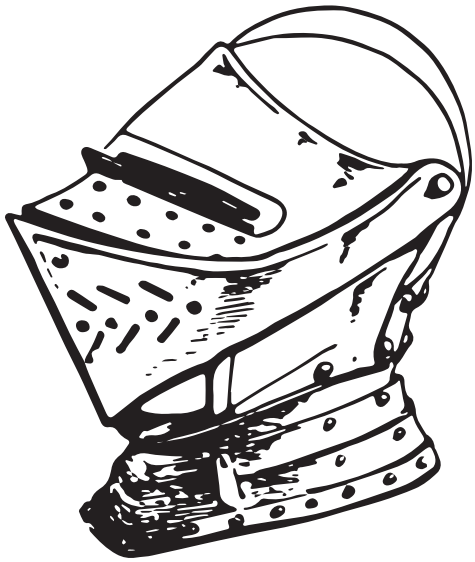
KASKBOOK

Algorytmy i zastosowania inteligencji obliczeniowej

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



Gdańsk 2022

PRZEWODNICZĄCY KOMITETU REDAKCYJNEGO
WYDAWNICTWA POLITECHNIKI GDAŃSKIEJ
Dariusz Mikielewicz

REDAKTOR PUBLIKACJI NAUKOWYCH
Michał Szydłowski

RECENZENCI
Wojciech Jędruch
Marek Moszyński
Marcin Woźniak

SKŁAD
Szymon Olewniczak

PROJEKT OKŁADKI
Mateusz Olszewski

Wydano za zgodą
Rektora Politechniki Gdańskiej

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

ISBN 978-83-7348-882-3

WYDAWNICTWO POLITECHNIKI GDAŃSKIEJ

Wyd. I. Ark. wyd. 14,5, ark. druku 16,5, 272/1196

Spis treści

Przedmowa

Dwunasta edycja monografii naukowej KASKBOOK Katedry Architektury Systemów Komputerowych Wydziału Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej poświęcona jest algorytmom sztucznej inteligencji i ich zastosowaniom. Opisano szereg podejść bardzo aktualnych, obrazujących dynamiczny postęp w dziedzinie uczenia maszynowego, jak również usystematyzowano i zobrazowano na prostych przykładach metody podstawowe. W rozdziałach nie zabrakło także rozważań nad pozyskiwaniem danych, które obok algorytmów są bardzo istotną składową dobrze działającego inteligentnego systemu. W monografii poruszono szereg zagadnień związanych z dziedziną uczenia maszynowego: przetwarzaniem języka naturalnego, obrazów, dźwięku z użyciem sieci neuronowych i klasycznych podejść. Znalazło się tu również miejsce dla rozważań związanych z bezpieczeństwem i efektywnością obliczeń oraz z metodami optymalizacji przetwarzania dużych ilości danych w środowiskach HPC.

W tym miejscu chciałbym podziękować osobom, dzięki których pomocy powstała ta monografia: mgr. inż. Szymonowi Olewniczakowi za skład tekstu, Mateuszowi Olszewskiemu za projekt okładki oraz Agnieszce Frankiewicz za korektę edytorską. Szczególne podziękowania składam dr. hab. inż. Marcinowi Woźniakowi z Politechniki Śląskiej, dr. hab. inż. Markowi Moszyńskiemu z Politechniki Gdańskiej i dr. hab. inż. Wojciechowi Jędruchowi z Akademii Marynarki Wojennej z Gdyni za recenzje i uwagi, dzięki którym udało zachować się wysoki poziom redakcyjny i merytoryczny tego wydania.

dr hab. inż. Julian Szymański, prof PG

Rozdział 1

Semantyczne wektory słów

Szymon Olewniczak[1]

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

Abstrakt

Niniejszy rozdział stanowi wstęp do rozległego zagadnienia, jakim są semantyczne wektory słów. W szczególności skupiono się w niej na metodach automatycznego tworzenia tego typu reprezentacji na podstawie dużych zbiorów danych. Omówiono także różne możliwe interpretacje tego, czym tak naprawdę jest podobieństwo słów, oraz przedstawiono wybrane zastosowania tego typu modeli.

Słowa kluczowe: semantyczne wektory słów, osadzenia słów, torowanie semantyczne

1.1. Wstęp

Słowa to nie tylko znaki na papierze czy wypowiadane głoski. Słowa to ukryte za nimi myśli, pojęcia, wspomnienia, odczucia i nadzieje. Słowa nie tylko nas informują. Mogą nas też umacniać, pocieszać, ale niestety też atakować i niszczyć. Trudno się więc dziwić, że komputerom tak trudno przychodzi ich prawidłowa interpretacja.

Słowa stanowią najmniejsze i podstawowe elementy, z których budujemy zdania, teksty czy wypowiedzi. Można by się tu spierać, czemu to nie literom przypada ten zaszczytny tytuł. Litery jednak same w sobie niczego nie znaczą. Dopiero słowo stanowi najmniejszą jednostkę, która przenosi myśl.

W niniejszym rozdziale przedstawiono najważniejsze trendy w dziedzinie modelowania języka za pomocą semantycznych wektorów słów (*semantic word vectors*), zwanych także osadzeniami słów (*word embeddings*). Dziedzina ta wpisuje się w szerszą tematykę przetwarzania języka naturalnego (*Natural Language Processing*, NLP) i stanowi ważny element wielu ze stosowanych tam rozwiązań.

1.2. Wektory słów

Zanim przejdziemy dalej, zdefiniujmy, czym właściwie są wektory słów.

Definicja 1.1 (Wektory słów) *Wektory słów to reprezentacje słów (pojęć) z danego słownika (D) w postaci n-wymiarowych wektorów w taki sposób, że zadana metryka odległości pomiędzy poszczególnymi wektorami odzwierciedla podobieństwo odpowiadających im słów.*

Najczęściej wektory słów tworzone są dla słownika zawierającego pojedyncze słowa, czasem jednak słownik jest rozszerzany o dodatkowe terminy składające się z kilku wyrazów, które po rozbiciu straciłyby swoje pierwotne znaczenie (np. Nowy Jork, Politechnika Gdańska). Z tego powodu, w celu podkreślenia, że słownik składa się nie tylko z pojedynczych słów, czasem zamiast bardziej popularnego terminu osadzenia słów stosuje się termin osadzenia terminów (*term embeddings*) [10].

Tworzone wektory słów z reguły składają się z dużej liczby wymiarów (popularne są na przykład wektory 300-wymiarowe). Każdy z wymiarów reprezentuje w sposób ilościowy pewną cechę danego pojęcia. Cechy te mogą być dobrane w sposób ręczny albo automatyczny na podstawie analizy dużych zbiorów danych. Cechami wybranymi ręcznie mogłyby być na przykład: „miastowość", „zwierzęcość", „domowość". Oczekiwalibyśmy wówczas, że słowa takie jak „Warszawa" i „Gdańsk" będą miały duży udział pierwszego wymiaru i stosunkowo mniejszy w pozostałych, słowa takie jak „lew", czy „słoń" reprezentowane byłyby przez duże wartości w drugim wymiarze, natomiast słowa takie jak „kot" i „pies" mogłyby być reprezentowane przez mieszankę wartości dwóch ostatnich wymiarów wektora osadzeń. W przypadku automatycznie tworzonych wektorów interpretacja poszczególnych wymiarów często nie jest aż tak oczywista, jednak tego typu wektory są zwykle dużo użyteczniejsze w praktyce.

Metryką, którą najczęściej stosuje się w celu wyznaczenia odległości pomiędzy poszczególnymi wektorami, jest podobieństwo cosinusowe:

$$sim(\vec{a}, \vec{b}) = cos(\vec{a}, \vec{b}) = \frac{\vec{a} \circ \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{1.1}$$

Wartość podobieństwa cosinusowego może przyjmować wartości z zakresu $\langle -1, 1 \rangle$. Czym bliżej 1 się znajdujemy, tym bardziej podobne są do siebie wektory.

Nieco rzadziej stosuje się też zwykłą odległość euklidesową:

$$d(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{1.2}$$

Zbiór wszystkich wektorów oraz wybrana metryka tworzą przestrzeń wektorową, w której wektory reprezentujące słowa bardziej zbliżone znajdują się bliżej siebie, a wektory mniej powiązane dalej.

Sama definicja wektorów słów nie jest skomplikowana, jednak określenie, co tak naprawdę oznacza podobieństwo słów, nie jest już kwestią aż tak oczywistą. W kolejnym rozdziale zostanie przedstawionych kilka możliwych podejść do tego zagadnienia.

1.3. Podobieństwo słów

Zanim przejdziemy do opisu konkretnych algorytmów umożliwiających tworzenie semantycznych wektorów słów w sposób automatyczny, musimy się zastanowić nad tym, czym w gruncie rzeczy jest podobieństwo słów. Dlaczego uważamy niektóre słowa za zbliżone, a inne nie.

W ogólności można wyróżnić trzy główne kategorie, według których możemy kategoryzować nasze słowa. Pierwszą stanowi gramatyka, drugą semantyka, a trzecią podobieństwo skojarzeniowe [3]. Przedstawimy teraz pokrótce te trzy możliwości.

1.3.1. Podobieństwo gramatyczne

Ta kategoria stanowi najprostszą i jednocześnie najbardziej techniczną możliwość grupowania słów. Słowa są sobie bliskie, jeżeli stanowią tę samą część mowy. Oznacza to, że nasza metryka odległości powinna zwracać wysoką wartość dla dwóch rzeczowników, ale już mniejszą przy porównaniu czasownika i rzeczownika. Różnice mogą być też bardziej subtelne. Na przykład czasowniki w czasie przeszłym mogą w przestrzeni wektorowej znajdować się bliżej siebie niż w czasie teraźniejszym.

W praktyce nie stosuje się wektorów słów przechowujących jedynie podobieństwo gramatyczne między słowami, choć wiele ze schematów automatycznego budowania semantycznych osadzeń słów przy okazji informacji semantycznej przechowuje również podobieństwo gramatyczne [3].

1.3.2. Podobieństwo semantyczne

Nie istnieje jedna przyjęta definicja, czym dokładnie jest podobieństwo semantyczne pomiędzy wyrazami. Z reguły jednak przyjmuje się, że dwa wyrazy są do

siebie semantycznie zbliżone, jeżeli oznaczają zbliżone (lub wręcz te same, synonimiczne) pojęcia oraz to podobieństwo jest w pewien sposób uogólnione, wspólne dla większości użytkowników danego języka. Często możemy też powiedzieć, jaka cecha wspólna łączy wybrane pojęcia. Na przykład słowa „samochód" i „motocykl" mogą być semantycznie bardziej powiązane (oba są środkami transportu) niż słowa „samochód" i „smutny". Innym przykładem zbliżonych semantycznie terminów mogą być nazwy emocji, które często występują razem, na przykład „przestraszony" i „zdenerwowany".

Warto tutaj wprowadzić również podział podobieństwa semantycznego na podobieństwo paradygmatyczne (*paradigmatic*), zwane też podobieństwem ze względu na typ (*typical*), oraz podobieństwo syntagmatyczne (*syntagmatic*), nazywane również tematycznym (*topical*).

W podobieństwie paradygmatycznym zbliżone do siebie są pojęcia reprezentujące obiekty tego samego typu (lub wręcz po prostu będące swoimi synonimami), które można stosować w podobnych kontekstach, na przykład: włócznia i miecz (oba stanowią rodzaj broni białej), politechnika i uniwersytet (oba reprezentują różne rodzaje szkół wyższych), Gdańsk i Warszawa (oba są miastami).

W przypadku podobieństwa tematycznego większy nacisk kładziemy na podobieństwo pojęć, które łączy jakaś wspólna tematyka i często występują razem, na przykład: Fontanna Neptuna, plaża, Lechia Gdańsk (wspólny temat: Gdańsk), albo okręt, rafa koralowa, ryba (wspólny temat: morze). Pojęcia te są różnych typów, jednak jesteśmy w stanie określić pewną łączącą je tematykę.

To, na ile przedstawione pojęcia są do siebie podobnie i czy dane podobieństwo jest bardziej paradygmatyczne, czy syntagmatyczne, oczywiście zawsze będzie w pewnym stopniu kwestią sporną. Warto jednak zwrócić uwagę na tę różnicę, gdyż czasem może okazać się istotna przy decyzji o tym, w jaki sposób chcemy budować nasze wektorowe reprezentacje pojęć.

Z reguły to właśnie podobieństwo semantyczne jest tym, co nas najbardziej interesuje przy tworzeniu osadzeń słów. W dalszej części tej pracy będziemy się odnosić właśnie do tego spojrzenia na podobieństwo.

1.3.3. Podobieństwo skojarzeniowe

Nieco osobny temat stanowi podobieństwo skojarzeniowe (*associative relationship*), wiążące się z badaniami prowadzonymi nad zjawiskiem tzw. torowania semantycznego (*semantic priming*) [11]. W eksperymentach prowadzonych w ramach tej dziedziny próbujemy określić podobieństwo słów na podstawie obserwacji reakcji uczestników eksperymentu na pewne pojęcia. W najpopularniejszej procedurze eksperymentalnej, zwanej torowaniem pojedynczym słowem (*single-word priming*), pokazujemy uczestnikowi najpierw pierwsze słowo, zwane pojęciem torującym (*prime*), a następnie słowo docelowe (*target*). Obserwując uczestnika badania, można zaobserwować, że pewne pary słów powodują szybsze reakcje niż inne. Na tej podstawie możemy tworzyć mapy podobieństw skojarzeniowych między słowami.

Podobieństwo skojarzeniowe może czasem odzwierciedlać podobieństwo semantyczne, choć nie zawsze musi tak być. Na przykład para słów: pleśń – chleb

wykazuje silne podobieństwo skojarzeniowe, choć można mieć wątpliwość, czy podane wyrazy są ze sobą semantycznie związane [4].

1.3.4. Problem polisemii i dryfu znaczeniowego

Niezależnie od sposobu, w jaki określamy podobieństwo słów, pojawiają się tutaj dwa istotne problemy, z którymi nie tak łatwo jest się zmierzyć. Pierwszy dotyczy słów, które posiadają wiele różnych znaczeń (polisemia), drugi dotyczy zjawiska dryfu znaczeniowego (*semantic drift*).

