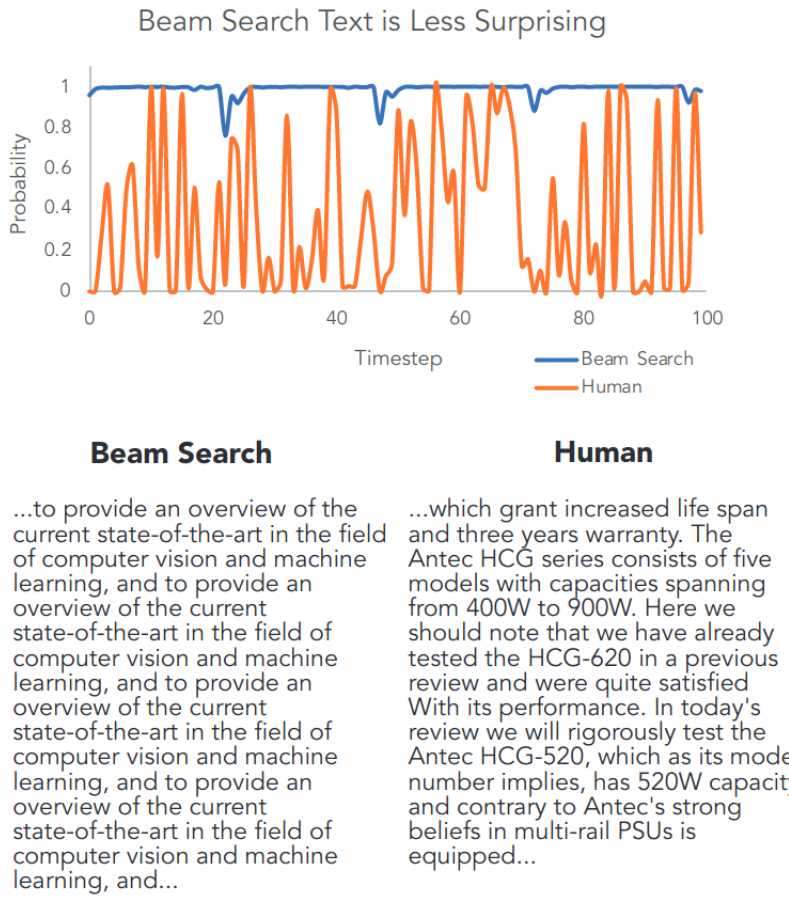
Drugą metodą, która jest najczęściej stosowana w praktyce, jest metoda przeszukiwania wiązkowego (ang. *beam search*), zaprezentowana na rys. 3.12. Jest rozszerzeniem metody algorytmu zachłannego, która w każdym kroku przeszukiwania zapamiętuje N optymalnych ścieżek. Pozwala to na zwiększenie różnorodności generowanych ciągów symboli, a także pozwala optymalizować generowanie w dłuższej perspektywie.



Rysunek 3.12. Przeszukiwanie wiązkowe

Metoda ta jest wrażliwa na długość generowanego ciągu symboli i ma problemy w przypadku generowania tekstu o nieznanej *a priori* długości, przykładowo generowanie dialogów. Występuje w niej również problem generowa-

nia powtarzalnych i zapętlonych ciągów symboli. W artykule [11] zbadano jak w stosunku do wygenerowanych automatycznie tekstów wygląda prawdopodobieństwo wygenerowania poszczególnych słów z tekstu napisanego przez człowieka, co zaprezentowano na rys. 3.13. Wniosek jaki nasuwa się na podstawie zaprezentowanego wykresu jest taki, że tekst napisany przez człowieka jest bardziej nieprzewidywalny i zaskakujący niż ten wygenerowany przez algorytm.



Rysunek 3.13. Prawdopodobieństwo wygenerowania przez model słów pochodzących z tekstu napisanego przez człowieka oraz zdekodowanych przeszukiwaniem wiązkowym wraz z przykładami [11]

Kolejną metodą dekodowania jest losowe próbkowanie generowanych symboli. Dzięki temu możliwe jest generowanie oryginalnych i wyjątkowych ciągów symboli, przykładowo słów, co lepiej oddaje rozkład prawdopodobieństwa wylo-

sowania kolejnych słów zaprezentowanego na rys. 3.13. Istotną wadą tej metody jest problem generowania niespójnych i niepoprawnych gramatycznie zdań [11].

Rozwiązaniem problemu generowania niepoprawnych ciągów symboli jest wyostrzenie rozkładu prawdopodobieństwa słów tak, aby symbole wysoce prawdopodobne były jeszcze bardziej prawdopodobne, a symbole z niską wartością prawdopodobieństwa miały tę wartość obniżoną. Przekształcenie to jest dokonywane przez obniżenie tak zwanej temperatury funkcji *softmax*.

Czwartą metodą dekodowania jest algorytm Top-K Sampling zaprezentowany w [6]. W tej metodzie K najbardziej prawdopodobnych symboli jest usuwanych, a funkcja rozkładu jest przeważana z pominięciem tych symboli. Dzięki pominięciu najbardziej prawdopodobnych słów generowany tekst jest poprawny, ale jednocześnie bardziej zaskakujący. Problemem w tej metodzie jest zdefiniowanie *a priori* liczby K określającej liczbę pomijanych słów, która istotnie wpływa na jakość generowania, a którą trudno jest wyznaczyć dla wszystkich przypadków dekodowanych ciągów symboli.

Rozszerzeniem algorytmu Top-K Sampling jest algorytm Top-p Sampling, która wyznacza liczbę pomijanych symboli na podstawie wartości dystrybuanty, która przekracza wartość progową p. Metoda ta może być stosowana w połączeniu z algorytmem Top-K Sampling, która dostarcza większej przewidywalności liczby pomijanych symboli.

Dodatkowo w stosunku do metod dekodowania stosuje się ograniczenia, które mają poprawić jakość generowanych tekstów. Jednym z nich jest blokowanie powtarzających się n-gramów, jednak ograniczenie to trzeba odpowiednio dobrać, żeby nie spowodowało to spadku jakości generowanych ciągów.

3.4 Podsumowanie

W niniejszym rozdziale opisano typy ciągów danych oraz przedstawiono architektury sieci neuronowych służących do ich modelowania. W podrozdziale 3.1.2 opisano ogólny sposób modelowania ciągów przez sieci neuronowe, co stanowiło punkt wyjścia dla zrozumienia działania opisanych dalej architektur. W rozdziale 3.2 opisano proste sieci rekurencyjne, a w podrozdziale 3.2.1 problem eksplodującego i zanikającego gradientu, który jest ich poważnym ograniczeniem. W kolejnych częściach przedstawiono kolejne generacje architektur sieci neuronowych służących do przetwarzania ciągów danych: LSTM, GRU, Transformer, BERT, GPT oraz T5. Obecnie dominującą pod względem skuteczności i powszechnej adopcji jest architektura typu Transformer, która dzięki uniwersalnemu charakterowi może być z sukcesem stosowana do szeregu zadań zarówno na różnych typach danych. W rozdziale 3.3 opisany został ogólny sposób działania modeli autoregresyjnych, a część 3.3.1 poświęcony został różnym metodom dekodowania, na przykładzie generowania tekstu, które mają znaczący wpływ na jakość generowanych ciągów symboli.

Bibliografia

1. Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer (2020).
2. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers (2021).
3. Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., Weller, A.: Rethinking attention with performers (2021).
4. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling (2014).
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2019).
6. Fan, A., Lewis, M., Dauphin, Y.: Hierarchical neural story generation (2018).
7. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS (2010).
8. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: A search space odyssey. CoRR **abs/1503.04069** (2015), `http://arxiv.org/abs/1503.04069`.
9. He, J., Kryscinski, W., McCann, B., Rajani, N.F., Xiong, C.: Ctrlsum: Towards generic controllable text summarization. CoRR **abs/2012.04281** (2020), `https://arxiv.org/abs/2012.04281`.
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015).
11. Holtzman, A., Buys, J., Du, L., Forbes, M., Choi, Y.: The curious case of neural text degeneration (2020).
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012), `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.
13. Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L.: Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension (2019).
14. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks (2013).
15. Radford, A., Narasimhan, K.: Improving language understanding by generative pre-training (2018).
16. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer (2020).
17. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection (2016).
18. Srivastava, R.K., Greff, K., Schmidhuber, J.: Highway networks (2015).

19. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks (2014).
20. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. CoRR **abs/1706.03762** (2017), http://arxiv.org/abs/1706.03762.
21. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity (2020).
22. Zhang, J., He, T., Sra, S., Jadbabaie, A.: Why gradient clipping accelerates training: A theoretical justification for adaptivity (2020).

4. Klasyfikacja tekstu przy użyciu grafowych sieci neuronowych

Robert Benke

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
robert.benke2@gmail.com

Streszczenie
Współczesnym algorytmom analizy tekstu wciąż daleko do ludzkiego poziomu jego zrozumienia. Jednym z wyzwań jest znajdowanie przez maszynę związków pomiędzy odległymi fragmentami tekstu. Próbą rozwiązania tego problemu są grafowe reprezentacje tekstu, które bardzo dobrze sprawdzają się w przedstawianiu złożonych zależności. W tekście opisane zostały dwie metody grafowej reprezentacji tekstu oraz algorytm grafowych konwolucyjnych sieci neuronowych.

Słowa kluczowe: grafowe sieci neuronowe, grafowe konwolucyjne sieci neuronowe, grafowa reprezentacja tekstu, klasyfikacja tekstu

4.1 Grafowe sieci neuronowe

Popularność sieci neuronowych sprawiła, że w ostatnich latach podejmowane są liczne próby wykorzystania ich w wielu dziedzinach. Grafy, ze względu na wysoki potencjał do reprezentowania rzeczywistych danych, stały się obiektem tych prób po raz pierwszy w 2004 roku. Podobieństwo pierwszych grafowych sieci neuronowych (ang. *graph neural network*, GNN) i klasycznych sieci ograniczało się do definicji warstwy (ang. *layer*) która w GNN była nakładana wielokrotnie, aż do osiągnięcia punktu stałego, czyli punktu dla którego wartość funkcji jest równa argumentowi. Nie było zatem możliwe optymalizowanie architektury takiej sieci, a sam proces uczenia, w praktyce ograniczony do pewnej ustalonej liczby iteracji, był czasochłonny. Mimo niskiej użyteczności tych metod, były one niezmiernie istotne w wypracowaniu parametryzowalnych funkcji z przestrzeni grafów w nią samą.

4.1.1 Algebraiczna reprezentacja grafu

Grafem G nazywamy parę (V, E), gdzie V oznacza wierzchołki, a E jest podzbiorem zbioru $V \times V$, który zawiera krawędzie pomiędzy wierzchołkami. Zbiór E może zawierać pary uporządkowane, wtedy graf nazywamy skierowanym, lub nieuporządkowane, gdy graf jest nieskierowany.

Podstawowa definicja grafu bywa często rozszerzana o dodatkowe elementy. Dla nas ważne będzie, aby każdy wierzchołek grafu mógł posiadać wektor atrybutów. Niech H będzie macierzą o wymiarze $|V| \times D$, w której każdy wiersz oznacza wektor atrybutów dla odpowiadającego mu wierzchołka. Tym samym wymaga się, aby zbiór V był uporządkowanym zbiorem wierzchołków, a graf definiujemy poprzez trójkę (V, E, H).

Do zrozumienia dalszej części pracy potrzebne będzie także pojęcie macierzy Laplasjanu. W tym celu zdefiniowana zostanie macierz sąsiedztwa i macierz stopni. Pierwsza z nich wskazuje, między którymi wierzchołkami grafu istnieją krawędzie. Druga natomiast zawiera informację o liczbie krawędzi wychodzących z każdego z wierzchołków.

Definicja 1. *Macierzą sąsiedztwa (ang. adjacency matrix) $A = (\{a_{ij}\})_{N\times N}$ nieskierowanego grafu G nazywamy symetryczną macierz, której elementy definiujemy przez*

$$a_{ij} := \begin{cases} 1, \textit{ jeśli } \{v_i, v_j\} \in E \\ 0, \textit{ w przeciwnym przypadku,} \end{cases}$$

gdzie $1 \leq i, j \leq N$, a $N = |V|$.

Definicja 2. *Macierzą stopni (ang. degree matrix) D nieskierowanego grafu G nazywamy diagonalną macierz o wymiarach $N \times N$, której elementami są*

$$d_{ii} := deg(v_i),$$

gdzie $1 \leq i \leq N$, a $deg(v) = |\{(v_1, v_2) : (v_1, v_2) \in E, v_1 = v \vee v_2 = v\}|$.

Macierz laplasjanu jest różnicą zdefiniowanych wyżej macierzy. Jest to więc macierz symetryczna, której wiersze i kolumny sumują się do zera.

Definicja 3. *Macierzą laplasjanu (ang. laplacian matrix) L nieskierowanego grafu G nazywamy symetryczną macierz zdefiniowaną poprzez relację*

$$L := D - A.$$

4.1.2 Vanilla GNN

Miano podstawowej wersji grafowych sieci neuronowych (VGNN) przypadło algorytmowi utworzonemu w 2009 roku ([9]). Vanilla GNN definiowana jest przez dwie parametryzowalne funkcje $f(\cdot, \theta)$ i $g(\cdot, \theta)$. Dla uproszczenia notacji, w dalszej części tego rozdziału, parametry θ będą pomijane. Działanie VGNN przypomina działanie rekurencyjnych sieci neuronowych. Funkcja f przyjmuje jako argument utworzony w poprzedniej iteracji stan ukryty grafu (wynik funkcji f) oraz stan początkowy atrybutów grafu. W odróżnieniu od rekurencyjnych sieci neuronowych, w których w każdej iteracji do sieci trafiają nieznane wcześniej informacje, rekurencyjna warstwa VGNN dostaje ciągle te same dane (atrybuty

oryginalnego grafu). Liczba iteracji warstwy rekurencyjnej w klasycznych sieciach zależy od długości wektora wejściowego. W przypadku VGNN taki proces nie ma naturalnie określonej liczby iteracji i kończy się, gdy reprezentacja ukryta przestaje ulegać zmianie. Funkcja g definiuje drugą warstwę sieci. Jej zadaniem jest przekształcenie reprezentacji ukrytej na wyjście sieci.

Zakłada się, że wielokrotne złożenie funkcji f nazywanej lokalną funkcją przejścia (*local transition function*) zbiega do punktu stałego. Pierwszym etapem VGNN jest odnalezienie tego punktu stałego, ponieważ jednak funkcja f nakładana jest iteracyjnie i w dowolnej kolejności na każdy wierzchołek z osobna, a jednym z jej parametrów jest bieżąca wartość atrybutów wszystkich sąsiadów tego wierzchołka, to punkt stały jest stanem nie tyle pojedynczego wierzchołka, ile całego grafu.

Dla grafu $G = (V, E, H)$ punkt stały funkcji f to taka wartość wektorów cech h_v, dla której dla każdego $v \in V$ zachodzi

$$h_v = f(h_v^{(0)}, h_{ne(v)}^{(0)}, h_{ne(v)}),$$

gdzie $h_v^{(0)}$ jest początkową wartością cech wierzchołka v podaną w macierzy H, natomiast $h_{ne(v)}$ to zbiór wektorów cech dla wszystkich sąsiadów wierzchołka v.

Poszukiwanie wektorów h_v odbywa się poprzez iteracyjne obliczanie nowych wektorów cech dla wszystkich wierzchołków:

$$\begin{aligned} h_v &= \lim_{i \to \infty} h_v^{(i+1)}, \\ h_v^{(i+1)} &= f(h_v^{(0)}, h_{ne(v)}^{(0)}, h_{ne(v)}^{(i)}). \end{aligned}$$

W praktyce proces ten jest zatrzymywany po ustalonej liczbie iteracji.

Wektory h_v nazywamy reprezentacją ukrytą (ang. *hidden representation*) wierzchołków grafu. W trakcie uczenia znajdowane są takie parametry funkcji f, które pozwalają na stworzenie ukrytej reprezentacji. Są w niej zagregowane informacje potrzebne do poprawnej predykcji wartości wyjściowej dla każdego wierzchołka w grafie. Rzutowaniem reprezentacji ukrytej do przestrzeni docelowej zajmuje się parametryzowalna lokalna funkcja wyjścia g (ang. *local output function*):

$$o_v = g(h_v, h_v^{(0)}).$$

Wartość funkcji g dla wierzchołka v, oznaczana przez o_v, może być skalarem albo wektorem. Pierwszy przypadek dotyczy jednowymiarowych problemów regresji i klasyfikacji binarnej, natomiast drugi – problemów wielowymiarowych.

Wielokrotne iterowanie po wszystkich wierzchołkach dla dużych grafów bywa kosztowne obliczeniowo, dlatego funkcje f i g, jeśli jest to możliwe, definiuje się w postaci macierzowej. Niech A będzie macierzą sąsiedztwa grafu $G = (V, E, H)$, a H_h macierzą ukrytych reprezentacji wierzchołków V. Wtedy

$$H_h = F(H_h, A, H)$$

jest macierzową postacią punktu stałego lokalnej funkcji przejścia, natomiast

$$O = G(H_h, H),$$

macierzowym odpowiednikiem lokalnej funkcji wyjścia.

Zakładając że funkcje f i g lub F i G są różniczkowalne względem parametru θ, możemy przeprowadzić proces uczenia metodami znanymi z klasycznych sieci neuronowych.

Przyczyn niewielkiej popularności VGNN należy doszukiwać się w licznych jej wadach:

- liczba iteracji potrzebnych do osiągnięcia punku stałego,
- poziom złożoności metody, a przez to jej zdolność do uczenia się, zdefiniowany jest niemal zupełnie przez funkcję f (lub F),
- obliczenia wykonywane przy pomocy funkcji f są mało efektywne, natomiast w postaci macierzowej metoda staje się trudna do zrównoleglenia.

Pomimo tych wad, zaprezentowana w tym rozdziale metoda stała się prekursorem podgrupy grafowych sieci neuronowych, które dzisiaj kolektywnie nazywamy rekurencyjnymi grafowymi sieciami neuronowymi (RGNN).

4.1.3 Przestrzenne versus spektralne grafowe sieci neuronowe

Równolegle do rekurencyjnych grafowych sieci neuronowych rozwinęły się przestrzenne i spektralne grafowe sieci neuronowe. Zarówno przestrzenne, jak i spektralne grafowe sieci neuronowe przyjęły konwencję budowania sieci neuronowej złożonej z wielu warstw. Pozwala to dopasowywać architekturę sieci do konkretnego problemu i znajdować kompromis między obciążeniem a wariancją modelu. Różnicą pomiędzy przestrzennymi i spektralnymi GNN jest przestrzeń, w której dokonuje się ekstrakcji nowych cech. W przypadku sieci spektralnych, wykorzystuje się grafową transformatę Fouriera, która pozwala zdefiniować splot cech wierzchołków z parametrami warstwy w przestrzeni spektralnej. Przykład takiej warstwy zostanie przedstawiony w kolejnym rozdziale, natomiast teraz opisana zostanie ogólną koncepcja i wynikające z niej własności. Zacznijmy od zdefiniowania grafowej transformaty Fouriera.

Definicja 4. *Niech $U^T \Lambda U$ będzie diagonalizacją macierzy laplasjanu L grafu $G = (V, E)$, dla którego $|V| = N$. Grafową transformatą Fouriera sygnału $x = (x_1, \ldots, x_N)^T$ nazywamy funkcję $F(x) := U^T x$.*

Macierz U tworzy bazę ortonormalną, dlatego $U^T U = I$. Zatem odwrotna transformata Fouriera jest zadana wzorem $F^{-1}(\hat{x}) = U\hat{x}$.

Potrzeba znalezienia wektorów i wartości własnych macierzy L jest dużą wadą metod korzystających z grafowej transformaty Fouriera. W praktyce oznacza ona wykonanie obliczeń, których złożoność obliczeniowa wynosi $O(N^3)$. Alternatywą dla diagonalizacji macierzy L okazało się aproksymowanie funkcji korzystającej z macierzy L przy pomocy wielomianów. Autorzy [3] wykorzystali w tym celu

wielomiany Czebyszewa, znacząco przyspieszając obliczenia i zachowując dotychczasową skuteczność.

Zauważenie drugiego istotnego problemu tych metod wymaga lepszego zrozumienia wektorów własnych macierzy L oraz zdefiniowania grafowego splotu. Definicja grafowego splotu oparta jest o własność klasycznej transformaty Fouriera która mówi, że transformata Fouriera splotu dwóch funkcji jest iloczynem transformat Fouriera tych funkcji.

Definicja 5. *Grafowy splot sygnału x z wektorem g przyjmuje następującą formę:*

$$x *_G g := F^{-1}\left(F(x *_G g)\right) = F^{-1}\left(F(x) \odot F(g)\right) = U\left(U^T x \odot U^T g\right),$$

gdzie $\odot$ jest iloczynem Hadamarda.

W rozszerzonej definicji grafu $G = (V, E, H)$ występują dwa rodzaje informacji. Pierwszy dotyczy samej struktury połączeń, ich liczby oraz sposobu, w jaki łączą one wierzchołki grafu. Na podstawie struktury połączeń grafu bada się rozkład stopni wierzchołków, średnicę grafu, odległości pomiędzy wierzchołkami i wiele innych charakterystyk, które w całości zawarte są w topologii grafu. Drugi rodzaj informacji o grafie dotyczy samych wierzchołków. Informacja ta zawarta jest w macierzy H i nadaje im pewną interpretację, albo przynajmniej pozwala je od siebie odróżniać na podstawie wektora atrybutów wierzchołka. Przyglądając się wektorom własnym odpowiadającym największym wartościom własnym macierzy L zauważymy, że każdy z tych wektorów identyfikuje skupiska wierzchołków występujące w grafie. Jeżeli zatem użyjemy macierzy H jako sygnału w grafowej transformacie Fouriera, to wynikiem (przynajmniej dla indeksów odpowiadającym najwyższym wartościom własnym) będzie agregacja atrybutów wierzchołków znajdujących się w konkretnych skupiskach. Przez skupiska rozumiemy tutaj grupy wierzchołków połączonych gęściej niż średnia w grafie. Mając w pamięci definicję grafowego splotu, widzimy, że wektor parametrów $U^T g$ będzie wzmacniał lub osłabiał informację płynącą z takich skupisk. Chociaż sama idea wydaje się bardzo ciekawa, to jednak rodzi pewne problemy. Iloczyn macierzy cech wierzchołków z parametrami w przestrzeni spektralnej sprawia, że wyuczone parametry działają dobrze jedynie w przypadku grafów o podobnej strukturze. Znacząca zmiana topologii grafu od tej, na której algorytm był uczony, sprawia, że również wektory własne macierzy L zyskują nową interpretację i algorytm będzie działał znacznie gorzej niż na zbiorze uczącym.

Pozbawione obu tych wad są przestrzenne grafowe sieci neuronowe. Warstwy tych sieci tworzą nowe ukryte reprezentacje wierzchołków, korzystając jedynie z atrybutów wierzchołków znajdujących się nie dalej niż zadana odległość K od rozpatrywanego wierzchołka. W ten sposób nie musimy już dokonywać diagonalizacji macierzy L, a zamiast tego potrzebujemy jedynie K pierwszych potęg macierzy A. Ponadto, przestrzenne grafowe sieci neuronowe znacznie lepiej uogólniają się na grafy o zupełnie różnych strukturach. Każda warstwa sieci przestrzennych uczy się jedynie, w jaki sposób zagregować informacje z sąsiednich wierzchołków, a więc nie zależy od struktury połączeń w skali makro.

Przestrzenne grafowe sieci neuronowe wydają się w tym zestawieniu algorytmem o większym potencjale jednak to, która z metod będzie działać najlepiej,

zależy przede wszystkim od problemu, jaki ma rozwiązać. Warto jednocześnie zwrócić uwagę na algorytm, po publikacji którego grafowe sieci neuronowe zyskały znacząco na popularności. Grafowe konwolucyjne sieci neuronowe (ang. *Graph Convolutional Neural Network*, GCN) z 2017 roku ([4]) osiągały wyniki znacząco lepsze od dotychczas znanych algorytmów na wielu różnych zbiorach danych. Jest to metoda, której teoretyczne wyprowadzenie ma korzenie w spektralnych grafowych sieciach neuronowych, jednak ostateczna forma przypomina bardziej metodę z rodziny przestrzennych grafowych sieci neuronowych. W kolejnej części metoda ta zostanie dokładniej opisana. Od czasu publikacji GCN powstało wiele nowych grafowych sieci neuronowych. Przegląd ciekawszych spośród nich można znaleźć między innymi w [7].

4.2 Grafowe konwolucyjne sieci neuronowe

Grafowe konwolucyjne sieci neuronowe okazały się przełomowe pod wieloma względami. Jest to metoda z grupy spektralnych grafowych sieci neuronowych, która jednak nie wymaga diagonalizacji macierzy laplasjanu grafu, dzięki czemu uczenie jej oraz wykorzystywanie jest o rzędy wielkości szybsze niż innych spektralnych algorytmów. Ponadto, dzięki przeniesieniu idei konwolucji znanej z konwolucyjnych sieci neuronowych wykorzystywanych głównie do problemów widzenia komputerowego, filtry w grafowych konwolucyjnych sieciach neuronowych są zlokalizowane przestrzennie. Działają lokalnie, korzystając jedynie z atrybutów najbliższych sąsiadów. Korzyścią z takiego działania, podobnie jak w przypadku przestrzennych grafowych sieci neuronowych, jest zdolność sieci do utrzymania błędu generalizacji, liczonego dla wszystkich elementów przestrzeni po rozkładzie, na podobnym poziomie do błędu uzyskanego na zbiorze uczącym.

Uogólnienie operacji konwolucji znanej z konwolucyjnych sieci neuronowych na grafy nie było oczywiste. Głównym powodem trudności była nieeuklidesowa natura grafów. W przeciwieństwie do zdjęć nie posiadają one siatkopodobnej ustrukturyzowanej postaci. Obraz dość łatwo można zanurzyć w przestrzeni R^3 lub R^5, poprzez przydzielenie każdemu pikselowi współrzędnych (x, y, g) dla obrazów monochromatycznych, oraz (x, y, r, g, b) w przypadku polichromatycznych, gdzie (x, y) oznaczają pozycje piksela, natomiast pozostałe wymiary – intensywność kolorów. Grafy są nieeuklidesowym typem danych, dlatego nie da się ich przedstawić w przestrzeni R^n zachowując jednocześnie całą informację w nich zawartą. Wynika to z definicji odległości pomiędzy wierzchołkami grafu, która jest równa długości najkrótszej drogi pomiędzy nimi oraz nieskończoność, gdy taka droga nie istnieje. Patrząc na ten sam problem z innej perspektywy, każdy piksel znajdujący się odpowiednio daleko od brzegu obrazu, posiada tę samą liczbę sąsiadów w zadanej odległości. Dodatkowo, dla każdego z rozważanych pikseli istnieje dokładnie jeden piksel sąsiadujący z nim w każdym z 8 podstawowych kierunków. Fakt ten wykorzystywany był w widzeniu komputerowym jeszcze przed erą konwolucyjnych sieci neuronowych. Pozwalał on ręcznie tworzyć macierze wielkości 3×3 (lub większe), zwane filtrami, które przyłożone do każdego piksela potrafiły wykrywać krawędzie, wyostrzać obraz czy wyszukiwać obiekty

na obrazie. W grafie każdy wierzchołek może mieć dowolną (ograniczoną przez liczbę wszystkich wierzchołków) liczbę sąsiadów w zadanej od niego odległości. Pomimo tej przeszkody, autorom grafowych konwolucyjnych sieci neuronowych udało się zdefiniować warstwę przypominającą klasyczną warstwę konwolucyjną, a lepsze poznanie konwolucyjnych sieci neuronowych pozwala dostrzec płynące z tego korzyści.

4.2.1 Konwolucyjne sieci neuronowe

Historia konwolucyjnych sieci neuronowych sięga lat 80. XX wieku. Pierwsze eksperymenty miały na celu zastąpienie ręcznie tworzonych filtrów, filtrami dedykowanymi konkretnemu problemowi, poprzez optymalizacje wartości parametrów w celu maksymalizacji użyteczności. Jedną z pierwszych konwolucyjnch sieci neuronowych była LeNet ([6]). Sieć ta została stworzona w celu odczytywania ręcznie pisanych cyfr w kodach pocztowych. Na wejściu LeNet przyjmowała monochromatyczne zdjęcie ręcznie pisanej cyfry o wielkości 28×28 pikseli, dla którego na wyjściu zwracała predykcję jednej z dziesięciu klas odpowiadającym cyfrom 0-9. Architektury późniejszych konwolucyjnych sieci neuronowych znacząco różniły się od LeNet, jednak wszystkie budowane były w oparciu o te same koncepcje warstwy konwolucyjnej.

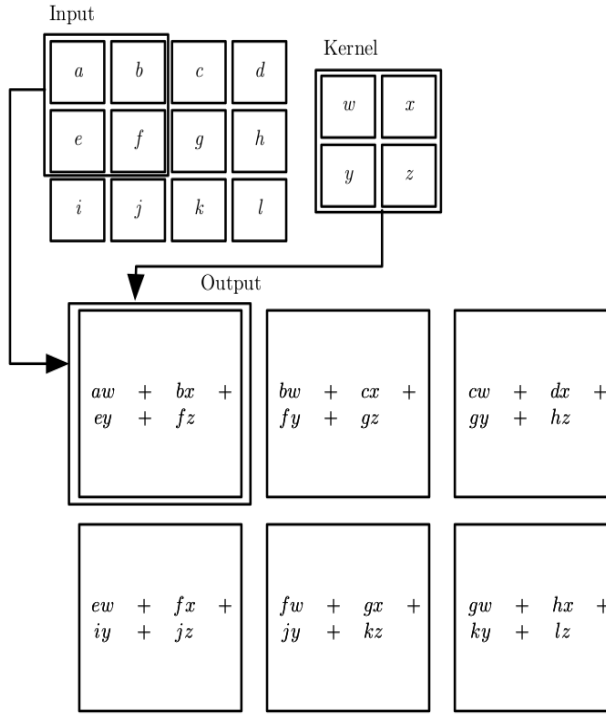
Warstwa konwolucyjna. Operacja konwolucji zilustrowana została na monochromatycznym obrazie o rozmiarach 3×4 piksele (rys. 4.1), których wartości oznaczone są literami od a do l oraz filtrze o rozmiarze 2×2, z elementami oznaczonymi literami: w, x, y i z. Operacja konwolucji tworzy nowy obraz, którego wartości pikseli mają wartość sumy elementów iloczynu Hadamarda filtru z podzbiorem pikselami obrazu początkowego.

Filtr z rysunku 4.1 może zostać wykorzystany do wykrywania krawędzi. Niech $w = x = 1$ oraz $y = z = -1$, a obrazem wejściowym niech będzie flaga Wielkiej Brytanii (rys. 4.2). Suma wartości tego filtra wynosi 0, dlatego działanie takiego filtra widoczne będzie jedynie na obszarach, w których następuje zmiana koloru. Nie każda zmiana będzie jednak widoczna. Patrząc na sam filtr, można zauważyć, że będzie on przyjmował wartości niezerowe tylko wtedy, gdy zmiana koloru będzie występowała w płaszczyźnie pionowej (rys. 4.2 – na dole po lewej stronie).

Znalezienie wszystkich krawędzi poziomych występujących na zdjęciu jest możliwe przy wykorzystaniu transpozycji wyżej zdefiniowanego filtra. Efekty konwolucji z filtrem poziomym zostały przedstawione w prawej dolnej części rysunku 4.2.

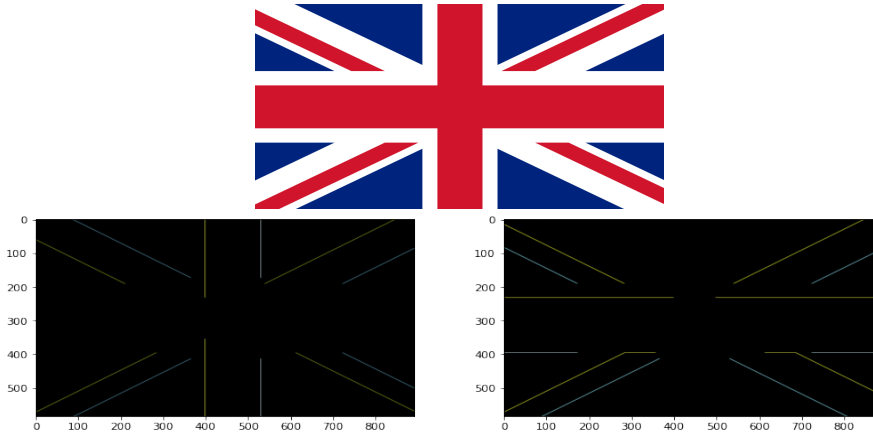
Analogicznie można zmodyfikować filtr w celu wyszukiwania krawędzi ukośnych. Możliwości filtru o rozmiarze 2×2 szybko się jednak kończą, dlatego nie są one używane w praktyce. LeNet korzystało z filtrów o rozmiarze 5×5, a zaprezentowana w 2012 roku AlexNet [5] posiadała w swojej architekturze filtr o rozmiarze 11×11. Duże filtry potrafią uczyć się złożonych obiektów, jakimi są na przykład występujące na obrazie okręgi. Obecnie większość warstw w konwolucyjnych sieciach neuronowych posiada filtry o rozmiarze 3×3, a możliwość

rozpoznawania złożonych kształtów jest zapewniona poprzez złożenie wielu takich warstw.



Rysunek 4.1. Operacja konwolucji z filtrem 2×2, źródło: [2]

Głębokie konwolucyjne sieci neuronowe. Wielkość filtra definiuje jego pole odbiorcze (ang. *receptive field*), czyli powierzchnię, z jakiej gromadzona jest informacja w jednym pikselu po operacji konwolucji. W przypadku jednej warstwy konwolucyjnej pole to jest równe wielkości filtra, zatem filtr o wielkości 3×3 będzie posiadał pole odbiorcze tej samej wielkości. Jeżeli do sieci zostanie dodana kolejna warstwa konwolucyjna z filtrem tej samej wielkości, to pole odbiorcze takiej sieci wzrośnie do obszaru 5×5 pikseli. Dzieje się tak dlatego, że druga operacja konwolucji agreguje informacje z sąsiednich pikseli, w których zawarta jest już informacja o ich sąsiadach uzyskana w trakcie pierwszej konwolucji. Zatem dwie warstwy z filtrami 3×3 posiadają takie samo pole odbiorcze, co jedna warstwa z filtrem 5×5. Jednakże pojedynczy filtr o rozmiarze 3×3 posiada 9 parametrów, a więc dwa takie filtry będą posiadały znacznie mniej parame-



Rysunek 4.2. U góry – flaga Wielkiej Brytanii; na dole po lewej – flaga Wielkiej Brytanii po operacji konwolucji z filtrem wykrywającym krawędzie pionowe; na dole po prawej – flaga Wielkiej Brytanii po operacji konwolucji z filtrem wykrywającym krawędzie poziome

trów niż jeden o rozmiarze 5×5. Ponadto, podobnie jak w klasycznych sieciach neuronowych, każda warstwa zwiększa nieliniowość modelu.

Liczba warstw konwolucyjnych jest fundamentalna dla stworzenia dużej powierzchni gromadzenia informacji. Można na nią dodatkowo wpłynąć poprzez aplikowanie konwolucji jedynie, co pewien krok (ang. *stride*). Na przykładzie z rysunku 4.1 filtr przesuwany był za każdym razem o jeden piksel w pionie albo poziomie. W ten sposób każdy piksel brał udział w obliczeniach wielokrotnie. Piksel znajdujący się na pozycji $(2, 2)$ (licząc od $(1, 1)$ w lewym górnym rogu), którego wartość f pojawiła się w obliczeniach czterech pikseli. Gdyby jednak filtr przesuwany był o dwa piksele zamiast o jeden, to wartość f byłaby wykorzystana tylko raz. Lepiej jest to widoczne na przykładzie większego filtra, na przykład o rozmiarze 11×11. Przesuwając taki filtr za każdym razem tylko o jeden piksel w pionie albo poziomie, wartości niektórych pikseli wykorzystywana jest nawet 121 razy. Co więcej, złożenie dwóch takich warstw dałoby powierzchnię gromadzenia informacji wynoszącą 12×12 pikseli, czyli niewiele większą niż pojedyncza warstwa. Dlatego wartość przesunięcia dobiera się tym większą, im większy jest rozmiar filtra.

Jednym z zadań widzenia komputerowego jest segmentacja obrazu. W tym celu dokonuje się klasyfikacji dla każdego piksela (ang. *pixel-wise classification*). Wynikiem segmentacji jest obraz o tej samej rozdzielczości, który każdemu pikselowi przyporządkowuje wartość odpowiadającą segmentowi, do którego należy. Przykład segmentacji obrazu znajduje się na rysunku 4.3. Próbując napisać własną konwolucyjną sieć neuronową do segmentacji, napotkamy jeszcze jeden problem. Dokładnie przyglądając się wymiarom obrazu wejściowego i wyjściowego z rysunku 4.1 można spostrzec, że obraz wyjściowy jest mniejszy od wejściowego.

Dzieje się tak, ponieważ w przypadku ostatnich pikseli z prawej strony oraz od dołu nie jest możliwe policzenie konwolucji obrazu z filtrem (obszar nakładania filtru wystawałby poza granice obrazu). Rozwiązaniem jest dodanie do obrazu (ang. *padding*) potrzebnych kolumn i wierszy. Najprostszą strategią uzupełnienia wartości nowych pikseli jest przypisanie im wartości zero.

Rysunek 4.3. Po lewej – zdjęcie człowieka na tle gór; po prawej – wynik segmentacji

Innym popularnym zadaniem konwolucyjnych sieci neuronowych jest klasyfikacja obrazów. Przypuśćmy, że w zbiorze obrazów znajdują się obrazy psów i kotów, a zadanie polega na automatycznym rozdzieleniem tych dwóch grup. W tym celu budowana jest sieć neuronowa, która w pierwszej kolejności wykorzystuje warstwy konwolucyjne do ekstrakcji cech, żeby następnie przekazać je do warstwy gęsto połączonych neuronów (ang. *dense layer*), na których budowany jest klasyfikator. Problemem okazuje się tutaj przejście z ostatniej warstwy konwolucyjnej do znanej z klasycznych sieci neuronowych warstwy gęsto połączonych ze sobą neuronów. Na wejściu taka sieć dostaje zdjęcie w rozdzielczości 1000×1000, które przechodząc przez kolejne warstwy konwolucyjne traci na rozdzielczości niewiele albo w ogóle (jeżeli używamy *paddingu*). Jeżeli nasza pierwsza warstwa gęsta będzie posiadała 1000 neuronów, a każdy z nich musi być połączony z każdym z pikseli, to taka warstwa posiadałaby ponad miliard parametrów. Sposobem zapobiegania tak dużej liczbie parametrów jest używanie warstwy *poolingu* po warstwach konwolucyjnych. *Pooling* jest operacją, która zmniejsza rozdzielczość obrazu. Działa ona podobnie do warstwy konwolucyjnej, przesuwając się po obrazie, z tą różnicą, że redukuje grupę pikseli do jednego. Zazwyczaj działa na oknie 2×2 pikseli i wybiera średnią lub maksymalną wartość spośród nich. W ten sposób *pooling* każdorazowo redukuje rozdzielczość obrazu o połowę, więc pięciokrotne jego użycie sprawi, że nasza warstwa gęsta będzie posiadała już tylko $31 * 31 * 1000 + 1000$ parametrów, czyli ponad tysiąc razy mniej niż uprzednio (obliczenia te nie biorą pod uwagę liczby kanałów, która w obu przypadkach byłaby taka sama).

4.2.2 Grafowa warstwa konwolucyjna

Konwolucyjne sieci neuronowe posiadały filtry o określonych rozmiarach, w których każdy parametr odpowiadał jednemu z sąsiadujących pikseli. W przypadku grafów liczba sąsiadów może się różnić pomiędzy wierzchołkami, dlatego zamiast ważonej sumy sąsiadów liczona jest zwykła suma atrybutów wszystkich sąsiadów. Odpowiada to przemnożeniu macierzy sąsiedztwa A z macierzą atrybutów H dla nieskierowanego i nieposiadającego pętli grafu $G = (V, E, H)$. Po takiej operacji, w każdym z wierzchołków zawarta jest informacje o atrybutach wszystkich jego bezpośrednich sąsiadach. Równocześnie zapominana przez wierzchołek jest informacja o jego własnych atrybutach. Utrata informacji o posiadanych przez wierzchołek atrybutach jest niepożądana, dlatego autorzy GCN zaproponowali zamianę macierzy A na $\hat{A} = A+I$, gdzie I jest macierzą identycznościową. Kolejnym etapem było wprowadzenie parametrów, które pozwoliłyby sieci się uczyć. Ten etap możemy postrzegać jako jednowarstwową sieć neuronową, która znajduje ukrytą reprezentację zagregowanych w poprzednim kroku atrybutów dla każdego z wierzchołków. Dochodzi tutaj do współdzieleniem parametrów, ponieważ dokładnie te same parametry wykorzystywane są przez wszystkie wierzchołki w grafie. W postaci macierzowej oznacza to iloczyn macierzy z kroku pierwszego z macierzą parametrów W. Macierz W jest macierzą, której liczba wierszy odpowiada liczbie wierzchołków w grafie, natomiast liczba kolumn jest dowolna i definiuje długość wyjściowego wektora atrybutów. Na każdy element macierzy wynikowej nakładana jest nieliniowa funkcja aktywacji σ. Całość w postaci macierzowej przybiera następującą formę:

$$H_h = \sigma\left(\hat{A}HW\right), \tag{4.1}$$

gdzie H_h jest nową macierzą atrybutów.

Znaczny zysk stabilności metody otrzymuje się poprzez wprowadzenie symetrycznej normalizacji macierzy $\hat{A}$:

$$H_h = \sigma\left((\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}})HW\right), \tag{4.2}$$

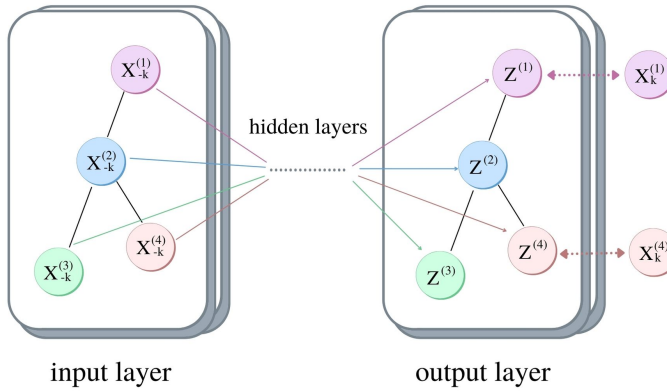
gdzie $\hat{D}$ jest macierzą diagonalną, której elementy na przekątnej są równe $\hat{D}_{ii} = \sum_{j=1}^{N}\hat{A}_{ij}$.

Grafowa warstwa konwolucyjna pozostawia strukturę grafu bez żadnych zmian, nadając jedynie wierzchołkom nowe atrybuty. Analogicznie do sieci konwolucyjnych, każda kolejna warstwa zwiększa odległość, z jakiej informacje gromadzone są w każdym z wierzchołków.

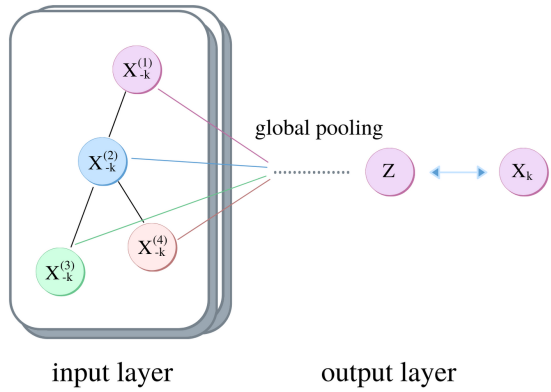
4.2.3 Klasyfikacja grafu versus klasyfikacja wierzchołków

Oryginalnie grafowe konwolucyjne sieci neuronowe zostały przedstawione jako metoda klasyfikacji wierzchołków (ang. *node-wise classification*, rys. 4.4). Możliwe jest również zastosowanie tej metody do klasyfikacji grafu, ale podobnie jak w przypadku sieci konwolucyjnych konieczne jest zdefiniowanie warstwy *poolingu*.

W przypadku grafów, stosuje się *pooling* globalny, którego działanie polega na wybraniu maksimum lub średniej ze wszystkich wierzchołków dla każdego z elementów wektora atrybutów i jest aplikowany tylko raz, po ostatniej grafowej warstwie konwolucyjnej (rys. 4.5).



Rysunek 4.4. Schemat klasyfikacji wierzchołków przy użyciu GCN, źródło: [4]



Rysunek 4.5. Schemat klasyfikacji grafu przy pomocy GCN i globalnego poolingu

4.3 Zastosowania grafowych konwolucyjnych sieci neuronowych w analizie tekstu

Największy sukces grafowe metody odniosły w rozwiązywaniu problemów, które miały od samego początku nieeuklidesową naturę. Oczywistym przykładem będą tutaj dane oryginalnie przechowywane w postaci grafów. Liczba takich danych dynamicznie rośnie, częściowo za sprawą dedykowanym grafom bazą danych. Systemy rekomendacji, sieci społecznościowe, mechanika płynów, modelowanie cząsteczek chemicznych czy interakcji pomiędzy białkami to tylko niektóre z licznych przykładów danych grafowych. Innym rodzajem danych, który dużo może zyskać na reprezentacji grafowej, są dane tekstowe. Skomplikowane zależności wynikające ze struktury i semantyki tekstu sprawiają, że analiza tekstu jest wciąż dużym wyzwaniem dla maszyn. Nawet najnowsze rozwiązania w tej dziedzinie, oparte o głębokie uczenie maszynowe posiadają bardzo silne założenia ograniczające ich zastosowania. W dalszej części przedstawione zostaną dwie metody grafowej reprezentacji tekstu. Obydwie zostały przygotowane w celu przeprowadzenia klasyfikacji dokumentów tekstowych.

4.3.1 Metody reprezentacji tekstu w postaci grafu

Reprezentacja dokumentu tekstowego w postaci grafu jest wciąż otwartym problemem. Budowa grafu determinuje jego zdolność do przechowywania informacji i ma przez to fundamentalne znaczenie dla jakości budowanego modelu.

Reprezentacja korpusu dokumentów. W roku 2018 ([10]) zaproponowana została grafowa reprezentacja korpusu dokumentów w postaci jednego heterogenicznego (posiadającego dwa rodzaje wierzchołków) grafu, przy pomocy którego możliwa jest klasyfikacja dokumentów. Zaprezentowane w poprzednim rozdziale grafowe konwolucyjne sieci neuronowe nie zostały przygotowane z myślą o grafach heterogenicznych. Autorzy tej grafowej reprezentacji zdecydowali się informacje o różnicach pomiędzy wierzchołkami reprezentującymi dokumenty i wierzchołkami reprezentującymi słowa zakodować w postaci wag połączeń, jakie mogły pomiędzy nimi występować. Macierz sąsiedztwa jest macierzą blokową o czterech podmacierzach:

$$\left[\begin{array}{c|c} \begin{array}{c} Token- \\ Token \end{array} & \begin{array}{c} Token- \\ Dokument \end{array} \\ \hline \begin{array}{c} Dokument- \\ Token \end{array} & \begin{array}{c} Dokument- \\ Dokument \end{array} \end{array}\right]. \tag{4.3}$$

Pierwsza podmacierz, znajdująca się w lewym górnym rogu macierzy (4.3), zawiera wagi połączeń pomiędzy słowami. Liczba kolumn oraz wierszy tej macierzy jest równa liczbie unikalnych słów występujących w całym korpusie. Do ustalenia siły (wagi) połączenia pomiędzy parą wyrazów użyta została funkcja

wzajemnej informacji punktowej (ang. *pointwise mutual information*, PMI) zdefiniowana w następujący sposób:

$$\begin{aligned} PMI(s_1; s_2) &:= \log \frac{p(s_1, s_2)}{p(s_1)p(s_2)} \\ &= \log \frac{p(s_1|s_2)}{p(s_1)} \\ &= \log \frac{p(s_2|s_1)}{p(s_2)}, \end{aligned}$$

gdzie $p(s)$ jest liczbą wystąpień słowa s we wszystkich dokumentach podzieloną przez liczbę wszystkich słów, natomiast $p(s_1, s_2)$ jest ilorazem liczby współwystąpień słów s_1 i s_2 oraz liczby wszystkich bigramów (par występujących po sobie słów). W celu zapewnienia symetryczności macierzy sąsiedztwa założono, że $p(s_1, s_2) = p(s_2, s_1)$. Inną zaproponowaną modyfikacją było użycie nieujemnej funkcji $PMI_+ := \max(PMI, 0)$ zamiast PMI.

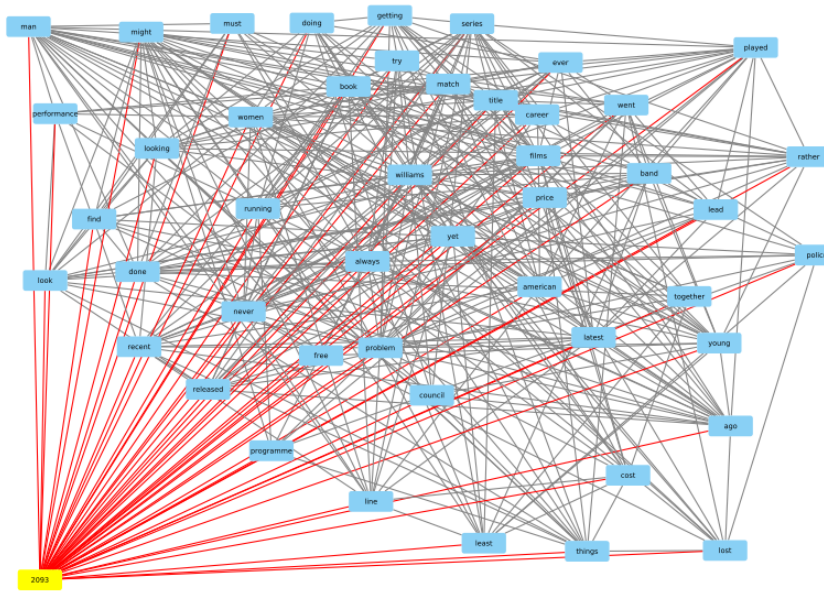
Podmacierz oznaczona w wyrażeniu 4.3 słowami „Token-Dokument" jest macierzą wag połączeń pomiędzy dokumentami a słowami. Jej wymiary odpowiadają liczbie unikalnych słów oraz liczbie dokumentów w korpusie. Siła połączeń dokumentu ze słowem, które się w nim znajduje (bo tylko z takimi się łączy) rośnie wraz z liczbą wystąpień tego słowa w rozpatrywanym dokumencie i spada, jeżeli słowo występuje w innych dokumentach. Do wyliczenia konkretnych wartości użyty został znormalizowany algorytm $TF - IDF$ (ang. *term frequency–inverse document frequency*).

Trzecia podmacierz, znajdująca się w lewym dolnym rogu macierzy (4.3), jest transpozycją podmacierzy „Token-Dokument".

Ostatnia z podmacierzy jest zarezerwowana na wagi połączeń pomiędzy dokumentami. Autorzy opisywanej metody uznali jednak, że cała informacja o dokumencie zawarta jest w jego tekście. Zatem wszystkie drogi prowadzące od jednego wierzchołka reprezentującego dokument do drugiego powinny prowadzić przez słowa, z którego oba dokumenty są zbudowane. Z tych powodów podmacierz w prawym dolnym rogu macierzy (4.3) pozostaje wypełniona zerami.

Konstrukcja macierzy sąsiedztwa (4.3) sprawia, że otrzymany graf jest duży i gęsty. Liczba jego wierzchołków dla dużego korpusu może z łatwością osiągać setki tysięcy. Wycinek grafu przygotowanego dla korpusu artykułów napisanych dla BBC ([1]) został przedstawiony na rysunku 4.6. Widoczny jest na nim jeden dokument o identyfikatorze '2093' (oznaczony na żółto) oraz niewielki podzbiór słów (wierzchołki niebieskie), przy użyciu których został napisany. Liczba artykułów w tym zbiorze danych wynosiła ponad dwa tysiące. W zbiorze danych tej wielkości istnieje duże prawdopodobieństwo, że dwa dowolne słowa znajdą się w którymś z artykułów w niewielkiej od siebie odległości. Wzajemna informacja punktowa dla takich słów będzie niezerowe, co odpowiada narysowanemu (szarym kolorem) połączeniu na rysunku 4.6. Każde słowo posiada również niezerową (oznaczoną kolorem czerwonym) krawędź łączącą je z wierzchołkiem dokumentu. Wybrany fragment zawiera tylko jeden wierzchołek reprezentujący dokument, ale

należy się spodziewać, że przedstawione wierzchołki słów posiadają połączenia do wielu innych wierzchołków dokumentów.



Rysunek 4.6. Fragment heterogenicznego grafu zawierający wierzchołek dokumentu o numerze 2093 i wierzchołki części słów zawartych w tym dokumencie

Dotychczas powiedziane zostało, czym są wierzchołki w grafowej reprezentacji tekstu oraz jak definiowane są połączenia pomiędzy nimi. Trzecim elementem definicji grafu jest macierz atrybutów. Często wykorzystywane są dwa podejścia do budowania macierzy H. Pierwsze z nich zakłada, że sieć sama nauczy się semantyki słów. Przy takim założeniu macierz atrybutów może być macierzą identycznościową (ang. *one-hot encoding*). Drugie podejście wykorzystuje wcześniej wyuczone wektorowe reprezentacje słów (ang. *embedding*) i z nich buduje macierz atrybutów. Oryginalnie w pracy przedstawiającej opisywaną metodę wykorzystane zostało podejście z macierzą identycznościową.

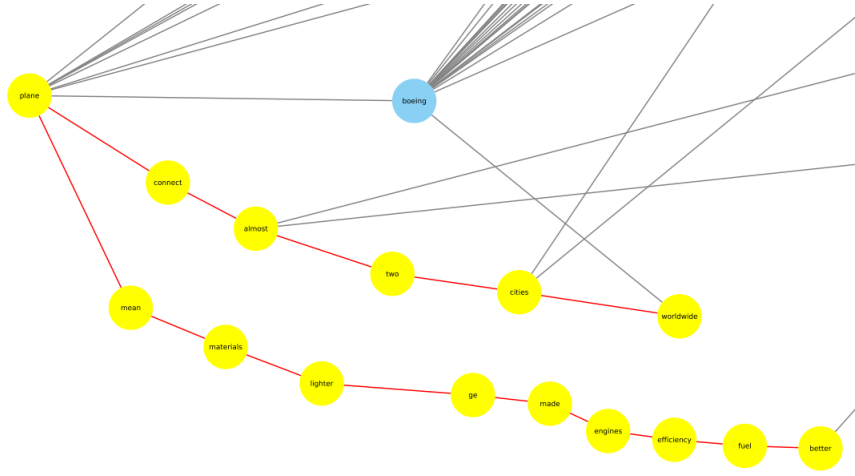
Reprezentacja pojedynczego dokumentu. Wielkość grafu przedstawionego w rozdziale wyżej rośnie liniowo wraz z wielkością korpusu, co znacząco utrudnia uczenie na dużych zbiorach danych. Podejście opisane w tym rozdziale będzie polegało na przygotowaniu oddzielnego grafu dla każdego z dokumentów w korpusie. Graf nie posiada wierzchołków reprezentujących dokumenty, a więc wynikowy graf będzie homogeniczny (posiadający wszystkie wierzchołki tego samego typu). Wierzchołki grafu w dalszym ciągu reprezentować będą zbiór unikalnych

słów, z tą różnicą, że będą to tylko słowa występujące w rozpatrywanym dokumencie, a nie, tak jak poprzednio, w całym korpusie. Algorytm budowania grafu dla dokumentu d wygląda następująco:

1. Zainicjalizuj macierz A o rozmiarze $|V| \times |V|$ zerami oraz zmienną $i = 1$.
2. Niech V będzie wektorem unikalnych słów w dowolnej, ustalonej kolejności.
3. Znajdź pozycje p_1 i p_2 słów d_i oraz d_{i+1} w wektorze unikalnych słów V i przypisz 1 elementowi macierzy A na pozycji (p_1, p_2).
4. Jeśli d_{i+1} było ostatnim słowem w dokumencie d, to zwróć A, w przeciwnym wypadku inkrementuj i i wróć do kroku 3.

Innymi słowy, dwa wierzchołki grafu są ze sobą połączone wtedy i tylko wtedy, gdy przynajmniej raz występują w tekście obok siebie.

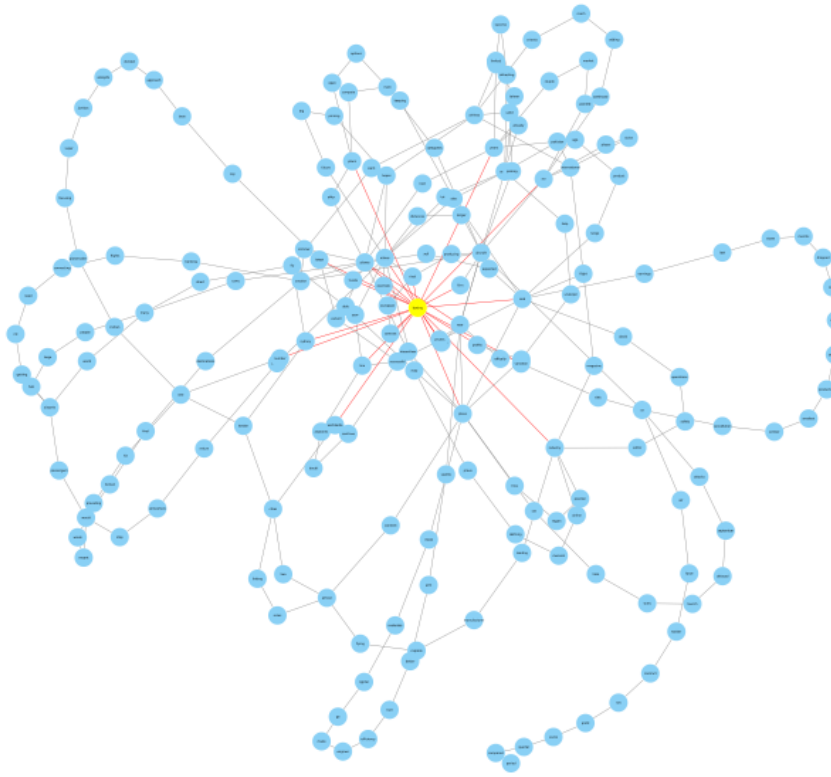
Do pokazania grafowej reprezentacji tekstu wykorzystywany zostanie artykuł opisujący przedpremierowe testy Boeing'a 777 oraz zawarte w nim zdanie "*Better fuel efficiency from engines made by GE and lighter materials mean that the plane can connect almost any two cities worldwide*". Fragment grafu zawierający słowa występujące w tym zdaniu przedstawiono na rysunku 4.7. Niewielka liczba krawędzi pomiędzy słowami pozwala niemal bez wysiłku odczytać oryginalne zdanie. Taka sytuacja ma miejsce w przypadku zdań, których słowa wystąpiły w całym dokumencie niewiele razy (w idealnym przypadku tylko raz).



Rysunek 4.7. Fragment grafu zawierający wyrazy zawarte tylko w jednym zdaniu

Rysunek 4.8 daje dobrą intuicję na temat sposobu, w jaki GCN będzie zbierało informacje z grafu. Wierzchołki mające największą liczbę sąsiadów – znajdujące się w centralnej części grafu (rys. 4.9) – reprezentują słowa kluczowe, przy użyciu których budowany jest główny wątek. Historie poboczne są zbudowane

ze słów rzadziej występujących w całym dokumencie, dlatego reprezentujące je wierzchołki biorą mniejszy udział w przesyłaniu informacji po grafie.

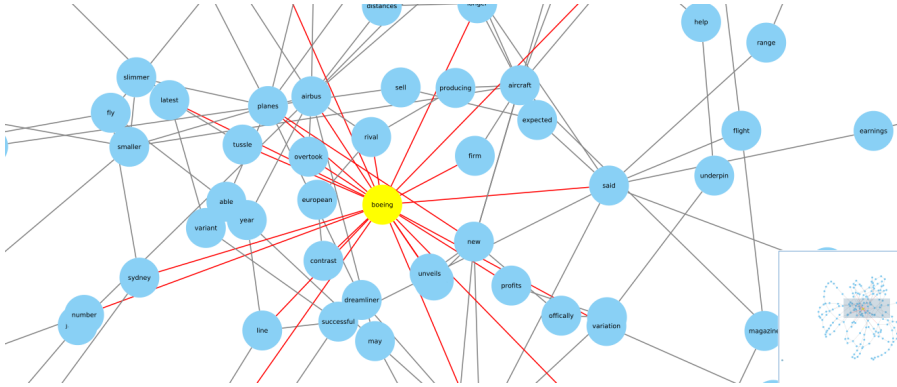


Rysunek 4.8. Grafowa reprezentacja całego dokumentu.

Przy budowaniu macierzy atrybutów wykorzystywane są wektorowe reprezentacje słów dostarczone przez wcześniej wyuczoną w tym celu sieć neuronową word2vec [8]. Korzystanie z wektorowych reprezentacji słów wyuczonych na znacznie większych korpusach może znacząco poprawić działanie grafowych konwolucyjnych sieci neuronowych i, co może ważniejsze, utrzymuje stałą liczbę kolumn (długość wektora atrybutów) macierzy atrybutów we wszystkich grafach.

4.3.2 Porównanie architektur grafowych konwolucyjnych sieci neuronowych

Mając dane w postaci grafowej oraz wiedzę na temat grafowych sieci neuronowych, można przejść do ostatniego kroku, jakim jest dobranie architektury sieci. Zaprezentowane zostaną dwie architektury dedykowane grafowym reprezentacjom tekstu opisanym w poprzednim rozdziale i wykorzystujące grafowe



Rysunek 4.9. Wierzchołki grafu z największą liczbą sąsiadów.

warstwy konwolucyjne. Nic nie stoi jednak na przeszkodzie, aby wykorzystać przedstawione grafowe reprezentacje tekstu do uczenia innych grafowych sieci neuronowych lub dowolnie zmodyfikowanych wersji przedstawionych poniżej architektur.

TextGCN jest siecią neuronową opisaną wspólnie z zaprezentowaną wcześniej grafową reprezentacją korpusu dokumentów w postaci jednego grafu [10]. Graf na potrzeby TextGCN budowany jest z połączonego zbioru treningowego, testowego i walidacyjnego, natomiast proces uczenia i predykcji odbywa się przy pomocy maskowania niebiorących w nim udziału wierzchołków. Na wejściu sieć otrzymuje trzy obiekty: macierz sąsiedztwa, macierz atrybutów oraz maskę. Wielkość macierzy sąsiedztwa oraz macierzy atrybutów jest kwadratowo zależna od sumy liczby wszystkich dokumentów w korpusie i liczby unikalnych słów występujących w korpusie. Nie stanowi to dużego problemu w przypadku diagonalnej macierzy atrybutów którą możemy efektywnie zapisać w postaci macierzy rzadkiej. Inaczej ma się sytuacja dla macierz sąsiedztwa. Z definicji 4.3 wynika, że tylko jej część definiująca połączenia pomiędzy reprezentantami dokumentów mogłaby zyskać na zapisie w postaci macierzy rzadkiej. Korzyść pamięciowa z wykorzystania macierzy rzadkich zależy w tym wypadku od stosunku liczby unikalnych słów do wielkości korpusu.

Gęstość grafu oraz jego średnica są ważnymi charakterystykami pomagającymi dobrać odpowiednią liczbę warstw ukrytych. Każda grafowa konwolucyjna warstwa w sieci neuronowej zwiększa pole odbiorcze o jeden. Mając na przykład graf, którego średnica wynosi dwa, wystarczą dwie ukryte warstwy, aby sieć mogła w każdym wierzchołku zagregować potrzebne informacje z całego grafu. Konstrukcja grafowej reprezentacji korpusu dokumentów pozwala przypuszczać, że średnica takiego grafu będzie w większości przypadków wynosiła od 3 do 5. Schemat architektury TextGCN przypomina zatem ten przedstawiony na rysunku 4.4, gdzie po pewnej określonej liczbie warstw konwolucyjnych następuje klasyfikacja każdego z wierzchołków.

Zalety TextGCN

- w przeciwieństwie do metody typu worek słów (ang. *bag-of-words*) potrafi kodować częściową informację o strukturze tekstu.

Wady TextGCN

- wymaga grafu zawierającego zbiór treningowy, testowy i walidacyjny – niezależnie od tego, czy sieć jest już wyuczona czy nie,
- ma problemy z nauczeniem się semantyki słów,
- predykcja wymaga stosunkowo dużo czasu,
- nie nadaje się do użycia na innych danych niż te, na których sieć była uczona,
- olbrzymi wpływ na wyniki ma przeprowadzony preprocessing tekstu.

DocumentGCN różni się od TextGCN pod kilkoma względami. Przede wszystkim grafy tworzone dla dokumentów mają zazwyczaj większą średnicę. Spowodowane jest to brakiem wierzchołków dla dokumentów oraz niższymi stopniami wierzchołków słów. Używanie głębszych sieci ma zatem sens i przyczynia się do wprowadzenia bardziej złożonych nieliniowości w modelu. DocumentGCN ma w procesie uczenia do dyspozycji wiele niewielkich grafów. Pozwala w ten sposób na lepsze zarządzanie pamięcią poprzez dobranie liczby grafów biorących udział w jednej iteracji (ang. *batch size*).

Przeprowadzenie klasyfikacji całego grafu wymaga, na pewnym etapie obliczeń, sprowadzenia grafu do przestrzeni euklidesowej. Znane są dwa sposoby wykonania tej operacji. Pierwszy z nich polega na dodaniu do grafu specjalnego wierzchołka połączonego ze wszystkimi wierzchołkami oryginalnego grafu. Po kilku grafowych warstwach konwolucyjnych zapominane są atrybuty wszystkich wierzchołków za wyjątkiem atrybutów wierzchołka wcześniej przez nas dodanego. W ten sposób otrzymujemy wektorową reprezentację grafu. Druga z możliwości wykorzystuje globalny pooling wyliczający średnią z atrybutów wszystkich wierzchołków. Wynikiem tej operacji jest 64-wymiarowy wektor będący zanurzeniem rozpatrywanego grafu w R^{64} (schemat na rys. 4.5). Ostatnie warstwy tworzą wielowarstwowy perceptron estymujący prawdopodobieństwa przynależności grafu do jednej z pięciu klas.

Zalety DocumentGCN

- użycie słów osadzonych ułatwia sieci nauczenia się semantyki słów umieszczonych w pewnym kontekście,
- pozwala elastycznie zarządzać potrzebną do obliczeń pamięcią.

Wady DocumentGCN

- czas potrzebny na wykonanie predykcji jest dłuższy niż w przypadku rekurencyjnych sieci neuronowych.

4.3.3 Podsumowanie

Jak pokazano, grafowe sieci neuronowe zajęły ważne miejsce wśród algorytmów uczenia głębokiego, w wielu dziedzinach przewyższając klasyczne sieci neuronowe. W problemie klasyfikacji tekstu o szczególnej pozycji rozwiązań grafowych decyduje efektywność, jaką osiągają one przy analizie bardzo długich dokumentów. Pozwala im na to specyfika metod grafowej reprezentacji tekstu, opisanych w rozdziale 3.1, w których to liczba wierzchołków grafu uzależniona jest od wielkości zbioru jedynie unikalnych, a nie wszystkich, słów obecnych w tekście. Inną przyczyną ich atrakcyjności jest zdolność skutecznego modelowania zależności między zdaniami znajdującymi się w znacznej odległości od siebie. Ta cecha jest szczególnie widoczna w DocumentGCN, w którym grafowa reprezentacja tekstu zachowuje częściowo oryginalną sekwencyjną naturę dokumentu. Grafy użyte w modelu TextGCN są natomiast bardziej skoncentrowane na częstotliwości wystąpień poszczególnych słów.

Bibliografia

1. Bbc news classification competition. `https://www.kaggle.com/c/learn-ai-bbc/data`, accessed: 2021-16-06.
2. Astuti, D.l., Samsuryadi, S., Rini, D.P.: Real-time classification of facial expressions using a princial component analysis and convolutional neural network. SINERGI (2019).
3. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural network on graphs with fast localized spectral filtering. Advances in Neural Information Processing Systems 29 (2016).
4. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional network. ICLR 2017 (2017).
5. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. Neural Information Processing Systems (2012).
6. Lecun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Back-propagation applied to handwritten zip code recognition. Neural Computation (1989).
7. Liu, Z., Zhou, J.: Introduction to Graph Neural Networks. Morgan & Claypool (2020).
8. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In Proceedings of Workshop at ICLR (2013).
9. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks (2009).
10. Yao, L., Mao, C., Luo, Y.: Graph convolutional networks for text classification. AAAI Conference on Artificial Intelligence (2018).

5. Rola i techniki eksploracji w uczeniu przez wzmacnianie

Piotr Januszewski

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
piotr.januszewski@pg.edu.pl

Streszczenie
W rozdziale podjęto rozważania na temat roli eksploracji w uczeniu się agentów sztucznej inteligencji przez wzmacnianie. Prezentuje przegląd współczesnych technik eksploracji i rozróżnia dwie główne rodziny technik: eksplorację nieukierunkowaną i eksplorację ukierunkowaną. Praca ta powinna pomóc zrozumieć dylemat pomiędzy eksploatacją wiedzy a eksploracją środowiska, któremu poddany jest agent w każdym kroku interakcji ze środowiskiem. Opisane tutaj techniki mogą być inspiracją do dalszych prac nad rozwiązaniem tego dylematu i tym samym ulepszenia metod uczenia się przez wzmacnianie.

Słowa kluczowe: sztuczna inteligencja, uczenie przez wzmacnianie, eksploracja

5.1 Wprowadzenie

Gdy inteligentny agent uczy się kontrolować nieznane środowisko, musi on pogodzić dwa przeciwstawne cele. Z jednej strony musi *eksplorować* środowisko w celu zidentyfikowania optymalnego sposobu zachowania (ang. *policy*). Dla przykładu, robot zbierający puszki musi przemierzać nieznane środowiskiem, aby zdobyć wiedzę na temat miejsc, gdzie zazwyczaj znajdują się puszki. Z drugiej jednak strony, agent musi *eksploatować* doświadczenie zdobyte podczas nauki. Chociaż robot musi badać swoje otoczenie, powinien unikać kolizji z przeszkodami po otrzymaniu negatywnej nagrody za kolizje. Ponadto, wiedza na temat miejsc, w których wcześniej znalezione zostały puszki powinna być również brana pod uwagę przez robota przy wyborze działań. W celu efektywnego uczenia się, działania powinny być dokonywane w taki sposób, że środowisko jest eksplorowane przy jednoczesnym eksploatowaniu wiedzy agenta – działania powinny maksymalizować efekty uczenia się przy minimalizacji kosztów nauki. Ten podstawowy kompromis nazywany jest dylematem eksploracji i eksploatacji (ang. *exploration and exploitation dilemma* [18]). Ten rozdział bada rozwiązania tego kompromisu w procesie uczenia się agenta w domenach ze skończoną przestrzenią akcji.

5.2 O eksploracji

Jakie są podstawowe pytania dotyczące roli eksploracji w uczeniu przez wzmocnianie? Zacznijmy od pytań charakteryzujących eksplorację i eksploatację. Jeśli eksploracja ma na celu skrócenie czasu nauki, to centralne pytanie dotyczące efektywnej eksploracji brzmi: "Jak czas uczenia się może zostać zminimalizowany?". W związku z tym pytanie o eksploatację brzmi: "Jak zminimalizować koszty nauki?". Te pytania są zwykle przeciwstawne, tj. im krótszy czas nauki, tym większe koszty i odwrotnie. Jak zobaczymy, czysta eksploracja również nie minimalizuje czasu nauki. Dzieje się tak, ponieważ czysta eksploracja maksymalizuje zdobywanie wiedzy, a tym samym może marnować dużo czasu na odkrywanie części środowiska nieistotnych dla zadania. Jeśli ktoś jest zainteresowany ograniczeniem eksploracji do odpowiednich części środowiska, często ma sens jednoczesne eksploatowanie zdobytej wiedzy. Dlatego eksploatacja jest częścią efektywnej eksploracji. Z drugiej strony, eksploracja jest również częścią efektywnej eksploatacji, ponieważ kosztów nie da się zminimalizować bez poznania środowiska.

Drugie ważne pytanie, które należy zadać, to: "Jaki wpływ ma dana zasada eksploracji na szybkość i koszt uczenia się?". Czyli, ile czasu powinien poświęcić projektant, który projektuje system uczenia się, na zaprojektowanie odpowiedniej reguły eksploracji? To pytanie będzie szeroko omawiane, ponieważ wpływ techniki eksploracji zarówno na czas nauki, jak i koszty nauki może być ogromny. W zależności od struktury środowiska, nieefektywna eksploracja może skutkować nawet eksponencjalnym wzrostem czasu potrzebnego na naukę w stosunku do wielkości problemu [13].

Trzecie kluczowe pytanie brzmi: „Jak odnaleźć kompromis pomiędzy eksploracją i eksploatacją?”. Ponieważ eksploracja i eksploatacja ustanawiają kompromis, to pytanie wymaga dalszej specyfikacja ograniczeń rozwiązania. Na przykład można zapytać: "Jak mogę znaleźć najlepszy kontroler w danym czasie?" lub "Jak mogę znaleźć najlepszy kontroler, nie przekraczając określonej kwoty kosztów?”. Oba pytania ograniczają dylemat w taki sposób, że optymalną kombinację między eksploracją a eksploatacją można znaleźć, biorąc pod uwagę, że problem można w ogóle rozwiązać przy tych ograniczeniach.
Załóżmy teraz, że ktoś posiadamy już sprawną technikę eksploracja i wydajną technikę eksploatacji. Rodzi to pytanie "Jak połączyć eksplorację i eksploatację?”. Czy każde działanie powinno jednocześnie eksplorować i eksploatować środowisko, czy też agent czasami bardziej powinien skupić się na eksploracji, a czasami bardziej na eksploatacji?

Na wszystkie te pytania nie można odpowiedzieć jedną, uniwersalną odpowiedzią. Kompromis pomiędzy eksploracją a eksploatacją, optymalna strategia eksploracji, a także właściwa technika łączenia eksploracji i eksploatacji w dużym stopniu zależą od konkretnego środowiska i zadania, którego wykonywania agent ma się nauczyć. Zależą również od konkretnej techniki uczenia się. Niniejszy rozdział dotyczy zagadnień eksploracji, opisuje i ocenia techniki eksploracji w kontekście uczenia się przez wzmacnianie za pomocą metody Q-Learning.

Omówione zostaną różne techniki eksploracji i porównane na podstawie wspólnej taksonomii.

Reszta rozdziału jest zorganizowana w następujący sposób. W podrozdziale 5.3 przedstawimy pojęcia wykorzystywane w tym rozdziale. W podrozdziale 5.4 opisujemy taksonomię technik eksploracji i analizujemy wybrane techniki eksploracji często spotykane w najnowszej literaturze na temat uczenie się przez wzmacnianie. Podrozdział 5.5 kończy ten rozdział dyskusją.

5.3 Uczenie przez wzmacnianie

Typowo problem który agent uczy się rozwiązywać modeluje się jako proces decyzyjny Markowa (ang. *Markov Decision Process* – MDP). MDP jest zdefiniowany przez krotkę $(\mathcal{S}, \mathcal{A}, R, P, \gamma, p_0)$, gdzie $\mathcal{S}$ jest ciągłą wielowymiarową przestrzenią stanów, $\mathcal{A}$ oznacza ciągłą wielowymiarową przestrzeń akcji, P jest modelem przejść, $\gamma \in [0, 1]$ oznacza współczynnik dyskontowy, p_0 odnosi się do dystrybucji stanu początkowego, a R jest funkcją nagrody. Agent uczy się z sekwencji przejść $\tau = [(s_t, a_t, r_t, s_{t+1}, d)]_{t=0}^{T}$, zwanych epizodami lub trajektoriami, gdzie $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim P(\cdot|s_t, a_t)$, $r_t = R(s_t, a_t, s_{t+1})$, d jest sygnałem terminalnym, a T oznacza moment czasu kroku terminalnego. Stochastyczna polityka $\pi(a|s)$ odwzorowuje każdy stan na rozkład akcji. Deterministyczna polityka $\pi : \mathcal{S} \to \mathcal{A}$ przypisuje każdemu stanowi jedną akcję. Polityka π^* stanowi optymalne rozwiązanie problemu decyzyjnego Markowa.

Jednym ze sposobów rozwiązania tak opisanego problemu jest Q-Learning. Agent uczy się funkcji Q, która ocenia jakość każdej akcji w każdym stanie. Funkcja Q może być modelowana klasycznie tablicą [18] lub np. siecią neuronową [11]. Jakość akcji jest definiowana przez oczekiwaną sumę przyszłych dyskontowanych nagród jeśli wybierzemy daną akcję a w stanie s, a później agent będzie postępował optymalnie do końca epizodu:

$$Q(s, a) = \mathop{\mathbb{E}}_{\pi^*, P}\left[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a\right] \tag{5.1}$$

Politykę która maksymalnie eksploatuje zdobytą wiedzę funkcji Q otrzymuje się wybierając zawsze akcję o najwyższej jakości $\pi(s) = \operatorname{argmax}_a Q(s, a)$. Zbieranie danych do treningu funkcji Q dokonywane jest inną polityką eksplorującą środowisko, patrz podrozdział 5.4. Optymalne rozwiązanie π^* otrzymuje się wybierając akcję z pomocą wytrenowanej funkcji Q^*, oznaczonej gwiazdką ze względu na jej optymalność, $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$. Szczegóły trenowania funkcji Q zostały opisane w rozdziale 6.

5.4 Techniki eksploracji

W tym rozdziale skupimy się na politykach eksploracyjnych i rozróżnimy dwie główne rodziny technik eksploracji, eksplorację nieukierunkowaną i eksplorację ukierunkowaną. Nieukierunkowane techniki eksploracji badają środowisko

w oparciu o losowość. Najbardziej prymitywną techniką eksploracji nieukierunkowanej jest eksploracja losowa, w której działania są wybierane losowo za pomocą jednostajnego rozkładu prawdopodobieństwa, niezależnie od oczekiwanych kosztów. Na przykład robot korzystający z losowej eksploracji będzie raz po raz zderzał się z przeszkodami, niezależnie od wcześniejszych doświadczeń. Inne opisane tutaj techniki nieukierunkowanej eksploracji uwzględniają koszty poprzez modyfikację rozkładu prawdopodobieństwa, z jakim wybierane są działania. Stopniowo unika się kar, zmniejszając prawdopodobieństwo niewłaściwych działań. Eksploracja ukierunkowana różni się od eksploracji nieukierunkowanej tym, że wykorzystuje pewną specyficzną wiedzę do kierowania eksploracją. Zamiast wybierać działania losowo, wybiera się działania w taki sposób, aby zoptymalizować oczekiwany zysk wiedzy. Podstawową wiedzą którą można wykorzystać do kierowania eksploracją jest informacja jak dobrze poznane są konkretne części środowiska. Techniki eksploracji ukierunkowanej pozwalają zatem na bezpośredni wybór działań, które maksymalizują efekt eksploracji np. kierują agenta w najgorzej poznane regiony środowiska. Chociaż wyższość eksploracji ukierunkowanej została uznana przez kilku autorów [1, 5, 12] techniki nieskierowane są często spotykane w najnowszej literaturze [7, 10, 21].

5.4.1 Eksploracja nieukierunkowana

Techniki eksploracji nieukierunkowanej charakteryzują się wykorzystaniem losowości do eksploracji. Najbardziej podstawowa i pozbawiona informacji technika nieukierunkowanej eksploracji nazywa się *błądzeniem losowym*. Akcje są wybierane losowo z jednostajnego rozkładu prawdopodobieństwa. Błądzenie losowe w ogóle nie uwzględnia kosztów ani nagród. Ta technika wciąż jest wykorzystywana w najnowszej literaturze pod postacią polityki ϵ-greedy [9, 11]. Polityka ta z prawdopodobieństwem ϵ eksploruje wybierając jednostajnie losową akcję, a z prawdopodobieństwem $1 - \epsilon$ eksploatuje swoją wiedzę wybierając akcję najlepszą według oceny aktualnej polityki. Zatem parametr ϵ kontroluje w jakim stopniu agent eksploruje środowisko, a w jakim stopniu eksploatuje swoją wiedzę. Można interpretować go jako współczynnik mieszania eksploracji z eksploatacją.

Istnieją techniki eksploracji nieukierunkowanej, które uwzględniają koszty podczas nauki. Zwykle wykorzystują one wyuczoną dotychczas politykę do oceny oczekiwanych kosztów lub nagród wybieranych działań. Akcje są wybierane losowo, ale oczekiwane nagrody są wykorzystywane do modyfikowania prawdopodobieństwa z jakimi akcje są wybierane: im lepsza akcja, tym większe prawdopodobieństwo, że zostanie wybrana i vice versa, gorsze akcje są częściej unikane. Jedną z technik takiej eksploracji jest strategia oparta na rozkładzie Boltzmanna nazywana polityką *softmax*.

Wiele algorytmów uczenia przez wzmacnianie oparte jest na metodzie iteracyjnego ulepszania polityki (ang. policy iteration) [18]. Q-Learning jest przykładem takiej metody. Q-Learning uczy się funkcji $Q(s, a)$, która pozwala na oszacowanie jakości – oczekiwanej sumy przyszłych nagród – każdego działania z osobna w konkretnym stanie. Na jej podstawie wybierana jest najlepsza akcja przez politykę zachłanną, kiedy zależy nam na eksploatowaniu wiedzy. Zamiast czarnej

skrzynki, która zwraca nam akcję do podjęcia, dostajemy narzędzie do oceny użyteczności działań. Polityka *softmax* wykorzystuje te szacunki i za pomocą dystrybucji Boltzmanna waży użyteczność eksploracyjną każdej akcji, biorąc pod uwagę oceny jakości wszystkich innych akcji:

$$\pi(a_i|s) = \frac{e^{Q(s,a_i)/T}}{\sum_j e^{Q(s,a_j)/T}} \tag{5.2}$$

Tutaj $\pi(a_i|s)$ oznacza prawdopodobieństwo wybrania akcji a_i w stanie s, a T jest dodatnią wartością stałą, często nazywaną temperaturą, która kontroluje w jakim stopniu wiedza agenta jest eksploatowana. Jeśli T jest duże, akcje są bardziej losowe, niezależnie od oceny funkcji Q. Jeśli T zbiega do zero, wtedy prawdopodobieństwo najlepszej akcji staje się coraz większe i zbiega do 1 – polityka staje się coraz bardziej zachłanna i bardziej eksploatuje wiedzę agenta. Kiedy zastanowimy się jakim działaniom polityka *softmax* przydziela wysokie prawdopodobieństwa, dojdziemy do wniosku iż im wyższa jakość akcji, $Q(s, a)$, w porównaniu do wszystkich pozostałych akcji, tym większe prawdopodobieństwo że zostanie ona wybrana.

Innym przykładem techniki eksploracji nieukierunkowanej, która uwzględnia koszty podczas nauki, jest *stochastyczna polityka* w algorytmach Actor-Critic np. Vanilla Policy Gradient [19]. Polityka ta wybiera akcje losowo (jest stochastyczna), ale jest trenowana za pomocą gradientów [22] w taki sposób, aby lepsze akcje pod względem oczekiwanych nagród były bardziej prawdopodobne, a te które powodują wysokie koszty były mniej prawdopodobne.

Podsumowując, nieskierowane techniki eksploracji wybierają działania stochastycznie, podczas gdy działania o lepszej jakości są wybierane z równym lub wyższym prawdopodobieństwem niż działania o gorszej jakości. Techniki eksploracji nieukierunkowanej obejmują szerokie spektrum stochastycznych mechanizmów selekcji działań, które monotonicznie odwzorowują użyteczność eksploatacyjną akcji na prawdopodobieństwa wyboru działań. Poszukiwania w oparciu o niejednostajne rozkłady prawdopodobieństwa, takie jak rozkłady Boltzmanna, nadal opierają się na losowości, ale częściowo unika się kosztownych akcji na rzecz tych które agent wie że przynoszą wysokie nagrody. Techniki te są często spotykane w domenach z dyskretną i skończoną przestrzenią akcji. Znacznie trudniej jest zastosować te metody w dziedzinach, w których akcje przyjmują wartości rzeczywiste. W takim wypadku prym wiodą metody oparte na deterministycznych politykach do których dodaje się losowy szum eksploracyjny [7, 10, 21], chociaż są pewne wyjątki [8].