Pewne słowa mogą mieć wiele, czasem zupełnie różnych, znaczeń. Na przykład słowo „język" może oznaczać zarówno organ ciała, jak i mowę ludzką, „zamek" może oznaczać budowlę albo mechanizm. Polisemia stanowi szczególnie duże wyzwanie w przypadku schematów osadzeń budowanych w sposób automatyczny, na podstawie dużego korpusu językowego, i większość najpopularniejszych rozwiązań ignoruje ten problem. Prowadzi to do tego, że wektory reprezentujące słowa polisemiczne stanowią w pewnym sensie uśrednienie wszystkich ich znaczeń. Z racji ich koncepcyjnej prostoty i popularności w ramach niniejszego rozdziału skupimy się jedynie na tego rodzaju uśrednionych wektorach. Warto jednak pamiętać, że istnieją też rozwiązania próbujące uwzględnić problem polisemii, na przykład ELMo [13] czy Probabilistic FastText [1].

Drugi z problemów, problem dryfu znaczeniowego, wynika z faktu, że język ciągle ewoluuje. Pewne wyrazy tracą swoje pierwotne znaczenie albo w ogóle zanikają, inne się pojawiają. To zjawisko może być szczególnie widoczne w wektorach dziedzinowych, na przykład reprezentujących nazwy różnych technologii informatycznych, gdzie trendy ulegają niezwykle dynamicznym zmianom. Zazwyczaj wymusza to na nas cykliczne rewidowanie naszych wektorów słów i dostosowywanie ich do zmieniającej się rzeczywistości.

1.4. Rodzaje reprezentacji wektorowych

Wiemy już mniej więcej, czym są wektory słów oraz jaką informację mogą one przechowywać. Teraz omówimy najważniejsze sposoby, w jaki takie wektory można tworzyć.

Na najbardziej ogólnym poziomie możemy podzielić wektorowe reprezentacje słów na lokalne (*local*) i rozproszone (*distributed*).

W reprezentacjach lokalnych wielkość wektorów osadzeń odpowiada rozmiarowi naszego słownika: $|D|$. Każde słowo jest reprezentowane jako wektor samych zer, z pojedynczą jedynką na pozycji unikalnej dla każdego słowa, tzw. kodowanie z gorącą jedynką (*one-hot encoding*), $\vec{v} \in \{0,1\}^{|D|}$. W tej reprezentacji wszystkie wektory są względem siebie ortogonalne, czyli nie jesteśmy w stanie w żaden sposób określić, które z terminów są do siebie bardziej podobne, a które mniej. Z definicji reprezentacja lokalna nie umożliwi nam zakodowania podobieństwa semantycznego poszczególnych terminów. W dalszej części tego opracowania nie będziemy już do niej wracać.

Inaczej sprawa wygląda w przypadku reprezentacji rozproszonych. Tutaj każdy termin w naszym słowniku reprezentowany jest przez wektor liczb rzeczywistych o zadanym rozmiarze k, $\vec{v} \in \mathbb{R}^k$. Poszczególne wymiary wektorów rozproszonych reprezentują pewne cechy naszych pojęć, które mogą się w pewnej części pokrywać, dzięki czemu możliwe jest obliczenie odległości pomiędzy poszczególnymi wektorami. Docelowo chcielibyśmy, aby ta odległość odpowiadała wybranemu przez nas podobieństwu słów.

Cechy, na podstawie których chcemy określać pojęcia w naszej reprezentacji rozproszonej, mogą zostać dobrane w sposób ręczny. W praktyce jednak zadanie to jest niezwykle skomplikowane – trudno jest określić zbiór atrybutów, który byłby uniwersalny dla wszystkich pojęć w języku. Innym rozwiązaniem jest tworzenie semantycznych wektorów w sposób automatyczny, bazując na dużych korpusach tekstu. Podejście to opiera się na spostrzeżeniu sformułowanym w 1957 roku przez Jona Firtha: „słowo można scharakteryzować poprzez inne pobliskie słowa" [6]. W dalszej części rozdziału skupimy się na różnych modelach bazujących na tym spostrzeżeniu i umożliwiających nam tworzenie różnych typów semantycznych osadzeń słów.

1.5. Analiza utajonych własności semantycznych

Analiza utajonych własności semantycznych (*Latent Semantic Analysis*, LSA) [5] jest jednym z najwcześniejszych pomysłów wykorzystujących duże zbiory danych w celu utworzenia semantycznych reprezentacji słów. Pomysłem leżącym u podstaw tego algorytmu jest spostrzeżenie, że słowa występujące często razem w jednym dokumencie są do siebie semantycznie zbliżone.

Algorytm LSA tworzący semantyczne wektory składa się z dwóch etapów. W pierwszym dla każdego słowa w naszym słowniku tworzymy wektor o długości odpowiadającej liczbie dokumentów w naszym zbiorze treningowym. W najprostszym wariancie wektor ten zawiera 0, jeżeli słowo nie istnieje w konkretnym dokumencie, a 1 w przeciwnym wypadku. Tego typu reprezentację nazywamy workiem słów (*bag-of-words*, BoW). Alternatywnie można zastosować bardziej wyrafinowane reprezentacje, takie jak np. TF-IDF, TF-ICF albo Okapi BM25 (przegląd różnych rozwiązań można znaleźć w artykule [7]), które mogą poprawić nieco jakość stworzonych osadzeń, jednak nie wpływają znacząco na istotę działania algorytmu.

Etap drugi polega na dokonaniu rozkładu naszej macierzy według własności osobliwych (*Singular Value Decomposition*, SVD). Algorytm SVD umożliwia nam przedstawienie dowolnej macierzy jako iloczynu trzech innych macierzy o specjalnych własnościach:

$$W_{m\times n} = U_{m\times p} S_{p\times p} V^T_{p\times n} \tag{1.3}$$

Macierz $W_{m\times n}$ stanowi naszą wejściową macierz słowo–dokument, w której m oznacza liczbę słów w naszym słowniku, a n liczbę dokumentów w korpusie treningowym. Macierz $U_{m\times p}$ nazywamy lewostronnymi wektorami osobliwymi,

p oznacza liczbę „tematów" w korpusie, która jest równa liczbie dokumentów n, macierz S nazywamy wartościami osobliwymi, a $V_{p\times n}$ prawostronnymi wektorami osobliwymi. Z punktu widzenia algorytmu LSA najbardziej interesująca jest pierwsza z tych trzech macierzy. $U_{m\times p}$ reprezentuje korelacje pomiędzy poszczególnymi słowami a tematami na podstawie częstości ich występowania w tych samych dokumentach. Słowa, które często występują razem, są tematycznie bardziej powiązane niż te występujące rzadziej. Inną ważną własnością macierzy $U_{m\times p}$ jest uporządkowanie według ważności tematów. W miarę jak przeglądamy naszą macierz od lewej strony do prawej, poszczególne tematy stają się coraz mniej istotne dla prawidłowego odtworzenia macierzy słowo–dokument. Dzięki temu możemy zignorować część tematów znajdujących się po prawej stronie macierzy, co prowadzi do zredukowania wymiarowości naszych wektorów słów. Ignorowanie wymiarów wpływa także pozytywnie na generalizację naszych osadzeń. Liczba wymiarów p, które chcemy zachować w naszych docelowych osadzeniach, jest często sporo mniejsza niż pierwotna liczba dokumentów: $p \ll n$ (na przykład dla $n = 1\,000\,000$ p może się równać 300), co jednak pozwala nadal zachować kluczowe korelacje pomiędzy poszczególnymi słowami.

Wektory osadzeń słów stworzone przy użyciu algorytmu LSA w dużej mierze zbliżone są do tego, co zdefiniowaliśmy jako tematyczne podobieństwo semantyczne. Jeżeli zależy nam na właśnie takim sposobie postrzegania podobieństwa językowego, algorytm ten może okazać się rozwiązaniem lepszym niż nowocześniejsze techniki, takie jak na przykład Word2vec.

1.6. *Hyperspace Analogue to Language*

Model HAL (*Hyperspace Analogue to Language*), zaproponowany w 1998 roku przez Curta Burgessa [3], stanowi obok LSA jeden z pierwszych algorytmów służących budowaniu semantycznych wektorów słów przy wykorzystaniu dużych zbiorów danych tekstowych. Jednak w przeciwieństwie do LSA, zamiast wykorzystywać współwystępowanie słów w jednym dokumencie jako wskaźnik podobieństwa, bierze pod uwagę najbliższe otoczenie danego słowa w zdaniu.

Zasada działania algorytmu HAL jest stosunkowo prosta. W pierwszej kolejności tworzymy macierz o rozmiarze $m \times m$, gdzie m stanowi liczbę słów w naszym słowniku. W celu wypełnienia macierzy tworzymy tzw. okno przesuwne (*sliding window*) o szerokości 10 wyrazów, które w każdym kroku przesuwamy po naszym korpusie o jeden wyraz, rozpoczynając od pierwszych dziesięciu wyrazów w pierwszym dokumencie, a kończąc na ostatniej dziesiątce w ostatnim z dokumentów. W każdym kroku algorytmu wybieramy kolumnę z naszej macierzy odpowiadającą słowu znajdującemu się bezpośrednio przed naszym oknem. Następnie do komórki macierzy znajdującej się na przecięciu wybranej kolumny oraz wiersza odpowiadającego pierwszemu słowu w naszym oknie dodajemy 10, do komórki odpowiadającej drugiemu słowu w oknie dodajemy 9, i tak dalej dla pozostałych wyrazów w oknie.

Ostatecznie po przetworzeniu całego korpusu wejściowego każdy wiersz ma-

cierzy reprezentuje częstości współwystępowania pozostałych słów za wybranym słowem, a każda kolumna częstości występowania innych słów przed odpowiadającym jej słowem. Finalny wektor osadzeń powstaje dla każdego słowa w wyniku połączenia reprezentujących go wektorów kolumnowego i wierszowego.

Autorzy modelu HAL stworzyli w ten sposób macierz dla 70 000 najpopularniejszych słów w języku angielskim, wykorzystując korpus treningowy składający się z około 300 milionów słów, wiadomości zebranych z sieci Usenet. W ten sposób dla każdego wyrazu powstał 140 000-wymiarowy wektor semantyczny. Wnikliwa analiza przeprowadzona przez autorów wykazała że powstałe wektory bardzo dobrze reprezentują podobieństwo semantyczne, które określilibyśmy raczej jako paradygmatyczne, co stanowi ważną różnicę względem algorytmu LSA. Co również było bardzo interesującym odkryciem, po zredukowaniu wektorów do 100–200 wymiarów charakteryzujących się największą wariancją model nie stracił praktycznie niczego ze swojej ekspresyjności i nadal bardzo dobrze reprezentował relacje semantyczne pomiędzy słowami. Odkrycie to pokazało, że wcale nie potrzeba zbyt wielu atrybutów, aby przechowywać złożone zależności pomiędzy słowami w języku naturalnym.

1.7. Word2vec, GloVe, fastText

Pomimo pomysłowości modelu HAL dopiero pojawienie się projektu Word2vec w 2013 roku sprawiło, że semantyczne osadzenia słów stały się gorącym tematem w dziedzinie NLP [9]. Podobnie jak HAL, Word2vec również wykorzystuje okno przesuwne podczas treningu osadzeń, jednak zamiast zliczać wystąpienia okolicznych słów, wykorzystuje sieci neuronową i algorytm propagacji wstecznej.

W swoim artykule Thomas Mikolov, autor Word2vec, proponuje dwa podejścia do budowania semantycznych wektorów słów na bazie dużych korpusów tekstowych. Pierwsze z nich nazwał *skip-gram*, a drugie ciągłym BoW (*continuous bag-of-words*, CBoW).

W pierwszym podejściu do treningu osadzeń wykorzystujemy jednokierunkową gęstą sieć neuronową (*feed-forward dense neural network*) z jedną warstwą ukrytą. Liczba neuronów w warstwie ukrytej jest równa liczbie wymiarów docelowych wektorów osadzeń (podczas swoich eksperymentów Mikolov uznał że wektory 300-wymiarowe sprawdzają się najlepiej), natomiast liczba neuronów w warstwie wejściowej i wyjściowej równa jest wielkości naszego słownika. Warstwa ukryta nie posiada żadnej funkcji aktywacji, natomiast funkcją aktywacji dla warstwy wyjściowej jest *softmax*. Trening sieci neuronowej polega na przesuwaniu 5-wyrazowego okna po naszym korpusie treningowym. Słowo znajdujące się w środku okna nazywane jest słowem środkowym, a pozostałe 4 słowa przed i po nim słowami otaczającymi. Każda taka piątka tworzy nam 4 pary treningowe dla naszej sieci neuronowej – słowo środkowe jest sparowane z każdym ze słów otaczających. Następnie każda taka para jest przekształcana na dwa wektory *one-hot*, kodujące odpowiednie słowa, i przepuszczana przez sieć neuronową. Jako funkcję straty w modelu wykorzystujemy entropię krzyżową (*cross-entropy*). Trening naszej sieci kontynuujemy aż do momentu przej-

rzenia wszystkich dokumentów w naszym korpusie. Po zakończeniu treningu macierz wag pomiędzy warstwą wejściową a ukrytą stanowi nasze osadzenia, gdzie każdemu wyrazowi w słowniku odpowiada 300-elementowy wiersz naszej macierzy.

W drugim podejściu architektury sieci neuronowej wielkość okna przesuwnego oraz sposób treningu pozostają identyczne, z tą różnicą, że zamiast rozbijać każde okno na 4 pary treningowe, wykorzystujemy je jako jedną parę treningową. Na wejście sieci zamiast pojedynczego wektora *one-hot*, przekazujemy wektor zawierający 1 na pozycjach odpowiadających wszystkim 4 słowom otaczającym i 0 na pozostałych pozycjach (reprezentacja ta czasem jest nazywana *multi-hot*). Zadaniem sieci jest predykcja słowa środkowego.

Według eksperymentów przeprowadzonych przez Mikolova podejście *skip--gram* sprawdza się lepiej w przypadku małych korpusów i rzadkich terminów, natomiast *CBoW* działa efektywniej dla częstszych słów i jest znacznie szybsze do wytrenowania.

W celu utworzenia gotowych osadzeń autorzy Word2vec wykorzystali fragment korpusu Google News liczący 100 miliardów słów oraz słownik składający się z 3 milionów wyrazów. Poza pojedynczymi słowami autorzy dołączyli do słownika również często występujące wyrażenia dwuwyrazowe, takie jak np. New York, San Francisco itp.

Tym co szczególnie zaskoczyło twórców Word2vec, ale także całą społeczność zajmującą się przetwarzaniem języka naturalnego, była niezwykła skuteczność powstałych osadzeń w tzw. pytaniach o analogię. Pytania takie mają formę typu: „What is to Germany as Warsaw is to Poland?"[1] albo „What is to a woman as a king is to a man?"[2]. Dzięki wektorom Word2vec można było odpowiedzieć na takie pytania, stosując proste operacje arytmetyczne na wektorach: $v['Poland'] - v['Warsaw'] + v['Germany']$, $v['king'] - v['man'] + + v['woman']$. Dla pierwszego przypadku odpowiedzią Word2vec jest „Berlin", a dla drugiego „queen".

Sukces osadzeń Word2vec zaowocował powstaniem kilku analogicznych pomysłów, nieróżniących się co do zasady działania, jednak usprawniających pewne aspekty oryginalnego projektu. Pierwszym z nich jest schemat osadzeń GloVe, zaproponowany w 2013 roku przez badaczy z uniwersytetu Stanforda [12]. Osadzenia utworzone przez algorytm GloVe są analogiczne do tych utworzonych przez Word2vec, jednak zamiast sieci neuronowej i algorytmu spadku wzdłuż gradientu wykorzystywany jest algorytm rozkładu według wartości osobliwej (SVD) na odpowiednio przygotowanej macierzy, co umożliwia znacznie szybszy trening.

Innym interesującym algorytmem rozwijającym koncepcję zaprezentowaną w Word2vec jest opracowany w 2016 algorytm fastText [2]. Algorytm ten podczas treningu poza samym słowem przekazuje na wejście sieci słowo podzielone na 3-znakowe gramy (np. wyraz *<where>* zostałby podzielony na następujące gramy: *<wh, whe, her, ere, re>*). Umożliwia to znaczne lepsze obsłużenie rzad-

[1] Co jest dla Niemiec tym, czym Warszawa jest dla Polski?
[2] Kto jest żeńskim odpowiednikiem króla?

kich słów niż w przypadku Word2vec.

Podobnie jak w przypadku HAL, wektory utworzone przez algorytmy Word2vec, GloVe oraz fastText są bardziej zbliżone do podejścia paradygmatycznego, do podobieństwa semantycznego i jeżeli zależy nam właśnie na tym rodzaju podobieństwa, mogą one być najlepszym wyborem.

1.8. Zastosowania

Zastosowania semantycznych wektorów słów możemy podzielić na dwie główne kategorie. Pierwsza to zastosowanie bezpośrednie w zadaniach, takich jak na przykład pytanie o analogię. Druga kategoria to wykorzystanie semantycznych osadzeń słów jako elementów większego systemu. Dwoma najważniejszymi problemami należącymi do drugiej kategorii są wyszukiwanie informacji (*information retrieval*) oraz transfer wiedzy (*transfer learning*) w sieciach neuronowych.

1.8.1. Wyszukiwanie informacji

Wyszukiwanie informacji stanowi jedno z najważniejszych zagadnień wchodzących w skład przetwarzania języka naturalnego. Najczęstszym problemem rozważanym w ramach tej dziedziny stanowi tzw. wyszukiwanie *ad-hoc* (*ad-hoc retrieval*), w którym mając dany zbiór dokumentów oraz zapytanie sformułowane w języku naturalnym, zwracamy dokumenty, które najlepiej pasują do danego zapytania.

W tradycyjnym podejściu do wyszukiwania *ad-hoc* zapytanie oraz dokumenty znajdujące się w zbiorze danych reprezentujemy w postaci wektorów BoW. Dokument najlepiej dopasowany do naszego zapytania to taki, który zawiera największą liczbę wspólnych terminów.

Podejście to jednak ma swoje wady. Jedną z najbardziej widocznych jest całkowite ignorowanie faktu istnienia synonimów oraz wyrazów zbliżonych znaczeniowo. Może to prowadzić do sytuacji, że nie zwrócimy użytkownikowi dokumentu dobrze pasującego do zapytania jedynie z tego powodu, że nie zawiera on dokładnie tych samych słów, które zostały użyte w zapytaniu.

Semantyczne wektory słów pozwalają nam poradzić sobie z tym problemem. Dwiema najpopularniejszymi stosowanymi technikami są gęste wektorowe reprezentacje dokumentów (*dense vector documents representations*) oraz rozszerzanie zapytań (*query expansion*).

W pierwszym podejściu zmianie ulega sposób indeksowania dokumentów. Zamiast reprezentować je jako worek słów, traktujemy je jako zbiór wektorów semantycznych odpowiadających terminom składającym się na dokument. Jedna z popularniejszych reprezentacji wykorzystuje po prostu średnią ze wszystkich wektorów semantycznych składających się na dokument (*average word embeddings*, AWE) [8]:

$$\vec{v_d} = \frac{1}{|d|} \sum_{t_d \in d} \frac{\vec{v_{t_d}}}{\|\vec{v_{t_d}}\|} \tag{1.4}$$

gdzie d to zbiór terminów składających się na dokument, a $\vec{v_{t_d}}$ semantyczny wektor reprezentujący konkretny termin ze słownika.

W celu określenia podobieństwa pomiędzy zapytaniem a dokumentem w naszym zbiorze możemy wykorzystać podobieństwo cosinusowe pomiędzy AWE dla zapytania i dokumentu:

$$sim(q, d) = cos(\vec{v_q}, \vec{v_d}) = \frac{\vec{v_q} \cdot \vec{v_d}}{\|\vec{v_q}\|\|\vec{v_d}\|} \tag{1.5}$$

gdzie:

$$\vec{v_q} = \frac{1}{|q|} \sum_{t_q \in q} \frac{\vec{v_{t_q}}}{\|\vec{v_{t_q}}\|}$$

Technika rozszerzania zapytań jest nieco bardziej konserwatywna od gęstych wektorów dokumentów i wykorzystuje semantyczne osadzenia słów jedynie w celu rozwinięcia naszego zapytania o dodatkowe słowa, stanowiące synonimy i wyrazy bliskoznaczne dla słów użytych w zapytaniu. Następnie, po dokonaniu rozszerzenia, wykorzystywane jest standardowe dopasowanie bazujące na worku słów. Jedną z popularniejszych metryk wykorzystywanych do wybrania najlepszych kandydatów do rozszerzenia zapytania stanowi średnie podobieństwo cosinusowe [14]:

$$score(t_c, q) = \frac{1}{|q|} \sum_{t_q \in q} cos(\overrightarrow{v_{t_c}}, \overrightarrow{v_{t_q}}) \tag{1.6}$$

gdzie t_c to termin kandydujący, a q zbiór wszystkich terminów z naszego zapytania.

Należy pamiętać, że niezależnie od wykorzystywanego algorytmu, wyniki systemu wyszukiwania informacji będą tak dobre, jak dobre są wykorzystywane osadzenia. Z reguły osadzenia bazujące na tematycznym podobieństwie semantycznym, takie jak LSA, sprawdzają się lepiej w tym kontekście.

1.8.2. Transfer wiedzy

Innym ważnym zastosowaniem semantycznych osadzeń słów jest wykorzystanie ich w sieciach neuronowych w procesie tzw. transferu wiedzy (*transfer learning*) na potrzeby różnych problemów z zakresu NLP, takich jak klasyfikacja czy tłumaczenie maszynowe.

Sieć neuronowa, na najbardziej ogólnym poziomie, stanowi pewną funkcję mapującą wejściowy wektor liczb rzeczywistych na wektor wyjściowy. Aby sieć neuronowa mogła służyć do przetwarzania języka naturalnego, musimy opracować mechanizm tłumaczenia sekwencji wyrazów na wektory liczb rzeczywistych. Jednym z prostszych rozwiązań jest sprowadzenie sekwencji wejściowej do wektora BoW i rozważanie jej z pominięciem kolejności wyrazów, jednak rozwiązania takie jak konwolucyjne sieci neuronowe (*convolutional neural networks*, CNN) czy sieci rekurencyjne (*recurrent neural networks*, RNN) umożliwiają przetwarzanie danych wejściowych jako sekwencji wektorów reprezentujących poszczególne słowa w słowniku.

Sekwencję kolejnych słów wejściowych dla wektora najprościej możemy reprezentować jako sekwencję wektorów *one-hot* o długości równej wielkości słownika, jednak uniemożliwia to sieci jakościowe odróżnienie zbliżonych do siebie słów. Rozwiązanie bardziej wyrafinowane polega na zastosowaniu tzw. warstwy osadzającej (*embedding layer*), trenowanej równolegle z całą siecią, która reprezentuje nasze słowa w postaci wektorów gęstych.

Jednak w przypadku, kiedy w procesie uczenia nadzorowanego nie posiadamy zbyt wielu danych treningowych, równoległe uczenie warstwy osadzającej z całą siecią może nie dać zadowalających rezultatów. W tym wypadku pomocna może okazać się właśnie technika transferu wiedzy. Polega ona na wykorzystaniu gotowych, przetrenowanych osadzeń słów jako wag dla warstwy osadzającej w naszym modelu. Dzięki temu zabiegowi wektory, które trafiają na wejście modelu, pozwalają sieci odróżnić słowa semantycznie zbliżone i tym samym osiągać lepsze rezultaty.

1.9. Podsumowanie

W niniejszym rozdziale przedstawiono ideę semantycznego podobieństwa słów, metody umożliwiające tworzenie wysokowymiarowych wektorów pozwalające reprezentować to podobieństwo w przestrzeni wektorowej oraz najważniejsze rozwiązania wykorzystujące tego typu reprezentacje.

Przetwarzanie i reprezentowanie języka naturalnego jest bardzo trudnym tematem i stanowi dziedzinę, w której stale prowadzone są intensywne badania. Na pewno czeka nas tutaj jeszcze wiele odkryć i zmian. Prawdopodobnie jednak idea semantycznych osadzeń słów zawsze będzie tu odgrywać istotną rolę.

Bibliografia

[1] Athiwaratkun B., Wilson A.G., Anandkumar A. *Probabilistic fasttext for multi-sense word embeddings. arXiv preprint arXiv:1806.02901*, 2018.

[2] Bojanowski P., et al. *Enriching Word Vectors with Subword Information. arXiv preprint arXiv:1607.04606*, 2016.

[3] Burgess C. *From simple associations to the building blocks of language: Modeling meaning in memory with the HAL model. Behavior Research Methods, Instruments, & Computers*, 30(2):188–198, 1998.

[4] Chiarello C., et al. *Semantic and associative priming in the cerebral hemispheres: Some words do, some words don't... sometimes, some places. Brain and language*, 38(1):75–104, 1990.

[5] Deerwester S., et al. *Indexing by latent semantic analysis. Journal of the American society for information science*, 41(6):391–407, 1990.

[6] Firth J.R. *A synopsis of linguistic theory, 1930-1955. Studies in linguistic analysis*, 1957.

[7] Kiela D., Clark S. *A systematic study of semantic vector space model parameters.* In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*, pp. 21–30. 2014.

[8] Le Q., Mikolov T. *Distributed representations of sentences and documents.* In *International conference on machine learning*, pp. 1188–1196. PMLR, 2014.

[9] Mikolov T., et al. *Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781*, 2013.

[10] Mitra B., Craswell N., et al. *An introduction to neural information retrieval.* Now Foundations and Trends Boston, MA, 2018.

[11] Neely J.H. *Semantic priming effects in visual word recognition: A selective review of current findings and theories. Basic processes in reading*, pp. 264–336, 1991.

[12] Pennington J., Socher R., Manning C.D. *Glove: Global vectors for word representation.* In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543. 2014.

[13] Peters M.E., et al. *Deep contextualized word representations. arXiv preprint arXiv:1802.05365*, 2018.

[14] Roy D., et al. *Using word embeddings for automatic query expansion. arXiv preprint arXiv:1606.07608*, 2016.

Rozdział 2

Zastosowanie grafowych sieci neuronowych w NLP

Wojciech Janowski[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
wojciech.janowski@pg.edu.pl

Abstrakt

W tym rozdziale zostaną zaprezentowane i omówione podstawowe pojęcia związane z grafami oraz grafowymi sieciami neuronowymi. Dokładniejszej analizie będą poddane sieci grafowe z warstwami konwolucyjnymi i atencją. Na koniec zostanie wyjaśniona koncepcja tworzenia wektorów modelujących konteksty węzłów w grafie (tzw. *node embeddings*). Opisane metody będą poparte różnymi przykładami zastosowań w rozwiązywaniu praktycznych problemów.

2.1. Wprowadzenie

W ostatnich latach dziedzina sztucznej inteligencji, jaką jest przetwarzanie języka naturalnego (*Natural Language Processing*, NLP), bardzo dynamicznie się rozwija. Zajmuje się ona zagadnieniami analizy języka naturalnego. Dzięki systemom NLP programy potrafią interpretować ludzką mowę i pismo, mogą w odpowiedni sposób reagować na wykryte nacechowanie emocjonalne. Można w ten sposób tworzyć w pełni autonomiczne *chatboty* lub *voiceboty* potrafiące

komunikować się z człowiekiem za pomocą tekstu lub głosu. Komponenty NLP są wykorzystywane również do wykrywania spamu, oceniania plagiatów lub kategoryzowania dokumentów.

Do tej pory w problematyce związanej z NLP korzystało się głównie z głębokich sieci neuronowych bazujących na sieciach rekurencyjnych. Różne architektury rekurencyjnych sieci neuronowych bardzo dobrze radzą sobie z modelowaniem długiego kontekstu wypowiedzi. Obecne rozwiązania typu *state-of-the-art* (SOTA) starają się tworzyć generyczne, wielojęzyczne modele bazujące na sieciach neuronowych typu *transformer* [5]. Generyczne modele wytrenowane na wielkich korpusach tekstowych są później dotrenowywane w sposób nadzorowany na mniejszej ilości danych pod konkretne zadanie, takie jak wykrywanie sentymentu czy rozpoznawanie intencji. Tego typu podejście nazywamy treningiem *downstream task*.

Innym, stosunkowo nowatorskim, podejściem w rozwiązywaniu problemów NLP jest zastosowanie grafowych sieci neuronowych. Rodzaje oraz sposób działania poszczególnych architektur grafowych sieci są główną tematyką niniejszego rozdziału monografii.

2.2. Grafowe sieci neuronowe

W tym podrozdziale, po krótkim wprowadzeniu do podstaw teorii grafów, zostaną szczegółowo omówione poszczególne architektury grafowych sieci neuronowych.