Eksploracja nieukierunkowana może być nieefektywna pod względem czasu nauki, co oznacza, że oczekiwany czas nauki może skalować się wykładniczo z rozmiarem przestrzeni stanów [13]. Jednakże, techniki te są proste w implementacji i często dają zadowalające rezultaty. Dlatego też są często stosowane. Praktyczna zasada mówi, że do rozwiązania problemu stosujemy najprostszy algorytm eksploracji, który daje zadowalające rezultaty.

5.4.2 Eksploracja ukierunkowana

Techniki eksploracji ukierunkowanej wykorzystują pewną wiedzę specyficzną dla eksploracji do prowadzenia poszukiwań w środowisku. Zamiast losowego wybierania akcji, schemat eksploracji bezpośrednio określa, którą akcję należy podjąć w następnej kolejności, aby jak najlepiej poznać środowisko. Ostatecznym celem ukierunkowanej eksploracji jest wybranie działań, które maksymalizują poprawę agenta w czasie. Jest to jednak niemożliwe do ustalenia, ponieważ nie można z góry wiedzieć, w jaki sposób akcja poprawi wydajność agenta w nieznanym lub częściowo nieznanym środowisku. Z tego powodu techniki ukierunkowanej eksploracji opisane w niniejszym podrozdziale mają charakter heurystyczny – wykorzystują heurystyki do optymalizacji zdobywania wiedzy. W szczególności eksplorację można osiągnąć poprzez wybór działań i/lub stanów, które były wybierane rzadziej [3], lub dawniej [1], lub zakłada się że mają wysoki błąd predykcji [17], lub wcześniej wykazywały wysoki błąd predykcji [16]. Wszystkie te metody można wspólnie nazwać optymistycznymi w obliczu niepewności, w skrócie OFU (od ang. *optimistic in the face of uncertainty*). Eksplorację ukierunkowaną można również uzyskać metodami Bayesowskimi, za pomocą próbkowania a posteriori (ang. *posterior sampling*)[14].

Techniki ukierunkowanej eksploracji są zwykle bardziej efektywne niż dowolna technika nieukierunkowana zarówno pod względem czasu nauki, jak i kosztów nauki. Praca [20] przedstawia twierdzenie i dowodzi wyższość ukierunkowanej eksploracji dla wielu skończonych dziedzin deterministycznych. Niemniej jednak, wiele problemów nie wymaga ukierunkowanej eksploracji do ich rozwiązania – wystarczają prostsze techniki nieukierunkowane opisane w poprzednim podrozdziale. Poniżej opisujemy kilka przykładów technik ukierunkowanej eksploracji.

Eksploracja oparta na licznikach. Ta technika eksploracji oparta się na adaptacyjnej mapie $N(s, a)$, zliczającej wykonania akcji a w każdym stanie s. Mapa ta służy do kierowania agenta do mniej zbadanych stanów. Prostym przykładem polityki wykorzystującej ten schemat eksploracji jest reguła "za każdym razem idź do najrzadziej odwiedzanego sąsiada". Bardziej wyrafinowana metoda eksploracji będzie łączyła zdobytą wiedzę z informacją o tym, jak często poszczególne akcje były wybierane np. w postaci bonusu eksploracyjnego dodawanego do funkcji Q:

$$U(s, a) = Q(s, a) + c \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{5.3}$$

Tutaj c jest stałą kontrolującą, jak bardzo agent eksploatuje swoją wiedzę, inną dla każdego środowiska, $Q(s, a)$ to funkcja oceniająca jakość akcji, agent uczy się przybliżać tę funkcję, a $U(s, a)$ ocenia użyteczność eksploracyjną akcji a w stanie s. Agent wybiera akcję która maksymalizuje użyteczność eksploracyjną: $\text{argmax}_a\, U(s, a)$. Akcja zostanie wybrana, jeśli dotychczas przynosiła duże nagrody lub była rzadko wybierana. Warto zauważyć, iż nawet jeśli akcja nie była ostatnio wybierana, ale jej jakość jest bardzo niska, wciąż nie zostanie wybrana.

Jeśli przestrzeń stanów i akcji jest zbyt duża, aby możliwe było wielokrotne odwiedzenie każdej pary stan-akcja, stosuje się metody przybliżające licznik (ang. *pseudo-count*), które generalizują wartość licznika pomiędzy podobnymi parami stan-akcja [2].

Eksploracja poprzez optymistyczną inicjalizację. W książce [18, s. 34-35] możemy znaleźć sposób prowadzenia ukierunkowanej eksploracji poprzez czystą eksploatację. Kluczowy pomysł opiera się na optymistycznym zainicjalizowaniu tablicy w Q-Learningu np. maksymalną możliwą sumą nagród do zdobycia. Wstępne przeszacowanie jakości każdej pary stan-akcja, $Q(s, a)$, sprawi, że agent będzie preferował niezbadane działań, a tym samym stany, ze względu na ich dużą ocenę jakości. W miarę uczenia się, zarówno estymaty jakości zbliżają się do rzeczywistych wartości, jak i ukierunkowana eksploracja stopniowo się zmniejsza. Technika ta może być łączona z inną techniką eksploracyjną, aby utrzymać eksplorację.

Eksploracja oparta na oszacowaniu górnym. Technika ta również opiera się na przeszacowaniu wartości funkcji Q. W tym przypadku wartości jakości są wyrażone jako górna granica oczekiwanej sumy nagród, inaczej niż w Q-Learningu, gdzie funkcja Q ma przybliżyć średnią oczekiwaną sumę nagród. Wykorzystana statystyka zapewnia poprawność tego ograniczenia górnego z pewnym z góry określonym prawdopodobieństwem. Uzyskujemy więc maksymalną, ale wciąż prawdopodobną jakość, jaką może osiągnąć dana akcja na podstawie dotychczas zebranych danych. Statystyka ta jest po części funkcją liczby doświadczeń, a wartości jakości są stopniowo pomniejszane w miarę zdobywania wiedzy. Wynika to z faktu, iż im więcej doświadczeń, tym węższy staje się przedział ufności na wartości jakości akcji i tym mniejsze górne ograniczenie.

Przykładami algorytmów wykorzystujących tę technikę eksploracji są Upper Confidence Bound (UCB) [18] oraz Optimistic Actor-Critic (OAC) [5]. W pierwszym przypadku, wykorzystywany jest licznik odwiedzeń par stan-akcja do oszacowania górnego ograniczenia jakości akcji. W drugim natomiast autorzy trenują kilka sieci neuronowych przybliżających funkcję Q i na tej podstawie oszacowują ograniczenie górne jakości akcji (technika nazywana w statystyce "bootstrap").

Eksploracja oparta na pamięci. W odróżnieniu od technik opartych na liczniku, techniki oparte na pamięci nie zliczają odwiedzeń stanów, natomiast zapisują ostatnio odwiedzone stany w buforze. Ponieważ stany mogą być duże, np. obrazki środowiska, metody te często trenują sieci neuronowe, lub wykorzystują inną metodę przetwarzania, np. skalowanie obrazka, do kodowania stanów w mniejszej przestrzeni. Pozwala to również generalizować pomiędzy podobnymi stanami. Następnie, tak utworzony bufor odwiedzonych stanów, może być wykorzystywany do kierowania agenta w rejony środowiska dotychczas rzadziej odwiedzane [1] lub prowadzenia agenta z powrotem do stanów, w których otrzymał on wcześniej wysokie nagrody i rozpoczęcie eksploracji z tamtego miejsca [6].

Eksploracja oparta na błędzie predykcji. Podobnie jak liczniki, błąd predykcji również może zostać wykorzystany do kierowania agenta w stronę niepoznanych dotychczas rejonów środowiska. Niezależnie od wybranego modelu, jego wysoki błąd predykcji oznacza, że agent wcześniej nie miał szansy nauczyć

się, jak reagować w danej sytuacji. Jednym ze sposobów implementacji tej techniki jest uczenie dedykowanej sieci neuronowej, $\hat{f}(s)$, jak przewidywać wyjścia bliźniaczej sieci, $f(s)$, ale inaczej zainicjalizowanej. Obie sieci jako wejście dostają stan środowiska s. Technika ta nazywa się destylacją wiedzy losowej sieci neuronowej (ang. *Random Network Distillation*) [4], ponieważ w miarę treningu, destylujemy predykcje losowej sieci do sieci trenowanej. Trenowana sieć będzie lepiej przewidywała wyjścia z losowej sieci, czyli będzie miała niższy błąd predykcji, w stanach, które były często odwiedzane, ponieważ miała wiele okazji do nauczenia się wyjść sieci losowej. Natomiast błąd predykcji będzie wysoki w nowych, niespotykanych wcześniej stanach. W ten sposób uzyskujemy sygnał uczący $r_i(s) = \left\| \hat{f}(s) - f(s) \right\|^2$. Ten, często nazywany "wewnętrzny", sygnał jest mieszany z nagrodą środowiska, sygnałem żewnętrznym"$r_e(s)$, dla zaobserwowanego stanu i agent jest uczony na podstawie tego sygnału $r(s) = r_e(s) + c \cdot r_i(s)$. Stała c waży stosunek sygnału wewnętrznego, eksploracyjnego r_i, do sygnału zewnętrznego, eksploatacyjnego r_e, i pozwala kontrolować agresywność eksploracji agenta. W miarę treningu, eksploracja będzie wygaszana, ponieważ trenowana sieć będzie lepiej przewidywać wyjścia sieci losowej.

Eksploracja oparta na próbkowaniu a posteriori. W artykule [12], autorzy wykorzystują statystyczną metodę *bootstrap* w jeszcze inny sposób. Autorzy trenują kilka sieci neuronowych przybliżających funkcję Q i wybierają jedną z nich na pewien okres, zazwyczaj jednego epizodu środowiska, do wybierania akcji. Ponieważ sieć neuronowa zwróci jakąś wartość dla każdej pary stan-akcja, nawet takiej, której nigdy wcześniej nie zaobserwowała, to różne inicjalizacje sieci neuronowych sprawią, iż sieci będę miały różne przewidywania dotyczące przestrzeni stanów i akcji – będą wstępnie optymistyczne lub pesymistyczne w stosunku do różnych rejonów środowiska. To natomiast sprawi, że agent będzie kierował się w różne rejony środowiska, w zależności, która sieć jest aktualnie wybrana do zbierania danych. Przypomina to metodę optymistycznej inicjalizacji z wcześniejszego paragrafu, ale zastosowaną do sieci neuronowych. W miarę postępu treningu, sieci lepiej przewidują rzeczywiste jakości akcji i ich wstępne przewidywania stają się mniej istotne przy wyborze akcji. Metoda ta osiąga dobre empiryczne rezultaty, a autorzy w innym artykule przedstawiają teoretyczne dowody na optymalność tej techniki eksploracji [15].

5.5 Dyskusja

Celem tego rozdziału było zilustrowanie podstawowych heurystyk używanych do efektywnej eksploracji. Opisując niektóre z najbardziej popularnych heurystyk, rozróżniliśmy dwie rodziny technik eksploracji: nieukierunkowane i ukierunkowane. Zajęliśmy się fundamentalnym kompromisem między eksploracją a eksploatacją i przedstawiliśmy parametry kontrolowania tego kompromisu. Jednakże istnieje kilka ograniczeń przyjętego podejścia, z którymi należy się zmierzyć przy zastosowaniu przedstawionych w tym rozdziale idei do bardziej złożonych dziedzin.

Złożone domeny. Wszystkie przedstawione schematy ukierunkowanej eksploracji mają na celu zbadanie całej przestrzeni stanów i akcji, zakładając, że możliwe jest wyczerpujące zbadanie tej przestrzeni lub przynajmniej części tej przestrzeni. W wielu problemach jest to rozsądne założenie, ale istnieje wiele problemów, w tym te typowo badane w kontekście rzeczywistych zastosowań sztucznej inteligencji, w których przestrzenie stanów i akcji są zbyt duże, aby można je było wyczerpująco zbadać. W takich problemach, zamiast szukać najlepszego rozwiązania, celem jest często skuteczne znalezienie dobrego, nieoptymalnego rozwiązania. W takich przypadkach inteligentny agent musi odróżnić istotne i nieistotne części problemu i odciąć eksplorację w nieistotnych regionach środowiska. Do pewnego stopnia połączenie eksploracji i eksploatacji może ograniczać eksplorację nieistotnych części problemu, ponieważ eksploatacja sprawia, iż agent koncentruje się na bardziej lukratywnych regionach środowiska. Wykluczanie nieistotnych części problemu można również uzyskać, dostarczając wiedzę zewnętrzną do prowadzenia eksploracji, która może pomagają ocenić użyteczność eksploracyjną działań bez faktycznego ich doświadczania.

Wiedza domenowa. W wielu podejściach do uczenia się, dostępna jest pewna wiedza domenowa. Ta wiedza może być reprezentowana za pomocą: inicjalizacji polityki, ograniczenia na akcje i/lub stany, wiedzy domenowej w postaci reguł środowiska, wcześniej podanych strategii eksploracji i tak dalej. Wiedza domenowa może drastycznie zmniejszyć złożoność uczenia się. Rozdział ten, nie brał pod uwagę roli eksploracji, kiedy dostępna jest dodatkowa wiedza o problemie. Niemniej jednak, ukierunkowane techniki powinny nadal się sprawdzać, zwłaszcza przy wykorzystaniu tejże wiedzy domenowej.

Eksploracja i generalizacja. Optymalna eksploracja mocno zależy od wyboru uczonego modelu (np. sieci neuronowej) i możliwości generalizacji tego modelu. Generalizacja przenosi wiedzę z pojedynczego przykładu do zwykle nieskończonego zbioru powiązanych sytuacji. Wpływ generalizacji na optymalną eksplorację jest oczywisty. Na przykład, jeśli agent ma wybór między zupełnie nowym działaniem a drugim działaniem podobnym do niektórych działań wybranych wcześniej, powinien spróbować nowej akcji, aby maksymalizować zdobywanie wiedzy, chociaż oba działania mogły nie być nigdy wcześniej testowane. Optymalna technika eksploracji musi uwzględniać, w jaki sposób model ekstrapoluje wiedzę z przykładów w celu oceny użyteczności eksploracyjnej każdej z akcji.

Bibliografia

1. Badia, A.P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., Blundell, C.: Never give up: Learning directed exploration strategies. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=Sye57xStvB`.
2. Bellemare, M.G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. In: Lee,

D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. pp. 1471–1479 (2016), https://proceedings.neurips.cc/paper/2016/hash/afda332245e2af431fb7b672a68b659d-Abstract.html.
3. Brafman, R.I., Tennenholtz, M.: R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. The Journal of Machine Learning Research **3**, 213–231 (2002), http://jmlr.org/papers/v3/brafman02a.html.
4. Burda, Y., Edwards, H., Storkey, A., Klimov, O.: Exploration by random network distillation. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=H1lJJnR5Ym.
5. Ciosek, K., Loftin, R., Vuong, Q., Hofmann, K.: Better exploration with optimistic actor-critic. In: Advances in Neural Information Processing Systems (2019).
6. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O., Clune, J.: First return, then explore. Nature **590**(7847), 580–586 (2021). https://doi.org/10.1038/s41586-020-03157-9, https://doi.org/10.1038/s41586-020-03157-9.
7. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 1582–1591. PMLR (2018), http://proceedings.mlr.press/v80/fujimoto18a.html.
8. Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Levine, S.: Soft actor-critic algorithms and applications. CoRR **abs/1812.05905** (2018), http://arxiv.org/abs/1812.05905.
9. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018).
10. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016), http://arxiv.org/abs/1509.02971.
11. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236, https://doi.org/10.1038/nature14236.

12. Osband, I., Blundell, C., Pritzel, A., Van Roy, B.: Deep exploration via bootstrapped DQN. In: Advances in Neural Information Processing Systems (2016).
13. Osband, I., Roy, B.V., Russo, D.J., Wen, Z.: Deep exploration via randomized value functions. J. Mach. Learn. Res. **20**, 124:1–124:62 (2019), http://jmlr.org/papers/v20/18-339.html.
14. Osband, I., Van Roy, B.: Why is posterior sampling better than optimism for reinforcement learning? In: 34th International Conference on Machine Learning, ICML 2017 (2017).
15. Osband, I., Van Roy, B.: Why is posterior sampling better than optimism for reinforcement learning? In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 2701–2710. PMLR (06–11 Aug 2017), http://proceedings.mlr.press/v70/osband17a.html.
16. Schmidhuber, J.: Adaptive confidence and adaptive curiosity. Tech. rep., Institut fur Informatik, Technische Universitat Munchen, Arcisstr. 21, 800 Munchen 2 (1991).
17. Simmons-Edler, R., Eisner, B., Yang, D., Bisulco, A., Mitchell, E., Seung, H.S., Lee, D.D.: Reward prediction error as an exploration objective in deep RL. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020. pp. 2816–2823. ijcai.org (2020). https://doi.org/10.24963/ijcai.2020/390, https://doi.org/10.24963/ijcai.2020/390.
18. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018).
19. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Proceedings of the 12th International Conference on Neural Information Processing Systems. p. 1057–1063. NIPS'99, MIT Press, Cambridge, MA, USA (1999).
20. Thrun, S.: The role of exploration in learning control. In: White, D., Sofge, D. (eds.) Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches. Van Nostrand Reinhold, Florence, Kentucky 41022 (1992).
21. Wang, C., Wu, Y., Vuong, Q., Ross, K.: Striving for simplicity and performance in off-policy DRL: Output normalization and non-uniform sampling. Proceedings of the 37th International Conference on Machine Learning **119**, 10070–10080 (13–18 Jul 2020), http://proceedings.mlr.press/v119/wang20x.html.
22. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8**(3), 229–256 (1992). https://doi.org/10.1007/BF00992696, https://doi.org/10.1007/BF00992696.

6. Jak wykraść złoto smokowi? – uczenie ze wzmocnieniem w świecie Wumpusa

Karol Draszawka

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
kadr@eti.pg.edu.pl

Streszczenie
Niniejszy rozdział zawiera łagodne wprowadzenie do problematyki uczenia ze wzmocnieniem, w którym podstawy teoretyczne wyjaśniane są na przykładzie przewodnim, jakim jest zagadnienie nauczenia agenta poruszania się w świecie potwora o imieniu Wumpus (ang. *Wumpus world*), klasycznym środowisku do testowania logicznego rozumowania agentów (problem nietrywialny dla algorytmów uczenia ze wzmocnieniem). Przedstawiona jest główna idea uczenia ze wzmocnieniem. Wprowadzono formalizację wieloetapowych procesów decyzyjnych w oparciu o model Procesu Decyzyjnego Markowa, zaznaczono dylemat eksploracja - eksploatacja, przestawiono klasyczny algorytm Q-learning, a także jego wariant głęboki, tj. algorytm DQN, jako przedstawiciela dziedziny głębokiego uczenia ze wzmocnieniem. Zaznaczono problemy częściowej obserwacji stanu oraz rzadko występującej nagrody, jak i sposoby poradzenia sobie z nimi, w tym: kumulacja wiedzy o stanie na podstawie częściowych obserwacji, kształtowanie funkcji nagrody, tzw. *curriculum learning* oraz innowacyjną *podwójną* strategię ϵ-zachłanną.

Słowa kluczowe: Uczenie ze wzmocnieniem, Q-learning, DQN, Wumpus World.

6.1 Wprowadzenie

Uczenie ze wzmocnieniem (ang. *reinforcement learning – RL*) jest coraz prężniej rozwijającą się dziedziną badań z obszaru sztucznej inteligencji i uczenia maszynowego. Pionierzy w tych dyscyplinach, czołowe uniwersytety techniczne świata czy firmy, takie jak DeepMind i OpenAI, wkładają ogrom pracy w rozwój algorytmów i metod RL, prześcigając się tutaj w innowacjach i ukazując coraz to bardziej imponujące możliwości systemów tego typu: wyuczenie najsilniejszych silników, przewyższających siłą gry ludzkich arcymistrzów, w klasycznych grach planszowych, takich jak szachy czy go [15, 16], jak i w popularnych e-sportowych strategicznych grach komputerowych, takich jak Starcraft II czy Dota 2 [3, 18]; nauczenie fizycznych robotów chodzenia i biegania [5], a nawet nauczenie zręcznego manipulowania robotyczną dłonią tak, by potrafiła ona ułożyć fizyczną kostkę Rubika nawet pomimo nietypowych utrudnień [2].

Naukowcy związani m. in. z wymienionymi instytucjami udoskonalają algorytmy uczenia ze wzmocnieniem. Te najnowsze metody są często dość skomplikowane matematycznie i wymagają dużej wiedzy dziedzinowej, aby ich działanie było zrozumiałe. Z drugiej strony, najczęściej są one rozwinięciem pewnych podstawowych algorytmów RL, które w dalszym ciągu stanowią podstawę działania (a co najmniej podstawę zrozumienia) tych bardziej wyrafinowanych wersji oprogramowania uczącego się.

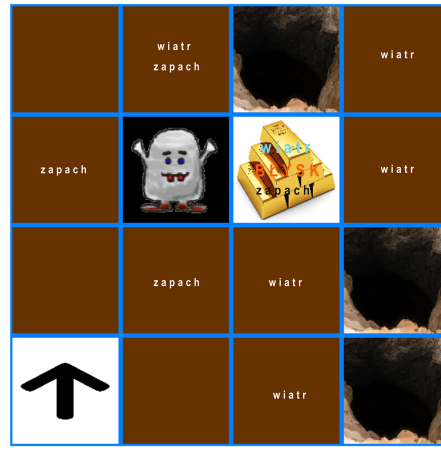
Niniejszy rozdział ma na celu zaprezentowanie dwóch podstawowych algorytmów uczenia ze wzmocnieniem: tzw. Q-learningu, a więc metody klasycznej, oraz algorytmu DQN (ang. *Deep Q-Network*), wykorzystującego głębokie sieci neuronowe, który zapoczątkował erę tzw. głębokiego uczenia ze wzmocnieniem. W celu demonstracji praktycznego użycia tych algorytmów, zostaną one wykorzystane do rozwiązania problemu nauki agenta poruszającego się w niebezpiecznym świecie smoka Wumpusa. Główne koncepty uczenia ze wzmocnieniem oraz formalizacja wieloetapowych procesów decyzyjnych, do którego ten typ uczenia jest stosowany, a także sam problem nauki agenta w świecie Wumpusa, przedstawiono w sekcji 6.2. Q-learning i jego użycie do rozwiązania uproszczonej wersji problemu opisano w sekcji 6.3. W sekcji 6.4 zaprezentowano algorytm DQN oraz jego usprawnienia. Szczegóły dostosowania algorytmu DQN do rozwiązywanego problemu wiodącego, sposoby wykorzystania wiedzy eksperckiej na temat tego problemu, tak by ułatwić proces nauki oraz uzyskane wyniki zawarto w sekcji 6.5. Całość kończy krótkie podsumowanie.

6.2 Uczenie ze wzmocnieniem na przykładzie świata Wumpusa

6.2.1 Przykład problemu: świat Wumpusa

Świat Wumpusa to klasyczne już środowisko, spopularyzowane przez [13], wymyślone jako prosty test na umiejętność zdobywania wiedzy, logicznego rozumowania i planowania agenta przy podejmowaniu akcji. W świecie tym, agent wchodzi do jaskini potwora Wumpusa, która składa się z 16 pomieszczeń, rozmieszczonych na planie 4×4 (zob. rys. 6.1), połączonymi ze sobą przejściami w centralnej części ścian – nie jest możliwe przejście agenta pomiędzy polami stykającymi się jedynie narożnikami. Gdzieś w jaskini, w ustalonych pomieszczeniach znajdują się obiekty niebezpieczne dla agenta: potwór Wumpus – który, choć nieruchliwy (nie zmienia położenia), w bezpośrednim kontakcie z agentem jest śmiertelnie niebezpieczny, oraz pułapki w postaci ukrytych głębokich dołów – przepaści – do których wpadnięcie także kończy przygodę agenta. Oprócz tych niebezpieczeństw, w jednym z pomieszczeń znajduje się upragniony przez agenta skarb – złoto. Wchodząc do jaskini agent nie wie, jakie jest w niej rozmieszczenie zagrożeń, ani gdzie jest złoto. Pomocą dla niego są jego zmysły, gdyż Wumpus wydziela nieprzyjemny zapach wyczuwalny w pomieszczeniach bezpośrednio sąsiadujących z tym, w którym się znajduje, a doły można wyczuć z sąsiednich pól na podstawie delikatnych podmuchów wiatru. Złoto z kolei, mimo panujących

ciemności, odbija resztki światła i dzięki temu jest dostrzegalne dla agenta, jeśli jest w pomieszczeniu, w którym się znajduje. Dodatkowo, bohater posiada łuk i jedną strzałę, która, jeśli zostanie wystrzelona w kierunku Wumpusa, unieszkodliwia go (ten wydaje jeszcze głośny ryk słyszalny przez strzelca). W tym świecie celem agenta jest bezpieczne odnalezienie złota, podniesienie go, a następnie powrót do punktu startowego i wyjście ze skarbem z jaskini.



Rysunek 6.1. Świat Wumpusa - losowa generacja położenia pól ze złotem, z dziurami i z Wumpusem

Jest możliwe napisanie bezbłędnej sztucznej inteligencji takiego agenta przy pomocy programowania w logice (zob. [7, 13]). Tutaj jednak, nie jest naszym celem znalezienie sposobu *napisania* odpowiedniego systemu sterowania w momencie, w którym my, jako autorzy programu, już wiemy jakie to działanie powinno być. Problem tutaj stawiany dotyczy *odkrycia* i *nauczenia* agenta właściwego sposobu działania, tak by kończył on wędrówkę po jaskini sukcesem, na podstawie samej interakcji agenta ze środowiskiem.

6.2.2 Uczenie ze wzmocnieniem

Ogólną metodę rozwiązania problemów takich, jak powyższy, dostarcza uczenie ze wzmocnieniem. Stosowane jest ono tam, gdzie optymalizacji podlega pewien *wieloetapowy proces decyzyjny*, tj. szukana jest taka strategia działania, która maksymalizuje stosowny *zysk* wynikający z interakcji agenta ze środowiskiem. Kluczowe jest tutaj *szukanie* optymalnego zachowania: w odróżnieniu od uczenia nadzorowanego, nie ma w uczeniu ze wzmocnieniem autorytatywnie danego zbioru uczącego, w którym podane są gotowe przykłady odpowiedniego działania. Przeciwnie, uczenie ze wzmocnieniem jest przydatne szczególnie tam, gdy nie jest z góry wiadomo, jakie jest najwłaściwsze działanie w zależności od zastanej

sytuacji, tak by długoterminowo osiągnąć największe korzyści. Dzięki uczeniu ze wzmocnieniem możliwe jest odnalezienie innowacyjnych strategii działania w zadanym wieloetapowych procesach decyzyjnych.

Szukanie optymalnej strategii działania w uczeniu ze wzmocnieniem to rodzaj sformalizowanej *metody prób i błędów*, podczas której agent próbuje, często, szczególnie w początkowej fazie uczenia, zupełnie na ślepo, różnych działań, a następnie rewiduje ich użyteczność na podstawie zdobytego, w interakcji ze środowiskiem, doświadczenia. Ta ciągła interakcja między agentem a środowiskiem, w którym się porusza, to centralny i niezbywalny rdzeń uczenia ze wzmocnieniem, nazywany *pętlą uczenia ze wzmocnieniem*: agent, znajdując się w pewnej sytuacji w środowisku, podejmuje jakieś działanie, które w konsekwencji umiejscawia go w nowej sytuacji, z czym może się również wiązać pewna nagroda (pozytywne wzmocnienie) lub kara (wzmocnienie negatywne, osłabienie).

W świecie Wumpusa, w każdym kroku pętli uczenia ze wzmocnieniem agent wybiera jedną z dostępnych mu akcji (np. skręt w prawo, krok do przodu czy strzał z łuku), czego konsekwencją będzie zmiana stanu środowiska i percepcji środowiska przez agenta oraz odebranie sygnału nagrody (np. dużej dodatniej, gdy wyjdzie z jaskini ze złotem lub dużej negatywnej np. za wpadnięcie na pole śmiercionośnego Wumpusa).

Pętla działania i doświadczania konsekwencji trwa tak długo, dopóki agent nie znajdzie się w jednym ze stanów końcowych (lub przerwiemy pętlę ze względu na przyjętą maksymalną liczbę kroków). Pojedynczy epizod nauki zostaje wtedy zakończony. Takich epizodów agent musi wykonać wiele, zanim zgromadzone doświadczenie pozwoli mu na obieranie najwłaściwszych decyzji.

6.2.3 Formalizacja: Procesy Decyzyjne Markowa

Aby cel i metody uczenia ze wzmocnieniem ująć precyzyjnie, wieloetapowe procesy decyzyjne przedstawia się za pomocą modelu Procesu Decyzyjnego Markowa (ang. *Markov Decision Process* – MDP). Pierwszym elementem tego modelu jest $\mathcal{S}$, tj. zbiór wszystkich możliwych stanów (S to pojedynczy, konkretny stan), w jakich może się znaleźć agent, w tym wyróżniony stan inicjalny S_0, oraz stany końcowe S_{term}. Następnie mamy $\mathcal{A}$, czyli zbiór akcji, jakie agent może podejmować (A to pojedyncza akcja). $p(s'|s,a)$ to funkcja określająca prawdopodobieństwo przejścia stanu s w stan s' po wybraniu akcji a. Środowisko jest deterministyczne, gdy dla dowolnej kombinacji stanu S i akcji A, stan następny S' zachodzi z prawdopodobieństwem równym 1. $r(s,a,s')$ to funkcja nagrody: określa ona, jaką bezpośrednią nagrodę R agent otrzymuje po przejściu ze stanu S do stanu S' poprzez wykonanie akcji A.

Do określenia celu optymalizacji niezbędne jest zdefiniowanie dyskontowanego zwrotu G_t, który :

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + R = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{6.1}$$

gdzie γ to współczynnik dyskontowania, $0 \leq \gamma \leq 1$, będący parametrem kontrolującym to, jak ważne mają być dla agenta nagrody, które w perspektywie

ma do zdobycia w przyszłych krokach. Np. przyjęcie wartości $\gamma = 0$ oznacza, że oczekujemy agenta skrajnie krótkowzrocznego, który w kalkulacji G_t uwzględnia wyłącznie bezpośrednią nagrodę.

Przez strategię działania $\pi(a|s)$ (ang. *policy*) rozumiana jest funkcja wyznaczająca akcję a na podstawie stanu s. Celem uczenia ze wzmocnieniem jest zatem znalezienie optymalnej strategii działania π^*, która maksymalizuje oczekiwany zwrot G_t, niezależnie od stanu S_t, w jakim w kroku t agent się znajduje.

6.3 *Q*-learning

6.3.1 Funkcja użyteczności *Q*

W algorytmie Q-learning [19], uczenie optymalnej strategii działania w odbywa się pośrednio: algorytm stara się właściwie oszacować wartość użyteczności każdej możliwej akcji a w każdym możliwym stanie s. W tym celu wprowadza się funkcję użyteczności pary stan-akcja $Q(s, a)$:

$$Q(s,a) \doteq \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \quad (6.2)$$

Zgodnie z powyższym wzorem, użyteczność Q można oszacować dla pewnej strategii działania π, estymując wartość oczekiwaną zwrotu, który jest do uzyskania za pomocą tej strategii. Okazuje się jednak, że jeśli *w czasie uczenia* strategia wyboru akcji π jest taka, że asymptotycznie pozwala ona na odwiedzenie wszystkich stanów i wykonanie wszystkich akcji w tych stanach, to oszacowane w ten sposób wartości użyteczności każdej pary stan-akcja będą optymalne. Oznacza to, że *docelowa* strategia działania polegająca na wybieraniu w danym stanie zawsze akcji maksymalizującej ową użyteczność będzie maksymalizowała zwrot. Dla takiej optymalnej użyteczności zachodzi wtedy tzw. równanie optymalności Bellmana dla funkcji użyteczności pary stan-akcja:

$$Q_*(s,a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') | S_t = s, A_t = a \right] \quad (6.3)$$

6.3.2 Uczenie metodą różnic czasowych

Ze względu na sposób, w jaki Q-learning stara się oszacować wartości funkcji użyteczności, algorytm ten zalicza się do tzw. metod *różnic czasowych* (ang. *temporal difference*). Różnica czasowa, o której tu mowa, to różnica między wartością użyteczności szacowaną na podstawie wyłącznie ostatnio zdobytego doświadczenia oraz wartością szacowaną przed tym doświadczeniem. Ostatnio zdobyte doświadczenie, a więc obserwacja nagrody R i kolejnego stanu S', daje szacunek użyteczności podjętej akcji A w stanie S równy $R + \gamma \max_a Q(S', a)$ (przy założeniu optymalnego działania oraz przyjęciu współczynnika dyskontowania γ). A zatem różnica między oceną użyteczności akcji A w stanie S, którą

właśnie agent doświadczył, a jego wcześniejszymi szacunkami tej użyteczności to:

$$\delta = R + \gamma \max_a Q(S', a) - Q(S, A) \tag{6.4}$$

Oczywiście właśnie zdobyte doświadczenie nie powinno zupełnie wymazywać poprzednich doświadczeń, wcześniejszych estymat użyteczności, a jedynie delikatnie je korygować. Dlatego, nowa estymata użyteczności jest w metodzie różnic czasowych delikatnie modyfikowana przez dodanie małej części różnicy (6.4) do wcześniejszej wartości estymaty:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right] \tag{6.5}$$

Hiperparametr α – współczynnik uczenia, $\alpha \in R^+$, kontroluje stopień wpływu ostatniego doświadczenia na modyfikację estymat. Wybór jego wartości jest kluczowy dla sukcesu uczenia: zbyt duża wartość α może sprawić, że uczenie nie będzie zbieżne (nie dojdzie do ustalenia stabilnej wartości $Q(s, a)$), natomiast zbyt mała wartość α przełoży się na powolne uczenie.

Dodajmy, że aktualizacja z (6.5) to przykład tzw. *bootstrapingu*, a więc poprawa oszacowania wartości, która sama korzysta z innego oszacowania (tu, estymaty jakości stanu kolejnego S'). Taka metoda mimo wszystko działa (przy odpowiednio małym α), bo każda nowa estymata uwzględnia rzeczywiste doświadczenie, agregując je w kolejnych oszacowaniach.

Można jeszcze zauważyć, porównując (6.4) z (6.3), że w przypadku osiągnięcia optimum, wartość oczekiwana różnic czasowych wynosi 0. Tym samym, odpowiednio mała wartość bezwzględna różnic czasowych może służyć jako kryterium zatrzymania uczenia metodą różnic czasowych. W praktyce jednak uczenie przerywane jest zazwyczaj wcześniej, po osiągnięciu zadanej maksymalnej liczby iteracji.

6.3.3 Ucząca strategia wyboru akcji

Jak wspomniano wyżej, strategia wyboru akcji π w trakcie szacowania wartości funkcji użyteczności powinna umożliwić asymptotyczne odwiedzenie wszystkich stanów i wykonanie w nich wszystkich akcji. W praktyce, ze względu na skończony czas uczenia, strategia taka powinna odpowiednio balansować między *eksploracją*, mających na celu lepsze oszacowanie użyteczności danej akcji A w danym stanie S, a *eksploatacją*, tj. wyborem akcji maksymalizującej użyteczność w danym stanie, po to by skupić przeszukiwanie kolejnych stanów i akcji przede wszystkim do tych najbardziej prosperujących. Wśród wielu mniej lub bardziej skomplikowanych strategii, które starają się znaleźć kompromis między eksploracją i eksploatacją, bardzo prostą, a dość efektywną i często wykorzystywaną strategią, jest strategia ϵ-zachłanna, przedstawiona jako Alg. 1

W strategii ϵ-zachłannej, parametr ϵ określa prawdopodobieństwo eksploracji. Jeśli jest stały, parametr ten przyjmuje wartości rzędu $1e^{-2} - 1e^{-5}$. W wielu aplikacjach korzystne jest rozpoczęcie nauki z silnym naciskiem na eksplorację, a następnie stopniowe przenoszenie go na eksploatację. W strategii ϵ-zachłannej

Algorithm 1: Strategia wyboru akcji ϵ-zachłanna

Input: Stan S, tablica/funkcja $Q(s, a)$
Parameters : ϵ
Output: Wybrana akcja A

if $r \sim U(0, 1) > \epsilon$ **then**
  $A \leftarrow \text{argmax}_a\, Q(S, a)$ – zachłanny wybór akcji (eksploatacja)
else
  $A \leftarrow$ losowa akcja ze zbioru $\mathcal{A}(S)$ (eksploracja)
end

takie zachowanie można osiągnąć stosując malejące ϵ, np. przemnażając współczynnik eksploracji ϵ przez mnożnik $\epsilon_{\text{decay}} < 1$, aż do osiągnięcia pewnego minimum eksploracji ϵ_{min}:

$$\epsilon \leftarrow \max\left(\epsilon \cdot \epsilon_{\text{decay}}, \epsilon_{\text{min}}\right) \tag{6.6}$$

Zauważmy, że ta prosta strategia, przy wyborze akcji eksplorującej działa zupełnie ślepo, nie uwzględniając ani dotychczasowych estymat użyteczności możliwych do podjęcia akcji, ani liczby prób podjęcia tychże akcji. Bardziej zaawansowane metody mogą tę wiedzę uwzględniać (zob. np. metodę softmax czy UCB [17]).

6.3.4 Algorytm

Pełny algorytm Q-learning dla problemu epizodycznego przedstawia Alg. 2.

Przyjęto maksymalną liczbę epizodów M, po której następuje zakończenie uczenia oraz współczynnik dyskontowania nagród równy γ. Funkcja użyteczności $Q(s, a)$ ma tutaj postać tabelaryczną, tj. dla każdej kombinacji S oraz A, z tablicy wartości Q możemy odczytać odpowiednią wartość lub do niej wartość wpisać.

Należy zauważyć, że klasyczny, tabelaryczny Q-learning nadaje się do rozwiązywania jedynie problemów z niewielką przestrzenią stanów i akcji, gdyż potrzebuje $O(|\mathcal{S}| \cdot |\mathcal{A}|)$ pamięci oraz czasu obliczeniowego. W sposób oczywisty nie nadaje się także do problemów z ciągła przestrzenią stanów lub akcji.

Dodajmy, że algorytm Q-learning klasyfikowany jest jako tzw. algorytm *off-policy*, gdyż docelowa strategia działania (i wykorzystywana do aktualizacji estymat) – tu strategia zachłanna, jest inna niż ucząca strategia działania (strategia pozyskiwania doświadczenia) – tu np. strategia ϵ-zachłanna. Klasycznym odpowiednikiem Q-learningu typu *on-policy* jest algorytm SARSA (zob. [17]).

6.3.5 Zastosowanie w Wumpus World

Ponieważ oryginalny problem agenta uczonego ze wzmocnieniem w świecie Wumpusa zawiera zbyt dużą przestrzeń możliwych stanów, aby zademonstrować efektywne działanie Q-learningu, pośród kilku przygotowanych wariantów gry, któ-

Algorithm 2: Algorytm Q-learning

Input: współczynnik uczenia – α, strategia wyboru akcji π (np. ϵ-zachłanna)
Output: Tablica $Q(s, a)$ z estymatami użyteczności każdej pary s i a

Zainicjuj $Q(s, a)$ dowolnymi wartościami dla wszystkich s i a, ale dla $Q(S_{term}, \cdot) = 0$
for *Epizod 1 do M* **do**
  Reset środowiska (inicjalizacja S)
  $t \leftarrow 0$
  while S *nie jest stanem końcowym* **and** $t < T$ **do**
    Wybierz akcję A na podstawie S używając strategii wyboru akcji π
    (Opcjonalnie: zaktualizuj strategię π, np. zredukuj ϵ)
    Wykonaj akcję A, zaobserwuj R, S'
    Zaktualizuj estymatę $Q(S, A)$ wg wzoru 6.5
    $S \leftarrow S'$
    $t \leftarrow t + 1$
  end
end

rych podsumowanie przestawia Tab. 6.1, przygotowano między innymi proste warianty gry: *Poziom0* i *Poziom0-ND*.

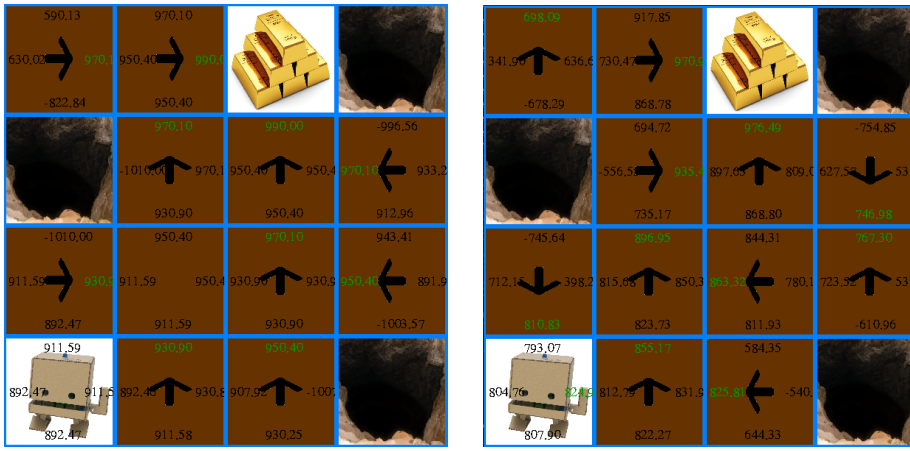
Tablica 6.1. Warianty gry Świat Wumpusa.

Wariant gry	**Rozmieszczenie przeciwników**	**Przestrzeń akcji**	$S'\|S, A$	**Liczba dziur**	**Liczba potworów**
Poziom0	Stałe	uproszczona: 4 akcje	Deterministyczny	3	0
Poziom0-ND	Stałe	uproszczona: 4 akcje	Niedeterministyczny	3	0
Poziom1	Losowe	pełna: 6 akcji	Deterministyczny	0	1
Poziom2	Losowe	pełna: 6 akcji	Deterministyczny	1	1
Poziom3	Losowe	pełna: 6 akcji	Deterministyczny	2	1
Poziom4	Losowe	pełna: 6 akcji	Deterministyczny	3	1

W wariantach *Poziom0* i *Poziom0-ND* rozmieszczenie niebezpieczeństw (tu, wyłącznie 3 doły, usunięto Wumpusa) jest zawsze jednakowe, tj. w każdym epizodzie, 3 doły znajdują się w miejscach z góry ustalonych i zawsze tych samych. Z tego powodu, akcje agenta mogą zostać podejmowane wyłącznie na podstawie numeru pomieszczenia, w którym agent się znajduje - nie musi on w ogóle bazować na swoich zmysłach. w konsekwencji, obserwację agenta można zredukować wyłącznie do numeru pomieszczenia, w którym się on znajduje (16 stanów). Przestrzeń akcji jest tu także zredukowana: w każdym stanie, agent wybiera spośród jedynie 4 akcji: ruch w lewo, ruch w prawo, ruch w górę, ruch w dół (sterowanie *z lotu ptaka*). Podniesienie skarbu odbywa się automatycznie, gdy tylko agent wejdzie na pole ze złotem, co, dodatkowo, od razu kończy epizod (w tych uproszczonych wariantach, agent nie musi powrócić do punktu początkowego i wyjść

z jaskini). Uwzględniając te uproszczenia, tabela zawierająca wartości użyteczności wszystkich par stan-akcja ma w tym wypadku wymiar 16×4. Tym samym, Q-learning idealnie do tego problemu się nadaje.

Rysunek 6.2 przedstawia wynik działania Q-learningu dla tych uproszczonych wariantów problemu.



(a) (b)

Rysunek 6.2. Strategie wyboru akcji odnalezione przez algorytm Q-learning, wraz z wartościami estymat $Q(s, a)$ (strzałki pokazują akcję, która maksymalizuje Q w danym stanie), dla wariantu uproszczonego: po lewej - deterministycznego, po prawej - niedeterministycznego. Warto zauważyć, że w przypadku niedeterministycznym (wybrany kierunek zostaje faktycznie wykonany z prawdopodobieństwem 0.8, możliwe jednak, że agent wyląduje na polu na prawo lub na lewo względem tego kierunku – każda z tym możliwości ma prawdopodobieństwo 0.1), w pomieszczeniach graniczących z dołami agent nauczył się wybierać ruch w kierunku przeciwnym do dołu – jedyny, dla którego prawdopodobieństwo śmierci w dole jest zerowe

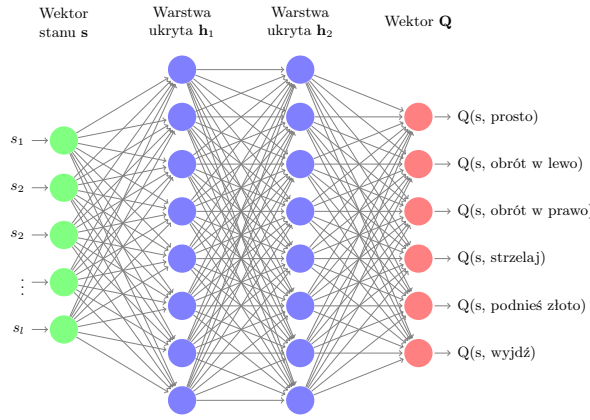
6.4 Algorytm DQN

Algorytm DQN (od użytych w metodzie *Deep Q-Networks*) [10, 11], stanowi rozwinięcie Q-learningu, które z powodzeniem wykorzystano w problemach z bardzo dużą liczbą stanów. Artykuł wprowadzający metodę, demonstrował działanie algorytmu DQN z powodzeniem uczącego się wygrywających strategii w szereg klasycznych gier Atari, tj. w zadaniu sterowaniu agentem ze świata danej gry w celu osiągnięcia w niej sukcesu, na podstawie zrzutu ekranu gry. Tak spektakularny sukces zapoczątkował bardzo szybki rozwój metod tzw. *głębokiego* uczenia ze wzmocnieniem.

6.4.1 Sieć neuronowa jako funkcja Q

Jak zauważyliśmy, ograniczeniem klasycznego Q-learningu jest to, że potrzebuje on miejsca na zapisanie każdej możliwej kombinacji stanu S i akcji A, a co więcej, potrzebuje czasu na oszacowanie każdej z wartości użyteczności $Q(S, A)$, gdyż nie ma żadnej generalizacji między stanami podobnymi. W algorytmie DQN, kluczowe jest zastąpienie podejścia tabelarycznego przez funkcją aproksymującą wartość użyteczności dowolnej pary stan-akcja: $Q(s, a; \theta) \approx Q(s, a)$, o formie której decyduje wektor parametrów θ. Od dawna znane były próby tego typu, czy to przy pomocy funkcji liniowych, czy nieliniowych, a nawet sieci neuronowych (zob. [17]), ale dopiero w [10] pokazano, jak z sukcesem wykorzystać do tego celu głębokie sieci neuronowe, np. sieci splotowe.

Aproksymująca sieć Q-Network, parametryzowana wagami θ, na wejściu otrzymuje wektor $\mathbf{s}$ (w ogólności może to być tensor) stanowiący przyjętą reprezentację stanu doświadczonego przez agenta, a jej wyjściem jest wektor $\mathbf{Q} \in \mathbb{R}^{|\mathcal{A}|}$ (przykład takiej sieci przedstawia rys. 6.3). Tego typu wielowyjściowa sieć pozwala w jednym przebiegu policzyć aproksymację wartości użyteczności wszystkich akcji w zadanym stanie, reprezentowanym przez wejście $\mathbf{s}$, co nie byłoby możliwe przy architekturze naiwnej, w której oba komponenty pary stan-akcja znajdowałyby się na wejściu, a wyjście byłoby wartością skalarną.



Rysunek 6.3. Architektura sieci Q-Network użyta w problemie Wumpus World

6.4.2 Dwie sieci neuronowe

Uczenie sieci odbywa się standardową metodą gradientową, przy czym funkcją straty w iteracji i jest średni błąd kwadratowy:

$$L_i(\theta_i) = \mathbb{E}_{S,A\sim\pi}\left[\left(y_i - Q(S, A; \theta_i)\right)^2\right], \tag{6.7}$$

gdzie $y_i = \mathbb{E}_{S' \sim \pi}\left[R + \gamma \max_a Q(S', a; \theta_i^-)\right]$ to cel regresji kroku i. π to strategia wyboru akcji w czasie uczenia (typowo ϵ-zachłanna, podobnie jak w zwykłym Q-learningu), na podstawie której (oraz dzięki interakcji ze środowiskiem) kolekcjonowane są kolejne dane uczące w postaci czwórki (S, A, R, S').

Można zauważyć, że, w odróżnieniu od klasycznego, nadzorowanego problemu regresji, tutaj cel optymalizacji zależy od optymalizowanych parametrów θ (*bootstraping* estymat użyteczności). Z tego powodu, wprowadzono drugą sieć (tzw. *target network*), która jest identyczna z siecią główną, z tym, że jej parametry θ^- są kopiowane z sieci oryginalnej jedynie co pewną, ustaloną liczbę kroków C. Taki zabieg pozwala krótkoterminowo pozorować stacjonarność wartości y_i, w efekcie zwiększając stabilność uczenia.

Zastosowane w oryginalnym algorytmie kopiowanie parametrów między sieciami skokowo co C kroków wprowadza jednak własne źródło niestabilności uczenia. W późniejszych pracach (np. w [9]) zastąpiono skokowe kopiowanie miękkim, ale ciągłym podążaniem, nazywane uśrednianiem Polyaka:

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^- \tag{6.8}$$

gdzie τ to mała wartość skalarna. Uśrednianie Polyaka występuje po każdym kroku optymalizacji parametrów θ.

Gradient funkcji straty (6.7) względem parametrów θ uczonej sieci Q-Network ma postać:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{S,A,R,S'}\left[\left(R + \gamma \max_a Q\left(S', a; \theta_i^-\right) - Q\left(S, A; \theta_i\right)\right) \nabla_{\theta_i} Q(S, A; \theta_i)\right] \tag{6.9}$$

Można zauważyć, że powyższy wzór można sprowadzić do (6.5), zastępując wartość oczekiwaną pojedynczą próbką doświadczenia S, A, R, S' oraz przyjmując parametryzację identycznościową (odtworzenie tabelarycznej wersji Q-learningu), gdyż wtedy $\nabla_{\theta_i} Q(S, A; \theta_i) = 1$.

6.4.3 Powtarzanie doświadczenia

Uczenie sieci neuronowej Q-Network odbywa się metodą optymalizacji gradientowej w wariancie z minipaczkami (ang. *mini-batch*), co jest typowe dla uczenia głębokiego. Wartość oczekiwana gradientu (6.9) przybliżona jest na podstawie paczki złożonej z B przykładów uczących. Przykłady te pochodzą z bufora pamięci (ang. *memory buffer*), który przechowuje N ($N \gg B$) ostatnich czwórek doświadczenia (S, A, R, S'). W oryginalnej wersji, B przykładów z bufora jest losowana (stosując rozkład jednorodny), bez uwzględniania ważności danej czwórki doświadczenia, choć możliwe jest tu stosowanie systemów priorytetowych uwzględniających taki czynnik w losowaniu [8, 14].

Użycie bufora pamięci, z którego losowane są przykłady uczące sieć Q-Network, ma kilka zalet nad standardowym uczeniem *online* (czyli tylko jednym, najnowszym przykładem). Po pierwsze, możliwe jest wielokrotne użycie tego samego przykładu, co znacznie zwiększa efektywność wykorzystania danych. Po

drugie, uczenie *online* przykładami na bieżąco pozyskiwanymi z doświadczenia jest nieefektywne ze względu na silną korelację między kolejnymi przykładami. Korelacja ta wynika ze stanu świata gry/symulacji i jego dynamiki. Losowe próbkowanie bufora pamięci sprawia, że przykłady uczące podawane na wejście sieci nie są skorelowane. Po trzecie, losowanie przykładów z dużego bufora pamięci doświadczenia pozwala też zmniejszyć oscylacje uczonej sieci między konkurującymi strategiami działania. Takie oscylacje, szczególne widoczne w warunkach *online*, mają miejsce ponieważ próbki uczenia (wynikające z podjęcia akcji zachłannych) generowane są w oparciu o tę samą sieć, która jest uczona (częściową pomocą jest tu zastosowanie drugiej sieci *target network*).

Dodajmy, że użycie bufora pamięci do powtarzania doświadczenia (ang. *experience replay*) jest możliwe dzięki temu, że algorytm DQN, podobnie jak *Q*-learning, jest algorytmem typu *off-policy*, i możliwe jest uczenie strategii docelowej (typowo, strategii zachłannej opartej o aktualną sieć Q-Network) z użyciem próbek doświadczenia zgromadzonych przy użyciu innej strategii (typowo, strategii ϵ-zachłannej z użyciem przeszłych stanów sieci Q-Network).

W doświadczeniach, prezentowanych w kolejnej sekcji, skorzystano z drobnej modyfikacji treningu z wykorzystaniem bufora pamięci, zaproponowej w [20], o nazwie *Combined Experience Replay – CER*. W tym wariancie, w każdej iteracji uczenia, do minipaczki przykładów uczących losowanych z bufora, dokładana jest zawsze świeżo zdobyta czwórka (S, A, R, S'). Dzięki temu, zapewniony jest udział każdego pozyskanego doświadczenia i to udział natychmiastowy. W oryginalnym wariancie, przy nieszczęśliwym losowaniu część próbek doświadczenia może nie zostać w ogóle wykorzystana.

Całość algorytmu DQN, w wersji która najlepiej sprawdziła się w problemie świata Wumpusa, a więc z *CER* oraz miękkim aktualizowaniem sieci *target*, przedstawia Alg. 3.

6.5 Agent DQN w świecie Wumpusa

6.5.1 Częściowa obserwacja

Problem częściowej obserwacji stanu Zgodnie z oryginalną specyfikacją świata Wumpusa, wiedza agenta o świecie, w którym się znajduje, daleka jest od wiedzy pełnej - nie ma on dostępu do rzeczywistego stanu środowiska S, a wiedzę gromadzi tylko na podstawie lokalnych obserwacji. Niemożliwe jest więc w tym wypadku uczenie strategii działania agenta w oparciu o rzeczywisty stan świata $\pi(a|\mathrm{s})$. Agent może opierać wybór swoich akcji jedynie na niepełnej wiedzy o S, jaką zdążył zgromadzić.

Tego typu środowiska można opisywać przy pomocy modelu częściowo obserwowalnego POMDP (ang. *Partially Observable Markov Decision Process*), które następnie można z powrotem wyrazić za pomocą MDP, w którym stan jest zastąpiony *przekonaniem* agenta o stanie. Ten formalizm nie będzie tu jednak użyteczny, gdyż w tego rodzaju modelach przekonanie o stanie wyrażane jest jako ciągły rozkład prawdopodobieństwa, przez co nie nadaje się do bezpośredniego zastosowania w kontekście omawianych algorytmów.

Algorithm 3: Algorytm DQN w wersji z dwiema modyfikacjami względem oryginału: techniką CER (ang. *Combined Experience Replay*) oraz z miękkim aktualizowaniem sieci *target* (uśrednianiem Polyaka).

Zainicjuj bufor pamięci D o pojemności N
Zainicjuj sieć $\mathcal{Q}$ wagami losowymi θ
Zainicjuj drugą sieć $\hat{\mathcal{Q}}$ wagami $\theta^- = \theta$
for *Epizod 1 do M* **do**
  Reset środowiska (inicjalizacja S)
  $t \leftarrow 0$
  Przetwórz $\mathbf{s} = \phi(S)$
  while S *nie jest stanem końcowym* **and** $t < T$ **do**
    Wybierz akcję A na podstawie $\mathbf{s}$ używając $\pi_{\mathcal{Q}}$
    (Opcjonalnie: zaktualizuj strategię $\pi_{\mathcal{Q}}$, np. zredukuj ϵ)
    Wykonaj akcję A, zaobserwuj R, S'
    Przetwórz $\mathbf{s}' = \phi(S')$
    Umieść czwórkę $d = (\mathbf{s}, A, R, \mathbf{s}')$ w D
    Utwórz mini-paczkę danych uczących złożoną z d oraz $B-1$ historycznych czwórek pobranych losowo z D
    Dla każdego przykładu j w mini-paczce, ustal y:

$$y_j = \begin{cases} r_j & \text{gdy s' jest terminalny} \\ r_j + \gamma \max_a \hat{\mathcal{Q}}\left(\mathbf{s}', a; \theta^-\right) & \text{w przeciwnym razie} \end{cases}$$

    Wykonaj krok optymalizacji parametrów θ na podstawie mini-paczki danych oraz funkcji straty (6.7)
    Zaktualizuj parametry sieci $\hat{\mathcal{Q}}$ uśrednianiem Polyaka
    $t \leftarrow t + 1$
  end
end

Jeśli wejściem sieci w algorytmie DQN nie może być faktyczny stan środowiska S, pozostaje przygotować możliwie pełną *reprezentację wiedzy* agenta o tym stanie – $\hat{S}$. Algorytm DQN będzie optymalizował strategię $\pi(a|\hat{s})$, niezmiennie starając się maksymalizować zwrot z podejmowanych akcji. W [10, 11], pomysłodawcy algorytmu DQN również napotkali problem częściowej obserwacji: zrzut ekranu gry Atari nie zawsze oddaje dynamikę tego, co akurat się w tej grze dzieje. Jako rozwiązanie tego problemu, zaproponowali prostą kumulację chwilowych obserwacji: na wejście sieci podawano złożenie czterech kolejnych zrzutów ekranu gry, każdy w odcieniach szarości, tworząc łącznie 4-kanałowy obraz. Tego typu sztuczne wejście, pozwalało już np. na oszacowanie kierunku i prędkości obiektów ruchomych, których nie da się odtworzyć z pojedynczego obrazu. Z kolei w [6], podejmującej ten sam problem gier Atari, kumulacja wiedzy pochodząca z kolejnych niepełnych obserwacji odbywała się wewnątrz modelu, gdyż jako sieć Q-Network użyto sieci rekurencyjnej – naturalnie przystosowanej do analizy sekwencji, w tym sekwencji obrazów.

Reprezentacja wiedzy agenta w świecie Wumpusa W świecie Wumpusa – podobnie – obserwacja z danej chwili nie jest wystarczająca, konieczne jest odpowiednie zagregowanie jej. Dopiero taki agregat, zawierający całość zdobytej wiedzy o stanie świata na daną chwilę, może zastać podany na wejście sieci neuronowej. W tym wypadku, agent opiera się na akumulacji informacji o środowisku pochodzących z jego zmysłów, w połączeniu ze śledzeniem akcji, jakie podjął i stanu inwentarza, jaki posiada. Zmysły dostarczają 5 bitów informacji dotyczących pola, na którym aktualnie znajduje się agent:

- czy wyczuwalny jest zapach Wumpusa;
- czy wyczuwalny jest podmuch świadczący o dole w sąsiedztwie;
- czy dostrzegalny jest błysk złota;
- czy agent odczuł uderzenie w ścianę;
- czy słyszano krzyk zranionego potwora.

Przyjęto, że te pierwsze cztery elementy agent odnotowuje na wyimaginowanej mapie: mapie, w której centrum zawsze znajduje się agent, a góra mapy zawsze odpowiada aktualnemu azymutowi obranemu przez agenta. Odpowiednie przesunięcie i obrót mapy jest możliwy, gdyż agent wie, jaką akcję ostatnio podjął, a środowisko w świecie Wumpusa jest deterministyczne (pomijając wariant *Poziom0-ND*, stworzony wyłącznie na potrzeby demonstracji prawidłowego działania Q-learningu), więc podjęta akcja jednoznacznie określa korektę ustawienia mapy. Mapa agenta to 5-kanałowy obraz 7×7 (specyfika informacji o uderzeniu w ścianę wymaga nawet obrazu 9×9, aby nie stracić informacji przy obrotach, ale na sieć wystarczy podać środek 7×7). Pierwsze 4 kanały służą do zapisu pierwszych czterech bitów obserwacji: 1 – w przypadku wystąpienia odczucia, -1 – w przypadku niewystąpienia. Piąty kanał służy agentowi do określenia swojej pozycji względem pozycji startowej – wejścia do jaskini. Kanał ten jest niezbędny, gdyż bez niego agent nie ma wiedzy o swoim położeniu i nie byłby w stanie wyróżnić pozycji startowej, a tylko z niej agent może z sukcesem opuścić jaskinię. Mapa inicjowana jest wszędzie wartościami 0, oprócz centralnego pola piątego kanału.

Piąty element obserwacji, ewentualny odgłos Wumpusa, nie potrzebuje umiejscowienia na mapie, ale także jest ważny: pozwala na zaktualizowanie wiedzy o ewentualnym unieszkodliwieniu potwora. Ta cecha, oraz dwie kolejne zawierające aktualne informacje o inwentarzu agenta, tj. czy dysponuje jeszcze strzałą, którą może wystrzelić oraz czy podniósł już złoto, wraz z zserializowaną mapą odczuć ($5 \times 7 \times 7$ cech), tworzy łącznie wektor wejściowy do trójwarstwowej sieci typu MLP (ang. *Multi Layer Perceptron*) posiadający długość $l = 248$ (zob. 6.3).

Alternatywnie, próbowano też użyć map bezwględnych o rozmiarze 4×4 i dodatkowymi kanałami przeznaczonymi na wystąpienie krzyku i stan inwentarza, a następnie użycie głębokiej sieci splotowej, tak jak w oryginalnej pracy o DQN. Wstępne eksperymenty wykazywały jednak znacznie wolniejsze tempo nauki w porównaniu z przedstawioną reprezentacją wiedzy i tradycyjną siecią MLP.

6.5.2 Rzadko występująca nagroda

Problem Sukcesem agenta w świecie smoka jest opuszczenie jaskini wraz ze zdobytym złotem. Każdy epizod, który kończy się inaczej ostatecznie jest porażką, niezależnie od tego, czy zakończył się po dwóch ruchach wpadnięciem do dołu, czy po dłuższej eksploracji jaskini, pokonując po drodze smoka i zdobywając złoto, ale nie zdążając wyjść z jaskini przed upływem maksymalnej liczby kroków w epizodzie. Przyjmijmy zatem, że w pierwotnej wersji środowiska agent otrzymuje pozytywną nagrodę, np. $R = 1000$ tylko wtedy, gdy osiągnął wyżej określony sukces, nagrodę ujemną, np. $R = -1000$, gdy zginął (niezależnie od przyczyny śmierci) oraz $R = 0$, w każdej innej sytuacji.

Przy tak określonej nagrodzie, uczenie agenta ze wzmocnieniem jest bardzo trudne i nieefektywne, ponieważ nagroda za sukces występuje niezwykle rzadko (problem rzadko występującej nagrody, ang. *sparse reward*): wybierający losowe akcje agent ma bardzo małą szansę przypadkowo ominąć wszystkie przeszkody, wejść do pomieszczenia ze złotem, podnieść je, wrócić do miejsca początkowego (znów unikając niebezpieczeństw) oraz opuścić jaskinię. Aby nauka była możliwa agent potrzebuje informacji zwrotnej – dodatniej czy ujemnej. Jeśli tej informacji nie ma, bądź jest, ale niezwykle rzadko, progres w nauce jest niemożliwy bądź minimalny. Przy nagrodach określonych jak wyżej, agent jest w stanie w miarę szybko nauczyć się działać w ten sposób, by nie ginąć (unikać kary $R = -1000$, która początkowo jest bardzo częsta), ale nie nauczy się upuszczać jaskini ze złotem, bo prawdopodobnie nigdy tego nie doświadczy. Przy tak sformułowanej nagrodzie, strategia agenta często kończyła się np. na kręceniu się w kółko w pomieszczeniu startowym lub uderzaniu w tymże pomieszczeniu w ścianę próbując przemieścić się w lewo lub w dół.

Kształtowanie nagrody Popularną metodą poprawy efektywności uczenia ze wzmocnieniem w przypadku rzadko występującej nagrody jest wykorzystanie wiedzy eksperckiej o problemie i modyfikacja funkcji nagrody w ten sposób, aby naprowadzić agenta na ścieżkę wiodącą do sukcesu, która bez tej pomocy byłaby trudna do odkrycia. Jest to tzw. kształtowanie nagrody (ang. *reward shaping*) [4, 17].

W przeprowadzonych eksperymentach przyjęto, że nagroda R dana agentowi po podjęciu akcji A w stanie S, będzie zawsze sumą dwóch składowych: 1) kosztu akcji A oraz 2) nagrody związanej ze zdarzeniem, które akcja A w stanie S wywołała. Konkretne, przyjęte arbitralnie, wartości kosztów akcji oraz nagród za zdarzenia, z którymi przeprowadzano końcowe eksperymenty, przedstawione są odpowiednio w Tab. 6.2 oraz Tab. 6.3. Szczególnie ważne jest pozytywne wzmacnianie samego podniesienia złota, odwiedzania nowych pomieszczeń, w których wcześniej agent jeszcze nie był (co jest zachętą do eksploracji) oraz unieszkodliwienie Wumpusa.

Podwójna strategia eksploracji Rzadkość sukcesu w świecie Wumpusa wynika z długości procesu decyzyjnego, w którym nie można popełnić żadnego

Tablica 6.2. Koszty akcji przyjęte w eksperymentach.

Akcja	**Koszt**
Ruch w przód	-0.1
Obrót w lewo	-5
Obrót w prawo	-5
Próba podniesienia złota	-20
Próba wystrzału z łuku	-20
Próba wyjścia z jaskini	-20

Tablica 6.3. Nagrody za poszczególne zdarzenia przyjęte w eksperymentach.

Zdarzenie	**Nagroda**
Wyjście z jaskini ze złotem	1000
Podniesienie złota	500
Śmierć agenta	-1000
Unieszkodliwienie Wumpusa	300
Zderzenie ze ścianą	-5
Wejście na nowe pole	50

poważnego błędu, aby go osiągnąć. Podczas eksperymentów zauważono, że strategia ϵ-zachłanna (Alg. 1) wraz z malejącą wartością ϵ, zgodnie z (6.6), umożliwia agentowi odpowiedni balans między eksploracją i eksploatacją wystarczający do nauczenia się bezpiecznego odnajdowania i podnoszenia złota, ale zajmuje to tyle kroków uczenia, że ϵ jest już bardzo małe (minimalna eksploracja) i agent nie jest w stanie odkryć właściwego zachowania przy powrocie tak, by kończyć epizody wyjściem z jaskini (agent musi spróbować podjąć akcję *wyjście z jaskini*, akurat na polu startowym, gdy już posiada złoto). Zwiększenie początkowej wartości ϵ czy zmiana tempa zmiany jego wartości (ϵ_{decay}) nie poprawiały sytuacji.

Autorską metodą poprawy eksploracji i podniesienia prawdopodobieństwa nauczenia agenta osiągania sukcesu, będącą kolejnym sposobem wykorzystania wiedzy dziedzinowej do ułatwienia zadania agentowi, jest zastosowanie *podwójnej* strategii ϵ-zachłannej. Przyjęto mianowicie, że są dwie wartości ϵ_1 oraz ϵ_2, pierwsza stosowana wtedy, gdy agent nie posiada jeszcze w inwentarzu złota, a druga, gdy złoto już podniósł. Obie wartości maleją zgodnie z (6.6), przy czym w danym kroku aktualizowana jest tylko ta z nich, która była użyta przy wyborze akcji. W ten sposób, na etapie uczenia, w którym agent potrafi już odszukiwać i podnosić złoto, wartość ϵ_1 jest już minimalna, a więc agent postępuje zachłannie, ale cały czas uczy się drogi powrotnej, korzystając z wciąż relatywnie wysokiej wartości ϵ_2.

Curriculum learning Ostatnim użytym sposobem przemycenia wiedzy eksperckiej do procesu uczenia agenta w świecie Wumpusa, jest stopniowanie poziomu trudności stawianego przed nim zadania, a więc zastosowanie metody *curriculum learning* [12]. W metodzie tej tworzy się uproszczone wersje zadania, które jest celem nauki, stopniując poziom ich trudności, tworząc spektrum zadań od stosunkowo łatwych aż do zadania docelowego. Następnie nauka agenta polega na sekwencyjnym opanowywaniu zadań od najprostszego do docelowego, zważając, by przechodzić do nauki zadania trudniejszego dopiero, gdy opanuje on zadanie prostsze.

Użycie *curriculum learning* w połączeniu z nauką algorytmem DQN polega na inicjalizacji sieci Q-Network wagami tejże sieci, wyuczonymi w prostszym wariancie zadania, zamiast wagami losowymi (linijka nr 2 w Alg.3. Można zauważyć, że *curriculum learning* to specyficzna forma znanego z innych dziedzin uczenia

maszynowego *transfer testingu*. Na potrzeby tego rodzaju uczenia, przygotowane zostały 4 poziomy trudności środowiska (poziomy od 1 do 4, zob. Tab. 6.1), różniące się między sobą jedynie liczbą dziur w jaskini.

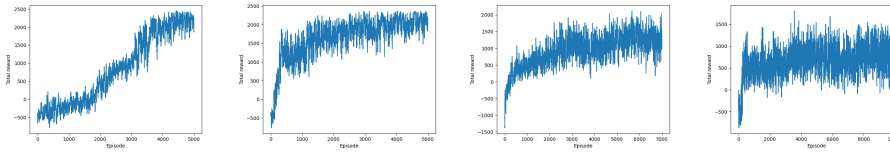
6.5.3 Wyniki

W Tab. 6.4 zebrano wszystkie wartości hiperparametrów, ustalone w drodze krótkiego przeszukiwania i przyjęte w końcowych eksperymentach.

Przebieg uczenia na poszczególnych poziomach trudności przedstawiają wykresy na Rys. 6.4, a Tab. 6.5 przedstawia ocenę jakości wyuczonych strategii działania.

Tablica 6.4. Przyjęte wartości hiperparametrów.

Hiperparametr	**Wartość**	**Hiperparametr**	**Wartość**
γ	0.99	M	5000-10000
ϵ_1, ϵ_2	1.0, 0.5	B	64
ϵ_{decay}	0.01	N	100000
ϵ_{min}	1e-5	$\lvert\mathbf{h}_1\rvert$	128
α	0.001	$\lvert\mathbf{h}_2\rvert$	64
T	50	τ	0.01



(a) Poziom 1 (b) Poziom 2 (c) Poziom 3 (d) Poziom 4

Rysunek 6.4. Średni zwrot z dziesięciu ostatnich epizodów w czasie uczenia agentów poszczególnych poziomów trudności świata Wumpusa

Zarówno po przebiegu procesów uczenia, jak i wynikach testów, widoczny jest wyraźny wzrost poziomu trudności wraz z dodawaniem liczby dziur do jaskini. W wariancie 1, agent nauczył się wygrywać w ponad 80% przypadków, co ciekawe zdobywając średnio powyżej 2000 punktów nagrody, co oznacza, że oprócz podniesienia i wyjścia ze złotem, agent nauczył się również odwiedzać większość pomieszczeń w jaskini oraz unieszkodliwiać Wumpusa. Z kolejnymi poziomami trudności, procent wygranych epizodów oraz średnia wartość zwrotu wyraźnie spada, choć należy podkreślić, że przeżywalność agenta utrzymuje się na bardzo wysokim poziomie. 41% epizodów, w których agent dotarł do złota w przypadku najtrudniejszym to wynik dobry, jeśli weźmie się pod uwagę to, że

4 niebezpieczeństwa losowo rozmieszczane na siatce 4×4 nierzadko uniemożliwiają bezpieczne, tj. bez podejmowania ryzyka, dotarcie do złota, a zdarza się też całkowite zablokowanie przez dziury pola ze skarbem.

W przypadku poziomu 3. i 4. procent wygranych epizodów jest około dwukrotnie niższy niż procent epizodów, w których agent podniósł złoto. Oznacza to, że nawet pomimo zastosowanych technik pomocy, w szczególności podwójnej strategii ϵ-zachłannej, agent nie był w stanie wystarczająco nauczyć się ścieżki powrotu i wyjścia z jaskini.

Tablica 6.5. Test agentów (1000 epizodów testowych), po ukończeniu nauki na danym poziomie.

Poziom	Przeżycie [%]	Podniesienie złota [%]	Wyjście z jaskini [%]	Średni zwrot
1	100.0	96.2	83.4	2080.52
2	99.8	91.5	62.4	1609.16
3	99.9	60.8	32.7	923.81
4	97.8	41.2	22.0	545.57

6.6 Podsumowanie

Niniejszy rozdział przedstawia dwa podstawowe algorytmy: tabelaryczny *Q*-learning, oraz algorytm DQN, stanowiących punkt wyjścia dla poznawania bardziej zaawansowanych metod klasycznego oraz głębokiego uczenia ze wzmocnieniem. Opis teoretyczny wzbogacono przykładem zastosowania wymienionych metod do rozwiązania problemu sterowania agentem w świecie Wumpusa. W sekcji 6.5 przedstawiono sposoby dostosowania algorytmu DQN, stworzonego z myślą o sterowaniu agentami w grach Atari na podstawie zrzutu ekranu gry, do rozwiązania gry logicznej: dobranie odpowiedniej reprezentacji wiedzy agenta oraz z odpowiednią architekturą sieci, kształtowanie nagrody, odpowiednia strategia eksploracji oraz *curriculum learning*. Wyniki, jakie udało się osiągnąć, można uznać za umiarkowany sukces. Prostsze warianty środowiska są rozwiązywane w 60-80% przypadków, agent praktycznie nie ginie, a w wariancie pełnym agent zwycięża w ok. 22% przypadków, ale ten wynik w znacznej mierze wynika z ostrożnej strategii: agent nie ginie w blisko 98% przypadków.

Aby jeszcze bardziej poprawić skuteczność uczenia ze wzmocnieniem w świecie Wumpusa (i innym podobnych problemach), w przyszłości planowane jest przetestowanie uczenia wykorzystującego niewielką liczbę demonstracji prawidłowego przejścia gry (a nawet pojedynczą, odwróconą demonstrację, jak w [1]). Dużym polem badawczym, którego eksploracja również pozostawiona została na przyszłość, jest wykorzystanie metod uczenia ze wzmocnieniem budujących model środowiska (ang. *model-based RL*). W szczególności, innowacyjne połączenie modeli głębokich z symbolicznymi modelami programowania w logice mogłoby bardzo dobrze sprawdzić się w świecie Wumpusa i problemach pokrewnych.

Bibliografia

1. Learning montezuma's revenge from a single demonstration. https://openai.com/blog/learning-montezumas-revenge-from-a-single-demonstration/, accessed: 2021-07-12.
2. Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al.: Solving rubik's cube with a robot hand. arXiv preprint arXiv:1910.07113 (2019).
3. Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al.: Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 (2019).
4. Grzes, M.: Reward shaping in episodic reinforcement learning (2017).
5. Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., Levine, S.: Learning to walk via deep reinforcement learning. arXiv preprint arXiv:1812.11103 (2018).
6. Hausknecht, M., Stone, P.: Deep recurrent q-learning for partially observable mdps. In: 2015 aaai fall symposium series (2015).
7. Jędruch, W.: Sztuczna inteligencja - materiały do wykładu, nieopublikowane.
8. Li, A.A., Lu, Z., Miao, C.: Revisiting prioritized experience replay: A value perspective. arXiv preprint arXiv:2102.03261 (2021).
9. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: Bengio, Y., LeCun, Y. (eds.) ICLR (2016).
10. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013).
11. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015).
12. Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M.E., Stone, P.: Curriculum learning for reinforcement learning domains: A framework and survey. arXiv preprint arXiv:2003.04960 (2020).
13. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson Education, 2 edn. (2003).
14. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015).
15. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al.: Mastering atari, go, chess and shogi by planning with a learned model. Nature **588**(7839), 604–609 (2020).
16. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018).

17. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018).
18. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al.: Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019).
19. Watkins, C.J.C.H.: Learning from delayed rewards (1989).
20. Zhang, S., Sutton, R.S.: A deeper look at experience replay. arXiv preprint arXiv:1712.01275 (2017).

7. Sieci neuronowe oparte na prawach fizyki

Bartłomiej Borzyszkowski, Karol Damaszke, Jakub Romankiewicz,
Marcin Swiniarski, Marek Moszyński

Wydział Elektroniki Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
marmo@eti.pg.edu.pl

Streszczenie
Wiele fizycznie nieuzasadnionych sieci neuronowych, mimo zadowalającej wydajności, generuje sprzeczności z logiką i prowadzi do rozbieżności wyników z rzeczywistością. Jedną z metod poprawy funkcjonowania typowego modelu typu "black-box" na etapie uczenia, jest rozszerzenie jego funkcji kosztu o zależność bezpośrednio inspirowaną wzorem fizycznym. Niniejszy rozdział wyjaśnia koncepcję budowy sieci neuronowych opartych na prawach fizyki, zawiera przegląd zaproponowanych rozwiązań w tej dziedzinie oraz opisuje możliwości implementacji funkcji strat wykorzystujących wzory fizyczne. Ponadto przedstawione badania pokazują, że przewidywania algorytmów inspirowanych przez fizykę mogą być nie tylko optymalne, ale również naukowo spójne z równaniami dziedzinowymi. Ostatecznie wykorzystanie wiedzy naukowej zawartej w dostosowanych funkcjach kosztów pokazuje, że metodyka ta gwarantuje wyniki spójne z prawami fizyki, a także lepszą generalizację w porównaniu z klasycznymi sieciami neuronowymi.

Słowa kluczowe: sztuczna inteligencja, sztuczne sieci neuronowe, wyjaśnialna sztuczna inteligencja, sieci neuronowe oparte na prawach fizyki

7.1 Wprowadzenie

Ciągły postęp w dziedzinie przetwarzania danych prowadzi obecnie do dynamicznego rozwoju technologicznego w wielu gałęziach przemysłu i nauki, wprowadzając najnowocześniejsze rozwiązania dzięki ulepszonym algorytmom, jednostkom obliczeniowym, a także zbiorom danych. Technologie uczenia głębokiego są z pewnością jednym z kluczowych kierunków innowacji, przyczyniając się do nowych odkryć w kolejnych zastosowaniach naukowych [1].

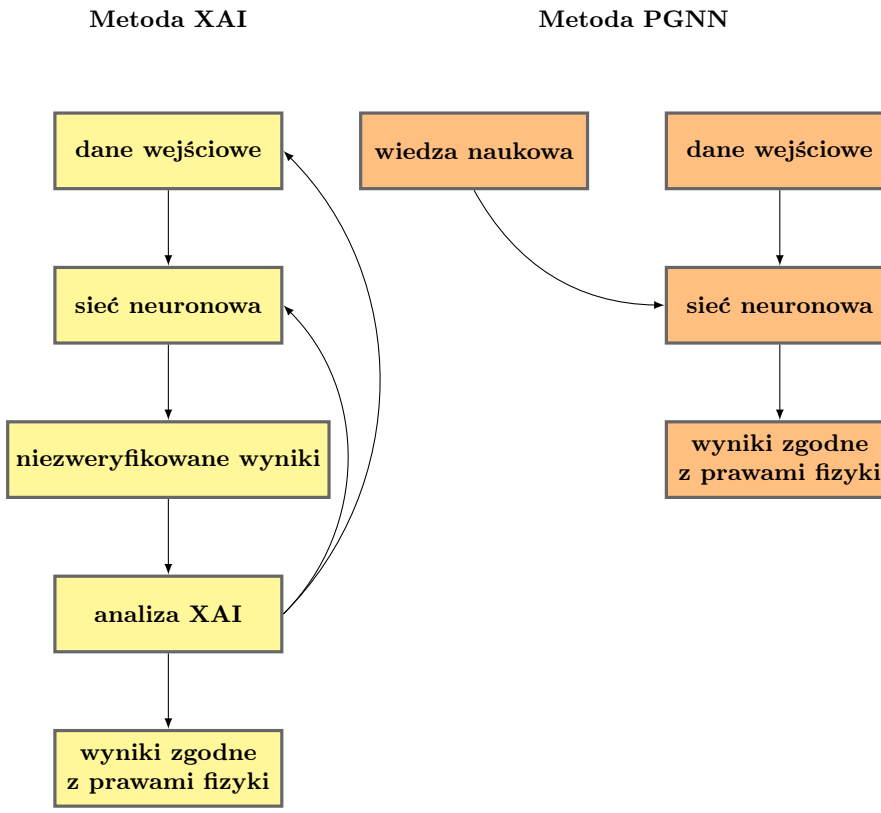
Niestety, wciąż wyróżniamy wiele dziedzin, w których klasyczne techniki sztucznej inteligencji - AI (ang. *Artificial Intelligence*), takie jak głębokie sieci neuronowe DNN (ang. *Deep Neural Networks*), nie osiągają oczekiwanej generalizacji. Do realizacji niektórych zadań niewystarczające okazują się również modele impulsowych sieci neuronowych SNN (ang. *Spiking Neural Networks*), które lepiej odzwierciedlają funkcjonowanie biologicznych neuronów, ale wiążą się z trudnościami implementacyjnymi [2]. Ponadto zdarzają się przypadki, w których proces uczenia sieci neuronowych lub ich parametry nie są optymalne,

aby zagwarantować osiągnięcie satysfakcjonujących wyników [3]. Pewne trudności napotyka się również podczas dostarczania danych, które są wprowadzane do modelu podczas uczenia, walidacji i testowania. Stworzenie dużego, spójnego i obiektywnego zbioru danych jest nie tylko trudne, ale także kosztowne i czasochłonne. Dużym wyzwaniem jest niwelowanie bardzo często pojawiających się problemów związanych z brakiem dostatecznej liczby informacji, jakością danych czy zakłóceniami [4]. Rozwiązaniem może być augmentacja danych [5] lub różne metody ewaluacji, takie jak walidacja krzyżowa[6], które w rzeczywistości pozwalają zminimalizować problemy, ale nie rozwiązują ich całkowicie. Jednym z podstawowych ograniczeń typowych modeli typu "black-box" jest ich wyraźna zależność od danych dostarczanych w procesie uczenia [7]. Na przykład, typowa sztuczna sieć neuronowa dla problemu nadzorowanego uczenia może być tylko tak wiarygodna, jak wiarygodne są dane, którymi jest zasilana. Oznacza to, że nawet jeśli model osiągnie satysfakcjonującą dokładność na zbiorze testowym, może być nadal bezużyteczny w rzeczywistych scenariuszach, ze względu na istotne różnice występujące w danych testowych i rzeczywistych.

Bardzo trudno jest uzyskać precyzyjne informacje o czynnikach, na podstawie których sieć neuronowa wyciąga wnioski w procesie decyzyjnym, podczas swojego treningu. Typowe modele typu "black-box" dostosowują wagi między neuronami, minimalizując w ten sposób funkcję kosztu, aby osiągnąć najlepsze wyniki na danych uczących. Niemniej jednak, często decyzje sztucznej inteligencji opierają się na zupełnie innych czynnikach niż oczekiwano [8]. Jeśli wyniki są ponad wszelką wątpliwość poprawne, może to nie mieć znaczenia, ale problem pojawia się, gdy przewidywanie sieci neuronowej jest sprzeczne z podstawowymi prawami fizyki lub logiki. Wytłumaczalna sztuczna inteligencja (ang. *Explainable AI - XAI*) to dziedzina, która bada proces decyzyjny sieci neuronowej po jej wytrenowaniu [9]. Pozwala na zastosowanie kompleksowego zestawu metod, takich jak debugowanie, wizualizacja, ocena, testowanie, wyjaśnianie decyzji, czy też monitorowanie wydajności i wiarygodności. Dzięki tego typu technikom jesteśmy w stanie dość dokładnie określić, na podstawie jakich czynników sieć podejmowała decyzje, a w przypadku pewnych niezgodności, zmodyfikować zbiór danych i ponownie wytrenować model [10]. Takie podejście z pewnością niesie ze sobą wiele korzyści, ale XAI nie można interpretować jako jakiejkolwiek formy prewencyjnej ingerencji w strukturę modelu, która całkowicie eliminowałaby sprzeczności z prawami fizyki. Jest to raczej rozwiązanie umożliwiające dostarczenie informacji o czynnikach dedukcyjnych, a późniejsze zwalczanie skutków nieprawidłowego treningu modelu musi odbywać się niezależnie. Pomimo wielu zalet, taka analiza często nie jest wykonywana ze względu na brak świadomości programistów, a także jej złożoność i czasochłonność.