2.2.1. Teoria grafów w informatyce

Formalnie grafem G nazywamy dwójkę zbiorów wierzchołków (węzłów) V i krawędzi E:

$$G = (V, E)$$

Zbiór krawędzi E jest zbiorem par wierzchołków $v_i \in V$:

$$E = \{(v_i, v_j); v_i, v_j \in V; i, j \in \mathbb{N}\}$$

Jeśli elementy zbioru E są parami uporządkowanymi (rozróżniamy kolejność elementów v_i), mówimy że krawędzie grafu G są skierowane. W takim przypadku możemy stwierdzić, który węzeł krawędzi jest początkowy, a który końcowy. Jeśli dopuszczamy istnienie krawędzi wielokrotnych (pomiędzy dwoma wierzchołkami może występować więcej niż jedna krawędź), graf G nazywamy multigrafem, w przeciwnym wypadku G jest grafem prostym.

Zarówno do krawędzi, jak i do wierzchołków możemy przypisać dodatkowe informacje nazywane wagami lub etykietami. Dzięki temu struktura grafu może kodować dodatkowe informacje. Stosując grafy, efektywnie modeluje się zaawansowane struktury i relacje między nimi, np. topologię sieci komputerowej lub strukturę przestrzenną związków chemicznych.

Reprezentacja grafu

Jedną z metod reprezentacji grafów jest tak zwana *macierz sąsiedztwa*. Macierzą sąsiedztwa grafu G nazywamy macierz kwadratową A, w której elementy a_{ij} oznaczają liczbę krawędzi prowadzących od wierzchołka i do wierzchołka j. Taka metoda reprezentacji pozwala w efektywny sposób przeprowadzać różne skomplikowane obliczenia na grafach. W celach optymalizacyjnych wykorzystuje się *macierze rzadkie*. Dzięki temu przetwarzanie macierzy jest szybsze i wymaga mniejszej ilości pamięci.

2.2.2. Podstawowe zagadnienia związane z grafowymi sieciami neuronowymi

W tym podrozdziale zostaną omówione krótko podstawowe zagadnienia, w których wykorzystuje się grafowe sieci neuronowe. Pozwoli to lepiej zrozumieć zasady działania poszczególnych architektur grafowych sieci.

Klasyfikacja węzłów

Zagadnienie to polega na klasyfikacji poszczególnych węzłów w grafie wejściowym. Na wyjściu otrzymujemy niezmienioną strukturę grafu z zaetykietowanymi węzłami. Podobnym zagadnieniem jest problem uzupełniania etykiet w niektórych węzłach. W takim przypadku na wejściu mamy częściowo zaanotowany graf. Tego typu problemy często występują w tematyce związanej z badaniem sieci społecznościowych.

Klasyfikacja grafu

Ogólniejszym tematem jest problem klasyfikacji całego grafu. W takim przypadku na wejściu otrzymujemy strukturę grafu. Celem zadania jest przypisanie określonych etykiet całej strukturze grafu. Zagadnienie to bardzo często przewija się w modelowaniu procesów fizycznych i chemicznych. Przykładowym problemem jest przewidzenie właściwości fizykochemicznych danego związku chemicznego na podstawie jego struktury, np. w procesie tworzenia nowych leków. Strukturę związku chemicznego można traktować jako graf. Doświadczalne badanie właściwości związków chemicznych jest bardzo czasochłonne, użycie w tym celu dedykowanych sieci neuronowych ułatwia pracę badaczom. W NLP można w ten sposób rozwiązywać problemy związane z sieciami społecznościowymi lub analizą dokumentów. Istnieje możliwość zamodelowania struktury dokumentu tekstowego za pomocą grafu. Dzięki temu jest możliwe zamienienie problemu klasyfikacji dokumentu na problem klasyfikacji grafu.

Dodawanie i usuwanie krawędzi

W tym przypadku celem jest stworzenie brakujących połączeń pomiędzy węzłami grafu. Na wejściu otrzymujemy graf, w którym brakuje pewnej liczby krawędzi. Łatwo można stworzyć podobne zadanie, które polega na usunięciu

z grafu wejściowego nadmiarowych krawędzi. Oba te zagadnienia można wykorzystać do optymalizowania struktury istniejącego już grafu. Problem jest podobny do minimalizowania automatów.

Generowanie grafu

Zadanie można traktować jako odwrócony problem klasyfikacji grafu. Celem jest wygenerowanie odpowiedniej struktury grafu na podstawie zestawu danych wejściowych. Doskonałym zobrazowaniem problemu jest zagadnienie stworzenia struktury związku chemicznego na podstawie żądanych właściwości. Jak większość problemów związanych z generowaniem treści przez sztuczną inteligencję, w praktyce zagadnienia tego typu są bardzo skomplikowane do rozwiązania.

2.2.3. Grafowe sieci konwolucyjne (GCN)

Klasyczne konwolucyjne sieci neuronowe (*convolutional neural networks*, CNN) [4] są wykorzystywane przede wszystkim w rozpoznawaniu obrazów. Za pomocą zestawu operacji splotowych poszczególne warstwy sieci tworzą i interpretują coraz bardziej abstrakcyjne zestawy cech danych wejściowych. Nazywa się to agregacją cech. Bardzo ważną zaletą sieci konwolucyjnych jest współdzielenie pomiędzy poszczególnymi warstwami wag filtrów używanych w operacjach splotowych (*shared-weight architecture*). Dzięki temu liczba parametrów sieci konwolucyjnej jest wiele rzędów wielkości mniejsza niż w klasycznej, gęstej sieci neuronowej.

Rozważania dotyczące grafowych sieci konwolucyjnych zaczniemy od pomysłu na bezpośrednie wykorzystanie zwykłych sieci konwolucyjnych do danych o strukturze grafowej. Niestety takie podejście nie sprawdzi się przy większości grafów. Ze względu na bardziej skomplikowaną topologię grafu niż struktura obrazów (siatkę pikseli możemy traktować jako graf, piksele są węzłami, krawędzie łączą ze sobą sąsiednie piksele) zwykłe filtry konwolucyjne nie są w stanie poprawnie wydobywać wszystkich cech zawartych w poszczególnych węzłach grafu. Aby stworzyć grafowy odpowiednik grafowych sieci konwolucyjnych, wprowadzono kilka modyfikacji do metod obliczenia filtrów.

Konwolucje spektralne

Tak zwane spektralne grafowe sieci konwolucyjne (*spectral graph convolutional networks*) zostały przedstawione w artykule poświęconym problemowi klasyfikacji węzłów za pomocą grafowych sieci konwolucyjnych [2]. Idea rozwiązania jest prosta. Chcemy stworzyć metodę agregacji cech, która dla każdego węzła zaagreguje cechy sąsiednich węzłów oddalonych maksymalnie o jeden krok. Dodatkowo wymagamy, żeby nasza funkcja spełniała warunek współdzielenia parametrów poszczególnych fragmentów grafu.

Macierz X oznacza macierz cech, gdzie x_i jest wektorem cech węzła i. Macierz X ma rozmiar $N \times D$, gdzie N oznacza liczbę węzłów w grafie, a D oznacza długość wektora cech każdego węzła. Przez macierz A rozumiemy macierz

sąsiedztwa naszego grafu wejściowego. Jako że problem dotyczy klasyfikacji węzłów, to na wyjściu oczekujemy macierzy Z o wymiarach $N \times F$, gdzie F oznacza długość wektora cech wyjściowych przypisanych do danego węzła. Ogólnie warstwę sieci neuronowej możemy przedstawić jako równanie:

$$H^{(l+1)} = f(H^{(l)}, A)$$

gdzie $H^{(0)} = X$, $H^L = Z$, L oznacza liczbę warstw w naszej sieci neuronowej. Idąc dalej, możemy przedstawić funkcję f jako funkcję propagacji $\sigma(\cdot)$:

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

gdzie funkcja $\sigma(\cdot)$ oznacza dowolną nieliniową funkcję agregacji, a przez macierz $W^{(l)}$ rozumiemy macierz wag warstwy l. Jak zauważyli autorzy pracy [2], z taką definicją wiąże się kilka problemów. Przy zastosowaniu mnożenia z macierzą A zostaną zamodelowane cechy sąsiednich węzłów, ale nie same węzły. By temu zapobiec, sztucznie dodaje się do każdego węzła krawędź prowadzącą do niego samego (tzw. *self-loops*). Można to osiągnąć w prosty sposób, dodając do macierzy A macierz jednostkową. Można to zapisać w postaci: $\widehat{A} = A + I$. Kolejną modyfikacją poprawiającą skuteczność sieci jest dodanie normalizacji. Bez niej wielokrotne mnożenia macierzy A mogą spowodować niestabilność rozwiązania. Znormalizowaną macierz $\widehat{A}$ oznaczmy symbolem $\widehat{B}$. W ten sposób powstaje ostateczna definicja równania grafowej sieci konwolucyjnej:

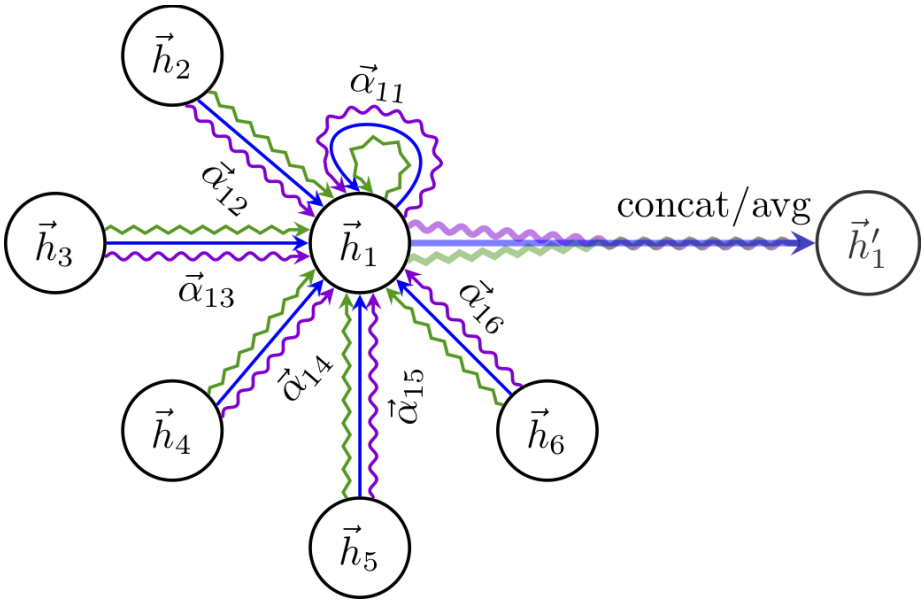
$$f(H^{(l)}, A) = \sigma(\widehat{B}H^{(l)}W^{(l)})$$

2.2.4. Grafowe sieci z atencją (GAT)

Mechanizm atencji (inaczej mechanizm uwagi) [1] zrewolucjonizował działanie głębokich sieci neuronowych. Dzięki niemu sieć neuronowa potrafi lepiej modelować długie konteksty. Sieć neuronowa potrafi się nauczyć, które fragmenty danych wejściowych mają decydujący wpływ na poprawny wynik predykcji. Pozwala to znacznie poprawić osiągane rezultaty w takich zagadnieniach, jak tłumaczenie maszynowe, modelowanie języka, generowanie mowy. Mechanizm atencji, który skupia się na analizowaniu danych wejściowych sieci neuronowej, nazywamy *self-attention*. Popularne modele typu *transformer* [5] bazują na architekturze zawierającej dużo warstw z atencją. Inne niewątpliwe zalety atencji to niska złożoność obliczeniowa i łatwa możliwość zrównoleglenia obliczeń.

Omówimy pomysł na implementację atencji w grafowych sieciach neuronowych zaproponowany w artykule [6]. Idea działania atencji w przypadku grafowych sieci nie zmienia się. Celem jest lepsze modelowanie kontekstu na podstawie sąsiednich węzłów. Podobnie jak w sieciach GCN, atencja będzie agregować dane z sąsiadów. Tak jak poprzednio, na wejściu warstwy mamy macierz cech wejściowych:

$$H = \{\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_N}\}, \overrightarrow{h_N} \in \mathbb{R}^F$$



Rysunek 2.1: Ilustracja atencji z trzema głowami. Kolory strzałek reprezentują poszczególne „głowy". Na końcu wyniki trzech głów są agregowane do $\overrightarrow{h'_1}$ [6]

gdzie N oznacza liczbę węzłów, a F długość wektora cech. Na wyjściu otrzymujemy nowe wartości oznaczone jako H' o wymiarach $N \times F'$. Żeby warstwa mogła się uczyć, należy dodać do każdego węzła macierz wag $W \in \mathbb{R}^{F' \times F}$. Definiujemy funkcję atencji $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbf{R}$, która oblicza tzw. *współczynniki atencji*:

$$e_{ij} = a(W\overrightarrow{h_i}, W\overrightarrow{h_j})$$

Współczynnik e_{ij} decyduje, jak bardzo cechy węzła j są ważne dla węzła i w kontekście predykcji wyniku. Autorzy artykułu [6] dodatkowo dodali mechanizm *maskowanej atencji* (*masked attention*). Polega to na tym, że przy obliczaniu atencji brany jest losowy podzbiór sąsiadów węzła i. Dla lepszej optymalizacji działania sieci współczynniki e_{ij} są normalizowane za pomocą funkcji *softmax*.

Dla lepszego modelowania kontekstu zaimplementowano również metodę *multi-head attention*. W obliczeniach jest używane wiele „głów atencji", z których każda ma swoje własne wagi. Na samym końcu wyjścia z poszczególnych głów są sumowane lub uśredniane. Pokazuje to rys. 2.1.

2.3. Reprezentacje wektorowe węzłów

W tym podrozdziale zostaną omówione przykładowe metody, jak efektywnie modelować kontekst w grafach, tworząc tzw. *node's embeddings*. W przypadku modelowania języka powszechną praktyką jest kodowanie słów lub dłuższych

wypowiedzi do postaci wektorowej. Celem jest przyporządkowanie każdej frazie wejściowej wektora o określonym wcześniej rozmiarze. Dodatkowo jest wymagane, żeby powstała w ten sposób przestrzeń wektorowa jak najlepiej modelowała kontekst wypowiedzi. Przykładowo wektory powinny być addytywne:

$$v_{\text{król}} - v_{\text{mężczyzna}} + v_{\text{kobieta}} = v_{\text{królowa}}$$

Dobrze stworzone kodowanie danych na wektory pozwala lepiej modelować cechy tychże danych, np. kontekst, w jakim występuje dane słowo. Znacząco zwiększa to skuteczność sieci neuronowych.

W przypadku węzłów w grafie sprawa nieco się komplikuje. Celem jest przypisanie wektora każdemu węzłowi w grafie, który będzie kodował cechy danego węzła, jak również modelował kontekst, w jakim występuje dany węzeł. Kontekst węzła jest rozumiany jako relacje z sąsiadami danego węzła. Sąsiedztwo jest dzielone na *sąsiedztwo lokalne* i *sąsiedztwo globalne*. Sąsiedztwo lokalne oznacza bezpośrednich sąsiadów wskazanego węzła. Natomiast sąsiedztwo globalne oznacza węzły oddalone o więcej niż jedną krawędź. Odpowiednio wyróżniamy *kontekst globalny* i *kontekst lokalny*. Dobre odwzorowanie wektorowe powinno modelować oba te konteksty.

2.3.1. Metoda DeepWalk

Jest to prosta metoda obliczania wektorów węzłów. Opiera się ona na algorytmie *random walk*, czyli przemieszczaniu się po grafie określoną liczbę kroków w sposób losowy, zapamiętując odwiedzane węzły. Każde takie przejście generuje ciąg węzłów, który nazywa się *ścieżką*. Algorytm *DeepWalk* polega na tym, że z danego węzła jest generowana określona liczba ścieżek o stałej długości. Każdą taką ścieżkę można traktować jak standardowe „zdanie". Podczas treningu na zbiorze takich ścieżek za pomocą metody *skip-gram* [3] powstanie model kodujący kontekst poszczególnych węzłów w postaci wektorów. Chcąc obliczyć wektor danego węzła, należy zaagregować wektory wszystkich ścieżek zaczynających się w wybranym węźle.

2.3.2. Metoda Node2vec

Niestety poprzednie rozwiązanie ma pewne braki. Sposób generowania ścieżek jest całkowicie losowy, stąd algorytm DeepWalk ma problemy z poprawnym modelowaniem lokalnego kontekstu. Metoda Node2vec jest modyfikacją DeepWalk, która udoskonala modelowanie lokalnego kontekstu. Dodatkowe parametry P i Q pozwalają kontrolować eksplorację grafu podczas generowania ścieżek. Parametr P określa prawdopodobieństwo powrotu metody DeepWalk do wcześniej odwiedzonego węzła. W ten sposób parametr P kontroluje lokalne sąsiedztwo. Natomiast parametr Q definiuje prawdopodobieństwo, że algorytm „odkryje" nową, nieodwiedzoną część grafu podczas eksploracji. W ten sposób jest kontrolowane globalne sąsiedztwo.

2.3.3. Structural deep network embedding (SDNE)

Jest to bardziej zaawansowane podejście zaproponowane w artykule [7]. Rozwiązanie całkowicie rezygnuje z losowego generowania ścieżek. Dzięki temu metoda jest bardziej stabilna niż poprzednie algorytmy. Pomysł opiera się na wykorzystaniu tzw. *syjamskich sieci neuronowych*. Są to dwie identyczne pod względem budowy sieci, które współdzielą część wag podczas treningu i inferencji. Najpierw omówimy dwie definicje, zaproponowane przez autorów artykułu.

Podobieństwo pierwszego rodzaju (first-order proximity)

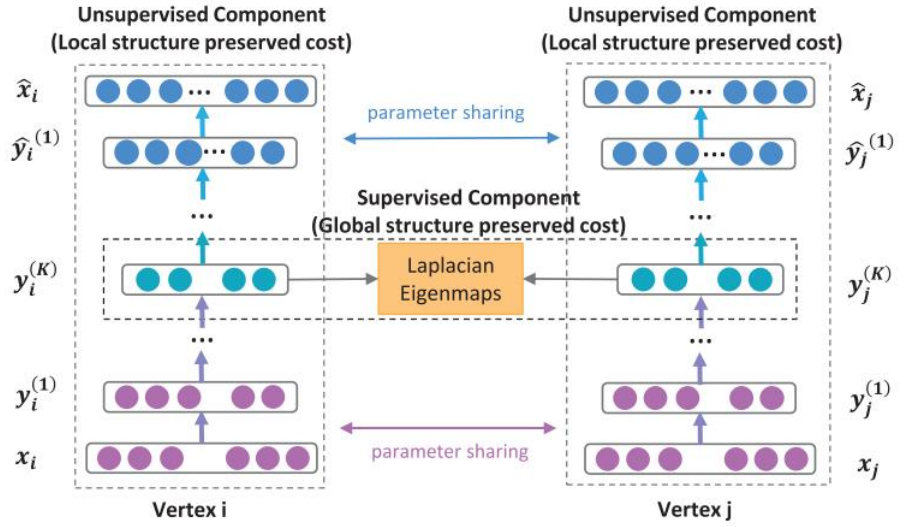
Podobieństwo pierwszego rodzaju reprezentuje stopień podobieństwa pomiędzy parami wierzchołków. Jeśli dwa wierzchołki są ze sobą połączone krawędzią, to podobieństwo pierwszego rodzaju ma wartość dodatnią, w przeciwnym wypadku wynosi 0. Podobieństwo pierwszego rodzaju można zobrazować w prosty sposób: jeśli jeden artykuł cytuje drugi, to znaczy, że oba powinny mieć podobną tematykę. Niestety w rzeczywistym świecie grafy nie zawsze posiadają wszystkie krawędzie, w konsekwencji istnieją podobne węzły, które nie są połączone krawędzią. Dlatego, żeby móc w pełni modelować kontekst węzłów, zdefiniowano inny rodzaj podobieństwa.

Podobieństwo drugiego rodzaju (second-order proximity)

Podobieństwo drugiego rodzaju wykorzystuje się do modelowania podobieństwa pomiędzy niepołączonymi ze sobą węzłami. Nazywamy to lokalną strukturą sąsiedztwa (*neighborhood structure*). Niech $\mathbf{N_u}$ oznacza zbiór podobieństw pierwszego rzędu pomiędzy wierzchołkiem u a resztą wierzchołków w grafie. Podobieństwem drugiego rodzaju pomiędzy wierzchołkami u i v nazywamy stopień podobieństwa pomiędzy $\mathbf{N_u}$ i $\mathbf{N_v}$. Taką zależność można zinterpretować w prosty sposób: jeśli dwa węzły w grafie mają wiele wspólnych sąsiadów, to węzły powinny być do siebie podobne. Przykładowo jeśli dwa słowa występują w zbliżonych kontekstach, to uznaje się je za podobne. Połączenie obu metryk pozwala skutecznie modelować relacje i kontekst w grafie.

Architektura sieci SDNE

Rozwiązanie składa się z dwóch sieci typu autoenkoder oraz jednej mniejszej sieci neuronowej, która łączy autoenkodery. Celem sieci autoenkoder jest nauczenie się w sposób nienadzorowany modelowania podobieństwa drugiego rodzaju. Na wejściu autoenkoder otrzymuje wektor sąsiedztwa dla danego węzła (wycinek macierzy sąsiedztwa, reprezentujący połączenia z sąsiednimi węzłami). Fragment modelu będący dekoderem ma za zadanie odtworzyć poprawnie wektor sąsiedztwa, minimalizując błąd rekonstrukcji. Jako że macierz sąsiedztwa jest zazwyczaj rzadka (liczba zerowych elementów jest znacznie większa w stosunku do elementów niezerowych) wprowadzono dodatkową karę za odtwarzanie zerowych elementów.



Rysunek 2.2: Architektura sieci SDNE, dwa autoenkodery połączone mniejszą siecią [7]

Wejściem obu autoenkoderów będą pary wierzchołków, które są ze sobą połączone krawędzią. Wagi poszczególnych warstw sieci są współdzielone pomiędzy sobą. Autoenkodery są połączone dodatkową siecią, trenowaną w sposób nadzorowany. Łącznik ma za zadanie modelować podobieństwo pierwszego rodzaju poprzez minimalizowanie odległości pomiędzy wektorami. Architekturę sieci przedstawiono na rys. 2.2. Jako że autoenkodery przetwarzają pary połączonych węzłów, ich wektory powinny być blisko siebie. W tym celu używa się np. algorytmu opartego na metodzie *Laplacian Eigenmaps* (metoda jest również wykorzystywana w algorytmach redukujących wymiary danych, takich jak PCA). Całkowity błąd sieci jest sumą błędów poszczególnych komponentów. Dodatkowo w celu zmniejszenia ryzyka przetrenowania błąd jest normalizowany za pomocą normy $L2$.

2.4. Podsumowanie

W rozdziale zaprezentowano i omówiono różne podejścia praktycznego zastosowania grafowych sieci neuronowych w problematyce związanej z przetwarzaniem języka naturalnego. Na podstawie konkretnych zagadnień wytłumaczono funkcjonowanie podstawowych algorytmów związanych z grafowymi sieciami neuronowymi. Przedstawiono między innymi metody tworzenia grafowych sieci konwolucyjnych, w tym konwolucje spektralne. Następnie wyjaśniono metody implementacji mechanizmu atencji w grafowych sieciach neuronowych oraz za-

awansowane sposoby generowania reprezentacji wektorowych węzłów.

Pomimo wielu problemów natury algorytmicznej grafowe sieci neuronowe zyskują coraz większą popularność wśród badaczy NLP. Jest to nowatorskie podejście, w niektórych zadaniach przewyższające klasyczne głębokie sieci neuronowe.

Bibliografia

[1] Bahdanau D., Cho K., Bengio Y. *Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473*, 2014.

[2] Kipf T.N., Welling M. *Semi-Supervised Classification with Graph Convolutional Networks.* In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17. 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

[3] Mikolov T., et al. *Distributed Representations of Words and Phrases and Their Compositionality.* In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, p. 3111–3119. Curran Associates Inc., Red Hook, NY, USA, 2013.

[4] O'Shea K., Nash R. *An Introduction to Convolutional Neural Networks. CoRR*, abs/1511.08458, 2015. URL `http://dblp.uni-trier.de/db/journals/corr/corr1511.html#OSheaN15`.

[5] Vaswani A., et al. *Attention is All you Need.* In I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, eds., *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[6] Veličković P., et al. *Graph Attention Networks. 6th International Conference on Learning Representations*, 2017.

[7] Wang D., Cui P., Zhu W. *Structural Deep Network Embedding.* In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, p. 1225–1234. Association for Computing Machinery, New York, NY, USA, 2016. ISBN 9781450342322. URL `https://doi.org/10.1145/2939672.2939753`.

Rozdział 3

Wielojęzyczny transfer wiedzy

Adam Wawrzyński[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
wawrzynski.adam@pg.edu.pl

Abstrakt

Rozdział prezentuje zjawisko zwane transferem wiedzy (*transfer learning*), motywacje stojące za jego wykorzystywaniem oraz problemy z nim związane. W początkowej części opisane zostały sposoby nienadzorowanego uczenia reprezentacji słów oraz zdań, które są wykorzystywane przy transferze wiedzy w zadaniach przetwarzania języka naturalnego. Następnie przedstawione zostały typy transferu: między zadaniami, między językami, domenami oraz transfer stylu. W kolejnej części przedstawiono różne typy wykorzystania transferu wiedzy. Opisano *zero-shot*, *few-shot learning* oraz nadzorowane dotrenowanie jako standardowe sposoby na wykorzystanie pretrenowanych modeli. W dalszej części omówione zostały wybrane techniki: stopniowe odmrażanie warstw, wielozadaniowość, *prompting* oraz zastosowanie adapterów. W kontekście adapterów przedstawiony został przykład wpływu ich zastosowania w zadaniu generowania streszczeń.

Słowa kluczowe: transfer wiedzy, przetwarzanie języka naturalnego, wielojęzyczność

3.1. Transfer wiedzy

Transfer wiedzy jest metodą, która pozwala na ponowne wykorzystanie modelu uczenia maszynowego do innego zadania niż to, na którym pierwotnie model został nauczony. Dzięki temu nie ma konieczności powtarzania kosztownych obliczeń za każdym razem, gdy opracowywany jest nowy model lub system. Metoda ta pozwala na tworzenie uniwersalnych modeli, które zostały opracowane w taki sposób, że są w stanie tworzyć bogate reprezentacje dla przetwarzanych danych, które następnie mogą zostać wykorzystane jako punkt startowy do opracowania systemów rozwiązujących konkretne zadanie. W przypadku przetwarzania języka naturalnego takie uniwersalne modele służą do generowania reprezentacji dla pojedynczych tokenów, wyrazów lub całych fragmentów tekstu. Bazując na uniwersalnych modelach, możemy je douczyć do rozwiązywania konkretnego zadania, przykładowo klasyfikacji tekstów, lub wykorzystać tworzone przez nie reprezentacje w innych systemach.

Głównym założeniem tej metody jest uniwersalność modeli, które są wykorzystywane jako punkt startowy. Reprezentacje te powinny zawierać jak najwięcej informacji o modelowanych danych, czyli powinny zostać stworzone na bazie możliwie największych zbiorów danych. W tym celu opracowane zostały nienadzorowane zadania, które umożliwiają wykorzystanie dużych nieoznaczonych korpusów tekstów do procesu uczenia.

3.1.1. Reprezentacje słów

Algorytmy Word2vec [20] oraz fastText [5] służą do tworzenia zanurzeń słów na podstawie nieoznaczonego korpusu tekstów. Obie metody wykorzystują założenie, że reprezentacja jednego słowa może zostać opisana za pomocą słów w jego sąsiedztwie. Algorytm fastText dodatkowo, zamiast analizować tekst na podstawie słów, dzieli je na zbiór znakowych n-gramów, dzięki czemu jest bardziej odporny na błędy w pisowni oraz występowanie słów spoza słownika.

Modele te w procesie uczenia wykorzystują duże zbiory nieoznaczonych korpusów tekstowych, dzięki czemu są w stanie w swoich parametrach zakodować dużą ilość informacji. Reprezentacje wyrazów są punktami w wielowymiarowej przestrzeni, które zachowują między sobą relacje umożliwiające wykonywanie na nich operacji znanych z algebry liniowej. Autorzy [20] zaprezentowali, że dodanie do siebie wektorów „Germany" oraz „capital" daje reprezentację najbardziej zbliżoną do słowa „Berlin". Zakodowane informacje dobrze odwzorowują semantyczne relacje między słowami, co pozwala na przeprowadzenie wnioskowania oraz szukanie analogii na bazie znanych przykładów, a także obliczanie podobieństwa między wyrazami.