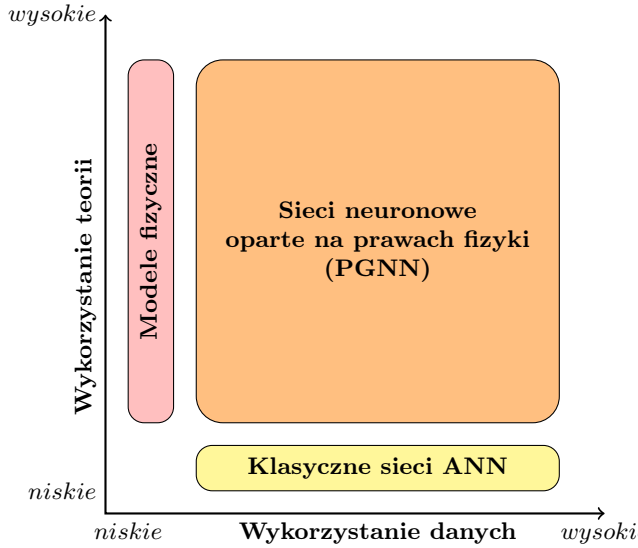
Rozwiązaniem eliminującym lub znacząco redukującym opisane problemy już w procesie uczenia jest uzależnienie funkcji kosztu modelu od formuły naukowej. Podejście to zostało wykorzystane w tak zwanych sieciach neuronowych opartych na prawach fizyki PGNN (ang. *Physics Guided Neural Networks*), które łączą możliwości oferowane przez klasyczne sieci neuronowe z wiedzą opartą na prawach fizyki. W tym przypadku można nie tylko zapewnić, że wyniki dostar-

czane przez sieć neuronową będą zgodne z określoną zależnością (na podstawie zadanego równania), ale także zapobiec potencjalnym sprzecznościom już na etapie uczenia. Możliwe jest również uzupełnienie sieci PGNN o dodatkową analizę zwrotną, co łączyłoby ją z wytłumaczalną sztuczną inteligencję XAI. Implementacja mechanizmu sprzężenia zwrotnego pozwala na sprawdzenie zgodności wyników, jak również sprzyja dokładniejszej eliminacji potencjalnych niezgodności. W tym przypadku sieć PGNN można traktować jako typową sieć neuronową, która podlega metodom XAI. Różnice w interpretacji wyników w przypadku podejścia XAI i PGNN pokazano na rysunku 7.1.



Rysunek 7.1. Dwa podejścia, które pozwalają zagwarantować naukową spójność przewidywań w architekturach z głębokim uczeniem DNN: wytłumaczalna sztuczna inteligencja XAI - dane wyjściowe sieci neuronowej są w fazie oceny przekazywane na wejścia wcześniejszych bloków, co pozwala na dalsze udoskonalenia przygotowywanego modelu (po lewej); sieci neuronowe oparte na prawach fizyki – połączenie sieci neuronowej z wiedza naukową, która pozwala wyjaśnić proces decyzyjny już w fazie uczenia (po prawej). W przypadku wymagających zadań możliwe jest również połączenie obu podejść

Można wyróżnić dwa typy modeli opartych na prawach fizyki: (1) równania, wzory i reguły, które implikują relacje między zmiennymi, a tym samym pozwalają opisać praktyczne scenariusze w teorii; (2) modele numeryczne złożonych układów fizycznych, które dokładnie symulują zjawiska w świecie rzeczywistym. Chociaż modele oparte na prawach fizyki są zdolne do poprawnej interpretacji relacji w kategoriach naukowych, mają jednak ograniczone możliwości interpretacji danych. Tym samym pewien kompromis pomiędzy sieciami neuronowymi, które wykorzystują przetwarzanie danych, a modelami fizycznymi, które interpretują relacje fizyczne, jest bardzo pożądanym kierunkiem badań, pozwalającym na połączenie zalet obu rozwiązań. Różnica między modelami opartymi na prawach fizyki, a sieciami neuronowymi typu "black-box" została schematycznie przedstawiona na rysunku 7.2. Oba podejścia prezentują dwa skrajne sposoby odkrywania wiedzy: polegające wyłącznie na wiedzy naukowej lub tylko na danych. Ich połączenie przedstawiono jako przecięcie obu rozwiązań i umieszczono w centralnej części wykresu.



Rysunek 7.2. Schematyczne przedstawienie sieci PGNN w kontekście innych podejść do odkrywania wiedzy, które wykorzystują prawa fizyki lub dane. Oś pozioma reprezentuje wykorzystanie danych a oś pionowa - wykorzystanie wiedzy naukowej. Tym samym, sieci PGNN łączą w sobie zalety modeli fizycznych oraz klasycznych sieci ANN, dzięki jednoczesnemu wykorzystywaniu danych oraz wiedzy teoretycznej

7.2 Elementy teorii

Za początki badań nad nowoczesnymi algorytmami sztucznej inteligencji uznaje się drugą połowę XX wieku. Już w roku 1950 Alan Turing zaproponował test zdolności maszyny do wykazywania inteligentnego zachowania równoważnego lub nieodróżnialnego od zachowania człowieka [11]. W 1955 roku po raz pierwszy użyto terminu „sztuczna inteligencja" z definicją zaproponowaną przez Johna McCarthy'ego: „nauka i inżynieria wytwarzania inteligentnych maszyn" [12]. Od tego czasu obserwujemy nowe zastosowania uczenia maszynowego w wielu dziedzinami nauki, co staje się coraz bardziej popularne [13]. Biorąc pod uwagę okres rozwoju AI, można stwierdzić, że koncept PGNN jest stosunkowo nową koncepcją, wdrażaną sukcesywnie na przestrzeni ostatnich kilku lat. Jedną z fundamentalnych prac w tej dziedzinie przedstawili w 2018 r. A. Karpatne i in. w swoim artykule „Physics-guided Neural Networks (PGNN): an application in lake temperature modeling" [14]. Od tego czasu połączenie możliwości sieci neuronowych z wiedzą naukową staje się coraz bardziej popularne.

Analizując zastosowania sieci PGNN warto podkreślić, że ich funkcjonowanie musi wynikać z powszechnie znanych praw. Modyfikacja modelu w celu zapewnienia jego fizycznej zgodności opiera się na wykorzystaniu już istniejącej zależności fizycznej, która jest zintegrowana ze wzorem funkcji kosztu, co przedstawiono w sekcji 7.3.1. Tym samym zastosowanie tego rozwiązania ogranicza się do problemów o jasno określonych podstawach naukowych, które można uzasadnić prawami fizyki. W związku z tym sieci PGNN znajdują liczne zastosowania w analizie przestrzennej, gdzie rozszerzają możliwości podstawowych modeli fizycznych. Kompleksowy przegląd rozwiązań w zakresie integracji modelowania opartego na prawach fizyki z uczeniem maszynowym został przedstawiony w ankiecie przeprowadzonej przez J. Willarda i in. [15].

Ze względu na sformalizowanie ram PGNN i wprowadzenie modeli hybrydowych (sekcja 7.3.2) za jedną z głównych inspiracji w tej dziedzinie można uznać wspomniany powyżej przykład modelowania temperatury jeziora. Wyniki tej pracy zostały rozszerzone przez autorów o modyfikacje struktury ANN poprzez wprowadzenie koncepcji PGNN w rekurencyjnych sieciach neuronowych [16], ze szczególnym uwzględnieniem warstw LSTM [17]. Kolejne ciekawe eksperymenty z architekturą modeli przedstawili Y. Yang i in. [18] i R. Singh i in. [19] którzy pokazują możliwości opartych na fizyce głębokich modeli generatywnych. Ponadto L. Wang i in. [20] używają opartych na prawach fizyki autoenkoderów do szacowania stanów systemu elektroenergetycznego. Te przykłady pokazują, że możliwości naukowo inspirowanych funkcji kosztu nie są ograniczone do żadnej konkretnej architektury sieci neuronowej, natomiast mogą funkcjonować w modelach o różnej złożoności.

Zastosowania sieci PGNN w problemach przestrzennych mogą występować wszędzie tam, gdzie występuje określona zależność fizyczna. Oprócz wspomnianych wcześniej przykładów warto przytoczyć pracę N. Muralidhara i in. o przewidywaniu siły oporu na zawiesiny cząstek w poruszających się płynach [21]. Co więcej, sieci PGNN zostały przebadane w kilku nowych zastosowaniach, takich jak szacowanie mocy farm wiatrowych [22], przewidywanie przepływu turbulent-

nego [23], prognozowanie powodzi [24] czy ptychografia Fouriera [25]. Istotne okazać mogą się również zastosowania PGNN w dziedzinie medycyny i bioinformatyki, m.in. do przetwarzania obrazu tomograficznego [26] lub odwzorowania mechanizmów aktywacji serca [27].

Oprócz praktycznych rozwiązań, testowanych na rzeczywistych danych, przeprowadzono powiązane prace na wygenerowanych przykładach, wykorzystując różnego rodzaju wzory. M. Raissi i in. przedstawiają treningi PGNN do rozwiązywania równań różniczkowych cząstkowych (PDE), takich jak równania Shrödingera i Allen-Cahna [28, 29]. Ponadto Z. Fang i in. prezentują liczne podejścia do rozwiązywania PDE na powierzchniach, z uwzględnieniem powierzchni trójwymiarowych [30]. Analizując te przykłady można dojść do wniosku, że technika ta posiada potencjał do uzyskiwania rozwiązań teoretycznych, które mogą być powszechnie stosowane w praktyce lub rozszerzone na bardziej złożone problemy.

7.3 Koncepcja sieci neuronowych opartych na prawach fizyki

W tej sekcji opisano techniczną ideę sieci neuronowych opartych na prawach fizyki i wyjaśniono wykorzystanie wiedzy naukowej w implementacji ich funkcji kosztu. Ponadto omówiono koncepcję hybrydowych sieci PGNN, które łączą modele oparte na fizyce z algorytmami uczenia maszynowego.

7.3.1 Funkcja strat oparta na prawie fizycznym

Podstawowa zasada działania sieci PGNN jest bardzo prosta na poziomie koncepcyjnym. Głównym założeniem jest wprowadzenie do funkcji kosztu wzoru fizycznego, tak aby w procesie wnioskowania można było wykorzystać znane prawa teoretyczne. Funkcja kosztu klasycznej sztucznej sieci neuronowej może zostać przestawiona za pomocą wzoru (7.1).

$$\underset{f}{arg\,min} \quad \underbrace{Loss(\hat{Y}, Y)}_{\text{Typowa funkcja kosztu}} \tag{7.1}$$

W tym przypadku oczekiwany wynik (tzw. prawda podstawowa) jest porównywany z uzyskanymi rezultatami modelu, tworząc tzw. funkcję kosztu, którą można przedstawić różnymi równaniami w zależności od rodzaju problemu. Błąd kwadratowy, błąd bezwzględny czy entropia wzajemna to tylko kilka przykładów najbardziej typowych funkcji wykorzystywanych w tym przypadku [31]. Niezależnie od zastosowanej funkcji, celem algorytmu jest minimalizacja kosztu tak, aby zapewnić jak największą skuteczność sieci neuronowej poprzez aktualizację wag modelu, na przykład z wykorzystaniem algorytmu stochastycznego gradientu (SGD) [32].

Standardowe podejście do uczenia modelu PGNN jest bardzo podobne i opiera się na tych samych zasadach. Nie zmienia się też architektura sieci neu-

ronowej. Jedyną różnicą jest rozszerzenie funkcji kosztu o powszechnie znany wzór fizyczny (tzw. niespójność fizyczną), jak pokazano we wzorze (7.2).

$$\underset{f}{arg\,min} \quad \underbrace{Loss(\hat{Y}, Y) + \lambda R(f)}_{\text{Typowa funkcja kosztu}} + \underbrace{\lambda_{PHY}\ Loss.PHY(\hat{Y})}_{\text{Niespójność fizyczna}} \tag{7.2}$$

W tym przypadku λ_{PHY} jest hiperparametrem wybranym przez naukowca, który decyduje o poziomie wpływu wzoru fizycznego na trening sieci neuronowej. Parametr ten ma kluczowe znaczenie dla generowanych wyników, ponieważ jego odpowiedni dobór pozwala na zwiększenie rzetelności, uniwersalności i zgodności wyników z prawami fizyki. Z drugiej strony jego nieprawidłowe ustawienie (zbyt wysoka wartość) może prowadzić do drastycznego pogorszenia efektywności sieci neuronowej, której wyniki będą zbytnio oparte na wzorze fizycznym, przez co model nie będzie w stanie wykorzystać potencjału informacji zawartych w danych treningowych. Z drugiej strony, zbyt mała wartość parametru λ_{PHY} może spowodować bardzo mały wpływ wzoru, a tym samym sieć przyjmie postać klasycznego modelu ANN.

Koncepcja funkcji kosztu dla modeli PGNN może wydawać się stosunkowo prosta, ale jej wpływ na generowane wyniki jest często znaczący. Kluczowym aspektem w tym przypadku jest odpowiedni dobór wzoru fizycznego. Jest to zwykle zadanie trudne, ponieważ wymaga znajomości konkretnej dziedziny nauki, w której model ma być wykorzystany. Ponadto konkretny wzór musi łączyć w sobie parametry, które są dostępne w zbiorze danych użytym do rozwiązania problemu. Niestety jest to główna wada sieci PGNN, która uniemożliwia ich wykorzystanie w niektórych problemach i narzuca użytkownikowi dobrą znajomość dziedziny zastosowania. Z tego powodu fizycznie inspirowane sieci neuronowe charakteryzują się raczej wąskim spektrum zastosowań badawczych. Niemniej jednak ich cechy mogą poprawić proces treningu sieci neuronowych, co może prowadzić do dużej poprawy w osiąganych wynikach.

7.3.2 Modele hybrydowe

Dużo łatwiejsza do zastosowania i częściej implementowana jest koncepcja modeli hybrydowych. Modele te również łączą prawa fizyki z możliwościami wykorzystania informacji zawartych w danych, jednak robią to w nieco inny sposób. W tym przypadku wykorzystywane są klasyczne sztuczne sieci neuronowe bez wprowadzania zmian w ich strukturze czy też funkcji kosztu. Różnica polega na łączeniu danych treningowych z wynikami czysto fizycznego modelu. Tego typu modele oparte na prawach fizyki stanowią opis pewnego rzeczywistego zjawiska fizycznego. Dzięki modelowaniu matematycznemu są w stanie w przybliżeniu odwzorować rzeczywiste zachowanie danego procesu lub obiektu. Prostym sposobem na połączenie modeli opartych na fizyce z sieciami neuronowymi jest wykorzystanie symulowanych wyników modelu jako kolejnego wejścia do sieci neuronowej, jak przedstawiono we wzorze (7.3).

$$f_{HPD} : \mathbf{X} = [\mathbf{D}, Y_{PHY}] \rightarrow Y, \tag{7.3}$$

W tej konfiguracji można korzystać ze spójnych wyników wygenerowanych przez model fizyczny, a także innych przesłanek zawartych w treningowym zbiorze danych. Połączenie tych informacji podawane jest na wejście sieci neuronowej, zmniejszając w ten sposób szanse wystąpienia potencjalnych sprzeczności w wynikach.

Należy podkreślić, że modele oparte na prawach fizyki są bardzo często trudne do uzyskania lub mogą dostarczać niepełną reprezentację obiektów rzeczywistych z powodu uproszczonych lub brakujących relacji fizycznych. Ponadto zwykle modele te wymagają kalibracji, co jest procesem czasochłonnym i nieefektywnym. Warto pamiętać jednak o ich użyteczności w połączeniu z algorytmami sztucznej inteligencji, co pozwala na tworzenie hybrydowych modeli opartych na prawach fizyki.

7.4 Wnioski

Dzięki koncepcji PGNN wykorzystanej w sieciach neuronowych, wyniki algorytmów mogą okazać się nie tylko optymalne, ale także spójne z prawami fizyki. Na podstawie przykładów z przedstawionych publikacji jesteśmy w stanie ocenić, że w wybranych zastosowaniach połączenie modeli opartych o dane z wykorzystaniem równań dziedzinowych daje pozytywne rezultaty pod względem dokładności. Co więcej, takie rozwiązanie pozwala jednocześnie wyjaśnić proces dedukcji algorytmu, a tym samym wyeliminować potencjalne sprzeczności z logiką lub nauką, często występujące w klasycznych sieciach neuronowych typu "black-box".

Podstawowym celem niniejszej publikacji było zwrócenie uwagi na istotność koncepcji PGNN, która wciąż jest tematem niszowym. Zastosowania tej techniki można poszerzyć o szereg teoretycznych i praktycznych przykładów, takich jak zaprezentowane w sekcji 7.2. Ponadto wykorzystywanie architektur PGNN może okazać się bardzo przydatne do rozwiązywania przestrzennych równań różniczkowych, których wzory mogłyby być zawarte w funkcji kosztu sieci neuronowych. Rozwiązania tego typu problemów zostały pokazane na kilku przykładach zaprezentowanych przez M. Raissi i in. [28, 29], jednak dalsze prace w tym kierunku zarówno po stronie rozwiązań teoretycznych, jak i ich docelowych wdrożeń wydają się bardzo obiecujące.

Rozwój sieci PGNN to stosunkowo nowa gałąź sztucznej inteligencji, biorąc pod uwagę etapy rozwoju tej dziedziny. Niemniej jednak coraz popularniejsze stają się kolejne artykuły i nowe zastosowania funkcji kosztów opartych na fizyce. Niniejszy rozdział ma na celu popularyzację sieci PGNN oraz wyjaśnienie na przykładach teoretycznych, jak wykorzystać wiedzę naukową zawartą w funkcjach kosztów. Mamy nadzieję, że ten wkład pobudzi przyszłe prace nad teoretycznymi aspektami PGNN, a także zastosowaniami takich algorytmów w nowych dziedzinach nauki.

Bibliografia

1. R. Vinuesa et al., "The role of artificial intelligence in achieving the Sustainable Development Goals,"*Nat. Commun.*, vol. 11, no. 1, pp. 1-10, 2020.
2. M. Grochowski, A. Kwasigroch, and A. Mikołajczyk, Ślected technical issues of deep neural networks for image classification purposes,"*Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 67, no. 2, 2019.
3. T. Poggio and Q. Liao, "Theory II: Deep learning and optimization,"*Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 66, no. 6, 2018.
4. A. Lüdeling, M. Walter, E. Kroymann, and P. Adolphs, "Multi-level error annotation in learner corpora,"*Proc. Corpus Linguistics Conf.*, vol. 1, pp. 14-17, 2005.
5. A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem,"*Proc. Int. Interdiscipl. PhD Workshop (IIPhDW)*, pp. 117-122, 2018.
6. A. W. Moore and M. S. Lee, Ęfficient algorithms for minimizing cross validation error,"*Proc. 11th Int'l Conf. Machine Learning*, pp. 190-198, 1994.
7. J. M. Benitez, J. L. Castro and I. Requena, Ąre artificial neural networks black boxes?,"*IEEE Trans. Neural Networks*, vol. 8, pp. 1156-1164, 1997.
8. T. Hagendorff, "The ethics of AI ethics: An evaluation of guidelines,"*Minds Mach.*, vol. 30, pp. 99-120, 2020.
9. T. Miller, P. Howe and L. Sonenberg, Ęxplainable AI: Beware of inmates running the asylum,"*Proc. IJCAI Workshop Explainable AI (XAI)*, pp. 36-42, 2017.
10. A. Rai, Ęxplainable AI: from black box to glass box,"*Journal of the Academy of Marketing Science*, vol. 48, pp. 137-141, 2020.
11. R. French, Śubcognition and the limits of the Turing test,"*Mind*, vol. 99, no. 393, pp. 53-65, 1990.
12. J. McCarthy. "What is artificial intelligence?,"1998.
13. I. Rojek, M. Macko, D. Mikołajewski, M. Saga, and T. Burczyński, "Modern methods in the field of machine modelling and simulation as a research and practical issue related to industry 4.0,"*Bull. Pol. Acad. Sci. Tech. Sci.*, vol. 69, no. 2, 2021.
14. A. Karpatne, W. Watkins, J. Read, and V. Kumar, "Physics-guided neural networks (PGNN): An application in lake temperature modeling,"2017, [online], Available: http://arxiv.org/abs/1710.11431.
15. J. Willard et al., "Integrating Physics-Based Modeling with Machine Learning: A Survey,"2020, [online], Available: http://arxiv.org/abs/2003.04919.
16. X. Jia et al. "Physics guided RNNs for modeling dynamical systems: A case study in simulating lake temperature profiles,"*Proc. SIAM Int. Conf. Data Mining*, pp. 558-566, 2019.
17. A. Daw et al., "Physics-Guided Architecture (PGA) of neural networks for quantifying uncertainty in lake temperature modeling,"*Proc. SIAM Int. Conf. Data Mining*, pp. 532-540, 2020.
18. Y. Yang and P. Perdikaris, "Physics-informed deep generative models,"2018, [online], Available: http://arxiv.org/abs/1812.03511.

19. R. Singh, V. Shah, B. Pokuri, and S. Sarkar, "Physics-aware deep generative models for creating synthetic microstructures,"2018, [online], Available: http://arxiv.org/abs/1811.09669.
20. L. Wang, Q. Zhou and S. Jin, "Physics-guided deep learning for power system state estimation,"*J. Mod. Power Syst. Clean Energy*, vol. 8, no. 4, pp. 607-615, 2020.
21. N. Muralidhar et al., "Physics-guided design and learning of neural networks for predicting drag force on particle suspensions in moving fluids,"2019, [online], Available: http://arxiv.org/abs/1911.04240.
22. J. Park and J. Park, "Physics-induced graph neural network: An application to wind-farm power estimation,"*Energy*, vol. 187, pp. 115883, 2019.
23. R. Wang, K. Kashinath, M. Mustafa, A. Albert and R. Yu, "Towards physics-informed deep learning for turbulent flow prediction,"*Proc. 26th SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pp. 1457-1466. 2020.
24. T. Yang et al., Ęvaluation and machine learning improvement of global hydrological model-based flood simulations,"*Environ. Res. Lett.*, vol. 14, no. 11, pp. 114027, 2019.
25. Y. Zhang et al., "Pgnn: Physics-guided neural network for fourier ptychographic microscopy,"2019, [online], Available: http://arxiv.org/abs/1909.08869.
26. M. G. Poirot et al., "Physics-informed deep learning for dual-energy computed tomography image processing,"*Sci. Rep.*, vol. 9, no. 1, 2019.
27. F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, E. Kuhl, "Physics-informed neural networks for cardiac activation mapping."*Front. Phys.*, vol. 8, no. 42, 2020.
28. M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations,"2017, [online], Available: http://arxiv.org/abs/1711.10561.
29. M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part II): Data-driven solutions of nonlinear partial differential equations,"2017, [online], Available: http://arxiv.org/abs/1711.10566.
30. Z. Fang and J. Zhan Ą physics-informed neural network framework for PDEs on 3D surfaces: Time independent problems,"*IEEE Access*, vol. 8, pp. 26328–26335, 2019.
31. K. Janocha, and W. M. Czarnecki, Ón loss functions for deep neural networks in classification,"2017, [online], Available: http://arxiv.org/abs/1702.05659.
32. L. Bottou, Śtochastic gradient descent tricks,"in *Neural networks: Tricks of the trade*, Berlin, Heidelberg: Springer 2012, pp. 421-436.

8. Klasyfikator Adaboost w detekcji i rozpoznawaniu obiektów graficznych

Jerzy Dembski

Katedra Inteligentnych Systemów Interaktywnych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
jerdembs@pg.edu.pl

Streszczenie
W pracy opisano metodę Adaboost w zastosowaniu do detekcji obiektów graficznych, takich jak twarze lub rozpoznawania np. osób na podstawie obrazu twarzy. Przedstawiono podstawy algorytm, wersję kaskadową, schemat przepływu danych i sterowania w zadaniu detekcji twarzy oraz sposoby adaptacji tej metody do problemów wieloklasowych. Opisano również zbiory cech obrazów, takie jak HAAR, LBP czy HOG stosowane w zadaniach detekcji i rozpoznawania obiektów graficznych.

Słowa kluczowe: Adaboost, klasyfikacja danych, detekcja twarzy

8.1 Wprowadzenie

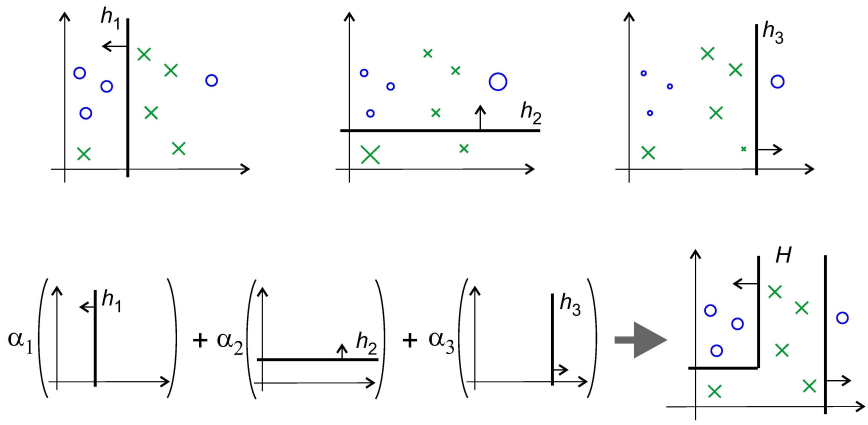
Metoda Adaboost (Adaptive Boosting [10]) jest rozwinięciem idei komitetów klasyfikatorów, w którym decyzja podejmowana jest poprzez głosowanie. Pierwszą ideą samej metody Adaboost jest ważenie wpływu decyzji z klasyfikatorów składowych komitetu i ich dobór w określonej kolejności, tak by kolejne klasyfikatory uzupełniały niedoskonałości poprzednich klasyfikatorów. Drugą ideą jest ważenie przykładów uczących, tak by przy doborze klasyfikatorów w większym stopniu brać pod uwagę przykłady, z klasyfikacją których gorzej radziły sobie poprzednio dobrane klasyfikatory. Metoda Adaboost jest najczęściej stosowana wtedy, gdy ważna jest szybkość działania klasyfikatora, co jest związane z małą złożonością obliczeniową klasyfikatorów należących do sekwencji. Z tych względów takie klasyfikatory składowe nazywane są słabymi klasyfikatorami, a cały klasyfikator Adaboost reprezentowany ważoną sumą słabych klasyfikatorów i wartością progową nazywany jest silnym klasyfikatorem. Kosztem zaledwie kilkunastu lub kilkudziesięciu operacji arytmetycznych możliwe jest sprawdzenie, czy na podanym obrazie lub oknie znajduje się poszukiwany obiekt. Metoda Adaboost jest więc ciągle stosowana w urządzeniach mobilnych oraz w fotografii cyfrowej ze względu na niewielkie wymagania pamięciowe i ze względu na niewielką czasową złożoność obliczeń w porównaniu do modeli neuronowych, które wymagają często tysięcy, a nawet milionów operacji arytmetycznych, w tym głównie mnożeń przy analizie pojedynczego okna.

Złożoność czasowa procesu uczenia, który opisano w rozdz. 8.2, jest natomiast najczęściej odwrotnie proporcjonalna do złożoności czasowej procesu klasyfikacji. Im dokładniejszy jest proces uczenia i z wykorzystaniem większych zbiorów cech, tym lepszy klasyfikator uzyskiwany jest w jego wyniku pod względem złożoności czasowej i uogólniania. Złożoności czasowe uczenia modeli Adaboost i modeli neuronowych dla tych samych problemów są często porównywalne. W rozdz. 8.3 opisane są zastosowania metody Adaboost w detekcji i rozpoznawaniu obiektów graficznych, w tym głównie twarzy wraz z przedstawieniem najpopularniejszych zbiorów cech, takich jak cechy prostokątowe (Haar-a), LBP czy HOG. W rozdz. 8.4 opisano wieloklasowe wersje metody Adaboost przydatne w problemach rozpoznawania. W rozdz. 8.5 znajduje się podsumowanie.

8.2 Uczenie klasyfikatora Adaboost

Na rys. 8.1 przedstawiony jest proces uczenia silnego klasyfikatora na przykładzie problemu klasyfikacji punktów należących do 2 klas (kółkami zaznaczone są przykłady klasy 1, krzyżykami przykłady klasy 2) leżących na płaszczyźnie dwuwymiarowej. Słabe klasyfikatory są w tym przykładzie wybierane spośród pionowych i poziomych linii, które wraz z polaryzacją - wskazaniem na lewą lub prawą półpłaszczyznę w przypadku linii pionowej czy na górną lub dolną półpłaszczyznę w przypadku linii poziomej, stanowią jedną z najprostszych baz klasyfikatorów. W przypadku przestrzeni trójwymiarowej klasyfikatorami mogą być spolaryzowane płaszczyzny prostopadłe do jednego z trzech kierunków, a w przypadku dowolnej liczby wymiarów - spolaryzowane hiperpłaszczyzny. Wagi poszczególnych przykładów są odzwierciedlone wielkością kółek lub krzyżyków. Przykładowo, gdy po pierwszym kroku wybrany został słaby klasyfikator h_1 klasyfikujący punkty leżące po lewej stronie prostej pionowej do klasy 1, wagi przykładów prawidłowo klasyfikowanych zostały zmniejszone, a wagi dwóch przykładów błędnie klasyfikowanych (jedno kółko i jeden krzyżyk) zostały zwiększone. W kolejnym kroku wybierany jest zatem taki słaby klasyfikator h_2, który w większym stopniu uwzględnia przykłady dotychczas w większym stopniu błędnie klasyfikowane. W prawym dolnym narożniku rysunku przedstawiona została granica decyzji dla sumarycznego, silnego klasyfikatora H. Czasami wygodnie jest przedstawić słaby klasyfikator w formie wyrażenia $h(\mathbf{x}, f(\mathbf{x}), \theta, p)$, gdzie $\mathbf{x}$ jest wektorem obserwacji, $f(\mathbf{x})$ jest cechą wektora obserwacji, np. pojedynczą współrzędną, θ jest granicą - wartością progową dla wybranej cechy, p jest polaryzacją - kierunkiem, w jakim wyznaczana jest klasa 1.

Na rys. 8.2 przedstawiony jest algorytm uczenia silnego klasyfikatora w pseudokodzie. Po ustaleniu wag początkowych o wartościach zależnych od liczności poszczególnych klas przeprowadzany jest dobór słabych klasyfikatorów. W każdym kroku wybierany jest klasyfikator o minimalnym ważonym błędzie klasyfikacji dla zbioru uczącego, a następnie zmniejszane są wagi przykładów poprawnie przez niego klasyfikowanych, co w połączeniu z normalizacją wag sprawia, że wagi przykładów błędnie klasyfikowanych wzrastają.



Rysunek 8.1. Proces uczenia klasyfikatora Adaboost klasyfikacji punktów na płaszczyźnie dwuwymiarowej

for każdy przykład do uczenia $\{(\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2) \ldots (\mathbf{x}_K, c_K)\}$

przypisz wagom wartości początkowe $w_i = \frac{1}{K_c}$, gdzie K_c jest liczbą przykładów, które należą do tej samej klasy c co i-ty przykład

end for
while błąd fałszywie pozytywnych $f > f_{\max}$ **or** proporcja prawdziwie pozytywnych $d < d_{\min}$

1. normalizuj wagi: $w_i \leftarrow \frac{w_i}{\sum_{j=1}^{K} w_j}$
2. wybierz słaby klasyfikator $h_t(\mathbf{x})$, który minimalizuje ważony błąd klasyfikacji: $\epsilon_t = \min_j \sum_{i=1}^{K} w_i |h_j(\mathbf{x}_i) - c_i|$
3. zmniejsz wagi poprawnie klasyfikowanych przykładów: $w_i \leftarrow w_i \frac{\epsilon_t}{1-\epsilon_t}$
4. znajdź wartość progu Θ dla silnego klasyfikatora dającą jak najmniejszą wartość błędu fałszywie pozytywnych, podczas gdy $d \geq d_{\min}$

end while

Rysunek 8.2. Zachłanny algorytm uczenia silnego klasyfikatora Adaboost

8.2.1 Adaboost jako klasyfikator parametryczny

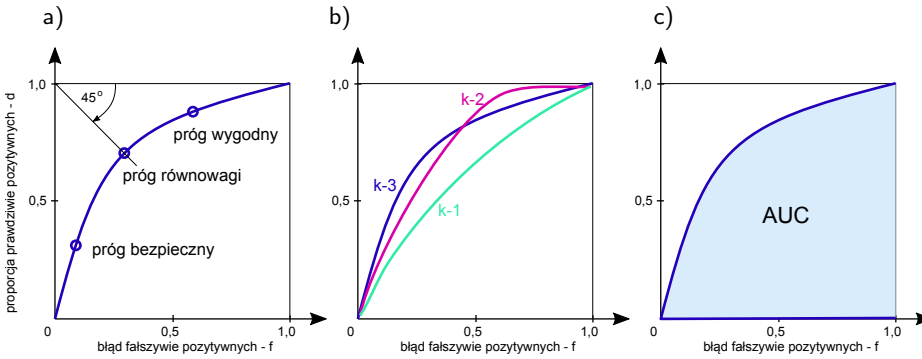
W opisie algorytmu Adaboost przedstawionym na rys. 8.2, dobór wartości progowej Θ wymaga iteracyjnego szukania optymalnego punktu na krzywej ROC (*Receiver operating characteristic*), często w oparciu o walidacyjny zbiór przykładów. Jest to związane z zastosowaniami praktycznymi, które wymagają często przyjęcia dodatkowych ograniczeń na błąd klasyfikacji fałszywie pozytywnych - f lub na proporcję klasyfikacji prawdziwie pozytywnych d. W przypadku braku takich ograniczeń, wartość progu można wyznaczyć ze wzoru $\Theta \leftarrow 0,5 \sum_{t=1}^{T} \alpha_t$. Warunek zakończenia pętli w tym opisie również wynika z przyjętych ograni-

czeń, a w przypadku ich braku może nim być przekroczenie pewnej zadanej liczby słabych klasyfikatorów.

Silny klasyfikator złożony z T słabych klasyfikatorów można przedstawić wzorem:

$$H(\mathbf{x}) = \begin{cases} 1 & \text{jeśli} \quad \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \geq \Theta \\ 0 & \text{w przeciwnym razie,} \end{cases} \tag{8.1}$$

gdzie $\alpha_t = \log(1-\epsilon_t) - \log \epsilon_t$ jest wagą t-tego słabego klasyfikatora, $h_t(\mathbf{x})$ – wartość na wyjściu t-tego słabego klasyfikatora, $\mathbf{x}$ – wektor obserwacji, Θ – wartość progu. Dzięki wartości progowej Θ, klasyfikator Adaboost jest klasyfikatorem parametrycznym w tym sensie, że manipulując wartością progu można uzyskiwać rozwiązania kompromisowe ze względu na błąd klasyfikacji fałszywie pozytywnych i błąd klasyfikacji fałszywie negatywnych (czyli przeciwieństwa proporcji klasyfikacji prawdziwie pozytywnych). Na rys. 8.3a przedstawiona jest typowa krzywa ROC przedstawiająca wartości obu błędów w zależności od wartości progowej. W skrajnych przypadkach, gdy $\Theta = 0$, wszystkie obrazy klasyfikowane



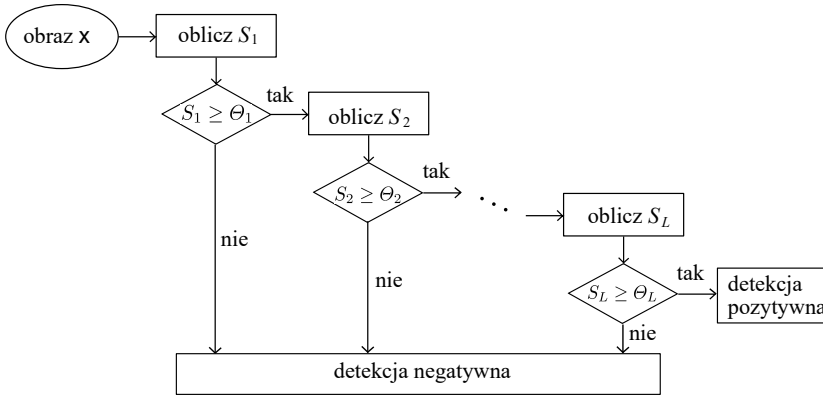
Rysunek 8.3. Krzywe ROC

są jako negatywne, gdy $\Theta = \sum_{t=1}^{T} \alpha_t$, jako pozytywne. W przypadku wcześniej wyuczonego klasyfikatora, wartość Θ można dobrać w zależności od potrzeb, na podstawie ryzyka. Przykładowo, jeśli system rozpoznawania twarzy ma służyć do uwierzytelnienia klienta banku, ważniejsze jest bezpieczeństwo niż wygoda klienta. W takiej sytuacji warto przyjąć nieco niższą wartość Θ, dzięki której, w razie jakichkolwiek wątpliwości, osoba podająca się za klienta nie zostanie dopuszczona do konta. Możliwa jest również sytuacja, gdy zarówno decyzja pozytywna, jak i negatywna powodują podobne konsekwencje. W takiej sytuacji można wykorzystać próg równowagi, przy którym oba błędy są równe. Z uwagi na to, że w przypadku klasyfikatorów parametrycznych o ich ocenie nie decyduje pojedyncza cecha, ale cała krzywa, porównanie różnych klasyfikatorów jest bardziej złożone. Na rys. 8.3b przedstawione są 3 krzywe ROC, z których krzywa k-1 zaznaczona kolorem zielonym jest całkowicie zdominowana przez krzywe od-

powiadające dwóm innym klasyfikatorom. W przypadku klasyfikatorów k-2 i k-3 nie da się już jednoznacznie wybrać lepszego z nich, gdyż k-2 jest lepszy w rozwiązaniach wygodnych, zaś k-3 w rozwiązaniach bezpiecznych. Istnieją jednak miary pozwalające na ocenę klasyfikatorów. Jedną z nich jest miara AUC obliczana jako pole powierzchni pod krzywą (rys. 8.3c).

8.2.2 Kaskada klasyfikatorów Adaboost

W zadaniach praktycznych często stosowana jest pokazana na rys. 8.4 kaskada silnych klasyfikatorów zamiast pojedynczego. Główną tego przyczyną jest możliwość znacznego zredukowania złożoności obliczeniowej. Przykładem jest zadanie detekcji obiektów, takich jak twarze [20] na obrazach dwuwymiarowych. S_i jest sumą ważoną wartości zero-jedynkowych zwracanych przez słabe klasyfikatory: $S = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$ w i-tej warstwie kaskady. Na typowych zdjęciach czy obra-

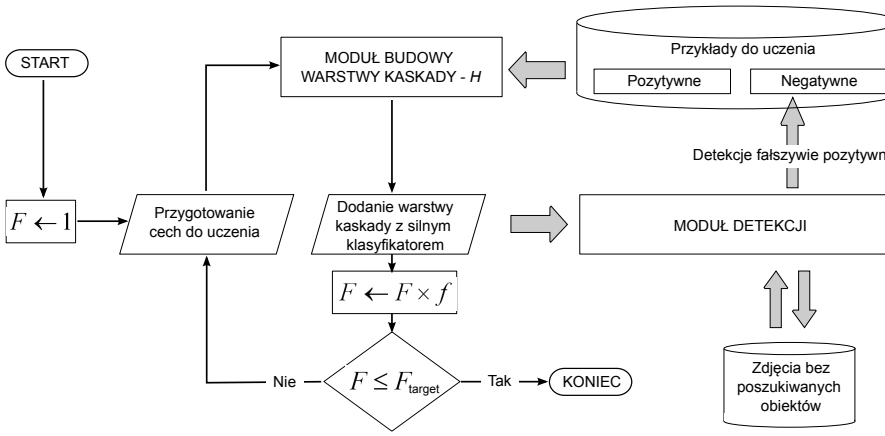


Rysunek 8.4. Kaskada silnych klasyfikatorów Adaboost

zach wideo skanowanych w różnych skalach okien przedstawiających poszukiwane obiekty jest wielokrotnie mniej niż okien przedstawiających tło. Głównym problemem w tym przypadku jest wyeliminowanie okien fałszywie pozytywnych. Zwykle ustawia się w tym celu niski próg Θ w każdym silnym klasyfikatorze, tak by nie wyeliminować okien zawierających obiekty oraz duży docelowy błąd klasyfikacji fałszywie pozytywnych - $f_{\max}$ np. równy $0,5$. Oznacza to eliminację 50% okien zawierających tło przez każdą warstwę kaskady z jednoczesnym zachowaniem okien zawierających obiekty. Dzięki takim ustawieniom każdy z silnych klasyfikatorów odpowiadający warstwie kaskady paradoksalnie nie jest zbyt silny, ale dzięki temu wymaga niewielu operacji arytmetycznych, podczas gdy błąd klasyfikacji fałszywie pozytywnych maleje w sposób wykładniczy wraz z każdą warstwą kaskady. Przykładowo, w przypadku gdy $f_{\max} = 0,5$ i $L = 20$ warstw kaskady, szacowany błąd całej kaskady $F = f_{\max}^{L} \cong 0,5^{20} \cong 10^{-6}$.

8.3 Zastosowanie kaskady Adaboost w detekcji obiektów graficznych

Na rys. 8.5 przedstawiono typowy schemat uczenia kaskady w zadaniu rozpoznawania na zdjęciach obiektów graficznych, takich jak twarze [7, 16]. Po zaini-



Rysunek 8.5. Uczenie kaskady klasyfikatorów Adaboost detekcji obiektów (twarzy)

cjowaniu zbioru cech, budowany jest silny klasyfikator w pierwszej warstwie kaskady z użyciem zbioru przykładów pozytywnych (okien zawierających obiekty) oraz początkowego zbioru przykładów negatywnych (obrazy tła). Po nauczeniu i utworzeniu warstwy taki klasyfikator jest używany do detekcji obiektów w zbiorze zdjęć, które na pewno nie zawierają poszukiwanych obiektów umieszczonych w specjalnym repozytorium. Istotne jest, żeby zdjęcia przedstawiały środowisko, w którym występują obiekty, ale bez ich obecności. Obrazy klasyfikowane jako pozytywne, czyli w tym przypadku zawsze fałszywie pozytywne, są dodawane do bazy przykładów negatywnych, które będą użyte do budowy kolejnej warstwy. Po modyfikacji szacowanej wartości błędu klasyfikacji fałszywie pozytywnych i sprawdzeniu warunku zakończenia cała procedura jest powtarzana aż do spełnienia tego warunku. W ten sposób do budowy kolejnych warstw używane są obrazy, które są coraz trudniejsze pod względem odróżnienia ich od przykładów prawdziwie pozytywnych, tzn. zawierają elementy w coraz większym stopniu przypominające poszukiwane obiekty.

8.3.1 Selekcja cech w detekcji obiektów graficznych

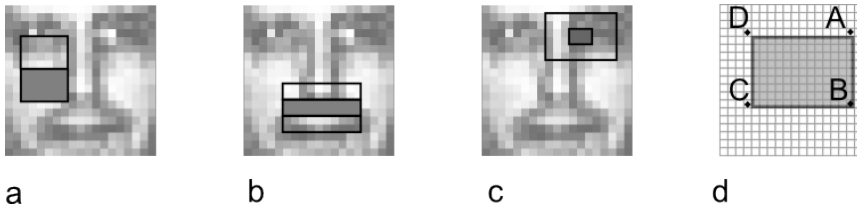
Do najczęściej stosowanych typów cech w zadaniach detekcji obiektów z wykorzystaniem klasyfikatora Adaboost należą:

- różnice średnich jasności pikseli w obszarach prostokątowych (HAAR Features) [16],
- lokalne cechy binarne (*CENSUS*, *Local Binary Features* (LBF) [1, 11]),
- histogramy kierunków gradientów (*Histogram of Oriented Gradients* (HOG) [3]),
- zgrupowane binarne cechy prostokątowe, lokalne zgrupowane cechy binarne (LAB) [17].

Dla wszystkich wyżej wymienionych typów cech możliwe jest wygenerowanie dosyć licznych zbiorów cech, które następnie podlegają selekcji na skutek wewnętrznego mechanizmu metody Adaboost, polegającego na wyborze najlepszych cech w ramach słabych klasyfikatorów. Jest to mechanizm nieco podobny do selekcji cech podczas uczenia drzew decyzyjnych [13] z tą różnicą, że do wyboru cechy podczas budowy drzewa decyzyjnego wykorzystywana jest miara entropii, podczas gdy w przypadku metody Adaboost jest to ważony błąd klasyfikacji.

8.3.2 Cechy prostokątowe (HAAR Features)

Na rys. 8.6 przedstawione są najczęściej stosowane rodzaje cech prostokątowych [16], takie jak cechy dwuprostokątowe (rys. 8.6a), trójprostokątowe (rys. 8.6b) i cechy centralne (rys. 8.6c). We wszystkich trzech przypadkach wartość cechy liczona jest jako różnica pomiędzy średnimi jasnościami pikseli w obszarze przezroczystym i w obszarze wypełnionym. Ukształtowanie obszarów prostokątowych odpowiada niektórym cechom obiektu, takim jak krawędzie czy plamy. W przypadku detekcji lub rozpoznawania twarzy takie ukształtowanie może sprzyjać wykrywaniu oczu czy ust. W literaturze [5, 12] spotyka się też cechy prostokątowe ukośne. W najprostszej wersji proporcje wielkości poszczególnych prosto-



Rysunek 8.6. Rodzaje cech prostokątowych (Haara) i obraz całkowy

kątów składowych są stałe, dzięki czemu nie jest konieczna żadna dodatkowa normalizacja wartości różnic jasności. W przypadku każdego z trzech podanych rodzajów cech prostokątowych cały prostokątny obszar cechy może być dowolnie skalowany w poziomie i pionie, a także dowolnie umiejscawiany w obrębie obrazu, tak by współrzędne położenia poszczególnych narożników prostokątów były zawsze wyrażane w całkowitych liczbach pikseli. Pomimo takich ograniczeń liczba cech jest zwykle dosyć duża i rośnie wraz z czwartą potęgą rozmiarów obrazu. Przykładowo, w przypadku obrazów o rozmiarach 24x24 piksele

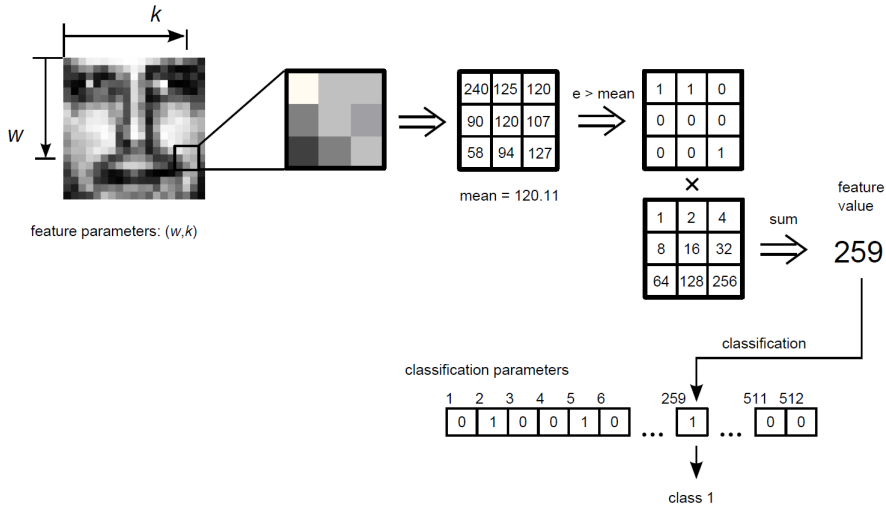
łączna liczba cech prostokątowych wynosi około 160000. W standardowym algorytmie uczenia klasyfikatora na rys. 8.2, w trakcie doboru słabego klasyfikatora (krok 2) konieczne jest wyznaczenie wartości każdej cechy dla każdego obrazu ze zbioru uczącego. W pracy [6] przedstawiono sposoby zmniejszenia średniej liczby operacji obliczania wartości cech poprzez przyjęcie pewnych uproszczeń, jednak pomimo tego proces uczenia klasyfikatora jest bardzo czasochłonny. Na szczęście samo obliczanie średnich jasności wymaga zaledwie kilku operacji arytmetycznych w przypadku zastosowania obrazów całkowych. Obraz całkowy jest tablicą sum jasności wszystkich pikseli leżących na lewo i do góry w stosunku do położenia elementu tablicy. Samo wyznaczanie obrazu całkowego jest operacją o złożoności liniowej w stosunku do liczby pikseli obrazu. Mając obraz całkowy, można szybko obliczyć średnią sumę jasności dowolnego obszaru prostokątnego. Przykładowo, suma jasności pikseli w prostokącie ABCD przedstawionym na rys. 8.6d jest równa $s(D) + s(B) - s(A) - s(C)$, gdzie s jest wartością podanego elementu obrazu całkowego. Zastosowanie cech prostokątowych wymaga spełnienia następujących warunków:

- znormalizowane położenie twarzy w przykładach uczących,
- znormalizowana średnia i wariancja jasności pikseli.

Pierwszy warunek oznacza, że obrazy do uczenia muszą być wycięte ze zdjęć w ściśle określony sposób. W przypadku danych do uczenia detektora twarzy - oczy i usta muszą znajdować się w określonych miejscach okna. Drugi warunek jest dosyć oczywisty ze względu na sposób obliczania cech, gdyż po zmianie kontrastu czy średniej jasności, wartości różnic jasności również ulegają zmianie. Prowadzi to do konieczności normalizacji tych dwóch parametrów również na etapie detekcji i to najlepiej normalizacji lokalnej - w obrębie okna obrazu, dla którego przeprowadzana jest detekcja. Pomagają w tym dodatkowe tablice: tablica sum jasności pikseli (obraz całkowy) oraz tablica sum kwadratów jasności pikseli - do szybkiego wyznaczenia wariancji jasności.

8.3.3 Lokalne cechy binarne (LBP)

Lokalne cechy binarne LBP (*Local Binary Patterns*), zwane też *CENSUS*[11], pozwalają na uwzględnienie lokalnych wzorców jasności pikseli, za pomocą których można niewielkim kosztem obliczeniowym wyznaczyć niektóre charakterystyczne cechy obrazu. Na rys. 8.7 przedstawiony jest jeden z najprostszych wariantów obliczania cech - na podstawie fragmentu obrazu o rozmiarach 3x3 piksele. Najpierw obliczana jest średnia jasność fragmentu, a następnie jasność każdego z pikseli fragmentu jest porównywana do średniej, i w przypadku, gdy jest od niej większa, do wzorca wpisywana jest wartość 1, a w przeciwnym razie 0. Następnie generowany jest numer wzorca w zakresie od 0 do 511, po czym ze specjalnej tablicy klasyfikatora dla cechy opisanej współrzędnymi (w, k) wydobywany jest numer klasy. Do każdej cechy używanej w ramach silnego klasyfikatora lub kaskady silnych klasyfikatorów przypisana jest specjalna tablica klasyfikacji każdego z 512 możliwych wzorców binarnych. W najprostszej wersji klasyfikatora cech jest tyle, ile pikseli mają obrazy do uczenia. W bardziej rozbudowanych



Rysunek 8.7. Przebieg klasyfikacji LBP dla pojedynczego słabego klasyfikatora związanego z pojedynczą cechą

wersjach, takich jak Multi Block LBP[21], zamiast jasności pojedynczych pikseli analizowane są średnie jasności całych obszarów prostokątnych, a sama siatka bloków może być dodatkowo skalowana w poziomie i w pionie. W tym przypadku, poza współrzędnymi położenia (w, k), cechy opisane są dwoma dodatkowymi parametrami i jest ich tym samym znacznie więcej. W jeszcze innej wersji wartości jasności pobierane są z pierścienia o zadanym promieniu[15].

Podobnie jak w przypadku cech prostokątowych, wymagane jest znormalizowane położenie obiektu na obrazach do uczenia. Nie jest jednak konieczna normalizacja średniej i wariancji jasności pikseli ze względu na binaryzację i brak wykorzystania dokładnych wartości jasności.

Uczenie klasyfikatora Adaboost z wykorzystaniem cech LBP[11] jest analogiczne jak w przypadku innych cech i w najprostszym przypadku wykorzystuje algorytm przedstawiony na rys. 8.2. Inaczej przebiega tylko wybór słabego klasyfikatora $h(\mathbf{x}, \mathbf{f}, \mathbf{c})$, który wymaga dodatkowo utworzenia specjalnej tablicy klasyfikacji dla związanej z nim cechy. Składa się on z parametrów cechy $\mathbf{f}$ oraz tablicy $\mathbf{c}$, zawierającej informacje o numerach klas, jakie należy przypisać poszczególnym wzorcom binarnym, których numery są indeksami tej tablicy.

W pierwszym kroku wybierana jest najlepsza cecha, dla której ważony błąd klasyfikacji przykładów uczących jest minimalny:

$$\mathbf{f}_{\text{best}} = \underset{\mathbf{f}}{\operatorname{argmin}}(\epsilon(\mathbf{f})).$$

Błąd obliczany jest ze wzoru:

$$\epsilon(\mathbf{f}) = \sum_{\gamma} \min\{g^0(\mathbf{f}, \gamma), g^1(\mathbf{f}, \gamma)\},$$

gdzie **f** - wektor parametrów cech: (w,k) lub (w,k,dw,dk) w wersji wieloblokowej, γ - numer wzorca (od 0 do 511), $g^c(\mathbf{f}, \gamma)$ - ważona suma przykładów uczących klasy $c \in \{0, 1\}$.

W drugim kroku ustalane są numery klas dla poszczególnych wzorców w tablicy **c**. Numer klasy dla wybranej cechy $\mathbf{f}_{\text{best}}$ i wzorca γ wyznaczany jest ze wzoru:

$$c_\gamma = \begin{cases} 0 \text{ jeśli} & g^0(\mathbf{f}_{\text{best}}, \gamma) < g^1(\mathbf{f}_{\text{best}}, \gamma) \\ 1 \text{ w przeciwnym wypadku.} \end{cases}$$

8.3.4 Histogramy kierunków gradientów (HOG)

Histogramy kierunków gradientów (*Histograms of Oriented Gradients*) są z powodzeniem wykorzystywane w systemach dopasowywania obrazów, takich jak stereografia czy wyszukiwanie za pomocą obrazów. Jednym z najsłynniejszych zastosowań był system rozpoznawania postaci ludzkich odporny na przysłonięcia i spore zróżnicowanie ułożenia poszczególnych części ciała [3]. Kierunek gradientu Ω obliczany jest ze wzoru:

$$\Omega = \arctan \frac{G_y(x, y)}{G_x(x, y)}, \tag{8.2}$$

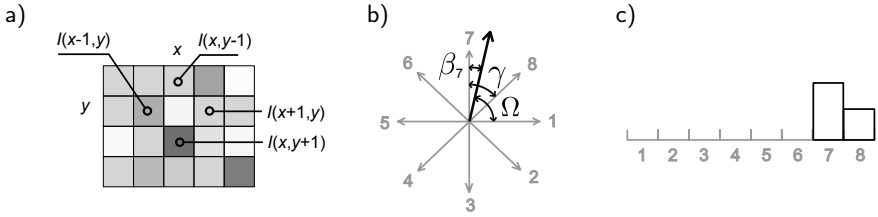
gdzie $G_x(x, y)$ i $G_y(x, y)$ to wartości gradientu w kierunku poziomym i pionowym obliczane ze wzorów:

$$\begin{aligned} G_x(x, y) &= I(x + 1, y) - I(x - 1, y) \\ G_y(x, y) &= I(x, y + 1) - I(x, y - 1), \end{aligned} \tag{8.3}$$

gdzie $I(p, q)$ - jasność piksela (p, q). Do obliczenia gradientu wykorzystywane są piksele sąsiadujące z pikselem x, y w poziomie i pionie, jak zobrazowane jest to na rys. 8.8a.

Samo obliczanie histogramów odbywa się poprzez przesuwanie okna o rozmiarach L_x na L_y pikseli w kierunku poziomym i pionowym okna obrazu oraz każdorazowym wyznaczeniu wektora N-słupków histogramu kierunków dla $L_x * L_y$ pikseli. Wartości poszczególnych słupków nie są na ogół dyskretne ze względu na ważenie wpływu każdego z pikseli wartością $G(x, y)$, jak również ze względu na często stosowaną interpolację liniową kierunku, pozwalającą na uwzględnienie jej wpływu na sąsiednie słupki. Przykładowo, jeśli kierunek pokrywa się ze środkiem zakresu kątowego słupka, to tylko wartość tego słupka jest zwiększana o pewną wartość w. Jeśli zaś kierunek pokrywa się z granicą zakresów dwóch słupków, to wartości obu słupków są zwiększane o połowę wartości w. Każdy słupek odpowiada jednemu ustalonemu kierunkowi gradientu i jest obliczany jako suma wag dla pikseli należących do konkretnego okna - fragmentu obrazu. Niezerowa wartość wagi jest możliwa, gdy ustalony kierunek jest najbliższy obliczonemu po lewej lub prawej stronie, jak pokazano na rys. 8.8b.

Wartość wagowa dla ustalonego kierunku i dana jest wzorem $w_i = (1 - \frac{\beta_i}{\gamma})G(x, y)$, gdzie β_i jest kątem pomiędzy kierunkiem ustalonym i a obliczonym, γ jest kątem pomiędzy sąsiednimi kierunkami ustalonymi, $G(x, y) = (G_x^2(x, y) +$



Rysunek 8.8. Modyfikacja histogramu HOG dla przykładowego punktu (x, y): a) piksele wykorzystane do obliczeń, b) ustalone kierunki i kierunek wyznaczony, c) wektor poprawki histogramu

$G_y^2(x, y))^{0.5}$ jest długością wektora gradientu w punkcie (x, y). Przedstawiona na rys. 8.8 modyfikacja histogramu jest powtarzana dla każdego piksela w określonym oknie obrazu. Liczbę cech HOG (histogramów) można zwiększać przesuwając okno histogramu wewnątrz analizowanego obrazu.

Cech HOG można użyć traktując poszczególne słupki histogramu jako cechy do słabych klasyfikatorów, lub podobnie jak w przypadku cech LBP, zamieniając je na wzorce binarne. Ciekawe efekty można uzyskać łącząc cechy LBP i HOG w jednym klasyfikatorze [18]. Cechy HOG mogą stanowić istotne uzupełnienie, gdyż są w dużym stopniu niewrażliwe na umiejscowienie obiektu wewnątrz okna i tym samym rozmieszczenie cech obiektu złożonego.

8.4 Wieloklasowa wersja klasyfikatora Adaboost

Główną wadą klasyfikatora AdaBoost w zadaniach rozpoznawania obiektów jest to, że jest klasyfikatorem dwuklasowym (binarnym), podczas gdy zadania rozpoznawania są zwykle wieloklasowe. Istnieje wiele rozszerzeń algorytmu Adaboost dla problemów wieloklasowych, które można podzielić na dwie grupy:

- rozszerzenia bezpośrednio stosujące słabe klasyfikatory wieloklasowe,
- rozszerzenia łączące wyniki klasyfikatorów binarnych.

Główną wadą metod bezpośrednich, takich jak AdaBoost.M1 [9] i SAMME [22] jest konieczność stosowania wieloklasowych rozbudowanych słabych klasyfikatorów, które kłócą się z ideą słabego klasyfikatora i mogą ograniczać uogólnianie. W drugiej grupie rozszerzeń, takich jak AdaBoost.M2 [9], AdaBoost.OC [14], One-vs-all czy All-vs-all, silne klasyfikatory wieloklasowe są zmontowane ze słabych lub silnych klasyfikatorów binarnych. Ciekawym rozszerzeniem jest AdaBoost.OC oparte na kodowaniu korygującym (*Error-Correcting Output Codes*) [8]. Technika ta polega na generowaniu kodów binarnych o ustalonej liczbie pozycji L_p, przy czym każdy kod odpowiada jednej klasie. Następnie dla każdej pozycji tworzone jest zadanie klasyfikacyjne dwuklasowe w ten sposób, że etykiety klas są zamieniane na wartości pobrane z odpowiadających klasom ciągów kodowych. Zadanie wieloklasowe sprowadza się do L_p zadań dwuklasowych. Po wytrenowaniu L_p klasyfikatorów binarnych klasyfikacja dowolnego wektora obserwacji polega na wyznaczeniu klas 0 lub 1 przez wszystkie wyuczone klasyfikatory binarne,

utworzenie z nich L_p pozycyjnego kodu, a następnie wybierana jest klasa, dla której odpowiadający jej ciąg kodowy jest najbliższy względem uzyskanego kodu pod względem odległości Hamminga. Metoda All-vs-all polega na wytrenowaniu $K(K-1)/2$ klasyfikatorów dla każdej pary klas, gdzie K jest liczbą klas, a następnie w fazie eksploatacji wykonanie tej samej liczby klasyfikacji i wybór klasy, która jest najczęściej wskazywana. W przypadku porównywalnych liczności poszczególnych klas wystarczającym wariantem wydaje się wersja niekaskadowa. W metodzie One-vs-all trenowanych jest K klasyfikatorów oddzielnie dla każdej klasy w ten sposób, że przykłady należące do wybranej klasy są traktowane jako przykłady pozytywne, a pozostałe przykłady jako negatywne. Ze względu na dużo większą liczność przykładów negatywnych, opłacalne jest stosowanie wersji kaskadowej. Podczas procesu klasyfikacji, obraz wejściowy podawany jest na wejście każdego z K klasyfikatorów i w przypadku, gdy tylko jeden odpowie pozytywnie, klasą jest numer tego klasyfikatora. W przypadkach, gdy pozytywnie odpowie więcej klasyfikatorów lub żaden, konieczne jest porównanie innych wskaźników, takich jak stopień przekroczenia progu. Dodatkowym problemem jest to, w jaki sposób porównać klasyfikatory o różnych liczbach warstw kaskady oraz różnych bezwzględnych wartościach sum ważonych S i wartości progowych Θ. W [4] przedstawiono system doboru metaparametrów określających sposób porównania wyników klasyfikacji klasyfikatorów kaskadowych Adaboost, tak by uzyskać jak najlepsze uogólnianie w podobnych wieloklasowych zadaniach klasyfikacyjnych, takich jak rozpoznawania twarzy wybranej grupy osób. Pokazano, że po zoptymalizowaniu metaparametrów dla pewnej liczby podobnych zadań klasyfikacyjnych - zadań źródłowych uzyskiwano lepsze uogólnianie w zadaniach docelowych, które nie było wykorzystywane w optymalizacji. Takie zjawisko określa się mianem pozytywnego transferu wyników uczenia (*Positive Transfer Learning*). Inne techniki *Transfer Learning* i *Multitask Learning* z wykorzystaniem metody Adaboost do reprezentacji klasyfikatora zostały opisane w [2] i [19].

8.5 Podsumowanie

W pracy przedstawiono główne idee metody Adaboost wraz ze szczegółowym opisem algorytmu uczenia, wersji kaskadowej i zbiorów cech wykorzystywanych w detekcji i rozpoznawaniu obiektów graficznych, takich jak twarze. Niewątpliwą zaletą tej metody jest mała złożoność obliczeniowa procesu klasyfikacji, nawet jeśli dotyczy on detekcji lub rozpoznawania obiektów na obrazach dwuwymiarowych. Dużo lepszą zdolnością do uogólnienia i większą tolerancją na wariancje obiektów pod względem umiejscowienia, kolorystyki, szumów i zniekształceń charakteryzują się głębokie modele neuronowe jednak za cenę znacznie większej złożoności obliczeniowej, która nie jest jeszcze akceptowalna w internecie rzeczy (*IoT*) czy częściowo dla urządzeń mobilnych. Jeśli za jakiś czas postęp technologiczny umożliwi stosowanie głębokich modeli w wymienionych powyżej urządzeniach, to i tak w pewnych zastosowaniach lepiej będzie rozpoznać więcej obiektów z mniejszą dokładnością niż mniej z większą. Pojawia się zatem

idea połączenia metody Adaboost z modelami neuronowymi lub modelami SVM w ramach systemu hybrydowego.

Bibliografia

1. Ahonen, T., Hadid, A., Pietikainen, M.: Face description with local binary patterns: Application to face recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(12), 2037–2041 (2006). https://doi.org/10.1109/TPAMI.2006.244.
2. Dai, W., Yang, Q., Xue, G.R., Yu, Y.: Boosting for transfer learning. In: Proceedings of the 24th International Conference on Machine Learning. p. 193–200. ICML '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1273496.1273521, https://doi.org/10.1145/1273496.1273521.
3. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). vol. 1, pp. 886–893 vol. 1 (June 2005). https://doi.org/10.1109/CVPR.2005.177.
4. Dembski, J.: Multiclass adaboost classifier parameter adaptation for pattern recognition. In: Choraś, R.S. (ed.) Image Processing and Communications Challenges 8. pp. 203–210. Springer International Publishing, Cham (2017).
5. Dembski, J.: Feature type and size selection for adaboost face detection algorithm. unknown (2010).
6. Dembski, J.: Feature reduction using similarity measure in object detector learning with haar-like features. In: Image Processing and Communications Challenges 7 (2016).
7. Dembski, J., Smiatacz, M.: Modular machine learning system for training object detection algorithms on a supercomputer. Advances in System Science pp. 353–361 (2010).
8. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. J. Artif. Int. Res. **2**(1), 263–286 (Jan 1995).
9. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Proceedings of the Thirteenth International Conference on International Conference on Machine Learning. p. 148–156. ICML'96, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996).
10. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences **55**(1), 119–139 (1997).
11. Küblbeck, C., Ernst, A.: Face detection and tracking in video sequences using the modifiedcensus transformation. Image and Vision Computing **24**(6), 564–572 (2006). https://doi.org/https://doi.org/10.1016/j.imavis.2005.08.005, https://www.sciencedirect.com/science/article/pii/S0262885605001605, face Processing in Video Sequences.

12. Lienhart, R., Maydt, J.: An extended set of haar-like features for rapid object detection. In: Proceedings. International Conference on Image Processing. vol. 1, pp. I–I (2002). https://doi.org/10.1109/ICIP.2002.1038171.
13. Quinlan, J.R.: Learning Efficient Classification Procedures and Their Application to Chess End Games, pp. 463–482. Springer Berlin Heidelberg, Berlin, Heidelberg (1983).
14. Schapire, R.E.: Using output codes to boost multiclass learning problems. In: Proceedings of the Fourteenth International Conference on Machine Learning. p. 313–321. ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997).
15. Suri, P.K., Verma, A.: Robust face detection using circular multi block local binary pattern and integral haar features. International Journal of Advanced Computer Science and Applications(IJACSA), Special Issue on Artificial Intelligence **1**(3) (2011). https://doi.org/10.14569/SpecialIssue.2011.010311, http://dx.doi.org/10.14569/SpecialIssue.2011.010311.
16. Viola, P., Jones, M.J.: Robust real-time face detection. International journal of computer vision **57**(2), 137–154 (2004).
17. Yan, S., Shan, S., Chen, X., Gao, W.: Locally assembled binary (lab) feature with feature-centric cascade for fast and accurate face detection. In: 2008 IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–7 (2008). https://doi.org/10.1109/CVPR.2008.4587802.
18. Yang, S., Liao, X., Borasy, U.: A pedestrian detection method based on the hog-lbp feature and gentle adaboost. International Journal of Advancements in Computing Technology **4**, 553–560 (10 2012). https://doi.org/10.4156/ijact.vol4.issue19.66.
19. Yao, Y., Doretto, G.: Boosting for transfer learning with multiple sources. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 1855–1862 (2010). https://doi.org/10.1109/CVPR.2010.5539857.
20. Zhang, C., Zhang, Z.: A survey of recent advances in face detection. Tech. Rep. MSR-TR-2010-66 (June 2010), https://www.microsoft.com/en-us/research/publication/a-survey-of-recent-advances-in-face-detection/.
21. Zhang, L., Chu, R., Xiang, S., Liao, S., Li, S.Z.: Face detection based on multi-block lbp representation. In: Lee, S.W., Li, S.Z. (eds.) Advances in Biometrics. pp. 11–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2007).
22. Zhu, J., Rosset, S., Zou, H., Hastie, T.: Multi-class adaboost. Statistics and its interface **2** (02 2006). https://doi.org/10.4310/SII.2009.v2.n3.a8.

9. Reprezentacja danych dźwiękowych w kontekście metod uczenia maszynowego

Tymoteusz Cejrowski

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
tymoteusz.cejrowski@pg.edu.pl

Streszczenie
Dźwięk odgrywa kluczową rolę w przekazywaniu informacji lub ostrzeganiu o niebezpieczeństwie. Do opracowania wydajnego cyfrowego asystenta głosowego zdolnego do efektywnej współpracy z człowiekiem niezbędne jest użycie algorytmów opisujących sygnał dźwiękowy w formie cyfrowej. W poniższej pracy skategoryzowano i opisano najpowszechniejsze metody opisu sygnałów audio używanych jako wejścia dla algorytmów uczenia maszynowego. Wskazano ich zastosowania i potencjalne problemy związane z ich wykorzystaniem.

Słowa kluczowe: analiza dźwiękowa, analiza spektralna, MFCC, uczenie maszynowe

9.1 Wprowadzenie

Komunikacja dźwiękowa jest umiejętnością pozwalającą na współdziałanie, planowanie czy ostrzeganie przed niebezpieczeństwem występującą u większości ssaków [23]. W przypadku zwierząt i ludzi głównym środkiem komunikacji są sygnały werbalne niosące ze sobą kluczowe informacje dla odbiorcy [38]. Przykładem takiej komunikacji może być szczekanie psa, które zostanie odebrane przez innego osobnika tego gatunku jako sygnał ostrzegawczy. Charakterystyka wydawanego dźwięku wyzwala emocje u odbiorcy i nakazuje mu podjęcie odpowiednich działań [42]. W przypadku ludzi zdolność rozróżniania sygnałów werbalnych a co za tym idzie, rozumienia mowy, nabierana jest z wiekiem. Człowiek uczy się interpretowania pojedynczych dźwięków i wykorzystuje tę umiejętność do bardziej zaawansowanej formy komunikacji, jaką jest logiczna rozmowa [13, 24].

Jednym z celów współczesnych inteligentnych systemów informacyjnych jest implementacja podobnej zdolności rozróżniania dźwięków w cyfrowej maszynie [11]. Filmy z gatunku Sci-Fi przedstawiają silną sztuczną inteligencję (ang. AGI – *Artificial General Intelligence*) jako system zdolny do rozumienia mowy i artykułowania złożonych odpowiedzi na zadane przez człowieka pytanie. Okazuje się, że perspektywa współpracy z maszyną rozumiejącą ludzką mowę jest dzisiaj rzeczywistością dzięki zastosowaniu metod uczenia maszynowego (ang. ML

– *Machine Learning*). Sieci neuronowe umożliwiają budowanie wirtualnych asystentów głosowych, których zdolności dotyczą już nie tylko odczytania intencji czy charakteru dźwięku, ale również interpretacji pojedynczych fonemów i składania ich w ciąg logicznych wyrazów [14, 27]. Proces uczenia maszyny zdolności interpretacji konkretnych dźwięków przebiega jednak w znacząco inny sposób niż u ludzi. Podejście to wymaga opisu dźwięku w sposób zrozumiały dla komputera. Naturalnym wydaje się więc fakt, iż im dokładniej przedstawimy dźwięk (będący falą mechaniczną rozchodzącą się w powietrzu) jako sygnał audio, tym lepiej algorytmy uczenia maszynowego poradzą sobie z interpretacją i rozumieniem języka mówionego. Okazuje się, że przekonanie to jest błędne w kontekście tradycyjnych metod uczenia maszynowego, niewymagających użycia głębokich sieci neuronowych [29]. W takim przypadku rozwiązanie zadania rozumienia mowy wymaga opracowania dodatkowych metod reprezentacji dźwiękowej. Należy jednak zaznaczyć, że ostatnie badania dotyczące użycie metod głębokiego uczenia (ang. *Deep Learning*) częściowo eliminuje ten problem i pozwala na użycie surowego sygnału audio w zadaniu rozpoznawania mowy [31]. Metody te wymagają jednak bardzo dużej liczby danych i znaczących zasobów obliczeniowych co może być przeszkodą w implementacji np. personalizowanych asystentów głosowych [7, 29]. Dodatkowo znajomość różnych metod opisu dźwięków pozwala na optymalny, w kontekście użytego algorytmu, dobór cech i tym samym zmaksymalizowanie zdolność klasyfikacji lub interpretacji sygnałów audio.

9.2 Sygnał dźwiękowy

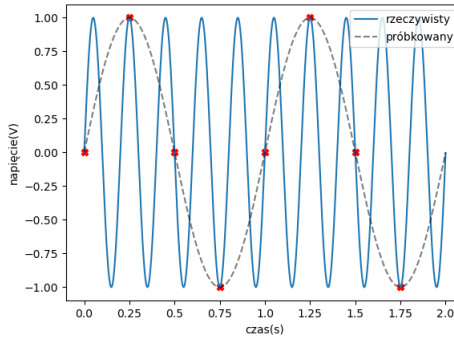
Dźwięk jest wrażeniem słuchowym, które człowiek doświadcza w odpowiedzi na falę mechaniczną rozchodzącą się w powietrzu. Fala ta musi zostać opisana w sposób jednoznaczny w pamięci komputera. W tym celu sygnał analogowy, czyli sygnał o nieskończonej rozdzielczości, będący wyjściem mikrofonu zostaje próbkowany z użyciem konwertera analogowo-cyfrowego (ang. ADC – Analog-Digital Converter) [20]. Najczęściej są to dedykowane urządzenia elektroniczne połączone z urządzeniem wejściowym, tj. mikrofonem. Uzyskuje się w ten sposób ciąg danych określanych jako sygnał audio będący reprezentacją słyszalnego dźwięku zawierającego wszystkie informacje potrzebne do jego bezbłędnej rekonstrukcji (przy pomocy konwertera cyfrowo-analogowego, tj. karty dźwiękowej). Sygnał audio, który został zarejestrowany przy użyciu jednego urządzenia wejściowego nazywany jest sygnałem monofonicznym. Rozszerzeniem monofonii jest stereofonią, w której do reprezentacji fali dźwiękowej wykorzystywane są dwa, niezależne kanały. Poniższe rozważania dotyczą pojedynczego kanału, czyli sygnału monofonicznego.

9.2.1 Próbkowanie i aliasing

Rekonstrukcja fali dźwiękowej na podstawie dyskretnego sygnału audio jest realizowana przez DAC (ang. Digital-Analog Converter). Część informacji niezbędnych do prawidłowej rekonstrukcji dźwięku może zostać jednak bezpowrotnie

utracona przez nieprawidłowe próbkowanie na etapie działania ADC [20, 39]. W takich przypadkach rekonstruowany dźwięk może być zniekształcony lub być całkowicie niesłyszalny. Niezwykle ważnym jest odpowiednie dobranie częstotliwości próbkowania, aby zminimalizować rozmiar zapisywanych danych przy jednoczesnym braku utraty informacji. Proces próbkowania to działanie mające na celu rejestrację i zapis wartości fali dźwiękowej w konkretnych chwilach czasowych. Częstotliwość próbkowania to wyrażona w hercach wartość opisująca ile razy na sekundę została zapisana wartość rejestrowanej fali dźwiękowej.

Rzeczywiste dźwięki są często złożeniem wielu fal o sinusoidalnym kształcie i określonych częstotliwościach. Zbyt rzadkie próbkowanie może powodować pojawienie się zjawiska *aliasingu* czyli nakładania się wyższych częstotliwości na niższe [10, 39]. Przykładowo, dźwięk będący falą mechaniczną o kształcie sinusoidalnym i częstotliwości 3 Hz może zostać błędnie zinterpretowany jako sygnał o niższej częstotliwości. Sytuację tą obrazuje rys. 9.1 w której sygnał sinusoidalny o częstotliwości 5 Hz został zinterpretowany jako sygnał o częstotliwości 1 Hz w wyniku zbyt niskiej częstotliwości próbkowania.



Rysunek 9.1. Sygnał sinusoidalny o częstotliwości 5 Hz i zjawisko aliasingu

Zapobieganie zjawisku aliasingu polega na próbkowaniu sygnał z częstotliwością co najmniej dwa razy większą od najwyższej częstotliwości występującej w sygnale [35]. Częstotliwość ta nazywana jest częstotliwością Nyqiusta i jest definiowana według wzoru 9.1. Z powodu powyższej zależności nagrania na płytach CD są zapisywane z częstotliwością próbkowania 44100 Hz. Maksymalna poprawnie zrekonstruowana częstotliwość będzie wynosić w takim przypadku 22050 Hz, co odpowiada górnemu zakresowi dźwięków słyszalnych przez człowieka.

$$F_N = \frac{f_s}{2} \tag{9.1}$$

9.2.2 Natężenie dźwięku i kwantyzacja

Natężenie dźwięku jest często wymiennie stosowane z poziomem głośności. Jest to jednak znaczący błąd, którego należy unikać. Pojęcie głośności jest subiektywną oceną natężenia dźwięku mierzoną w fonach [3]. Okazuje się, że głośność zależy od natężenia, częstotliwości i czasu trwania słyszanego dźwięku [21, 30]. Mówiąc, że dźwięk jest "głośny" wyraża się subiektywną ocenę. Dla przykładu, dźwięki o niższych częstotliwościach są odbierane jako cichsze, a dłużej trwający dźwięk o tej samej częstotliwości jest odbierany jako głośniejszy.

Natężenie dźwięku jest jednak proporcjonalne do kwadratu amplitudy rejestrowanego sygnału audio (9.2) co daje podstawy do estymacji mocy sygnału na podstawie informacji o kształcie fali dźwiękowej.

$$I = \frac{(\Delta p)^2}{2\rho v_w}, \tag{9.2}$$

gdzie Δp jest amplitudą ciśnienia akustycznego, ρ gęstością ośrodka natomiast v_w reprezentuje prędkość rozchodzenia się fali w tym ośrodku.

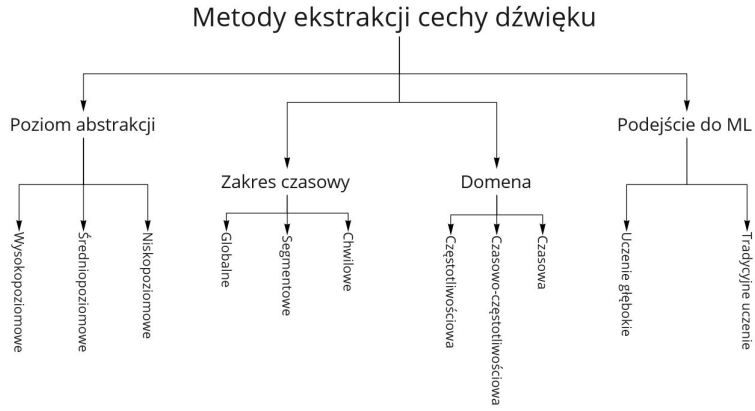
Fala dźwiękowa rozchodząca się w przestrzeni charakteryzuje się energię mierzoną w Wattach na m^2 ($\frac{W}{m^2}$). Wielkość ta nazywana jest natężeniem dźwięku i wpływa bezpośrednio na subiektywne wrażenie głośności. Najczęściej określa się ją w skali decybelowej (9.3) z wykorzystaniem tzw. dolnej granicy słyszalności, czyli wartości $T_{OH} = 10 \times 10^{-12}\,\mathrm{m}^2$.

$$db(I) = 10 \cdot \log_{10}(\frac{I}{I_{TOH}}) \tag{9.3}$$

Do rejestracji analogowego sygnału audio konieczny jest proces próbkowania, który został przedstawiony w rozdziale 9.2.1. Mierzoną w chwili t wartość sygnału audio zamienia się na postać bitową. Działanie to nazywane jest kwantyzacją i zakłada użycie skończonej liczby bitów do reprezentacji pojedynczej próbki sygnału audio. Przykładem może być standard płyt CD wykorzystujący próbkowanie 44100 Hz i rozdzielczość 16 bitów.

9.3 Cechy dźwięku

Sygnał audio będący reprezentacją dźwięku jest ciągiem liczb, który odpowiada dźwiękowi rozchodzącemu się w przestrzeni. Interpretacja takiego surowego ciągu danych jest kłopotliwa i nie pozwoli na zbudowanie systemu interpretującego złożoną mowę. W praktyce sygnał audio jest superpozycją wielu pojedynczych sygnałów o konkretnych częstotliwościach [15]. Użycie algorytmów przetwarzania sygnałów pozwala na opisanie badanego sygnału i wyznaczenie informacji ukrytych w sygnale. Dzięki temu badacz lub system uczący może m.in. zdecydować, kiedy wystąpiło rozpoczęcie zdania, lub gdzie wystąpiła kropka kończąca wypowiedź. Cechy wyliczone przy pomocy poniższych metod mogą być indywidualnym wejściem lub stanowić uzupełnienie innych cech w algorytmach uczenia maszynowego. Metody ekstrakcji cech można skategoryzować zgodnie z diagramem z rys. 9.2 [12].



Rysunek 9.2. Kategoryzacja metod ekstrakcji cech z dźwięku.

Poziom abstrakcji, będący jedną z kategorii metod ekstrakcji cech dźwięku, określa jak bardzo wyznaczone atrybuty dźwięku są intuicyjne dla człowieka. Kategoria określona jako "niskopoziomowa" jest grupą zawiera cechy statystyczne sygnału, które mają sens dla maszyny cyfrowej. Przykładem jest wartość średnia sygnału lub współczynniki takie jak RMS (ang. *Root Mean Square*) i ZCR (ang. *Zero Crossing Rate*) omawiane w kolejnych akapitach. Kategoria "średniopoziomowa" to cechy, które reprezentują intuicyjną wartość dla człowieka jednak wciąż są zdefiniowane z użyciem formuł matematycznych. Przykładem dla grupy może być tzw. *pitch*, czyli percepcyjna wysokość słyszanego dźwięku. Grupa "wysokopoziomowa" zawiera cechy opisowe dźwięku mające sens tylko dla człowieka, np. dźwięki "głośne" czy "niskie".

Kategoria zakresu czasowego dotyczy liczby próbek sygnału audio, która zostanie poddana ekstrakcji cech. Za algorytmy z grupy "chwilowych" uznaje się metody działające na ramkach danych o długości $\sim$ 50 ms, ramki o zakresie liczonym w sekundach należą do grupy "segmentowych", natomiast "globalne" dotyczą całych nagrań.

Domena czasowa dotyczy cech, które są wyliczane z sygnału audio reprezentowanego w dziedzinie czasu, tj. amplitudy sygnału w funkcji czasu. Metody częstotliwościowe działają na sygnale reprezentowanym w dziedzinie częstotliwości. Przejście z dziedziny czasu do dziedziny częstotliwości jest realizowane z użyciem Transformaty Fouriera a ściślej algorytmu Szybkiej Transformaty Fouriera (ang. FFT – *Fast Fourier Transform*), która została omówiona w rozdziale 9.3.2.

Kategoria algorytmów tradycyjnego uczenia maszynowego dotyczy metod, których wynik jest wejściem m.in. dla Maszyny Wektorów Nośnych (ang. SVM – *Support Vector Machine*) czy prostych sieci neuronowych. Drugą grupą są algorytmy uczenia głębokiego, które zakładają, że cechy dźwięku są generowane z wykorzystaniem sieci neuronowych m.in. typu Autoencoder. Wejściem dla sieci

jest surowy lub wstępnie przetworzony (z wykorzystaniem algorytmów m.in. z podejścia tradycyjnego) sygnał audio. Sieć uczy się rekonstruować sygnał z wykorzystaniem tzw. wektora ukrytego (ang. *latent vector*), który jest ostatecznie cechą rekonstruowanego dźwięku.

W rozdziale zdecydowano się na przedstawienie cech sygnału audio w ujęciu domenowym. Motywacją jest możliwość opisania kluczowych algorytmów przetwarzania sygnałów i przedstawienia czytelnikowi potencjalnych problemów, które metody te mogą generować.

9.3.1 Analiza w dziedzinie czasu

Badania przeprowadzane w dziedzinie czasu oznaczają, że sygnał audio jest analizowany pod kątem zmian zachodzących w czasie. Podstawą badań w takim podejściu jest amplituda mierzona w czasie $T = [t_0, t_1, t_2, ..., t_n, t_n + 1]$, gdzie T_s jest okresem próbkowania sygnału, czyli wartością czasu pomiędzy dwoma następującymi po sobie próbkami a t_i. W przypadku sygnału audio wartością mierzoną jest to amplituda fali określana jako $s(t_i)$. Wartość definiowana jako $f = \frac{1}{T_s}$ jest częstotliwością próbkowania.

Sygnał mowy czy inny powszechnie występujący dźwięk jest trudny do analizy w dziedzinie czasu. Wnioski płynące z takiej analizy nie dostarczają wielu informacji użytecznych np. dla metod rozpoznawania mowy (ang. *speech recognition*) [36]. Jest to spowodowane faktem, iż powszechnie słyszane przez człowieka dźwięki są superpozycją wielu fal dźwiękowych o różnych częstotliwościach. Istnieją jednak metody opisujące dźwięk w dziedzinie czasu, które mogą służyć jako uzupełnienie informacji pochodzących z analizy metodami częstotliwościowymi i czasowo-częstotliwościowymi.