Reprezentacje słów są skutecznie używane do zadań na poziomie słów, ale są niewystarczające w przypadku przetwarzania zdań lub dłuższych dokumentów. W takich przypadkach dawniej stosowane były warstwy rekurencyjnych sieci neuronowych, takich jak LSTM [26] lub GRU [6], których wejściem były zanurzenia Word2vec lub fastText dla przetwarzanego tekstu. Tak zastosowany transfer wiedzy pozwala na wykorzystanie gotowych reprezentacji wyrazów oraz

nauczenie sieci jedynie odwzorowywania relacji między nimi. Skutkuje to skróceniem czasu oraz zmniejszeniem zużycia zasobów podczas rozwijania modeli. Obecnie do dłuższych tekstów z powodzeniem wykorzystywane są sieci neuronowe oparte na architekturze *transformer* [28], przykładowo BERT [8] lub GPT [24].

3.1.2. Reprezentacje zdań

Do kontekstowego modelowania dłuższych tekstów opracowany został model BERT (*Bidirectional Encoder Representations from Transformers*) [8]. Jest to architektura sieci neuronowej opartej wyłącznie na koderach typu *transformer* służącej do przetwarzania zdań oraz paragrafów. Dzięki zastosowaniu mechanizmu *multihead attention* [28] pozwala na efektywniejsze modelowanie ciągów danych. Dzięki nienadzorowanemu procesowi uczenia możliwe było wykorzystanie dużego korpusu tekstów, przez co zakodowane kontekstowe reprezentacje wyrazów są w stanie poprawnie odwzorować wiedzę nabytą podczas uczenia na bardziej różnorodnych kontekstach.

Proces uczenia obejmuje dwa zadania. Pierwszym z nich jest maskowane modelowanie języka (*masked language modelling*), które polega na odtworzeniu przez model oryginalnego tekstu z pewną liczbą zamaskowanych tokenów. Drugim zadaniem jest przewidywanie kolejnego zdania – model na wejście dostaje dwa zadania i musi zdecydować binarnie, czy zdania te występują po sobie, czy też są przypadkowe.

Główną różnicą i przewagą modelu BERT nad Word2vec jest kontekstowe modelowanie tekstów, co pozwala na uchwycenie bardziej skomplikowanych relacji między słowami oraz zdaniami. Model ten, tak jak Word2vec, również posiada warstwę zamieniającą przetwarzany tekst na ich zanurzenia. Różnica polega na tym, że poza tą warstwą architektura składa się z wielu warstw, które są nauczone dokonywania specjalistycznych transformacji pojedynczych tokenów w taki sposób, że są w stanie wydobyć dużo informacji pod kątem składniowym oraz semantycznym. Dzięki tej przewadze modele oparte na tej architekturze umożliwiły poprawę wyników dla szerokiej gamy zadań przetwarzania języka naturalnego.

3.2. Motywacje

Podrozdział ten poświęcony został motywacjom stojącym za popularnością wykorzystywania zjawiska transferu wiedzy.

3.2.1. Poprawa wyników

Transfer wiedzy umożliwia wykorzystanie dużych zbiorów nieoznaczonych danych do tego, aby zakodować możliwie najwięcej informacji w parametrach modelu. Taki model jest w stanie generować zanurzenia wyższej jakości oraz przetwarzać dane w sposób, który umożliwia modelowanie bardziej skomplikowanych

relacji i wydobycie większej ilości informacji. Pozwala to uzyskać lepsze wyniki w zadaniach docelowych.

3.2.2. Oszczędność czasu i zasobów

Zastosowanie techniki transferu wiedzy pozwala na wykorzystanie gotowych modeli bez konieczności ich uczenia, co ogranicza czas oraz zasoby potrzebne do rozwiązania postawionego zadania. Dzięki temu skrócony został czas trwania eksperymentów, co przyczyniło się do przyspieszonego rozwoju dziedziny głębokiego uczenia i sztucznej inteligencji.

3.2.3. Małe zbiory danych

W przypadku małych zbiorów danych trening modelu może powodować przetrenowanie, niestabilność treningu oraz niską skuteczność i generalizację. Dzięki zastosowaniu transferu wiedzy jesteśmy w stanie wykorzystać wiedzę zakodowaną w modelach otrzymanych w wyniku nienadzorowanego uczenia na dużych zbiorach danych i jedynie dostroić model na małej próbce oznaczonych danych.

3.3. Problemy

Transfer wiedzy ma wiele zalet, ale nie jest pozbawiony wad. W tym podrozdziale zostały opisane dwa zjawiska, które należy wziąć pod uwagę przy analizowaniu wielojęzycznego transferu wiedzy.

3.3.1. Przekleństwo wielojęzyczności

Zjawisko to jest związane z pojemnością wielojęzycznych modeli językowych [7]. Każdy model ma określoną pojemność, która może być wyrażona liczbą parametrów modelu, które z przybliżeniem określają, ile wiedzy model może nabyć i zakodować. Wraz ze zwiększeniem liczby przetwarzanych języków dla modelu językowego zmniejsza się pojemność modelu przeznaczona dla zakodowania informacji niezbędnych dla każdego języka. Wykorzystywana pojemność modelu jest zależna m.in. od wielkości zbioru treningowego, jaki został wykorzystany do treningu, a więc języki z małymi zbiorami danych są obecne w parametrach modelu w mniejszym stopniu. Z tego powodu modele z małą liczbą parametrów osiągają gorsze wyniki w stosunku do swoich większych wersji [30].

3.3.2. *Catastrophic forgetting problem*

Ze względu na zakodowanie w parametrach modelu dużej ilości informacji, przykładowo dla wielojęzycznych modeli językowych, może dojść do nadpisania parametrów modelu w czasie treningu [9]. Innymi słowy, w wyniku dotrenowania modelu może dojść do degeneracji reprezentacji uzyskiwanych przez model. Takie zjawisko może występować w przypadku wielojęzycznych modeli językowych,

które podczas procesu uczenia na danych w jednym języku przez aktualizację parametrów nadpisują zakodowaną wiedzę z pozostałych języków. Rozwiązaniem tego problemu może być znalezienie optymalnych parametrów do transferu międzyjęzykowego lub zastosowanie innych technik, takich jak adaptery lub rzadkie dotrenowanie [2].

3.4. Typy transferu

W tym podrozdziale opisane zostały różne typy transferu wiedzy w przetwarzaniu języka naturalnego.

3.4.1. Transfer między zadaniami

Modele językowe ogólnego przeznaczenia mogą zostać w łatwy sposób dostosowane do rozwiązywania konkretnego zadania przez ich dotrenowanie na oznaczonych zbiorach danych. Jeśli jednak nie mamy wystarczająco dużego zbioru danych, możemy posłużyć się transferem wiedzy między zadaniami. Jeśli zadania te są ze sobą związane, jest prawdopodobne, że wykorzystanie pierwszego z nich do procesu uczenia pozwoli osiągnąć dobre wyniki na drugim zadaniu, natomiast w przypadku treningu na połączonych zadaniach wytrenowany model sieci może działać synergicznie.

Efekt poprawienia wyników dla pary zadań został opisany przez autorów artykułu [25], w którym zastosowanie jednocześnie zadania przewidywania wielkich liter w tekście oraz interpunkcji pozwoliło poprawić wyniki na obu zadaniach.

Wpływ modeli oraz zadań na transfer wiedzy między zadaniami został opisany w artykule [1], w którym autorzy zaprezentowali nowy zbiór danych do oceny transferu między zadaniami dla scenariusza *few-shot* dla danych konwersacyjnych. W przypadku wszystkich modeli uczenie jednocześnie na wielu zadaniach pozwoliło poprawić wyniki w stosunku do treningu modelu na jednym zadaniu. Wiedza modelu między zadaniami jest współdzielona i część wiedzy nabytej do rozwiązania jednego zadania pomaga w kolejnych. Autorzy zwracają uwagę, że selekcja zadań do transferu wiedzy między zadaniami jest specyficzna dla konkretnego modelu. Dodatkowo, wybór do treningu wszystkich, niekoniecznie ze sobą związanych, zadań może pogorszyć wyniki, więc konieczna jest uważna selekcja.

3.4.2. Transfer stylu

Innym typem transferu jest transfer stylu polegający na wygenerowaniu tekstu, który zawiera pewne pożądane cechy stylistyczne. W artykule [13] przedstawiono problem wielojęzycznego przekształcania tekstu do stylu formalnego. Podobnym problemem jest wykorzystywanie sieci neuronowych do generowania poezji lub prozy. W tym celu model jest uczony generacji tekstu w sposób autoregresywny na podstawie dużego korpusu książek lub wierszy, dzięki czemu jest

w stanie odtworzyć styl swoich danych treningowych. Autorom artykułu [18] udało się stworzyć system, który generuje limeryki, czyli groteskowe wierszyki o stałej strukturze.

3.4.3. Transfer między językami

Gdy mamy do dyspozycji wielojęzyczny model językowy i zbiór danych dla jednego języka, istnieje pokusa nauczenia modelu rozwiązywania zadania na zbiorze danych dla jednego języka i wykorzystania go dla pozostałych. Standardowe podejście związane z dotrenowaniem modelu nie zagwarantuje odpowiedniego transferu między językami. Model może się nauczyć generować odpowiedzi tylko w języku, na którym był uczony. Może to być spowodowane występowaniem problemów opisanych w 3.3. W przypadku, w którym dysponujemy oznaczonymi zbiorami danych dla wielu języków docelowych, możemy eksperymentalnie wyznaczyć optymalne parametry treningów. Taka sytuacja jednak jest dość rzadka ze względu na dostępność oznaczonych danych dla języków z grupy *low-resource*, czyli takich, dla których dysponujemy ograniczonymi zbiorami danych. Częściej mamy do czynienia z sytuacją, w której dostępny jest oznaczony zbiór danych dla jednego bądź dwóch języków.

Autorzy modelu ZmBART [19] rozwiązali problem związany z transferem między językami dla zadania generowania streszczeń dzięki wykorzystaniu specjalnych tokenów sterujących językiem docelowym. Zaproponowany algorytm składa się z zadania pomocniczego, w którym model językowy uczony jest warunkowania języka generowanego tekstu specjalnym tokenem sterującym. Ten etap wykorzystuje wielojęzyczne nieoznaczone zbiory uczące. Kolejny etap to dotrenowanie modelu na zbiorze oznaczonych danych w jednym języku do rozwiązania zadania docelowego z wykorzystaniem tokenów sterujących. Autorzy wykorzystali zadanie generowania streszczeń, ale nic nie stoi na przeszkodzie, aby wykorzystać tę metodę w innych zadaniach tekstowych, np. parafrazowaniu lub generowaniu słów kluczowych. Dzięki temu model jest w stanie nauczyć się generalizować i przenosić wiedzę z jednego języka na pozostałe.

3.4.4. Transfer między domenami

Poza transferem między językami, zadaniami i stylem możemy mieć również do czynienia z transferem wiedzy między domenami lub inaczej adaptacją domeny. W tej sytuacji model został nauczony na tekstach pochodzących z jednej domeny, a jest wykorzystywany na zupełnie innym typie tekstów.

Dla przykładu, ucząc model generowania streszczeń na artykułach prasowych, możemy użyć go do streszczania dokumentów naukowych w celu automatycznego generowania abstraktów. W przypadku domen, które znacznie różnią się pod kątem stylu oraz występującego słownictwa, do poprawnego działania modelu niezbędne będzie dostosowanie go za pomocą technik adaptacyjnych.

3.5. Metody

Istnieje wiele metod zapewniania transferu wiedzy. W niniejszym podrozdziale przedstawione zostały techniki: *zero-shot*, *few-shot*, dotrenowanie, stopniowe odmrażanie warstw, wielozadaniowość, *prompting* oraz adaptery.

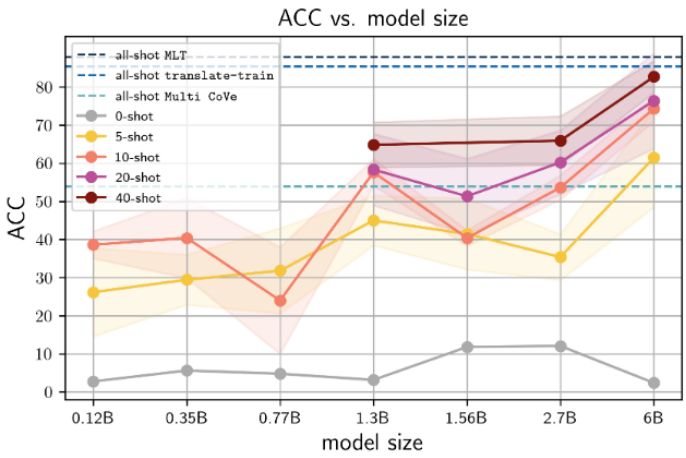
3.5.1. *Zero-shot transfer learning*

Jednym z bardziej oczywistych sposobów transferu wiedzy między zadaniami jest nauczenie modelu i bez żadnych dodatkowych kroków zastosowanie go do rozwiązania innego problemu. Takie podejście jest powszechnie wykorzystywane w przypadkach, w których potrzebujemy uzyskać reprezentacje słowa, zdania lub dokumentu, a następnie wykonać na nich operacje. Takie podejście dobrze sprawdza się w zadaniach służących badaniu podobieństwa między słowami lub zadaniami.

W zadaniach, w których danymi zarówno wejściowymi, jak i wyjściowymi modelu są teksty, stosowane są techniki, które pozwalają na zakodowanie informacji w sposób pośredni. Przykładem takiego rozwiązania jest model do tłumaczenia maszynowego, opisany w [12], w którym autorzy wprowadzili specjalne tokeny określające docelowy język, na który tekst jest tłumaczony. Podczas treningu modelu na parach tekstów w dowolnych językach model nauczył się tłumaczyć teksty z dowolnego obsługiwanego języka na język docelowy przy wykorzystaniu tokenów sterujących. W przypadku, w którym brakowało przykładów dla pary języków, model nauczył się tłumaczenia w sposób pośredni. Taka sytuacja występowała dla pary języków hiszpańskiego i portugalskiego, dla której osiągnięto wyniki porównywalne do modeli uczonych na tekstach zawierających tę parę języków.

3.5.2. *Few-shot transfer learning*

Technika *few-shot transfer learning* jest specjalnym przypadkiem dotrenowania modelu na bardzo małych zbiorach danych, które liczą od kilku do kilkuset próbek uczących. W przypadku modeli językowych ogólnego przeznaczenia, czyli takich, które zostały nauczone głównie tworzenia reprezentacji i transformacji tekstów, technika ta jest skuteczna dla dużych modeli. W artykule [29] autorzy zaprezentowali wyniki klasyfikacji intencji z wykorzystaniem modelu GPT oraz techniki *prompting*. Na rys. 3.1 widać trend, w którym wraz ze wzrostem rozmiaru modelu rośnie skuteczność treningu na małej liczbie przykładów. Wyniki te są związane z ilością informacji zakodowanych w parametrach modelu. Im jest ich więcej, tym mniej potrzeba przykładów uczących, aby rozwiązać określone zadanie. W stosunku do nadzorowanego dotrenowania modeli klasyfikacyjnych na tak małym zbiorze danych wyniki GPT są wielokrotnie lepsze, co pokazuje przydatność stosowania transferu wiedzy szczególnie dla dużych modeli językowych.



Rysunek 3.1: Wyniki skuteczności klasyfikacji intencji w scenariuszu *few-shot* [29]. Oś pozioma prezentuje rozmiar modelu w liczbie parametrów, oś pionowa skuteczność, kolory oznaczają różne rozmiary zbioru uczącego

3.5.3. Dotrenowanie

Z transferem wiedzy podczas dotrenowania mamy do czynienia w sytuacji, w której do rozwiązania zadania wykorzystujemy wiedzę spoza zbioru uczącego. Może to być więc zarówno dotrenowanie modelu językowego ogólnego przeznaczenia na konkretnym zadaniu klasyfikacyjnym, jak i dotrenowanie wielojęzycznego modelu językowego na kilku zbiorach danych, w celu poprawy generalizacji.

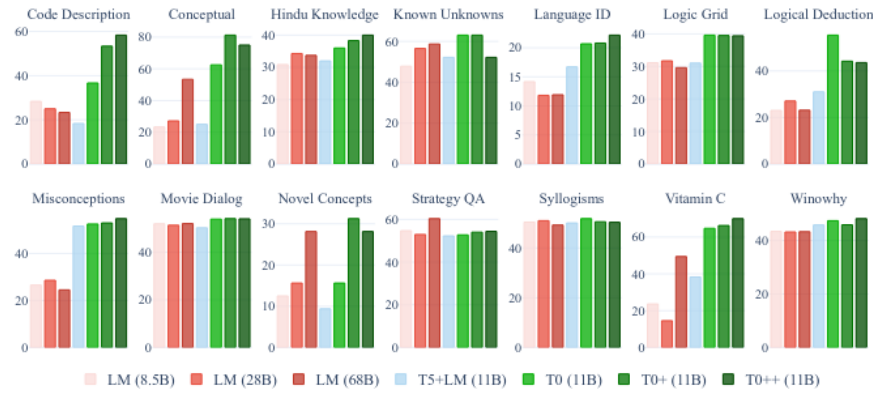
Autorzy w artykule [4] zaprezentowali wyniki badań nad klasyfikacją języka nienawiści w tekstach pisanych w językach: angielskim, hindi i marathi. Z wykorzystaniem modelu wielojęzycznego i treningu na połączonych zbiorach danych dla trzech wymienionych języków uzyskano poprawę wyników w stosunku do trenowania tego samego modelu na każdym języku osobno. Eksperyment ten pokazuje, że transfer wiedzy może być skuteczną metodą poprawy wyników dla potencjalnie dowolnych zadań z wykorzystaniem wielojęzycznych modeli językowych oraz wielojęzycznych zbiorów danych.

3.5.4. Stopniowe odmrażanie warstw

Według autorów pracy [11] techniką przeciwdziałającą *catastrophic forgetting problem* jest stopniowe odmrażanie warstw. Metoda ta polega na stopniowym odmrażaniu jednej warstwy w modelu na każdą epokę treningu. Według autorów w połączeniu z innymi technikami pozwala to poprawić wyniki w zadaniu klasyfikacji tekstu.

3.5.5. Wielozadaniowość

Wielozadaniowość jest kolejnym przykładem techniki, która pozwala na transfer wiedzy między zadaniami, a tym samym uzyskanie efektu synergicznego poprawiającego wyniki modelu na wszystkich zadaniach. Autorzy prezentujący model T0 [27], czyli model oparty na T5, zaprezentowali wpływ uczenia modelu rozwiązywania wielu zadań jednocześnie. Rys. 3.2 pokazuje, że wielozadaniowość umożliwia uzyskanie lepszych wyników *zero-shot*.



Rysunek 3.2: Porównanie wyników uzyskanych dla różnych zadań dla modeli wielozadaniowych (T0) oraz jednozadaniowych (T5) [27]

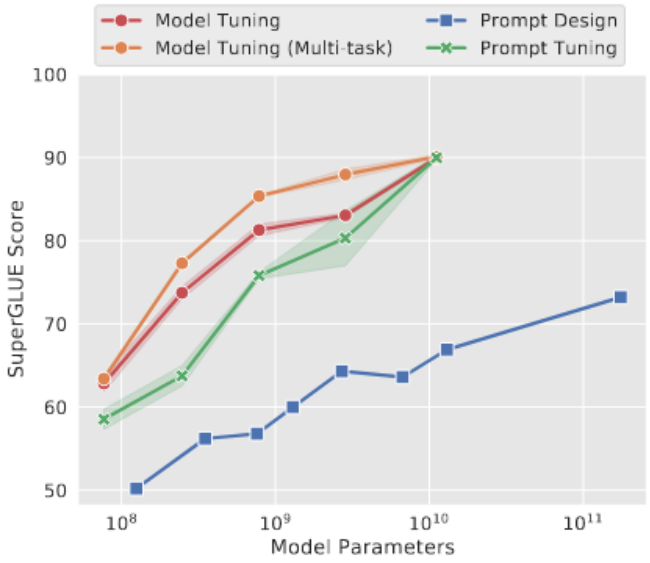
3.5.6. *Prompting*

Prompting jest techniką stosowaną w generatywnych modelach językowych, często wykorzystywaną w scenariuszach *zero-* lub *few-shot*. Tekstowe modele językowe, które jako danych wejściowych i wyjściowych używają tekstów, mogą zostać ukierunkowane w celu poprawnego rozwiązania modelu odpowiednią komendą. Do tego celu stosuje się tekstowe zachęty, które instruują model, jakie zadanie ma wykonać, przykładowo dla zadania tłumaczenia możemy skonstruować następującą komendę: *Translate French to English:*, która jest przedrostkiem tekstu, jaki chcemy przetłumaczyć.

Wybór komendy jest arbitralny, a jego wpływ na działanie modelu nie jest oczywisty, w związku z tym wybór odpowiedniej tekstowej zachęty jest zadaniem trudnym, co sugeruje rys. 3.3 [14]. Problematyka technik związanych z komendami została obszernie opisana w opracowaniu [17].

3.5.7. Adaptery

Ostatnią przedstawioną w tym podrozdziale techniką transferu wiedzy jest architektura adapterów. Generalizując, adaptery są małymi sieciami, dołączanymi do warstw bazowego modelu. W odróżnieniu od zwykłego dotrenowania,

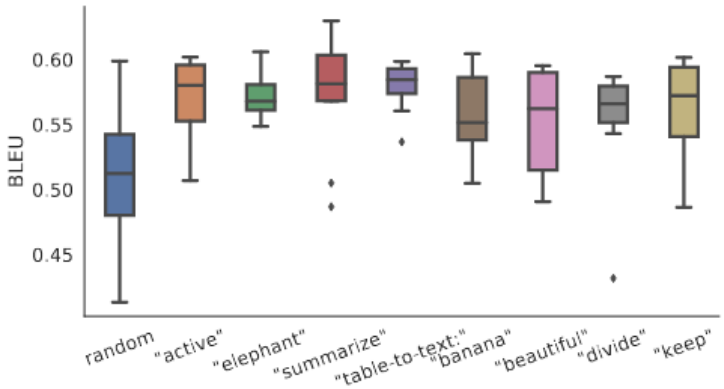


Rysunek 3.3: Porównanie wyników uzyskanych przy zastosowaniu manualnie wybieranych komend oraz techniki *prompt tuning* [14]

w czasie uczenia parametry modelu bazowego są zamrażane i nie są aktualizowane. Optymalizowane są jedynie warstwy adaptera, co pomaga przeciwdziałać degeneracji inicjalnych reprezentacji modelu. Adaptery, ze względu na mały rozmiar, pozwalają ograniczyć rozmiar nauczonego modelu a także poprawić jego skuteczność. Zastosowanie adapterów językowych oraz adapterów zadania umożliwia transfer wiedzy między językami, również w scenariuszu *zero-shot*.

Prefix tuning

Rozszerzeniem metody *prompting* jest *prefix tuning* [15], który zamiast dyskretnej komendy w postaci tekstowej operuje na komendzie ciągłej. Innymi słowy, zamiast przedrostka w postaci tekstowej do sieci dołączana jest macierz o znanych *a priori* długości oraz rozmiarze, która jest optymalizowana w trakcie uczenia. Dzięki temu możliwe jest znalezienie optymalnego przedrostka bez konieczności arbitralnej selekcji komendy tekstowej. Autorzy tej metody pokazują, że nawet w przypadku zastosowania tej techniki inicjalizacja wag macierzy przedrostka reprezentacjami komendy tekstowej adekwatnej do zadania poprawia szanse znalezienia optymalnej reprezentacji oraz wyniki samego modelu w zadaniu docelowym. Jest to szczególnie istotne dla przypadków, w których dysponujemy małym zbiorem uczącym. Przykładowo dla zadania generowania streszczeń inicjalizacja macierzy reprezentacją komendy *summarize:* dała lepsze wyniki niż inicjalizacja losowymi wagami, co pokazano na rys. 3.4.



Rysunek 3.4: Wpływ inicjalizacji macierzy wag przedrostka na wyniki modelu w przypadku, gdy dostępny jest mały zbiór uczący [15]

MAD-X

Adaptery są z powodzeniem wykorzystywane przy zadaniach wymagających dostosowania modelu językowego do rozwiązywania zadań dla wielu języków jednocześnie. Jedną z takich architektur jest MAD-X [22], która została z powodzeniem przetestowana na wielu typologicznie różnych językach do zadania rozpoznawania jednostek nazewniczych, odpowiadania na pytania oraz wnioskowania. Metoda ta wykorzystuje trzy typy adapterów: adaptery językowe, adaptery zadania oraz adaptery odwracalne. Pierwsze dwa zostały oparte na architekturze AdapterDrop, ostatni typ został zaproponowany przez autorów artykułu i stanowi kluczowy element tej architektury.

W pierwszej fazie adaptery językowe oraz odwracalne są uczone na zadaniu MLM (*masked language modelling*) w sposób nienadzorowany oddzielnie dla każdego języka. W trakcie treningu wagi modelu bazowego są zamrażane. Adapter językowy koduje informacje charakterystyczne dla języka, a odwracalny uczy się zależnych od języka transformacji na poziomie poszczególnych tokenów. Druga faza treningu wykorzystuje nauczone w poprzednim kroku adaptery, do których dołącza adapter zadania. W trakcie treningu wagi modelu bazowego oraz adapterów językowych i odwracalnych zostają zamrożone, a model jest uczony w sposób nadzorowany na danych w dostępnych językach. Taki sposób treningu poza zaletami samych adapterów pozwala na transfer wiedzy między językami bez konieczności posiadania nadzorowanych zbiorów uczących w języku docelowym. W czasie inferencji *zero-shot* należy do modelu dołączyć adaptery odwracalne i językowe odpowiedniego języka wraz z adapterem zadania.

MAD-G

Kolejną architekturą adapterów używaną do dostosowywania wielojęzycznych modeli językowych bez konieczności dotrenowania całego modelu jest MAD-G

[3], która wykorzystuje generator adapterów językowych w odniesieniu do cech lingwistycznych języków na podstawie bazy URIEL [16]. Rozwiązanie to umożliwia współdzielenie wiedzy między adapterami językowymi oraz generowanie adapterów dla nieznanych w czasie uczenia języków. W odróżnieniu od innych adapterów, które dotrenowują dołączane warstwy sieci na zadaniu docelowym, autorzy tego rozwiązania zaproponowali wykorzystanie generatora, który na podstawie cech URIEL każdego języka generuje dla niego zanurzenia. Generator jest inicjalizowany cechami lingwistycznymi, a następnie dotrenowywany na zadaniu MLM w sposób nienadzorowany. Po nauczeniu generatora do sieci dołączane są odpowiednie adaptery językowe oraz adaptery zadania. Adaptery językowe oraz sieć bazowa są zamrażane, a w procesie uczenia na zadaniu docelowym aktualizowane są wagi jedynie adaptera zadania.

Dzięki zastosowaniu generatora zanurzeń na podstawie cech URIEL możliwe jest rozszerzenie adapterów o kolejne języki bez konieczności dotrenowania adapterów. Wykorzystanie adapterów pozwala zachować oryginalne wagi modelu bazowego oraz zapobiega występowaniu problemów związanych z degeneracją reprezentacji. Ponadto MAD-G osiąga porównywalne wyniki do architektury MAD-X dla zadań oznaczania części mowy oraz parsowania zależnościowego.

AdapterFusion

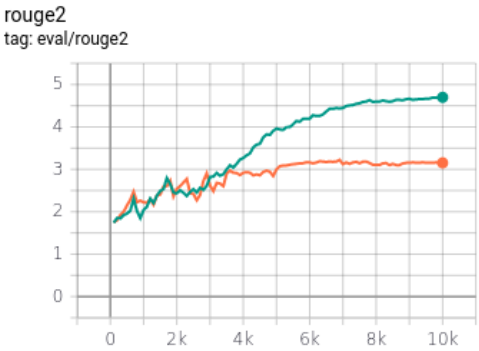
Ostatnią opisaną w tym podrozdziale architekturą adapterów jest AdapterFusion [21], która rozwiązuje problem wielozadaniowości sieci neuronowych. Dzięki zastosowaniu modelu fuzji adapterów możliwe jest wykorzystanie wiedzy zakodowanej w wielu adapterach, dzięki czemu możliwe jest osiągnięcie efektu synergii między zadaniami. Zastosowanie fuzji adapterów chroni przed interferencją w trakcie procesu uczenia na różnych zadaniach. W tym przypadku, tak jak poprzednio, w pierwszym kroku sekwencyjnie uczone są adaptery dla różnych zadań. W czasie treningu wagi modelu bazowego są zamrożone. W drugim etapie do sieci dołączane są wszystkie adaptery zadań nauczone w pierwszym kroku, zamrażane są wagi modelu bazowego oraz adapterów zadań, a następnie dołączana jest warstwa fuzji adapterów, która trenowana jest na wszystkich poprzednio wykorzystanych zbiorach danych. Takie podejście pozwala poprawić wyniki wielozadaniowych modeli w stosunku do dotrenowania modelu bazowego na wszystkich zadaniach.