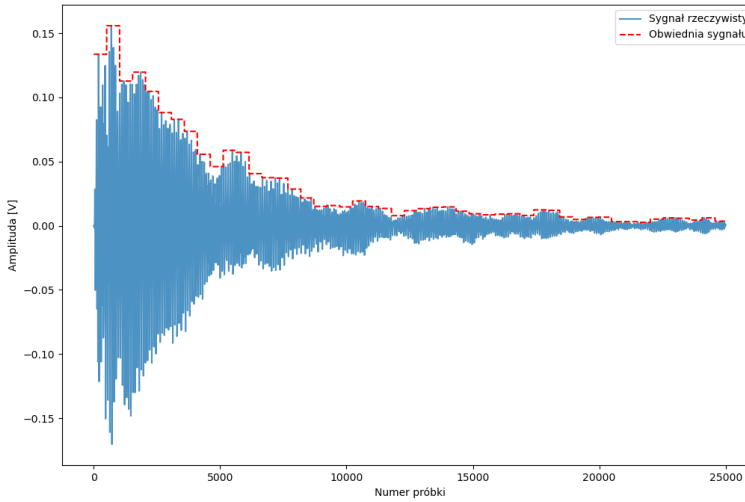
Obwiednia sygnału. Metoda obwiedni sygnału (ang. AE – *Amplitude Envelope*) zakłada podzielenie sygnału $S = [s(t_1), s(t_2), s(t_3), s(t_4), ..., s(t_N)]$, na kolejne ramki sygnału. W ten sposób uzyskuje się zbiór $\lfloor \frac{N}{K} \rfloor$ krótszych przebiegów, w których określone zostaje maksimum sygnału. Obwiednia sygnału jest zdefiniowana według wzoru 9.4.

$$AE_t = \max_{k=t \cdot K}^{(t-1) \cdot K - 1} s(k) \tag{9.4}$$

gdzie t jest numerem ramki a K długością ramki.

Obwiednia daje ogólne pojęcie o poziomie subiektywnej głośności sygnału. Dodatkowo porównywanie obwiedni sygnału (np. poprzez odjęcie dwóch obwiedni) może być początkowym etapem filtracji danych tj. odrzucenia ze zbioru testowego tych dźwięków w znaczącym stopni odbiegających od referencyjnego. Cecha obwiedni sygnału służy również do wyznaczenia zakresu, w którym rejestrowany sygnał zawiera istotne dla dalszego procesowania dane [40]. Przykładem może być początek lub koniec wypowiedzi. Surowe wartości obwiedni sygnału rzadko są jednak wykorzystywane jako bezpośrednie dane wejściowe dla algorytmów uczenia maszynowego z uwagi na wysoką czułość dotyczącą próbek

będący anomaliami. Jedna próbka sygnału, znacząco odbiegająca od średniej może zniekształcić właściwą obwiednię sygnału. Przykładowy przebieg AE wyliczony dla sygnału o długości $N = 25000$ i $K = 512$ został zaprezentowany na rys. 9.3.



Rysunek 9.3. Cecha obwiedni dla sygnału o długości $N = 25000$ próbek i wartości $K = 512$

Średnia kwadratowa sygnału. Średnia kwadratowa energii sygnału (ang. *Root Mean Square Energy*) operuje na ramkach lub całych przebiegach dźwiękowych i jest zdefiniowana jako według wzoru 9.5.

$$RMS_t = \sqrt{\frac{1}{K} \cdot \sum_{j=t \cdot K}^{(t+1) \cdot K - 1} s(k)^2} \tag{9.5}$$

RMS Służy głównie do określenia "głośności" sygnału, która może być wykorzystywana jako wejście dla algorytmów uczenia maszynowego. Cecha ta dostarcza relatywnej (w obrębie zbioru danych treningowych) informacji o średniej mocy sygnału. Jest mniej czuła na anomalie w porównaniu do AE. Wykorzystywana jest jako narzędzie identyfikacji nowych segmentów dźwięku lub w zadaniach klasyfikacji gatunku muzyki.

Współczynnik przejścia przez zero. Współczynnik przejścia przez zero (ang. *Zero Crossing Rate*) ma określać częstotliwość z jaką amplituda sygnału zmienia znak w każdej ramce. Upraszczając, jest to wartość określająca ile razy wartość sygnału zmieniła znak z dodatniego na ujemny i odwrotnie. Cecha ZCR w ramce i jest zdefiniowana równaniem 9.6. ZCR jest zwykle używany do wykrywania rozpoczęcia aktywności głosowej. Identyfikuje czyste dźwięki (ang. *pitch sounds*) w których dla kolejnych ramek współczynnik ten jest stały. Pozwala on także na estymację częstotliwości podstawowej dźwięku (im wyższe ZCR tym wyższa częstotliwość dźwięku). Może służyć jako surowe wejście dla algorytmów uczenia maszynowego.

$$ZCR_t = \frac{1}{2} \sum_{k=t\cdot K}^{(t+1)\cdot K-1} |sgn[s(k)] - sgn[s(k+1)]|, \tag{9.6}$$

gdzie $sgn(\cdot)$ jest funkcją znaku np.

$$sgn[x_i(n)] = \begin{cases} 1, x_i(n) \geq 0, \\ -1, x_i(n) < 0 \end{cases} \tag{9.6a}$$

9.3.2 Analiza w dziedzinie częstotliwości

Reprezentacja sygnału audio w dziedzinie częstotliwości polega na rozbiciu fali dźwiękowej, będącej złożeniem wielu fal o różnych częstotliwościach, na pojedyncze składowe. Następnie, szacowany jest poziom istotności konkretnej składowej w całym sygnale. Wynikiem tego procesu jest wykres przypominający kształtem histogram, w którym na osi X umieszczone są częstotliwości, a na osi Y znajduje się poziom istotności. Proces ten jest analogiczny do zjawiska dyspersji (rozszczepienia) światła, w którym fala świetlna padająca na pryzmat zostaje rozszczepiona na składowe o różnych długościach. Pryzmatem w analizie audio jest algorytm transformaty Fouriera, który służy do konwersji sygnału audio z dziedziny czasu do dziedziny częstotliwości.

Dyskretna Transformata Fouriera. Dla sygnału audio będącą próbkowaną wersją rejestrowanej fali dźwiękowej wyznacza się tzw. dyskretną transformatę Fouriera (DFT)[41]. Jej działanie jest identyczne do transformaty Fouriera z wyjątkiem danych, na których operuje. W przypadku DFT są to sygnały dyskretne zamiast przebiegów ciągłych o nieskończonej rozdzielczości. Algorytm opisany jest wzorem 9.7.

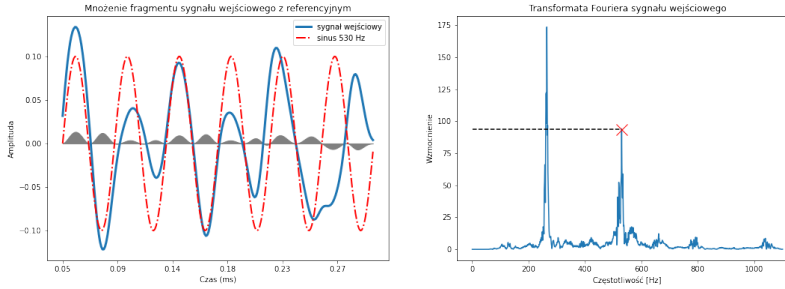
$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kn}, \tag{9.7}$$

gdzie N jest liczbą wszystkich próbek, n numerem próbki, x_n wartością próbki sygnału, i jednostką urojoną a k numerem harmonicznej (częstotliwości).

Powyższe równanie może wydawać się skomplikowane jednak jest tylko reprezentacją mnożenia dwóch sygnałów. U podstaw poznania dyskretnej transformaty Fouriera leży formuła 9.8:

$$e^{-i\frac{2\pi}{N}kn} = \left(\cos\left(\frac{2\pi}{N}kn\right) - i\sin\left(\frac{2\pi}{N}kn\right)\right), \tag{9.8}$$

w której prawa część równania opisuje bezpośrednio sygnał sinusoidalny reprezentowany w postaci zespolonej (amplituda i faza). Wzór 9.7 jest mnożeniem sygnału sinusoidalnego o częstotliwości proporcjonalnej do k z sygnałem audio. Wyliczana jest suma dla wszystkich próbek, która odzwierciedla "ilość" sygnału sinusoidalnego k w sygnale wejściowym. Proces ten powtarzany jest dla zakresu częstotliwości (najczęściej do częstotliwości Nyquista). Na rys. 9.4 został zobrazowany jeden krok algorytmu. Rzeczywisty sygnał audio, będący fragmentem nagrania fortepianu, został poddany transformacji Fouriera. W kroku $k = 530$ sygnał wejściowy został wymnożony z sygnałem sinusoidalnym o częstotliwości $530Hz$. Wynik mnożenia reprezentowany jest przez zacieniowany obszar na wykresie. Całka (suma dla DFT) z zacienionego obszaru daje wynik $X_k = 94$, który jest miarą podobieństwa sygnału wejściowego z sinusoidalnym sygnałem referencyjnym.



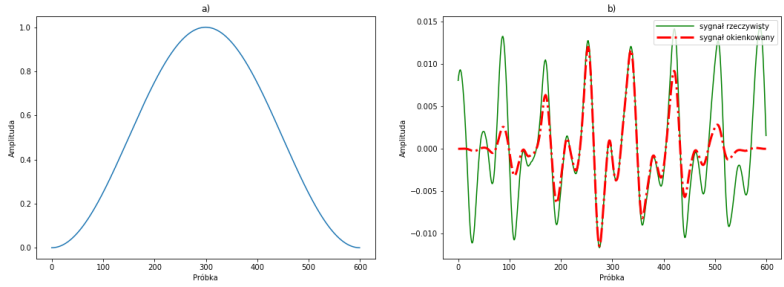
Rysunek 9.4. Sygnał sinusoidalny o częstotliwości 530 Hz z sygnałem wejściowym. Zacieniowany obszar jest rezultatem mnożenia z sumą (całka z obszaru dla sygnałów ciągłych) zaznaczoną na wykresie po prawej, będącym wynikiem transformacji Fouriera

W praktyce, wyznaczanie dyskretnej transformaty Fouriera według wzoru 9.7 jest kosztowne obliczeniowo – $O(n^2)$. W 1965 roku został opracowany algorytm szybkiej transformaty Fouriera (ang. FFT – *Fast Furier Transform*) [5] o złożoności obliczeniowej $O(n \log n)$. Algorytm FFT o podstawie 2 jest dzisiaj głównym algorytmem do wyliczenia DFT.

Przeciek widma. Aby poprawnie interpretować wyniki działania algorytmów transformacji Fouriera konieczne jest zrozumienie ograniczeń, z jakimi są one

związane. Algorytm DFT posiada wymaganie dotyczące sygnału wejściowego, który ma być jednym, pełnym okresem sygnału okresowego. Niespełnienie tego warunku powoduje przekłamania i stłumienie rzeczywistych częstotliwości obecnych w sygnale [18]. Zjawisko to nazywane jest przeciekiem widma (ang. *spectral leakage*).

Rzeczywiste sygnały audio bardzo rzadko mają postać sygnałów okresowych w skali marko (sekundy), przykładem może być rys. 9.3. W takich przypadkach, spełnienie wymagania okresowości sygnału jest realizowane przez tzw. wygładzanie (ang. *windowing*), czyli przemnożenie sygnału wejściowego z funkcją okna (rys. 9.5). Operacja ta tłumi wartości sygnału na końcu i początku przedziału tworząc sygnał okresowy. Aby nie utracić informacji zawartych w stłumionych obszarach, kolejną ramkę sygnału tworzy się przesunięcie okna o liczbę próbek mniejszą od rozmiaru okna. Takie złożenie ramek opisywane jest tzw. skokiem okna (ang. *hop length*). Dobór rozmiaru okna i skoku jest związany z charakterem analizowanego sygnału. W zagadnieniach rozpoznawania mowy najczęściej przyjmuje się rozmiar okna odpowiadający długości od 20 do 50 ms i parametr skoku równy co najwyżej połowie długości okna [6, 36]. Algorytm FFT wymaga również, aby liczba próbek sygnał poddawanego analizie była potęgą dwójki. Najczęściej spotyka się uzupełnianie sygnału zerami, które dodatkowo wpływa na rozdzielczość wyniku FFT [1].



Rysunek 9.5. Okno Hanna (a) i wynik operacji okienkowania (b)

Band Energy Ratio. Współczynnik energii pasma (ang. BER – *Band Energy Ratio*) definiuje stosunek ilości energii niższych częstotliwości do wyższych [26]. Definiowany jest wzorem 9.9 i wymaga wyznaczenia arbitralnej częstotliwości F stanowiącej granicę dla zbioru niższych częstotliwości i początku dla zbioru częstotliwości wyższych.

$$BER = \frac{\sum_{k=1}^{F-1} m(k)^2}{\sum_{k=F}^{K} m(k)^2}, \tag{9.9}$$

gdzie k jest częstotliwością, $m(k)$ jej widmową amplitudą a F arbitralnie wybraną częstotliwością.

Widmowy środek ciężkości. Widmowy środek ciężkości (ang. *Spectral Centroid*) oblicza się jako średnią ważoną częstotliwości obecnych w sygnale, wyznaczonych za pomocą transformaty Fouriera. Służy do określania jasności sygnału. Pojęcie to nawiązuje do zawartości harmonicznych w sygnale, dźwięki postrzegane jako jaśniejsze to zawierają silne wyższe harmoniczne [32, 33]. Widmowy środek ciężkości jest zdefiniowany wzorem 9.10.

$$SC = \frac{\sum_{K}^{k=1} m(k) \cdot k}{\sum_{K}^{k=1} m(k)} \tag{9.10}$$

Pasmo przenoszenia. Pasmo przenoszenia (ang. *Bandwidth*) jest bezpośrednio związane z widmowym środkiem ciężkości i jest odpowiednikiem statystycznej wariancji w dziedzinie analizy sygnałów [9, 43]. Wielkość ta rośnie w przypadku obecności częstotliwości o znaczącym wzmocnieniu zlokalizowanych z dala od SC i maleje w przeciwnym wypadku. Pasmo przenoszenia zdefiniowane jest wzorem 9.11.

$$BW = \frac{\sum_{k=1}^{K} |k - SC| \cdot m(k)}{\sum_{k=1}^{K} m(k)} \tag{9.11}$$

9.3.3 Analiza czasowo-częstotliwościowa

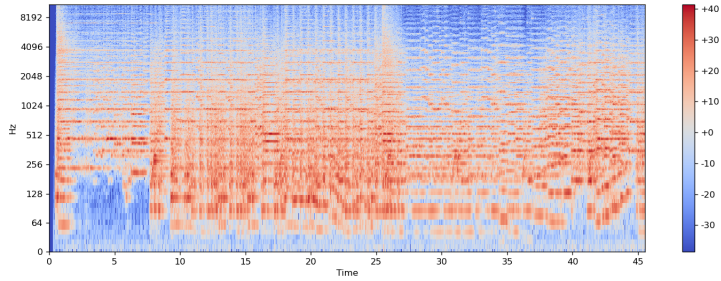
Do zbudowania maszyny cyfrowej, zdolnej do zrozumienia i wyartykułowania logicznej odpowiedzi na zadane pytanie, konieczne jest przetworzenie przez nią całego pytania. Z uwagi na złożoność ludzkiej mowy tj. obecność różnych częstotliwości w sygnale, przerw czy zmiennej mocy sygnału, analiza czasowa lub częstotliwościowa jest niewystarczająca. Modele opracowane w takim podejściu reprezentują szum częstotliwościowy lub są niezdolne do generalizacji. Reprezentacja zdań, wyrazów czy głosek wymaga użycia algorytmów z domeny czasowo-częstotliwościowej.

Spektrogram. Spektrogram jest wykresem widma amplitudowego sygnału audio. Jest on używany do reprezentacji długich i złożonych sygnałów dźwiękowych takich jak ludzka mowa. Do wyznaczania spektrogramu używa się algorytmu STFT (ang. *Short Time Fourier Transform*) [2]. W praktyce procedura obliczania STFT polega na podzieleniu sygnału o dłuższym czasie na krótsze segmenty o jednakowej długości, a następnie obliczeniu transformaty Fouriera oddzielnie dla każdego krótszego segmentu. Następnie wykreśla się zmieniające się widma w funkcji czasu określane jako spektrogram zdefiniowany wzorem 9.12.

$$S(m,k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}}, \tag{9.12}$$

gdzie m jest kolejnym numerem ramki sygnału na którym będzie przeprowadzana Dyskretna Transformacja Fouriera, N długością okna, k wartością odpowiadającą częstotliwości referencyjnej w DFT, H skokiem okna, $w(n)$ wartością funkcji okna dla próbki n.

Dzięki STFT możliwe jest przeprowadzenie analizy częstotliwościowej na sygnale reprezentującym pojedyncze głoski i przedstawienie ich w funkcji czasu [28]. Spektrogram jest najczęściej reprezentowany w skali decybelowej, w której amplituda spektrogramu jest konwertowana z użyciem wzoru 9.3. Operacja ta ma na celu odzwierciedlenie wrażliwości ucha ludzkiego, w której amplitudy niższych częstotliwości są przez ludzi lepiej słyszalne [19, 22]. Na rys. 9.6 został zaprezentowany spektrogram fragmentu ścieżki dźwiękowej *Star Wars*.



Rysunek 9.6. Spektrogram 46 sekund ściezki dźwiękowej filmu Star Wars

Mel Spektrogram. Fakt, że ucho człowieka odbiera amplitudę w skali logarytmicznej jest motywacją dla konwersji amplitudy spektrogramu na skalę decybelową. Okazuje się, że ludzka percepcja częstotliwości również jest nieliniowa [19]. Przykładem mogą być dwie pary dźwięków, których różnica jest taka sama: dźwięk o częstotliwości 200 Hz i dźwięk o częstotliwości 400 Hz w zestawieniu z 1500 Hz i 1700 Hz. Według empirycznych eksperymentów przeprowadzonych przez badaczy w [16] wykazano, że ludzkie ucho odbiera różnicę w częstotliwościach pomiędzy dwoma dźwiękami zależnie od zakresu częstotliwości, w jakich te dźwięki się znajdują. Upraszczając, człowiek charakteryzuje dźwięki 1500 i 1700 jako bardziej podobne niż 200 Hz i 400 Hz. Dzieje się tak z uwagi na nieliniową czułość ucha ludzkiego w kontekście częstotliwości.

W celu odzwierciedlenia percepcji częstotliwości ludzkiego ucha wejściowy spektrogram jest konwertowany do logarytmicznej skali *mel* [37] opisanej wzorem 9.13.

$$m = 2595 \cdot log(1 + \frac{f}{700}) \tag{9.13}$$

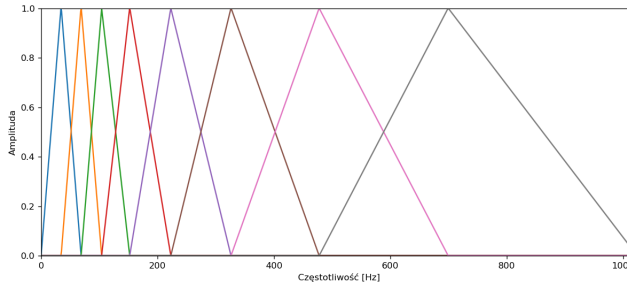
Przejście ze skali częstotliwościowej do skali mel jest realizowane przy użyciu trójkątnych *mel filter* banków, które estymują energię w każdym krytycznym paśmie częstotliwości i przybliżają kształt widma. Liczba banków, bezpośrednio wpływających na kształt wynikowego mel spektrogramu jest hiperparametrem określanym przez badacza i zależy od charakteru badanego sygnału audio. Etapy konwersji częstotliwości spektrogramu do skali mel można rozbić na trzy punkty.

1. Wybranie liczby zakresów (K) dla skali *mel*.

   Liczba zakresów dla skali mel jest zależna od charakteru dźwięku poddanemu badaniu. Najczęściej używa się liczby z przedziału 40 – 128, która może być utożsamiana z liczbą tonów rozróżnialnych przez człowieka.

2. Zbudowanie trójkątnych *mel filter* banków.

   Wyznaczenie trójkątnych banków częstotliwości rozpoczyna się konwersją zakresu częstotliwości (najczęściej od 0 Hz do częstotliwości Nyquista) do skali *mel*. Następnie, otrzymany zakres mel dzielony jest na K równych przedziałów. Punkty będące początkiem lub końcem przedziału są konwertowane z powrotem do częstotliwości. Punkt ten jest początkiem lub końcem dla poprzedniego bądź następnego *mel filter* baku (rys. 9.7).



Rysunek 9.7. Przykładowych 8 filter banków na osi częstotliwości

3. Złożenie *mel filter* banków z wejściowym spektrogramem.

   Proces złożenia filtrów mel i spektrogramu jest mnożeniem dwóch macierzy której wynikiem jest macierz o rozmiarze $K \times M$, gdzie M jest liczbą ramek w spektrogramie.

Mel-Frequency Cepstral Coefficients. Współczynniki Mel-Ceprstralne (ang. Mel-Frequency Cepstral Coefficients) są kolejnym sposobem na opis dźwięków

w kategorii dziedziny czasu i częstotliwości. Metoda opisu opiera się na założeniu, że dźwięki (w szczególności mowa ludzka) mogą być interpretowane jako wyjście filtru pobudzonego sygnałem impulsowym [4, 8]. W takim ujęciu trakt głosowy składający się z krtani, gardła, jamy ustnej i jamy nosowej jest filtrem, natomiast pobudzeniem jest głośnia i tzw. puls krtaniowy (ang. *glottal pulse*). Sygnał impulsowy produkowany w głośni odpowiada za wysokość dźwięku, natomiast filtr głosowy jest elementem nadającym kształt sygnałowi audio. Opis filtru głosowego w danej chwili odpowiada więc barwie danego dźwięku. Współczynniki MFCC służą do opisu tego filtru i są wyliczane według algorytmu zaprezentowanego na rys. 9.8.

Rysunek 9.8. Kolejne krotki wyliczania współczynników MFCC

Logarytmiczne spektrum w analizie MFCC jest interpretowane jako złożenie spektralnej obwiedni (ang. *spectral envelope*) będącej odpowiednikiem odpowiedzi impulsowej filtru głosowego i pobudzenia, które jest wynikiem operacji odejmowania oryginalnego sygnału logarytmicznego spektrogramu i spektralnej obwiedni. Charakteryzacja odpowiedzi impulsowej przy pomocy dyskretnej transformaty kosinusowej dostarcza współczynników Mel-Cepstralnych.

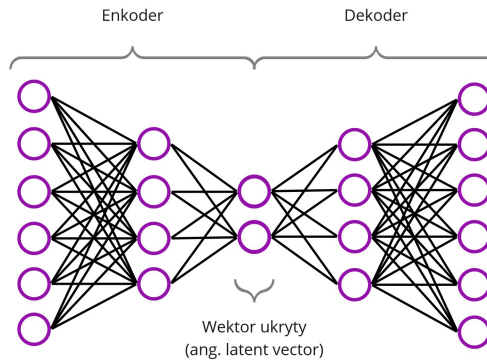
Intuicja stojąca za współczynnikami MFCC dotyczy założenia, że logarytmiczne widmo spektralne sygnału dźwiękowego można traktować ponownie jako sygnał okresowy i poddać transformacie Foureria (lub podobnej, np. transformacie kosinusowej). Otrzymany w ten sposób wykres Cepstrum (podwójne spektrum) zawiera r-harmoniczne, czyli składowe częstotliwościowe logarytmicznego wyniku pierwszego DTF.

Spektrogram, Melspektrogram i macierz współczynników MFCC są popularnymi i powszechnie stosowanymi cechami czasowo-częstotliwościowymi w problemach rozpoznawania czy klasyfikacji dźwięków. Ich macierzowy charakter pozwala na bezpośrednie użycie algorytmów uczenia maszynowego dedykowanym rozpoznawaniu obrazów (np. konwolucyjne sieci neuronowe).

9.3.4 Cechy dźwięku bazujące na sieciach neuronowych

Nowym podejściem w generacji opisu sygnału audio jest użycie uczenia głębokiego do generowania cech sygnału audio [17, 25, 34]. Służą do tego sieci neuronowe typu Autoenkoder, których struktura warstwowa pozwala na kompre-

sowanie sygnału audio do tzw. wektora ukrytego (ang. *latent vector*). Schemat ideowy sieci neuronowej typu Autoenkoder został zaprezentowany na rys. 9.9.



Rysunek 9.9. Schemat Sieci typu Autoenkoder

Architektura sieci składa się z enkodera będącego siecią neuronową dowolnego typu (konwolucyjna, prosta, rekurencyjna) i jej symetrycznego odbicia nazywanego dekoderem [44]. Wejściem są najczęściej cechy wyliczane z użyciem algorytmów z domeny czasowo-częstotliwościowej opisanych w rozdziale 9.3.3. Sieć uczy się poprzez minimalizację błędu rekonstrukcji wejścia (np. z użyciem błędu średniokwadratowego).

9.4 Podsumowanie

Aktualny trend opracowywania coraz ściślej współpracujących z człowiekiem systemów narzuca wymaganie przeprowadzenia bezbłędnej komunikacji głosowej człowieka z maszyną. Zadanie to może zostać z powodzeniem zrealizowane przy użyciu najnowszych osiągnięć z dziedziny sztucznej inteligencji i uczenia maszynowego. Wypracowanie maksymalnej skuteczności wiąże się jednak z faktem, iż algorytmy SI potrzebują wiarygodnych i reprezentatywnych danych dźwiękowych.

W pracy przedstawiono kategoryzację metod opisu sygnałów audio. Zaprezentowano czytelnikowi kompletny proces zbierania, przygotowywania i wyboru metody ekstrakcji cech dźwiękowych. Opisano potencjalne błędy takie jak *aliasing* czy wyciek widma wpływające na przekłamania w reprezentacji dźwiękowej. Informacje zawarte w niniejszej pracy pozwalają na dobranie optymalnej metody opisu sygnału audio do zadanego problemu.

Bibliografia

1. Aamir, K.M., Maud, M.A., Loan, A.: On cooley-tukey fft method for zero padded signals. In: Proceedings of the IEEE Symposium on Emerging Technologies, 2005. pp. 41–45. IEEE (2005).
2. Allen, J.B., Rabiner, L.R.: A unified approach to short-time fourier analysis and synthesis. Proceedings of the IEEE **65**(11), 1558–1564 (1977).
3. Bauer, B.B., Torick, E.L., Allen, R.G.: The measurement of loudness level. The Journal of the Acoustical Society of America **50**(2A), 405–414 (1971).
4. Bridle, J.S., Brown, M.D.: An experimental automatic word recognition system. JSRU report **1003**(5), 33 (1974).
5. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of computation **19**(90), 297–301 (1965).
6. Cutajar, M., Gatt, E., Grech, I., Casha, O., Micallef, J.: Comparative study of automatic speech recognition techniques. IET Signal Processing **7**(1), 25–46 (2013).
7. Dai, W., Dai, C., Qu, S., Li, J., Das, S.: Very deep convolutional neural networks for raw waveforms. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 421–425. IEEE (2017).
8. Davis, S., Mermelstein, P.: Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE transactions on acoustics, speech, and signal processing **28**(4), 357–366 (1980).
9. De Poli, G.: A tutorial on digital sound synthesis techniques. Computer Music Journal **7**(4), 8–26 (1983).
10. Feuer, A., Goodwin, G.: Sampling in digital signal processing and control. Springer Science & Business Media (2012).
11. Goertzel, B., Pennachin, C.: Artificial general intelligence, vol. 2. Springer (2007).
12. Knees, P., Schedl, M.: Music similarity and retrieval: an introduction to audio-and web-based strategies, vol. 36. Springer (2016).
13. Kuhl, P.K.: Cracking the speech code: How infants learn language. Acoustical science and technology **28**(2), 71–83 (2007).
14. Kumar, A., Gupta, A., Chan, J., Tucker, S., Hoffmeister, B., Dreyer, M., Peshterliev, S., Gandhe, A., Filiminov, D., Rastrow, A., et al.: Just ask: building an architecture for extensible self-service spoken language understanding. arXiv preprint arXiv:1711.00549 (2017).
15. Lerch, A.: An introduction to audio content analysis: Applications in signal processing and music informatics. Wiley-IEEE Press (2012).
16. Licklider, J.C.R.: A duplex theory of pitch perception. The Journal of the Acoustical Society of America **23**(1), 147–147 (1951).
17. Liu, H., Li, L., Ma, J.: Rolling bearing fault diagnosis based on stft-deep learning and sound signals. Shock and Vibration **2016** (2016).
18. Lyon, D.A.: The discrete fourier transform, part 4: spectral leakage. Journal of object technology **8**(7) (2009).
19. McEachern, R.: How the ear really works. In: [1992] Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis. pp. 437–440. IEEE (1992).

20. Miranda, E.: Computer sound design: synthesis techniques and programming. Taylor & Francis (2012).
21. Moore, B.C., Glasberg, B.R.: A revision of zwicker's loudness model. Acta Acustica united with Acustica **82**(2), 335–345 (1996).
22. Moore, B.C., Wojtczak, M., Vickers, D.A.: Effect of loudness recruitment on the perception of amplitude modulation. The Journal of the Acoustical Society of America **100**(1), 481–489 (1996).
23. Newman, J.D.: Vocal communication and the triune brain. Physiology & behavior **79**(3), 495–502 (2003).
24. O'grady, W.: How children learn language. Cambridge University Press (2005).
25. Oh, D.Y., Yun, I.D.: Residual error based anomaly detection using auto-encoder in smd machine sound. Sensors **18**(5), 1308 (2018).
26. Peltonen, V., Tuomi, J., Klapuri, A., Huopaniemi, J., Sorsa, T.: Computational auditory scene recognition. In: 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing. vol. 2, pp. II–1941. IEEE (2002).
27. Polyakov, E., Mazhanov, M., Rolich, A., Voskov, L., Kachalova, M., Polyakov, S.: Investigation and development of the intelligent voice assistant for the internet of things using machine learning. In: 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT). pp. 1–5. IEEE (2018).
28. Portnoff, M.: Short-time fourier analysis of sampled speech. IEEE Transactions on Acoustics, Speech, and Signal Processing **29**(3), 364–373 (1981).
29. Purohit, T., Agarwal, A.: Acoustic scene classification using deep cnn on raw-waveform. Tech. Rep., DCASE2018 Challenge (2018).
30. Robinson, D.: The subjective loudness scale. Acta Acustica united with Acustica **7**(4), 217–233 (1957).
31. Sainath, T.N., Weiss, R.J., Senior, A., Wilson, K.W., Vinyals, O.: Learning the speech front-end with raw waveform cldnns. In: Sixteenth Annual Conference of the International Speech Communication Association (2015).
32. Schubert, E., Wolfe, J.: Does timbral brightness scale with frequency and spectral centroid? Acta acustica united with acustica **92**(5), 820–825 (2006).
33. Schubert, E., Wolfe, J., Tarnopolsky, A.: Spectral centroid and timbre in complex, multiple instrumental textures. In: Proceedings of the international conference on music perception and cognition, North Western University, Illinois. pp. 112–116. sn (2004).
34. Serizel, R., Bisot, V., Essid, S., Richard, G.: Acoustic features for environmental sound analysis. In: Computational analysis of sound scenes and events, pp. 71–101. Springer (2018).
35. Shannon, C.E.: Communication in the presence of noise. Proceedings of the IRE **37**(1), 10–21 (1949).
36. Shrawankar, U., Thakare, V.M.: Techniques for feature extraction in speech recognition system: A comparative study. arXiv preprint arXiv:1305.1145 (2013).
37. Stevens, S.S., Volkmann, J., Newman, E.B.: A scale for the measurement of the psychological magnitude pitch. The journal of the acoustical society of america **8**(3), 185–190 (1937).

38. Thorpe, W.H.: The comparison of vocal communication in animals and man. Non-verbal communication pp. 27–47 (1972).
39. Tsividis, Y.: Digital signal processing in continuous time: a possibility for avoiding aliasing and reducing quantization error. In: 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing. vol. 2, pp. ii–589. IEEE (2004).
40. Vallet, G.T., Shore, D.I., Schutz, M.: Exploring the role of the amplitude envelope in duration estimation. Perception **43**(7), 616–630 (2014).
41. Winograd, S.: On computing the discrete fourier transform. Mathematics of computation **32**(141), 175–199 (1978).
42. Wright, J.C.: Severe attacks by dogs: characteristics of the dogs, the victims, and the attack settings. Public health reports **100**(1), 55 (1985).
43. Yost, W.A., Zhong, X.: Sound source localization identification accuracy: Bandwidth dependencies. The Journal of the Acoustical Society of America **136**(5), 2737–2746 (2014).
44. Yu, S., Principe, J.C.: Understanding autoencoders with information theoretic concepts. Neural Networks **117**, 104–123 (2019).

10. Blockchain: zdecentralizowane zaufanie

Stanisław Barański

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
stanislaw.baranski@pg.edu.pl

Streszczenie
Bitcoin wprowadził innowację na wielu płaszczyznach. Jako pierwszy rozwiązał problem osiągania konsensusu w sieciach otwartych, stworzył zagadkę ekonomiczną w postaci globalnej waluty deflacyjnej, pozwolił na transfer pieniędzy tym, którzy wcześniej byli wykluczeni bankowo, ale przede wszystkim stworzył fundamenty pod platformę zdecentralizowanego zaufania. Zapoczątkował technologie zdecentralizowanych aplikacji, które dotychczas nie mogły istnieć bez zaufanej trzeciej strony. W pracy opisana została ta ostatnia płaszczyzna.

Słowa kluczowe: blockchain, zaufanie, kryptowaluty, Bitcoin, Ethereum

10.1 Wprowadzenie

Zaufanie jest fundamentalnym komponentem relacji społecznych. Mówimy, że ufamy komuś, kiedy spodziewamy się uczciwego zachowania, czyli postępowania według ustalonych zasad.

Korzystając z mediów społecznościowych, poczty elektronicznej, przestrzeni w chmurze, ufamy, że nasze dane są odpowiednio zabezpieczone oraz że dostawca nie używa ich do innych celów niż świadczenie nam usług. Poddając się badaniom genetycznym, ufamy, że nasze dane nie są dostępne dla pracowników laboratorium. Pożyczając koledze samochód, ufamy, że odda nam go w nienaruszonym stanie. W sytuacji gdyby się tak nie stało, ufamy, że sąd uczciwie rozwiąże konflikt na naszą korzyść. Sprzedawca, który sprzedaje nam swój towar, ufa nam, że pieniądze które otrzymał nie są podrobione oraz że w przyszłości będzie mógł kupić za nie inny towar. Jeśli płatność otrzymał przelewem, ufa bankowi, że ten nie cofnie przelanych mu pieniędzy. Ufa również, że bank nie zablokuje mu jego środków. Oddając głos w wyborach, ufamy, że nasz głos zostanie uwzględniony podczas liczenia oraz że zostanie poprawnie zaliczony dla zaznaczonego kandydata. Ufamy, że tylko uprawnieni wyborcy mogą oddać głos oraz że każdy wyborca może oddać tylko jeden głos. Na koniec, ufamy, że wyniki, które zostaną ogłoszone nie zostały zafałszowane.

Zaufanie jest spoiwem łączącym grupy społeczne. Pozwala się rozwijać i sprawia, że nasze życie staje się prostsze. Socjologowie są zgodni co do tego, że

"codzienne życie społeczne, które uznajemy za naturalne, jest bez zaufania po prostu niemożliwe" [14], zaufanie jest również "podstawową potrzebą transakcyjną" [21] (cyt. według [23]). Co jednak, jeśli chcielibyśmy osiągnąć ten sam, a nawet większy poziom zaufania, ale bez instytucji zaufania publicznego?

10.2 Zaufanie do waluty

Zacznijmy od waluty, bo to od niej zaczyna się historia zdecentralizowanego zaufania. Aby jakakolwiek waluta miała wartość, musi być deficytowa — jej ilość lub chociaż dostęp do niej musi być ograniczony. Pierwszą walutą tego typu było złoto, jego naturalne ograniczenie oraz koszt wydobycia idealnie spełniały te założenie. Złoto było walutą, która pozwalała na transakcje bez zaufanej trzeciej strony, było jednak niepraktyczne w transporcie oraz niewielkich płatnościach. System bankowy rozwiązał ten problem wprowadzając papiery wartościowe; każdy mógł zdeponować w banku złoto w zamian za certyfikat (banknot) pozwalający w późniejszym czasie na odzyskanie zdeponowanego złota, tworząc w ten sposób "papierowe" złoto, które było łatwiejsze w transporcie oraz pozwalało na mniejsze transakcje. Bank gwarantował, że liczba banknotów zawsze będzie odzwierciedlać ilość złota w depozycie, a ludzie *ufali*, że faktycznie tak będzie. O ile zagwarantować, że liczba złota nie przewyższy liczby banknotów, nie jest trudno, o tyle zagwarantować, że liczba banknotów nie będzie większa niż ilość rezerw złota, nie jest tak łatwo. Tworzenie banknotów jest o wiele mniej kosztowne niż wydobywanie złota.

Podczas I wojny światowej większość krajów Europy popadła w ogromne długi spowodowane wydatkami na działania wojenne. Banki pożyczały pieniądze, jednak przez standard złota ich zasoby były ograniczone. Kraje jednak potrzebowały kredytu, aby móc dalej się rozwijać. Rozwiązaniem tego problemu miało być tymczasowe zerwanie ze standardem złota, pozwalającym na dodruk pieniądza bez pokrycia kruszcem, co za tym idzie sztuczne pobudzenie gospodarki. Niestety, tymczasowe rozwiązania często okazują się ponadczasowe. W normalnej sytuacji odejście od parytetu złota spowodowałoby niepokoje społeczne. Wyjątkowa sytuacja jaką jest depresja, wojna lub kryzys pozwoliła jednak na wprowadzenie takich zmian na stałe. Do dziś większość krajów nie powróciła do standardu złota, a waluty fiducjarne — które bazują na zaufaniu do emitenta — nie mają pokrycia w żadnym surowcu. Jej wartość bazuje na monopolu jako legalnego środka płatniczego, popycie generowanym przez podatki oraz na zaufaniu pomiędzy dwoma stronami transakcji [11, 20].

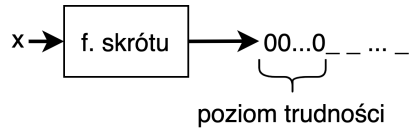
10.3 Historia kryptowalut

Po odejściu od parytetu złota oraz po powstaniu Internetu pojawiła się dodatkowa motywacja do stworzenia waluty, która posiadałaby deficytowe cechy złota, nie była kontrolowana przez żaden rząd, była anonimowa, której transfer byłby tak prosty i szybki jak wysyłanie e-maila.

W 1998 Wei Dai zaproponował walutę **b-money** [1], której wytwarzanie wymaga rozwiązywania zagadek kryptograficznych. Zagadka powinna być w klasie problemów NP, tak aby znalezienie rozwiązania było trudne — wymagało poświęcenie mocy obliczeniowej, a zweryfikowanie poprawności było łatwe. Przykładem może być znajdowanie liczby, która podana do funkcji skrótu zwróci liczbę z pewnego przedziału. Formalnie taką zagadkę możemy zapisać:

$$\text{znajdź } x \in \mathbb{N}, \text{takie że, } H(x) < c, \tag{10.1}$$

gdzie H jest funkcją skrótu np. SHA-256, a c jest liczbą definiującą poziom trudności — im mniejsza, tym ciężej znaleźć x spełniające nierówność; c można również interpretować jako liczba wymaganych zer w wyniku funkcji skrótu (patrz rysunek 10.1).



Rysunek 10.1. Zagadka kryptograficzna typu dowód pracy (ang. *proof-of-work*)

Szansa na znalezienie skrótu zaczynającego się od jednego zera wynosi 50%, od dwóch zer 25%, trzech 12,5%; poziom trudności rośnie wykładniczo wraz ze wzrostem wymaganych zer. Rozwiązywanie zagadki konsumuje moc obliczeniową, która jest deficytowym zasobem. Ilość wytworzonej waluty odpowiada ilości pracy włożonej w rozwiązanie zagadki. W b-money, ilość otrzymanej waluty obliczana jest poprzez stosunek kosztu rozwiązania zagadki na najbardziej optymalnym komputerze do ceny statystycznego koszyka zakupowego.

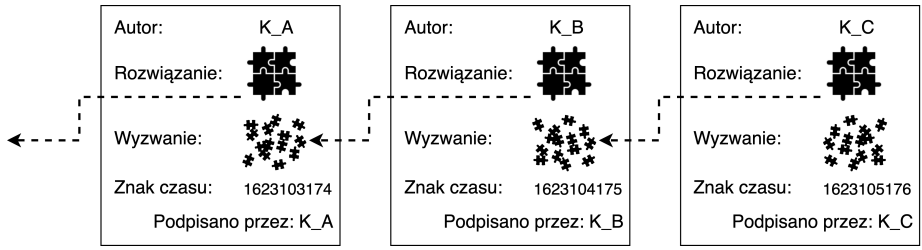
Każdy, kto rozwiąże taką zagadkę, może wymienić dowód (wartość x) na jednostki b-money. Dowód rozgłaszany jest w sieci *peer-to-peer* (p2p) które po otrzymaniu poprawnie rozwiązanej zagadki, aktualizują stan konta autora dowodu pracy (ang. *proof-of-work*). Podmioty w sieci identyfikowane są za pomocą pseudoanonimowych adresów (kluczy publicznych), a weryfikacja ich tożsamości jest możliwa za pomocą podpisów cyfrowych stworzonych przy pomocy klucza prywatnego.

Aby uniemożliwić podwójne tworzenie waluty, każdy węzeł w takiej sieci p2p zapisuje w swojej bazie danych wszystkie dowody, które otrzymał, nie pozwalając na duplikaty.

Załóżmy sytuację, w której Alice identyfikująca się kluczem publicznym K_A chce przesłać pieniądze do Boba identyfikującego się kluczem publicznym K_B. Transfer odbywa się przez stworzenie transakcji «Ja K_A przekazuję x jednostek z konta K_A na konto K_B. Podpisano K_A» i rozesłanie jej wszystkim węzłom w sieci. Po otrzymaniu wiadomości, każdy węzeł sprawdza poprawność podpisu oraz czy stan jednostek przypisanych do konta K_A jest nie mniejszy niż x, następnie obciąża konto K_A kwotą x i dodaje tą wartość do konta K_B.

Pomysł b-money był niestety mało praktyczny, zakładał istnienie *idealnego rozgłoszenia*, gdzie każda transakcja bezproblemowo i natychmiastowo zostaje odebrana przez każdego uczestnika sieci, zakładał, że każdy węzeł będzie zawsze dostępny oraz nie będzie zachowywał się szkodliwie (nie będzie węzłem bizantyjskim[16]). B-money nigdy nie został wcielony w życie.

W 2005 r. Nick Schabo oficjalnie opublikował koncept zdecentralizowanej cyfrowej waluty **BitGold** [2]. Podobnie jak Wei Dai, chciał jak najbardziej odzwierciedlić charakterystyczną cechę złota — deficytowość, jednak w świecie cyfrowym, na dodatek w sposób zdecentralizowany, bez zaufanej trzeciej strony. Podobnie jak b-money nowe jednostki tworzone są przez rozwiązanie zagadki kryptograficznej. Dowód pracy oznaczany jest znakiem czasowym (ang. *timestamp*) przez rozproszony system serwisów, a następnie dodawany do — również rozproszonego — rejestru tytułów własnościowych. Każda zagadka kryptograficzna bazuje na poprzednim rozwiązaniu zagadki, tworząc łańcuch dowodów (patrz rysunek 10.2).



Rysunek 10.2. Łańcuch dowodów w BitGold

Podobnie jak b-money, BitGold nie był odporny na węzły bizantyjskie, a rozproszona sieć serwerów rejestru tytułów własnościowych oraz serwerów znakujących czas nie miała motywacji do zachowywania się poprawnie.

10.4 Bitcoin

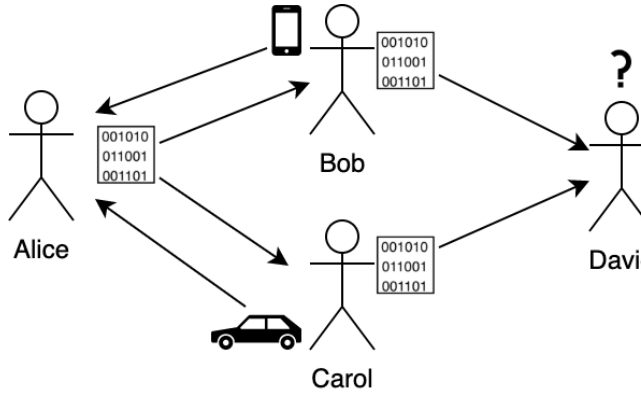
Dopiero w 2008 r. anonimowy autor pod pseudonimem Satoshi Nakamoto opublikował dokument pod tytułem "Bitcoin: A Peer-to-Peer Electronic Cash System", wyjaśniający jak osiągnąć otwarty, bezpieczny i zdecentralizowany system płatności *peer-to-peer* [18]. Biorąc pod uwagę b-money oraz BitGold, Bitcoin wprowadził niewiele innowacji. Jednak wystarczająco dużo aby zrewolucjonizować system bankowy, a później również inne systemy oparte na zaufanej trzeciej stronie.

10.4.1 Problem podwójnego wydawania

Bitcoin tak samo jak jego poprzednicy użył algorytmu *proof-of-work* [9], jednak nie tylko do wytwarzania nowych jednostek, ale przede wszystkim do osiągnię-

cia globalnego konsensusu — rozwiązując w ten sposób problem podwójnego wydawania (ang. *double spending problem*). Problemu, który dotychczas rozwiązywany był przez centralny podmiot — lub jak w przypadku BitGold przez konsorcjum podmiotów. Problem podwójnego wydawania polega na trudności stwierdzenia, czy dana jednostka waluty nie została już wcześniej wydana, innymi słowy, czy wcześniej nie została opublikowana transakcja, która wydaje te same jednostki waluty.

Rozważmy przykład z rysunku 10.3; Alice jest w posiadaniu ciągu bitów, które reprezentują pewną ilość waluty, Alice chcąc kupić od Boba telefon, wysyła mu ciąg bitów, Alice jednak jest nieuczciwa i zawczasu zrobiła sobie kopie tych bitów, następnie wydaje je ponownie, tym razem kupując od Carol samochód. Przewinienie to zostaje niezauważone do momentu, gdy zarówno Bob i Carol będą chcieli wydać swoje środki u jednego sprzedawcy — Davida, który nie będzie chciał przyjąć ciągu bitów, które już wcześniej otrzymał.



Rysunek 10.3. Problem podwójnego wydawania, w którym Alice wydaje dwa razy swoje środki, a David nie jest w stanie określić, kto jest ich faktycznym właścicielem

W przypadku banknotów problem ten jest rozwiązany przez fizyczne przemieszczenie, oraz nielegalność i trudność w kopiowaniu ich. W cyfrowym świecie nic nie stoi na przeszkodzie, aby zduplikować ciąg bitów i przesłać go do wielu osób, wydając jedną jednostkę wiele razy. W systemach z zaufaną trzecią stroną na ten problem odpowiada centralna jednostka (np. Visa, Mastercard lub sam bank), która autorytatywnie odpowiada, która transakcja miała miejsce wcześniej, unieważniając kolejną. Na taką centralną jednostkę można patrzeć jak na zegar, który znakuje czasem każdą transakcję.

10.4.2 Serwer znakowania czasem

Problem znakowania czasem jest łatwy dla pojedynczego serwera, ponieważ jednoznacznie może stwierdzić, która transakcja pojawiła się jako pierwsza, a która

jako druga. Dla systemów rozproszonych nie jest to już tak oczywiste, ponieważ każda jednostka może obserwować inną kolejność zdarzeń. Problem spotykany pod nazwą rozproszonego znakowania czasem, ma swoje korzenie w teorii względności[15] — dwie informacje wysłane dwoma różnymi kanałami (np. zapchanym łączem i wolnym łączem) pojawiają się w różnym czasie u odbiorcy, zakłamują faktyczną kolejność zdarzeń, tj. pierwsza wiadomość, która została wysłana wolniejszym łączem, może dotrzeć później do odbiorcy, niż druga wiadomość wysłana szybszym łączem. Sytuacja dodatkowo komplikuje się, gdy odbiorców jest wiele i każdy z nich obserwuje inną kolejność. Rozwiązaniem problemu jest *total-order broadcast*, w którym węzły uzgadniają między sobą wspólną kolejność, tworząc w ten sposób wirtualny zegar.

10.4.3 Konsensus

Ustalanie wspólnej kolejności jest szczególnym przypadkiem problemu rozwiązywanego przez algorytm konsensusu. Węzły muszą dojść do porozumienia odnośnie pewnej wartości, którą w tym przypadku jest kolejności zdarzeń.

Tradycyjne algorytmy konsensusu, jak Paxos[17] czy Raft[19] rozwiązują ten problem, jednak nie są odporne na węzły bizantyjskie (węzły, które mogą wysyłać sprzeczne informacje do różnych węzłów w sieci). Dzięki pracy Lessiego Lamporta [16] wiemy, że algorytm konsensusu może tolerować n węzłów bizantyjskich, jeśli łączna liczba węzłów w sieci wynosi co najmniej $3n + 1$, czyli aby sieć tolerowała co najwyżej jeden węzeł bizantyjski, musi składać się z łącznie czterech węzłów, dla dwóch węzłów bizantyjskich minimalna liczba węzłów wynosi siedem itd. Algorytmem tego typu jest PBFT [13], jednak nawet on nie sprawdzi się w sieciach p2p.

Wszystkie dotychczas wymienione algorytmy bazują na głosowaniu większościowym, w którym wspólnie uznaną wersją jest ta, która otrzymała większość głosów.

W sieciach otwartych — gdzie węzły mogą dowolnie dołączać i odłączać — liczba członków nie jest stała, dlatego określenie większości głosów jest problematyczne. Sytuacja staje się jeszcze gorsza, gdy uwzględnimy atak *sybila*, w którym to jeden podmiot dołącza do sieci wieloma tożsamościami osiągając zakłamaną opinie większościową. Przez długi czas problem osiągania konsensusu w otwartych sieciach p2p pozostawał nierozwiązany.

10.4.4 Dowód pracy jako rozwiązanie konsensusu na globalną skalę

Bitcoin rozwiązał ten problem w genialny sposób. Słuszną kolejnością zdarzeń jest ta kolejność, której została poświęcona największa ilość pracy obliczeniowej, bez znaczenia, czy pokrywa się z fizyczną kolejnością zdarzeń. Co istotne, taki mechanizm pozwala każdemu na uczestniczenie w algorytmie konsensusu, przy jednoczesnej odporności na atak *sybila*; dzieje się tak, ponieważ w algorytmie konsensusu z wykorzystaniem *proof-of-work*, na wynik nie wpływa liczba głosów — jak ma to miejsce w algorytmach bazujących na głosowaniach ilościowych — lecz sumaryczna moc obliczeniowa, która jest taka sama przy podawaniu się za

jeden, jak i wiele podmiotów. Co więcej, protokół Bitcoin gwarantuje poprawne działanie tak długo, jak większość sieci (liczona w mocy obliczeniowej) zachowuje się poprawnie, t.j. conajmniej 51% jest uczciwa.

Aby zrozumieć, jak Bitcoin to osiągnął, musimy spojrzeć na każdą część z osobna.

10.4.5 Transakcje

W Bitcoinie transfer pieniędzy pomiędzy Alice i Bob odbywa się poprzez stworzenie transakcji «Ja K_A zmieniam właściciela bitcoinów które otrzymałam w poprzedniej transakcji o skrócie h na konto o adresie K_B. Podpisano K_A.» Każdy może zweryfikować taki transfer cofając się w łańcuchu transakcji dochodząc do miejsca w którym dana jednostka została stworzona. Jeśli w którymś miejscu skrót transakcji się nie zgadza bądź podpis cyfrowy jest niepoprawny, transakcja jest odrzucana.

Wciąż pozostaje jednak problem podwójnego wydawania. Co jeśli Alice stworzy dwie transakcje,

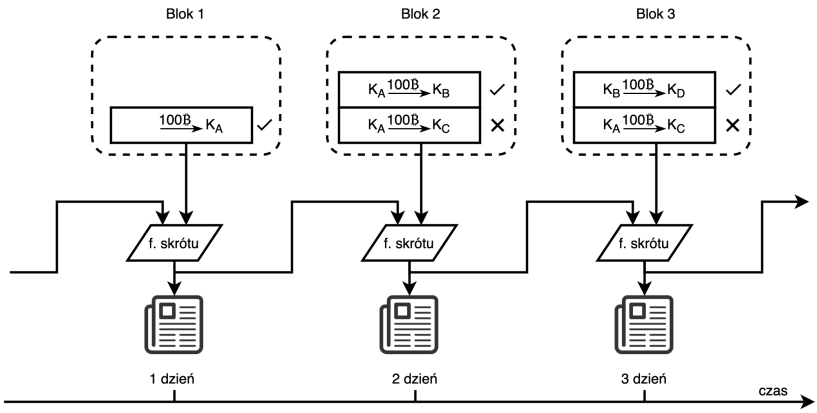
1. «Ja K_A zmieniam właściciela bitcoinów które otrzymałam w poprzedniej transakcji o skrócie h na konto o identyfikatorze $\mathbf{K_B}$. Podpisano K_A».
2. «Ja K_A zmieniam właściciela bitcoinów które otrzymałam w poprzedniej transakcji o skrócie h na konto o identyfikatorze $\mathbf{K_C}$. Podpisano K_A».

Jedną transakcję wyśle do Boba, a drugą do Carol. Obydwaj pomyślnie przeprowadzą weryfikację. W konsekwencji Alice dwa razy wyda środki o skrócie h.

Bitcoin znalazł rozwiązanie tego problemu bez zaufanej trzeciej strony.

Transakcje grupowane są w bloki; następnie skrót takiego bloku publikowany jest w miejscu, gdzie nie da się go usunąć ani zaprzeczyć, że istniał, np. zamieszczając w gazecie codziennej. Tysięcy wydrukowanych egzemplarzy nie da się zmienić, co za tym idzie nie da się zmienić zamieszczonego w nich skrótu. W konsekwencji każdy może zaufać, że dany skrót zostanie tam na zawsze. Takie powiązanie kryptograficznego skrótu z gazetą działa jak znak czasowy. Transakcje zawarte w bloku, którego skrót ukazał się we wczorajszej gazecie, są uznawane za wcześniejsze od transakcji zawartych w bloku, którego skrót ukazał się w dzisiejszej gazecie. Dodatkowo, aby zachować integralność w sytuacji, gdy gazeta wyda dwa egzemplarze jednego dnia, lub jakiegoś dnia nie wyda wcale, do funkcji skrótu podajemy zarówno blok, jak i skrót z poprzedniego bloku, otrzymując w ten sposób kryptograficzne powiązany łańcuch bloków (ang. *blockchain*).

Wracając do przykładu z Alice, która chciałaby wydać dwa razy swoje bitcoiny. Rysunek 10.4 przedstawia hipotetyczny blockchain oparty o gazetę codzienną. Pierwszego dnia, Alice otrzymała 100 jednostek bitcoinów (dla uproszczenia pomijamy ich pochodzenie), transakcja trafia do pierwszego bloku, którego skrót trafia do gazety z pierwszego dnia. Alice uzgadnia z Bobem, że prześle mu 100 bitcoinów w zamian za telefon, następnie z Carol że również prześle jej 100 bitcoinów w zamian za samochód. Bob i Carol zgadzają się pod warunkiem, że ich transakcja pojawi się w bloku w gazecie z drugiego dnia. Alice próbuje



Rysunek 10.4. Blockchain oparty o gazetę codzienną

opublikować dwie transakcje, pierwszą do Boba, drugą do Carol. Bob akceptuje transakcję i wydaje telefon dla Alice; Carol widząc, że środki Alice zostały już przelane, nie akceptuje transakcji. Alice może próbować kolejnego dnia w gazecie z dnia trzeciego, lecz Carol przechodząc po łańcuchu rozpozna, że konto Alice, po transakcji z Bobem, jest puste. David przyjmie transakcje od Boba, ponieważ nakładając wszystkie (poprawne) transakcje otrzyma stan w którym Bob jest w posiadaniu 100 bitcoinów.

Nietrudno się jednak nie zgodzić, że waluta cyfrowa oparta o gazety nie jest najlepszym rozwiązaniem. Bitcoin osiągnął jednak ten sam mechanizm znakowania czasem przy użyciu zdecentralizowanej sieci p2p, oraz innowacyjnego podejścia do osiągania konsensusu za pomocą algorytmu dowodu pracy.

Dowód pracy w Bitcoinie polega na poszukiwaniu liczby, która w połączeniu z blokiem i podaniu do funkcji skrótu SHA-256 zwróci liczbę która zapisana binarnie rozpocznie się od zadanej długości zer. Każde dodatkowe zero wykładniczo zwiększa trudność zagadki. Wymagana liczba zer zależy od sumy mocy obliczeniowej całej sieci; jeśli kolektywna móc obliczeniowa rośnie, wzrasta liczba wymaganych zer, jeśli moc spada, wraz z nią liczba wymaganych zer na początku skrótu. Protokół Bitcoin stara się produkować bloki co 10 min — innymi słowy, co 10 min ten "globalny zdecentralizowany zegar" znakuje czasem aktualny blok.

Zmiana transakcji w bloku jest możliwa, ale wymaga ponownego obliczenia dowodu pracy dla zmodyfikowanego bloku. Co więcej, ponieważ każdy kolejny blok wskazuje na skrót poprzedniego bloku, dla każdego kolejnego bloku również trzeba policzyć nowy dowód pracy. Jednak to nie koniec, aby sieć przyjęła taką zmianę musi ona "wyprzedzić" aktualny łańcuch bloków; wyprzedzenie jest możliwe jeśli podmiot jest w stanie produkować nowe dowody pracy szybciej niż reszta sieci, stąd też założenie, że sieć jest bezpieczna tak długo, jak większość mocy obliczeniowej sieci jest kontrolowana przez uczciwe podmioty.

10.4.6 Motywacja ekonomiczna

Węzły chcą wykonywać dowody pracy i konsumować energię ponieważ za znalezienie zagadki czeka ich nagroda w postaci bitcoinów. Dopóki wartość nagrody jest większa niż koszt energii elektrycznej dopóty liczenie dowodów pracy — nazywane również kopaniem (ang. *mining*) — jest opłacalne i znajdą się ludzie którzy będą chcieli taka inwestycje podjąć.

Nagroda za każdy blok zmniejsza się w czasie, początkowo wynosiła 50 BTC, i co każde 210 tys. bloków (około cztery lata) zostaje zmniejszona o połowę, następuje tak zwane dzielenie (ang. *halving*). W latach 2012-2016 nagroda wynosiła 25 BTC, od 2016 do 2020 o połowę mniej — 12,5 BTC, aktualnie wynosi 6,25 BTC, a po roku 2140 wynosić będzie zero. Jak w takim razie węzły będą zmotywowane do liczenia dowodów pracy i zabezpieczania sieci? Poprzez system prowizji. Każda transakcja w sieci Bitcoin może zawierać "napiwek" dla górnika, jest on dobrowolny, jednak górnicy zachowując się racjonalnie wybierają transakcje, z których otrzymają największe wynagrodzenie. Im bardziej sieć obciążona, tym większe opłaty transakcyjne, tym większe wynagrodzenie dla górników. Mechanizm opłat transakcyjnych jest również istotny ze względu na ochronę sieci przed atakiem odmowy dostępu (ang. *deny-of-service*). Gdyby transakcje nie wymagały opłat, z łatwością można byłoby zalać sieć mikrotransakcjami. Opłaty transakcyjne sprawiają, że takie działanie jest nieopłacalne.

10.4.7 Podsumowanie

W ten sposób Bitcoin jako pierwszy osiągnął system płatności bez zaufanej trzeciej strony. W tym systemie sprzedawca nie musi ufać kupującemu, że ten nie wydał wcześniej swoich jednostek gdzie indziej. Sam może to zweryfikować w swojej lokalnej kopii blockchaina. Ma też pewność, że nikt nie wykona obciążenia zwrotnego, ponieważ jest to trudne obliczeniowo, co za tym idzie, bardzo kosztowne do wykonania.

Bitcoin zagwarantował, że liczba wszystkich jednostek w systemie nigdy nie przekroczy 21 mln bitcoinów. Jest jednak bardzo prawdopodobne, że liczba ta będzie malała; spowodowane jest to nieustanną utratą środków przez zgubione klucze prywatne oraz przelewów na nieistniejące konta. W teorii zmniejszająca się podaż i zwiększający się popyt prowadzić musi do wzrostu ceny. W praktyce jednak istnieje wiele czynników z powodu których nie musi się tak dziać.

10.5 Ethereum

Bitcoin osiągnął coś więcej niż system płatności w modelu p2p. Zbudował system bezstronnej, otwartej, neutralnej, motywowanej prawami wolnego rynku oraz zdecentralizowanej zaufanej trzeciej strony, której zaufanie nie leży w wierze w uczciwość, lecz w prawa matematyki.

Po kilku latach Vitalik Buterin, Gavin Wood oraz kilku innych twórców zauważyli, że potencjał technologii stojącej za Bitcoinem jest dużo większy niż sam transfer pieniędzy [12, 22].

Dotychczas, gdy ktoś chciał zmodyfikować lub rozszerzyć funkcjonalność Bitcoina, musiał zmodyfikować protokół i postawić całkowicie nową sieć, następnie przekonać górników, aby poświęcili swoją moc obliczeniową na liczenie dowodów pracy (co za tym idzie zabezpieczali sieć) dla jego projektu, a nie Bitcoina. Powodowało to dwa problemy, po pierwsze waluta taka musiała mieć już jakąś wartość, aby kopanie jej było opłacalne. Po drugie górnik, który przenosił swoją moc obliczeniową na kopanie innej kryptowaluty, zabierał ją z kopania Bitcoina, co za tym idzie obniżał jego bezpieczeństwo.

Ethereum rozwiązało ten problem tworząc platformę wykonawczą dla programowalnych transakcji, tzw. smart kontraktów. Podczas gdy transakcja w Bitcoinie stanowi o zmianie właściciela bitcoinów, smart kontrakt może być dowolnym programem wykonywanym w ramach transakcji. Transakcja może — podobnie jak w Bitcoinie — być zwykłym przelewem waluty, może mieć też inne znaczenie jak na przykład oddanie głosu w wyborach, zablokowanie środków w ramach lokaty, wzięcie pożyczki, wynajem krótkoterminowy samochodu, zmiana właściciela nieruchomości, nadanie uprawnień lekarzowi do wykonania obliczeń na naszych danych medycznych, obstawienie zakładu bukmacherskiego, i wiele innych.

Transakcje wykonywane są na wirtualnej maszynie Ethereum (ang. *Ethereum Virtual Machine*, EVM), która jest maszyną *quasi-Turing-skończoną* — pozwala na implementację dowolnego algorytmu wraz z rozgałęzieniami oraz pętlami; *quasi* oznacza, że nie występuje problem stopu — maszyna nie może się zapętlić w nieskończoność, gdyż każda instrukcja procesora konsumuje kryptowalutę; jeśli transakcja zużyje wszystkie dostępne środki, zostaje przerwana.

Na sieć Ethereum można patrzeć jak na publiczny klaster węzłów. na których uruchomiony jest JVM (*Java Virtual Machine*), do którego każdy może wysłać *bytecode* do wykonania. Zadanie wykonywane jest przez wszystkie węzły w klastrze; trwały stan (zapisywany między wykonaniami smart kontraktu) przechowywany jest w łańcuchu bloków; algorytm konsensusu zapewnia, że każdy węzeł w sieci posiada tą samą wersję łańcucha; użytkownik wysyłający *bytecode* musi również "podpiąć swoją kartę płatniczą", z której węzły konsumują środki za każdą instrukcje procesora. EVM w porównaniu do JVM, nie posiada dostępu do interfejsu sieciowego, ani innego interfejsu systemowego. Aby uniknąć wyścigów, EVM nie obsługuje wielowątkowości, a wykonywany smart kontrakt nie może komunikować się z niczym innym, niż inne smart kontrakty. EVM może zapisywać swój stan do dedykowanej dla niego przestrzeni w łańcuchu bloków, która również jest dodatkowo płatna.

Wykonanie smart kontraktu jest transparentne i możliwe do zweryfikowania. Ethereum stworzyło platformę do aplikacji, które posiadają wszystkie cechy bezpieczeństwa, transparentności, oraz odporności na cenzurę, jakie posiadał Bitcoin. Aplikacje tworzone w oparciu o Ethereum zostały nazwane Decentralized Applications (DApps), a całe podejście do tworzenia całkowicie zdecentralizowanych aplikacji zostało okrzyknięte mianem web 3. Aplikacje tego typu cechuje decentralizacja wszystkich komponentów: backendu, frontendu, przechowywania danych, wymiany wiadomości oraz systemu nazw. Zamiast tworzyć aplikację,

która komunikuje się z centralnym serwerem, który korzysta z prywatnej bazy danych, można stworzyć aplikację, która komunikuje się ze smart kontraktami, a dane przechowuje w sieciach p2p (np. IPFS [10] lub Ethereum Swarm [6]).

Takie podejście oferuje wiele zalet niedostępnych dla zcentralizowanych aplikacji:

1. Dostępność — sieci Ethereum działa jak ogromny klaster, awaria nawet wielu węzłów nie powoduje braku dostępności aplikacji. Co więcej, każdy użytkownik może uruchomić swój własny lokalny węzeł Ethereum przeprowadzając interakcje bezpośrednio z nim, uniezależniając się od jakiegokolwiek dostawcy.
2. Transparentności — każda transakcja, stan konta, smart kontrakt oraz jego stan w blockchainie jest dostępny publicznie dla każdego.
3. Niezmienialność — każda interakcja z blockchainem zostaje zapisana w nim na zawsze, zapewniając niezaprzeczalność. Każdy smart kontrakt zawsze będzie miał ten sam kod, zapewniając determinizm wykonywania. Każda treść będzie miała ten sam skrót, zapewniając jednoznaczność nazewnictwa.
4. Odporność na cenzurę — sieć Ethereum działa na globalną skalę, nie jest kontrolowana przez żadną organizację ani żaden kraj. Każdy węzeł jest finansowo motywowany do przetwarzania wszystkich transakcji bez względu na ich pochodzenie i przeznaczenie. Zatrzymanie sieci Ethereum jest tak samo ciężkie jak zatrzymanie każdej innej sieci p2p.

Wszystkie te zalety mają jednak swoją cenę. Tworzenie zdecentralizowanych aplikacji wymaga zmiany podejścia w projektowaniu, wytwarzaniu oraz wdrażaniu. Raz opublikowany smart kontrakt nie może być zmieniony, co powoduje duży nacisk na wytwarzanie kodu wysokiej jakości. Tworzenie smart kontraktów można porównać do tworzenia sprzętu komputerowego, naprawa każdego błędu wymaga kosztownych akcji serwisowych oraz wpływa negatywnie na renomę marki. Niektóre błędy są katastrofalne w skutkach i pociągają za sobą utratę środków, których nie da się odzyskać [3].

Używanie aplikacji tego typu oferuje dużo wolności, ale w zamian wymaga wiele odpowiedzialności. Użytkownicy są właścicielami swoich danych, w przypadku utraconego klucza prywatnego nie ma możliwości przywrócenia go, nie ma żadnego wsparcia technicznego, do którego można się zgłosić.

10.6 Otwarte problemy

Wiele aplikacji nie może istnieć w paradygmacie web 3 z powodu (1) słabej skalowalności (mierzonej w liczbie transakcji na sekundę), (2) wysokich opłat transakcyjnych, (3) braku regulacji prawnych, (4) słabej prywatności, (5) ograniczonej interoperacyjności pomiędzy systemami, (6) braku standaryzacji.

Skalowalność jest problematyczna, ponieważ jest związana z trylematem skalowalności [5, 7] który cechują trzy właściwości. System blockchainowy chciałby posiadać każdą z nich, ale można wybrać tylko dwie:

1. Skalowalność — kolektywna przepustowość przetwarzania transakcji całej sieci większa niż przepustowość jednego węzła w sieci.
2. Decentralizacja — węzeł sieciowy powinien być możliwy do uruchomienia na przeciętnym laptopie konsumenckim.
3. Bezpieczeństwo — sieć powinna być odporna na znaczną część (idealnie 50%) szkodliwych węzłów partycypujących w algorytmie konsensusu.

Większość blockchainów wybiera bezpieczeństwo, pozostawiając wybór między skalowalność a decentralizację. Tradycyjne blockchainy, jak Bitcoin czy Ethereum, wybierają decentralizację, kosztem skalowalności osiągając jedynie 3-15 transakcji na sekundę. Te które stawiają na skalowalność, podnoszą wymagania sprzętowe oraz wprowadzają "superwęzły", stając się mocno zcentralizowane.

Zwiększenie wydajności i obniżenie kosztów transakcyjnych próbuje się osiągnąć na kilka sposobów: wprowadzeniem *shardingu*, który sprawi, że każdy węzeł w sieci będzie przetwarzał nie wszystkie, lecz podzbiór transakcji w ramach swojego *shardu*; kolejnym pomysłem jest wprowadzenie rozwiązań drugiej warstwy, na której to węzły nie będą musiały komunikować się nieustannie z blockchainem, lecz wysyłać mu zbiorcze transakcje co jakiś czas. Rozwiązania drugiej warstwy można implementować na wiele sposobów, jednym z nich (najbardziej obiecującym [4]) jest technologia *zk-rollup* [8], która pozwala na weryfikację wykonanych transakcji bez konieczności wykonywania ich na każdym węźle oraz zapewnia wysoki poziom prywatności poprzez zastosowanie dowodów wiedzy zerowej (ang. *zero-knowledge proofs*).

Różne projekty blockchanowe rozwiązują każdy z tych problemów w nieco inny sposób, sprawiając, że środowisko to jest chaotyczne i brak mu standaryzacji; prawnikom natomiast przysparza wyzwanie precyzyjnego zdefiniowania technologii w kontekście prawnym.

10.7 Wnioski końcowe

Fenomenem technologii blockchain nie jest rozproszona i bezpieczna baza danych, nie jest nim również możliwość uruchamiania smart kontraktów; fenomenem jest platforma zdecentralizowanego zaufania umożliwiająca tworzenie aplikacji, które wcześniej mogły istnieć tylko w modelu z centralną zaufaną jednostką.

Kryptowaluty są jedynie pierwszą — a zarazem fundamentalną — aplikacją tej technologii. Są dowodem dojrzałości, wiarygodności i rzetelności. Pozwalają na finansowanie dalszych prac — elementu, bez którego wiele projektów open-source upada. Co najistotniejsze, zapewniają model bezpieczeństwa bazujący na teorii gier i prawach wolnego rynku.

Kolejnymi aplikacjami są usługi rządowe, internetowe wybory, tokenizacja udziałów w organizacjach (tzw. ICO), katalogi zasobów cyfrowych, instrumenty finansowe (tzw. DeFi), zakłady bukmacherskie, rynek energetyczny w modelu *peer-to-peer*, systemy nazw domen, a wiele z nich wciąż czeka na odkrycie.

Bibliografia

1. b-money. http://www.weidai.com/bmoney.txt, (Accessed on 06/08/2021).
2. Bit gold | satoshi nakamoto institute. https://nakamotoinstitute.org/bit-gold/, (Accessed on 06/08/2021).
3. The dao attack: Understanding what happened – coindesk. https://www.coindesk.com/understanding-dao-hack-journalists, (Accessed on 06/14/2021).
4. A rollup-centric ethereum roadmap. https://ethereum-magicians.org/t/a-rollup-centric-ethereum-roadmap/4698, (Accessed on 10/06/2021).
5. Sharding-faqs | ethereum wiki. https://eth.wiki/sharding/Sharding-FAQs, (Accessed on 06/14/2021).
6. Swarm. https://www.ethswarm.org/, (Accessed on 06/13/2021).
7. Why sharding is great: demystifying the technical properties. https://vitalik.ca/general/2021/04/07/sharding.html, (Accessed on 06/13/2021).
8. Zk-rollups - ethhub. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/, (Accessed on 10/06/2021).
9. Back, A., et al.: Hashcash — a denial of service counter-measure (2002).
10. Benet, J.: Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014).
11. Bordo, M.D., Kydland, F.E.: The gold standard as a rule: An essay in exploration. Explorations in Economic History (4), 423–464 (1995). https://doi.org/10.1006/exeh.1995.1019, https://www.sciencedirect.com/science/article/pii/S0014498385710194.
12. Buterin, V., et al.: Ethereum white paper. GitHub repository **1**, 22–23 (2013).
13. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI. vol. 99, pp. 173–186 (1999).
14. Good, D.: Individuals, interpersonal relations, and trust. Trust: Making and breaking cooperative relations pp. 31–48 (2000).
15. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. In: Concurrency: the Works of Leslie Lamport, pp. 179–196 (2019).
16. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. In: Concurrency: the Works of Leslie Lamport, pp. 203–226 (2019).
17. Lamport, L., et al.: Paxos made simple. ACM Sigact News (4), 18–25 (2001).
18. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep. (2019).
19. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). pp. 305–319 (2014).
20. Song, H., Sierakowski, T.: Wojna o pieniądz: prawdziwe źródła kryzysów finansowych. Wydawnictwo "Wektory"(2010), https://books.google.pl/books?id=jUuVpwAACAAJ.
21. Turner, J.H.: Face to face: Toward a sociological theory of interpersonal behavior. Stanford University Press (2002).

22. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger .
23. Zemła, Ł., et al.: Zaufanie. fundament społeczeństwa. Studia Politicae Universitatis Silesiensis (4-5), 322–328 (2009).

11. Najczęstsze problemy usługowych środowisk wdrożeniowych

Andrzej Sobecki

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
andrzej.sobecki@pg.edu.pl

Streszczenie
Zakres dostępnych obecnie rozwiązań informatycznych umożliwia znaczące usprawnienie procesu wytwarzania i dostarczania oprogramowania do klienta. Poprawna integracja środowiska wytwórczego pozwala wyeliminować szereg problemów dotyczących wewnętrznej współpracy zespołów developerskich. Zmiana architektur aplikacji z monolitycznych na rozproszone heterogeniczne zbiory usług wymaga innego podejścia do wdrażania usług. Dotychczasowe metody wydają się być nieefektywne i mogą istotnie zwiększać koszty operacyjne. Nowe typy narzędzi i bibliotek pozwalają wreszcie na pełną automatyzację procesu konfiguracji i wdrażania usług, utrzymywania ich wysokiej dostępności oraz dostarczają nowe mechanizmy zabezpieczania usług. Są one również dostosowane m.in. do wykorzystania chmur obliczeniowych, które oferują znacznie większa funkcjonalność niż jedynie zbiór połączonych ze sobą serwerów. W rozdziale opisano nowoczesne architektury oprogramowania, ich wymagania względem środowiska wdrożeniowego oraz zaprezentowano przykład takiego środowiska. Przy czym nie jest wymagane posiadanie chmury obliczeniowej, aby skorzystać z przedstawianych mechanizmów.

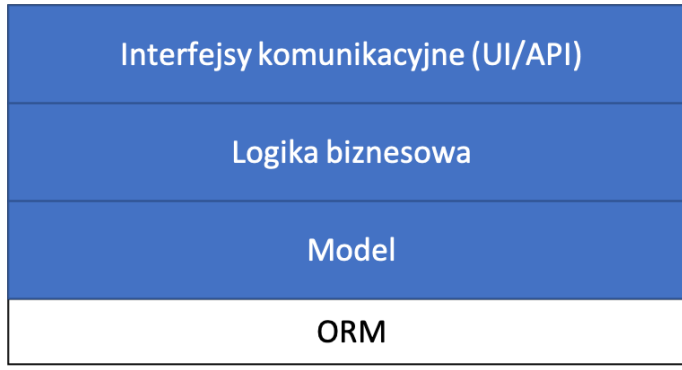
Słowa kluczowe: konteneryzacja, docker, swarm, kubernetes, gitlab, cqrs, mesh

11.1 Wprowadzenie

Jednym z kamieni milowych w technikach wytwarzania oprogramowania było przejście z paradygmatu programowania proceduralnego do podejścia obiektowego [20]. Obecnie większość systemów informatycznych wykorzystuje paradygmat obiektowy do tworzenia wirtualnych reprezentacji wybranego obszaru i organizacji kodu.

Przez wiele lat przyzwyczajeni zostaliśmy do wytwarzania aplikacji kompletnych, monolitycznych [13], w których całościowo podchodziliśmy do postawionego przed nami problemu (rys. 11.1). Takie podejście gwarantuje spójność przetwarzanych danych, a zachodzące w systemie procesy i utworzone relacje są przejrzyste dla programistów. Jednak stały rozwój systemów informatycznych,

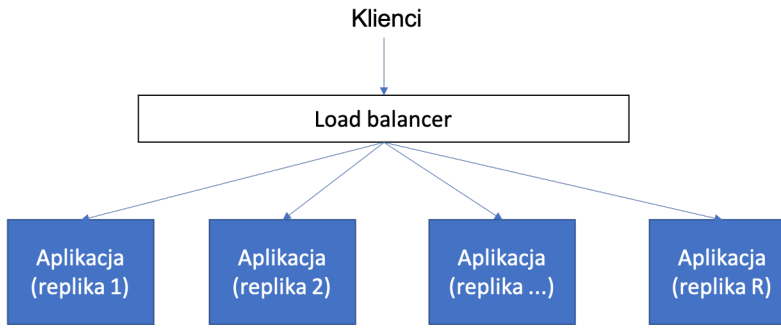
istniejących czasami od 20 lat, prowadzi za każdym razem do rozszerzania funkcjonalności i tworzenia ogromnych zestawów obiektów, bibliotek i funkcji. Takie środowiska są trudne w utrzymaniu i rozwoju [19]. Czasami do tego stopnia, że podejmowane są decyzje o konieczności napisania wszystkiego od początku.



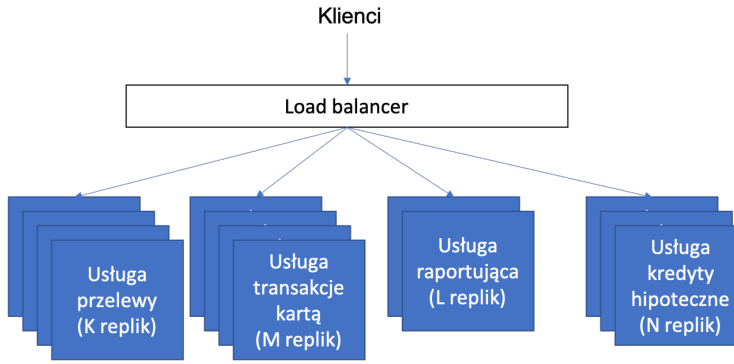
Rysunek 11.1. Architektura monolityczna

Jednak to niejedyne zagrożenie związane z utworzeniem monolitycznego rozwiązania. Wraz z rozwojem systemu zwiększa się zakres oferowanych przez niego funkcji, należy również obsługiwać większą niż pierwotnie planowano liczbę żądań i użytkowników. Naturalnym kierunkiem wydaje się replikacja systemu (rys. 11.2) i uruchomienie wielokrotnie kopii systemu, połączonej komponentem brzegowym typu *load balancer* [2]. Ostatecznie jednak okazuje się, że część usług jest nadmiernie obciążona (transakcje kartami płatniczymi) a inne jedynie w niewielkim stopniu (kredyty hipoteczne) lub okresowo (raporty z transakcji danym miesiącu). Jednak architektura monolityczna mimo wielu swoich zalet nie pozwala na podział aplikacji na poszczególne usługi (problem ograniczonej ziarnistości). Przez co stajemy przed wyzwaniem zmiany architektury systemu na usługową.

Rozważenie na początku architektury usługowej SOA [9] pozwala zaoszczędzić wielu problemów w przyszłości, kosztem intensyfikacji i komplikacji prac programistycznych na każdym etapie projektu. Nowoczesne architektury mikrousługowe [17] poza możliwością dostosowania ziarnistości komponentów pozwalają osiągnąć dodatkowe korzyści, niedostępne dla podejść monolitycznych (rys. 11.3). Dzięki możliwość swobodnego wyboru technologii do wykonania poszczególnych usług możemy zoptymalizować czas wytworzenia odpowiedniej logiki biznesowej. Innym przykładem jest możliwość stosowania ustalonych, asynchronicznych interfejsów komunikacyjnych wykorzystując komunikaty. Dzięki temu uniezależniamy kolejne implementacje usług od siebie, dając sobie szansę na podmianę usługi w przyszłości. Poprawnie wdrożona aplikacja może być aktualizowana bez czasowej niedostępności dla użytkownika końcowego.



Rysunek 11.2. Przykład replikacji dla aplikacji z architekturą monolityczną



Rysunek 11.3. Przykład replikacji dla aplikacji z architekturą mikroserwisową

Oczywiście proces zarządzania takim środowiskiem i wdrażanie aplikacji opartej na zestawie usług wymaga szczególnego podejścia. Bez specjalnych narzędzi, próba utworzenia i utrzymania takich aplikacji byłaby bardzo kosztowna. Istotną zmianą jaka dokonała się w kilku ostatnich latach jest możliwość konteneryzacji [11] usług (np. Docker), a nie wirtualizacji całych środowisk uruchomieniowych. W dalszej części opisane zostaną korzyści płynące z tego podejścia. Na rynku zaczęły pojawiać się narzędzia automatyzujące proces wdrażania aplikacji opartych na usługach, w tym dedykowane platformy jak np. Kubernetes [16]. Ponadto programiści i administratorzy otrzymali zestaw narzędzi do automatyzacji procesu konfiguracji systemów. Wreszcie zasoby obliczeniowe, które dotychczas oferowane były jedynie w formie VPS (ang. Virtual Private Server) lub serwerów dedykowanych w klastrach, są dostępne on-demand w środowiskach chmur obliczeniowych [3]. Jednak obecne chmury obliczeniowe oferują znacznie więcej m.in. możliwość konfiguracji przepływu danych, warstw zabezpieczeń, definiowania programowalnych bram API [21] czy nawet uruchamiania funkcji jako usług FaaS/Server-less architecture [5].

Żeby skorzystać ze wszystkich ułatwień jakie obecnie oferuje rynek informatycznych dla programistów należy poprawnie zidentyfikować główne problemy klasycznych systemów i środowisk wdrożeniowych. Następnie łatwiej będzie zrozumieć obecny kierunek zmian, wybrać narzędzia odpowiednie do swoich projektów i wykorzystać je jak najlepiej do optymalizacji kosztów operacyjnych. Dlatego zaproponowano następującą kolejność rozdziałów: główne założenia dla rozproszonych architektur usługowych, identyfikacja głównych problemów podczas wdrażania aplikacji i utrzymania środowiska wdrożeniowego, charakterystyka modelowego środowiska wdrożeniowego.

11.2 Rozproszone architektury usługowe

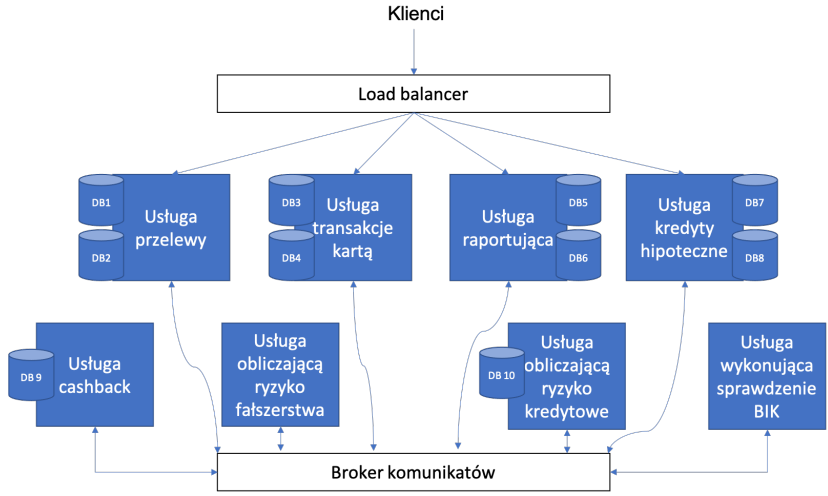
Podstawowym założeniem, o którym wspomniano wcześniej, jest świadomy podział aplikacji na komponenty i zdefiniowanie zależności pomiędzy nimi w formie standaryzowanych interfejsów komunikacyjnych API. Dzięki temu mamy możliwość określenia zakresu odpowiedzialności poszczególnych komponentów, zdefiniować schematy i zakres wymiany danych oraz osiągnąć większą odporność środowiska na starzenie się aplikacji. Ostatnia kwestia jest szczególnie istotna, przy założeniu że aplikacja będzie używana przez kilka lat i zakres oferowanych funkcji będzie rozszerzany. Proces wyodrębnienia z aplikacji usług składowych daje nam również możliwość podmiany technologii, w jakiej usługa została wykonana.

W niewielkich aplikacjach stosuje się wzorzec skryptu transakcji [10] do obsługi dedykowanych żądań zmieniających stan obiektów w bazie danych. W takich aplikacjach problem transakcyjności pokonuje się zazwyczaj z pomocą bibliotek dla warstwy ORM [8], korzystając przy tym z dobrodziejstw i wad pojedynczego, trwałego źródła danych. W przypadku aplikacji rozproszonych mamy również możliwość wykorzystania tzw. rozproszonych transakcji [7], ale ze względu na ich wady nie jest to podejście często wykorzystywane. Głównymi wadami tego podejścia jest niska efektywność, brak wsparcia przez wszystkich dostawców oprogramowania oraz obniżenie dostępności aplikacji (poniżej poziomu dostępności poszczególnych jej usług). Model rozproszonej transakcji zakłada bowiem realizację w formie synchronicznej, w której w jednym momencie muszą być dostępne wszystkie usługi biorące udział w transakcji.

Takie założenie naturalnie kłóci się z podejściem, które chcemy ostatecznie osiągnąć tzn. dostępność aplikacji mimo czasowej niedostępności niektórych jej usług. Wraz ze wzrostem liczby komponentów, które współtworzą naszą aplikację rozproszoną musimy być świadomi zwiększonego prawdopodobieństwa wystąpienia awarii któregoś z jej komponentów. W skrajnych sytuacjach musimy przygotować się na działanie naszej aplikacji, gdy zawsze któraś z jej usług nie jest obecnie dostępna. Może to wydawać się szalone, ale docelowo takie podejście zapewni nam bezproblemową możliwość aktualizacji poszczególnych usług bez konieczności ogłaszania prac serwisowych w naszym systemie.

Rozwój systemów rozproszonych obecnie zakłada wykorzystanie m.in. wzorca agregata [18] do separacji modeli, za które odpowiada usługa. Oznacza to, że

usługa ma jasno zdefiniowany zestaw obiektów za który odpowiada. Stan tych obiektów w ramach pojedynczego agregatu przechowywany jest w niezależnej bazie danych, a każdy agregat posiada własne systemy bazodanowe. Dzięki temu możemy zwiększyć efektywność zapytań do bazy danych, ponieważ jednocześnie możemy stosować wiele instancji systemów baz danych. Zakres agregata determinuje nam wielkość mikrousługi, tak więc w niektórych przypadkach nie musi ona wcale być mikro. Szczególnie przy niewłaściwym zdefiniowaniu podziałów odpowiedzialności.

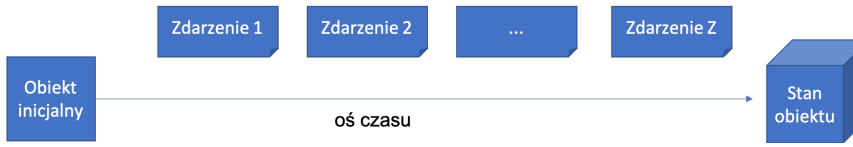


Rysunek 11.4. Przykład architektury aplikacji wykorzystującej wzorzec CQRS

Kolejnym elementem jaki wprowadza się do nowoczesnych architektur oprogramowania jest wzorzec CQRS [14] (ang. Command and Query Responsibility Separation) - rys. 11.4. Zakłada on podział usług już nie tylko ze względu na utworzony model domeny, ale również ze względu na rodzaj obsługiwanych żądań do danych. Wyróżnia się żądania modyfikujące dane tzw. komendy, które obsługuje odrębna usługa oraz żądania odczytu danych tzw. zapytania - obsługiwane również przez dedykowaną do tego usługę. Korzyścią jaka płynie z tego wzorca jest możliwość zastosowania dużo efektywniejszych systemów bazodanowych w usługach odpowiedzialnych za zapytania. Ponadto tego typu systemy mogą mieć inną architekturę niż systemy bazodanowe stosowane przez usługi do obsługi poleceń (no-SQL vs SQL). Wreszcie baza danych do odczytu może mieć strukturę danych dostosowaną do obsługiwanych zapytań, nie zważając na strukturę danych w usłudze do obsługi poleceń. Pozostaje kwestia efektywnego odwzorowywania zmian pomiędzy bazami danych.

Wzorzec CQRS stosuje się również w połączeniu ze wzorcem pozyskiwanie zdarzeń (ang. event sourcing) [12], ze względu na specyfikę przechowywania da-

nych o stanie obiektów (rys. 11.5). W klasycznych aplikacjach stan obiektu jest odzwierciedlony poprzez mapowanie obiektowo-relacyjne na tabele w bazie danych. Tego typu podejście jest często krytykowane przez twórców aplikacji, ze względu na wprowadzane ograniczenia dotyczące modelu i konieczność dostosowania się do ograniczeń bazy danych. Ponadto przechowywanie pojedynczego stanu obiektu w środowisku rozproszonym jest znacznie trudniejsze niż w klasycznych, monolitycznych aplikacjach.



Rysunek 11.5. Przykład działania wzorca event sourcing

Wzorzec pozyskiwanie zdarzeń zakłada, że w bazie danych trzymamy jedynie zdarzenia opisujące zmiany stanu obiektu (od obiektu inicjalnego). Zdarzenia muszą być na tyle informatywne, że nałożenie ich w odpowiedniej kolejności na obiekt inicjalny powoduje odtworzenie aktualnego stanu obiektu. Zdarzenia emitowane przez poszczególne komponenty systemu rozproszonego są w tym przypadku obligatoryjne, zatem wymagany jest również efektywny broker komunikatów. Wreszcie dlaczego CQRS jest tutaj istotny? Ze względu na problem w udzieleniu odpowiedzi na proste zapytanie tzn. zwróć mi obiekt typu X o kluczu głównym Y. W tradycyjnym systemie odpytalibyśmy system bazodanowy o tabelę reprezentująca stan obiektu typu X z kluczem Y. W systemach opartych na pozyskiwaniu zdarzeń nie ma prostej odpowiedzi na tak zadane pytanie. Trzeba stworzyć dedykowane usługi do obsługi zapytań, które odtworzą stan obiektu na podstawie zgromadzonych informacji o zarejestrowanych zdarzeniach.

W środowiskach rozproszonych istnieje jeszcze szereg rozwiązań, które są stosowane aby zwiększyć efektywność systemu. Warto wskazać jeszcze na stosowanie skoordynowanych sag [15] poprzez choreografię lub orkiestrację usług oraz stosowanie bezpieczników (ang. circuit-breaker) [4]. Pierwsze umożliwiają stworzenie mechanizmu do wykonywania rozproszonych transakcji asynchronicznych, wykorzystując komunikaty. Drugie rozwiązanie stosowane jest w połączeniu z monitoringiem stanu usług. Jego zadaniem jest wykrywanie braku dostępności pewnej usługi i minimalizację zjawiska propagowania błędu w aplikacji rozproszonej. Bezpiecznik może w przypadku awarii zwrócić np. stałą odpowiedź, dane z pamięci cache lub zablokować przesyłanie żądań dalej na pewien czas.

Tworząc aplikację w architekturze rozproszonej oczekujemy, że wraz ze zmieniającym się strumieniem żądań będziemy w stanie zwiększać lub zmniejszać liczbę wybranych komponentów naszej aplikacji w celu dostosowania jej przepustowości. Jednocześnie chcemy wyeliminować z tego procesu kosztowny zespół specjalistów, monitorujących 24/7 stan i przepustowość poszczególnych komponentów. Dlatego obecnie widać trend przechodzenia z prostych środowisk obli-

czeniowych tzn. jednego lub kilku serwerów połączonych siecią wewnętrzną na rzecz profesjonalnych chmur obliczeniowych. Oferują one bowiem narzędzia pozwalające nam monitorować środowisko i na tej podstawie elastycznie tworzyć lub usuwać maszyny obliczeniowe z określonymi zasobami.

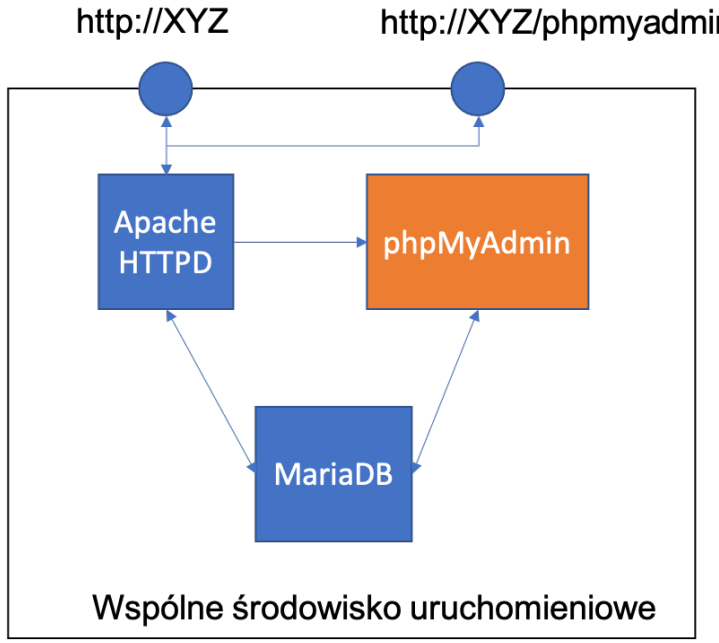
Obecnie chmury obliczeniowe oferują nam coraz więcej komponentów, które niedawno wymagały ręcznej konfiguracji i monitorowania. Takie komponenty oferowane są w modelu XaaS (ang. Everything as a Service) [6]. Dostawca środowiska kontroluje tego typu komponenty, zapewniając ich niezawodność. Ostatecznie otrzymujemy środowisko, które w sposób programowy pozwala nam elastycznie konfigurować dostępne w nim zasoby poprzez dedykowane API. Dzięki temu możemy cały proces zautomatyzować, a nawet przenosić lub replikować pomiędzy dostawcami zasobów. O ile nie zdecydujemy się na jedno z zamkniętych środowisk jak np. Amazon AWS czy Google Cloud.

11.3 Główne problemy związane z wdrażaniem aplikacji

W poprzednim rozdziale dokonano charakterystyki zmian jakie dokonały się w architekturach aplikacjach i starano się uzasadnić powód tych zmian oraz potencjalne korzyści z nich płynące. W tym rozdziale skupiono się na problemach jakie mogą wystąpić podczas konfiguracji środowiska dedykowanego do wdrażania systemów posiadających zarówno architekturę klasyczną (monolityczną) jak i rozproszoną (usługową).

Podstawowa metoda wdrażania aplikacji monolitycznej zakłada zastosowanie pojedynczej instancji aplikacji oraz usług zależnych tj. serwera bazodanowego, serwera cache itp. Środowiskiem do wdrożenia jest najczęściej pojedynczy wirtualny serwer (VPS) z systemem Linux. Ze względu na możliwość wykorzystania wielu domyślnych wartości konfiguracyjnych dla usług zależnych, proces przygotowania środowiska sprowadza się do zainstalowania pakietów oprogramowania oraz importu aktualnej wersji aplikacji. W zależności od technologii w jakiej wytworzono aplikację może ona być dystrybuowana w formie paczki ZIP (dla kodów HTML/CSS/JS/PHP itp. opartych na parserach), JAR/WAR (dla kodów JAVA wykorzystującej kompilator), DLL (dla kodów .NET Framework wyk. kompilator) i plików pomocniczych. Wydaje się, że takie środowisko jest niezawodne, a proces dostarczania nowej wersji sprowadza się do podmiany pliku lub plików z kodem aplikacji. Wbrew pozorom takie podejście generuje szereg problemów i w dłuższej perspektywie znacząco podraża koszty operacyjne.

Pierwszy problem dotyczy **braku izolacji** komponentów aplikacji i poszczególnych usług zależnych (rys. 11.6). Najprościej można to przedstawić na popularnej konfiguracji: Apache Httpd (serwer aplikacyjny np. dla kodu PHP), MariaDB (popularny serwer bazodanowy SQL) i phpMyAdmin (narzędzie pomocnicze umożliwiające manipulację bazą danych poprzez interfejs GUI). Każda z tych trzech usług wymaga innych bibliotek i posiada pewne podatności, których lista jest publikowana w Internecie. Załóżmy, że powyższa konfiguracja została przygotowana na publicznym serwerze VPS. Administrator udostępnił sobie dwa adresy URL, jeden dla wdrożonej aplikacji i jeden dla phpMyAdmin. W tym

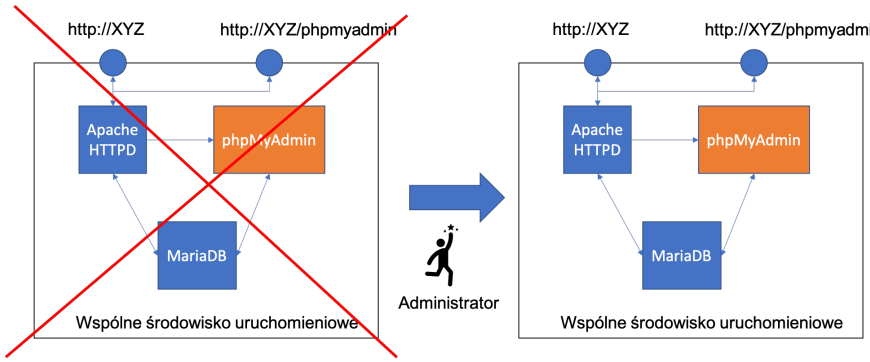


Rysunek 11.6. Przykład braku izolacji

przypadku każdy użytkownik sieci Internet może wywołać zarówno wdrożoną aplikację jak i phpMyAdmin z dowolnego komputera podłączonego do sieci Internet. Podatności naszej aplikacji nie są publikowane, zatem teoretycznie jesteśmy bezpieczni. Jednak potencjalny intruz może bez problemu odnaleźć podatności dla phpMyAdmin, a także narzędzia do wykorzystania tej podatności. Dzięki uzyskaniu dostępu do phpMyAdmina potencjalny intruz może dokonać dowolnych modyfikacji w bazie danych, ponadto może też pobrać wszystkie zapisane w niej informacje. Jeśli podatność okaże się krytyczna, to w wyjątkowych sytuacjach intruz może uzyskać dostęp do VPS z prawami root-a. Wiedząc, że usługi na tym serwerze są pozbawione jakiejkolwiek izolacji, nasza aplikacja może zostać podmieniona na dowolną inną treść.

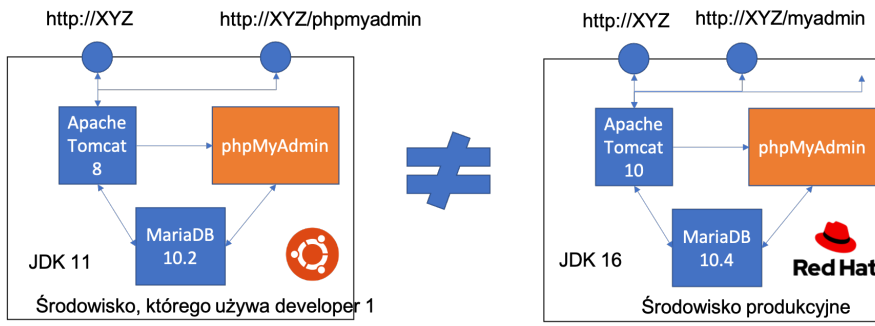
Brak izolacji skutkujący przejęciem kontroli nad wszystkimi usługami stanowi istotny problem. Jednak generuje on kolejny, równie poważny problem jakim jest przywrócenie środowiska wdrożeniowego do normalnego stanu. Dotychczas administrator na hipotetycznym serwerze wykonał kilkanaście komend w celu jego konfiguracji. Zatem będzie musiał odtworzyć z pamięci wykonane operacje tzn. ręcznie zaangażować się ponownie w proces konfiguracji serwera. Takie podejście niestety wydłuża czas, w którym nasza aplikacja jest niedostępna i nie generuje przychodów. Problemem w tym przypadku jest **brak lub ograniczona automatyzacja** (rys. 11.7). Nawet jeśli administrator przygotuje pewien zestaw skryptów powłoki (np. bash albo zsh) to nie ma żadnej gwarancji,

że te same skrypty będą działały bezbłędnie w systemie Ubuntu 10,11 i 12 oraz w dystrybucjach CentOS czy RedHat. Jakakolwiek zmiana dystrybucji systemu operacyjnego lub jego wersji może prowadzić do braku automatyzacji.



Rysunek 11.7. Przykład braku lub ograniczonej automatyzacji

W tym miejscu można by zaproponować zastosowanie obrazów maszyn wirtualnych, które dostarczyłyby kompletne środowiska dla aplikacji. Takie podejście również nie jest najlepsze ze względu na problemy w dystrybucji takiego środowiska, jak również w utrzymaniu go. Należy wspomnieć w tym miejscu, że rozpatrujemy problemy również pod względem integracji środowisk produkcyjnego i wytwórczego. Oczekujemy, że programiści mają wcześniej dostęp do konfiguracji środowiska docelowego, mogą je sobie odtworzyć na własnych komputerach i testować nowe funkcjonalności w tworzonej aplikacji. Wersjonowanie obrazów maszyn wirtualnych z użyciem narzędzi GitLab/GitHub jest praktycznie niemożliwe. Zatem należałoby utworzyć dodatkowy kanał dystrybucyjny dla środowiska. Nie jest to najlepsze rozwiązanie. To zagadnienie określać będziemy jako **problem wersjonowania konfiguracji środowiska**.



Rysunek 11.8. Przykład unifikacji i wersjonowania środowiska

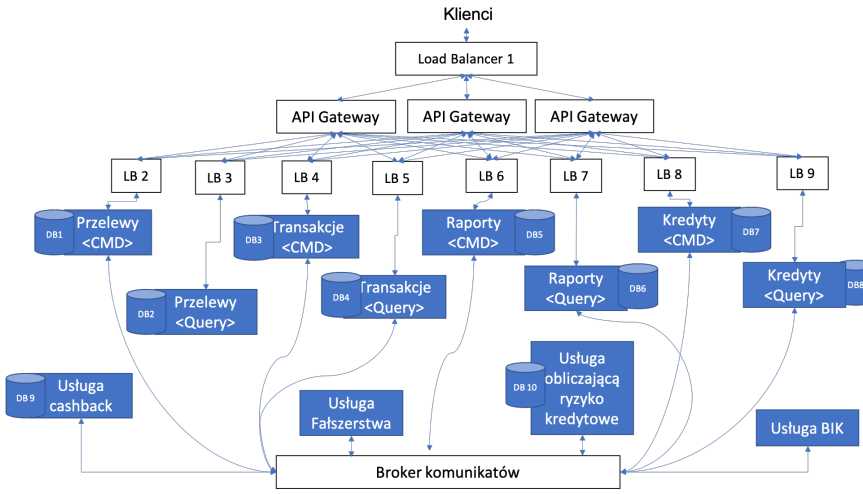
Kolejnym problemem będzie stosowanie dodatkowych zabezpieczeń, usług zależnych czy wreszcie specjalnych haseł i kluczy dedykowanych jedynie dla środowiska docelowego. Zatem istnieje duże prawdopodobieństwo, że w pewnym momencie środowiska developerskie i produkcyjne nie będą ze sobą zgodne. Ten problem określimy jako **brak unifikacji środowisk dev i prod** (rys. 11.8).

Następnie należy przedstawić problem **braku standaryzacji dla procesu dostarczania oprogramowania**. Początkowo administrator pobierał pewien zbiór plików z serwera i ręcznie podmieniał wersję aplikacji na nowszą. Trudno powiedzieć, czy podmieniał cały folder czy może wybrane pliki. Nie mamy żadnej gwarancji, że każdy administrator wykona tę czynność tak samo i w dowolnym czasie. Co w przypadku, gdy programiści dodali nową funkcjonalność, która wymaga rekonfiguracji serwera Apache httpd lub zainstalowania dodatkowej biblioteki? Wreszcie jak długo może potrwać proces wymiany wersji aplikacji i związana z tym niedostępność usługi dla klientów? W rozpatrywanym przypadku również trudno jest oszacować te wartości.

Jeśli aplikacja staje się popularna to prawdopodobnie środowisko wdrożeniowe zostanie rozbudowane np. do 3 serwerów VPS. Załóżmy, że usługi zależne zostały skonfigurowane na jednym z serwerów, a kopie aplikacje zostały uruchomione na pozostałych dwóch. Ponownie należy przypomnieć problem wersjonowania konfiguracji środowiska. W tym przypadku musielibyśmy dystrybuować co najmniej dwa obrazy maszyn wirtualnych. Warto jednak zastanowić się nad jeszcze jednym problemem, związanym z awarią serwera. Jeśli utracilibyśmy jeden VPS, na którym pracuje kopia aplikacji to nie stanowiłoby to dużego problemu. Jednak utrata serwera, na którym uruchomiono system bazodanowy powoduje brak możliwości oferowania usług przez naszą aplikację. Taki serwer w konfiguracji środowiska określa się jako SPF (ang. Single Point of Failure). W takiej sytuacji administrator powinien jak najszybciej zorganizować nowy serwer i skonfigurować na nim usługę, lub uruchomić usługę na pozostałych dwóch serwerach i dokonać rekonfiguracji aplikacji - w zakresie adresów do połączenia z bazą danych. Ponownie występuje tutaj problem braku automatyzacji, braku unifikacji środowisk dev i prod oraz ponownie braku izolacji. Jak efektywnie zarządzać uruchamianiem usług i aplikacji na serwerach jeśli mamy ich setki, a ponadto liczba dostępnych serwerów może się dynamicznie zmieniać? Mając do dyspozycji jedynie przedstawione procedury, firma nie będzie prawdopodobnie mogła poradzić sobie z tym problemem ze względu na koszty utrzymania aplikacji. Warto podkreślić, że powyższe problemy dotyczyły stosunkowo prostej architektury monolitycznej. W przypadku zwiększenia ziarnistości aplikacji, należałoby uwzględnić dodatkowe uwarunkowania jak np. różne cykle wytwórcze, mnogość wykorzystywanych technologii czy efektywna replikacja wybranych typów usług. Ponadto poszczególni klienci mogą sobie życzyć dedykowane środowiska świadczenia usług, zatem liczba problemów raczej nie ulegnie redukcji.

11.4 Charakterystyka modelowego środowiska wdrożeniowego

Przedstawienie modelowego środowiska wdrożeniowego wymaga zdefiniowania demonstracyjnej aplikacji internetowej. Załóżmy, że nasza aplikacja nie posiada architektury monolitycznej lecz nowoczesną architekturę mikroserwisową np. CQRS. Oznacza to, że składa się ona z pewnej liczby np. N komponentów (usług), z których każdy może być wykonany w różnej technologii. W tej grupie wyróżnimy usługi odpowiadające za przetwarzanie komend, które będą posiadać własne relacyjne źródła danych (systemy bazodanowe SQL). Wyróżnimy również pewien podzbiór usług odpowiadający za wykonywanie zapytań, dla których przypiszemy dedykowane nierelacyjne źródła danych (system bazodanowe no-SQL). Załóżmy również, że komunikacja pomiędzy usługami zachodzi przy wykorzystaniu asynchronicznego stylu interakcji żądanie-odpowiedź lub publikuj-asynchr. odpowiedź. To założenie prowadzi nas do uwzględnienia w naszej architekturze brokera komunikatów. Wymienione usługi zależne tj. systemy SQL, no-SQL i broker komunikatów tworzą zbiór dodatkowych M usług. Przykład takiej architektury przedstawiono na rys. 11.9.



Rysunek 11.9. Przykład rozpatrywanej architektury aplikacji

Należy również uwzględnić usługę bramy API (ang. API Gateway), który może jednocześnie pełnić rolę równoważnika obciążenia (ang. load-balancer). Rolą bramy API będzie otrzymywanie żądań od klientów i przekierowanie ich do odpowiedniej usługi z logiką biznesową. Brama API poza rozróżnianiem endpointów może uwzględniać również dodatkowe kryteria jak np. nazwisko użytkownika, numer PESEL itp. Na podstawie dodatkowych kryteriów żądanie może być

kierowane do innej podgrupy usług. Taką właściwość nazywamy partycjonowaniem, pozwoli nam ona osiągnąć większą efektywność i dostępność naszej aplikacji. Przykładem partycjonowania może być obsługa żądań od użytkowników: Andrzeja i Zbigniewa. Zakładając, że istnieją 2 partycje (A-M, N-Z), żądania od użytkownika Andrzej trafią do usług obsługujących pierwszą partycję, a od Zbigniewa trafią do usług obsługujących drugą partycję.

Zaproponowana architektura aplikacja wykorzystuje wszystkie trzy wymiary sześcianu skalowania tj.: możliwość uruchamiania wielu instancji tych samych usług (replikacja), skalowanie aplikacji przez podział odpowiedzialności usług oraz partycjonowanie danych.

Wdrożenie i utrzymanie przedstawionej aplikacji internetowej wymaga odpowiedniej konfiguracji środowiska developerskiego i wdrożeniowego. Błędy na etapie projektowania środowisk i brak automatyzacji podstawowych czynności mogą doprowadzić do wydłużenia procesu dostawy oprogramowania do klienta oraz wydłużenia czasów niedostępności aplikacji.

Podstawowym wymaganiem dla tworzonego środowiska jest stosowanie w jak największym stopniu rozwiązań informatycznych, które oferują otwarte i standaryzowane API. W przeciwnym razie na pewnym etapie rozwoju napotkamy problem określany jako uzależnienie od dostawcy (ang. vendor lock-in). W wielu przypadkach można uzyskać ten sam efekt korzystając z rozwiązań dostępnych u konkretnego dostawcy. Jednak po pewnym czasie migracja środowiska do innego dostawcy może być niemożliwa lub bardzo kosztowna. Starajmy się unikać tego typu rozwiązań.

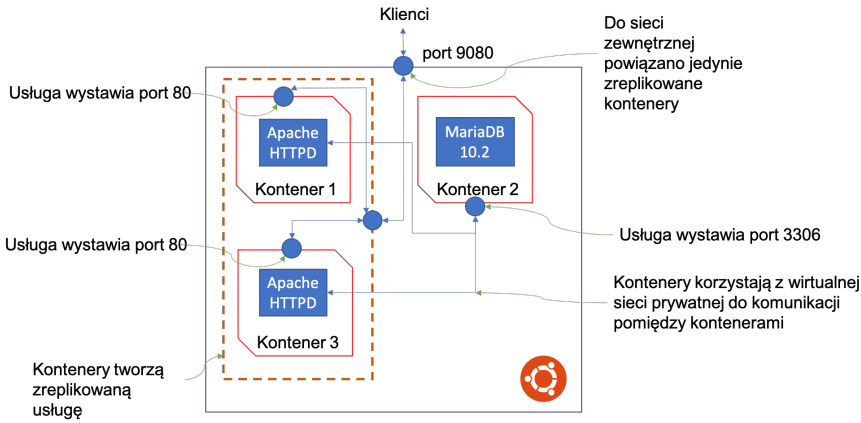
Opis środowiska podzielono ze względu na wymienione wcześniej problemy, zaznaczając jednocześnie jak wprowadzone usprawnienia pozwalają pokonać wymienione trudności. Zakładamy przy tym, że oprogramowanie jest rozwijane z użyciem systemu kontroli wersji *GIT* i korzystamy z otwartego środowiska wersjonowania kodu *GitLab*. Ponadto zakładamy, że w projekcie wykorzystujemy określone narzędzie do konteneryzacji usług np. *docker* lub *podman* [1].

11.4.1 Izolacja usług

Problem braku izolacji możemy pokonać w prosty sposób, korzystając z narzędzi do konteneryzacji usług. Narzędzia tego typu dostarczają mechanizmy do programowego wyznaczenia granic pomiędzy kontenerami. Środowisko świadczone w każdym z kontenerów posiada minimalny zestaw komponentów potrzebnych do uruchomienia poszczególnych usług. Kontener uruchamiany jest niezależnie od innych i usługa uruchomiona w nim nie ma możliwości komunikacji z usługą z innego kontenera.

Oznacza to również, że przejęcie kontroli nad dowolną skonteneryzowaną usługą nie oznacza możliwości swobodnej komunikacji z wszystkimi innymi usługami znajdującymi się poza tym kontenerem. Poziom izolacji kontenerów można programowo definiować i udostępniać im niezbędne zasoby zewnętrzne np. konkretne dyski sieciowe (wolumeny) lub umożliwiać komunikację przez jedną z wirtualnych sieci łączących kontenery.

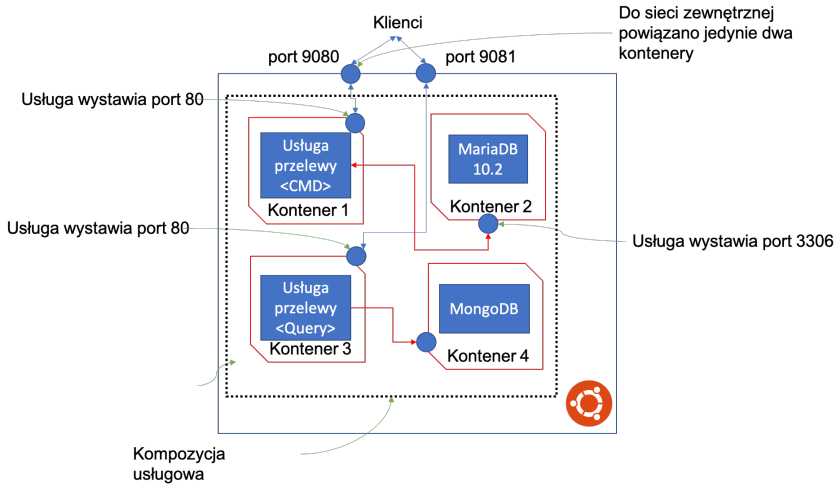
Izolacja usługi uruchomionej w kontenerze od środowiska hosta ma dodatkową zaletę związaną z kontrolą nad portami jakie są udostępniane na zewnątrz. Jeśli w środowisku hosta uruchomimy trzy kontenery z usługą Apache Httpd (port 80/443), to samodzielnie musimy zdefiniować tzw. wiązania portów (ang. binding) do hosta. Ponadto jeśli usługa w kontenerze oferuje kilka portów, to mamy kontrolę nad tym które z tych portów zostaną wystawione do hosta, a w rezultacie mogą być wywołane przez inne serwery. Przykład takiego rozwiązania, w którym wykorzystano konteneryzację przedstawiono na rys. 11.10.



Rysunek 11.10. Przykład izolacji usług, wraz z replikacją i powiązaniem z siecią zewnętrzną

Na podstawie dostępnych narzędzi do programowania izolacji możemy również tworzyć tzw. kompozycje usług. Najczęściej kompozycją jest zbiór usług wzajemnie od siebie zależnych, tworzących zestaw funkcjonalny. W naszym przypadku taką kompozycją może być zestaw komponentów potrzebnych do świadczenia usługi realizującej wzorzec agregata przyjmujący żądania typu **polecenie**. W ramach kompozycji można zdefiniować uruchomienie komponentu aplikacji oraz systemu bazodanowego. Usługi wewnątrz kompozycji mogą domyślnie komunikować się ze sobą. Przykład środowiska wykorzystującego kompozycję przedstawiono na rys. 11.11.

Tworzenie kompozycji ma dodatkową zaletę w postaci możliwości sparametryzowania sposobu uruchamiania usług poprzez zestaw zmiennych środowiskowych. Korzystając z kompozycji można dokonać izolacji opisu środowiska, konfiguracji usług od samego kodu usługi potrzebnego do jej uruchomienia. Dzięki temu możliwa jest standaryzacja procesu dostarczania oprogramowania.



Rysunek 11.11. Przykład wykorzystania kompozycji do opisania zestawu usług CQRS

11.4.2 Standaryzacja procesu dostarczania oprogramowania

Kolejnym problemem jest przekazywanie nowych wersji usług od developerów do środowisk testowych, a następnie produkcyjnych. Chcemy, aby proces w miarę możliwości był jak najbardziej zautomatyzowany. Wiemy już, że poszczególne usługi możemy definiować jako kompozycje, będące pewnym opisem sposobu uruchamiania usługi.

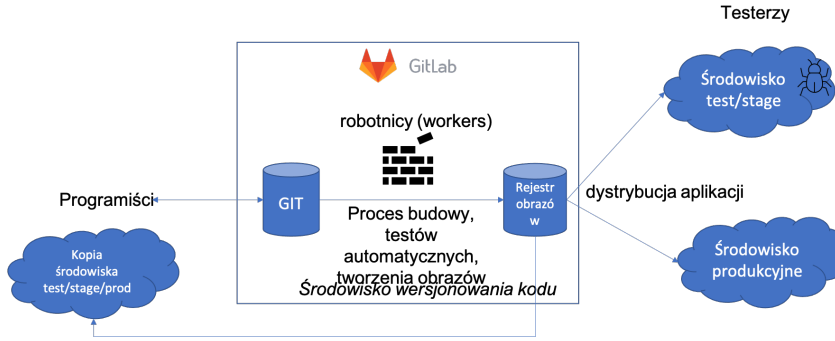
W tym miejscu wykorzystamy kolejną właściwość konteneryzacji jaką są obrazy. Nazywamy tak gotowe, skonfigurowane środowisko docelowe, w którym świadczona jest określona usługa. Obraz jest uniwersalny dla potencjalnych odbiorców. Oznacza to, że może być pobrany przez klienta typu docker/podman i uruchomiony na dowolnym komputerze o określonej architekturze (np. amd64, arm). Innymi słowy każdy programista, serwer w środowisku testowym lub produkcyjnym mogą skorzystać z tej samej wersji obrazu. Po uruchomieniu go otrzymają kontener, który będzie dokładnie tak samo skonfigurowany. Omawiane obrazy wykorzystywane do tworzenia kontenerów nie są tożsame z obrazami stosowanymi do uruchamiania maszyn wirtualnych.

Należy zauważyć, że w rozważanym przypadku chcemy dystansować się od kompilacji aplikacji w środowisku testowym lub produkcyjnym. Dlatego też warto rozważyć utworzenie dla każdej naszej usługi opisu procesu kompilacji/translacji kodu naszej aplikacji oraz opisu przygotowania środowiska dedykowanego dla tej aplikacji.

Obydwa wymienione problemy możemy rozwiązać, korzystając z tzw. pipelineów oferowanych na platformie GitLab. Poprzez utworzenie pliku **.gitlab-ci.yml** mamy możliwość automatyzacji zadań wykonywanych po zarejestrowaniu w repozytorium nowej wersji kodu (zdarzenie po wydaniu komendy *git push*). Celem tego procesu może być kompilacja aplikacji, wykonanie automatycznych testów

i przygotowanie obrazu. Zadania zapisane w tym pliku mogą być różne dla gałęzi master i pozostałych gałęzi. Wszystkie zadania są realizowane przez tzw. robotników (ang. workers) zarejestrowanych na platformie GitLab.

Dla kompletności rozważanego środowiska brakuje jeszcze centralnego punktu dystrybucji obrazów. W tym celu również wykorzystamy platformę *GitLab*. Oferuje ona domyślnie dla każdego projektu dedykowane rozwiązanie jakim jest **rejestr obrazów**. Wystarczy wskazać w pliku **.gitlab-ci.yml**, aby gotowe obrazy były wysyłane do rejestru powiązanego z projektem. Zalecanym rozwiązaniem jest stosowanie etykiet przy nazwach obrazów. Przykładowo obrazy tworzone z gałęzi master powinny posiadać etykietę *latest*, a z gałęzi dev etykietę *dev*. Dzięki temu łatwiej będzie wskazywać w kompozycjach usług jakiego typu obrazy chcemy wdrażać. W zależności od rodzaju środowiska (developerskie, produkcyjne) na poziomie definiowania kompozycji można rozróżnić źródło obrazów. Przykład takiego środowiska przedstawiono na rys. 11.12.



Rysunek 11.12. Przykład konfiguracji środowiska w celu standaryzacji procesu dostarczania aplikacji

11.4.3 Unifikacja środowisk dev i prod

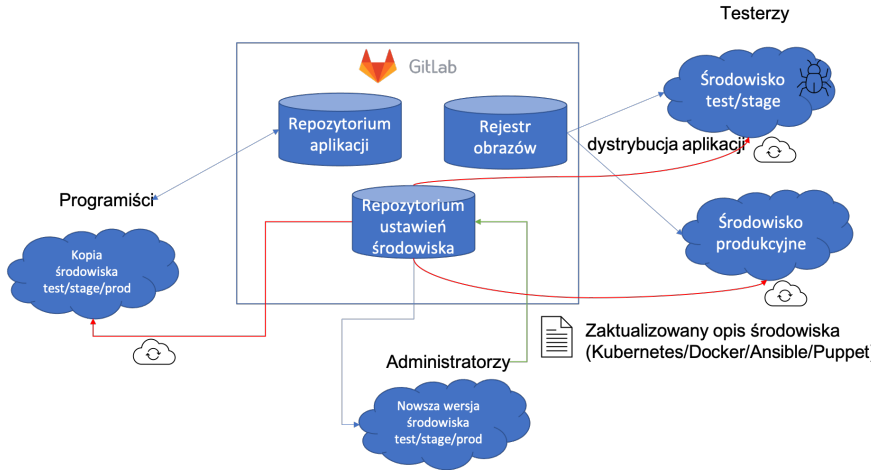
Zastosowanie kompozycji, wykorzystujących obrazy do tworzenia kontenerów pozwala rozwiązać problemy związane z uruchamianiem usług w różnych konfiguracjach sprzętowych i systemowych. Pobranie z repozytorium plików opisujących kompozycje usługowe pozwala odtworzyć dokładne warunki w jakich będą pracowały usługi w określonym środowisku.

W tym przypadku każdy programista będzie mógł na własnym komputerze odtworzyć błąd jaki wystąpił podczas testowania aplikacji w środowisku developerskim lub użycia aplikacji przez klienta w środowisku produkcyjnym.

Poprzez zastosowanie konteneryzacji, kompozycji i obrazów udało się wyeliminować problemy związane z tworzeniem notatek opisujących sposób konfiguracji środowiska, zapisywaniem historii komend, niezgodności wersji oprogramowania itd.

11.4.4 Wersjonowanie konfiguracji środowiska

Kolejnym wyzwaniem było wersjonowanie opisu konfiguracji środowiska wdrożeniowego. Dotychczas zredukowaliśmy opis środowiska uruchamiania usług do definicji kompozycji. Taka definicja domyślnie opisana jest plikiem tekstowym *docker-compose.yml* o strukturze YAML. Format tekstowy powoduje, że taki plik jest niewielki i dobrze wersjonuje się w środowisku kontroli wersji GIT.



Rysunek 11.13. Przykład realizacji koncepcji wersjonowania opisu środowiska

Zatem uzyskanie najnowszego opisu środowiska wdrożeniowego sprowadza się do pobrania pojedynczego pliku z repozytorium. Przed kolejnym uruchomienie środowiska wystarczy ściągnąć zmiany z serwera (*git pull*) i ew. pobrać nowsze wersje obrazów, wskazanych w pliku konfiguracyjnym kompozycji (*docker-compose pull*). Tak proste komendy nie powodują konieczności zaglądania do rozległej dokumentacji i śledzenia zmian wprowadzonych przez administratorów w ostatnich dwóch tygodniach. Przykład omawianej koncepcji przedstawiono na rys. 11.13. W literaturze osiągnięcie takiej właściwości środowiska uruchomieniowego dla usług określa się jako IaaC (ang. Infrastructure as a Code).

11.4.5 Ograniczona automatyzacja

Pozostał jeszcze problem ograniczonej automatyzacji środowiska, które w przypadku wystąpienia awarii wymaga reakcji ze strony administratora. Obecnie, zaproponowane środowisko wymaga minimalnego nakładu pracy ze strony administratora. Obejmuje ono zalogowanie się na serwer i zainstalowanie narzędzi wymaganych do przeprowadzenia konteneryzacji usług.

Istnieje pewne prawdopodobieństwo popełnienia błędu w tym procesie dlatego dobrym rozwiązaniem będzie skorzystanie z dobrodziejstw chmur oblicze-

niowych. W takich środowiskach mamy możliwość dynamicznego tworzenia i usuwania serwerów dostępnych do uruchamiania naszych usług. Poszczególne serwery będziemy nazywać instancjami (zgodnie z OpenStack). W środowisku *Amazon Web Service* takie serwery nazywa się EC2 (ang. Elastic Compute Cloud). Niezależnie od tego podczas uruchamiania instancji należy wskazać podstawowy obraz systemowy (dla AWS jest to AMI). Obrazy systemowe nie są tożsame z omawianymi obrazami środowisk skonteneryzowanych.

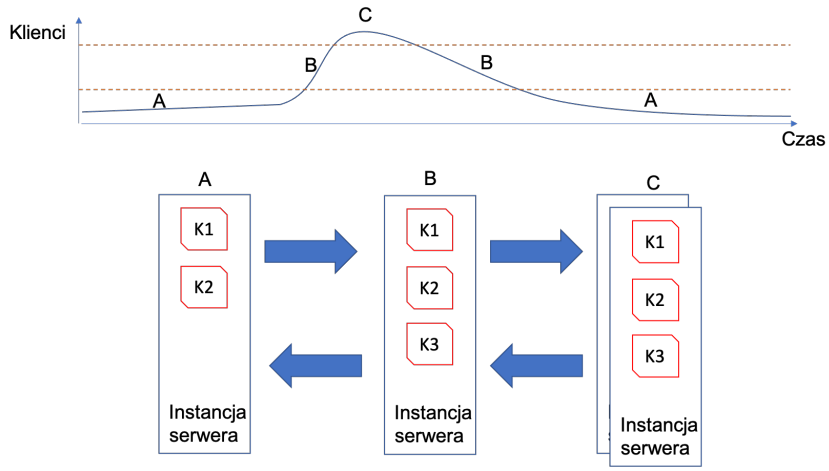
Obraz systemowy określa parametry systemu hosta oraz dostępne w nim biblioteki. W celu utworzenia obrazu systemowego należy przygotować jedną wzorcową instancję, na której zainstalowano i skonfigurowano wszystkie niezbędne komponenty. Następnie należy w środowisku chmury obliczeniowej wskazać taką instancją i zażądać przygotowania obrazu systemowego (lub AMI). Następnie dla każdej kolejnej instancji będzie można wskazać odpowiedni obraz systemowy i proces konfiguracji środowiska nie będzie już problemem.

Proces tworzenia instancji jakkolwiek jest prosty, to również wymaga czasami uwagi ze strony administratorów. W szczególności, gdy w trakcie dnia dynamicznie zmienia się strumień żądań od klientów. Przykładowo oferujemy serwis internetowy do analizy technicznej danych giełdowych. Zapotrzebowanie na tego typu usługi jest funkcją czasu. Dlatego w godzinach porannych lub wieczornych zapotrzebowanie na nasze usługi jest zazwyczaj mniejsze, podobnie jak w weekendy. W środowiskach chmur obliczeniowych najczęściej ponosimy koszty za wykorzystywane zasoby, zatem m.in. za liczbę i rodzaj uruchomionych instancji. W celu obniżenia kosztów operacyjnych powinniśmy utrzymywać minimalną, wymaganą liczbę instancji - tzn. niewiele wyższą od aktualnego zapotrzebowania. Oznacza to, że przy braku automatyzacji tego obszaru, należy przewidzieć konieczność oddelegowania doświadczonej osoby do zarządzania zasobami.

Jednak środowiska chmur obliczeniowych oferują nam narzędzia do automatyzacji procesu tworzenia, uruchamiania i wyłączania instancji. Do tego celu przygotowano odpowiednie API, które można wywoływać aby zmieniać stan i liczbę wykorzystywanych instancji. API może być wywoływane automatycznie przez dedykowaną usługę, która monitoruje aktualny strumień żądań. W środowiskach komercyjnych można wykorzystać jedną z dostępnych (dodatkowo płatnych) usług jak np. AWS ELB (ang. Elastic Load Balancer). Oferują one opisaną funkcjonalność i pozwalają na monitorowanie obciążenia serwerów, ruchu przychodzącego itd. oraz na tej podstawie mogą automatycznie podejmować decyzję o wymaganej liczbie instancji. Przykład zastosowania automatycznego skalowania kontenerów i serwerów przedstawiono na rys. 11.14.

Dotychczas przedstawione środowisko pozwala zautomatyzować liczbę i rodzaj dostępnych instancji oraz standaryzuje sposób dostarczania usług. Jednak pozostały jeszcze dwie kwestie: automatyczne radzenie sobie z awarią usługi, awarią instancji oraz wdrażanie nowych wersji usług.

Problem awaryjności można rozwiązać poprzez integrację zasobów obliczeniowych tj. instancji jakie są dostępne dla naszej aplikacji. Domyślnie instancje będą połączone ze sobą poprzez wirtualne interfejsy sieciowe, ale nie jest to rozwiązanie wystarczające. Należy zintegrować instancje w taki sposób, aby możliwe było



Rysunek 11.14. Przykład automatycznego skalowania kontenerów i serwerów względem obciążenia

radzenie sobie z wymienionymi awariami i w możliwie jak największym stopniu wykorzystać dostępne zasoby.

W tym celu możemy utworzyć dodatkową warstwę odpowiedzialną za zarządzanie dostępnymi zasobami w klastrze, która wspiera technologię konteneryzacji usług. Na rynku istnieje kilka platform jakie można wykorzystać w tym celu. Wyróżnić można dwie podstawowe grupy technologiczne: *docker swarm* oraz *kubernetes*. Ogólna zasada ich działania jest zbliżona, ale częściej w środowiskach komercyjnych stosuje się technologię *kubernetes*. Głównie ze względu na rozbudowane mechanizmy konfiguracji środowiska i łatwą jego rozszerzalność o nowe komponenty. Jednak dla zachowania kompletności opisu modelowego środowiska skupimy się na prostszej technologii jaką jest *docker swarm*. Należy zauważyć, że w jednym i drugim przypadku osiągniemy zakładane korzyści poprzez eliminację omawianych problemów. Jednak instalacja i konfiguracja klastra *docker swarm* jest znacznie prostsza i możliwa do przeprowadzenia dla większości użytkowników systemu konteneryzacji *docker*. Dodatkowo, przygotowane pliki konfiguracyjne opisujące kompozycje mogą w przyszłości zostać przekonwertowane do formatu zgodnego z platformą *kubernetes*. Takie, etapowe wdrażanie klastra do konteneryzacji usług jest znacznie bezpieczniejsze i pozwala rozłożyć koszty zmian w środowisku wdrożeniowych. Programiści również będą szczęśliwsi, gdy będą mogli uruchamiać środowisko na swoich komputerach bez konieczności obserwowania dedykowanej platformy testowej, znajdującej się na osobnych serwerach.

Uruchomienie klastra obliczeniowego *docker swarm* sprowadza się do wydania dwóch komend:

- *docker swarm init –advertise-addr 192.168.100.X* – włączenie trybu swarm (klastra) i ustawienie adresu rozgłoszeniowego dla pozostałych maszyn na adresie 192.168.100.X.
- *docker swarm join –token PEWIEN-TOKEN 192.168.100.X:2377* – dołączenie do klastra kolejnego serwera.

Środowisko domyślnie składa się z jednego węzła w roli **manager** i pozostałych w roli **worker**. Węzeł zarządzający kontroluje dostępne serwery, zasoby, uruchomione kontenery. Zaleca się zwiększenie liczby węzłów tego typu, aby uniknąć w swojej architekturze węzłów **SPF**.

Po uruchomieniu klastra w środowisku obliczeniowym uzyskujemy lepszą kontrolę nad dostępnymi zasobami. Taki klaster sam wykryje awarię zarówno poszczególnych węzłów jak i kontenerów w nim uruchomionych. W pierwszym przypadku uruchomiona zostanie automatyczna procedura relokacji kontenerów uruchomionych na wadliwym serwerze. Oznacza to, że bez ingerencji administratora kontenery zostaną przeniesione na jeden z pozostałych serwerów. W drugim przypadku (awarii kontenera) platforma automatycznie rozpocznie proces uruchamiania dokładnie tego samego kontenera. Oznacza to, że już tak podstawowa wersja środowiska oferuje znaczące obniżenie kosztów operacyjnych oraz skrócenie czasu niedostępności usług. Oczywiście należy pamiętać o ulotności danych w kontenerach. Jeśli chcemy zachować pewien stan plików należy do kontenera dołączyć wolumen, który zabezpieczy pliki przed awarią serwera i relokacją kontenera.

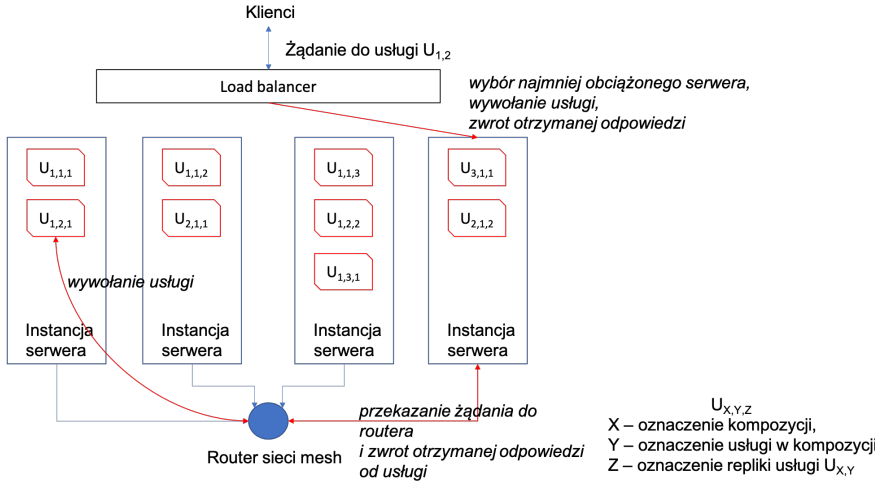
Dodatkową korzyścią jaką można uzyskać jest alokacja kontenerów ze względu na specyfikację serwera. W przypadku, gdy serwery różnią się pod względem konfiguracji sprzętowej (np. posiadają więcej RAMu, karty graficzne, szybsze dyski itp.) możemy oznaczyć je dodatkowymi etykietami. Następnie w konfiguracji kompozycji wskazujemy etykiety, których bezwzględnie wymagamy od serwera na którym zostanie uruchomiony kontener.

Kolejnym zagadnieniem jest zarządzanie replikami usług. Jak wspomniano wcześniej dobrze jest utrzymywać więcej niż jedną kopię usługi. W klastrach typu *swarm* możemy samodzielnie definiować liczbę replik per usługa w kompozycji lub zdefiniować usługę jako globalną. W pierwszym przypadku klaster automatycznie dba o to, aby uruchomiona była zdefiniowana liczba kontenerów. W drugim przypadku usługa zarządcy klastra uruchamia po jednej kopii usługi na każdym dostępnym serwerze, również gdy dołączony zostanie nowy serwer do klastra.

W tym przypadku warto podkreślić jeszcze jedną zaletę takiego środowiska tj. sieć typu mesh. Wyobraźmy sobie, że dokonując replikacji usług musielibyśmy ręcznie śledzić stan dostępnych kontenerów i ich adresy IP (wraz z portem). Następnie w bramie API/load-balancerze należałoby ręcznie korygować wpisy, dostosowując się do aktualnego stanu środowiska. Takie rozwiązanie kłóci się z naszymi założeniami odnośnie zmniejszenia kosztów operacyjnych poprzez m.in. automatyzację. Dlatego sieć typu mesh jest idealnym rozwiązaniem opisanego problemu. Polega ona na tym, że wołając dowolny z węzłów klastra uzyskujemy dostęp usługi mimo, że nie jest ona uruchomiona konkretnie na wy-

woływanym serwerze. Następuje routowanie żądania do odpowiedniego serwera. W rzeczywistości uzyskujemy również dodatkową korzyść związaną z równoważeniem obciążenia przez samą platformę. Wybór serwera, do którego żądanie zostanie skierowane można kontrolować i równoważyć tym samym liczbę żądań przetwarzanych przez określone kontenery.

Jak najlepiej wykorzystać zalety sieci typu mesh? Na serwerze/serwerach oferujących bramę API należy wskazywać pulę serwerów klastra, a nie pojedynczy serwer. W przypadku oprogramowania **nginx** możemy w tym celu wykorzystać ustawienie *upstream* i zdefiniować pulę serwerów do obsługi określonego żądania. Dzięki temu uodpornimy naszą aplikację na awarię dowolnego serwera z klastra. Przykład działania sieci typu mesh przedstawiono na rys. 11.15. Należy podkreślić, że aktualne rozmieszczenie kontenerów może się zmieniać, a wykorzystanie sieci typu mesh pozwala łatwo rozwiązać problem lokalizacji usługi.



Rysunek 11.15. Przykład działania sieci typu mesh

Warto w tym miejscu zwrócić uwagę na różnice w platformach *swarm* oraz *kubernetes*. W tym drugim przypadku porty hostów jakie można wykorzystać znajdują się w puli **30000 - 32767**. Usługi nie mogą przyjmować portów spoza tego zakresu. Ponadto nie można kierować żądań do serwerów typu **manager**.

11.5 Podsumowanie

Przedstawiony kierunek rozwoju architektur aplikacji jednoznacznie uzasadnia potrzebę wykorzystania narzędzi do efektywniejszego zarządzania kodem i wdrażania aplikacji w środowiskach skonteneryzowanych. Modelowe środowisko jakie zostało opisane może być z powodzeniem zastosowane w większości ma-

łych i średnich aplikacji, również tych posiadających pewien dług technologiczny utrudniających wdrażanie i rozwój aplikacji.

Korzyści wynikające z zastosowania konteneryzacji oraz środowisk wspierających konteneryzację przewyższają nakład pracy związany z przygotowaniem dodatkowych opisów kontenerów oraz ich kompozycji. Zamiana przestarzałej dokumentacji środowiska na zestaw skryptów działających w każdym środowisku jest dodatkowym argumentem za przeprowadzeniem zmian w swoim warsztacie pracy.

Nie należy zapominać o mądrym wyborze stosowanych bibliotek i środowisk oferujących API w otwartym standardzie. Dobrym przewodnikiem po tego typu rozwiązaniach jest CNCF [1] (ang. Cloud Native Computing Foundation). Udostępniają oni na swojej stronie interaktywną mapę takich rozwiązań w podziale na obszary odpowiedzialności.

Kolejnymi zagadnieniami, nad którymi należy się zastanowić jest uruchomienie centralnej platformy do monitorowania parametrów usług (np. Prometheus i Grafana), platformy do rejestrowania błędów usług (np. Sentry) czy regulowania zasad bezpieczeństwa w klastrze (np. Open Policy Agent). W dalszych etapach istotne może być zastosowanie managerów kompozycji (np. Helm), którzy pozwalają instalować na klastrze gotowe kompozycje usług zależnych. Proces konfiguracji chmury obliczeniowej może być zautomatyzowany, korzystając ze skryptów Ansible. Ewentualnie możemy powołać swoje własne rozproszone środowisko obliczeniowe, integrując serwery w modelu MaaS (ang. Metal as a Service), korzystając z oprogramowania MAAS.

Niniejsze opracowanie prezentuje jedną z wielu możliwych konfiguracji środowiska, rozwiązującego zasygnalizowane problemy. Jego celem było przedstawienie kierunku rozwoju technologii informatycznych i dostępnych obecnie możliwości.

W tym miejscu warto podkreślić, że Katedra Architektury Systemów Komputerowych uruchomiła środowiska testowe i produkcyjne, w których wykorzystano opisane rozwiązania technologiczne. Dotychczasowe rozwiązania wykorzystujące wirtualizację zasobów w formie maszyn wirtualnych był bardzo kłopotliwe i drogie w utrzymaniu, oferując przy tym niższą efektywność i dostępność uruchomionych aplikacji. Środowiska skonteneryzowane są dostępne dla studentów realizujących na katerze projekty inżynierskie oraz prace magisterskie, a także dla rozwiązań produkcyjnych tworzonych przez zespoły naukowe. Są one wykorzystywane również w trakcie realizacji zajęć dydaktycznych m.in. z Biznesu elektronicznego, Architektury Usług Internetowych, Serwisów internetowych .NET, Rozproszonych Systemów Wysokiej Wydajności.

Obecnie Katedra posiada klaster oparty na oprogramowaniu MAAS, na którym uruchomiono środowisko GitLab, Kubernetes oraz Docker Swarm. Każdy pracownik katedry może dostarczyć aplikację do uruchomienia w opisanym klastrze. W tym celu wystarczy przygotować pojedynczy plik z opisem kompozycji oraz zapewnić dostępność obrazów wskazanych w kompozycji (poprzez rejestr obrazów). Znacząco uprościło to proces wdrażania i zarządzania dziesiątkami

[1] `https://www.cncf.io`

aplikacji, które dotychczas były oferowane na serwerach katedralnych. Środowisko nie powstałoby bez wsparcia i doświadczenia Pana dr Tomasza Boińskiego, który podjął się roli wdrożenia i konfiguracji takiego klastra.

Bibliografia

1. Abraham, S., Paul, A.K., Khan, R.I.S., Butt, A.R.: On the use of containers in high performance computing environments. In: 2020 IEEE 13th International Conference on Cloud Computing (CLOUD). pp. 284–293. IEEE (2020).
2. Al Nuaimi, K., Mohamed, N., Al Nuaimi, M., Al-Jaroodi, J.: A survey of load balancing in cloud computing: Challenges and algorithms. In: 2012 second symposium on network cloud computing and applications. pp. 137–142. IEEE (2012).
3. Antonopoulos, N., Gillam, L.: Cloud computing. Springer (2010).
4. Aquino, G., Queiroz, R., Merrett, G., Al-Hashimi, B.: The circuit breaker pattern targeted to future iot applications. In: International Conference on Service-Oriented Computing. pp. 390–396. Springer (2019).
5. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., et al.: Serverless computing: Current trends and open problems. In: Research Advances in Cloud Computing, pp. 1–20. Springer (2017).
6. Banerjee, P., Friedrich, R., Bash, C., Goldsack, P., Huberman, B., Manley, J., Patel, C., Ranganathan, P., Veitch, A.: Everything as a service: Powering the new information economy. Computer **44**(3), 36–43 (2011).
7. Chen, G.: Distributed transaction processing standards and their applications. Computer standards & interfaces **17**(4), 363–373 (1995).
8. De Troyer, O., Casteleyn, S., Plessers, P.: Using orm to model web systems. In: OTM Confederated International ConferencesÓn the Move to Meaningful Internet Systems". pp. 700–709. Springer (2005).
9. Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., Newling, T.: Patterns: service-oriented architecture and web services. IBM Corporation, International Technical Support Organization New York, NY (2004).
10. Fowler, M.: Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch. Addison-Wesley (2012).
11. Kukade, P.P., Kale, G.: Auto-scaling of micro-services using containerization. International Journal of Science and Research (IJSR) **4**(9), 1960–1963 (2015).
12. Pacheco, V.F.: Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices. Packt Publishing Ltd (2018).
13. Ponce, F., Márquez, G., Astudillo, H.: Migrating from monolithic architecture to microservices: A rapid review. In: 2019 38th International Conference of the Chilean Computer Science Society (SCCC). pp. 1–7. IEEE (2019).

14. Rajković, P., Janković, D., Milenković, A.: Using cqrs pattern for improving performances in medical information systems. In: CEUR Workshop Proceedings. CEUR-WS. vol. 1036 (2013).
15. Rudrabhatla, C.K.: Comparison of event choreography and orchestration techniques in microservice architecture. International Journal of Advanced Computer Science and Applications **9**(8), 18–22 (2018).
16. Sayfan, G.: Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes. Packt Publishing Ltd (2019).
17. Sill, A.: The design and architecture of microservices. IEEE Cloud Computing **3**(5), 76–80 (2016).
18. Taibi, D., Lenarduzzi, V., Pahl, C.: Architectural patterns for microservices: A systematic mapping study. In: CLOSER. pp. 221–232 (2018).
19. Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., Gil, S.: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: 2015 10th Computing Colombian Conference (10CCC). pp. 583–590. IEEE (2015).
20. White, G., Sivitanides, M.: Cognitive differences between procedural programming and object oriented programming. Information Technology and management **6**(4), 333–350 (2005).
21. Zhao, J., Jing, S., Jiang, L.: Management of api gateway based on microservice architecture. In: Journal of Physics: Conference Series. vol. 1087, p. 032032. IOP Publishing (2018).

12. Problemy jakości w metodach Agile

Jarosław Kuchta

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki,
Politechnika Gdańska, Gdańsk
j.kuchta@eti.pg.edu.pl

Streszczenie
Zwinne metody wytwarzania osiągnęły w zawrotnym tempie niebywały sukces. Według różnych doniesień od 50 do 70% firm IT stosuje metody zwinne na stałe lub okazjonalnie[1]. Jednak znaczna część firm stosuje wybiórczo praktyki zalecane przez Agile[2]. Jakie to praktyki? Jakie problemy występują przy ich stosowaniu i jak firmy radzą sobie z tymi problemami? Jak wpływają na jakość wytwarzanego oprogramowania? Jakie są warunki krytyczne dla zapewnienia jakości produktu? Na te pytania odpowiemy w niniejszym rozdziale.

Słowa kluczowe: Agile, practices, quality, issues

12.1 Co to jest jakość oprogramowania?

Jakość oprogramowania jest definiowana i rozumiana na dwa sposoby [1]:

1. jako spełnienie wyspecyfikowanych wymagań,
2. jako zaspokojenie potrzeb i oczekiwań klientów i użytkowników.

Pierwsze rozumienie jest mocno uproszczone i zawodne. Jeśli w oprogramowaniu brak jakiejś funkcji, to programista może się usprawiedliwiać: „klient nic o tym nie mówił". Takie podejście ignoruje model wymagań Kano [18], według którego wymagania dotyczące oprogramowania można podzielić na pięć kategorii:

1. Wymagania niezbędne, których są ujmowane w specyfikacji jako najważniejsze. Ich spełnienie jest warunkiem koniecznym, aby produkt został zaakceptowany przez klientów.
2. Wymagania „jednowymiarowe", które niekoniecznie znajdują się w specyfikacji, ale ich spełnienie jest konieczne, aby klienci byli zadowoleni.

[1] https://www.consultancy.eu/news/4153/half-of-companies-applying-agile-methodologies-practices https://hygger.io/blog/why-agile-is-so-popular-in-project-management

[2] https://assets.kpmg/content/dam/kpmg/pl/pdf/2019/10/pl-raport-kpmg-pt-project-management-zarzadzanie-projektami-w-przedsiebiorstwach-dzialajacych-w-polsce.pdf

3. Wymagania „atrakcyjne", które niekoniecznie znajdują się w specyfikacji, ale ich spełnienie daje zadowolenie klientów.
4. Wymagania „neutralne", które nie mają żadnego wpływu na zadowolenie klientów.
5. Cechy niepożądane – których występowanie w produkcie powoduje niezadowolenie klientów.

Najwięcej kłopotu przysparzają wymagania z grupy 2 i 3. Wymagania z grupy 2 są znane dla klienta, ale mogą być nieznane dla wykonawcy. Z pewnych względów klient może o nich nie mówić (np. może je uważać za oczywiste). Przykładem może być zapisywanie kopii roboczej dokumentu przy jego edycji. Wymagania z grupy 3 pochodzą nie od klienta, a z dbałości wykonawcy o zadowolenie klienta z produktu. Np. klient raczej nie będzie oczekiwać podpowiedzi przy podawaniu adresu dostawy dla sklepu internetowego. Jednak dobry programista będzie wiedział, że takie podpowiedzi skracają czas składania zamówienia i postara się wypełniać formularz zamówienia wartościami domyślnymi.

12.2 Praktyki Agile a jakość oprogramowania

Metodyki Agile opierają się dobrych praktykach. Niektóre praktyki (np. związane z testowaniem) są bezpośrednio związane z jakością produktu. Inne (np. częste dostarczanie działającego oprogramowania) dotyczą raczej organizacji procesu i wówczas wpływają na jakość produktu pośrednio. W tym rozdziale przyjrzymy się, jak poszczególne praktyki Agile wpływają na jakość powstającego oprogramowania, jakie problemy występują przy realizacji praktyk i jak te problemy mogą być rozwiązywane.

Praktyki Agile można podzielić przypisując je do odpowiednich poddziedzin wiedzy o inżynierii oprogramowania:

1. Praktyki związane z zarządzaniem i organizacją.
2. Praktyki związane z wymaganiami i planowaniem.
3. Praktyki związane z projektowaniem systemowym.
4. Praktyki związane z kodowaniem i integracją oprogramowania.
5. Praktyki związane z testowaniem i wdrażaniem.

Inny podział praktyk Agile został zaproponowanych w monografii „*Agile Software Development Quality Assurance*" [25], gdzie praktyki podzielono na pięć kategorii dotyczących rozwiązywanych problemów [19]:

1. Praktyki rozwiązujące problemy ze zbieraniem wymagań i planowaniem.
2. Praktyki umożliwiające reagowanie na zmieniające się wymagania.
3. Praktyki dotyczące współpracy, komunikacji, pracy zespołowej
4. Praktyki zapewniające szybkie i częste dostarczanie produktu do klienta.
5. Praktyki polegające na ciągłym ulepszaniu produktu.

W tab. 1. pokazano przypisanie praktyk Extreme Programming [28] do powyższych kategorii. Nie wszystkie praktyki XP stosowane są w innych metodykach (np. Scrum). Jest to zaznaczone symbolem „(–)". Jeśli praktyka może być stosowana w innych metodykach, ale nie jest jawnie zalecana, to jest ona oznaczona symbolem „(?)".

Część praktyk Agile ma bezpośredni wpływ na jakość produktu. Tu najważniejsze są praktyki związane z pozyskiwaniem wymagań i testowaniem rozwiązania. Trzeba jednak pamiętać, że jakość procesu wytwarzania w znacznym stopniu wpływa na jakość produktu. Ponieważ proces Agile w znacznym stopniu polega na współpracy i komunikacji, więc praktyki w zakresie zarządzania i organizacji też mają znaczny wpływ na jakość. Poniżej przedstawimy kilka wybranych praktyk, których wpływ na jakość produktu jest największy, a z drugiej strony co do których występują największe problemy ze stosowaniem. Najczęściej zgłaszane są praktyki związane z organizacją procesu, niezrozumieniem zasad Agile ze strony kierownictwa i z brakiem zaangażowania klienta w pracę zespołu. Trzeba jeszcze zwrócić uwagę na niedocenianie potrzeby wydobywania wymagań od klienta oraz na niezrozumienie potrzeby przeprowadzania testowania akceptacyjnego przez użytkowników

12.3 Zarządzanie i organizacja

Manifest Agile wprowadził rewolucję do zarządzania projektami [14]. Rzeczywiście sygnatariusze Agile Manifesto postawili na ludzi, zespoły ludzkie, ich samoorganizację i zaangażowanie. Wcześniej w tym zakresie dominowała metoda *zarządzania przez cele* (ang. *Management By Objectives*), zwanej też *zarządzaniem przez wyniki* (ang. *Management By Results*) [11]. W tej metodzie opisanej przez Druckera w latach pięćdziesiątych ubiegłego wieku [12] skuteczne zarządzanie procesem w przedsiębiorstwie składa się z pięciu kroków:

1. ustalenia celów organizacji przez kierownictwo,
2. wyznaczenia konkretnych celów działania dla pracowników,
3. monitorowania prac,
4. oceny prac,
5. nagradzania pracowników osiągających wyznaczone cele.

Zwolennicy Agile odrzucili to tradycyjne podejście, dając programistom większą swobodę pracy, a jednocześnie większą swobodę organizowania swojej pracy. W ślad za tym musi jednak iść zmiana nastawienia do pracy samych programistów, którzy nie są już tylko pracownikami firmy wytwarzającej oprogramowanie, ale twórcami oprogramowania.

12.3.1 Zespół jako całość

W Agile podmiotem tworzącym oprogramowanie jest *zespół deweloperów*. Deweloper oprogramowania musi mieć zarówno umiejętności programisty, jak i testera, analityka i projektanta, a także do pewnego stopnia zdolności kierownicze,

Tablica 12.1. Klasyfikacja praktyk Agile

Praktyka	Problemy wymagań i planowania	Zmienność wymagań	Współpraca i komunikacja	Szybka i częsta dostawa	Ciągłe ulepszanie produktu
Zarządzanie i organizacja					
Zespół jako całość			√		
Samoorganizacja zespołu			√		
Otwarta przestrzeń pracy			√		
Codzienne spotkanie na stojąco			√		
Stały rytm pracy				√	
Monitorowanie szybkości projektu				√	
Programowanie w parach (–)			√		
Przenoszenie ludzi (–)			√		
Ciągłe podtrzymywanie projektu			√	√	
Przegląd i retrospekcja procesu			√		
Wymagania i planowanie					
Opowieści użytkownika	√	√	√		
Klient na miejscu (w zespole)	√	√	√		√
Ograniczone planowanie	√	√	√		
Częste wydania, krótkie iteracje	√	√	√	√	√
Priorytetyzacja potrzeb klienta	√	√	√		
Planowanie oparte na szacowaniu	√	√	√		
Dodatkowe iteracje dla poprawy jakości	√	√	√		√
Projektowanie systemowe					
Prostota rozwiązań			√	√	
Metafora systemu (?)			√		
Sesje projektowania (?)	√		√	√	
Rozwiązania próbne (?)	√	√		√	
Unikanie nadmiarowych funkcjonalności				√	
Odraczanie decyzji projektowych		√		√	√
Kodowanie i integracja					
Refaktoryzacja				√	√
Standardy kodowania			√	√	√
Programowanie w parach (–)			√	√	
Kolektywna własność kodu			√	√	
Ciągła integracja				√	√
Testowanie i wdrażanie					
Podejście TDD				√	√
Automatyczne testy jednostkowe				√	√
Testowanie przez zespół				√	√
Pokrycie kodu testami				√	√
Zaliczanie testów przed każdym wydaniem					√
Wydzielone środowisko wdrożenia testowego					√
Testowanie regresyjne					√
Testowanie akceptacyjne					√

a na pewno umiejętności współpracy w grupie. Kod napisany przez dewelopera nie stanowi jego własności ani w sensie prawnym, ani w sensie intelektualnym. Kod musi być pisany tak, aby mógł być modyfikowany przez każdego z członków zespołu (co wyraża się w odrębnej zasadzie kolektywnej własności kodu). To wymaga też zespołowego podejścia do odpowiedzialności – każdy z członków zespołu jest odpowiedzialny za cały zespół.

Trzeba zauważyć, że takie podejście stanowi wyzwanie dla ludzi – zarówno dla deweloperów, jak i dla kierownictwa firmy. Od deweloperów wymaga znacznie większego spektrum umiejętności niż to było w metodach klasycznych. W tych metodach był osobny zespół programistów, osobny projektantów, osobny testerów. W zespołach zwinnych każdy z członków zespołu jest angażowany i w programowanie, i w projektowanie, i w testowanie (chociaż pewna specjalizacja jest możliwa). Stanowi również wyzwanie dla kierownictwa firm, bo Agile wymaga często kompetencji wykraczających poza możliwości informatyków [23].

Od kierownictwa firmy Agile wymaga zaufania do zespołu, do jego umiejętności merytorycznych, wewnętrznej motywacji i zdolności do samoorganizacji. To też jest wyzwanie dla kierownictwa, któremu trudno zagwarantować klientom realizację wszystkich wymagań w określonym czasie. Dlatego zrozumienie zasad organizacji zwinnej ze strony klienta jest tak istotna. To też wprowadza ograniczenia w możliwościach stosowania metod Agile do tych projektów, w których pewne opóźnienie nie spowoduje fiaska całego projektu. Jest to trudne do uzyskania w projektach mocno zbiurokratyzowanych, np. wykonywanych na zamówienie publiczne lub finansowanych z ograniczonych czasowo programów krajowych czy międzynarodowych.

12.3.2 Samoorganizacja zespołu

Samoorganizacja pracy zespołu jest jednym z filarów metodyk zwinnych. Cóż to oznacza? W Agile nie ma roli kierownika projektu! Nie ma osoby, która mogłaby rozdzielać zadania, określić terminy i egzekwować ich wykonanie. Lider zespołu nie pełni takiej roli. Może on owszem nadawać kierunek pracom zespołu, ale na zasadzie przekonywania pozostałych członków zespołu. Również właściciel produktu w Scrumie nie ma kompetencji kierowniczych.

W Agile zespół sam sobie określa zadania do realizacji na najbliższą iterację. Owszem właściciel produktu określa priorytety opowieści, może, a nawet powinien określić cel Scrumu. Ostateczna decyzja należy jednak do członków zespołu. Każdy z nich przyjmuje na siebie pewne zobowiązanie. Powinien to robić świadomie, aktywnie uczestnicząc w szacowaniu pracochłonności.

Co można zrobić, jeśli ktoś z zespołu nie ma wystarczająco dużo samodyscypliny, a może po prostu umiejętności, aby podołać zadaniom, które na siebie przyjmuje? Cierpi na tym cały zespół. Może być tak, że ktoś inny mu pomoże, że ktoś inny przejmie część obowiązków na siebie. Jednak nie może to się odbywać odgórnie, na zasadzie polecenia kierownika projektu, bo takiego nie ma. Jeśli ktoś nie potrafi współpracować z zespołem, to często jest z niego eliminowany. W firmach regularnie (np. corocznie) odbywają się oceny pracowników

przez kierownictwo, które decydują o ewentualnych podwyżkach i awansach. Jeśli ktoś przeszkadza w pracy zespołu, to zespół może się zwrócić do kierownictwa organizacji o przeniesienie takiej osoby [15].

W samoorganizacji zespołu, a także w ustawieniu relacji między zespołem a kierownictwem organizacji pomaga Scrum Master. Jest to rola unikatowa w Scrumie, w innych metodykach nie występuje. Jednak Scrum Master nie ma kompetencji do rozwiązywania konfliktów, do rozstrzygania, do podejmowania decyzji. Jego zadaniem jest mentoring, uświadamianie wszystkim interesariuszom i aktywnym uczestnikom procesu wytwarzania, jaka jest specyfika podejścia Agile. Wraz ze wzrostem doświadczenia organizacji w stosowaniu Agile, rola Scrum Mastera staje się coraz mniej potrzebna.

Czy jednak to, że zespół ma się samoorganizować oznacza, że kierownictwo firmy ma w ogóle się nie interesować organizacją zespołów? Jaka tu jest rola kierownictwa? Jaka jest rola lidera zespołu? Rolą kierownictwa jest organizowanie właściwego środowiska do pracy zespołu. Środowiska ekonomicznego, intelektualnego, psychologicznego. To znaczy zapewnianie środków potrzebnych do pracy, określanie celu, do którego zespół powinien dążyć, tworzenie atmosfery wyzwalającej siły tkwiące w zespole. Lider zespołu nie jest formalną rolą metodyk Agile, ale pozycja lidera często się kształtuje w sposób naturalny. Liderem może być główny pomysłodawca projektu pracujący razem z zespołem, może być najbardziej doświadczony deweloper. ***Lider musi pociągać innych za sobą, a nie ich popychać!***

Aby zespół dobrze funkcjonował, lider zespołu powinien zespół nieustannie budować. Po pierwsze zespół Agile powinien zostać właściwie dobrany. Zaleca się tu tworzenie niewielkich zespołów (od 5 do 8 osób) i dobieranie do zespołu osób o różnorodnych kompetencjach [26]. Oczywiście wszyscy członkowie zespołu muszą umieć programować, ale powinny się w nim znaleźć osoby o zdolnościach analitycznych, a także o umiejętności dokładnego i rzetelnego testowania.

Budowanie zespołu nie powinno się kończyć wraz z jego tworzeniem, ale powinno trwać przez cały czas projektu. Tu stosuje się różne podejścia, przy czym niekoniecznie są to typowe „techniki budowania zespołu" stosowane przez działy HR dużych korporacji [13], jak „wyjazdy integracyjne". Dwie ogólne zasady są tu zalecane przez metodykę Lean [21] oraz Crystal Clear [9]:

- ***umożliwienie zespołowi podejmowania własnych decyzji*** – tak aby nie musiał czekać na decyzje kierownictwa firmy;
- ***zapewnienie członkom zespołu osobistego poczucia bezpieczeństwa*** – tak aby wszyscy mogli otwarcie brać udział w dyskusji nad pomysłami bez obawy o zlekceważenie czy wyśmianie.

Reasumując: ***Samoorganizacja zespołu nie zwalnia kierownictwa firmy z odpowiedzialności za działanie zespołów!***

12.4 Wymagania i planowanie

Agile przeciwstawia się planowemu podejściu do tworzenia oprogramowania. To nie znaczy, że w Agile w ogóle nie ma planowania, lecz że jest ograniczone do

najbliższej iteracji. Planowanie musi się opierać na wymaganiach. Trzeba wiedzieć, co zaprogramować, aby móc zaplanować programowania. W Agile jednak wymagania nie są znane z góry na cały projekt. Znana jest pewna koncepcja, cel ogólny programowania, ale nie szczegóły. Te szczegóły są określane przez klienta (lub właściciela produktu) w trakcie programowania. Dlatego rola klienta, właściwej współpracy zespołu deweloperów z klientem, jest nie do przecenienia. Jak to się układa w praktyce, to zobaczymy poniżej.

12.4.1 Klient na miejscu (w zespole)

Opowieści użytkownika są precyzowane w czasie ich implementacji przez przedstawiciela klienta. Aby zespół deweloperów nie musiał domyślać się o co chodzi w opowieści, powinien mieć możliwość uzyskania informacji z pierwszej ręki bez zbędnej zwłoki. Dlatego przedstawiciel klienta powinien na co dzień być w bliskim kontakcie z zespołem, przebywać w jednym miejscu wraz z zespołem, a nawet być jego częścią.

Niestety ta reguła jest często trudna do realizacji w praktyce. Około 50% zespołów zwinnych ma problem z uzyskaniem i utrzymaniem zaangażowania klienta w projekt [2]. Przyczyny tego mogą być różne [16]:

- ***przyzwyczajenie do tradycyjnej roli klienta*** („*klient płaci i wymaga*"),
- ***niezrozumienie szczególnej roli klienta w metodykach zwinnych***, w których od zaangażowania klienta zależy w istocie powodzenie projektu,
- ***przywiązanie organizacji klienta do tradycyjnych metod***, zwłaszcza w dużej organizacji,
- ***brak zainteresowania po stronie klienta***, np. umieszczenie projektu na końcu listy priorytetów,
- ***wysokie koszty i ograniczenia z dostępnością personelu*** – oddelegowanie pracownika odpowiednio wysokiego szczebla do firmy wykonawcy kosztuje, a zdobycie wysokiej klasy specjalistów na rynku pracy jest trudne,
- ***duża odległość geograficzna między klientem a firmą wykonawcy*** (np. w kontraktach międzynarodowych), gdy różnice czasowe utrudniają komunikację
- ***stosowanie kontraktów o ustalonym wynagrodzeniu*** – wówczas klient nie ma odpowiedniej motywacji finansowej do angażowania się,
- ***niekompetencja przedstawiciela klienta*** – brak wystarczającej wiedzy lub możliwości decyzyjnych u osoby wyznaczonej przez klienta na swojego przedstawiciela.

Skutkami braku zaangażowania klienta mogą być:

- ***presja na ścisłe ustalenie warunków kontraktowych*** (zakresu, terminu, wynagrodzenia) ze strony klienta,
- ***problemy ze zbieraniem wymagań*** od klienta i uzyskaniem wyjaśnień co do szczegółów wymagań,
- ***problemy z ustaleniem priorytetów*** wymagań przez klienta,

– ***problemy z uzyskaniem informacji zwrotnej*** od klienta po dostarczeniu wydania,

a w konsekwencji:

– ***opóźnienia i spadek wydajności*** pracy zespołu,
– ***straty finansowe***, a w ostateczności fiasko projektu.
Zespoły zwinne próbują sobie radzić z tym problemem [27] stosując takie strategie, jak:
– ***dążenie do zmiany postawy klienta*** (przez podkreślanie zalet Agile na tle wad metod tradycyjnych),
– ***zapewnienie elastycznych opcji kontraktowych*** (np. zakup kilku kolejnych wydań zamiast finansowania całego projektu z góry, wybór funkcjonalności, klauzula o wypowiedzeniu umowy),
– ***dodawanie marginesu czasowego*** do planowanego harmonogramu (aby uwzględnić niepewność wymagań),
– ***obniżanie priorytetów opowieści*** użytkowników, które czekają na wyjaśnienia klienta,
– ***stosowanie pojęcia „gotowości" opowieści do implementacji*** (opowieść „gotowa" ma określone cele biznesowe, wynik i szczegóły implementacji niezbędne do oszacowania pracochłonności),
– ***wstępna ocena ryzyka*** związanego z brakiem zaangażowania klienta,
– ***przydzielanie właścicieli do opowieści***, a nie tylko do całego produktu,
– ***wprowadzenie roli zastępczego przedstawiciela klienta***, działającego jako pośrednik między klientem a zespołem (w metodyce Scrum tę rolę pełni właściciel produktu),
– ***zachęcanie klientów do udziału w regularnych pokazach*** działającego oprogramowania, co umożliwia zbieranie informacji „zwrotnych na gorąco",
– ***ukrywanie stosowania metod zwinnych*** przez zespół przed klientem, który nie akceptuje tych metod.

Trzeba jednak jasno powiedzieć, że bez zaangażowania klienta w projekt, bez jasnego określenia przez klienta, czego on oczekuje od projektu, bez weryfikacji przez klienta rozwiązania dostarczonego przez deweloperów i bez uwzględniania zwrotnych opinii klienta przez deweloperów nie da się metodami zwinnymi stworzyć oprogramowania dobrej jakości.

12.4.2 Opowieści użytkownika

W metodykach Agile nie stosuje się formalnej specyfikacji wymagań. Zamiast tego przedstawiciel klienta przekazuje do zespołu deweloperskiego opowieści użytkownika (ang. *user stories*), w których opowiada, jak oprogramowanie ma być wykorzystywane i jakie funkcjonalności powinny się w nim znaleźć. Przyjmuje się, że opowieści powinny być proste – jedna opowieść powinna przedstawiać jeden, prosty scenariusz działań.

Opowieści użytkownika mogą być spisywane. Scrum wprost nakazuje wpisywanie opowieści do rejestru produktu, z którego potem przenoszone się do

rejestru Scrumu. Jednak rejestr produktu to nie to samo co klasyczna specyfikacja wymagań. Między nimi są zasadnicze różnice zwłaszcza co do kompletności i niezmienności (tab. 2).

Tablica 12.2. Różnice między klasycznymi wymaganiami a opowieściami użytkownika

Klasyczne wymagania	**Opowieści użytkownika**
Powinny być kompletne	Z założenia nie są kompletne
Powinny być stabilne	Z założenia mogą się zmieniać
Powinny być precyzyjne	Są precyzowane dopiero w czasie implementacji
Powinny być wystarczające do implementacji	Potrzebne do szacowania pracochłonności
Są funkcjonalne i niefunkcjonalne	Dotyczą zasadniczo tylko funkcjonalności
Powinny być spójne i niesprzeczne	Mogą zmieniać poprzednie wymagania

Jedna różnica jest szczególnie ważna w kontekście jakości produktu. Klasyczna specyfikacja wymagań powinna zawierać zarówno wymagania funkcjonalne, jak i niefunkcjonalne. Wymagania niefunkcjonalne to np. wymagania na dane, ale nas interesują zwłaszcza wymagania jakościowe. Powinny być one znane na początku projektu, tak aby decyzje co do architektury i innych rozwiązań podejmować z uwzględnieniem tych wymagań. Tymczasem pierwsze opowieści użytkownika, które zgłasza klient, dotyczą z reguły najważniejszych funkcjonalności. To na nich koncentruje się uwaga i praca deweloperów. Tymczasem może się okazać, że bez znajomości wymagań jakościowych rozwiązania projektowe pójdą w niewłaściwym kierunku i trzeba będzie po pewnym czasie zmieniać architekturę, stosowane frameworki, czy bazy danych.

Przykładem mogą być wymagania dotyczące wydajności. Jeśli klient zamawiający serwis internetowy nie powie, ilu obsługuje kontrahentów w ciągu dnia i ilu żądań można się spodziewać maksymalnie w ciągu godziny, to architektura może być przewidywana w oparciu o pojedynczy serwer webowy. Jeśli poniewczasie okaże się, że wydajność serwisu jest zbyt mała, to trzeba będzie przebudować architekturę ze scentralizowanej na wielopienną. Zastosować osobny serwer www i osobny serwer aplikacji. Wprowadzić warstwę komunikacyjną między tymi serwerami. Być może zmienić serwer bazy danych na bardziej wydajny. Znajomość wymagań wydajnościowych od początku umożliwiłaby wprowadzenie tych rozwiązań od razu i zaoszczędziłaby wielu zmian.

Te zespoły, które są świadome możliwych problemów, stosują odpowiednio wcześnie pewną analizę potrzeb klientów i użytkowników. Nie ograniczają się tylko do przyjmowania tego, co mówi klient, ale same stymulują pewne wymagania i wpisują je do rejestru produktu. Wielkie znaczenie ma tu doświadczenie zespołu (a w przypadku Scrumu – właściciela produktu). Co pewien czas są podejmowane iteracje, które mają za cel realizację nie nowych opowieści użytkownika, lecz wymagań jakościowych wynikających z ujawnianych lub przewidywanych problemów.

12.4.3 Priorytetyzacja potrzeb klienta

Przy planowaniu kolejnych iteracji trzeba wybierać opowieści do implementacji. Zasadą jest wybór tych opowieści, które są najważniejsze dla klienta. W rejestrze produktu opowieściom przypisuje się priorytety, które podaje klient. Co jednak, jeśli przedstawicieli klienta jest kilku i inne rzeczy są dla nich ważne? Część zespołów stosuje podejście HiPPO (ang. *Highest Paid Person's Opinion*) polegające na nadawaniu najwyższego priorytetu opiniom osób z najwyższymi zarobkami. Wówczas kierownictwo wyższego szczebla będzie miało wyższy priorytet niż kierownictwo niższego szczebla, a oni wyższy niż szeregowi pracownicy. Jednak nie zawsze jest to słuszne.

Tu wprowadza się ***rozróżnienie pojęć klienta i użytkownika***. Jeśli oprogramowanie jest tworzone na zamówienie jakiejś firmy lub instytucji, to klientem jest *osoba prawna*. Użytkownikami będą pracownicy tej osoby prawnej. Przedstawicielem klienta jest zazwyczaj kierownik odpowiednio wysokiego szczebla. Problem pojawia się, gdy przedstawiciel klienta nie do końca orientuje się w potrzebach swoich pracowników. Wówczas to, co mówi klient i to, jakie funkcjonalności powinny się znaleźć w oprogramowaniu, to nie koniecznie to samo. Programista nie powinien się chować za stwierdzeniem „to klient tak chciał" ani „klient nic o tym nie mówił". A czy zadał sobie tyle trudu, aby porozmawiać z końcowymi użytkownikami? A może nawet tylko poobserwować ich pracę?

Dla przykładu, nawet pobieżna obserwacja pracy kasjerek w supermarkecie może wykazać, że sporo czasu zajmuje im skasowanie towaru, który nie ma kodu paskowego na opakowaniu (np. bułek). Wówczas kasjerka musi przejrzeć w osobnym skoroszycie wydrukowane zdjęcia towarów, wyszukać w nim towar, który się pojawia na taśmie i wprowadzić kod ręcznie. Gorzej, gdy nie może znaleźć towaru i musi pytać inne kasjerki lub zadzwonić do działu informatyków. W bardziej skomplikowanych przypadkach ktoś musi przyjść, obejrzeć towar, zlokalizować na hali półkę z towarem i tam odczytać kod. Minuty płyną nieubłaganie. A wystarczyłoby wyposażyć aplikację kasową w funkcjonalność wyszukiwania towaru po nazwie, tak jak to się robi w sklepach internetowych.

Inna sytuacja jest wtedy, gdy klientami są osoby fizyczne nabywające oprogramowanie do własnego użytku. Wówczas klienci i użytkownicy to ten sam zbiór ludzi. Oczywiście w dużym zbiorze użytkowników mogą być grupy ludzi o zróżnicowanych potrzebach. Wówczas z reguły im większa grupa użytkowników ma określoną potrzebę, tym chętniej realizuje się opowieści zgłaszane przez tę grupę. Wiąże się to z interpretacją zasady Pareto: 80% użytkowników korzysta jedynie z 20% funkcjonalności oprogramowania. Niestety prowadzi to do wytwarzania oprogramowania, które jest pozbawione wielu przydatnych funkcjonalności i zaniedbywania tej części użytkowników, którzy chcieliby z tego skorzystać.

Przykład stanowią aplikacje do przeglądania i udostępniania zdjęć (np. zdjęć z wakacji). Wiele tych aplikacji ma bardzo prostą funkcjonalność typu: załaduj zdjęcia, pogrupuj w albumy, udostępniaj albumy znajomym. Nieco bardziej zaawansowane funkcjonalności to usuwanie czerwonych oczu i kadrowanie zdjęć. Niektóre aplikacje dają możliwość dodawania do zdjęć różnych zabawnych efektów, np. „kociego pyszczka" albo „zakręconych oczu". Mało jest takich, które

dają bardziej zaawansowane możliwości edycji obrazu, np. rozjaśnienie ciemnych obszarów metodą korekcji gamma, czy też retuszowanie przez kopiowanie fragmentów. W niektórych aplikacjach brak jest nawet przechodzenia po plikach graficznych w katalogu!

Czasami stosuje się też inne techniki priorytetyzacji [4], takie jak:

- ***technika oparta na wartości biznesowej*** (im wyższa wartość funkcjonalności, tym wyższy jej priorytet),
- ***technika oparta na ryzyku technologicznym*** (najpierw funkcjonalności o najwyższym ryzyku technologicznym, aby zorientować się czy koncepcja techniczna jest wykonalna),
- ***technika oparta na ryzyku biznesowym*** (najpierw funkcjonalności o najwyższym ryzyku biznesowym, aby jak najszybciej uzyskać z rynku potwierdzenie opłacalności),
- ***technika „chodzącego szkieletu"*** (ang. *walking skeleton*), polegająca na zrealizowaniu najpierw tylko tych funkcjonalności, które są potrzebne, aby sprawić wrażenie, że system chodzi. Technika ta jest też zwana techniką MVP (ang. *Minimal Viable Product*) [3], czyli najmniejszego produktu, który zapewnia opłacalność.
- ***technika MoSCoW*** (ang. *Must have, Should have, Could have, Won't have*), czyli najwyższy priorytet dla tych cech, które produkt musi mieć, potem dla tych, które produkt powinien mieć, a na końcu dla tych, które może mieć [8].
- ***technika oparta na modelu wymagań Kano*** – podobna do poprzedniej, ale oparta na przedstawionym już wcześniej modelu Kano [18].

12.4.4 Realizacja wymagań jakościowych w dodatkowych iteracjach

Większość wymagań, które klient przekazuje zespołowi w opowieściach użytkownika, dotyczy funkcjonalności. Pozostają jednak wymagania niefunkcjonalne, takie jak wydajność i bezpieczeństwo. Zazwyczaj wychodzą one dopiero po pewnym czasie. W Agile nie ma fazy wydobywania wymagań, wymagania pojawiają się na bieżąco, wraz z kolejnymi wydaniami oprogramowania trafiającymi do oceny użytkowników. Oczywiście może się zdarzyć, że klient jest od początku świadomy wymagań niefunkcjonalnych i je zgłasza do zespołu. Częściej jednak klient rozpoczyna opowieści od prostych funkcjonalności, z czasem je rozbudowuje, a inne wymagania pojawiają się „przy okazji".

Wymagania niefunkcjonalne obejmują wymagania jakościowe, wymagania na dane, wymagania dotyczące konfiguracji sprzętowej lub softwarowej (np. dotyczące wykorzystania określonej wersji systemu operacyjnego). Spośród nich najbardziej „niekonkretne" są wymagania jakościowe. Wyróżnia się kilka ogólnych atrybutów jakościowych, takich jak wydajność, wiarygodność, elastyczność, użyteczność [22]. Nie wszystkie atrybuty są równie ważne we wszystkich typach projektów, np. wiarygodność jest istotna przede wszystkim w systemach medycznych, w systemach finansowych i systemach czasu rzeczywistego. Przy braku wymagań jakościowych na wczesnym etapie projektu, zespół musi posługiwać się

intuicją lub wiedzą o standardach (formalnych i nieformalnych), aby podejmować decyzje projektowe nie blokujące w przyszłości realizację nieznanych wymagań funkcjonalnych. Taką praktykę nazywa się odraczaniem decyzji projektowych.

Zazwyczaj wymagania niefunkcjonalne pojawiają się w trakcie projektu i często są zauważane przez zespół, a niekoniecznie przez klienta. W Scrumie praktykuje się wówczas włączanie wymagań niefunkcjonalnych jako opowieści do backlogu projektu. Gdy właściciel produktu ustala z klientem priorytety opowieści, aby przenieść je do backlogu sprintu, klient staje się świadomy konieczności implementacji tych opowieści niefunkcjonalnych, a zagadnienia jakościowe są w ten sposób włączane w normalny cykl iteracji.

12.5 Problemy testowania

Testowanie kodu jest uważane w Agile za podstawową metodę sprawdzenia jakości oprogramowania. Trzeba jednak zauważyć, że praktyki Agile skupiają się na testowaniu jednostkowym, trochę zaniedbując inne poziomy testowania.

Nie można powiedzieć, że w Agile nie wykonuje się testów akceptacyjnych. W Extreme Programming jest specjalna praktyka przeprowadzania testów akceptacyjnych pisanych na podstawie opowieści użytkownika. Problem w tym, że testy te są przeprowadzane przez zespół, a nie przez samych użytkowników. W innych metodykach zakłada się, że użytkownicy dostarczają informacji zwrotnych do zespołu na podstawie testów wykonywanych w czasie próbnego lub normalnego użycia. Czy to rzeczywiście można nazwać testowaniem akceptacyjnym i jak się to ma do testów regresyjnych – zobaczymy poniżej.

12.5.1 Podejście TDD

Wielu praktyków Agile zaleca pisanie testów przed pisaniem właściwego kodu modułu lub klasy. Jest to podejście zwane TDD (ang. *Test-Driven Development*)[7]. Programista, który ma do zaimplementowania pewną funkcjonalność, powinien zacząć od napisania testu, który powinien się zakończyć niepowodzeniem. Potem powinien napisać taki kod, aby test został zaliczony. Wreszcie, przez refaktoring, powinien przygotować kod do kolejnej zmiany.

Trzeba zauważyć, że podejście TDD nie polega na tym, aby najpierw napisać *wszystkie* testy, a potem dopiero pisać właściwy kod implementacyjny. Przy podejściu TDD cykl test-kod-refaktoryzacja jest krótki i dotyczy tylko jednej funkcji.

Udowodniono, że przy podejściu TDD kod implementacyjny jest krótszy i lepiej pokryty testami [17]. Wykazano też, że efektywność pracy programistów jest nieco lepsza w przypadku TDD, chociaż jednocześnie można zauważyć, że mimo entuzjastycznych badań pilotażowych, jakość dostarczanego oprogramowania w badaniach rygorystycznych wcale nie okazuje się być lepsza od innych metod [24]. TDD nie jest panaceum na problemy implementacyjne. Kod implementacyjny będzie tak dobry (i tylko tak dobry) jak napisany test. Jeżeli ten sam programista pisze test i testowany kod, to prawdopodobne jest, że będzie

tak samo myślał podczas pisania testu i podczas pisania kodu. Może popełnić ten sam błąd myślowy. Owszem pisanie testu przed kodem powoduje, że programista skupia się tylko na potrzebach zewnętrznych wobec kodu, a nie na problemach implementacyjnych. Łatwiej mu zrozumieć potrzeby i potem do tych potrzeb się dopasować. Z drugiej strony nie daje to gwarancji, że programista nie popełni błędu i w teście i w kodzie.

12.5.2 Testowanie akceptacyjne przez użytkowników

W Extreme Programming Kent Beck zalecił przeprowadzanie testów akceptacyjnych przed każdym małym wydaniem oprogramowania. Testy te powinny być pisane na podstawie opowieści użytkownika przekazywanych przez klienta na początku cyklu iteracji i powinny symulować różne scenariusze działań użytkownika. Znane są wyspecjalizowane narzędzia do takiego testowania, np. Selenium do testowania aplikacji webowych.

Problem jest jednak w wiarygodności takich testów, a wynika on z dwóch wątpliwości:

1. Czy klient przekazał scenariusze działań użytkownika w opowieściach użytkownika w sposób wiarygodny?
2. Czy deweloperzy pisząc scenariusze testowe na podstawie opowieści użytkownika zrobili to w sposób wiarygodny?

Odnośnie pierwszej wątpliwości: pierwsze opowieści użytkowników przekazywane przez klienta opisują zazwyczaj najprostsze sytuacje zakończone powodzeniem. Z czasem opowieści się komplikują, jednak zawsze opowiadający będzie miał tendencję do upraszczania scenariuszy. Czy klient będzie umiał przedstawić scenariusze zakończone niepowodzeniem, scenariusze rzadkie? Czy użytkownicy zawsze postępują w sposób przedstawiony przez klienta? A może czasami postępują niezgodnie z procedurami, niezgodnie z regulaminem?

Rozwiązania tego problemu mogą być dwa:

1. Analiza i modelowanie procesów biznesowych, np. przy użyciu notacji BPMN [6], a następnie generowanie testów na podstawie modeli [20].
2. Rejestrowanie rzeczywistych działań użytkowników w środowisku produkcyjnym i ich odtwarzanie w środowisku testowym.

Pierwsze rozwiązanie wymaga jednak działań nietypowych dla Agile – programowania w oparciu o modele (ang. MDD – *Model-Driven Development*). Agile przewiduje takie podejście [5], ale nie w takim samym wymiarze jak metodyki tradycyjne. Modele tworzy się jedynie tak dobre, na ile to potrzebne dla zrozumienia najbliższej iteracji.

Drugie rozwiązanie jest bardziej wiarygodne, ale użycie oprogramowania typu *keylogger* dla wielu klientów będzie nieakceptowalne ze względów bezpieczeństwa.

Ze względu na powyższe wątpliwości nowsze metodyki Agile już nie zalecają formalnych testów akceptacyjnych przed wydaniem, ale opierają się na założeniu,

że użytkownicy, którzy dostaną oprogramowanie do sprawdzenia zmian wprowadzonych w ostatniej iteracji, przeprowadzą jakieś testy i na tej podstawie zgłoszą swoje uwagi. Niestety jest to założenie często nieprawdziwe. Przeprowadzenie testów akceptacyjnych z prawdziwego zdarzenia jest kosztowne i dlatego stosuje się je tylko w systemach wybitnie potrzebujących niezawodności, w wydzielonym środowisku wdrożeniowym. W pozostałych przypadkach „testowanie akceptacyjne" tak naprawdę polega na sprawdzeniu, czy „oprogramowanie działa", tzn. czy funkcjonalność wymagana w ostatniej iteracji została zaimplementowana. Użytkownicy jednak nie będą przeprowadzać testów regresyjnych, tzn. nie będą sprawdzać, czy coś, co już kiedyś działało, w dalszym ciągu działa. Może to wyjść przypadkowo podczas dalszego użytkowania, ale specjalnie tego robić nie będą.

Aby zmniejszyć wysiłek testowania akceptacyjnego można zastosować tzw. *testy dymu* (ang. *smoke testing*). Polega to na sprawdzeniu, czy poprawnie działają krytyczne funkcje oprogramowania. Innymi słowy, czy system działa stabilnie. Sprawdzenie stabilności jest warunkiem minimum testów akceptacyjnych, chociaż nie zastąpi bardziej rygorystycznych testów.

Bardziej rygorystyczne testy powinny obejmować też wymagania niefunkcjonalne. Np. w przypadku systemów wrażliwych na bezpieczeństwo, zleca się przeprowadzenie testów penetracyjnych wyspecjalizowanym firmom [10].

12.6 Podsumowanie

Aby zapewnić jakość procesu i produktu softwarowego w Agile potrzebne jest stosowanie praktyk zapewniających spełnienie kluczowych warunków jakości, takich jak:

- **zapewnienie dobrej współpracy z klientem** w czasie całego procesu – co może być trudne przy niewłaściwej postawie klienta,
- **oparcie projektu na rzeczywistych potrzebach klienta i użytkowników** – a nie tylko przekazywanych przez nich opowieściach, co wymaga pewnego doświadczenia, umiejętności obserwacji i analizy, w tym przewidywania **sytuacji wyjątkowych** i planowania reakcji na te sytuacje,
- **zapewnienie dobrego przepływu informacji** w zespole i między wszystkimi zainteresowanymi stronami – najlepiej praca w bezpośrednim kontakcie, a jeśli tego nie można zapewnić, to utrzymywanie częstych kontaktów,
- **zachowywanie standardów pracy inżynierskiej** – co wymaga utrzymywania programistów wysokiej klasy i utrzymanie zasad zwinnego działania (zapewnienia czasu i swobody podejmowania decyzji przez zespół).
- **weryfikacja i walidacja** wyników pracy przez klienta/użytkowników – utrzymywanie **dyscypliny testów** jednostkowych i regresyjnych oraz przeprowadzanie **testów akceptacyjnych** przez samych użytkowników.

Ważne, aby zespoły zwinne znały te praktyki i stosowały je dla zapewnienia jakości procesu i produktu softwarowego.

Bibliografia

1. IEEE standard glossary of software engineering terminology. Tech. rep. (1991).
2. Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E., Sarwar, S.Z.: Agile software development: Impact on productivity and quality. In: 2010 IEEE International Conference on Management of Innovation Technology. pp. 287–291 (2010).
3. Aliance, A.: Minimum viable product (mvp), https://www.agilealliance.org/glossary/mvp/.
4. AltexSoft: 25 scrum process best practices that set your agile workflow for efficiency (2018), https://www.altexsoft.com/blog/business/25-scrum-process-best-practices-that-set-your-agile-workflow-for-efficiency/.
5. Ambler, S.W.: Agile Model driven development. John Wiley & Sons (2007).
6. Avendi: BPNM - podstawowe informacje (2020), https://avendi.edu.pl/analiza-i-modelowanie/bpmn/bpmn-podstawowe-informacje/.
7. Beck, K.: Test-Driven Development by Example. Addison-Wesley Professional (2003).
8. Clegg, D., Barker, R.: Case method fast-track: a RAD approach. Addison-Wesley Longman Publishing (1994).
9. Cockburn, A.: Crystal clear: a human-powered methodology for small teams. Pearson Education (2004).
10. Cruzes, D.S., Felderer, M., Oyetoyan, T.D., Gander, M., Pekaric, I.: How is security testing done in agile teams? a cross-case analysis of four software teams. In: Baumeister, H., Lichter, H., Riebisch, M. (eds.) Agile Processes in Software Engineering and Extreme Programming. pp. 201–216. Springer International Publishing, Cham (2017).
11. Deming, W.E.: Out of the Crisis. MIT Press (1986).
12. Drucker, P.: The Practice of Management. Harper, New York (1954).
13. High5Test: The best way to motivate employees: 5 excellent team building techniques, https://high5test.com/team-building-techniques-to-motivate-employees/.
14. Highsmith, J.: History: The agile manifesto, https://agilemanifesto.org/history.html.
15. Hoda, R.: Self-organizing agile teams: A grounded theory (phd thesis). Tech. rep. (2011).
16. Hoda, R., Noble, J., Marshall, S.: The impact of inadequate customer collaboration on self-organizing agile teams **53**, 521–534 (2011).
17. Janzen, D., Saiedian, H.: Does test-driven development really improve software design quality? **25**, 77–84 (2008).
18. Kano, N., Seraku, N., Tkahashi, F., Tsuji, S.i.: Attractive quality and must-be quality (in japanese) **14**, 39–48 (1984).
19. Mnkandla, E., Dwolatzky, B.: Agile Software Methods: State-of-the-Art. Information Science Reference (Idea Group) (2007).

20. de Moura, J.L., Charão, A.S., Lima, J.C.D., de Oliveira Stein, B.: Test case generation from bpmn models for automated testing of web-based bpm applications. In: 2017 17th International Conference on Computational Science and Its Applications (ICCSA). pp. 1–7 (2017).
21. Poppendieck, M., Poppendieck, T.: Lean Software Development: An Agile Toolkit. Addison-Wesley (2003).
22. Pressman, R.S.: Software Engineering. A Practitioner's Approach. McGraw-Hill (1992).
23. Radigan, D.: Bringing in a ringer (or, 'how to work with specialists'), `https://www.atlassian.com/agile/teams/working-with-specialists`.
24. Shull, Forrest, e.a.: What do we know about test-driven development? **27**, 16–19 (2010).
25. Stamelos, I.G., Sfetsos, P.: Agile Software Development Quality Assurance. Information Science Reference (Idea Group) (2007).
26. Thorn, M., Galen, B.: How to create and lead high-performing agile teams: 8 secrets, `https://techbeacon.com/app-dev-testing/how-create-lead-high-performing-agile-teams-8-secrets`.
27. van Waardenburg, G., van Vliet, H.: When agile meets the enterprise **55**, 2154–2171 (2013).
28. Wells, D.: Extreme programming: A gentle introduction (1999), `http://www.extremeprogramming.org/`.

13. Licencjonowanie oprogramowania

Tomasz Boiński i Szymon Olewniczak

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
tomasz.boinski@eti.pg.edu.pl
szymon.olewniczak@pg.edu.pl

Streszczenie
Wolne i otwarte oprogramowanie przeżywa ostatnimi laty rozkwit. Coraz więcej przedsiębiorstw komercyjnych opiera rozwój swoich firm na otwartym oprogramowaniu. Zarówno mali, jak i duzi gracze mają świadomość komplikacji współczesnych systemów i niemożności samodzielnego ich rozwoju. Z pomocą przychodzi otwarte podejście do wytwarzania oprogramowania. Wymaga to jednak pewnego zrozumienia uwarunkowań prawnych, a w szczególności licencji, na jakich wydawane jest oprogramowanie. Niniejsze opracowanie ma na celu omówienie tychże uwarunkowań, odpowiada na pytanie dlaczego potrzebujemy licencji na oprogramowanie. Opisywane zostały również najważniejsze wg autorów licencje, oraz wskazuje, czym należy kierować się przy doborze licencji.

Słowa kluczowe: oprogramowanie, licencje, prawo autorskie.

13.1 Wstęp

Ostatnimi laty zaobserwować można rosnące zainteresowanie wykorzystaniem otwartych komponentów programowych. Wolne i otwarte oprogramowanie wspierane i wykorzystywane jest przez praktycznie wszystkich większych graczy na rynku, nawet tych, którzy do tej pory zaciekle zwalczali ruch Free/Open Source (jak np. Microsoft).

Rosnąca akceptacja dla wolnego i otwartego oprogramowania wynika z kilku faktów. Przede wszystkim z komplikacji współczesnych systemów czy aplikacji, a tym samym rosnących kosztów jego wytwarzania. Drugim istotnym aspektem jest postępująca migracja modeli biznesowych ze sprzedaży oprogramowania na sprzedaż usług. W tym pierwszym wypadku zastosowanie licencji wolnych oraz otwartych miało poważny wpływ na licencje finalnego produktu. Warunki licencyjne narzucane przez otwarte komponenty mogły być nie do zaakceptowania dla wielu firm. W przypadku sprzedaży usług samo oprogramowanie jest tylko elementem modelu biznesowego, często bezwartościowego bez zasobów kadrowych, sprzętowych czy know–how przedsiębiorstwa.

W niniejszym rozdziale omówione zostały uwarunkowania prawne, odpowiadające na pytanie, dlaczego potrzebujemy licencji na oprogramowanie (część 13.2).

Następnie w części 13.3 przedstawiono podstawowe różnice pomiędzy licencjami wolnymi i otwartymi. Część 13.4 opisuje najważniejsze wg autorów licencje na oprogramowanie. W części 13.5 zaprezentowano zagadnienie zgodności licencji, a następnie część 13.6 wskazuje, czym należy kierować się przy doborze licencji. Całość wieńczy krótkie podsumowanie.

13.2 Po co nam licencje?

Oprogramowanie, jak każdy utwór, z definicji chroniony jest prawem autorskim, które definiuje ramy wykorzystania oprogramowania, w jakich poruszać może się użytkownik końcowy. Przykładowo art. 75 ust. 1. ustawy o prawie autorskim i prawach pokrewnych [12], jeżeli umowa nie stanowi inaczej, zezwala przy pewnych warunkach (np. gdy kopia nie będzie używana równolegle z oryginałem) użytkownikowi końcowemu na sporządzenie kopii zapasowej oprogramowania, na analizę konstrukcji i zasady działania oprogramowania, zwielokrotnianie kodu lub tłumaczenie jego formy, jeżeli jest to konieczne dla umożliwienia współpracy z oprogramowaniem zewnętrznym. Analogicznie np. w USA, sekcja 117 Copyright Act zezwala właścicielowi kopii oprogramowania na wykorzystywanie tej kopii za pomocą komputera, nawet jeżeli to wykorzystywanie będzie wymagało wykonania kopii lub modyfikacji oprogramowania. Kopiowanie takie czy też modyfikacja, nie wymaga więc zgody właściciela praw autorskich i zezwala na wykorzystanie kopii posiadanej kopii oprogramowania bez licencji. W ogólności więc właściciel kopii oprogramowania może wykonać kopię utworu na użytek własny, a nawet podzielić się nią z osobami pozostającymi z nim w kręgu osobistym.

Zapisy prawa autorskiego stoją w sprzeczności zarówno z interesem komercyjnych dostawców oprogramowania (możliwość analizy, badania i modyfikacji kodu, prawo do wykonywania kopii), jak i przedstawicieli ruchu wolnego i otwartego oprogramowania (np. brak możliwości tworzenia dzieł pochodnych czy redystrybucji zmian). Wprowadzono więc pojęcie licencji na oprogramowanie, oryginalnie w odpowiedzi na potrzeby komercyjnych dostawców oprogramowania.

Licencje komercyjne, będące de facto umową między dystrybutorem a nabywcą, posiadają więc zapis o utrzymanie własności kopii przez dystrybutora (stąd ich nazwa licencje własnościowe). Jako że odbiorca oprogramowania nie jest w ich rozumieniu właścicielem kopii, odpowiednie sekcje (np. art. 75 ust. 1 ustawy o prawie autorskim i prawach pokrewnych czy sekcja 117 Copyright Act) przestają obowiązywać, a użytkownik końcowy, chcąc korzystać z oprogramowania, musi zaakceptować ostrzejsze wymogi licencyjne, niżby to wynikało z prawa autorskiego. Otrzymywane oprogramowanie nie staje się jego własnością, a jedynie narzędziem niezbędnym do realizacji umowy.

Z kolei licencje wolne i otwarte w ogólności przekazują własność kopii, jednak ich akceptacja daje użytkownikowi dodatkowe prawa, np. prawo do modyfikacji oprogramowania i redystrybucji tychże zmian. Należy tutaj jednak pamiętać, że właściciel kopii nie jest tożsamy z właścicielem praw autorskich. Użytkownik końcowy uzyskuje prawa, wynikające z ustawy, tylko do tej jednej kopii, reszta praw pozostaje przy właścicielu praw autorskich.

Licencje na oprogramowanie pełnią również dodatkowe funkcje. Oprócz nadawania praw czy wymuszania ograniczeń licencje opisują zależności oraz odpowiedzialność za kod i wykorzystanie oprogramowania zachodzące pomiędzy stronami porozumienia licencyjnego. Zazwyczaj mówią, że producent nie ponosi odpowiedzialności za wszelkie uszkodzenia wynikłe z prawidłowego jak i nieprawidłowego wykorzystania oprogramowania. Ograniczają się jedynie do stwierdzenia, że dystrybutor dołożył wszelkich starań do sprawdzenia poprawności oprogramowania, a korzystanie z niego zgodnie z jego zastosowaniem nie powinno spowodować start czy uszkodzenia sprzętu. W rozwiązaniach komercyjnych zazwyczaj dodatkowo chronią przedsiębiorstwo dystrybutora opisując zakres odpowiedzialności, np. za stan usług związanych z oprogramowanie.

13.3 Wolne a otwarte licencje

Wolne oraz otwarte licencje posiadają pewne cechy wspólne. W ogólności, w obu rodzajach licencji, prawo własności kopii przekazywane jest użytkownikowi końcowemu. Niesie to za sobą pewne istotne następstwo – akceptacja licencji jest opcjonalna – użytkownik końcowy może używać oprogramowanie bez konieczności akceptacji licencji, a tym samym może badać oprogramowanie czy np. wykonywać zmiany, ale jedynie na własne potrzeby. Jeżeli użytkownik chce skorzystać z dodatkowych praw wynikających z licencji (np. możliwość redystrybucji samego oprogramowania czy też zmian) to musi ją zaakceptować i przestrzegać.

Z punktu widzenia użytkownika końcowego licencje wolne i otwarte mogą nie być rozróżnialne. Warunki typowe dla każdego z rodzajów licencji niosą za sobą dość istotne następstwa w kwestii możliwości wykorzystania oprogramowania wydanego na poszczególnych licencjach.

Licencje wolne mają na celu zachowanie wolności oprogramowania. Wolność użytkownika jest pochodną tejże. Licencje te dają końcowemu użytkownikowi duże uprawnienia, takie jak możliwość redystrybucji, inżynierii wstecznej czy innej formy modyfikacji kodu. Zazwyczaj dodatkowe prawa wymagają czegoś w zamian od użytkownika, musi się on zgodzić na dodatkowe warunki, by móc z tych praw skorzystać, np. licencje tego rodzaju zazwyczaj wymuszają, by wszelkie modyfikacje czy dzieła pochodne dystrybuowane były na tej samej licencji co oryginalne dzieło. Warunek ten często ogranicza możliwe pola zastosowania oprogramowania (np. prawnie nie możemy takiego oprogramowania włączyć do aplikacji komercyjnej, a następnie redystrybuować na licencji własnościowej). Tym samym również uniemożliwiają łączenie oprogramowania z niektórymi innymi otwartymi licencjami. Najbardziej znanym przedstawicielem licencji wolnych jest licencja GPL [1].

Licencje otwarte dzielą się na dwa rodzaje:

1. 'copyleft' – analogicznie jak licencje wolne mają na celu zachowanie wolności oprogramowania. Cechują się podobnymi atrybutami, w ogólności każda wolna licencja to licencja otwarta typu 'copyleft'. Najbardziej znanym typem tej licencji jest licencja GPL [1];

2. licencje „zezwalające” (ang. *permissive licenses*) – mają na celu zachowanie wolności końcowego użytkownika. Zezwalają końcowemu użytkownikowi na pełną dowolność w wykorzystaniu oprogramowania nie nakładając związanych z tym obowiązków. Oprogramowanie na takiej licencji może być bez przeszkód wykorzystywane przez oprogramowanie własnościowe. Przykładem takiej licencji są np. licencje BSD [14, 15] czy MIT [16].

13.4 Najistotniejsze wolne i otwarte licencje

Istnieje wiele licencji zaliczanych zarówno do grupy licencji wolnych, jak i otwartych [28]. Część z nich stosowana jest powszechnie, część jedynie w dedykowanych projektach (np. licencja PostgreSQL [25] czy też Mozilla Public License [24]). Należy również pamiętać o domenie publicznej (ang. *public domain*). Nie jest to licencja, jednak w praktyce działa tak, jakby to była jedna z nich. Oprogramowanie znajdujące się w przestrzeni publicznej może być dowolnie używane, jest jednak rzadkością – do domeny publicznej zazwyczaj trafiają dzieła określony czas (70–100 lat) po śmierci oryginalnego autora. W przypadku oprogramowania musi być ono więc jawnie umieszczone w przestrzeni publicznej (taką formę licencjonowania przyjęło np. SQLite [26]). Jednakże na przestrzeni najbliższych 50–100 lat lawinowo oprogramowanie zacznie trafiać do domeny publicznej. W dalszej części niniejszego rozdziału zaprezentowane zostaną najpopularniejsze licencje, powszechnie wykorzystywane przez społeczność.

13.4.1 MIT/X11

Jest to prosta, zezwalająca licencja, zgodna z GPL [16]. Oryginalnie stosowana była głównie przez projekt XFree86 (najpopularniejsza implementacja systemu X Window dla systemów UNIX'owych). Licencja ta hołduje ogólnej zasadzie: „Możesz rozbić co tylko zechcesz z kodem, ale nie możesz powiedzieć, że go napisałeś”. W rezultacie oprogramowanie wydane na licencji MIT/X11 może być wykorzystywane w zamkniętym kodzie, pod warunkiem dołączenia treści licencji.

13.4.2 BSD

Kolejna prosta, powszechnie wykorzystywana zezwalająca licencja. Istnieją jej cztery główne wersje:

- Oryginalna, czterozdaniowa licencja zezwalająca na dowolne wykorzystanie kodu, ale zabraniająca używania w materiałach reklamowych nazwisk/nazw autorów i wymagająca wspomnienia o nich w dokumentacji [21]. Wersja ta nie jest zgodna z licencją GPL w wersji 2, ale jest zgodna z GPL w wersji 3.
- Zmodyfikowana, trzyzdaniowa wersja, zabraniająca używania w materiałach reklamowych nazwisk/nazw autorów [15]. Dla odróżnienia od oryginalnej licencji BSD występuje pod nazwami: „BSD License 2.0”, „Revised BSD License”, „New BSD License”, lub „Modified BSD License”. Jest zgodna z każdą wersją licencji GPL.

- Uproszczona, dwuzdaniowa wersja [14], znana pod nazwami: „Simplified BSD License" lub „FreeBSD License". Nie posiadająca ograniczeń narzucanych przez pierwsze 2 wersje. Jest zgodna z każdą wersją licencji GPL.
- zerozdaniowa [17], zwana również „BSD Zero Clause License". Podobnie jak wersja 2-zdaniowa nie nakłada żadnych ograniczeń, dodatkowo nie wymaga umieszczania jakiejkolwiek informacji w kodzie, dokumentacji czy wersji binarnej dzieła pochodnego. Jest zgodna z każdą wersją licencji GPL.

Jest to powszechnie wykorzystywana rodzina licencji, jednakże nie zaleca się używania jej oryginalnej, czterozdaniowej wersji. W tym przypadku, jako że wszystkie wersje funkcjonują pod jedną wspólną nazwą, należy każdorazowo zwrócić uwagę na dokładne zapisy licencyjne. Stąd też Free Software Foundation zaleca, by zamiast licencji BSD używać jednoznacznej co do nazewnictwa licencji MIT/X11 [10].

13.4.3 Apache 2.0

Licencja [22] ta wymaga zachowania informacji o posiadaczu praw autorskich, zezwala na stosowanie oprogramowania w trakcie wytwarzania zarówno zamkniętego, jak i wolnego czy otwartego kodu. Nie jest więc licencją typu 'copyleft'. Jest to jedna z niewielu licencji dotykająca również problemu patentów. W jej myśl każdy współtwórca udziela automatycznie licencje na patenty implementowane przez to oprogramowanie. Dzięki temu licencja jest dość powszechnie wykorzystywana przez podmioty komercyjne. Warunek też powoduje, że licencja ta jest zgodna z GPLv3 (ale nie vice versa), ale nie jest zgodna z GPLv1 and GPLv2.

13.4.4 GPL

Chyba najbardziej znana wolna i otwarta licencja [1]. Została opracowana przez Richarda Stallmana, jest emanacją filozofii stojącej za Free Software Foundation. Użytkownikowi końcowemu, który zaakceptował licencję, daje ona szerokie możliwości wykorzystania kodu. Użytkownik ma więc prawo do dowolnego wykorzystania oprogramowania, badania zasady jego działania, modyfikacji oraz dalszej redystrybucji zarówno oryginału, jak i wprowadzonych zmian. Uprawnienia te nie są jednak bezwarunkowe, jak to miało miejsce we wcześniej omawianych licencjach. Jako że jest to licencja typu 'copyleft', wymaga by każde dzieło pochodne dystrybuowane również było na dokładniej tej samej licencji. Każdy odbiorca oprogramowania (w dowolnej formie, również binarnej) musi więc otrzymać pełną kopię kodu źródłowego.

Dzieło pochodne jest tu rozumiane bardzo szeroko – każde oprogramowanie, które wykorzystuje, łączy, zmienia, czy nawet linkuje (zarówno dynamicznie, jak i statycznie) kod wydany na licencji GPL, jeżeli tylko jest rozpowszechniane, musi być wydane na tej samej licencji [9]. Ma to na celu zachowanie przechodniości i niezmienności praw wynikających z tejże licencji w sytuacji, gdy dzieło zostało zmienione lub rozbudowane. W praktyce przyjmuje się więc (nawet mimo pewnych dywagacji prawnych dotyczących np. europejskiego systemu

prawnego [20]), że jeżeli tylko oprogramowanie jest dystrybuowane i wykonywane w tym samym procesie co oprogramowanie wydane na licencji GPL, niezależnie od związku między poszczególnymi fragmentami tego oprogramowania, całość musi być wydana na licencji GPL. Ma to szczególne znaczenie w sytuacji, gdy korzystamy z bibliotek systemowych, czy tworzymy np. rozszerzenia (tzw. *plug–in*). Ponadto obecnie funkcjonują 2 wersje licencji – 2 i 3. Nie są one ze sobą zgodne, a dzieło wydane na licencji np. w wersji 2 nie może być bez zgody autorów wydane na licencji w wersji 3 (patrz część 13.5)! Wynika to wprost z warunku, że każde dzieło pochodne musi być wydane na dokładnie tej samej licencji.

"Wirusowość" licencji GPL może rodzić pewne problemy z praktycznym zastosowaniem wydanego na niej oprogramowania. Wprowadzono więc szereg wyjątków od tej reguły. Dla przykładu implementacja openJDK środowiska Java posiada tzw. *CLASSPATH exception*. Problem polega na tym, że każda aplikacja napisana w języku Java jest kompilowana i wykonywana w obrębie procesu maszyny wirtualnej Java. Każdy kod jest również dziełem pochodnym po maszynie wirtualnej (chociażby poprzez dziedziczenie po klasie java.lang.Object). Wyjątek mówi więc, że jeżeli kod odwołuje się do bibliotek systemowych, zdefiniowanych w ramach maszyny wirtualnej, poprzez ich ładowanie za pomocą zmiennej środowiskowej CLASSPATH, to kod ten nie musi być wydany na licencji GPL. Z podobnych przyczyn zdefiniowano inne wyjątki, takie jak *libstdc++* (w celu umożliwienia kompilacji oprogramowania kompilatorem GCC i włączenie nagłówków bibliotek systemowych) czy *system* (w celu umożliwienia wykonywania wywołań systemowych).

Należy oczywiście pamiętać, że warunki licencji GPL obowiązują nas w przypadku, gdy udostępniamy komuś kopię oprogramowania. Sam fakt wykorzystania czy modyfikacji kodu wydanego na licencji GPL nie wywołuje obowiązku dystrybucji kodu, o ile nie nastąpiła dystrybucja oprogramowania w innej formie!

13.4.5 LGPL

GNU Lesser General Public License (LGPL) [8] jest to zmodyfikowana, bardziej zezwalająca wersja licencji GPL, oryginalnie pomyślana jako licencja dla bibliotek. Obecnie Free Software Foundation nie zaleca tego kroku ze względu na dysproporcje w liczbie bibliotek dostępnych na licencjach GPL i LGPL [6].

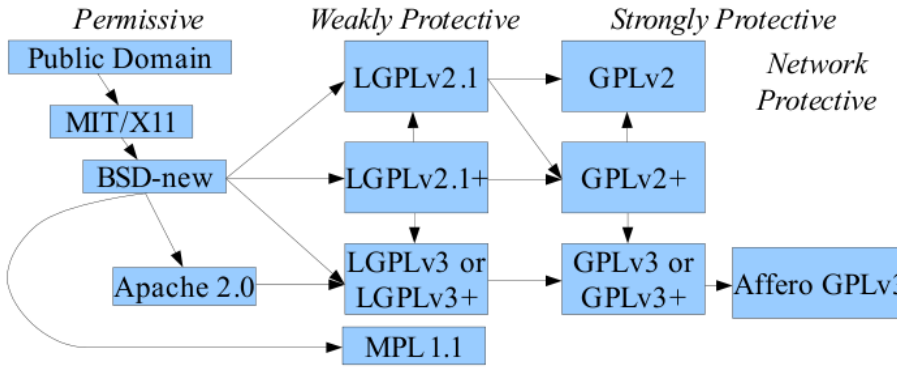
W odniesieniu do samego kodu licencja LGPL zachowuje się analogicznie jak licencja GPL – każdy odbiorca oprogramowania musi więc otrzymać kod źródłowy zarówno samego programu, jak i wprowadzonych do niego modyfikacji. Licencja LGPL pozwala jednak na łączenie kodu opartego na LGPL z kodem wydanym na dowolnej innej licencji, w tym komercyjnej. Warunkiem jest jednak, by kod LGPL dystrybuowany był niezależnie (np. jako biblioteka w systemie), a licencja oprogramowania zezwalała na modyfikację wersji wykorzystywanej biblioteki oraz reverse engineering potrzebny do debugowania tych zmian. W przeciwieństwie do GPL, linkowanie dynamiczne do biblioteki wydanej na licencji LGPL nie wymusza wykorzystania tejże licencji dla dzieła pochodnego.

13.4.6 AGPL

Affero General Public License (GNU AGPL) [7] jest tożsama z licencją GPL. O tyle o ile jednak licencja GPL obejmuje przypadki, kiedy użytkownik otrzymuje kopię oprogramowania, to licencja AGPL obejmuje również sytuacje, gdy użytkownik korzysta z oprogramowania jako z usługi bez otrzymania jego kopii (np. przez zdalny dostęp za pomocą przeglądarki internetowej). Licencja ta wymaga, by kod źródłowy aplikacji był dostępny dla każdego użytkownika, już na etapie logowania do aplikacji.

13.5 Zgodność licencji

Warunki nałożone przez licencje, zwłaszcza warunki nakładane na licencje dzieł pochodnych przez licencje wolne, w niektórych przypadkach mogą ograniczać czy wręcz uniemożliwiać mieszanie oprogramowania. Mówimy tutaj o tzw. kompatybilności licencji. Graf z rys. 13.1 obrazuje kompatybilność najpopularniejszych licencji. Strzałka od A do B oznacza, że można łączyć oprogramowanie wydane na licencji A z oprogramowaniem wydanym na licencji B, ale całość musi być wydana na licencji B.



Rysunek 13.1. Kompatybilność najpopularniejszych wolnych i otwartych licencji [27].

Licencje możemy tutaj podzielić na licencje zezwalające (ang. *permissive*), które nie nakładają praktycznie żadnych wymagań i mogą być łączone z dowolnymi innymi licencjami, nawet komercyjnymi. Kolejną warstwę stanowią tzw. licencje typu *weakly protective*. Licencje te dość silnie chronią bezpośrednio oprogramowanie na nich wydane, jednak pozwalają zazwyczaj na włączanie go do oprogramowania wydanego na innych licencjach. Kolejną warstwą są licencje typu *strongly protective*. Licencje te zazwyczaj nie umożliwiają łączenia oprogramowania z innymi licencjami, zazwyczaj z powodu konieczności udostępniania dzieła pochodnego na takiej samej licencji. Ostatnią, najsilniejszą warstwą są

licencje z grupy tzw. *network protective*. W tym przypadku działanie licencji jest bardzo szerokie, obejmuje również zdalny dostęp do oprogramowania, nie tylko jako bezpośrednio do kopii, a również jako usługi.

Zgodność licencji niesie za sobą istotne następstwa prawne rozstrzygające o możliwości (bądź nie) integracji oprogramowania. Licencje zezwalające w ogólności można włączyć do dowolnego innego oprogramowania wydanego na dowolnej licencji. Im licencja jest jednak bardziej „chroniąca" (ang. *protective*), tym mniejszy jest zakres dopuszczalnych do integracji licencji.

Szczególną uwagę należy zwrócić na licencje z rodziny licencji GPL. Każda z tych licencji wymaga, by dzieło pochodne wydane zostało na dokładnie takiej samej licencji, nawet z dokładnością do jej wersji. W praktyce oznacza to, że oprogramowanie wydane na licencji GPL w wersji 2 może być łączone tylko z oprogramowaniem wydanym na licencji GPL w wersji 2 lub innej, bardziej zezwalającej licencji. Nie ma możliwości, by oprogramowanie takie połączyć z wydanym na licencji GPL w wersji 3. Licencja ta wymaga wydania dzieła pochodnego również na licencji GPL w wersji 3 co stoi w sprzecznością wymogowi wydania tegoż na licencji GPL w wersji 2. Aby uniknąć tego problemu Free Software Foundation [2] zaleca, by używać dopisku „or later" (*lub późniejsze*, na rys. 13.1 oznaczone jako „+" przy nazwie licencji). Oznacza to, że autor wydając oprogramowanie na licencji np. „GPL wersja 2 lub późniejsze" w ogólności wydaje oprogramowanie na tejże wprost wskazanej licencji (GPL w wersji 2), jednak zgadza się również na włączanie oprogramowania do wydanego na dowolnej późniejszej wersji licencji GPL. Podejście to ma jednak pewną, zasadniczą wadę – zgadzamy się na wydanie naszego oprogramowania na licencji, której warunków nie znamy. Problem ten dało się zauważyć w chwili wprowadzenia wersji 3 licencji GPL, co do której część developerów wyrażało dość krytyczne stanowisko [4, 5, 19].

13.6 Wybór licencji

Wybór właściwej licencji ma duży wpływ na przyszłe zastosowanie wytworzonego przez nas kodu. Należy tutaj wziąć pod uwagę takie czynniki, jak nasz osobisty stosunek do upublicznianego kodu, zakres praw jakie chcemy udzielić, czy chcemy coś w zamian. W ogólności ważne jest również, by w ogóle jakąś licencję wybrać. W przeciwnym razie zakres możliwych zastosowań naszego kodu jest znacznie ograniczony. Nie możemy dla przykładu dystrybuować dzieł pochodnych.

Istotne jest również, by wziąć pod uwagę zgodność licencji oraz ich rozpoznawalność w środowisku. Część projektów tworzy własne licencje, które de facto są powtórzeniem już istniejących (np. licencja PostgreSQL [25]), co wzbudza zbędną dyskusję na temat jej natury, zasadności, konsekwencji czy zgodności z innymi licencjami [3, 11, 13, 18, 23]. Nierozważna zmiana treści licencji może naruszyć zasadę zgodności i doprowadzić wręcz do upadku projektu. W 2004, wraz z wydaniem XFree86 4.4, licencja MIT/X11 została zmodyfikowana, wzbudzając tym samym wielkie niezadowolenie w społeczności. Modyfikacja licencji polegała na dodaniu jednego, prostego warunku: „The end–user documentation

included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by The XFree86 Project, Inc (http://www.xfree86.org/) and its contributors", in the same place and form as other third–party acknowledgments. Alternately, this acknowledgment may appear in the software itself, in the same form and location as other such third–party acknowledgments.". W skrócie autorzy zażądali większego uznania, czy to w dokumentacji czy też w samym oprogramowaniu w postaci umieszczenia tamże stosownego zapisu. Tak zmodyfikowana licencja okazała się zgodna z GPL w wersji 3, ale nie z obowiązującą wówczas wersją 2. Narzucała również daleko idącą komplikację w utrzymaniu dokumentacji oprogramowania. Zmiana ta spowodowała zaprzestanie wykorzystania XFree86 przez wszystkie główne dystrybucje systemu Linux. Te, które tego nie zrobiły od razu (np. Mandrake), wycofały się z tej decyzji w przeciągu kliku miesięcy od wydania pierwszej wersji dystrybucji zawierającej kod na nowej wersji licencji.

Więc jaką licencję należy wybrać? Na to pytanie nie ma jednej słusznej odpowiedzi. Należy zastanowić się w dużej mierze, jakie mamy oczekiwania co do dalszego wykorzystania kodu oraz ewentualnych zobowiązaniach osób z niego korzystających.

Jeżeli napisaliśmy jakieś oprogramowanie i po prostu chcemy się nim podzielić z innymi, nie mamy nic przeciwko komercyjnemu wykorzystaniu go oraz nie zależy nam na ewentualnych poprawkach do tego kodu, najlepszą licencją będzie jedna z licencji zezwalających, takie jak MIT czy też dwuzdaniowa licencja BSD. Podejście to jest dobre również w sytuacji, gdy sami nie rozwijamy aktywnie kodu. FSF zaleca licencję MIT/X11 jako jednoznaczną (tożsama z nią licencja BSD ma co najmniej cztery wersje). Z drugiej jednak strony licencje BSD są bardziej precyzyjne. Licencja X11 zawiera nieprecyzyjne sformułowania „zajmować się oprogramowaniem" (ang. *to deal in the Software*) oraz „oprogramowanie i związane [dołączone] pliki dokumentacji" (ang. *this software and associated documentation files*), które mogą sprawiać problemy w interpretacji prawnej.

W sytuacji, gdy chcielibyśmy jednak utrudnić komercyjne zastosowanie naszego rozwiązania czy też chcemy mieć możliwość skorzystania z poprawek wprowadzonych przez użytkowników naszego kodu, należy zastosować licencję typu „copyleft", w szczególności właściwy wariant licencji GPL. W przypadku zgody na zastosowanie w rozwiązaniach komercyjnych warto również sięgnąć po licencję LGPL. W większości zastosowań wersja 3 licencji będzie odpowiednia. Wprowadza ona jednak pewne obostrzenia natury moralnej czy filozoficznej, np. brak możliwości wykorzystania oprogramowania wydanego na licencji GPL do implementacji mechanizmu DRM (ang. *Digital Rights Management*). Należy tutaj rozważyć, czy chcemy nakładać takie ograniczenia na użytkownika naszego oprogramowania.

W każdym przypadku musimy również pamiętać o patentach. Znaczna większość licencji w ogóle nie porusza tego problemu, wolne i otwarte oprogramowanie może więc być obciążone koniecznością posiadania licencji na patent, który został w tymże oprogramowaniu zaimplementowany. W sytuacji, gdy nasze oprogramowanie implementuje patent, do którego mamy prawa, warto zastosować licencję

Apache 2.0 lub (L)GPL w wersji 3. Licencje te jako jedne z niewielu poruszają kwestie patentów. Należy jednak pamiętać, że licencje z rodziny GNU są pod tym względem dość agresywne.

Jak już zdecydujemy się na właściwą z naszego punktu widzenia licencję, należy pamiętać, by właściwie oznaczyć projekt. Każda z licencji ma inne wymagania co do formy, w jakiej należy przekazać jej treść. W ogólności jednak użytkownik końcowy powinien otrzymać wraz z oprogramowaniem treść licencji. Ta najczęściej zawarta jest w pliku tekstowym, zwyczajowo nazwanym COPYING, umieszczonym w katalogu głównym projektu. Część licencji (przykładowo wszelkie odmiany licencji BSD) wymaga, by pliki z kodem źródłowym również oznaczyć stosowną informacją. Zazwyczaj czyni się to poprzez dodanie na początku każdego z plików komentarza zawierającego nazwę licencji oraz właściciela praw autorskich majątkowych. Często, zwłaszcza dla licencji krótkich, takich jak BSD czy MIT, przytacza się pełną treść licencji. Należy również zwrócić uwagę, że część licencji wymaga stosownego komentarza w wersji binarnej aplikacji. Niestety każdorazowo należy posilić się treścią samej licencji, gdyż nie ma tutaj ujednoliconej formy przekazywania stosownej informacji o licencji.

13.7 Podsumowanie

Wytwarzając oprogramowanie, oprócz kwestii jakościowych, musimy baczną uwagę zwrócić również na kwestie prawne. W sytuacji, gdy planujemy upubliczniać nasz kod, wybór właściwej licencji może być kluczowy dla potencjalnych użytkowników naszego rozwiązania, a czasami może wręcz uniemożliwić jego zastosowanie. Każdorazowo należy rozważyć poziom ochrony, jaki chcemy zapewnić swojemu rozwiązaniu, oraz zakres uprawnień, jakie nadamy użytkownikom końcowym. Niezależnie jednak od wyboru pamiętajmy, żeby jakąś licencję wybrać. W przypadku jej braku, nasze, nawet najbardziej doskonałe, oprogramowanie będzie dla użytkowników całkowicie bezużyteczne. W takiej sytuacji zakres uprawnień wynika wprost z prawa autorskiego i praw pokrewnych i jest dość wąsko zdefiniowany. Nadając licencję, niezależnie od jej charakteru, dajemy możliwość użytkownikowi nabycia praw dodatkowych, które mogą znacznie wpłynąć na użyteczność i popularność naszego rozwiązania, a często umożliwić jego dalszy rozwój nawet wtedy, kiedy my sami nie będziemy w stanie dalej tegoż oprogramowania rozbudowywać.

Bibliografia

1. GNU GENERAL PUBLIC LICENSE, Version 3, 29 June 2007. `https://www.gnu.org/licenses/gpl-3.0.html`, [Online, dostęp: 15.04.2021].
2. The Free Software Foundation (FSF). `https://www.fsf.org/`, [Online, dostęp: 19.04.2021].
3. alavoor: PostgreSQL Licence: GNU/GPL. `https://www.postgresql.org/message-id/20020120210341.90756.qmail%40web10407-.mail.yahoo.com` (2002), [Online, dostęp: 22.04.2021].

4. Bottomley, J.E., Chehab, M.C., Gleixner, T., Hellwig, C., Jones, D., Kroah-Hartman, G., Luck, T., Morton, A., Myklebust, T., Woodhouse, D.: The Dangers and Problems with GPLv3 (2006).
5. Debconf: Linus Torvalds says GPL v3 violates everything that GPLv2 stood for. https://www.youtube.com/watch?v=PaKIZ7gJlRU (2014), [Online, dostęp: 19.04.2021].
6. Free Software Foundation, Inc.: Czemu nie należy stosować Lesser GPL dla kolejnej biblioteki. https://www.gnu.org/licenses/why-not-lgpl.html, [Online, dostęp: 06.05.2021].
7. Free Software Foundation, Inc.: GNU AFFERO GENERAL PUBLIC LICENSE, Version 3, 29 June 2007. https://www.gnu.org/licenses/agpl-3.0.html, [Online, dostęp: 06.05.2021].
8. Free Software Foundation, Inc.: GNU LESSER GENERAL PUBLIC LICENSE, Version 3, 29 June 2007. https://www.gnu.org/licenses/lgpl-3.0.html, [Online, dostęp: 06.05.2021].
9. Free Software Foundation's Licensing and Compliance Lab: Frequently Asked Questions about the GNU Licenses. http://www.gnu.org/licenses/gpl-faq.html#IfInterpreterIsGPL, [Online, dostęp: 05.05.2021].
10. Free Software Foundation's Licensing and Compliance Lab: Rozmaite licencje i komentarze na ich temat#ModifiedBSD. https://www.gnu.org/licenses/license-list.pl.html#ModifiedBSD, [Online, dostęp: 05.05.2021].
11. Gunduz, D.: License clarification: BSD vs MIT. https://www.postgresql-archive.org/License-clarification-BSD-vs-MIT-td2017011.html (2009), [Online, dostęp: 22.04.2021].
12. Kancelaria Sejmu: Ustawa z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych. In: Dz.U. 1994 nr 24 poz. 83. (1994).
13. Lilly, N.: proposed improvements to PostgreSQL license. https://www.postgresql.org/message-id/3960ACA1.C911664A%40greatbridge.com (2000), [Online, dostęp: 22.04.2021].
14. Open Source Initiative: The 2-Clause BSD License. https://opensource.org/licenses/BSD-2-Clause, [Online, dostęp: 15.04.2021].
15. Open Source Initiative: The 3-Clause BSD License. https://opensource.org/licenses/BSD-3-Clause, [Online, dostęp: 15.04.2021].
16. Open Source Initiative: The MIT License. https://opensource.org/licenses/MIT, [Online, dostęp: 15.04.2021].
17. Open Source Initiative: Zero-Clause BSD (0BSD). https://opensource.org/licenses/0BSD, [Online, dostęp: 23.04.2021].
18. Page, D.: PostgreSQL licence - status? https://web.archive.org/web/201608080930-31/http://www.crynwr.com/cgi-bin/ezmlm-cgi?17:mmp:969 (2010), [Online, dostęp: 22.04.2021].
19. Petreley, N.: A fight against evil or a fight for attention? https://www.linuxjournal.com/content/fight-against-evil-or-fight-attention (2006), [Online, dostęp: 19.04.2021].
20. Schmitz, P.E.: Why viral licensing is a ghost. https://joinup.ec.europa.eu/collection/eupl/news/why-viral-licensing-ghost (2015), [Online, dostęp: 05.05.2021].

21. SPDX: BSD 4-Clause Original or Old License. https://spdx.org/licenses/BSD-4-Clause, [Online, dostęp: 23.04.2021].
22. The Apache Software Foundation: Apache License, version 2.0. https://www.apache.org/licenses/LICENSE-2.0.html, [Online, dostęp: 05.05.2021].
23. The Hermit Hacker: PostgreSQL & the BSD License. https://www.postgresql.org/message-id/Pine.BSF.4.21.0007052208380.33627-100000%40thelab.hub.org (2000), [Online, dostęp: 22.04.2021].
24. The Mozilla Foundation: Mozilla Public License. https://www.mozilla.org/en-US/MPL/, [Online, dostęp: 06.05.2021].
25. The PostgreSQL Global Development Group: PostgreSQL licence. https://www.postgresql.org/about/licence/, [Online, dostęp: 22.04.2021].
26. The SQLite Team: SQLite Is Public Domain. https://www.sqlite.org/copyright.html, [Online, dostęp: 06.05.2021].
27. Wheeler, D.A.: The Free-Libre / Open Source Software (FLOSS) License Slide. https://dwheeler.com/essays/floss-license-slide.html (01 2017), [Online, dostęp: 15.04.2021].
28. Wikipedia contributors: Comparison of free and open-source software licences — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Comparison_of_free_and_open-source_software_licences&oldid=1017649175 (2021), [Online, dostęp: 23.04.2021].

14. Implementacja wykrywalnych usług typu REST na platformie Jakarta EE

Michał Wójcik

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
michal.wojcik@eti.pg.edu.pl

Streszczenie
Niniejszy rozdział przedstawia propozycję w jaki sposób może być realizowana implementacja wykrywalnych usług sieciowych opartych na stylu architektonicznym REST na platformie Jakarta EE. Zostały tutaj przedstawione zarówno podstawy teoretyczne niezależne od zastosowanej platformy technologicznej, jak i szczegóły implementacji w technologii JAX-RS wchodzącej w skład platformy Jakarta EE. W szczególności zostały tutaj przedstawione sposoby poprawnego budowania hierarchicznego API zgodnie z modelem dojrzałości Richardsona jak i techniki opisywania reprezentacji zwracanej przez usługi sieciowe zgodnie ograniczeniem HATEOAS i językiem opisu HAL. Wszystkie aspekty teoretyczne poruszane w rozdziale zostały poparte praktyczną implementacją dostępną publicznie w postaci repozytorium kodu opartego na systemie kontroli wersji Git[1].

Słowa kluczowe: REST, REST API, Jakarta EE, Java EE, JAX-RS, HAL, HATEOAS, Richardson Maturity Model, Discoverable Services

14.1 Wprowadzenie

Usługi sieciowe to najogólniej mówiąc oprogramowanie zaprojektowane do obsługi interakcji między maszyną a maszyną za pośrednictwem sieci [4]. Interakcja między maszynami powinna cechować się interoperacyjnością. Wspomniana definicja zakłada jeszcze, że usługi powinny być opisane przez format, który można przetwarzać maszynowo, w szczególności WSDL (*Web Services Description Language*) [7], a komunikacja z nimi powinna zachodzić za pomocą komunikatów SOAP (*Simple Object Access Protocol*) [18] wymienianych zazwyczaj w formacie XML (*Extensible Markup Language*) [6] za pomocą protokołu HTTP (*Hypertext Transfer Protocol*) [11].

Istotne tutaj jest to, że protokół HTTP jest wykorzystywany jedynie jako protokół transportowy w warstwie aplikacji gdzie SOAP jest protokołem wymiany wiadomości. Równie dobrze, zamiast z protokołu HTTP, można skorzystać z protokołu SMTP (*Simple Mail Transfer Protocol*) [23] i nadal będziemy mieli do

[1] https://git.pg.edu.pl/p650304/jakartaee-discoverable-rest

czynienia z usługą sieciową. Oznacza to, że z punktu widzenia wywoływania usług sieciowych opartych na protokole SOAP, istotne są wyłącznie informacje zawarte w wiadomości, której format jest zdefiniowany jako koperta zawierająca nagłówki i ciało wiadomości. Od razu widać tutaj podobieństwo do protokołu HTTP, gdzie zarówno żądanie, jak i odpowiedź składają się z listy nagłówków oraz ciała wiadomości. Jednak w przypadku wspomnianych usług sieciowych nagłówki protokołu HTTP nie mają znaczenia w wymianie komunikatów z usługą. Oczywiście nadal są istotne dla samego transportu wiadomości, chociażby czy wiadomość udało się w ogóle dostarczyć.

W praktyce podczas wytwarzania aplikacji webowych coraz rzadziej słyszy się o SOAP czy WSDL na rzecz ogólnego stylu architektonicznego REST (*Representational State Transfer*) [12]. Raporty wykonane przez takie firmy jak JetBrains, Postman i SmartBear wykazują, że ponad 80% ankietowanych pracowników, związanych z wytwarzaniem oprogramowania, wskazuje REST jako najczęściej używany i najlepiej znany styl architektoniczny[19, 31, 35].

Czym dokładnie są usługi oparte na stylu REST? Bardzo często pierwsze skojarzenie wskazuje, że są to usługi udostępniające dostęp do określonych zasobów za pośrednictwem protokołu HTTP w formacje JSON (*JavaScript Object Notation*) [5] jednak jest to tylko niewielka część prawdy.

O ile pojęcie REST faktycznie definiuje możliwość zarządzania zasobami, to w żaden sposób nie ogranicza się wyłącznie do protokołu HTTP lub jakiegokolwiek innego konkretnego protokołu. O ile nie jest to szczególnie popularne i jednak większość usług opartych na architekturze REST wykorzystuje protokół HTTP, to poprawne będzie wykorzystanie innego protokołu jak FTP (*File Transfer Protocol*) [30] czy nawet Gopher [24] lub WAIS (*Wide Area Information Server*) [34].

Styl architektoniczny REST nie definiuje także formatu, w jakim dostępne są reprezentacje zasobów. Tak naprawdę specyfikacja JSON powstała znacznie później niż pierwsza propozycja zdefiniowania architektury REST. Jako analogię można tutaj podać szeroko wykorzystywaną technologię AJAX (*Asynchronous JavaScript and XML*) [37], czasami błędnie utożsamianą z REST, która wykorzystywała format XML do przesyłania danych pomiędzy serwerem a klientem, którym zazwyczaj jest przeglądarka internetowa. Fakt, że teraz wykorzystuje głównie (nie jedynie) format JSON wynika z popularności tego formatu. W połączeniu z REST należy skorzystać z takiego formatu, jaki spełnia postawione w projekcie wymagania biznesowe. Co do wspomnianej już technologi AJAX to nic nie stoi na przeszkodzie, aby asynchroniczne żądania z przeglądarki zawierały wiadomość w formacie JSON i były skierowane do API (*Application Programming Interface*) zgodnego z REST.

Jednym z podstawowych założeń REST jest taka realizacja komunikacji, żeby była ona bezstanowa. Oznacza to, że każde żądanie musi zawierać wszystkie elementy koniecznie do jego realizacji. W szczególności oznacza to, że interpretacja żądań nie może bazować na stanie sesji przechowywanej w pamięci serwera lub klienta. Takie podejście skutkuje określonymi korzyściami:

- widoczność jest poprawiona przez fakt, że interpretacja żądania nie musi wychodzić poza dane zawarte w żądaniu aby określić jego pełny charakter,
- niezawodność jest zwiększona, ponieważ proces odzyskiwania po częściowych awariach jest prostszy,
- skalowalność jest ulepszona, ponieważ brak konieczności przechowywania stanu pomiędzy żądaniami pozwala na sprawne zwalnianie zasobów,
- implementacja jest uproszczona, ponieważ serwer nie musi zarządzać wykorzystaniem zasobów pomiędzy żądaniami.

O ile interpretacja żądań nie może bazować na stanie sesji, to nic nie stoi na przeszkodzie aby odpowiedzi od serwera były poddane procesowi buforowania. Trudno sobie wyobrazić korzystanie z Internetu bez opcji buforowania dla statycznych zasobów jak style CSS (*Cascading Style Sheets*) [21], obrazki czy też pliki audio i video. Architektura REST nie mówi co prawda, w jaki sposób implementować buforowanie, ale wymaga, aby odpowiedź od serwera zawierała etykietę informującą, że może być poddana buforowaniu.

Kolejnym istotnym elementem architektury REST jest tak zwany ujednolicony interfejs. Na to pojęcie składa się kilka elementów:

- identyfikacja zasobów w żądaniach – każdy zasób posiada własny identyfikator dzięki czemu serwer na podstawie już pojedynczego żądania może w pełni zidentyfikować, którego zasobu ono dotyczy,
- manipulacja zasobami poprzez reprezentację – komponenty REST (klient, serwer) wykonują akcje na zasobie, używając reprezentacji do przechwytywania bieżącego lub zamierzonego stanu tego zasobu i przesyłania tej reprezentacji między komponentami,
- kompletny opis w wiadomości – pojedyncza wiadomość powinna zawierać wystarczające informacje aby mogła być poprawnie przetworzona, tyczy się to wykorzystania standardowych typów mediów do wskazania semantyki, a także wyraźnego wskazania, czy dane mogą być buforowane,
- hipermedia jako silnik stanu aplikacji (ang. *Hypertext As The Engine Of Application State* – HATEOAS) – pojęcie hipermedia rozszerzające pojęcia hipertekst zakłada organizację danych w postaci pojedynczych zamkniętych całości połączonych hiperłączami, oznacza to, że pojedyncze zasoby mogą być ze sobą połączone za pomocą odnośników pozwalającymi nawigować między nimi bez znajomości z góry struktury zasobów.

14.2 Przykładowe zasoby

Kolejne części tego rozdziału wymagają zdefiniowania przykładowych zasobów jakie będą prezentowane. Zarówno dla przykładów teoretycznych, jak i implementacji dołączonej do tego opracowania, wybrano jeden spójny motyw przewodni realizujący zarządzanie postaciami w grze RPG (*Role Playing Game*) [1].

W grach RPG każdy z graczy definiuje swoją postać, która może być opisana przez cechy fabularne, jak i te związane z mechaniką gry. Do pierwszych można zaliczyć takie cechy jak imię, pochodzenie, wiek i wygląd (na przykład w postaci

portretu). Do tych drugich można zaliczyć m.in. siłę, konstytucję i charyzmę. Dodatkowo, zazwyczaj każdy z graczy wybiera dla swojej postaci profesję, którą w zależności od przyjętego świata może być np. wojownik, bard czy łotrzyk. Ponadto każda z postaci może posiadać ekwipunek (listę przedmiotów) wykorzystywany podczas przygody.

Poniżej zostały przedstawione przykładowe reprezentacje poszczególnych zasobów w formacie YAML (*YAML Ain't Markup Language*) [3].

Skoro głównym zasobem w zaimplementowanym przykładzie są bohaterowie gry PRG, to podstawową funkcjonalnością API pozwalającego na zarządzanie nimi jest zwrócenie listy wszystkich dostępnych bohaterów. Założono tutaj, że pojedynczy bohater jest jednoznacznie identyfikowany przez swoje imię. Oznacza to, że nie będzie możliwe dodanie dwóch różnych bohaterów posiadających to samo imię. Przykładowa lista wygląda następująco:

```
---
characters:
  - name: Calvian
  - name: Eloise
  - name: Uhlbrecht
  - name: Zereni
```

Lista dostępnych bohaterów pozwala na sprawdzenie szczegółów dotyczących konkretnego bohatera. Pojedynczy bohater będzie opisany przez listę cech fabularnych i tych związanych z mechaniką gry, swoją profesję oraz listę dostępnych przedmiotów. Reprezentacja pojedynczego bohatera wygląda następująco:

```
---
name: Calvian
age: 18
background: A young bard with some infernal roots.
profession: Bard
charisma: 16
constitution: 12
strength: 8
items:
  - name: Lute
  - name: Rainbow Hat
```

Opis bohatera zawiera tylko nazwy posiadanych przez niego przedmiotów. Poznanie szczegółów przedmiotu wymaga zdobycia dokładniejszego opisu:

```
---
name: Lute
description: Simple music instrument.
```

Do stworzenia bohatera potrzebna jest informacja o wszystkich dostępnych profesjach. Podobnie jak w przypadku bohaterów, są one identyfikowane przez nazwę. Przykładowa lista wygląda:

```
---
professions:
  - name: Bard
  - name: Cleric
  - name: Rogue
  - name: Warrior
```

W celu uproszczenia implementacji oraz reprezentacji zasobów przyjęto, że profesje w prezentowanych przykładach różnią się od siebie wyłącznie nazwą oraz opisem fabularnym. W praktyce dochodziłyby dedykowane właściwości oraz mocne i słabe strony. Przykładowy opis profesji:

```
---
name: Bard
description: >
  Bards encourages their teammates
  with the sound of their song.
```

14.3 Model dojrzałości

Model dojrzałości według Leonarda Richardsona to sposób na ocenę interfejsu API zgodnie z ograniczeniami REST. Zakłada on skalę, na którą składają się cztery poziomy, gdzie najniższym jest poziom 0 [16, 33]:

0. wykorzystanie protokołu HTTP jedynie jako protokołu transportowego,
1. wyszczególnienie zasobów za pomocą identyfikatorów URI (Uniform Resource Identifier),
2. wykorzystanie czasowników HTTP i kodów odpowiedzi jako część wiadomości,
3. wykorzystanie HATEOAS.

Poniżej zostało przedstawione opracowanie kolejnych poziomów modelu dojrzałości wraz z przykładami reprezentacji zasobów zdefiniowanych zasobów oraz operacji na nich.

14.3.1 Protokół HTTP jako warstwa transportowa

Pierwszym punktem jest wykorzystanie protokołu HTTP jako protokołu transportowego. Od razu widać tutaj podobieństwo do usług sieciowych opartych na protokole SOAP. W przypadku tych usług zazwyczaj wszystkie żądania realizowane są poprzez metodę `POST` protokołu HTTP. Oczywiście teoretycznie możliwe jest wykorzystanie metody `GET` do pobierania reprezentacji zasobów bez zmieniania ich w jakikolwiek sposób, jednak w praktyce jest to rozwiązanie rzadziej wykorzystywane w implementacji.

Zakładając, że protokół HTTP jest wykorzystywany jedynie do transportu, a wszelkie informacje potrzebne do zrealizowania żądania są zawarte w jego treści, żądanie pobrania reprezentacji listy bohaterów mogłoby wyglądać w taki sposób:

```
POST /api HTTP/1.1
Content-Type: application/json

{
  "method": "get_characters",
  "resource": "characters"
}
```

i skutkowałoby następującą odpowiedzą:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "result": [
    {
      "name": "Calvian"
    },
    {
      "name": "Eloise"
    },
    {
      "name": "Uhlbrecht"
    },
    {
      "name": "Zereni"
    }
  ]
}
```

Idąc dalej, żądanie pobrania reprezentacji konkretnego bohatera mogłoby być wysyłane na ten sam adres i mieć taką postać:

```
POST /api HTTP/1.1
Content-Type: application/json

{
  "method": "get_character",
  "params": {
    "name": "Calvian"
  }
}
```

Z kolei żądanie stworzenia nowego bohatera mogłoby wyglądać następująco:

```
POST /api HTTP/1.1
Content-Type: application/json
```

```
{
  "method": "create_character",
  "params": {
    "name": "Sigrid",
    "background": "No one special.",
    "age": 18,
    "strength": 12,
    "constitution": 12,
    "charisma": 12,
    "profession": "Bard"
  }
}
```

Trzymając się założenia, że nie można dodać dwóch bohaterów o takim samym imieniu, ponowne wywołanie powyższego żądania mogło by skutkować błędem w postaci następującej odpowiedzi:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "error": {
    "message": "Bad Request"
  }
}
```

Zaprezentowane powyżej przykłady komunikacji pokazują, że wszelkie informacje potrzebne do poprawnego zinterpretowania żądania i odpowiedzi znajdują się w ramach wiadomości. Niezależnie, czy pobierana jest reprezentacja zasobu, czy tworzony jest nowy zasób, zawsze wykorzystywana jest metoda `POST` protokołu HTTP. Podobnie niezależnie, czy żądanie zostało zrealizowane poprawnie, czy nie, zawsze zwracany jest kod odpowiedzi `200 OK`. Dodatkowo żądania są zawsze wysyłane na ten sam adres, zdefiniowany w przykładzie jako `/api`. Dla uproszczenia zapisu pominięto protokół, nazwę hosta, port i nazwę aplikacji.

Przedstawiony sposób komunikacji jest typowy dla mechanizmów opartych na protokole wywoływania zdalnych procedur RPC (*Remote Procedure Call*) [27]. Zastosowana składnia wiadomości nawiązuje składni JSON-RPC [26]. Podobny schemat byłby zastosowany w przypadku XML-RPC [38] czy SOAP.

14.3.2 Identyfikacja zasobów

Kolejnym punktem w modelu dojrzałości jest identyfikacja zasobów w postaci identyfikatorów URI. W przeciwieństwie do poprzedniego poziomu, zamiast wykonywać żądania na jeden zdefiniowany zasób, np. `/api`, żądania powinny być kierowane na adresy reprezentujące zasoby, których dotyczą.

Żądanie pobrania reprezentacji listy bohaterów skierowane na adres definiujący listę bohaterów wyglądałoby następująco:

```
POST /api/characters HTTP/1.1
Content-Type: application/json

{
  "method": "get"
}
```

Z kolei żądanie pobrania reprezentacji konkretnego bohatera wysłane na adres identyfikujący wyłącznie zasób wskazanego bohatera miałoby postać:

```
POST /api/characters/Calvian HTTP/1.1
Content-Type: application/json

{
  "method": "get"
}
```

Wykorzystanie identyfikatorów zasobów pozwala także na przedstawienie ich hierarchii. Przykładowo zamiast pobierać reprezentację wszystkich bohaterów, można wysłać żądanie tylko o bohaterów należących do konkretnej profesji:

```
POST /api/professions/Bard/characters HTTP/1.1
Content-Type: application/json

{
  "method": "get"
}
```

Idąc dalej, zamiast ustawiać nowo dodawanemu bohaterowi wybraną profesję, można w zamian dodać go bezpośrednio do kolekcji bohaterów z danej profesji:

```
POST /api/professions/Bard/characters HTTP/1.1
Content-Type: application/json

{
  "method": "create_character",
  "params": {
    "name": "Sigrid",
    "background": "No one special.",
    "age": 18,
    "strength": 12,
    "constitution": 12,
    "charisma": 12
  }
}
```

Głównym problemem poziomu 0 modelu dojrzałości było uzyskanie efektu czarnej skrzynki. Wprowadzenie tylko jednego punktu końcowego w postaci `/api`

nie dawało jasnej informacji, na czym tak właściwie są wykonywane operacje albo czego reprezentacja jest zwracana. Analogicznie sytuacja jest w mechanizmach opartych na jakimkolwiek protokole RPC. Wykorzystanie identyfikatora URI do identyfikacji zasobu sprawia, że już nie tylko treść żądania czy odpowiedzi jest potrzebna do interpretacji komunikatu. HTTP powoli przestaje być tylko protokołem transportowym.

14.3.3 Wykorzystanie protokołu HTTP

Do tej pory, niezależnie od intencji w żądaniach, wykorzystywana była metoda `POST` protokołu HTTP. Pierwszą myślą jest, czemu nie zastosować metody `GET` do sytuacji, gdy chodzi tylko o pobranie reprezentacji danego zasobu? Uprościłoby to znacznie wszelkie żądania pobierania reprezentacji. Przykład takiego żądania dla pobrania reprezentacji wszystkich bohaterów:

```
GET /api/characters HTTP/1.1
```

oraz pobrania reprezentacji jednego wskazanego bohatera:

```
GET /api/characters/Calvian HTTP/1.1
```

Zgodnie ze specyfikacją protokołu HTTP, metoda `GET` służy do żądania pobrania reprezentacji danego zasobu i nie może służyć jego zmodyfikowaniu. Skoro wykorzystywana jest odpowiednia metoda protokołu HTTP, to dlaczego nie skorzystać też z gotowych kodów odpowiedzi? Na poziomach 0 i 1, niezależnie od wyniku wszystkie odpowiedzi były opatrzone kodem `200 OK`, a ewentualne błędy były przekazywane w ramach ciała odpowiedzi. Zgodnie ze specyfikacją protokołu HTTP [11], w przypadku żądań `GET`, które zakończyły się znalezieniem zasobu poprawnym kodem jest `200 OK`. W przypadku nieznalezienia zasobu odpowiednim wydaje się kod `404 NOT FOUND`. Odpowiedź w przypadku nieznalezienia określonego zasobu nie musiałaby zawierać dodatkowych informacji i wyglądałyby następująco:

```
HTTP/1.1 404 NOT FOUND
```

Przy okazji omawiania żądania `GET` należy zwrócić uwagę na istotny fakt, że jest ono idempotentne. Oznacza to, że efekty uboczne dowolnej liczby żądań większej od zera będą takie same jak w przypadku pojedynczego żądania. Jest to istotna i pożądana cecha dla architektury opartej na REST.

Skoro skutecznie można wykorzystać metodę `GET`, to czemu nie wykorzystać kolejnych metod protokołu HTTP do określenia akcji wykonywanej na zasobie? Przykładowo do usunięcia określonego zasobu bohatera wystarczy wysłać żądanie:

```
DELETE /api/professions/Bard/characters/Calvian HTTP/1.1
```

Wśród możliwych odpowiedzi zdefiniowanych przez specyfikację protokołu HTTP, w przypadku REST najodpowiedniejszą wydaje się kod `204 NO CONTENT` oznaczający poprawne wykonanie żądania i brak ciała odpowiedzi. Podobnie

jak żądanie `GET`, żądanie `DELETE` może być idempotentne. Niezależnie, ile razy zostanie wysłane żądanie usuniętego zasobu, po stronie serwera zasób ten cały czas będzie usunięty.

Można natrafić na dyskusje, jaki kod odpowiedzi powinien być zwrócony w przypadku próby usunięcia nieistniejącego zasobu. Naturalnym wydaje się tutaj `404 NOT FOUND`. Oznaczałoby to jednak, że podczas usuwania istniejącego zasobu, pierwsze wywołanie zwróci `204 NOT CONTENT`, a każde kolejne `404 NOT FOUND`. Czy w takiej sytuacji żądanie jest dalej idempotentne? Definicja idempotentności wskazuje na efekty uboczne, które można rozumieć jako stan zasobu, a nie treść odpowiedzi. Dodatkowo specyfikacja protokołu HTTP jasno mówi, że kod `404 NOT FOUND` powinien zostać zwrócony w sytuacji, gdy nie udało się znaleźć wskazanego zasobu po stronie serwera.

Po zrealizowaniu usuwania zasobów należy się zastanowić nad możliwością ich dodawania. W wielu opracowaniach można znaleźć błędne informacje, że do dodawania nowego zasobu należy zawsze korzystać z metody `POST`. Zgodnie ze specyfikacją protokołu HTTP, metoda `POST` służy do przesłania danych na serwer. To, w jaki sposób zostaną one obsłużone, nie wynika wprost ze specyfikacji a ze zdefiniowanego kontraktu między klientem a serwerem. Te same opracowania wskazują, że metoda `PUT` powinna być wykorzystywana jedynie do aktualizacji istniejącego zasobu. Ponownie jest to błędem. Specyfikacja HTTP jasno mówi, że metoda `PUT` służy do zapisania przesłanej encji pod wskazanym adresem. Oznacza to, że można jej użyć zarówno do stworzenia nowego zasobu, jak i całkowitego zaktualizowania istniejącego. W pierwszym przypadku serwer powinien odpowiedź kodem `201 CREATED`, a w drugim `204 NO CONTENT`. Dopuszczalna jest także odpowiedź `200 OK` w sytuacji, gdy serwer załącza ciało odpowiedzi, np. reprezentację nowo dodanego zasobu.

Zgodnie z powyższym, dodanie nowego lub zaktualizowanie istniejącego bohatera mogłoby wyglądać następująco:

```
PUT /api/professions/Bard/characters/Sigrid HTTP/1.1
Content-Type: application/json

{
  "background": "No one special.",
  "age": 18,
  "strength": 12,
  "constitution": 12,
  "charisma": 12
}
```

Podobnie jak w przypadku żądań `GET` i `DELETE` żądanie `PUT` jest idempotentne. Nie ma znaczenia, ile razy zostanie wysłane takie samo żądanie stworzenia/aktualizacji zasobu to stan po stronie serwera będzie identyczny. Nie istnieje tutaj ryzyko dodania wielu kopii zasobu, ponieważ żądanie jest wysyłane na konkretny adres, gdzie zasób ma być stworzony.

Kolejnym błędnym zrozumieniem metody `PUT` jest wykorzystywanie jej do częściowej aktualizacji zasobu przez przesyłanie częściowej reprezentacji. O ile

jest to na pewno praktyczne z punktu widzenia API, aby wystawić mechanizm pozwalający na aktualizację tylko części zasobu, to z punktu widzenia specyfikacji HTTP metoda `PUT` nie jest poprawnym rozwiązaniem.

W przypadku częściowej aktualizacji zasobu wskazane jest stosowanie metody `PATCH` [9] pozwalającej na przesłanie reprezentacji różnic względem oryginalnego zasobu. Przykładowe żądanie zmiany jedynie fabularnych cech bohatera może wyglądać następująco:

```
PATCH /api/professions/Bard/characters/Sigrid HTTP/1.1
Content-Type: application/json

{
  "background": "Someone well known.",
  "age": 23
}
```

O ile powyższe żądanie z pewnością jest idempotentne (nieważne, ile razy zostanie wysłane, stan zasobu po stronie serwera zostanie taki sam), to sama definicja metody `PATCH` w specyfikacji protokołu HTTP nie zapewnia idempotentności. Wynika to z faktu, że możliwe jest zaprojektowanie takiej obsługi zmian, że kolejne żądania będą skutkowały różnymi stanami zasobu po stronie serwera.

Odpowiednim przykładem będzie operacja dodania nowego przedmiotu do ekwipunku bohatera. Załóżmy żądanie pobrania reprezentacji bohatera:

```
GET /api/professions/Bard/characters/Calvian HTTP/1.1
```

na, które odpowiedzą jest:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "items": [
    {
      "name": "Lute"
    },
    {
      "name": "Rainbow Hat"
    }
  ],
  "age": 18,
  "background": "A young bard with some infernal roots.",
  "charisma": 16,
  "constitution": 12,
  "name": "Calvian",
  "profession": "Bard",
  "strength": 8
```

```
}
```

Następnie załóżmy następujące żądanie modyfikujące zasób bohatera o nową listę przedmiotów:

```
PATCH /api/professions/Bard/characters/Calvian HTTP/1.1
Content-Type: application/json

{
  "items": [
    {
      "name": "Lute",
      "description": "Simple music instrument."
    }
  ]
}
```

Ponowne pobranie reprezentacji zasobu bohatera po wywołaniu powyższego żądania powinno zwrócić następujący wynik:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "items": [
    {
      "name": "Lute",
    }
  ],
  "age": 18,
  "background": "A young bard with some infernal roots.",
  "charisma": 16,
  "constitution": 12,
  "name": "Calvian",
  "profession": "Bard",
  "strength": 8
}
```

Teraz załóżmy, że `PATCH` nie nadpisywałoby listy przedmiotów, a raczej zawierało informację o operacji dodania nowego elementu:

```
PATCH /api/professions/Bard/characters/Calvian HTTP/1.1
Content-Type: application/json

{
  "items": [
    {
      "operation": "add",
```

```
      "name": "Violin",
      "description": "Classic music instrument."
    }
  ]
}
```

Odpowiedź na powyższe żądanie mogłaby mieć następującą postać:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "items": [
    {
      "name": "Lute",
    },
    {
      "name": "Violin",
    }
  ],
  "age": 18,
  "background": "A young bard with some infernal roots.",
  "charisma": 16,
  "constitution": 12,
  "name": "Calvian",
  "profession": "Bard",
  "strength": 8
}
```

Kolejne wywołania żądania `PATCH` dodającego skrzypce sprawiałoby, że zasób bohatera zmieniałby się (rosłaby liczba posiadanych skrzypiec). Oczywiście dodatkowe zabezpieczenie obsługi żądania niepozwalające dodać więcej niż jedną sztuka danego przedmiotu sprawiłoby, że żądanie mogłoby być idempotentne.

Do tej pory udało się zrealizować pobieranie, dodawanie, modyfikowanie i usuwanie zasobów bez wykorzystania żądania `POST`. Czy w takim razie jest dla niego miejsce w API opartym na stylu architektonicznym REST? Dobrym przykładem jest dodawanie nowych elementów do kolekcji korzystając z punktu końcowego wskazującego na kolekcję. Zgodnie ze specyfikacją protokołu HTTP, metoda `POST` może być wykorzystana do dodania nowego elementu do kolekcji. Rozważmy następujące żądanie:

```
POST /api/professions/Bard/characters/ HTTP/1.1
Content-Type: application/json

{
  "name": "Sigrid",
  "background": "No one special.",
```

```
  "age": 18,
  "strength": 12,
  "constitution": 12,
  "charisma": 12
}
```

Rezultatem powyższego żądania powinno być dodanie nowego zasobu bohatera do kolekcji bohaterów. Poprawną odpowiedzią w tym przypadku jest kod `201 CREATED`. Dodatkowo odpowiedź powinna zawierać lokalizację nowego zasobu w postaci nagłówka `Location`:

```
HTTP/1.1 201 CREATED
Location: /api/professions/Bard/characters/Sigrid
```

Czy powyższe żądanie jest idempotentne? Zgodnie ze specyfikacją protokołu HTTP, metoda `POST` nie jest domyślnie idempotentna. Wielokrotne wysłanie powyższego żądania może spowodować dodanie kilku kopii tego samego bohatera. W opracowaniach można znaleźć propozycję rozwiązania tego problemu w postaci nowego nagłówka `Idempotency-Key`. W przypadku przykładu dołączonego do tego rozdziału taki zabieg nie jest potrzebny, ponieważ z punktu widzenia implementacji nie jest możliwe dodanie kilku bohaterów o tym samym imieniu lub kilku profesji o tej samej nazwie. W sytuacji, gdy to samo żądanie `POST` zostanie wysłanie więcej niż jeden raz, po stronie serwera nic się nie zmieni, a klient otrzyma odpowiedź:

```
HTTP/1.1 400 Bad Request
```

14.3.4 Wykorzystanie HATEOAS

W ramach pierwszego poziomu dojrzałości zostały zdefiniowane zasoby identyfikowane przez URI. Następnie zostały one ułożone w hierarchiczną strukturę definiującą relacje miedzy nimi. Finalnie, w ramach poziomu drugiego zostały określone standardowe metody, w jaki sposób wykonywać określone operacje na zasobach. Pojawia się problem, że klient musi być świadomy struktury tych zasobów, aby móc skutecznie wchodzić z nimi w interakcję. Pobierając reprezentację jakiegokolwiek zasobu klient nie ma pojęcia o jego relacjach z innymi zasobami.

Rozwiązaniem tego jest zastosowanie hipermedia jako silniku stanu aplikacji, czyli HATEOAS. Zasoby, które są ze sobą powiązane, powinny być połączone hiperłączami pozwalającymi na nawigowanie między nim. Idealną sytuacją jest, gdy klientowi wystarczy główny punkt końcowy, z którego może przejść na dowolny zasób.

Właściwym początkiem będzie tutaj żądanie `GET` na główny punkt końcowy:

```
GET /api HTTP/1.1
```

Odpowiedź, która pozwalałby na dalsze odkrywanie zasobów mogłaby wyglądać następująco:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "links": [
    {
      "rel": "self",
      "href": "/api"
    },
    {
      "rel": "/api/relations/professions",
      "href": "/api/professions"
    },
    {
      "rel": "/api/relations/characters",
      "href": "/api/characters"
    }
  ],
  "name": "Simple RPG"
}
```

Z kolei odpowiedź na żądanie pobrania reprezentacji listy dostępnych profesji miałoby postać:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "links": [
    {
      "rel": "self",
      "href": "/api/professions"
    }
  ],
  "professions": [
    {
      "name": "Bard",
      "links": [
        {
          "rel": "self",
          "href": "/api/professions/Bard"
        }
      ]
    },
    {
      "name": "Cleric",
      "links": [
```

```
        {
          "rel": "self",
          "href": "/api/professions/Cleric"
        }
      ]
    },
    {
      "name": "Rogue",
      "links": [
        {
          "rel": "self",
          "href": "/api/professions/Rogue"
        }
      ]
    },
    {
      "name": "Warrior",
      "links": [
        {
          "rel": "self",
          "href": "/api/professions/Warrior"
        }
      ]
    }
  ]
}
```

W obu przypadkach można zauważyć, że reprezentacja zasobu zawarta w odpowiedzi została wzbogacona o wpis `links` realizujący odnośniki do innych zasobów. Jako że REST jest stylem architektonicznym, nie definiuje dokładnie sposobu, w jaki powinny być definiowane relacje pomiędzy zasobami. Powyższy przykład inspirowany jest specyfikacją standardu Atom [29]. Każdy z odnośników został opisany przed dwie wartości: `href` definiujący identyfikator URI oraz `rel` definiujący relację. Należy zwrócić uwagę, że relacje zostały przedstawione w postaci odnośników do zasobu, który wyjaśnia ich znaczenie. Nie ma potrzeby definiowania relacji dobrze znanych, czyli tych dostępnych w rejestrze relacji[2] IANA (*Internet Assigned Numbers Authority*) [28], np. relacji `self`.

Teraz klient nie musi znać dokładnie struktury zasobów, ponieważ odpytując już główny punkt końcowy dostaje informacje o podstawowych dwóch zbiorach: bohaterów i profesji oraz odnośniki wyjaśniające, czym są te zbiory. Idąc dalej, w momencie, gdy klient pobierze reprezentację listy profesji dostanie nie tylko ich listę, ale też do każdej profesji zostanie dołączony odnośnik pozwalający pobrać jej pełną reprezentację. W podobny sposób powinny być zrealizowane reprezentacje pozostałych zasobów. Reprezentacja profesji zawierałaby odnośnik

[2] `https://www.iana.org/assignments/link-relations/link-relations.xhtml`

do postaci, reprezentacja pojedynczej postaci posiadałaby odnośnik do profesji oraz odnośnik do każdego przedmiotu w ekwipunku postaci.

14.4 Język opisu HAL

O ile styl architektoniczny REST definiuje wykorzystanie HATEOAS, to nie definiuje wprost sposobu definiowania hipermediów. Nie oznacza to, że nie ma dostępnych standardów, które można by wykorzystać. Wartym uwagi jest język opisu HAL (*Hypertext Application Language*).

Należy tutaj pamiętać, że styl REST nie narzuca konkretnego formatu wiadomości. Przykłady w tym rozdziale bazują na formacie JSON głównie ze względu na jego powszechne wykorzystanie. Kolejne przykłady będą dotyczyły JSON HAL [22] jednak nic nie stoi na przeszkodzie, żeby zaaplikować to samo podejście do innych formatów jak zrobiono to np. w XML HAL [25].

Składnia JSON HAL formalizuje nazewnictwo dla elementów reprezentacji zasobu zawierających hiperłącza prowadzące do hipermediów. Jako pierwszy przykład posłuży reprezentacja zwraca przez główny punkt końcowy:

```
HTTP/1.1 200 OK
Content-Type: application/hal+json

{
  "_links": {
    "self": {
      "href": "/api/"
    },
    "rpg:professions": {
      "href": "/api/professions"
    },
    "rpg:characters": {
      "href": "/api/characters"
    },
    "curies": [
      {
        "href": "/api/relations/{rel}",
        "name": "rpg",
        "templated": true
      }
    ]
  },
  "name": "Simple RPG"
}
```

Pierwszą rzeczą, jaka rzuca się w oczy, jest zmiana typu MIME (Multipurpose Internet Mail Extensions) [17] użytego w odpowiedzi. Zamiast do tej pory używanej wartości `application/json` wykorzystano typ, który od razu informuje

klienta, że odpowiedź jest w formacie JSON HAL, czyli `application/json+hal`. Analogicznie w przypadku wykorzystania XML HAL odpowiedź byłaby zapisana w `application/xml+hal`. Kolejna rzecz to zastosowanie zdefiniowanego w standardzie klucza `_links`. Klucz `_links` zawiera w sobie kolejne klucze, których nazwy odpowiadają nazwą relacji, a wartości są pojedynczym linkiem lub tablicą linków. Relacja `self` jest obowiązkowa i powinna wskazywać na zasób, którego dotyczy reprezentacja. Dozwolone właściwości pojedynczego linka również zostały zdefiniowane i można je znaleźć w tabeli 14.1.

Tablica 14.1. Dozwolone własności obiektu Link w JSON HAL [22]

Własność	**Wymagana**	**Wartość**	**Znaczenie**
`href`	wymagana	`Uri/UriTemplate`	Lokalizacja zasobu
`templated`	opcjonalna	`boolean`	Powinien posiadać wartość `true` gdy wartość `href` jest typu `UriTemplate`
`type`	opcjonalna	`string`	Definiuje typ mediów zasobu
`deprecation`	opcjonalna	`Url`	Oznacza, że link nie będzie wspierany w przyszłości i prawdopodobnie zostanie usunięty. Wartość powinna kierować do wyjaśnienia
`name`	opcjonalna	`string`	Może zostać wykorzystany do wyszukiwania wśród linków, które współdzielą ten sam typ relacji
`profile`	opcjonalna	`Uri`	Identyfikacja profilu mogącego dostarczyć dodatkowy informacji o naturze zasobu
`title`	opcjonalna	`string`	Etykieta przyjazna użytkownikowi
`hreflang`	opcjonalna	`string`	Identyfikuje język zasobu

Warto zwrócić uwagę na brak własności `rel` dla linków. Zamiast tego, wszystkie relacje niezdefiniowane w katalogu relacji IANA opatrzone są odpowiednim prefiksem. W przykładzie jest to jedna wartość `rpg` dla wszystkich relacji, jednak w praktyce może być ich więcej. Każdy z prefiksów znajduje swoje odzwierciedlenie jako pozycja przypisana do klucza `curies`. Specyfikacja HAL proponuje wykorzystanie składni CURIE (*Compact URI*) [20] dla zapewniania zwięzłości odnośników do definicji poszczególnych relacji. Wartość `href` każdego linka w tablicy `curies` jest w postaci szablonu URI zawierającego token `rel`, pod który można podstawić nazwę relacji.

Przyjrzyjmy się dalej reprezentacji kolekcji. Potencjalnie kolekcje zasobów mogą być na tyle duże, że nie ma sensu zwracania wszystkich elementów do klienta. W takich sytuacjach stosuje się paginację (stronicowanie), np. poprzez dodanie parametrów żądania `offset` i `limit` albo `page` i `size`. Załóżmy następujące żądanie:

```
GET /api/professions HTTP/1.1
```

Teoretycznie żądanie dotyczy całej kolekcji, jednak nic nie stoi na przeszkodzie, aby serwer zastosował domyślne wartości dla stronicowania. W celu skrócenia przykładowych odpowiedzi przyjęto, że domyślnie jedna strona zawiera dwa elementy. Odpowiedź wygląda następująco:

```
HTTP/1.1 200 OK

{
  "_embedded": {
    "rpg:professions": [
      {
        "_links": {
          "self": {
            "href": "/api/professions/Bard"
          },
          "rpg:characters": {
            "href": "/api/professions/Bard/characters"
          }
        },
        "name": "Bard"
      },
      {
        "_links": {
          "self": {
            "href": "/api/professions/Cleric"
          },
          "rpg:characters": {
            "href": "/api/professions/Cleric/characters"
          }
        },
        "name": "Cleric"
      }
    ]
  },
  "_links": {
    "next": {
      "href": "/api/professions?page=1&size=2"
    },
    "last": {
      "href": "/api/professions?page=1&size=2"
    },
    "self": {
      "href": "/api/professions?page=0&size=2"
    },
    "first": {
```

```
      "href": "/api/professions?page=0&size=2"
    },
    "curies": [
      {
        "href": "/api/relations/{rel}",
        "name": "rpg",
        "templated": true
      }
    ]
  }
}
```

Pierwsza rzecz, która znacznie wyróżnia odpowiedź zgodną z formatem JSON HAL jest fakt, że wszystkie elementy kolekcji zostały zebrane w ramach zdefiniowanego w standardzie klucza `_embedded`. Co więcej, każdy elementy kolekcji posiada klucz `_links` zawierający w sobie listę relacji. Podobnie jak wcześniej, relacja `self` jest obowiązkowa. W przykładzie zdecydowano się na dodanie relacji prowadzącej do listy bohaterów dla każdej profesji. Zgodnie ze standardem HAL, reprezentacja zasobów w kolekcji nie musi być pełna i zależy od tego, co jest istotne z punktu widzenia projektowania konkretnego API. Należy pamiętać, że obiekty zawarte w ramach `_embedded` to też relacje i wszystkie niestandardowe powinny być opatrzone odpowiednim prefiksem.

Następnie należy skupić się na relacjach zdefiniowanych dla reprezentacji zwróconej kolekcji. Oczywiście znajduje się tutaj obowiązkowa relacja `self` ale zostały dodane także relacje odpowiedzialne za stronicowanie (`first`, `last`, `next`, `previous`, wszystkie zdefiniowane w katalogu IANA). Dzięki takiemu zabiegowi klient nie musi widzieć a priori, ile jest elementów w kolekcji oraz jak nazywają się parametry żądania odpowiedzialne za stronicowanie.

Jako ostatni przykład rozważymy reprezentacje pojedynczego bohatera. Załóżmy następujące żądanie:

```
GET /api/characters/Calvian HTTP/1.1
```

W tym przypadku odpowiedź będzie miała postać:

```
HTTP/1.1 200 OK

{
  "_embedded": {
    "rpg:items": [
      {
        "_links": {
          "self": {
            "href": "/api/characters/Calvian/items/Lute"
          }
        },
        "name": "Lute"
```

```
      },
      {
        "_links": {
          "self": {
            "href": "/api/characters/Calvian/items/Rainbow%20Hat"
          }
        },
        "name": "Rainbow Hat"
      }
    ]
  },
  "_links": {
    "self": {
      "href": "/api/characters/Calvian"
    },
    "rpg:portrait": {
      "href": "/api/characters/Calvian/portrait"
    },
    "rpg:profession": {
      "href": "/api/professions/Bard"
    },
    "curies": [
      {
        "href": "/api/relations/{rel}",
        "name": "rpg",
        "templated": true
      }
    ]
  },
  "age": 18,
  "background": "A yong bard with some infernal roots.",
  "charisma": 16,
  "constitution": 12,
  "name": "Calvian",
  "strength": 8
}
```

W powyższym przykładzie można zauważyć obecność zarówno klucza `_links` zawierającego relacje, jak i klucza `_embedded` zawierającego relacje dołączonych obiektów. Projektując API zdecydowano, że skoro postać posiada ekwipunek, to elementy ekwipunku powinny być zawsze zwracane z postacią. Zdecydowano także o niewystawianiu osobnego zasobu reprezentującego przedmioty danej postaci. Oczywiście każdy z elementów ekwipunku zawiera relację `self` z odnośnikiem pod którym dostępna jest reprezentacja zawierająca szczegółowe informacje o danym przedmiocie. Dodatkowo zasoby definiujące profesję i portret bohatera zostały wskazane przez wydzielone osobne relacje.

14.5 Implementacja

W tej części zostały przedstawione fragmenty implementacji wraz z wyjaśnieniem. Pełną implementację można znaleźć w postaci repozytorium kodu opartego na systemie kontroli wersji Git dołączonego do rozdziału[3].

Przykładowa aplikacja została przygotowana z myślą o platformie Jakarta EE będącej następcą szeroko uznanej w wytwarzaniu złożonych aplikacji webowych, platformy Java EE (Java Enterprise Edition) [14]. Jako że Jakarta EE jest zbiorem kilku specyfikacji, to właściwą specyfikacją dla implementacji usług typu REST jest Jakarta RESTful Web Services (JAX-RS) znana wcześniej jako Java API for RESTful Web Services. Specyfikacja JAX-RS posiada kilka niezależnych implementacji (np.: Apache CFX [2], Jersey [10], RestEasy [32] i Restlet [36]), które są używane zarówno w połączeniu z serwerami aplikacji jak i samodzielnie.

Przykładowy projekt został przygotowany tak, aby mógł być zbudowany automatycznie za pomocą narzędzia Apache Maven [15] i uruchomiony na serwerze aplikacji Open Liberty 21 [8]. Oczywiście aplikacja może zostać zbudowana do archiwum `war` i wdrożona na dowolny serwer aplikacji zgodny z Jakarta EE 8, ale wymagałoby to samodzielnego skonfigurowania serwera. Ponieważ projekt bazuje jedynie elementach platformy Jakarta EE 8, jedyną wymaganą zależnością jest:

```
<dependency>
  <groupId>jakarta.platform</groupId>
  <artifactId>jakarta.jakartaee-api</artifactId>
  <version>8.0.0</version>
  <scope>provided</scope>
</dependency>
```

Implementacje usług opartych na specyfikacji JAX-RS najlepiej rozpocząć od zdefiniowania globalnego prefiksu dla wszystkich punktów końcowych. W przykładzie zastosowano wartość `api`. Realizuje się to poprzez zastosowanie adnotacji `@ApplicationPath` na klasie dziedziczącej po klasie `Application`:

```
package pl.edu.pg.eti.kask.rpg.controller;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/api")
public class Config extends Application {
}
```

Kolejnym krokiem jest zdefiniowanie pomocniczych klas, które zostaną wykorzystane podczas przetwarzania żądania lub budowania odpowiedzi przez aplikacje. Na pewno potrzebna jest wygodna reprezentacja dla linków zgodnych ze

[3] `https://git.pg.edu.pl/p650304/jakartaee-discoverable-rest`

specyfikacją HAL. Co prawda JAX-RS zawiera definicję klasy `Link`, lecz nie spełnia ona wymagań zdefiniowanych w standardzie. Proponowana implementacja (definicja metod `get` i `set`, konstruktorów, budowniczych i szeroko rozumianego kodu wzorcowego pominięta dla uproszczenia):

```
package pl.edu.pg.eti.kask.rpg.controller.dto;

import java.net.URI;
import java.net.URL;

public class Link {

    private String href;

    private Boolean templated;

    private String type;

    private URL deprecation;

    private String name;

    private URI profile;

    private String title;

    private String hreflang;

}
```

Zastosowanie typu `String` dla własności `href` wynika z faktu, że specyfikacja przewiduje tutaj wartość `Uri` lub `UriTemplate` jednak w standardowych bibliotekach nie ma takich typów będących w hierarchii dziedziczenia.

Kolejną przydatną klasę będzie reprezentacja definicji stronicowania (kod wzorcowy pominięty dla uproszczenia):

```
package pl.edu.pg.eti.kask.rpg.controller.param;

import javax.ws.rs.DefaultValue;
import javax.ws.rs.QueryParam;

public class Page {

    @QueryParam("page")
    @DefaultValue("0")
    private int page;
```

```
    @QueryParam("size")
    @DefaultValue("5")
    private int size;

}
```

Zastosowanie adnotacji `@QueryParam` i `@DefaultValue` pozwoli na wygodne odwzorowanie parametrów żądania na parametry metod zdefiniowanych w kontrolerach.

Ponieważ zbiór typów multimediów zdefiniowanych w standardzie JAX-RS w klasie `MediaType` nie pokrywa wszystkich potrzebnych typów w projekcie, zwłaszcza typu `application/json+hal`, proponuje się samodzielne zdefiniowanie tych typów w analogiczny sposób:

```
package pl.edu.pg.eti.kask.rpg.controller;

import javax.ws.rs.core.MediaType;

public class MediaTypes {

    public final static String IMAGE_PNG
            = "image/png";

    public final static MediaType IMAGE_PMG_TYPE
            = new MediaType("image", "png");

    public final static String APPLICATION_HAL_JSON
            = "application/hal+json";

    public final static MediaType APPLICATION_HAL_JSON_TYPE
            = new MediaType("application", "hal+json");
}
```

Kolejnym etapem jest przygotowanie implementacji kontrolerów. Powszechna praktyka przyjmuje podział aplikacji na przynajmniej trzy elementy: kontroler obsługujący żądania od klienta, serwis wykonujący operacje biznesowe i repozytorium realizujące dostęp do danych (np. baza danych). Szczegóły implementacji serwisów i repozytoriów są poza zakresem tego rozdziału i nie będą omawiane. Implementacja kontrolera wygląda następująco (pominięto metody obsługujące pozostałe żądania):

```
package pl.edu.pg.eti.kask.rpg.character.controller;

import pl.edu.pg.eti.kask.rpg.character.dto.GetCharactersResponse;
import pl.edu.pg.eti.kask.rpg.controller.MediaTypes;
import pl.edu.pg.eti.kask.rpg.controller.PageRequest;
import pl.edu.pg.eti.kask.rpg.controller.param.Page;
```

```
import pl.edu.pg.eti.kask.rpg.character.service.CharacterService;

import javax.inject.Inject;
import javax.ws.rs.BeanParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;


@Path("")
public class CharacterController {

    private CharacterService service;

    @Inject
    public void setService(CharacterService service) {
        this.service = service;
    }

    @Path("characters")
    @GET
    @Produces(MediaTypes.APPLICATION_HAL_JSON)
    public Response getCharacters(@BeanParam Page page,
                                  @Context UriInfo uriInfo) {

        PageRequest pageRequest = PageRequest
                .ofPage(page)
                .count((int) service.countAll()
                        / page.getSize())
                .build();

        return Response
                .ok(GetCharactersResponse
                        .entityToDtoMapper(uriInfo)
                        .apply(service.findAll(
                                pageRequest.getOffset(),
                                pageRequest.getSize()),
                                pageRequest))
                .build();
    }

}
```

Wykorzystanie JAX-RS pozwala na zdefiniowanie metod, które będą automatycznie wywołane do obsługi konkretnych żądań. Powiązanie metody z adresem docelowym żądania realizowane jest przez adnotacje `@Path` nałożone zarówno na klasę, jak i metody. Wynikowa ścieżka jest złączeniem wartości z adnotacji `@ApplicationPath`, `@Path` na klasie i `@Path` na metodzie. Wynikowy adres w powyższym przykładzie ma wartość `/api/characters/`.

Obiekt klasy `CharacterService` realizuje operacje logiki biznesowej i zostaje wstrzyknięty za pomocą mechanizmu CDI (*Context and Dependency Injection*). W przykładzie wstrzyknięcie następuje przez metodę `set` zamiast przez konstruktor, ponieważ nie wszystkie implementacje JAX-RS poprawnie wspierają wstrzykiwanie obiektów zarządzanych przez kontener CDI za pomocą konstruktora.

Metoda zwracająca reprezentację bohaterów została adnotowana trzema adnotacjami: opisaną wcześniej `@Path`, `@GET` ustalającą metodę HTTP (dostępne są także `@POST`, `@PUT`, `@DELETE`, `@PATCH` i inne wynikające ze specyfikacji protokołu) oraz `@Produces` informującą, w jakim formacie powinno zostać zapisane ciało odpowiedzi. Implementacja JAX-RS podczas przygotowania odpowiedzi automatycznie wykorzysta odpowiedni mechanizm do zapisania ciała odpowiedz. Dla typu `application/json` wykorzysta implementację standardu JSON-B (JSON Binding) [13]. Nie ma tutaj znaczenia, że w implementacji wykorzystany został typ `application/json+hal`. Domyślny mechanizm dla formatu JSON zostanie w tym przypadku użyty automatycznie bez dodatkowych zmian w konfiguracji.

Metoda `getCharacters` posiada dwa parametry. Pierwszy parametr, adnotowany `@BeanParam`, definiuje zbiór parametrów żądania zebranych w pomocniczym obiekcie. Alternatywnie można zdefiniować każdy parametr żądania jako osobny parametr metody adnotowany `@QueryParam`. Adnotacja `@Context` służy m.in. do wstrzyknięcia obiektu `UriInfo` posiadającego informacje o oryginalnym URI żądania.

Wartością zwracaną przez metodę jest obiekt klasy `Response`. Pozwala on na ustawienie kodu odpowiedzi, dodatkowych nagłówków jeśli to potrzebne oraz obiektu, który zostanie przekształcony do formatu zdefiniowanego w adnotacji `@Produces`. Należy tutaj zwrócić uwagę na fakt, że o ile metoda biznesowa `CharacterService#findAll` zwraca obiekt typu `List<Character>`, to nie jest to wartość bezpośrednio przekazywana do obiektu `Response`. Obiekt z warstwy biznesowej zostaje tutaj odwzorowany na obiekt DTO (*Data Transfer Object*) `GetCharactersResponse`, który zawiera wszystkie niezbędne własności wymagane do poprawnego zapisu reprezentacji bohaterów w formacie JSON HAL.

Poniżej przedstawiono klasę `GetCharactersResponse` (kod wzorcowy pominięty dla uproszczenia), której obiekty realizują reprezentację kolekcji bohaterów na potrzeby zbudowania odpowiedzi na żądanie HTTP.

```
package pl.edu.pg.eti.kask.rpg.character.dto;

import pl.edu.pg.eti.kask.rpg.character.entity.Character;
import pl.edu.pg.eti.kask.rpg.controller.PageRequest;
```

```
import javax.json.bind.annotation.JsonbProperty;
import javax.ws.rs.core.UriInfo;
import java.util.Collection;
import java.util.Map;
import java.util.function.BiFunction;
import java.util.function.Function;

public class GetCharactersResponse {

    public static class EmbeddedCharacter {

        private String name;

        @JsonbProperty("_links")
        private Map<String, Object> links;

        public static Function<Character,
                EmbeddedCharacter>
        entityToDtoMapper(UriInfo uriInfo) {
            //...
        }
    }

    @JsonbProperty("_embedded")
    private Map<String, Object> embeddeds;

    @JsonbProperty("_links")
    private Map<String, Object> links;

    public static BiFunction<Collection<Character>,
            PageRequest,
            GetCharactersResponse>
    entityToDtoMapper(UriInfo uriInfo) {
        //...
    }

}
```

Aby zapewnić zgodność odpowiedzi ze standardem HAL, zdefiniowano dodatkowe pola `embeddeds` i `links`, które zostaną przetłumaczone na odpowiednie właściwości w formacie JSON. Zastosowanie typu `Map<String, Object>` wynika z faktu, że zgodnie ze specyfikacją HAL, wartością może być zarówno jeden link, jak i tablica linków. Implementacja metod `entityToDtoMapper` została pominięta, ponieważ służy jedynie odwzorowaniu jednego obiektu w drugi i można ją znaleźć w załączonej implementacji.

Jako istotny element poprawnego przygotowania odpowiedzi, zamieszczono poniżej sposób poprawnego zbudowania obiektu `Link` tak, aby zawierał w sobie protokół, nazwę hosta i port na jaki klient pierwotnie wysłał żądanie. Ponieważ aplikacja może być wystawiona pod różnymi adresami jednocześnie, istotne jest, aby poprawnie skorzystać z obiektu `UriInfo` zawierającego informacje o oryginalnym żądaniu.

```
Link self = Link.builder()
        .href(uriInfo.getBaseUriBuilder()
                .path(CharacterController.class,
                        "getCharacter")
                .build(character.getName())
                .toString())
        .build();

}
```

14.6 Podsumowanie

Niniejszy rozdział przedstawił dyskusję, w jaki sposób może być zrealizowana implementacja wykrywalnych usług sieciowych opartych na stylu architektonicznym REST. W szczególności zostały tutaj przedstawione sposoby poprawnego budowania hierarchicznego API zgodnie z modelem dojrzałości Richardsona jak i techniki opisywania treści zwracanej przez usługi sieciowe zgodnie ograniczeniem HATEOAS. Co prawda specyfikacja REST nie nakłada żadnego konkretnego formatu opisu, jednak przedstawiona tutaj specyfikacja HAL jest dobrym źródłem inspiracji podczas budowania API opartego na stylu architektonicznym REST.

Do rozdziału dołączony został praktyczny przykład zrealizowany jako aplikacja zgodne ze specyfikacją Jakarta EE a w szczególności JAX-RS. Implementacja jest publicznie dostępna w postaci repozytorium kodu opartego na systemie kontroli wersji Git[4].

Bibliografia

1. Anders, D., Marinka, C., Markus, M., Mirjam, E., Michael, H., Jaakko, S.: Role-playing games: The state of knowledge [panel abstracts]. In: DiGRA '09 - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory. Brunel University (September 2009), `http://www.digra.org/wp-content/uploads/digital-library/09287.23528.pdf`.
2. Apache Software Foundation: Apache CXF Project. `https://cxf.apache.org/`, dostęp: 2021-10-24.

[4] `https://git.pg.edu.pl/p650304/jakartaee-discoverable-rest`

3. Ben-Kiki, O., Evans, C., döt Net, I.: YAML Ain't Markup Language (YAML™) Version 1.2. https://yaml.org/spec/1.2/spec.html (2009), dostęp: 2021-06-04.
4. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture, W3C Working Group Note. https://www.w3.org/TR/ws-arch/ (2004), dostęp: 2021-06-04.
5. Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259 (Dec 2017). https://doi.org/10.17487/RFC8259, https://rfc-editor.org/rfc/rfc8259.txt.
6. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation. https://www.w3.org/TR/xml/ (2008), dostęp: 2021-06-04.
7. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Recommendation. https://www.w3.org/TR/wsdl20/ (2007), dostęp: 2021-06-04.
8. Corp., I.: https://openliberty.io/docs/21.0.0.5/overview.html (2021), dostęp: 2021-06-04.
9. Dusseault, L.M., Snell, J.M.: PATCH Method for HTTP. RFC 5789 (Mar 2010). https://doi.org/10.17487/RFC5789, https://rfc-editor.org/rfc/rfc5789.txt.
10. Eclipse Foundation: Jersey Project. https://eclipse-ee4j.github.io/jersey/, dostęp: 2021-10-24.
11. Fielding, R.T., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231 (Jun 2014). https://doi.org/10.17487/RFC7231, https://rfc-editor.org/rfc/rfc7231.txt.
12. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California (2000).
13. Foundation, E.: http://json-b.net/docs/user-guide.html (2019), dostęp: 2021-06-04.
14. Foundation, E.: Jakarta EE Platform, Version 8. https://jakarta.ee/specifications/platform/8/platform-spec-8.html (2019), dostęp: 2021-06-04.
15. Foundation, T.A.S.: https://maven.apache.org/guides/index.html (2021), dostęp: 2021-06-04.
16. Fowler, M.: Richardson Maturity Model: steps toward the glory of REST. https://martinfowler.com/articles/richardsonMaturityModel.html (2010), dostęp: 2021-06-04.
17. Freed, N., Borenstein, D.N.S.: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046 (Nov 1996). https://doi.org/10.17487/RFC2046, https://rfc-editor.org/rfc/rfc2046.txt.
18. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J., Nielsen, H.F., Karmarkar, A., Lafon, Y.: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation. http://www.w3.org/TR/soap12/ (2007), dostęp: 2021-06-04.

19. JetBrains s.r.o.: The State of Developer Ecosystem 2021. https://www.jetbrains.com/lp/devecosystem-2021/ (2020), dostęp: 2021-10-24.
20. Jr., T.A., Etemad, E.J., Rivoal, F.: CURIE Syntax 1.0, A syntax for expressing Compact URIs, W3C Working Group Note. https://www.w3.org/TR/curie/ (2010), dostęp: 2021-06-04.
21. Jr., T.A., Etemad, E.J., Rivoal, F.: CSS Snapshot 2020 W3C Working Group Note. https://www.w3.org/TR/css-2020/ (2020), dostęp: 2021-06-04.
22. Kelly, M.: JSON Hypertext Application Language. Internet-Draft draft-kelly-json-hal-08, Internet Engineering Task Force (May 2016), https://datatracker.ietf.org/doc/html/draft-kelly-json-hal-08, work in Progress.
23. Klensin, D.J.C.: Simple Mail Transfer Protocol. RFC 5321 (Oct 2008). https://doi.org/10.17487/RFC5321, https://rfc-editor.org/rfc/rfc5321.txt.
24. McCahill, M.P., Johnson, D., Alberti, B., Torrey, D., Anklesaria, F., Linder, P.: The Internet Gopher Protocol (a distributed document search and retrieval protocol). RFC 1436 (Mar 1993). https://doi.org/10.17487/RFC1436, https://rfc-editor.org/rfc/rfc1436.txt.
25. Michaud, J.: XML Hypertext Application Language. Internet-Draft draft-michaud-xml-hal-02, Internet Engineering Task Force (Feb 2018), https://datatracker.ietf.org/doc/html/draft-michaud-xml-hal-02, work in Progress.
26. Morley, M.: JSON-RPC 2.0 Specification. https://www.jsonrpc.org/specification (2013), dostęp: 2021-06-04.
27. Nelson, B.J.: Remote Procedure Call. Ph.D. thesis, Carnegie Mellon University, USA (1981), aAI8204168.
28. Nottingham, M.: Web Linking. RFC 8288 (Oct 2017). https://doi.org/10.17487/RFC8288, https://rfc-editor.org/rfc/rfc8288.txt.
29. Nottingham, M., Sayre, R.: The Atom Syndication Format. RFC 4287 (Dec 2005). https://doi.org/10.17487/RFC4287, https://rfc-editor.org/rfc/rfc4287.txt.
30. Postel, J., Reynolds, J.: File Transfer Protocol. RFC 959 (Oct 1985). https://doi.org/10.17487/RFC0959, https://rfc-editor.org/rfc/rfc959.txt.
31. Postman, Inc.: 2020 State of The API Report. https://www.postman.com/state-of-api/ (2020), dostęp: 2021-10-24.
32. Red Hat: RESTEasy Project. https://resteasy.github.io/, dostęp: 2021-10-24.
33. Richardson, L., Ruby, S.: Restful Web Services. O'Reilly, first edn. (2007).
34. Schiettecatte, F., Morris, H., Kahle, B., Fullton, J., Gamiel, K., Goldman, J., Kunze, J.A., Pierre, M.S.: WAIS over Z39.50-1988. RFC 1625 (Jun 1994). https://doi.org/10.17487/RFC1625, https://rfc-editor.org/rfc/rfc1625.txt.
35. SmartBear Software: The state of api report 2020. https://smartbear.com/resources/ebooks/the-state-of-api-2020-report/ (2020), dostęp: 2021-10-24.

36. Talend: Restlet Project. https://restlet.talend.com/, dostęp: 2021-10-24.
37. Ullman, C., Dykes, L.: Beginning Ajax. Wrox Press Ltd., GBR (2007).
38. Winer, D.: XML-RPC Specification. http://xmlrpc.com/spec.md (1999), dostęp: 2021-06-04.

15. Wzajemne wykluczanie w programowaniu współbieżnym

Mariusz Matuszek

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
Mariusz.Matuszek@eti.pg.edu.pl

Streszczenie
W rozdziale opisano wzajemne wykluczanie wątków w programach współbieżnych. Przedstawiono zarówno podejście proceduralne (semafory), jak i obiektowe (monitory). Omówiono sposoby działania obu mechanizmów synchronizacji oraz różnice pomiędzy nimi. Sposoby użycia omawianych mechanizmów zostały zilustrowane wzorcami: wzajemnego wykluczania oraz producent–konsument.

Słowa kluczowe: programowanie współbieżne, producent–konsument, wzajemne wykluczanie, architektury wielordzeniowe

15.1 Potrzeba synchronizacji i wzajemnego wykluczania

Opisana we wstępie do rozdziału 16 ewolucja architektury procesorów i akceleratorów prowadzi do logicznej konkluzji, że pełne wykorzystanie mocy obliczeniowej współczesnego sprzętu wymaga biegłości w programowaniu współbieżnym oraz zrozumienia zagrożeń (m.in. dla spójności danych) wynikających z złożonych zależności czasowych pomiędzy poszczególnymi elementami aplikacji współbieżnej.

Podstawowym zagrożeniem dla spójności danych współdzielonych (stanu globalnego) w aplikacji współbieżnej jest zjawisko wyścigu, którego geneza również została omówiona w wyżej wymienionym rozdziale. Aby uniknąć tego zagrożenia potrzebna jest koordynacja dostępu wątków aplikacji współbieżnej do współdzielonych struktur danych. Koordynację uzyskuje się poprzez czasowe zawieszenie współbieżności wykonania kodu w zakresie dostępu do danych współdzielonych, przy pomocy metod wzajemnego wykluczania opisanych w dalszej części rozdziału. Fragment kodu, w którym występuje spowodowane współbieżnością zagrożenie naruszenia spójności danych nazywany jest *sekcją krytyczną*, kod koordynujący wejście wątku (lub procesu) do sekcji krytycznej nazywany jest *protokołem wejścia*, natomiast kod informujący o zakończeniu działania wewnątrz sekcji krytycznej (i potencjalnie udostępniający dostęp do tej sekcji innym wątkom oczekującym na wejście) nazywany jest *protokołem wyjścia*.

15.2 Ewolucja metod wzajemnego wykluczania

Historycznie, pierwszym mechanizmem umożliwiającym strukturalną realizację wzajemnego wykluczania w środowiskach pozwalających na współbieżne wykonywanie kodu były semafory, zaproponowane przez Dijkstrę w 1965 roku [1, 2]. Propozycja ta była zbieżna z wprowadzanym w tamtych latach nurtem *programowania strukturalnego*. Autorem tego określenia i głównym orędownikiem jego zasad był także E.W. Dijkstra.

Wcześniejsze podejście do problemu wzajemnego wykluczania wykorzystywało metody sprzętowe, takie jak blokowanie przerwań (co uniemożliwia wywłaszczenie procesu), jednak metoda ta powoduje szereg niepożądanych efektów ubocznych, jak np. dryft zegara czy zwiększony potencjał całkowitego zawieszenia działania komputera. Inną metodą było aktywne oczekiwanie (ang.: *busy-wait*) wykorzystujące atomowe instrukcje procesora typu testuj-i-ustaw (ang.: *test-and-set*), co jednak niepotrzebnie obciąża procesor.[1]

W połowie lat 70. XX wieku zbiegły się dwie okoliczności. Pierwszą było upowszechnienie programowania współbieżnego, związane z rozwojem systemów operacyjnych wspierających wieloprocesowość. Drugą okolicznością było pojawienie się pewnego rodzaju mody na programowanie obiektowe. Naturalną konsekwencją takiego stanu rzeczy były próby wyrażenia metod synchronizacji i wzajemnego wykluczania w sposób obiektowy. Podejściem, które się przyjęło, są *monitory*, zaproponowane w 1973 roku przez Hansena [3]. Rok później koncepcja została udoskonalona przez Hoare [4].

15.2.1 Semafory

Zaproponowane przez Dijkstrę semafory można opisać jako pewien abstrakcyjny typ danych *S*, na którym zdefiniowane są dwie operacje *P(S)* i *V(S)*. Operacje te mają specjalne właściwości (opisane poniżej), wpływające na wykonanie wątku lub procesu, który je wywołał. W praktyce typem danych stosowanym podczas implementacji semaforów jest liczba całkowita, jednakże programista nie posiada bezpośredniego dostępu do wartości tej liczby. Nazwy operacji P i V, zaproponowane przez Dijkstrę, pochodzą od pierwszych liter wyrazów oznaczających w języku holenderskim przejście-z-opuszczeniem semafora i wyjście-z-podniesieniem semafora. Osobom nie władającym tym językiem nazwy te mówią niewiele, stąd w praktyce programowania spotyka się ich odpowiedniki *sem_wait()* i *sem_post()*. W literaturze teoretycznej nadal można spotkać pierwotnie zaproponowane nazwy i niekiedy potrzebna jest chwila namysłu, aby połączyć je z ich funkcją. Jedną z technik ułatwiających zapamiętanie sposobu działania tych operacji jest uporządkowanie ich w kolejności alfabetycznej, zgodnej z kolejnością ich używania w protokołach wejścia do sekcji krytycznej i wyjścia z sekcji krytycznej.

Od strony funkcjonalnej opis budowy i działania semafora można przedstawić w następujących punktach:

[1] Warto zauważyć, że metody te nadal są używane w swoich niszach, na przykład w systemach wbudowanych.

– ukryta zmienna całkowita S z dwiema wyróżnionymi operacjami,
– brak bezpośredniego dostępu i możliwości testowania wartości tej zmiennej z poziomu kodu aplikacji,
– operacja *P(S)* (*sem_wait(S)*), zwana także opuszczeniem semafora, której działanie opisuje pseudokod:

```
if (S > 0)
   S = S - 1;        // zmniejsz wartość zmiennej wewnętrznej
else
   p_suspend();      // wstrzymaj działanie wołającego procesu
```

– operacja *V(S)* (*sem_post(S)*), zwana także podniesieniem semafora, której działanie opisuje pseudokod:

```
if (p_suspended())  // są procesy wstrzymane na semaforze
   p_activate();    // wznów jeden z nich
else
   S = S + 1;       // zwiększ wartość zmiennej wewnętrznej
```

Użyte w pseudokodzie funkcje *p_suspend(), p_activate()* oraz test *p_suspended()* powodują, odpowiednio:

p_suspend() – wstrzymanie bieżącego procesu i przeniesienie go do kolejki procesów wstrzymanych na danym semaforze,
p_activate() – wznowienie *któregoś* z procesów wstrzymanych uprzednio na tym semaforze,
p_suspended() – zwrócenie wartości logicznej, informującej czy na tym semaforze są wstrzymane procesy.

Należy podkreślić, że operacje na semaforze oraz powiązane z nimi struktury danych (kolejka procesów wstrzymanych, oczekujących na podniesienie semafora) i działania (wstrzymywanie i wznawianie procesów) są implementowane po stronie systemu operacyjnego i współpracują z zarządcą procesów (ang.: *scheduler*, moduł systemu operacyjnego odpowiedzialny za przydzielanie procesom/wątkom czasu procesora).

Opisany powyżej sposób działania dotyczy *semafora ogólnego* (określanego czasem nazwą *semafor zliczający*), którego wewnętrzna zmienna przyjmuje nieujemne wartości całkowite. Poniższy fragment kodu w języku C ilustruje funkcję udostępniającą semafor ogólny i funkcje działające na nim:

```
#include <semaphore.h>
#define N 10      // wartość początkowa semafora

sem_t semafor;    // reprezentacja semafora w programie
sem_init(&semafor, 0, N); // utworzenie semafora

// kod współbieżny
sem_wait(&semafor);
...
sem_post(&semafor);
```

Semafor binarny Szczególnym przypadkiem semafora ogólnego jest *semafor binarny*. Semafor binarny może przyjmować jedynie wartość 0 lub 1, a do jego implementacji wystarczy zmienna logiczna – pojedynczy bit. Semafor binarny B udostępnia takie same operacje (P i V) jak semafor ogólny, jednak ich sposób działania się różni, co ilustruje poniższy pseudokod:

– operacja P(B) (sem_wait(B)):

```
if (B == 1)          // jeśli semafor podniesiony
   B = 0;            // .. to go opuść
else
   p_suspend();      // wstrzymaj proces wołający
```

– operacja V(B) (sem_post(B)):

```
if (p_suspended())   // są procesy wstrzymane
   p_activate();     // wznów jeden z nich
else
   B = 1;            // podnieś semafor
```

Należy zauważyć, że w przypadku semafora binarnego operacje P i V nie są symetryczne. Sekwencja operacji V jest dozwolona, ale ich wynik będzie taki sam jak wynik pojedynczej operacji V. Ilustrując to bardziej opisowo: możliwe i dozwolone jest podnoszenie już podniesionego semafora binarnego, ale nie zmienia to stanu semafora.

Kolejną obserwacją, której należy dokonać, jest fakt, że semafory nie znają pojęcia właściciela. Oznacza to, że w przypadku kodu współbieżnego nie ma znaczenia, czy operacje P i V zostały wykonane przez ten sam proces (albo wątek), czy przez różne procesy. W sytuacji, gdy programista poprawnie implementuje logikę programu nie ma to znaczenia, natomiast jeżeli z powodu błędu w kodzie nastąpi błędne (przedwczesne) wykonanie operacji V, to dalsze działanie aplikacji ma charakter przypadkowy, co z kolei prowadzi do trudnych do zdiagnozowania błędów. Z tego powodu zaproponowano [5] mechanizm pod wieloma względami analogiczny do semafora binarnego i nazwano go *mutexem*.

Mutex Koncepcyjnie mutex (MX) jest bardzo zbliżony do semafora binarnego. Tak jak semafor binarny, mutex może posiadać dwa stany: otwarty i zamknięty. Mutex też udostępnia dwie operacje: *lock()* i *unlock()*. Podobnie jak w przypadku operacji P(B) semafora binarnego, operacja MX.lock() jest blokująca i powoduje wstrzymanie wołającego wątku, ale to zatrzymanie jest warunkowe, o czym niżej. Podobieństwo to bywa mylące, istnieją bowiem zasadnicze różnice:

– w przypadku mutexów wprowadzono koncepcję *właściciela*. Właścicielem mutexu jest wątek, któremu udało się wykonać (zakończyć) operację lock();
– wątek będący właścicielem mutexu może ponownie wielokrotnie wywoływać operację lock() na posiadanym mutexie i nie powoduje to jego wstrzymania. Pozwala to na pisanie kodu w sposób rekurencyjny oraz na wzajemne

wywoływanie funkcji wykorzystujących ten sam mutex – należy jedynie dopilnować, aby wątek będący aktualnym właścicielem mutexu wykonał taką samą liczbę operacji unlock(), jak liczba operacji lock();
– wykonanie operacji unlock() przez wątek *nie będący* właścicielem mutexu nie spowoduje zmiany jego stanu (ale może zwrócić błąd do wołającego kodu).

Poniższy pseudokod ilustruje algorytm działania operacji lock i unlock:

– *MX.lock()*

```
switch get_mx_state() {

  UNLOCKED:
     set_mx_owner(ID_caller);
     mx_lock_count = 1;
     set_mx_state(LOCKED);

  LOCKED:
     if (caller_is_a_lock_owner(ID_caller))
         mx_lock_count++;
     else
         t_suspend();
}
```

– *MX.unlock()*

```
switch get_mx_state() {

  LOCKED:
     if (caller_is_a_lock_owner(ID_caller))
         mx_lock_count--;
         if (mx_lock_count == 0)
             set_mx_owner(NULL);
             set_mx_state(UNLOCKED);
     else
         // do nothing
         // or raise error

  UNLOCKED:
     // do nothing
}
```

15.2.2 Monitor

Jak już wspomniano na wstępie, *monitor* jest konstrukcją umożliwiającą przedstawienie i implementację operacji wzajemnego wykluczania i synchronizacji w sposób obiektowy. Tak więc: monitor jest jednolitą konstrukcją programową – najczęściej obiektem – posiadającą własne, wyróżnione (prywatne) zmienne

oraz procedury i funkcje działające na tych zmiennych. Spośród tych funkcji (metod) niektóre są udostępniane na zewnątrz (publiczne), co umożliwia dostęp do zmiennych ukrytych w sposób kontrolowany. Liczba funkcji monitora udostępnianych na zewnątrz jest nieograniczona, ale podstawowym założeniem działania monitora jest możliwość wykonywania tylko jednego wywołania którejkolwiek spośród jego funkcji w danym momencie. Mniej formalnie opisuje się ten wymóg określeniem "tylko jeden wątek/proces może przebywać w monitorze". Oznacza to, że jeżeli jakikolwiek wątek aktualnie wykonuje którąkolwiek z metod monitora, inne próby wywołania którejkolwiek z metod tego monitora będą blokujące.

wait & signal Oprócz podstawowego założenia, dotyczącego sposobu wykonywania metod, wprowadzono specjalny typ danych *condition*, oraz zdefiniowano trzy operacje działające na zmiennych tego typu, są to:

wait(c) – wykonanie tej operacji powoduje:
- wstrzymanie wątku/procesu wykonującego tę operację,
- wstawienie tego wątku na koniec kolejki FIFO związanej z zmienną *c*,
- zwolnienie monitora (odblokowanie możliwości wykonania jego funkcji);

signal(c) – wykonanie tej operacji powoduje wznowienie *pierwszego*[2] procesu oczekującego w kolejce procesów wstrzymanych związanej z zmienną *c*. Jeżeli w chwili wywołania tej operacji kolejka jest pusta, to wykonanie tej operacji również ma efekt pusty.
!! Jeżeli wywołanie operacji *signal* nie jest ostatnią instrukcją procedury monitora, to proces wykonujący tę operację zwolni monitor i zostanie wstrzymany aż do chwili, gdy wznowiony przezeń proces zwolni monitor. Procesy wstrzymywane w ten sposób trafiają do kolejki procesów wstrzymanych o organizacji LIFO;

empty(c) – wykonanie tej operacji zwraca wartość logiczną zależną od tego, czy kolejka procesów wstrzymanych powiązana z zmienną *c* jest pusta (zwracaną wartością jest *true*), czy też są w niej wstrzymane procesy (zwracaną wartością jest *false*).

Jak wynika z opisu działania, integralną częścią konstrukcji i funkcjonowania monitora są również kolejki procesów/wątków skojarzone z każdą z zmiennych warunkowych. Ponadto należy zwrócić szczególną uwagę na warunkowe funkcjonowanie operacji *signal* w przypadku, gdy nie jest to ostatnia operacja funkcji monitora. Takie zachowanie jest konieczne, aby mógł być spełniony warunek "przebywania w monitorze" tylko jednego wątku. Dla jego spełnienia potrzebne jest podjęcie działania na etapie kompilowania kodu źródłowego, stąd też monitory są konstrukcjami wbudowanymi w języki programowania, choć możliwe jest implementowanie podobnej funkcjonalności w językach niewyposażonych w monitory. Przykład monitora ilustruje poniższy pseudokod:

[2] Jest to istotna różnica w stosunku do semaforów, które gwarantują jedynie wznowienie jednego z oczekujących procesów, bez ich uporządkowania.

```
monitor M
{
   var i:integer;
   var c:condition;

   public procedure P1() {
      ... // some code
      wait(c);
      ... // more code
   }

   public procedure P2() {
      ... // some code
      signal(c);
      ... // special treatment if code here
   }

   // constructor
   procedure M() {
      i = 0;
   }
}
```

Monitor może być wykorzystany do implementacji innych mechanizmów koordynacji i synchronizacji. Poniższy pseudokod ilustruje implementację semafora ogólnego przy pomocy monitora:

```
monitor SEMAFOR
{
   var semafor:integer;
   var kolejka:condition;

   public procedure P() {
      if (semafor == 0) wait(kolejka);
      semafor--;
   }

   public procedure V() {
      semafor++;
      signal(kolejka);
   }

   procedure SEMAFOR() {
      semafor = 0;
   }
}
```

15.3 Realizacja podstawowych wzorców

Ostatnia część rozdziału poświęcona jest zilustrowaniu użycia semaforów i monitorów do realizacji podstawowych wzorców programowania współbieżnego.

15.3.1 Wzajemne wykluczanie przy użyciu semaforów

Realizacja wzajemnego wykluczania dowolnej liczby procesów przy użyciu semafora binarnego jest jednym z najprostszych wzorców programowania współbieżnego. Poniższy pseudokod ilustruje użycie semafora binarnego w takim zastosowaniu:

```
var B:binarysemaphore = 1;
const N = ... ;                // liczba procesów

#concurrently {

   process P(i:1..N) {
      while (true) {
         // sekcja lokalna
         ...
         ...

         P(B);           // protokół wejścia
         ...             // kod sekcji krytycznej
         V(B);           // protokół wyjścia

         ...
         ...
      }
   }

}
```

W niektórych zastosowaniach, zwłaszcza w systemach i środowiskach czasu rzeczywistego przyjmuje się, że w sytuacjach wymagających wzajemnego wykluczania właściwszym od użycia semafora binarnego jest użycie mutexu. Jednym z argumentów na korzyść takiego podejścia jest potencjalnie mniejszy koszt użycia mutexu, co wynika z sposobu implementacji.

15.3.2 Wzajemne wykluczanie przy użyciu monitora

Wzajemne wykluczanie stosowane jest z reguły w celu ochrony jakiegoś zasobu przed współbieżną modyfikacją. Takim zasobem może być jakaś struktura danych, ale równie dobrze może to być jakiś zasób sprzętowy, na przykład terminal. Jeżeli procedurę dokonującą operacji na zasobie zdefiniujemy jako procedurę monitora, to wzajemne wykluczanie otrzymamy w sposób naturalny, wynikający wprost z definicji działania monitora:

```
monitor ZASÓB
{
   var zasób:typ;

   public procedure DOSTĘP() {
      // kod działający na zasobie
   }

   procedure ZASÓB() {}
}
```

W ten niewątpliwie elegancki sposób uzyskujemy ochronę zasobu przed dostępem współbieżnym. Jeżeli jakiś kod współbieżny dokonywałby próby jednoczesnego wywołania operacji ZASÓB.DOSTĘP(), to tylko jeden z wątków/procesów uzyska dostęp, zaś pozostałe zostaną wstrzymane.

15.3.3 Producent-konsument przy użyciu semaforów

Wzorzec producenta i konsumenta (lub szerzej, producentów i konsumentów) jest dosyć często spotykany. Podstawową strukturą danych w tym wzorcu jest *bufor* (zazwyczaj wymaganym wzorcem dostępu do bufora jest FIFO) oraz dwie operacje: *wstaw* i *pobierz*.

Jeden producent i jeden konsument Jeżeli aplikacja nie wymaga współpracy wielu producentów i konsumentów, zastosować można wzorzec uproszczony, pokazany poniżej. Taka sytuacja może mieć miejsce np. przy łączeniu dwóch elementów potoku w przetwarzaniu potokowym pod warunkiem, że oba łączone elementy są jednowątkowe:

```
const N:integer = ... ;              // rozmiar bufora
var bufor:array[0..N-1] of porcja;
var MIEJSCE:semaphore = N;
var  PORCJE:semaphore = 0;

process Producent() {

   var p:porcja;
   var zapis:integer = 0;

   while (true) {
      produkuj(var p:porcja);
      P(MIEJSCE);
         bufor[zapis] = p;
         zapis = (zapis + 1) modulo N;
      V(PORCJE);
   }
```

```
}

process Konsument() {

   var p:porcja;
   var odczyt:integer = 0;

   while (true) {
      P(PORCJE);
         p = bufor[odczyt];
         odczyt = (odczyt + 1) modulo N;
      V(MIEJSCE);
      konsumuj(p);
   }
}
```

Przedstawiony wzorzec pozwala na jednoczesny dostęp producenta i konsumenta do bufora, a operacje na semaforach zapewniają, że zmienne *zapis* i *odczyt* nigdy nie mają tej samej wartości (po wstawieniu pierwszej porcji danych do bufora).

Wielu producentów i wielu konsumentów W przypadku, gdy producentów lub konsumentów będzie więcej, powyższy wzorzec nie będzie wystarczający. Aby go dostosować do takiej sytuacji należy wprowadzić ochronę zmiennych *zapis* i *odczyt* przez wzajemne wykluczanie między sobą producentów i konsumentów, zgodnie z wzorcem:

```
  const N:integer = ...    // rozmiar bufora
  const P:integer = ...    // liczba producentów
  const K:integer = ...    // liczba konsumentów

  var MIEJSCE:semaphore = N;
  var  PORCJE:semaphore = 0;
  var   ZAPIS:binarysemaphore = 1;
  var  ODCZYT:binarysemaphore = 1;
  var   zapis:integer = 0;
  var  odczyt:integer = 0;
  var   bufor:array[0..N-1] of porcja;

#concurrently{

  process Producent(i:1..P) {
     var p:porcja;

     while(true) {
        produkuj(p);
        P(MIEJSCE);
```

```
          P(ZAPIS);
            bufor[zapis] = p;
            zapis = (zapis +1) modulo N;
          V(ZAPIS);
        V(PORCJE);
     }
  }

  process Konsument(i:1..K) {
     var p:porcja;

     while(true) {
        P(PORCJE);
          P(ODCZYT);
            p = bufor[odczyt];
            odczyt = (odczyt + 1) modulo N;
          V(ODCZYT);
        V(MIEJSCE);
        konsumuj(p);
     }
  }
}
```

W tym wzorcu w danej chwili czasu tylko jeden producent może wstawiać dane do bufora, analogiczna zależność dotyczy konsumentów. Natomiast wzorzec zachowuje współbieżne czytanie i pisanie danych do/z bufora.

15.3.4 Producent-konsument przy użyciu monitora

Umieszczenie bufora danych wewnątrz monitora i udostępnienie dwóch procedur monitora: *wstaw* i *pobierz* prowadzi do wzorca zilustrowanego pseudokodem:

```
#define ILE ...

monitor BUFOR
{
   const N = ILE;
   var bufor:array[0..N-1] of porcja;
   var ile:integer;
   var zapis:integer;
   var odczyt:integer;
   var PRODUCENCI:condition;
   var KONSUMENCI:condition;

   public procedure WSTAW(element:porcja) {
      if (ile == N) wait(PRODUCENCI); // brak miejsca
```

```
      zapis = (zapis + 1) modulo N;
      bufor[zapis] = element;
      ile++;
      signal(KONSUMENCI);              // jest porcja danych
   }

   public procedure POBIERZ(var element:porcja) {
      if (ile == 0) wait(KONSUMENCI); // brak danych
      odczyt = (odczyt + 1) modulo N;
      element = bufor[odczyt];
      ile--;
      signal(PRODUCENCI);              // jest wolne miejsce
   }

   procedure BUFOR() {
      ile = 0;
      zapis = 0;
      odczyt = 0;
   }
}
```

Tak jak w przypadku wzajemnego wykluczania, implementacja wzorca przy użyciu monitora jest czytelna, a jego użycie oczywiste (P – liczba procesów producentów, K – liczba procesów konsumentów):

```
#concurrently{

   process Producent(i:1..P) {
      while(true) {
         porcja = produkuj();
         BUFOR.WSTAW(porcja);
      }
   }

   process Konsument(i:1..K) {
      while(true) {
         BUFOR.POBIERZ(var porcja);
         konsumuj(porcja);
      }
   }

}
```

Bibliografia

1. Dijkstra E.W.: *Solution to a problem in concurrent programming control.* Communications of the ACM, 8(9):569, 1965.

2. Dijkstra E.W.: *EDW-123 – Cooperating Sequential Processes* (Technical Report). Technische Hogeschool Eindhoven, 1968.
3. Hansen P.B.: *Operating System Principles*. Prentice-Hall, 1973.
4. Hoare C.A.R.: *Monitors: An Operating System Structuring Concept*. CACM, Vol. 17:10, pp. 549–557, 1974
5. Sha L., Rajkumar R., Lehoczky J.P.: *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*. IEEE Transactions on Computers, September 1990, p. 1175

16. Zjawisko wyścigu w programowaniu współbieżnym

Mariusz Matuszek

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
Mariusz.Matuszek@eti.pg.edu.pl

Streszczenie
W rozdziale przedstawiono omówienie podstawowego problemu, z jakim, prędzej czy później, styka się każdy programista piszący oprogramowanie wykorzystujące współbieżność. W praktyce będzie to każdy programista starający się w pełni wykorzystywać moc obliczeniową współczesnych wielordzeniowych procesorów i akceleratorów.

Słowa kluczowe: programowanie współbieżne, wyścig, architektury wielordzeniowe

16.1 Zarys ewolucji procesorów

Aby zrozumieć, jak istotne jest biegłe posługiwanie się metodami programowania współbieżnego należy, choćby pobieżnie, prześledzić ewolucję procesorów, pamiętając, że nadrzędnym celem wprowadzanych udoskonaleń było zawsze przyspieszenie działania procesora. W ramach tego procesu można wyróżnić kilka podstawowych nurtów.

Pierwszym nurtem było proste zwiększanie częstotliwości taktowania procesora oraz, w mniejszym stopniu, podzespołów peryferyjnych, takich jak pamięć RAM. Początkowo metoda ta dawała bardzo zadowalające efekty, ponieważ częstotliwości taktowania pierwszych procesorów były rzędu pojedynczych megaherców. Jednakże, w miarę zwiększania częstotliwości taktu procesora, natrafiono na istotne bariery fizyczne i ograniczenia technologiczne, co zatrzymało tę ścieżkę ewolucji na częstotliwościach taktowania w zakresie pojedynczych gigaherców.

Kolejnym nurtem było powiększanie rozmiaru rejestrów (architektury 4–, 8–, 16–, 32– i 64–bitowe) oraz szerokości i częstotliwości taktowania szyny łączącej procesor z pamięcią RAM. W tym obszarze możliwa jest dalsza ewolucja, która jednakże nie niesie z sobą istotnych zmian jakościowych, jedynie ilościowe. Odłamem tego nurtu są też architektury VLIW (ang. *Very Long Instruction Word*) oraz dedykowane instrukcje wektorowe. Również wprowadzenie wielopoziomowych pamięci podręcznych (ang. *cache*) można umieścić w ramach tego nurtu, którego główną ideą jest przyspieszenie komunikacji procesor–pamięć, oraz przetworzenie lub przesłanie większej porcji danych w ramach jednego taktu.

Inną metodą przyspieszania działania procesorów było wprowadzenie zaawansowanych optymalizacji wykonywania kodu maszynowego na poziomie rdzenia procesora [1]. W tej grupie znajdują się takie techniki jak: przetwarzanie potokowe, wewnętrzne kopie rejestrów (ang. *shadow registers*), czy spekulatywne wykonywanie kodu obejmujące fragmenty obu potencjalnych ścieżek wykonania po wystąpieniu instrukcji warunkowej.

Wszystkie wymienione do tej pory drogi ewolucji koncentrowały się na budowaniu lepszego, szybszego, bardziej zoptymalizowanego pojedynczego rdzenia. Łatwo zauważyć, że każda z tych dróg ewolucji osiągnęła stan, w którym, po okresie początkowych szybkich wzrostów wydajności, możliwe są jedynie drobne, inkrementalne ulepszenia.

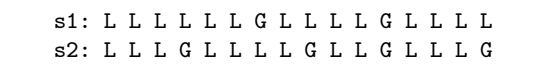
Kolejna z metod zwiększenia wydajności procesorów odchodzi radykalnie od koncentrowania się na ulepszeniu pojedynczego rdzenia i proponuje dodanie do procesora kolejnych rdzeni, działających równolegle. Koncepcja ta jest znakomicie dopasowana do systemów operacyjnych wspierających wieloprocesowość. Dodawanie kolejnych rdzeni okazało się na tyle skuteczną metodą zwiększania mocy obliczeniowej, że pozwoliło producentom procesorów na częściowe zrezygnowanie z niektórych wcześniejszych osiągnięć (np. obniżono częstotliwość taktowania procesorów z ponad 4 GHz do zakresu 2–3 GHz), co pozwoliło na zwiększenie rozmiarów struktury półprzewodnikowej, umożliwiając umieszczenie wielu rdzeni na jednej strukturze.

Zwiększanie liczby rdzeni w celu zwiększenia mocy obliczeniowej procesora okazało się na tyle skuteczne, że w niektórych przypadkach producenci zrezygnowali z używania swoich najbardziej udoskonalonych rdzeni na rzecz umieszczenia w ramach jednej struktury większej liczby rdzeni prostszych (z poprzednich generacji procesora) – w ten sposób skonstruowane są niektóre akceleratory o architekturze many-core [3]. Według tej samej zasady konstruowane są współczesne akceleratory graficzne, integrujące duże liczby prostych rdzeni w ramach jednego układu.

Ostatnia z wymienionych ścieżek ewolucji postawiła przed programistami nowe wymagania związane z efektywnym wykorzystaniem mocy obliczeniowej architektur wielordzeniowych. Wprawdzie programowanie współbieżne było możliwe także na procesorach jednordzeniowych, jednakże było tam stosowane jako np. wygodna forma logicznej organizacji poszczególnych etapów przetwarzania, bądź forma wyodrębnienia elementów architektury programu. Użycie technik programowania współbieżnego na procesorach jednordzeniowych nie było warunkiem niezbędnym do uzyskania dostępu do ich pełnej mocy obliczeniowej, natomiast jest konieczne, aby efektywnie wykorzystywać możliwości architektur wielordzeniowych. Należy podkreślić, że współczesne procesory są prawie bez wyjątku wielordzeniowe oraz daje się zaobserwować trend do stałego zwiększania liczby rdzeni. To zaś pozwala twierdzić, że również w najbliższej przyszłości dobra znajomość zasad programowania współbieżnego (oraz, w szerszej perspektywie, rozproszonego) będzie umiejętnością niezbędną.

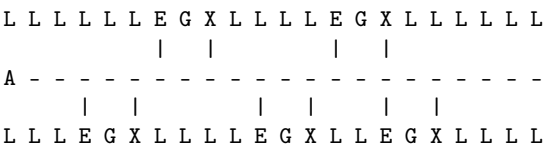
16.2 Program współbieżny

Program współbieżny to program, w którym jednocześnie współistnieją co najmniej dwa strumienie wykonania. Od strony technicznej strumienie te są realizowane przy pomocy wątków lub procesów. Analizując kod takiego wątku/procesu można wyróżnić dwie kategorie operacji: kategoria pierwsza, to operacje na lokalnych zmiennych bądź strukturach danych; kategoria druga, to działania operujące na zmiennych, bądź strukturach danych, współdzielonych przez dwa bądź więcej strumieni wykonania aplikacji. Sytuacja ta została przedstawiona na rysunku 16.1.



Rysunek 16.1. Operacje w strumieniach (s) wykonania: L – dostęp do danych lokalnych, G – dostęp do danych globalnych

Operacje wykonywane na lokalnych danych (ogólnie – lokalnych zasobach) nie mają wpływu na działanie innych strumieni wykonania. Formalnie taki fragment kodu nazywany jest *strefą lokalną*. Szczególnej ostrożności wymagają sekwencje instrukcji operujące na zasobach globalnych (zmiennych współdzielonych etc.), ponieważ istnieje tu bardzo duży potencjał zakłócenia poprawnego działania aplikacji. Z tego powodu takie fragmenty kodu nazywane są (zamiennie) *strefą krytyczną* albo *sekcją krytyczną*. Aby uniknąć błędnego działania aplikacji, w większości przypadków konieczne jest chwilowe zawieszenie współbieżności wykonywania strumieni instrukcji w taki sposób, aby w sytuacji, gdy dwa (bądź więcej) strumienie jednocześnie chcą wykonywać kod sekcji krytycznej, tylko jeden strumień mógł kontynuować działanie. Takie zaaranżowanie wyłączności dostępu do strefy krytycznej formalnie nazywane jest *wzajemnym wykluczaniem*. W celu zapewnienia poprawnego działania wzajemnego wykluczania korzysta się z mechanizmów synchronizacji wspieranych przez system operacyjny. Przykładem takiego mechanizmu są semafory binarne. Kod odpowiedzialny za ochronę wejścia do strefy krytycznej nazywany jest *protokołem wejścia* bądź *protokołem wstępnym*, natomiast kod sygnalizujący zakończenie operacji wewnątrz strefy krytycznej nazywany jest *protokołem końcowym* bądź *protokołem wyjścia*. Wszystkie wymienione elementy zilustrowane są na rysunku 16.2.



Rysunek 16.2. Ochrona stref krytycznych: L – operacje lokalne, G – operacje dostępu do danych globalnych, E – protokół wejścia, X – protokół wyjścia, A – arbiter dostępu

Omawianie konstrukcji typowych wzorców programowania współbieżnego wykracza poza zakres tego rozdziału. Czytelnika zainteresowanego pogłębieniem wiedzy w tym zakresie zachęcam do zapoznania się z [2] oraz rozdziałem 15. Dalsza część rozdziału poświęcona jest zilustrowaniu kluczowego zjawiska, jakim jest wyścig (ang. *race condition*).

16.3 Wyścig

Analizę zjawiska wyścigu rozpoczniemy od prostego programu sekwencyjnego.

```
#include <stdio.h>

#define INCREMENTS  40000

int global_variable = 0;

int main()
{
   int local_counter = INCREMENTS;

   while(local_counter--) {
      global_variable = global_variable + 1;
   }

   printf("wynik = %d\n", global_variable);
   return 0;
}
```

W programie występuje tylko pojedyncza ścieżka wykonania kodu, nie jest więc to program współbieżny, niemniej ilustruje kilka istotnych koncepcji. Linia

```
#define INCREMENTS  400000
```

określa *ilość pracy* do wykonania w ramach zaimplementowanego algorytmu. Kolejna linia

```
int global_variable = 0;
```

definiuje zmienną globalną. Jest to bardzo prosta, ale w zupełności wystarczająca, ilustracja *globalnego stanu* w aplikacji. Wreszcie linie

```
while(local_counter--) {
   global_variable = global_variable + 1;
}
```

implementują główny algorytm aplikacji, w ramach którego następuje zarówno użycie zmiennej lokalnej (local_counter), jak i modyfikowanie stanu globalnego

aplikacji (zmienna global_variable). Algorytm inkrementuje zmienną globalną zgodnie z zadaną ilością pracy. Oczekiwanym wynikiem działania jest stan końcowy zmiennej globalnej równy 400000. I rzeczywiście, po skompilowaniu i wykonaniu tej aplikacji taki wynik zostanie wyświetlony.

```
$ gcc -Wall prog1.c -o prog1
$ ./prog1
wynik = 400000
```

Dla współczesnych procesorów zadana w tym przykładzie ilość pracy jest trywialna i najprawdopodobniej więcej czasu od właściwych obliczeń zajmie odczytanie pliku wykonywalnego z pamięci masowej i jego uruchomienie. Niemniej, dla celów dydaktycznych załóżmy, że zadana ilość pracy jest duża i jest zasadne jej rozdzielenie pomiędzy osobne wątki w celu uzyskania przyspieszenia działania aplikacji oraz wykorzystania możliwości sprzętowych współczesnych procesorów. W tym celu przenieśmy samo wykonywanie algorytmu do wnętrza wątków, przydzielając każdemu wątkowi część pracy do wykonania. Ponieważ współczesne procesory są wielordzeniowe możemy oczekiwać, że system operacyjny przydzieli każdy z wątków do osobnego rdzenia i uzyskamy faktycznie współbieżne wykonanie zadanej pracy. Program realizujący (w sposób błędny!) ten sam algorytm w wersji wielowątkowej zaprezentowany jest poniżej.

```
#include <stdio.h>
#include <pthread.h>

#define INCREMENTS  400000

int global_variable = 0;

void *thread_f(void *vp)
{
   int local_counter = *(int *)vp;

   while(local_counter--) {
      global_variable = global_variable + 1;
   }
   return NULL;
}


int main()
{
   pthread_t t1, t2, t3, t4;
   int work_per_thread = INCREMENTS / 4;

   pthread_create(&t1, NULL, thread_f, &work_per_thread);
   pthread_create(&t2, NULL, thread_f, &work_per_thread);
```

```
   pthread_create(&t3, NULL, thread_f, &work_per_thread);
   pthread_create(&t4, NULL, thread_f, &work_per_thread);

   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   pthread_join(t3, NULL);
   pthread_join(t4, NULL);

   printf("wynik = %d\n", global_variable);
}
```

Główne zmiany w stosunku do programu sekwencyjnego to przeniesienie implementacji algorytmu do wnętrza funkcji wątku:

```
void *thread_f(void *argp)
{
   int local_counter = *(int *)argp;

   while(local_counter--) {
      global_variable = global_variable + 1;
   }
   return NULL;
}
```

oraz uruchomienie wątków z wnętrza funkcji main, z przydzieleniem ilości pracy do wykonania wewnątrz wątku za pomocą zmiennej work_per_thread.:

```
pthread_create(&t1, NULL, thread_f, &work_per_thread);
...
```

Ponieważ przedwczesne zakończenie funkcji main spowodowałoby zakończenie działania programu, co byłoby tożsame z przedwczesnym przerwaniem i zakończeniem działania wątków, funkcja main czeka na zakończenie działania każdego z wątków przy pomocy funkcji pthread_join:

```
pthread_join(t1, NULL);
...
```

Czytelnik zainteresowany dokładniejszym poznaniem arsenału funkcji tworzących interfejs do programowania wielowątkowego zgodnego z specyfikacją POSIX znajdzie obszerne informacje w podręczniku systemu UNIX, między innymi:

```
$ man pthreads
$ man pthread_create
$ man pthread_join
```

Jak już zostało zaznaczone, powyższy program wykona się niepoprawnie. Po skompilowaniu i uruchomieniu programu wyświetlony zostanie błędny wynik, którego wartość będzie przypadkowa:

```
$ gcc -Wall -pthread prog2.c -o prog2
$ ./prog2
wynik = 132003
$ ./prog2
wynik = 181383
$ ./prog2
wynik = 120757
...
```

Powodem błędnego działania programu jest wyścig, czyli sytuacja, w której dwa (bądź więcej) wątki jednocześnie dokonują modyfikacji stanu globalnego. Aby zrozumieć istotę problemu należy zauważyć, że operacja inkrementacji zmiennej:

```
global_variable = global_variable + 1;
```

nie jest operacją atomową (to znaczy, nie jest niepodzielna). Na poziomie instrukcji procesora, operacja ta składa się z sekwencji rozkazów:

- prześlij z pamięci do akumulatora aktualną wartość zmiennej,
- zwiększ akumulator,
- prześlij nową wartość z akumulatora do pamięci.

Aby przekonać się, że tak jest w istocie, należy przeanalizować kod asemblera generowany przez kompilator C. W tym celu należy wykonać polecenie:

```
$ gcc -Wall -c -S prog2.c
```

Wynikiem powyższego polecenia jest plik prog2.s, zawierający kod źródłowy na poziomie asemblera (domyślnie w notacji AT&T). Fragment tego kodu, odnoszący się do funkcji wątku, wygląda następująco:

```
thread_f:
.LFB2:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp)
        movq    -24(%rbp), %rax
        movl    (%rax), %eax
        movl    %eax, -4(%rbp)
        jmp     .L2
.L3:
        movl    global_variable(%rip), %eax
        addl    $1, %eax
        movl    %eax, global_variable(%rip)
.L2:
```

```
        movl    -4(%rbp), %eax
        leal    -1(%rax), %edx
        movl    %edx, -4(%rbp)
        testl   %eax, %eax
        jne     .L3
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Należy zwrócić uwagę na sekwencję instrukcji pomiędzy etykietami L3 i L2. Jest ona odpowiedzialna za modyfikację zmiennej globalnej. Występują tu dwa problemy. Po pierwsze, każda z instrukcji procesora jest atomowa, ale już wykonanie sekwencji instrukcji może zostać w dowolnej chwili wstrzymane (przerwane) przez zarządcę (algorytm przydziału czasu procesora do zadań, będący składową systemu operacyjnego) *pomiędzy instrukcjami* (w takiej sytuacji mówimy, że wątek lub proces został *wywłaszczony*). W takim przypadku możliwy jest, na przykład, taki scenariusz:

- wątek1: pobiera wartość aktualną zmiennej globalnej i zostaje wstrzymany,
- wątek2: inkrementuje zmienną globalną wielokrotnie,
- wątek1: zostaje wznowiony, inkrementuje swoją (starą, już nieaktualną) kopię wartości zmiennej globalnej i zapisuje wynik.

Po drugie, nawet gdyby zarządca nie wstrzymał wykonania żadnego z wątków, możliwa jest na przykład taka kombinacja zdarzeń:

- wartość początkowa zmiennej globalnej równa x,
- wątek1: wczytuje wartość początkową zmiennej (x),
- wątek2: wczytuje wartość początkową zmiennej (x),
- wątek1: zwiększa lokalną kopię wartości (wynik: $x+1$),
- wątek2: zwiększa lokalną kopię wartości (wynik: $x+1$),
- wątek1: zapisuje wynik ($x+1$) do zmiennej globalnej,
- wątek2: zapisuje wynik ($x+1$) do zmiennej globalnej,
- wartość końcowa zmiennej globalnej równa $x+1$ (zamiast $x+2$).

Jak widać, źródłem błędnego działania programu jest wyścig w sekcji krytycznej. Aby naprawić błąd i przywrócić poprawne działanie programu należy użyć odpowiedniego protokołu wejścia do sekcji krytycznej, który pozwoli tylko jednemu wątkowi na raz wykonać kod zmieniający stan zmiennej globalnej. Następnie należy użyć odpowiedniego protokołu wyjścia z sekcji krytycznej, który pozwoli kolejnemu wątkowi na wejście do sekcji krytycznej. Należy więc odpowiednio zmodyfikować poniższy fragment funkcji wątku:

```
while(local_counter--) {
   // protokół wejścia do sekcji krytycznej
   global_variable = global_variable + 1;
   // protokół wyjścia z sekcji krytycznej
}
```

W kodzie realizującym protokoły wejścia i wyjścia z sekcji krytycznej należy odwołać się do wymienionego na rysunku 16.2 mechanizmu arbitrażu dostępu, przy pomocy którego zostanie zrealizowane wzajemne wykluczanie się wątków w dostępie do sekcji krytycznej. W zaprezentowanym poniżej przykładzie takim mechanizmem jest *semafor binarny*. Poprawny kod programu wygląda następująco:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <errno.h>

#define INCREMENTS  400000

int global_variable = 0;
sem_t semaphore;


void *thread_f(void *vp)
{
   int local_counter = *(int *)vp;

   while(local_counter--) {
      sem_wait(&semaphore);
      global_variable = global_variable + 1;
      sem_post(&semaphore);
   }
   return NULL;
}


int main()
{
   pthread_t t1, t2, t3, t4;
   int work_per_thread = INCREMENTS / 4;

   if (sem_init(&semaphore, 0, 1) != 0) {
      perror("sem_init() failed\n");
      return 1;
   }

   pthread_create(&t1, NULL, thread_f, &work_per_thread);
   pthread_create(&t2, NULL, thread_f, &work_per_thread);
   pthread_create(&t3, NULL, thread_f, &work_per_thread);
   pthread_create(&t4, NULL, thread_f, &work_per_thread);
```

```
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    pthread_join(t4, NULL);

    printf("wynik = %d\n", global_variable);
}
```

Tym razem skompilowanie kodu i jego wykonanie da wynik poprawny:

```
$ gcc -Wall -pthread prog3.c -o prog3
$ ./prog3
wynik = 400000
```

Celem rozdziału było przybliżenie zjawiska wyścigu. Dokładne omówienie zasady działania semaforów wykracza poza zakres tego rozdziału i zainteresowany czytelnik jest ponownie zachęcany do zapoznania się z literaturą [2, 4, 5], jednakże, w skrócie, działanie operacji semaforowych (funkcji) sem_wait i sem_post jest następujące:

- *sem_wait(semaphore)*: funkcja sprawdza, czy zmienna wewnętrzna wskazanego semafora jest różna od zera. Jeżeli nie, wołający wątek zostaje wstrzymany i przeniesiony do kolejki wątków oczekujących na tym semaforze. W przeciwnym przypadku zmienna wewnętrzna semafora jest dekrementowana i funkcja wraca, co jest jednoznaczne z wpuszczeniem wątku do ochranianej sekcji krytycznej;
- *sem_post(semaphore)*: jeżeli w kolejce wątków oczekujących na tym semaforze są zarejestrowane oczekujące wątki (kolejka nie jest pusta), to wywołanie tej operacji powoduje przeniesienie jednego z oczekujących wątków do kolejki gotowych do wykonania (wznowienie wykonania). Następnie operacja wraca do wątku wołającego, który kontynuuje działanie poza chronioną sekcją krytyczną. W przypadku, gdy kolejka wątków oczekujących jest pusta, funkcja inkrementuje zmienną wewnętrzną wskazanego semafora (o ile nie spowoduje to przekroczenia górnego ograniczenia wartości tej zmiennej) i wraca do wątku wołającego.

Bibliografia

1. Shen J.P., Lipasti M.H.: *Modern Processor Design: Fundamentals of Superscalar Processors*. Waveland Press, Inc.; 1st edition (July 30, 2013), ISBN:978-1478607830
2. Ben-Ari M.: *Podstawy Programowania Współbieżnego*, WNT Warszawa.
3. Intel® Many Integrated Core Architecture – https://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html
4. https://en.wikipedia.org/wiki/Concurrent_computing
5. https://www.toptal.com/software/introduction-to-concurrent-programming

17. Klasyfikacja aktywności kory wzrokowej za pomocą elektroencefalografu

Jakub Atroszko

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
jakub.atroszko@pg.edu.pl

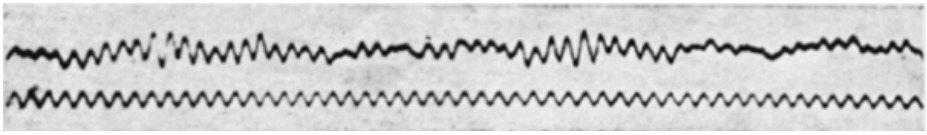
Streszczenie
W niniejszej pracy została przedstawiona metodologia konstrukcji i oceny systemu cyfrowego automatycznie klasyfikującego dane pochodzące z elektroencefalografu. Opracowana procedura badawcza pozwoliła na przetestowanie rozwiązania na różnych osobach, w różnym wieku, o różnych porach dnia, z wykorzystaniem różnych konfiguracji urządzeń i modeli zjawiska. Uzyskano stuprocentową skuteczność automatycznego rozpoznania stanu spoczynkowego kory wzrokowej. Architektura rozwiązania została oparta o algorytm maszyny wektorów nośnych, a także wektorowe reprezentacje aktywności mózgu utworzone
z wykorzystaniem szybkiej transformacji Fouriera.

Słowa kluczowe: elektroencefalografia, fale, aktywność mózgu, analiza sygnału, transformacja Fouriera, maszyna wektorów nośnych, klasyfikacja

17.1 Wprowadzenie

Elektryczność jest podstawowym elementem w życiu każdego człowieka, a mimo tego każdy z nas może inaczej ją postrzegać. Współczesny człowiek przyzwyczajony do pracy przy świetle żarówki inaczej oceni znaczenie elektryczności w przypadku awarii sieci energetycznej, niż przykładowo grecki filozof Tales z Miletu żyjący na przełomie VII i VI w. p.n.e., który rozpoznał charakterystyczne zachowanie lekkich obiektów w otoczeniu bursztynu potartego futrem [12].

Potrzebny był czas do osiągnięcia technologii pozwalającej na pełniejsze zrozumienie istoty elektryczności w życiu codziennym. Szczególnie ważnym i interesującym odkryciem dokonanym przez Hansa Bergera opublikowanym w 1929 roku było nagranie elektrycznej aktywności ludzkiego mózgu zarejestrowanej na powierzchni głowy. Wykorzystano w tym celu galwanometr z podwójną cewką. Nagrano w ten sposób sygnał, w którym rozpoznano fale oscylujące w zakresie częstotliwości nazywanym alfa. Poniżej (rys. 17.1) znajduje się pierwszy w historii zapis elektroencefalograficzny (EEG) człowieka. W 1932 roku Hans Berger wraz z asystentem Guntherem Dietschem w celu lepszego poznania obserwowanego zjawiska zastosowali jedną z dostępnych wówczas metod - analizę Fouriera [35].



Rysunek 17.1. Górna linia jest zapisem dokonanym na młodym synu Bergera. Dolna linia reprezentuje 10 Hz sinusoidę i służy jako znacznik czasu [35]

17.2 Analiza Fouriera

Analiza Fouriera jest znaną i powszechnie stosowaną metodą analizy danych w niezliczonej liczbie dziedzin naukowych i inżynieryjnych między innymi w komunikacji, przetwarzaniu sygnałów, spektroskopii czy projektowaniu obwodów [32].

Podstawowa idea stojąca za różnymi metodami opartymi na koncepcjach znanych z analizy Fouriera to twierdzenie, że dowolny sygnał okresowy jest równoważny sumie serii odpowiednio wyliczonych sinusoid [36].

Dekompozycja sygnału w zbiór sinusoidalnych funkcji o różnych częstotliwościach, amplitudach i fazach jest znanym i szeroko stosowanym podejściem do ekstrakcji cech z szeregów czasowych jednakże z definicji wykorzystywane w niej funkcje są nieskończone w dziedzinie czasu, dlatego podejście oparte o wyłącznie matematyczne koncepty ma ograniczone zastosowanie w rzeczywistych scenariuszach inżynierskich [28].

Obecnie większość analiz Fouriera jest wykonywanych przy pomocy cyfrowych komputerów i danych w postaci dyskretnej stąd fundamentalnie są to zbiory skończone. Przejście z sygnału na równoważny zbiór sinusoid jest znany jako transformacja Fouriera natomiast wersja dyskretna jako dyskretna transformacja Fouriera. Poniżej znajduje się wzór implementowany jako metoda komputerowa pozwalający na wyliczenie dyskretnych w czasie szeregów Fouriera [36]:

$$X[m] = \sum_{n=1}^{N} x[n] e^{-\frac{j2\pi mn}{N}}, \tag{17.1}$$

gdzie:
m - liczba harmoniczna, m = 1, 2, 3, ... , M
M - liczba ewaluowanych komponentów
n - indeks tablicy z danymi długości N.

W popularnym algorytmie znanym jako szybka transformacja Fouriera, M jest równe liczbie punktów w danych N [36].

Szybka transformacja Fouriera jest praktyczną implementacją metody obliczeniowej najprawdopodobniej znanej już od 1805 roku, która pozwala na zastosowanie analizy Fouriera w rzeczywistych problemach. Podstawową ideą jest wyrażenie wartości w dziedzinie czasu za pomocą wartości w dziedzinie częstotliwości, tak aby dane wejściowe mogły zostać odtworzone na podstawie danych

wyjściowych. W wykorzystanej w niniejszej pracy implementacji jeśli długość wektora danych jest liczbą parzystą, to wartość zwrócona odpowiadająca środkowemu elementowi ciągu reprezentuje częstotliwość Nyquista [5].

17.3 Częstotliwość Nyquista

Częstotliwość Nyquista jest równa połowie częstotliwości próbkowania, oznacza ona maksymalną częstotliwość dla której można precyzyjnie odtworzyć zarejestrowany sygnał. W przypadku wystąpienia składowych powyżej tej częstotliwości przy danej częstości próbkowania nastąpi nieodwracalne zniekształcenie sygnału ze względu na niewystarczającą liczbę informacji pozwalających na jego wierne odtworzenie. Kryterium Nyquista jest bardzo istotną zasadą odnoszącą się do cyfrowej analizy spróbkowanego sygnału. Definiuje ono minimalną wymaganą częstotliwość próbkowania do jednoznacznej reprezentacji wartości obserwowanego widma [19, 27, 31, 33, 38].

17.4 Elektroencefalografia

Współcześnie sądzimy, że sygnał EEG pochodzi z potencjału powstałego na skutek synchronicznych wyładowań wystarczająco dużych grup neuronów piramidalnych w korze mózgowej, które są zorientowane prostopadle do powierzchni mózgu [30].

Zmiany elektrochemiczne zachodzące w organizmie generują zaburzenia pola elektromagnetycznego, które może być modulowane w czasie i przestrzeni za pomocą obecnie dostępnych narzędzi. Potrafimy wyróżnić charakterystyczne pasma częstotliwości od bardzo wolnych poniżej 0.01Hz do ultraszybkich powyżej 1kHz, a także przypisać im znaczenie powiązane ze stanami behawioralnymi takimi jak sen, pobudzenie, świadomość [24].

Powszechnie znanych jest pięć tak zwanych fal mózgowych. W poniższej tablicy 17.1 znajduje się przykładowy podział [1, 18].

Tablica 17.1. Charakterystyka fal mózgowych [1, 18]

Nazwa	**Pasmo**	**Stan**
Gamma	powyżej 35 Hz	Koncentracji i działania
Beta	12–35 Hz	Gotowości, aktywnej zewnętrznej uwagi
Alpha	8–12 Hz	Spoczynkowy, zrelaksowany
Theta	4–8 Hz	Odprężenia, wewnętrznego skupienia
Delta	0.5–4 Hz	Snu, głębokiej medytacji

Do ekstrakcji interesujących cech obserwowanego zjawiska wykorzystuje się różne podejścia oparte o analizę danych w dziedzinach czasu lub częstotliwości. Wśród nich można wyróżnić szybką transformację Fouriera (ang. *Fast Fourier*

Transform, FFT), ale warto również wspomnieć o metodach opartych o wektory własne (ang. *eigenvector methods*), transformacje falkowe (ang. *wavelet transform*) czy modele autoregresyjne (ang. *auto regressive model*) [21]. Niektórzy badacze sugerują również użyteczność analiz opartych o teorie grafowe w celu rozróżnienia sieci funkcjonalnych w mózgu na podstawie zmian obserwowanych w stanie spoczynku wpływających na zachodzącą komunikację w systemie [39].

Spoczynek jest jednym ze stanów, który można zaobserwować i scharakteryzować za pomocą EEG. Rejestrowane różnice występujące w zależności, czy oczy są otwarte pozwalają na wnioskowanie odnośnie sposobu działania mózgu [22].

W jednym z badań EEG przeprowadzonym na grupie dwudziestu młodych i dwudziestu starszych uczestników stwierdzono, że otworzenie oczu powoduje redukcję globalnej aktywności mózgu we wszystkich pasmach niezależnie od wieku, przy czym w grupie starszych badanych, wykryto więcej zmian lokalnych w paśmie beta sugerując dodatkową mobilizację zasobów w wybranych obszarach, co może tłumaczyć sposób, w jaki mózg organizuje swoją pracę wraz z procesem starzenia się [23].

Częstotliwość rytmu alfa osiąga najwyższą wartość około dwudziestego roku życia i z wiekiem zaczyna spadać. Asymetria w zakresie alfa, może być wykorzystana jako neuromarker depresji, zaś brak rytmu alfa jest rozpoznany u dziesięciu procent populacji charakteryzującej się zaburzeniami lękowymi [29].

17.5 Maszyna wektorów nośnych

Problem przypisania obiektów do dwóch grup jest znanym zagadnieniem w uczeniu maszynowym określanym jako klasyfikacja binarna. Zbiór danych wraz z informacją o przynależności do wybranej grupy można wyrazić za pomocą następującego wyrażenia 17.2:

$$(y_1, x_1), ..., (y_l, x_l), y_i \in \{-1, 1\} \tag{17.2}$$

Jednym ze skutecznych sposobów na automatyczne sklasyfikowanie takiego zbioru danych jest algorytm znany jako maszyna wektorów nośnych (ang. *Support-Vector Machine* lub *Support-Vector-Networks* w skrócie *SVM*) [26].

Metoda opiera się na idei zmapowania danych na podstawie przynależności obiektów do grupy na wysokowymiarową przestrzeń cech w celu znalezienia marginesu wyznaczonego przez tak zwane wektory wspierające (ang. *support vectors*), umożliwiającego optymalną separacje obiektów za pomocą hiperpłaszczyzny liniowej [26].

Zbiór danych jest liniowo separowalny, jeśli istnieje wektor w i skalar b taki, że poniższa nierówność 17.3 jest prawdziwa dla każdego elementu zbioru treningowego [26]:

$$y_i(w \cdot x_i + b) \geq 1, i = 1, ..., l \tag{17.3}$$

Optymalna hiperpłaszczyzna dana jest równaniem 17.4 [26]:

$$w_0 \cdot x + b_0 = 0 \tag{17.4}$$

Determinuje ona kierunek $w/|w|$ w taki sposób, aby odległość obiektów po projekcji była maksymalna co przedstawia poniższe równanie 17.5 [26]:

$$p(w,b) = min_{x:y=1}\frac{x \cdot w}{|w|} - max_{x:y=-1}\frac{x \cdot w}{|w|} \tag{17.5}$$

17.6 System

System bazujący na metodologii przedstawionej w niniejszej pracy został oparty o zbiór programów napisanych w języku Python. Odpowiednie ich użycie w formie spójnej procedury badawczej opisanej szczegółowo pozwala na zebranie danych wymaganych do stworzenia i oceny klasyfikatora.

Urządzenie EEG wykorzystane w niniejszej pracy do rejestracji sygnału to OpenBCI Cyton Daisy, jednakże ze względu potrzebę minimalizacji czasu wymaganego do przebadania i przetestowania rozwiązania na różnych osobach zdecydowano się ograniczyć liczbę kanałów do 1, stąd wykorzystano w rzeczywistości platformę Cyton bez modułu Daisy [2, 3].

Akwizycja danych oparta została o bibliotekę BrainFlow napisaną w języku Python, która wspiera różnego rodzaju narzędzia do rejestracji sygnałów biologicznych [10].

Implementacja w języku Python klasyfikatora opartego o algorytm maszyny wektorów nośnych pochodzi z biblioteki scikit-learn [9].

Selekcja cech w postaci średniej mocy sygnału w paśmie alfa została zaimplementowana z wykorzystaniem dyskretnej szybkiej transformacji Fouriera zaimplementowanej w języku Python w bibliotece NumPy [5].

17.6.1 Akwizycja danych

W celu akwizycji danych w ramach systemu potrzebne jest odpowiednie oprogramowanie obsługujące urządzenie EEG, rejestrujące sygnał przez wybraną długość czasu i zapisujący dane w ustalonej formie na dysku. Dlatego został stworzony program bazujący na formacie danych opisanym w dokumentacji BrainFlow [4] pozwalający na rejestrację sygnału przez ustaloną długość czasu t za pomocą skonfigurowanych c kanałów i zapisujący zebrane dane w pliku na dysku. Realizował on następujące instrukcje:

1. Konfiguracja urządzenia EEG w tym między innymi liczby kanałów c faktycznie rejestrujących dane.
2. Odmierzenie x sekund.
3. Usunięcie wszystkich zarejestrowanych do tej pory danych z bufora.
4. Odmierzenie t sekund.
5. Zapisanie wszystkich danych z bufora do pliku.

Pierwsze x sekund ma na celu przygotowanie się uczestników do eksperymentu. Na samym początku mogą wystąpić różnego rodzaju aktywności wpływające na jakość rejestrowanych danych, ale zazwyczaj po pewnym czasie różnica zmian

w środowisku ustali się na pewnym poziomie zapewniającym minimalizację rejestrowanego szumu. Natomiast t wyznacza rozmiar okna czasowego, tj. liczbę próbek. W niniejszej pracy arbitralnie zostały wybrane parametry $x = 10$ i $t = 120$, gdzie przy częstości próbkowania urządzenia równej 250Hz, 120 sekund oznacza 30 000 próbek sygnału. W rzeczywistości rozmiar zebranych danych był większy od oczekiwanego. Błąd pomiaru można ustalić na podstawie faktycznej liczby zebranych próbek, czasu odmierzanego przez program, częstości próbkowania urządzenia podawanej przez producenta w specyfikacji produktu i czasu zegarowego. Różnica w liczbie danych względem oczekiwań wynika z niedoskonałości narzędzi badawczych, między innymi ze względu na niewykorzystanie rezonatora kwarcowego w platformie Cyton, z tego powodu częstość próbkowania może się wahać i wynosić np. 250.05 lub 249.92 itp. [7]

Również istotne znaczenie mogą mieć różnice pomiędzy czasem rzeczywistym w trakcie którego były zbierane dane EEG przez urządzenie, a czasem odmierzanym przez program wykonujący różnego typu operacje w tym komunikację z urządzeniem EEG.

W niniejszej pracy do obliczeń opisanych poniżej były wykorzystywane pierwsze trzydzieści tysięcy zapisanych wartości z pliku reprezentującego dwuminutowy zapis aktywności podczas wybranego stanu. Warto również wspomnieć o znaczeniu zmiennej c. Przetestowano działanie systemu dla różnych wartości $c \in \{1, 4, 8\}$. Czas potrzebny na założenie większej liczby elektrod znacząco wydłuża czas badania, zaś samo urządzenie posiada wbudowane mechanizmy rejestracji i wstępnego przetworzenia sygnału oparte o wykorzystanie ustawionych kanałów. Dlatego test w formie wykorzystania różnej liczby kanałów jest istotny z perspektywy zapewnienia jakości uzyskanych wyników przez system.

17.6.2 Konstrukcja i walidacja klasyfikatora

Posiadając zebrane próbki za pomocą powyżej przedstawionego programu, można utworzyć reprezentacje wektorowe na podstawie ustalonego algorytmu selekcji cech sygnału. W niniejszej pracy wykorzystano poniższy schemat:

1. Wczytaj n nagrań z etykietami. Wyrównaj liczbę próbek w każdym nagraniu do ustalonej liczby m.
2. Podziel nagranie będące ciągiem próbek o długości m na g podciągów o rozmiarze r.
3. Przetransformuj uzyskane ciągi za pomocą szybkiej transformacji Fouriera do postaci ciągu reprezentującego widmo częstotliwościowe.
4. Dla każdego ciągu wybierz wartości z przedziału reprezentujące pasmo od a do b Hz.
5. Na podstawie wybranych wartości wylicz średnią moc w danym paśmie.
6. Utwórz reprezentacje wektorowe na podstawie wybranych cech.
7. Utworzone wektory podziel na dwa ciągi uporządkowane względem przebiegu w czasie reprezentujące różne klasy.
8. Na podstawie wybranego momentu w czasie s podziel zbiór na dane trenujące i walidujące.

9. Wykorzystaj zbiór trenujący do utworzenia klasyfikatora opartego o maszynę wektorów nośnych.
10. Sklasyfikuj pozostałe dane za pomocą utworzonego modelu.
11. Wypisz wynik w formie procentowej stosunku liczby poprawnie przewidzianych etykiet wektorów do łącznej liczby wektorów w zbiorze walidującym.

W niniejszej pracy liczba nagrań była równa $n = 4$, po dwa nagrania dwuminutowe dla wybranej klasy, zaś $m = 30000$, dając łącznie 120 000 próbek. Przyjęto a priori $r = 5000$, co pozwoliło na utworzenie $g = 24$ wektorów, które przy $s = 12$ wyrażonym w liczbie wektorów lub równoważnie $s = 240$ definiujące liczbę sekund pozwoliło na utworzenie klasyfikatora i jego walidację.

17.6.3 Program rejestrujący dane testowe

Wynik liczbowy określający skuteczność utworzonego klasyfikatora za pomocą przedstawionego w poprzedniej podsekcji algorytmu został wyznaczony dzięki programowi bazującemu na metodach opisanych w poprzedniej podsekcji o akwizycji danych i realizował on następujące instrukcje:

1. Konfiguracja urządzenia EEG w tym liczby używanych kanałów. Odmierzenie x sekund. Usunięcie wszystkich zarejestrowanych do tej pory danych.
2. Wypisanie oczekiwanego stanu badanego: oczy otwarte lub zamknięte. Badający mógł dzięki temu ustalić poprawność etykiet przypisywanych do zbieranych próbek.
3. Z prawdopodobieństwem p zagranie dźwięku sinusoidalnego o częstotliwości 440Hz i długości nagrania 50 milisekund. Była to informacja dla badanego, aby zmienił swój stan na przeciwny, czyli jeśli miał oczy otwarte to je zamykał i vice versa.
4. Odmierzenie t sekund.
5. Zapisanie do pliku zarejestrowanych danych wraz z etykietą określającą przynależność do klasy wybranego zbioru próbek.
6. Powrót do punktu 2 dopóki nie zebrano n wektorów.
7. Odtworzenie serii pięciu dźwięków informujących o zakończeniu badania.

W niniejszej pracy zostały wykorzystane $t = 20$, $n = 50$, $p = 0{,}5$. Analogicznie postępując jak w algorytmie opisanym w poprzedniej sekcji, utworzono na podstawie $n = 50$, wektory reprezentujące średnią moc w paśmie alfa. Pięćdziesiąt okien czasowych o długości dwudziestu sekund oznacza niecałe siedemnaście minut badania, tj. tysiąc sekund.

17.6.4 Program automatycznie klasyfikujący zbierane dane

Program umożliwiający prezentację systemu w praktyce i poznanie opinii użytkownika odnośnie sposobu jego działania opierał się o następujący schemat:

1. Utworzenie i walidacja modelu zgodnie z wcześniej przedstawionym algorytmem z wykorzystaniem wcześniej zebranych danych - definicja rozmiaru r okna czasowego.

2. Konfiguracja urządzenia EEG w tym liczby kanałów.
3. Zbieranie i przetwarzanie zgodnie z wcześniej przedstawionym algorytmem danych o rozmiarze r.
4. Granie dźwięku na podstawie wyniku klasyfikacji przetworzonego okna czasowego.

17.7 Akwizycja danych

W niniejszej pracy do rejestracji sygnału EEG z częstością próbkowania 250Hz [2] zostały użyte złote pasywne elektrody [6] przyklejone do ciała za pomocą pasty przewodzącej Ten20 Conductive Paste [11] i zestaw narzędzi Cyton Daisy Biosensing Boards bez modułu Daisy [2].

17.7.1 Ustalenie lokalizacji elektrod

Lokalizacja obszaru umożliwiającego badanie aktywności kory wzrokowej została ustalona przy pomocy międzynarodowego standardu znanego jako 10-20 system. Konkretnie w niniejszej pracy przetwarzanie danych zostało ograniczone do jednego kanału zarejestrowanego sygnału na pozycji O1 [13, 16, 17]. Procedura umiejscowienia elektrody na głowie uczestnika badania była następująca; za pomocą miarki krawieckiej:

1. Zmierz odległość d od miejsca pomiędzy oczami, gdzie kończy się nos (punkt A) do charakterystycznego wybrzuszenia na kości potylicznej, wyczuwalnego przez przesunięcie ręki wzdłuż szyi przy odpowiednim nacisku na skórę w kierunku czubka głowy (punkt B) [15].
2. Zmierz obwód głowy O wzdłuż linii przechodzącej przez punkty odległe o $0{,}1d$ od A i B w kierunku czubka głowy.
3. Przymocuj elektrodę na lewo od B w odległości $0{,}05O$ wzdłuż linii pomiaru obwodu głowy.

17.7.2 Procedura badawcza

W celu przeprowadzenia badania postępowano według następującej procedury:

1. Wyjaśnienie celu badań i kolejnych kroków niniejszej procedury wraz z przybliżonym czasem potrzebnym na ich zrealizowanie. Zwrócenie uwagi i wyjaśnienie wpływu na jakość zapisów wszelkich ruchów ciała, takich jak nadmierne mruganie lub ruszanie językiem. Rekomendowane było usytuowanie języka z tyłu dolnych zębów w celu redukcji szumów i mruganie w sposób naturalny, kiedy jest taka potrzeba.
2. Prośba o zajęcie wygodnej, zrelaksowanej pozycji siedzącej.
3. Wyznaczenie na wprost osoby badanej punktu na którym można komfortowo skupić swój wzrok i oczekiwać w stanie spoczynku na komendy.
4. Dokonanie pomiarów głowy wraz z umieszczeniem elektrody zgodnie z opisem przedstawionym w poprzedniej podsekcji.

5. Włączenie urządzenia EEG i poproszenie o przygotowanie się i potwierdzenie gotowości do rozpoczęcia nagrywania.
6. W chwili uzyskania potwierdzenia gotowości, uruchomienie programu rejestrującego za pomocą EEG wybraną aktywność w ustalonym oknie czasowym. Poinformowanie o rozpoczęciu nagrania.
7. Badający cały czas zapewnia, że osoba badana jest w ustalonym i oczekiwanym stanie.
8. Po zakończeniu pracy programu i nagraniu próbki zgodnej z oczekiwaniami - potwierdzenie istnienia pliku na dysku poprzez nadanie mu odpowiedniej etykiety i poinformowanie o tym badanego.
9. Powyższe cztery punkty były powtarzane czterokrotnie na przemian z oczami otwartymi lub zamkniętymi u badanego.
10. W chwili zebrania wszystkich próbek - dwóch z oczami zamkniętymi i dwóch z oczami otwartymi, uruchomienie programu walidującego model.
11. Poinformowanie badanego o uzyskanej skuteczności klasyfikatora na zbiorze walidującym, w przypadku uzyskania oczekiwanego wyniku przejście do kolejnych punktów, w przypadku przeciwnym zatrzymanie procedury do czasu ustalenia przyczyny.
12. Potwierdzenie gotowości badanego do etapu zebrania danych testowych. Uruchomienie programu.
13. Poinformowanie o skuteczności klasyfikatora na zbiorze testowym.
14. Uruchomienie programu demonstrującego działanie systemu.
15. Ustalenie opinii użytkownika o działaniu programu.

17.8 Wyniki

W niniejszej pracy zgodnie z opisaną procedurą badawczą postąpiono w przypadku dwóch osób: osoba A poniżej 30 roku życia i osoby B powyżej 60 roku życia. Klasyfikator wytrenowany na danych pochodzących z badania EEG osoby X oznaczmy jako Klasyfikator X, analogicznie skuteczność uzyskaną w klasyfikacji danych testowych zebranych z wykorzystaniem osoby X jako Skuteczność X. Poniższa tablica 17.2 przedstawia skuteczność klasyfikatora na danych testowych zebranych dla osób w różnym wieku:

Tablica 17.2. Porównanie skuteczności klasyfikatora w zależności od osoby

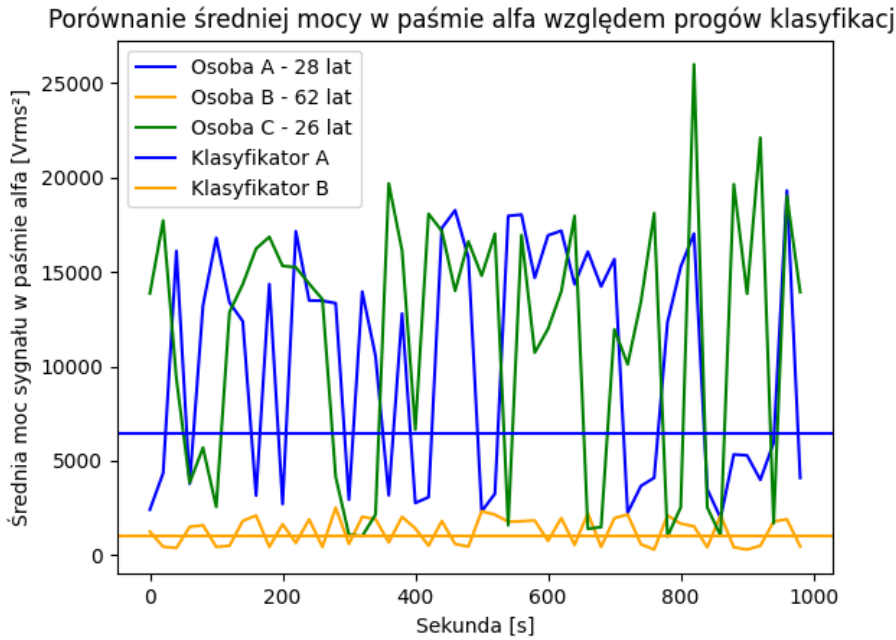
Skuteczność	**Klasyfikator A**	**Klasyfikator B**
A	100%	58%
B	44%	100%
C	96%	72%

W celu zrozumienia różnic wynikających z uzyskanych wyników warto się przyjrzeć bliżej średniej mocy sygnału w paśmie alfa zarejestrowanej podczas badania, w którym zbierano dane testowe. Moduł wartości wyjściowych z transformacji

Fouriera podniesiony do kwadratu reprezentuje spektrum mocy (ang. *Power Spectrum*) sygnału co przedstawia poniższy wzór [37]:

$$PS(f) = |X(f)|^2 \tag{17.6}$$

Oznacza on średnią kwadratową napięcia elektrycznego, którą oznaczamy symbolem - $Vrms^2$ [8, 14, 20, 25]. Poniżej na rys. 17.2 możemy porównać ustalone progi klasyfikacji tj. Klasyfikator A = 6429, Klasyfikator B = 1053, a także średnią moc sygnału w paśmie alfa w zależności od wieku badanego:



Rysunek 17.2. Porównanie średniej mocy sygnału w paśmie alfa i progów klasyfikacji w zależności od wieku osoby badanej

Powyższy rysunek 17.2 doskonale obrazuje różnice w rejestrowanej mocy w paśmie alfa u różnych osób w zależności od wieku. Osoba C oznaczona na wykresie zielonym kolorem jest faktycznie młodsza o dwa lata względem osoby A i trzydzieści sześć lat względem osoby B. Obserwacja dynamiki rytmu alfa w zależności od wieku przedstawiona w niniejszej pracy jest zgodna z literaturą [29].

Warto tutaj zwrócić uwagę, że wpływ wieku na wynik klasyfikacji jest dodatkowym aspektem badanego zjawiska. Wynik uzyskany przy klasyfikacji danych testowych osoby C, jest zanieczyszczony ze względu na niezachowanie procedury

badawczej podczas eksperymentów. Konkretnie osoba C podczas badania miała problem ze wstrzymaniem nadmiernego mrugania wynikającego ze zmęczenia oczu po całym dniu. Postępowanie według procedury wymusiło nadmierne łzawienie oczu, a w wyniku śmiech i zmieszanie osoby badanej przez całokształt zaistniałej sytuacji. Jest to istotna uwaga z perspektywy osoby zainteresowanej prowadzeniem tego typu eksperymentów, ponieważ praca z drugim człowiekiem może odbiegać od oczekiwań stawianych przez system komputerowy i suchą metodologię pracy z urządzeniami. Ostatecznie zebrane dane w przypadku osoby C posiadały nieprawdziwe etykiety. Zostało to odnotowane przez osobę badającą i najprawdopodobniej zarejestrowane w ramach systemu, co można przypuszczać analizując rysunek 17.2. Konkretnie w okolicach 100 sekundy, a także w okolicach 400 sekundy średnia moc w paśmie alfa osoby C nie przekracza progu klasyfikacji Klasyfikatora A, dlatego w tablicy 17.2 przypadek ten posiada 96 procentową skuteczność. Można w tym miejscu zadać pytanie czy brakująca moc potrzebna do poprawnego zaklasyfikowania aktywności wynika faktycznie z zarejestrowanych i opisanych problemów. Jednoznaczna odpowiedź na to pytanie nie jest możliwa na etapie pisania niniejszej pracy, wymaga ona dalszych eksperymentów, w szczególności wielokrotnego powtórzenia procedury badawczej zarówno w przypadku wybranej osoby jak i pomiędzy osobami. Podobnie na podstawie tablicy 17.2 i rysunku 17.2 możemy zadać pytanie, co jest przyczyną spadku skuteczności klasyfikatora trenowanego na danych osoby B, a wykorzystanego do testów skuteczności na danych zebranych przy pomocy osoby C? Zebrane wartości liczbowe pozwalają stwierdzić, że próg klasyfikacji w Klasyfikatorze B jest znacząco poniżej większości wartości zarejestrowanych dla osoby C, stąd poprawnie klasyfikowane są wyłącznie próbki reprezentujące stan spoczynku w obu zbiorach danych. Biorąc pod uwagę odkrycia związane z zachodzącymi w mózgu zmianami na skutek starzenia, w tym zmniejszający się jego rozmiar i zmiany unaczynienia [34], można przypuszczać, że fundamentalną przyczyną obserwowanego spadku skuteczności klasyfikacji jest średni spadek mocy sygnału w paśmie alfa na skutek zachodzących zmian, dlatego próg klasyfikacji aktywności kory wzrokowej wyznaczony dla osoby B, która jest starsza od osoby C, jest niemiarodajny pomiędzy osobami o znacząco dużej różnicy wieku.

17.9 Podsumowanie

Niniejsza praca przedstawia czytelnikowi krok po kroku proces konstrukcji i oceny systemu wykorzystującego dedykowane urządzenie EEG do rejestracji sygnału pochodzącego z mózgu, reprezentacji danych w formie cyfrowej, ich analizy i klasyfikacji. Jakość prezentowanych treści została poparta wynikami numerycznymi własnych badań z wykorzystaniem opisanej architektury w praktyce, a także nawiązaniami do pozycji w literaturze szczegółowo prezentujących poruszane zagadnienia.

Ponadto wiedza zawarta w tej pracy dotyczy również kontekstu historycznego, prawidłowego uruchomienia dedykowanego urządzenia będącego otwartym rozwiązaniem, umiejscowienia elektrod, integracji z urządzeniem, przetwarzania

sygnałów, ekstrakcji cech, automatycznej klasyfikacji danych EEG, metodologii oceny uzyskiwanych wyników i prezentowania rezultatów w sposób zgodny z literaturą.

Podstawowym problemem osoby rozpoczynającej swoją przygodę z akwizycją i przetwarzaniem danych EEG za pomocą dedykowanego urządzenia jest ustalenie punktu odniesienia. Najistotniejszą kwestią jest powtarzalność wyników uzyskiwanych przez wybranego uczestnika, na którym należałoby powtórzyć wielokrotnie całą procedurę jak i również pomiędzy różnymi osobami. Warto rozważyć różnego typu błędy pomiaru, które mogą pojawić się z różnych przyczyn np. przy lokalizacji pozycji elektrod. Jednakże celem niniejszej pracy jest umożliwienie dzięki postępowaniu zgodnie z opisaną procedurą weryfikacji elementów składowych systemu i połączeń między nimi przed rozpoczęciem bardziej zaawansowanych badań uwzględniających znacznie większą od strony statystycznej liczbę informacji potrzebnych do konstruktywnego wnioskowania odnośnie prowadzonych obserwacji zjawisk jeszcze nieznanych. Dlatego w niniejszej pracy nie zostały przedstawione wyniki w kontekście zbierania danych dla pojedynczego pacjenta przy wielokrotnym umieszczeniu elektrod lub analiza błędów związanych z pomiarem, w tym dopuszczalnego błędu lokalizacji elektrod. Opisana w niniejszej pracy metodologia została oparta o znane już od blisko stu lat zjawisko, stąd celem nie jest zapewnienie prawdziwości obserwowanego zjawiska od strony statystycznej i metrologicznej, a jedynie wskazanie znacząco uproszczonego podejścia do wyznaczenia punktu początkowego do dalszej pracy badawczej z wybranym urządzeniem.

Bibliografia

1. Brain Waves - an overview | ScienceDirect Topics, `https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/brain-waves`.
2. Cyton + Daisy Biosensing Boards (16-Channels), `https://shop.openbci.com/products/cyton-daisy-biosensing-boards-16-channel`.
3. Cyton Biosensing Board (8-channels), `https://shop.openbci.com/products/cyton-biosensing-board-8-channel`.
4. Data Format Description — BrainFlow documentation, `https://brainflow.readthedocs.io/en/stable/DataFormatDesc.html#generic-format-description`.
5. Discrete Fourier Transform (numpy.fft) — NumPy v1.20 Manual, `https://numpy.org/doc/stable/reference/routines.fft.html#rfb1dc64dd6a5-ct`.
6. Gold Cup Electrodes, `https://shop.openbci.com/products/openbci-gold-cup-electrodes`.
7. How steady is Cyton's sampling rate? [resolved], `https://openbci.com/forum/index.php?p=/discussion/2329/how-steady-is-cytons-sampling-rate-resolved`.

8. Peak vs. Average vs. RMS Voltage, https://testguy.net/content/270-Peak-vs-Average-vs-RMS-Voltage.
9. sklearn.svm.SVC — scikit-learn 0.24.2 documentation, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.
10. Supported Boards — BrainFlow documentation, https://brainflow.readthedocs.io/en/stable/SupportedBoards.html#cyton.
11. Ten20 Conductive Paste 8oz Jar, https://shop.openbci.com/products/ten20-conductive-paste-8oz-jar.
12. Thales of Miletus | Biography & Facts | Britannica, https://www.britannica.com/biography/Thales-of-Miletus.
13. Kora wzrokowa (Dec 2016), https://pl.wikipedia.org/w/index.php?title=Kora_wzrokowa&oldid=47836101, page Version ID: 47836101.
14. Średnia kwadratowa (Aug 2018), https://pl.wikipedia.org/w/index.php?title=%C5%9Arednia_kwadratowa&oldid=54372959, page Version ID: 54372959.
15. Kość potyliczna człowieka (Apr 2019), https://pl.wikipedia.org/w/index.php?title=Ko%C5%9B%C4%87_potyliczna_cz%C5%82owieka&oldid=56361812, page Version ID: 56361812.
16. Płat potyliczny (May 2020), https://pl.wikipedia.org/w/index.php?title=P%C5%82at_potyliczny&oldid=59682767, page Version ID: 59682767.
17. 10–20 system (EEG) (Jan 2021), https://en.wikipedia.org/w/index.php?title=10%E2%80%9320_system_(EEG)&oldid=1003622197, page Version ID: 1003622197.
18. Fale mózgowe (Apr 2021), https://pl.wikipedia.org/w/index.php?title=Fale_m%C3%B3zgowe&oldid=62944478, page Version ID: 62944478.
19. Nyquist frequency (Apr 2021), https://en.wikipedia.org/w/index.php?title=Nyquist_frequency&oldid=1016916999, page Version ID: 1016916999.
20. Root mean square (May 2021), https://en.wikipedia.org/w/index.php?title=Root_mean_square&oldid=1022556694, page Version ID: 1022556694.
21. Al-Fahoum, A.S., Al-Fraihat, A.A.: Methods of EEG Signal Features Extraction Using Linear Analysis in Frequency and Time-Frequency Domains. ISRN Neuroscience **2014** (Feb 2014). https://doi.org/10.1155/2014/730218, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4045570/.
22. Barry, R.J., Clarke, A.R., Johnstone, S.J., Magee, C.A., Rushby, J.A.: EEG differences between eyes-closed and eyes-open resting conditions. Clinical Neurophysiology **118**(12), 2765–2773 (Dec 2007). https://doi.org/10.1016/j.clinph.2007.07.028, https://www.sciencedirect.com/science/article/pii/S1388245707004002.
23. Barry, R.J., De Blasio, F.M.: EEG differences between eyes-closed and eyes-open resting remain in healthy ageing. Biological Psychology **129**, 293–304 (Oct 2017). https://doi.org/10.1016/j.biopsycho.2017.09.010.

24. Buskila, Y., Bellot-Saez, A., Morley, J.W.: Generating Brain Waves, the Power of Astrocytes. Frontiers in Neuroscience **13** (2019). https://doi.org/10.3389/fnins.2019.01125, https://www.frontiersin.org/articles/10.3389/fnins.2019.01125/full#B18, publisher: Frontiers.
25. Cerna, M., Harvey, A.F.: The Fundamentals of FFT-Based Signal Analysis and Measurement p. 20.
26. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3), 273–297 (Sep 1995). https://doi.org/10.1007/BF00994018, http://link.springer.com/10.1007/BF00994018.
27. Dondurur, D.: Chapter 2 - Marine Seismic Data Acquisition. In: Dondurur, D. (ed.) Acquisition and Processing of Marine Seismic Data, pp. 37–169. Elsevier (Jan 2018). https://doi.org/10.1016/B978-0-12-811490-2.00002-5, https://www.sciencedirect.com/science/article/pii/B9780128114902000025.
28. Kropotov, J.D.: Chapter 8 - Methods of Analysis of Background EEG. In: Kropotov, J.D. (ed.) Quantitative EEG, Event-Related Potentials and Neurotherapy, pp. 116–160. Academic Press, San Diego (Jan 2009). https://doi.org/10.1016/B978-0-12-374512-5.00008-6, https://www.sciencedirect.com/science/article/pii/B9780123745125000086.
29. Kropotov, J.D.: Chapter 2.2 - Alpha Rhythms. In: Kropotov, J.D. (ed.) Functional Neuromarkers for Psychiatry, pp. 89–105. Academic Press, San Diego (Jan 2016). https://doi.org/10.1016/B978-0-12-410513-3.00008-5, https://www.sciencedirect.com/science/article/pii/B9780124105133000085.
30. Louis, E.K.S., Frey, L.C., Britton, J.W., Frey, L.C., Hopp, J.L., Korb, P., Koubeissi, M.Z., Lievens, W.E., Pestana-Knight, E.M., Louis, E.K.S.: Introduction. American Epilepsy Society (2016), https://www.ncbi.nlm.nih.gov/books/NBK390346/, publication Title: Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants [Internet].
31. Onajite, E.: Chapter 8 - Understanding Sample Data. In: Onajite, E. (ed.) Seismic Data Analysis Techniques in Hydrocarbon Exploration, pp. 105–115. Elsevier, Oxford (Jan 2014). https://doi.org/10.1016/B978-0-12-420023-4.00008-3, https://www.sciencedirect.com/science/article/pii/B9780124200234000083.
32. Osgood, B.: Lecture Notes for EE 261 The Fourier Transform and its Applications, https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf.
33. Oshana, R.: 4 - Overview of Digital Signal Processing Algorithms. In: Oshana, R. (ed.) DSP Software Development Techniques for Embedded and Real-Time Systems, pp. 59–121. Embedded Technology, Newnes, Burlington (Jan 2006). https://doi.org/10.1016/B978-075067759-2/50006-5, https://www.sciencedirect.com/science/article/pii/B9780750677592500065.
34. Peters, R.: Ageing and the brain. Postgraduate Medical Journal **82**(964), 84–88 (Feb 2006). https://doi.org/10.1136/pgmj.2005.036665, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2596698/.

35. Schomer, D.L., Silva, F.H.L.d.: Niedermeyer's Electroencephalography: Basic Principles, Clinical Applications, and Related Fields. Oxford University Press (2018), google-Books-ID: y3o7DwAAQBAJ.
36. Semmlow, J.: Chapter 3 - Fourier Transform: Introduction. In: Semmlow, J. (ed.) Signals and Systems for Bioengineers (Second Edition), pp. 81–129. Biomedical Engineering, Academic Press, Boston (Jan 2012). https://doi.org/10.1016/B978-0-12-384982-3.00003-1, https://www.sciencedirect.com/science/article/pii/B9780123849823000031.
37. Semmlow, J.: Chapter 4 - The Fourier Transform and Power Spectrum: Implications and Applications. In: Semmlow, J. (ed.) Signals and Systems for Bioengineers (Second Edition), pp. 131–165. Biomedical Engineering, Academic Press, Boston (Jan 2012). https://doi.org/10.1016/B978-0-12-384982-3.00004-3, https://www.sciencedirect.com/science/article/pii/B9780123849823000043.
38. Shannon, C.E.: Communication in the Presence of Noise. PROCEEDINGS OF THE IEEE **86**(2), 11 (1998).
39. Tan, B., Kong, X., Yang, P., Jin, Z., Li, L.: The Difference of Brain Functional Connectivity between Eyes-Closed and Eyes-Open Using Graph Theoretical Analysis. Computational and Mathematical Methods in Medicine **2013**, e976365 (Apr 2013). https://doi.org/10.1155/2013/976365, https://www.hindawi.com/journals/cmmm/2013/976365/, publisher: Hindawi.

18. Paradoks decyzyjny – racjonalne i intuicyjne podejmowanie decyzji

Henryk Krawczyk

Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska, Gdańsk
hkrawk@eti.pg.edu.pl

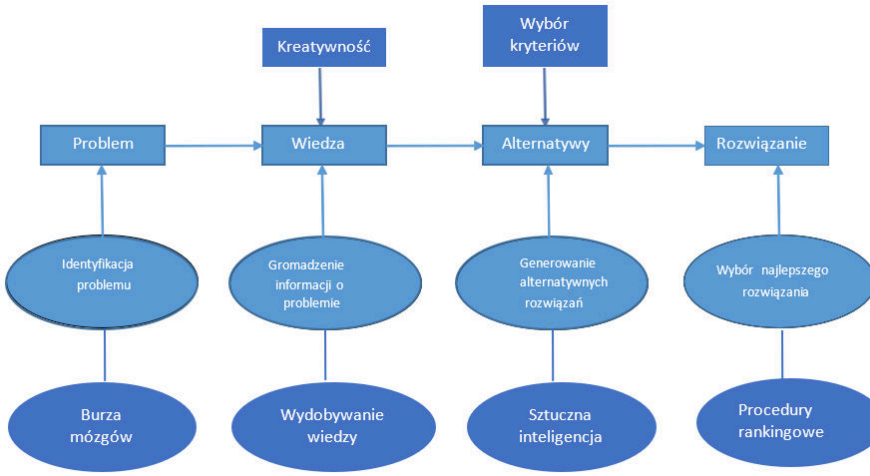
Streszczenie
W pracy scharakteryzowano poszczególne etapy działań prowadzące do znajdowania najlepszych rozwiązań dla rozpatrywanego problemu. Zwrócono uwagę na paradoks decyzyjny który wskazuje, że mądre rozwiązanie problemu wymaga zarówno racjonalnego, jak i intuicyjnego podejścia. Na przykładzie sortowania obrazów zaprezentowano niezależnie oba podejścia podkreślając potrzebę ich wzajemnego uzupełniania się. Podkreślono trudność budowy algorytmów, które będą uwzględniać jednocześnie te dwa odmienne podejścia. Prowadzi to do znalezienia rozwiązań nieoptymalnych, ale satysfakcjonujących wszystkich zainteresowanych. Pozwoli też na stopniowe wyjaśnianie ludzkiej mądrości i jej roli przy rozwiązywaniu życiowych problemów.

Słowa kluczowe: Rozwiązywanie problemów, metody racjonalne i intuicyjne, paradoks decyzyjny, szeregowanie rysunków, istota mądrości

18.1 Wprowadzenie

Człowiek w swoim życiu spotyka się z różnymi problemami, które wymagają rozwiązania, by wyeliminować ewentualne trudności bądź osiągnąć zamierzony cel. Spektrum spotykanych problemów jest bardzo duże: od różnych spraw osobistych, poprzez sprostanie wymaganiom zawodowym, do udziału w pilnych wyzwaniach cywilizacji. Część z tych problemów człowiek rozwiązuje samodzielnie, korzystając z własnych doświadczeń; przy innych korzysta z różnego typu doradców lub zleca ich rozwiązanie odpowiednim firmom konsultacyjnym; a jeszcze inne, zorientowane zawodowo lub społecznie, podejmuje w zespołach, które już funkcjonują, bądź które zostały do tego celu powołane.

Na rys. 18.1 przedstawiono ogólną koncepcję znajdowania rozwiązania dla zadanego problemu. Z reguły proces decyzyjny obejmuje logicznie powiązaną grupę operacji myślowych, uporządkowaną w odpowiedniej kolejności, umożliwiającą ocenę sytuacji decyzyjnej i wybór najkorzystniejszego wariantu rozwiązania. Oczywiste, że problem nie musi być precyzyjnie zdefiniowany, może się albo ujawnić w pewien niezbyt oczywisty sposób, albo trzeba go dopiero zidentyfikować analizując bieżący przebieg spraw. Tak więc pierwszym etapem rozwiązywania problemu jest jego uświadomienie bądź wyłuskanie z lawiny różnego typu



Rysunek 18.1. Etapy znajdowania rozwiązania dla zadanego problemu

dostępnej informacji. Przyczyny inicjujące proces decyzyjny można podzielić na: reaktywne, kiedy odnoszą się do zaistniałych już zdarzeń oraz proaktywne, kiedy należy podjąć działania wyprzedzające kreujące przyszłe wydarzenia. Tworzenie pomysłów rozwiązania problemu jest zadaniem niezwykle trudnym i możliwym do wykonania tylko wtedy, gdy problem decyzyjny jest sformułowany prawidłowo i zgromadzona jest niezbędna ilość rzetelnych informacji. Podstawą generowania pomysłów są: różnorodność wiedzy, intuicja i twórcze myślenie [4]. Często w obu przypadkach owocna jest burza mózgów, wśród wszystkich, którym zlecono jego identyfikację bądź którzy są aktualnie zaniepokojeni rodzajem dostarczanej informacji. Istotne jest, by problem został w pewien sposób zauważony, tak by można było zająć się nim szczegółowo.

Następnym etapem postępowania jest zgromadzenie jak największej informacji o już zidentyfikowanym problemie. Należy rozpatrzyć, czy tego typu problemy pojawiły się już w przeszłości, jakie były tego konsekwencje bądź jako metody wykorzystano do ich rozwiązania. Chodzi o informacje na podstawie których można wydobyć wiedzę, która lokuje ten problem w określonej kategorii oraz wskazuje na możliwości jego rozwiązania. Zakłada się, że problemy mogą być dostatecznie dobrze poznane oraz ustrukturalizowane, a ich rozwiązanie jest możliwe przy wykorzystaniu odpowiednich modeli matematycznych oraz wynikających z nich algorytmów [14]. Przykładem tego typu problemów może być wybór trasy przemieszczenia z miejscowości wyjściowej do docelowej, przy założeniu znajomości wszystkich możliwych dróg. Wówczas możemy podać różne algorytmy wyznaczania optymalnej trasy ze względu na przyjęte kryterium (najszybszy czas dotarcia czy najmniejsze zużycie paliwa). Inny, przeciwstawny rodzaj problemów stanowią problemy nieustrukturalizowane, które dają się ująć tylko w sposób jakościowy, w postaci opisu słownego i dotyczą zależności, które

trudno zmierzyć [10]. Przykładem może być wyznaczanie kierunków badań w danej dyscyplinie naukowej, przewidywanie pogody, czy radzenie z epidemią, gdzie wiele nieznanych zależności utrudnia rozwiązanie problemu. Jednak z uwagi na postęp w metodach obliczeniowych, zapewnienie dostępu do różnych danych oraz wykorzystanie metod wydobywania wiedzy, proponowane rozwiązania stają się coraz bardziej wiarygodne. Generalnie ten etap działań wymaga dużej wyobraźni oraz kreatywności, by uwzględniać i przeanalizować dostępne fakty oraz odpowiednio skojarzyć je ze sobą. Warto podkreślić, że problemy życiowe są z reguły nieustrukturalizowane.

Kolejnym krokiem jest poszukiwanie możliwych rozwiązań z wykorzystaniem różnych metod postępowania. Tutaj oprócz wysokiej kreatywności jest niezbędna odpowiednia wiedza oraz krytyczna ocena adekwatności wykorzystywanych modeli do rozpatrywanych sytuacji. Najbardziej obiecujące rozwiązania powinny zostać zaimplementowane i ocenione. Ważne jest więc uzmysłowienie właściwych kryteriów takiej oceny oraz procedur, które ją umożliwią. Na ogół proponuje się wiele różnych rozwiązań zależnych od kontekstu zdarzeń oraz przydatnych przy konkretnych ograniczeniach [1]. Bardziej popularne stają się metody sztucznej inteligencji.

Ostatnim etapem poszukiwania najlepszego rozwiązania przyjętego problemu, jest adekwatny wybór rozwiązania przy wykorzystaniu przygotowanych procedur oceny. Decyzje te mogą być podejmowane w sytuacji pewności, kiedy do dyspozycji są dokładne i wiarygodne informacje, na których można oprzeć swoje finalne działania, a analiza decyzyjna sprowadza się do sformułowania kilku opcji i wyboru tej najlepszej. Z sytuacją niepewności mamy do czynienia wówczas, gdy występuje duża złożoność problemu, a dynamika zmian w otoczeniu uniemożliwia oszacowanie możliwego rozwoju sytuacji. Wówczas decydent nie może określić, jakie czynniki będą oddziaływać na sytuację decyzyjną, a tym samym jakie będą skutki podjętych decyzji. Należy też oszacować ryzyko związane z konkretnym działaniem przewidując odpowiednie procedury zredukowania jego wpływu. Często można posłużyć się odpowiednią metodą rankingową, która może, ale nie musi zapewniać jednoznacznej decyzji wyboru. Stąd potrzeba dalszego krytycznego spojrzenia na dotychczasowe rozważania, co może oznaczać powracanie do poprzednich etapów rozumowania i postępowania. Dopiero metoda stopniowego udoskonalania rozwiązań [8] może zakończyć się sukcesem znalezienia satysfakcjonującego rozwiązania.

Przedstawiony schemat na rys. 18.1 odpowiada normatywnym modelom podejmowania decyzji, które opierają się na założeniu, że człowiek jest istotą racjonalną, a postępowanie zgodnie z odpowiednim algorytmem doprowadzi do znalezienia rozwiązania najlepszego w kontekście przyjętego wcześniej kryterium. Cechuje je dedukcyjna metoda postępowania badawczego, polegająca na przejściu od ogólnych (abstrakcyjnych) założeń, poprzez formułowanie szczegółowych prawidłowości, aż do konkretnych sposobów postępowania. Można stwierdzić, że przy rozwiązywaniu różnych problemów życiowych świadomie czy nieświadomie decydujemy podobnie o wielu sprawach, działając często jednak w sposób bardziej uproszczony niż to podano na rys. 18.1. zaóważalnaas wykorzystuje się

alternatywne, nazywane deskryptywnymi (opisowo-wyjaśniającymi) lub behawioralnymi modele podejmowania decyzji. Przykładem jest model suboptymalizacji [15], czy heurystyczne modele wspierania twórczego myślenia [19]. Heurystyki to metody rozumowania, którymi posługujemy się nieświadomie, które ignorują część informacji oraz zaawansowane metody wnioskowania po to, aby proces decydowania był szybki i absorbował mniej wysiłku. Wyróżnia się przy tym następujące charakterystyki: *dostępności* (ang. *availability*) – gdzie decyzje najczęściej podejmowane są na podstawie informacji, które są najłatwiej dostępne w pamięci; *zakotwiczenia* (ang. *anchoring*) – polega na „zakotwiczeniu" się na jakiejś informacji, a następnie zmodyfikowaniu jej i dostosowaniu się do niej, przy podejmowaniu decyzji, *reprezentatywności* (ang. *representativeness*), gdy wnioskowanie odnosi się do częściowego podobieństwa problemów, do pewnego reprezentatywnego przypadku, bez szacowania statystycznych szans jego wystąpienia. Najczęściej tego typu podejścia wykorzystuje się w ekonomii behawioralnej [16], która na podstawie analizy rzeczywistych możliwości i zachowań ludzi, stara się wyjaśnić, jak przebiegają procesy decyzyjne, odnosząc je do zachowania się konsumentów, przedsiębiorstw, a w konsekwencji rynków oraz gospodarki.

Dlatego w pracy rozpatrzymy dwa podstawowe przypadki ludzkich zachowań, gdzie ich decyzje wynikają z działań racjonalnych bądź czysto intuicyjnych. Zilustrujemy je na przykładzie oceny zestawu prac wykonanych przez kandydatów na studia architektoniczne z zakresu malarstwa. Wydaje się, że ten akademicki przykład wymaga takiego dualnego podejścia. Zakładamy więc, że przygotowane rysunki będą oceniane i uszeregowane według ich jakości artystycznej, najpierw posługując się wybraną metodą intuicyjną, a następnie metodą racjonalną. Tą pierwszą metodą posłużą się eksperci (malarze – nauczyciele akademiccy), druga zaś ma być zaprojektowana jako komputerowa aplikacja. Okaże się, że budowa takiej komputerowej aplikacji wymaga też pewnych działań intuicyjnych, gdyż w przeciwnym przypadku ocena racjonalna zbyt mocno odbiega od oceny ekspertów. To w pewnym sensie potwierdza istnienie paradoksu decyzyjnego [13].

W rozdziale 2 opisano paradoks decyzyjny, abyby następnie przeanalizować dwa eksperymenty związane z podejściem intuicyjnym (rozdz. 3) i racjonalnym (rozdz. 4). We wnioskach końcowych (rozdz. 5) zasygnalizowano dalsze kierunki badań dotyczących podejmowania mądrych decyzji.

18.2 Paradoks decyzyjny

Wszystkie procesy poznawcze są realizowane w mózgu człowieka. Badania mózgu, dzięki postępowi cyfryzacji są już daleko zaawansowane [9]. Nie wiadomo jednak, który element mózgu odpowiada za poprawność podejmowania decyzji. Jest wysoce prawdopodobne, że odbywa się to dzięki współpracy różnych elementów, gdzie istotną rolę odgrywa hipokamp. Trzeba jednak podkreślić, że pewien wpływ mają też inne elementy, odpowiedzialne za koncentrację, percepcję czy empatię. Stanowią one dodatkowe źródła informacji, również istotne przy podejmowaniu różnych decyzji. Złożona struktura i nieznane działanie ta-

kich elementów stanowią poważną przeszkodę w zrozumieniu procesów poznawczych i ich wpływu na zachowania się człowieka. Dodatkowo należałoby jeszcze uwzględnić różnorodność rozwiązywanych problemów, różnorodność uwarunkowań zewnętrznych (konteksty), czy różnorodność ludzkich postaw. To sprawia, że w tych samych okolicznościach, dla tego samego problemu, zarówno podejmowane decyzje przez różne osoby, jak i skutki tych decyzji mogą być różne. Trzeba założyć, że każdy proces decyzyjny jest indywidualnym procesem rozumowania uwzględniającym wartości, preferencje i przekonania osoby decydującej i prowadzi do ostatecznego wyboru, który może, ale nie musi skłaniać do podjęcia odpowiednich działań. Często dużą rolę odgrywają emocje [22], które utrudniają zrozumienie podejmowanych decyzji.

Metody wykorzystywane w procesie podejmowania decyzji zależą też od charakteru i kategorii problemu którego dotyczą. Te dobrze strukturalizowane problemy są najczęściej rozwiązywane przy wykorzystywaniu metod z obszaru badań operacyjnych, stanowiących elegancki, ale złożony aparat matematyczny. W praktyce najczęściej wykorzystuje się programowanie liniowe, programowanie dynamiczne, programowanie całkowitoliczbowe, a także teorię gier, bayesowską metodę podejmowania decyzji, a także systemy masowej obsługi oraz analizę statystyczną [20]. Chociaż możliwości zastosowania tych metod nieustannie się zwiększają, to w przypadku zagadnień o charakterze wyłącznie jakościowym, powyższe narzędzia okazują się z reguły zawodne. W takich przypadkach wykorzystywane mogą być odpowiednie metody heurystyczne [3], a ostatnio również i sztuczna inteligencja [12]. Heurystyki, jako praktyczne metody postępowania, oparte są w dużej mierze na intuicyjnej ocenie rzeczywistości, są regułami niepewnymi, niedającymi gwarancji uzyskania poprawnych albo optymalnych wyników. Do najpopularniejszych metod heurystycznych należą: burza mózgów (doskonalenie decyzji grupowych), synektyka (poszukiwaniu analogii w myśleniu twórczym), myślenie lateralne (przeformułowanie problemu dające szansę innego rozwiązania) i metoda delficka (wykorzystanie wiedzy ekspertów).

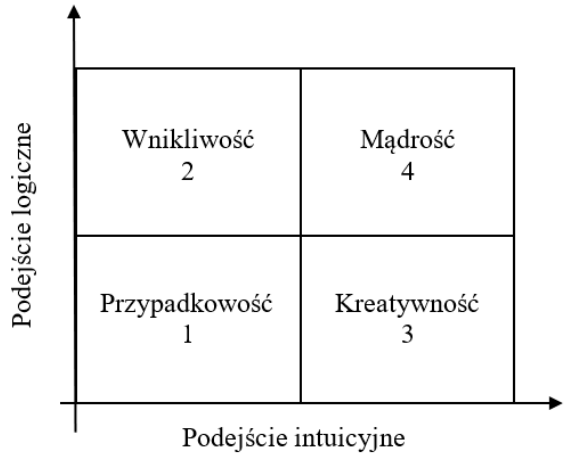
W życiu codziennym i zawodowym, podejmowanie decyzji wymaga na ogół uwzględnienia wielu sprzecznych kryteriów. Na przykład w branży usługowej trudno pogodzić wysoką satysfakcję klienta z niskim kosztem świadczenia usług. Podobnie niską cenę towaru z jego wysoką jakością, czy wysokie zyski giełdowe z niskim ryzykiem ich pozyskania. Sprzeczności tego typu generują bardzo złożone kwestie i dotyczą często wielu osób, które są głęboko dotknięte skutkami podjętych niewłaściwych decyzji. Często odpowiednie strukturyzowanie takich problemów prowadzi do bardziej świadomych i lepszych decyzji. Wiąże się to z rozwojem, w ramach badań operacyjnych, nowej (wielokryterialnej) dyscypliny podejmowania decyzji. Rozróżniono przy tym dwa podejścia: MCDM (podejmowanie decyzji na podstawie wielu kryteriów) oraz MCDA (analiza decyzji na podstawie wielu kryteriów) [17]. Opracowano różnorodne metody i odpowiadające im rozwiązania, z których wiele zostało wdrożonych jako specjalistyczne oprogramowanie służące do podejmowania decyzji w szeregu dyscyplin, w polityce i biznesie, w środowisku naturalnym, czy przy wykorzystaniu energii lub planowaniu strategii rozwojowych.

Formalnie paradoks decyzyjny [18] wynika z obserwacji, że różne metody podejmowania decyzji, zarówno normatywne, jak i opisowe, dają różne wyniki, choć dotyczą tego samego problemu decyzyjnego i tych samych danych. W szczególności zauważono, że wybór najlepszego kryterium spośród: metody sumy ważonej (WSM), metody produktu ważonego (WPM) oraz dwóch wariantów procesu hierarchii analitycznej (AHP) nie jest jednoznaczny. I tak, gdy stosuje się jedną metodę, (powiedzmy metodę X, która jest jedną z czterech powyższych metod), to okazuje się, że inna metoda jest wówczas najlepsza (np. metoda Y). Taka tendencja dotyczy wszystkich kombinacji wyżej przedstawionych metod. Co więcej wykazano, że inne metody np. metoda ELECTRE (przewyższania stopniem, ang. *outranking*) czy metoda SIR (rankingu wyższości i niższości) również zachowują te tendencje.

Paradoks decyzyjny można odnieść [6] do dwóch podejść: podejścia logicznego (*logos*) i podejścia intuicyjnego (*mythos*). Pierwsze z nich oznacza konstruktywną formę wypracowywania decyzji wykorzystującą różnego typu rozumowanie logiczne. Z kolei podejście intuicyjne jest zaprzeczeniem podejścia logicznego bowiem wykorzystuje jedynie własne skojarzenia czy wyczucia, umożliwiające ukryte dochodzenie do sedna sprawy. Ujmując problem skrótowo, rozpatrujemy dwa skrajne przypadki podejmowania decyzji: jeden stawia na rozum, drugi na intuicję. Paradoksem jest, że te całkiem odmienne postępowania mogą jednak wzajemnie się uzupełniać.

Już Cyceron stwierdził, *że żyć to myśleć*. Z kolei Antoine de Saint Exupéry zaprzeczył temu podając, że *logika zabija życie*. Zestawienie tych dwóch cytatów jednoznacznie podkreśla że decydowanie racjonalne i intuicyjne nie mają wiele wspólnego. Jednak w rzeczywistości jest odwrotnie. Tak jak formalny paradoks decyzyjny zasygnalizowany powyżej ukazuje z jednej strony niemożliwość wyboru najlepszego rozwiązania (zawodzą metody oceny), z drugiej zaś konieczność wyboru z pośród dostępnych rozwiązań. To co z pozoru wydaje się niemożliwe, jest jednak w pewnym sensie możliwe i stanowi rozwiązanie satysfakcjonujące. Na rys. 18.2 zilustrowano to graficznie, przedstawiając dwa przeciwstawne podejścia na różnych osiach układu współrzędnych. Założono przy tym niskie i wysokie zaangażowanie intuicji oraz niskie i wysokie wykorzystanie metod racjonalnych. W ten sposób wyróżniono cztery różne pola (na wykresie prostokąty), które odzwierciedlają różne powiązania intuicyjnego i racjonalnego (logicznego) podejścia. Postępowanie o niskiej logice i intuicji (pole 1) prowadzi jedynie do przypadkowych decyzji wyboru z pośród satysfakcjonujących rozwiązań. Z kolei przewaga logicznego (analitycznego) podejścia związana jest z wnikliwością analizy sytuacji, która staje się wówczas bardziej systematyczna (pole 2). Przewaga intuicyjnego podejścia daje szansę wykorzystania wyższej kreatywności (pole 3) oraz inteligencji, gdyż umożliwia wygenerowanie większej liczby różnorodnych przypadków decyzyjnych. Dopiero powiązanie analitycznego i intuicyjnego podejścia (pole 4) sprawia, że podejmowane decyzje mogą być najodpowiedniejsze, co często właśnie kojarzy się ludziom z mądrością.

Takie ujęcie problemu decyzyjnego jest zasadne, ponieważ człowiek w praktyce niejednokrotnie zmuszony jest podejmować decyzje, mając tylko fragmen-



Rysunek 18.2. Ilustracja paradoksu decyzyjnego

taryczną wiedzę o dostępnych alternatywach i ich możliwych konsekwencjach. Nawet posiadanie wszystkich niezbędnych informacji nie daje zresztą gwarancji dokonania najlepszego wyboru, ponieważ umysł ludzki ma ograniczoną zdolność analizowania, rozumienia i zapamiętywania. Te bariery, wynikające z ludzkich możliwości obliczeniowych, zmuszają często do osiągania jedynie satysfakcjonującego rozwiązania. Co więcej, każdy człowiek bardzo często dąży do realizacji nie jednego, lecz wielu celów jednocześnie, które wcale nie muszą być ze sobą zbieżne, a jego ograniczone zdolności intelektualne uniemożliwiają dokonanie wszechstronnych analiz. Dlatego jego uwagę przykuwają rozwiązania najłatwiej dostępne, np. dotyczące pierwszego znalezionego wariantu rozwiązania, spełniającego określone kryteria, wówczas rezygnuje się z dalszych poszukiwań, mimo że mogłyby one doprowadzić do jeszcze lepszych rezultatów. Takie postępowanie jest paradoksem, bardzo istotnym również z punktu widzenia podejmowania decyzji.

Poza tym zbiór wszystkich możliwych rozwiązań nie jest znany podejmującemu decyzje w jednym momencie czasu, a różne możliwe nowe warianty uświadamia sobie po kolei z upływem czasu. Należy więc zadecydować, w którym momencie należy przerwać proces poszukiwania rozwiązań i podjąć ostateczną decyzję. Nie ma oczywiście pewności, że w zbiorze opcji rozwiązań wziętych pod uwagę jest ta optymalna oraz że dalsze poszukiwania nie wyłonią lepszej. Trzeba po prostu ustalić, jaki poziom zaspokojenia potrzeby uznaje się za satysfakcjonujący i przerwać poszukiwania, gdy znajdzie się rozwiązanie spełniające te wymagania. W sytuacji, gdy przy stosunkowo niewielkim wysiłku można znaleźć lepsze rozwiązanie, to motywacje poszukującego rosną, gdy zaś poszukiwania nie przynoszą oczekiwanych rezultatów, jego motywacje poszukiwań będą maleć [15]. Taka reguła wyboru nazywana jest „poszukiwaniem satysfakcjonującego

rozwiązania" (ang. *satisfying*), stanowi kwintesencję podejścia Simona, laureata nagrody Nobla, do procesu decyzyjnego.

Zgodnie z innymi opiniami [21] podejmowanie decyzji w organizacjach nie jest racjonalne, gdyż organizacje są słabo sformalizowane. Często funkcjonują na zasadach niejasnych, niepewnych i podlegających częstym zmianom i trudno jest ustalić wspólne spójne cele i wartości. Ich członkowie nie rozumieją zachodzących w nich procesów, łączność między nimi a jednostkami organizacyjnymi jest słaba i zawodna, a sam proces podejmowania decyzji przebiega w sposób chaotyczny. Bardzo często zdarza się, że brakuje jasności, kto ma podjąć decyzję i kogo ma ona dotyczyć.

Ze względu na fakt, że podejmowane decyzje bardzo często dotyczą wielu niespójnych celów, racjonalny proces decyzyjny powoduje zwątpienie i niepewność wśród załogi organizacji. W konsekwencji ogranicza to motywację i gotowość do działania. Podobnie roztrząsanie pozytywnych i negatywnych skutków dokonanych decyzji jeszcze bardziej osłabia te cechy i przyczynia się do braku porozumienia i wzrostu konfliktów między osobami podejmującymi decyzje. Na ogół [2] racjonalne podejmowanie decyzji jest uzasadnione w przypadku problemów nieskomplikowanych i dobrze zdefiniowanych, a sam proces racjonalnego podejmowania decyzji staje się dyskusyjny z punktu widzenia złożonych problemów, słabo sformalizowanych.

Stany emocjonalne również zakłócają procesy poznawcze, a wpływ na nasze decyzje mają również sympatie i antypatie. Różnie też spostrzegamy sytuacje, odnosząc je do perspektywy własnych korzyści bądź ich braku. To samo rozwiązanie może być uznane za szansę albo zagrożenie w zależności od sposobów jego zaprezentowania, co radykalne zmienia podejście do problemu i też wpływa na podjęte decyzje [5]. Wypracowane heurystyki [7] pozwalają na szybsze i oszczędniejsze podejmowanie decyzji, w stosunku do metod formalnych, bowiem wymagają znacznie mniejszej ilości informacji, a także liczby obliczeń, co w wielu przypadkach jest zaletą. Poza tym automatyzm niektórych czynności i operacji intelektualnych pozwala na przeznaczenie większej ilości czasu na ważniejsze zadania, co przyczynia się pośrednio do postępu cywilizacyjnego.

Powyższe rozważania oznaczają, że każdą decyzję dotyczącą człowieka uzasadnianą w sposób racjonalny, należy podać intuicyjnej ocenie bowiem umożliwia to dodatkową weryfikację i ewentualnie uwzględnienie dodatkowych wymagań. Podobnie każdą intuicyjną decyzję można skonfrontować (nawet po jej podjęciu) z pewnym modelem formalnym, co zwiększa przekonanie o wartości tej decyzji, a poza tym uczy i rozwija ludzkie doświadczenie. W następnych podrozdziałach zajmiemy się pewnym prostym problemem w celu weryfikacji tak rozumianego paradoksu decyzyjnego.

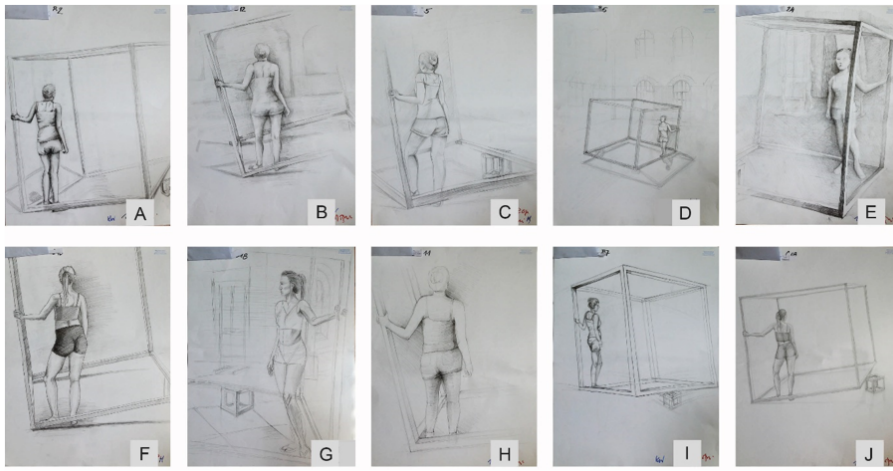
18.3 Eksperyment 1 - podejście intuicyjne

Z uwagi na złożoność problemów życiowych, jako eksperyment przyjmiemy uporządkowanie rysunków pod względem artystycznym. Załóżmy, że mamy do posortowania zestaw rysunków pochodzących np. z egzaminu wstępnego na architek-

turę. Zadanie egzaminacyjne było następujące: na podstawie obserwacji zadanej kompozycji należy ją narysować ołówkiem na dostarczonym papierze (format 80x65 cm), uwzględniając szczególną uwagę na formę, relacje przestrzenne elementów, ich ustawienie oraz na zakomponowanie rysunku. Kompozycja składała się z postaci ludzkiej oraz dwóch sześcianów. Przykład takiego zestawu prac jest przedstawiony na rys. 18.3. Osoby egzaminowane (około 500 osób) znajdują się w różnych lokacjach, co sprawia, że widok na kompozycję z różnych miejsc obserwacji jest różny. Stąd powstałe rysunki różnią się wieloma szczegółami. Ich ocenę zleca się zespołowi artystów, których zadaniem jest pogrupowanie rysunków na bardzo dobre, dobre, średnie, akceptowalne i nieakceptowalne. Zespół pracuje wspólnie i wykonuje dwa podejścia do oceny rysunków. Jednego dnia dokonuje wstępnego podziału, a drugiego podchodzi do niego krytycznie i dokonuje niezbędnych korekt. Tego typu postępowanie jest typowym działaniem intuicyjnym.

Nasze eksperymenty ograniczamy do mniejszej próby rysunków (np. po 10 sztuk, losowo dobranych, patrz rys. 18.3), a zadaniem jest dla każdego takiego zestawu ustawienie rysunków od najlepszego do najgorszego, gdzie kryterium porównania jest jakość artystyczna rysunków sformułowana w zadaniu egzaminacyjnym. Dla losowego zestawu rysunków R, (patrz rys. 18.1) $R = \{\text{A,B,C,D,E,F,G,H,I,J}\}$, zaś uszeregowanie dokonane przez ekspertów (artystów) wynosi U_1 i jest podane na rys. 18.4, gdzie $U_1 = \{\text{G,F,C,B,E,H,J,A,I,D}\}$. Przy dokonywaniu oceny rysunków nie wprowadzano limitu czasu na to zadanie, To samo zadanie wykonane przy wyraźnym ograniczeniu czasowym daje już inny wynik $U_2 = \{\text{G,F,E,B,C,A,I,H,J,D}\}$. Jako miarę zgodności α dwóch uporządkowań przyjęto liczbę pozycji na których występują te same rysunki. W rozważanym przykładzie $\alpha = 4$, z uwagi na to, że pozycje 1, 2 oraz 4 i 10 zawierają te same rysunki. Dodatkowym parametrem może być maksymalna różnica pozycji β na których występują te same rysunki. W rozpatrywanym przypadku $\beta = 2$ ponieważ w pierwszym uporządkowaniu rysunek C występuje na trzeciej pozycji, a w drugim na pozycji piątej, taka sama wartość β dotyczy też innych niezgodnych pozycji. Z tego wynika, że presja czasu jest istotnym czynnikiem wpływającym na zgodność uporządkowania. Określając dla porównania zgodność zestawów rysunków przedstawionych na rys. 18.1 i rys. 18.2 parametry α i β wynoszą: $\alpha = 3$, $\beta = 7$. Tak więc uporządkowanie pod presją czasu jest jednak lepsze od uporządkowania przypadkowego.

Powyżej założono, ze zespół ekspertów wspólnie dokonywał uporządkowania rysunków. Można jednak rozważyć taką strategię uporządkowania, że członkowie zespołu robią to dla tego samego zestawu rysunków najpierw indywidualnie i mając własne wyniki, dopiero uzgadniają wspólnie rozwiązanie. Na ogół oceniający rysunki indywidualnie skupiali się na sąsiednich rysunkach zamieniając je pozycjami (najczęściej parami), w przypadku uznania potrzeby takiej operacji. W przypadku oceny zespołowej przestawianie było dokonywane nie tylko dla dwóch rysunków sąsiednich. Znajomość indywidualnych ocen i uzgodnionego uporządkowania pozwala na głębszą dyskusję i z czasem, dla dalszych zestawów zauważalna jest wyższa zgodność indywidualnych uporządkowań.



Rysunek 18.3. Zestaw rysunków do oceny ułożony przypadkowo

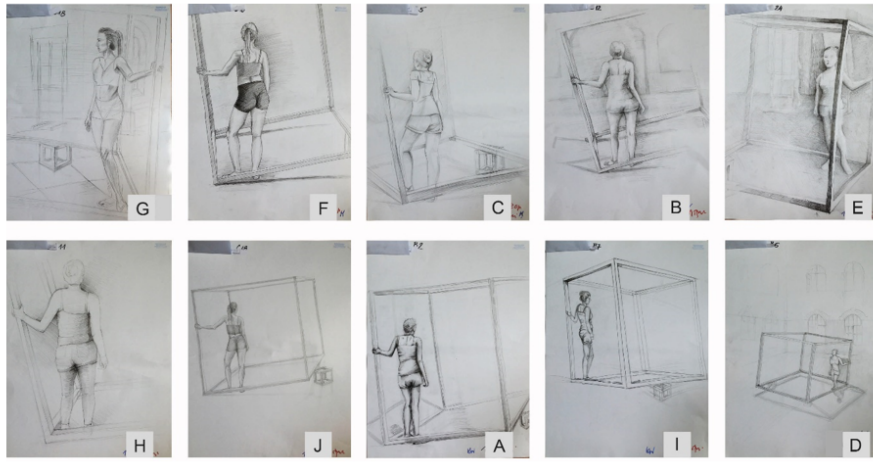
Podobne eksperymenty wykonano z udziałem oceniających o mniejszym doświadczeniu i wiedzy (np. studenci architektury). W tym przypadku zgodność ich uporządkowania w odniesieniu do wyniku ekspertów była wyraźnie niższa, $\alpha = 3$, $\beta = 4$. Zauważono również, że różnice w ocenie zależały od doboru rysunków do ocenianego zbioru. Kiedy rysunki do uporządkowania były wizualnie bardziej zróżnicowane artystycznie, to uzyskane oceny były bardziej zbliżone do siebie, jeśli różnice między rysunkami nie były zbyt wielkie, to oceny były bardziej zróżnicowane. Co ciekawsze, kiedy zamiast sortowania wszystkich rysunków zadanego zbioru, zalecono ich pogrupowanie w podgrupy o dość zbliżonej jakości artystycznej, to wówczas decyzje wśród artystów były prawie bezbłędne, zaś wśród studentów niewielkie (różnica co najwyżej jednego – dwóch obrazów w podgrupie).

Wydaje się, że tego typu eksperymenty nie wymagające żadnych obliczeń w dużym stopniu uwzględniają naturalne działania intuicyjne człowieka, poparte większą (artyści) lub mniejszą wiedzą (studenci) oraz odpowiednio różnym doświadczeniem. Nasuwa się dość istotny wniosek, że intuicja ludzka sprawuje się lepiej, kiedy nie występuje zbyt duża presja czasu (decydują większe emocje), bądź gdy wymagana jest mniejsza precyzja decyzji (np. zamiast szeregowania obrazów tylko ich grupowanie). W celu poznania sposobu myślenia oceniających (decydentów) zapytano ich po dokonaniu eksperymentów, jakimi przesłankami kierowali się przy ocenach. Odpowiedź była jednoznaczna; wymaganiami egzaminacyjnymi. Jednak przy tym akcentowano ważność ogólnego kryterium takiego jak „ogólne wrażenie artystyczne" czy „subiektywny odbiór całego rysunku". Po dyskusji z oceniającymi uzgodniono, że w zasadzie kierowano się następującymi kryteriami:

1. Zachowanie formy – odzwierciedlenie kształtów i trójwymiarowości;

2. Utrzymanie proporcji np. pomiędzy poszczególnymi elementami dotyczących postaci oraz sześcianu, w którym się ona znajduje,
3. Uchwycenie kierunków i perspektywy linearnej,
4. Ukazanie efektu przestrzeni – przedmioty bliższe i dalsze,
5. Kompozycja rysunku – wzajemny układ i wielkość elementów narysowanych w stosunku do wykorzystywanego arkusza rysunku.

Przeprowadzony eksperyment zwrócił również uwagę na ogólne traktowanie powyższych kryteriów przy dokonywaniu intuicyjnej oceny i uporządkowania obrazów. Wydaje się, że podejście racjonalne pozwala na pełniejsze skoncentrowanie się właśnie na tych kryteriach i na wyróżnionych mierzalnych cechach rysunków.



Rysunek 18.4. Zestaw rysunków uporządkowany wg oceny ekspertów. W lewym górnym rogu z oceną najwyższą, zaś w prawym dolnym najniższą

18.4 Eksperyment 2 – podejście racjonalne

Założymy, że człowiek staje przed dylematem oceny rozwiązań spośród przedstawionych mu propozycji $R = \{R_1, R_2, \ldots, R_m\}$. Każde rozwiązanie $R_i \in R$, $i = 1, 2, \ldots, m$; posiada zbiór cech $C_i = \{c_{ij}\}$, $j = 1, 2, \ldots, n$; z których każda może spełniać ustalone wymagania w różnym stopniu, bądź ich nie spełniać wcale, co możemy zapisać w sposób następujący:

$$C_{ij} = \begin{cases} 2 - \text{jeśli w pełni spełnione jest wymaganie} \\ 1 - \text{jeśli jest ono spełnione w sposób zadawalający} \\ 0 - \text{jeśli nie spełnione jest wymaganie} \end{cases} \quad (18.1)$$
$$i = 1, 2, \ldots, m; \quad j = 1, 2, \ldots, n;$$

W przypadku oceny rysunków: zbiór R stanowi ich pewien zestaw, zaś odpowiadający każdemu rysunkowi wymagania stanowią kryteria oceny zdefiniowane w rozdziale poprzednim. W rozpatrywanym przypadku m = 10, n = 5. Wektor $C_i = [1, 1, 1, 2, 2]$ oznacza, że i-ty rysunek, np. dla $i = 5$ rysunek E, w miarę zachowuje formę, utrzymuje proporcję i perspektywę, a bardzo wyróżnia się co do wykazania efektu przestrzeni oraz kompozycji. Przyjmiemy, że suma wartości c_{ij} dla wektora C_i stanowi jakość artystyczną rysunku W_i, gdzie :

$$W_i = \sum_{j=1}^{n} c_{ij} \quad (18.2)$$

W rozpatrywanym przypadku $W_5 = 7$. Przypisując wartości W_i do poszczególnych rysunków tworzymy zbiór liczb W, którego wartości wyniosą: $W = \{3, 7, 8, 1, 7, 8, 8, 6, 3, 6\}$. Zatem problem uporządkowania rysunków został sprowadzony do problemu sortowania zbioru liczb. Uporządkowany zbiór rysunków dla takich wartości wektora W po posortowaniu zbioru R wyniesie: $U = \{\underline{C,F,G;B,E;H,J;A,I;D}\}$. W zbiorze U zaznaczono przez podkreślenie podzbiory równoważne, tzn. posiadające obrazy o tej samej wartości artystycznej. Porównując tak obliczoną decyzję U z decyzją przedstawioną na rys. 18.2 stwierdzamy pełną ich zgodność, przy założeniu, że rysunki z podzbiorów równoważnych mogą występować w dowolnej kolejności, tzn. $\alpha = 10$, $\beta = 0$. Jednak takie grupowe uporządkowanie zbioru R, wykonane na zlecenie przez ekspertów tworzy już cztery inne podzbiory $U^* = \{\underline{C,F,G},\underline{B,E},\underline{H,J},\underline{A,I,D}\}$. Wówczas otrzymujemy pełną zgodność uporządkowania z dokładnością do wyróżnionych podzbiorów, $\alpha = 10$, $\beta = 0$! Oczywiście nie zawsze tak być musi, a nawet najczęściej tak nie jest. Zaproponowaną metodę można rozbudować opisując każdą cechę C_i kilkoma parametrami, a także wprowadzając wagi do poszczególnych kryteriów. W ten sposób można zdefiniować odpowiedni problem optymalizacyjny i wybierać do jego rozwiązania, jedną z metod dotyczącą badań operacyjnych.

Istnieje jednak poważny problem z wyznaczeniem cech C_i bądź parametrów reprezentujących te cechy. Powyżej założyliśmy, że wartości wektora W są określane przez ekspertów. Jednakże racjonalne podejście wymagałoby wyliczenia tych wartości na podstawie opracowanych algorytmów analizy rysunków zapisanych w postaci pdf czy jpg. Konieczny jest więc algorytm komputerowy oceny jakości formy, proporcji, perspektywy oraz przestrzeni i kompozycji dla dowolnego rysunku ze zbioru R, a w zasadzie dla każdego dostępnego rysunku. Dopiero na tej podstawie można byłoby wyznaczyć jakość artystyczną rysunku. Obecnie nie istnieją precyzyjne algorytmy, które pozwalałyby na automatyczne wyliczanie takich parametrów i ich normalizację do standardowych wartości: 2, 1, 0. Możliwe jest jedynie wyznaczenie pewnych charakterystycznych cech analizowanych

rysunków (odpowiednie heurystyki), które w pewien sposób ujmują oceniane cechy [11]. To jednak wymaga dalszych szczegółowych badań i analiz. Na przykład, identyfikacja konturów umożliwiłaby w pewnym sensie sprawdzenie proporcji i perspektywy, a być może i kompozycji rysunku. Analiza formy i przestrzeni wymagałaby zamianę rysunku dwuwymiarowego na trójwymiarowy oraz również badanie występujących tam zależności. Można byłoby dobierać algorytmy zorientowane na identyfikację różnych obiektów, np. przy różnie ulokowanej postaci ludzkiej, czy różnie rozmieszczonych dwóch sześcianów. Warto przy tym podkreślić, że rysunki są tworzone na podstawie obserwacji zadanej kompozycji obiektów z różniących się punktów obserwacji, co jeszcze bardziej utrudnia obliczenie czy szacowanie niezbędnych parametrów. Tak więc racjonalne podejście wymaga wielu dalszych badań związanych z przetwarzaniem obrazów, w tym wykorzystanie złożonych metod obliczeniowych, w tym metod sztucznej inteligencji. Tylko przygotowanie aplikacji internetowej, która gromadzi wykonane rysunki i wspomaga proces oceny jest na dzisiaj jak najbardziej realne. W przypadku oceny jedynym postępowaniem jest więc połączenie podejścia racjonalnego z intuicyjnym.

Zdefiniowanie uporządkowania rysunków jako problemu optymalizacyjnego jest możliwe, ale jego rozwiązanie wymaga konkretnych danych, które powinni dostarczać eksperci. Niestety w takim przypadku znacznie większy nakład pracy pozostaje po stronie ekspertów niż systemów obliczeniowych. Zatem należałoby w pewien sposób ułatwić pracę wykonywaną przez ekspertów. Zakładając że komputerowa analiza obrazu jest trudniejsza niż porównanie dwóch obrazów (tak robią też eksperci) można w dużej mierze przekazać tą pracę komputerom. Konieczne jest tylko przyjęcie rysunku wzorca i wykorzystanie go do porównania rysunków ze zbioru ocenianego. Rysunek bądź rysunki wzorcowe odpowiadające różnym punktom obserwacji mogą utworzyć eksperci. Cały proces oceny sprowadza się wówczas do porównania rysunku wzorcowego z analizowanym. Wykonując dodatkowe operacje na rysunkach, takie jak dopasowanie skali przedstawionych obiektów, dokonanie obrotu obiektów, można wyznaczyć (z pewnym przybliżeniem, ale z mniejszym wysiłkiem) analizę porównawczą wszystkich rysunków oraz podać skalę podobieństwa rysunku badanego ze wzorcem. W konsekwencji można też odnieść ją do jakości artystycznej ocenianego rysunku. Ponownie takie porównanie ze wzorcem wymaga dalszych studiów i przygotowania odpowiednich algorytmów. Co ciekawe, zakładając, że rysunkami wzorcowymi są dwa pierwsze rysunki przedstawione w rankingu na rys. 18.2 analiza zgodności wybranych obszarów pikseli zachowuje wypracowaną przez ekspertów kolejność uporządkowania obrazów.

18.5 Wnioski końcowe

W rozwoju naszej cywilizacji były okresy, które zachwycały się podejściem racjonalnym i w nim upatrywano rozwiązanie wszelkich problemów ludzkich. Z czasem przekonano się, że nie jest to panaceum na wszystko i zaczęto doceniać również podejście intuicyjne. W pracy przedstawiono koncepcję synergii podejścia

racjonalnego i intuicyjnego wynikającą z paradoksu decyzyjnego. Wartościowe jego wykorzystanie jest uwarunkowane wykonaniem odpowiednich badań po obu stronach. Już na prostym przykładzie oceny rysunków wykazano, że samo racjonalne podejście nie jest wystarczające z uwagi na złożoność problemu, brak gotowych metod analizy, czy trudności w rozstrzyganiu pewnych kwestii. Podobnie podejście intuicyjne wymaga dalszych badań związanych z planowaniem odpowiednich eksperymentów (dobór reprezentatywnych problemów), określenia właściwych metod podejmowania decyzji, czy uwzględnienia różnych czynników zewnętrznych (presja czasu, ludzka postawa) na jakość podejmowania decyzji.

Z rozważań wynika, że właściwa symbioza obu metod wyzwala nowe możliwości mądrego decydowania, tzn. takiego, które świadomie akceptuje nie optymalne a satysfakcjonujące rozwiązania, które nie prowadzą jednak do nieprzewidywalnych i groźnych konsekwencji. W naszych rozważaniach taki wybór symbolizuje uszeregowanie rysunków. Zaakcentowano też potrzebę określenia pewnego wzorca odniesienia, który wyznaczałby płaszczyznę integracji obu podejść: racjonalnego i intuicyjnego. Określanie takiego wzorca jest też jednak ogromnym wyzwaniem i wymaga złożonych modeli i zaawansowanych eksperymentów.

Posłużono się rzeczywistym problemem szeregowania rysunków ze względu na wielowymiarowy aspekt analizy, jakim jest ich jakość artystyczna, trudna do racjonalnego opisu. Dlatego w przedstawionych rozważaniach dominuje podejście intuicyjne. Oczywiste jest, że podejmowanie decyzji jest tym łatwiejsze i bardziej precyzyjne, gdy wiedza o problemie jest coraz szersza oraz dostępne są coraz lepsze algorytmy analizy obrazów. Czy to oznacza, że z czasem dominującą rolę przejmie w tym przypadku podejście racjonalne? Wraz z postępem zmieniają się też problemy do rozwiązania, odkrywane będą nowe obszary do zbadania (choćby porównywanie obrazów czy fotografii, a nie prostych rysunków), więc intuicja nie straci swojego znaczenia.

Tego typu eksperymenty należałoby jeszcze skojarzyć z badaniami neurologicznymi czy psychologicznymi, by łatwiej rozumieć ludzkie zachowania, w tym rozpoznawanie stanu emocjonalnego członków zespołu podejmującego różne decyzje oraz związanego z nim kontekstu i wyboru odpowiedniej reakcji na zaistniałe zdarzenie. Dzięki połączeniu sztucznej inteligencji, technik projektowych ukierunkowanych na człowieka oraz nowoczesnych technologii, takie badania i eksperymenty staną się coraz bardziej realne. Ułatwi to w przyszłości budowę nowych inteligentnych algorytmów decyzyjnych, jak na razie ukrytych w ludzkich umysłach oraz stworzy możliwości właściwego reagowania na bieżąco w przypadku nagle pojawiających się złożonych i niebezpiecznych sytuacji.

Bibliografia

1. Adair, J.: Decision Making and Problem Solving. Kogan Page (2019).
2. Brunson, J.: In the Middle with Trueness: The Transforming Resonance of a Leader. Createspace Independent Publishing Platform (2017).
3. Epitropakis, M.G., Burke, E.K.: Handbook of Heuristics. Springer International Publishing (2018).

4. Kahneman, D.: Thinking, Fast and Slow. Penguin Books Limited (2011).
5. Kahneman, D., Slovic, P., Tversky, A.: Judgment Under Uncertainty. Heuristics and Biases. Cambridge University Press (1982).
6. Krawczyk, H., Targowski, A.: Wisdom in the Context of Globalization and Civilization. Cambridge Publishing Company (2019).
7. Kruglanski, A., Gigerenzer, G.: Intuitive and deliberate judgments are based on common principles. Psychological review **118**(1), 97–109 (07 2011).
8. Medinilla, A.: Managing Continuous Improvement Far Beyond Retrospectives. Springer (2016).
9. Nolte, J.: Elsevier's Integrated Neuroscience. Mosby (2007).
10. Peterson, J.B.: Beyond Order: 12 More Rules for Life. Allen (2021).
11. Rafael, G.C.: Judgment Under Uncertainty. Heuristics and Biases. Digital Image Processing, Pearson Education Limited (2018).
12. Russell, S., Norvig, P.: Artificial Intelligence – A Modern Approach. Prentice Hall International (2016).
13. Schwartz, B.: The Paradox of Choice. Why More is Less. Harper Collins Publisher (2004).
14. Sedgewick, R., Wayne, K.: Algorytmy. Helion SA. (2019).
15. Simon, H.A.: Models of man, social and rational. New York: John Wiley & Sons (1957).
16. Thaler, R.: Misbehaving — The Making of Behavioral Economics. Norton (2016).
17. Triantaphyllou, E.: Multi-criteria Decision Making Methods. Springer Verlag New York Inc. (2010).
18. Triantaphyllou, E., Mann, S.H.: An examination of the effectiveness of multidimensional decision-making methods: A decision-making paradox. Decision Support Systems **5**(3), 303–312 (1989).
19. Tversky, A., Kahneman, D.: The framing of decisions and the psychology of choice. Science **211**(4481), 453–458 (1981).
20. Wagner, H.M.: Principles of Operations Research, with Applications to Managerial Decisions. Prentice Hall (1969).
21. Werner, J.: A garbage can model of organizational choice. In: The Oxford Handbook of Classics in Public Policy and Administration, pp. 300–315. Oxford University Press (2015).
22. Witulani, J.: Bez ograniczeń. Jak rządzi nami mózg. PWN (2015).