Case study: generowanie streszczeń

Przeprowadzono eksperyment dla zadania generowania streszczeń, w celu porównania osiąganej skuteczności dla zastosowanych adapterów oraz modelu bazowego. Do eksperymentu wykorzystano model mt5-base[1] oraz zbiór danych XL-Sum [10]. W celu sprawdzenia transferu między językami trening wykonano na zbiorze treningowym języka kirgiskiego, ewaluację natomiast przeprowadzono na zagregowanym zbiorze testowym dla wszystkich języków w zbiorze. W celu ograniczenia złożoności obliczeniowej wybrano po 100 próbek z każdego języka.

[1] https://huggingface.co/google/mt5-base

Na rys. 3.5 widzimy, że zastosowanie adapterów MAD-X [23] znacząco poprawia wyniki w generowaniu streszczeń, o około 50%.



Rysunek 3.5: Porównanie wyników procesu uczenia dla modelu mT5 oraz wersji z dołączonymi adapterami MAD-X. Zielonym kolorem na wykresie oznaczono wyniki modelu z adapterami, kolorem pomarańczowym model mT5.
Oś X przedstawia osiągnięty wynik ROUGE-2 w procesie uczenia, oś Y przedstawia numer iteracji procesu uczenia. Większa wartość ROUGE-2 oznacza lepszy model

3.6. Podsumowanie

W rozdziale omówiono, czym jest transfer wiedzy w przypadku wielojęzycznego przetwarzania języka naturalnego, problemy, motywacje oraz metody, które mogą zostać wykorzystane. Zaprezentowane w tym rozdziale techniki mogą służyć jako punkt odniesienia dla kolejnych eksperymentów w tym obszarze. Metody oparte na dotrenowaniu całej sieci (3.5.2, 3.5.3, 3.5.4 oraz 3.5.5) narażone są na *catastrophic forgetting problem* i wymagają dużej uwagi przy tworzeniu zbioru treningowego. Ze względu na modyfikację nauczonego modelu podczas treningu mogą wystąpić problemy z degradacją zakodowanych w nim informacji. W przypadku metod opartych na przedrostkach (3.5.6) lub adapterach (3.5.7) problem ten jest zniwelowany, ponieważ modyfikowane są jedynie warstwy adaptera, a model bazowy pozostaje niezmieniony. Ze względu na stosunkowo niewielką pojemność informacyjną warstw adaptera ma ograniczone możliwości w stosunku do pełnego dotrenowania. Wybór odpowiedniej techniki transferu wiedzy zależy w dużym stopniu od typu i skomplikowania zadania, zastosowanej sieci oraz dostępnych zbiorów danych.

Bibliografia

[1] Albalak A., et al. *FETA: A Benchmark for Few-Sample Task Transfer in Open-Domain Dialogue*, 2022. URL `https://arxiv.org/abs/2205.`

06262.

[2] Ansell A., et al. *Composable Sparse Fine-Tuning for Cross-Lingual Transfer*, 2021. URL https://arxiv.org/abs/2110.07560.

[3] Ansell A., et al. *MAD-G: Multilingual Adapter Generation for Efficient Cross-Lingual Transfer*. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 4762–4781. Association for Computational Linguistics, Punta Cana, Dominican Republic, 2021. URL https://aclanthology.org/2021.findings-emnlp.410.

[4] Bhatia M., et al. *One to rule them all: Towards Joint Indic Language Hate Speech Detection*, 2021. URL https://arxiv.org/abs/2109.13711.

[5] Bojanowski P., et al. *Enriching Word Vectors with Subword Information*, 2016. URL https://arxiv.org/abs/1607.04606.

[6] Cho K., et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014. URL https://arxiv.org/abs/1406.1078.

[7] Conneau A., et al. *Unsupervised Cross-lingual Representation Learning at Scale*, 2019. URL https://arxiv.org/abs/1911.02116.

[8] Devlin J., et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2018. URL https://arxiv.org/abs/1810.04805.

[9] Goodfellow I.J., et al. *An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks*, 2013. URL https://arxiv.org/abs/1312.6211.

[10] Hasan T., et al. *XL-Sum: Large-Scale Multilingual Abstractive Summarization for 44 Languages*. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 4693–4703. Association for Computational Linguistics, Online, 2021. URL https://aclanthology.org/2021.findings-acl.413.

[11] Howard J., Ruder S. *Universal Language Model Fine-tuning for Text Classification*. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 328–339. Association for Computational Linguistics, Melbourne, Australia, 2018. URL https://aclanthology.org/P18-1031.

[12] Johnson M., et al. *Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation*, 2016. URL https://arxiv.org/abs/1611.04558.

[13] Lai H., Toral A., Nissim M. *Multilingual Pre-training with Language and Task Adaptation for Multilingual Text Style Transfer*, 2022. URL https://arxiv.org/abs/2203.08552.

[14] Lester B., Al-Rfou R., Constant N. *The Power of Scale for Parameter-Efficient Prompt Tuning*, 2021. URL https://arxiv.org/abs/2104.08691.

[15] Li X.L., Liang P. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*, 2021. URL https://arxiv.org/abs/2101.00190.

[16] Littell P., et al. *URIEL and lang2vec: Representing languages as typological, geographical, and phylogenetic vectors*. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 8–14. Association for Computational Linguistics, 2017. URL http://aclweb.org/anthology/E17-2002.

[17] Liu P., et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*, 2021. URL https://arxiv.org/abs/2107.13586.

[18] Lo K.L., Ariss R., Kurz P. *GPoeT-2: A GPT-2 Based Poem Generator*, 2022. URL https://arxiv.org/abs/2205.08847.

[19] Maurya K.K., et al. *ZmBART: An Unsupervised Cross-lingual Transfer Framework for Language Generation*, 2021. URL https://arxiv.org/abs/2106.01597.

[20] Mikolov T., et al. *Distributed Representations of Words and Phrases and their Compositionality*, 2013. URL https://arxiv.org/abs/1310.4546.

[21] Pfeiffer J., et al. *AdapterFusion: Non-Destructive Task Composition for Transfer Learning*. 2020. URL https://arxiv.org/abs/2005.00247.

[22] Pfeiffer J., et al. *MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer*. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7654–7673. Association for Computational Linguistics, Online, 2020. URL https://aclanthology.org/2020.emnlp-main.617.

[23] Pfeiffer J., et al. *MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer*, 2020. URL https://arxiv.org/abs/2005.00052.

[24] Radford A., et al. *Language Models are Unsupervised Multitask Learners*. 2019.

[25] Rücklé A., et al. *AdapterDrop: On the Efficiency of Adapters in Transformers*. pp. 7930–7946. 2021.

[26] Sak H., Senior A., Beaufays F. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*, 2014. URL https://arxiv.org/abs/1402.1128.

[27] Sanh V., et al. *Multitask Prompted Training Enables Zero-Shot Task Generalization*, 2021. URL https://arxiv.org/abs/2110.08207.

[28] Vaswani A., et al. *Attention Is All You Need*, 2017. URL `https://arxiv.org/abs/1706.03762`.

[29] Winata G.I., et al. *Language Models are Few-shot Multilingual Learners*, 2021. URL `https://arxiv.org/abs/2109.07684`.

[30] Xue L., et al. *mT5: A massively multilingual pre-trained text-to-text transformer*, 2020. URL `https://arxiv.org/abs/2010.11934`.

Rozdział 4

Architektury klasyfikatorów obrazów

Konrad Zawora[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
konrad.zawora@pg.edu.pl

Abstrakt

Klasyfikacja obrazów jest zagadnieniem z dziedziny widzenia komputerowego. Polega na całościowej analizie obrazu i przypisaniu go do jednej lub wielu kategorii (klas). Współczesne rozwiązania tego problemu są w znacznej części realizowane z wykorzystaniem konwolucyjnych głębokich sieci neuronowych (*convolutional neural network*, CNN). W tym rozdziale opisano przełomowe architektury CNN oraz ewolucję *state-of-the-art* w klasyfikacji obrazów na przestrzeni lat 2014–2021. Łącznie opisano 28 topologii głębokich konwolucyjnych sieci neuronowych, należących do 7 rodzin: EfficientNet, ResNet, DenseNet, Inception, NasNet, MobileNet oraz VGG.

Słowa kluczowe: sztuczna inteligencja, uczenie maszynowe, uczenie głębokie, widzenie komputerowe, rozpoznawanie obrazów

4.1. Wprowadzenie do klasyfikacji obrazów

Widzenie komputerowe jest rozwijającą się poddziedziną uczenia maszynowego. Polega na rozwiązywaniu problemów i automatyzacji zadań, za które typowo

odpowiedzialny jest wzrok człowieka. Początki dziedziny sięgają lat 60. XX wieku [9]. Pierwotnie traktowana była jako zadanie trywialne, możliwe do wykonania przez studentów w ramach projektu letniego, jednak po 60 latach i wielu innowacjach problem nadal nie został w pełni rozwiązany.

Znaczącymi czynnikami utrudniającymi widzenie komputerowe są jego otwartość, złożoność oraz brak pełnego zrozumienia procesów poznawczych związanych z widzeniem u ludzi. Zadania proste oraz intuicyjne dla człowieka potrafią okazać się bardzo złożonymi problemami dla komputera, np. rozróżnienie krawędzi obiektu w przestrzeni, umiejętność nazwania i opisania go razem z jego otoczeniem.

Zadania realizowane w widzeniu komputerowym są bardzo różnorodne. Do przykładowych problemów należą:

- klasyfikacja obiektów;
- detekcja i lokalizacja obiektów;
- segmentacja semantyczna i panoptyczna;
- opisywanie obrazów;
- *content-based image retrieval* (CBVIR) – wyszukiwanie na podstawie obrazków;
- optyczne rozpoznawanie znaków (*optical character recognition*, OCR);
- rozpoznawanie i weryfikacja twarzy;
- wizyjna odometria;
- superskalowanie obrazów;
- interpolacja klatek w filmach;
- przenoszenie stylu (*neural style transfer*);
- generowanie obrazów.

W tym rozdziale skupiono się wyłącznie na klasyfikacji obiektów. Polega ona na przypisaniu jednej lub wielu określonych etykiet do obrazu, na podstawie wykrytych cech. Problem przypisania pojedynczej etykiety określa się jako klasyfikację wieloklasową (*multiclass classification*), a przypisania wielu etykiet jako klasyfikację wieloetykietową (*multi-label classification*).

Obecnie najdokładniejszymi metodami klasyfikowania obiektów są algorytmy oparte na głębokich konwolucyjnych sieciach neuronowych. Bazują one na operacji dwuwymiarowego splotu (konwolucji), który umożliwia wykrywanie szablonów w obrazie źródłowym. W konwolucyjnych sieciach neuronowych rodzaj wykrywanego szablonu nie jest ustalony, lecz wyznaczany w ramach procesu treningu. Parametry charakteryzujące splot muszą być jednak ustalone – np. rozmiar filtra lub liczba kanałów wyjściowych. Istnieje nieskończenie wiele sposobów organizacji splotów wewnątrz głębokiej sieci neuronowej, a poszukiwanie

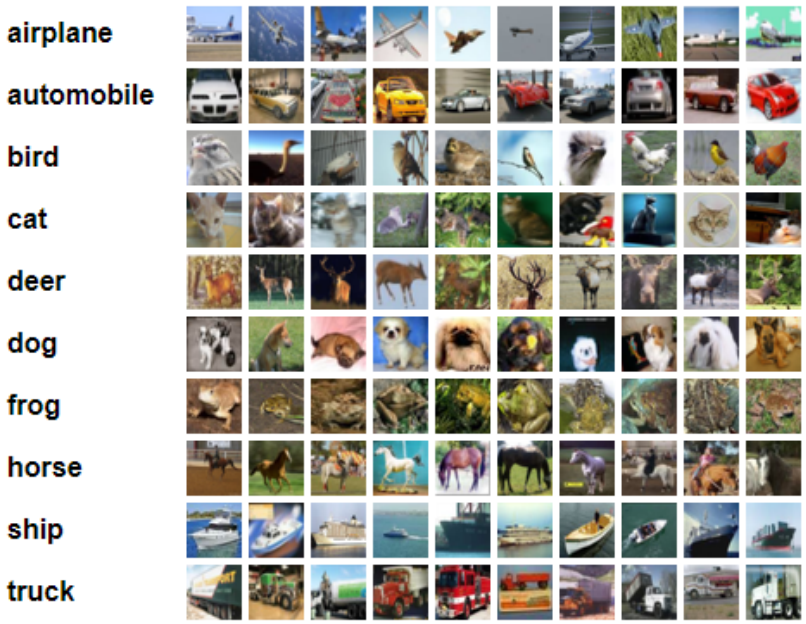
optymalnych topologii (architektur) jest obiektem badań wielu naukowców. Typową głęboką sieć konwolucyjną można podzielić na dwa logiczne komponenty: ekstraktory cech oraz klasyfikator. Rolą ekstraktorów cech jest przekształcenie obrazu wejściowego do pewnej mapy cech, która określi obecność lub brak pewnych atrybutów w obrazie. Rolą klasyfikatora jest przekształcenie wynikowej mapy cech na rozkład prawdopodobieństwa przynależności do klas. Jako bardzo uproszczony przykład można rozważyć sieć klasyfikującą zdjęcie jako obraz samochodu, kota lub człowieka. Model otrzymuje na wejściu zdjęcie człowieka, które przekształca do mapy cech. Otrzymana mapa może wskazywać na obecność np. ludzkich rąk, nóg, głowy i brak obecności kocich uszu oraz kół. Te dane są przekazywane do klasyfikatora, który na podstawie tych informacji powinien zwrócić wektor, który wskaże na wysokie prawdopodobieństwo obecności człowieka na zdjęciu. W praktyce mapa cech zazwyczaj nie reprezentuje typowych atrybutów rozróżnialnych przez ludzi, lecz szablony wynikające z procesu treningu. Trening typowej sieci konwolucyjnej odbywa się poprzez propagowanie obrazów wejściowych przez sieć, w sposób analogiczny do wcześniej wspomnianego. Następnie na podstawie otrzymanych wyników, oblicza się wartość funkcji kosztu, która będzie określać różnicę między dokonanymi predykcjami a prawidłową etykietą (*ground truth*). Trening odbywa się przez minimalizację funkcji kosztu. Osiągane to jest za pomocą obliczenia gradientów trenowalnych parametrów. Następnie parametry są aktualizowane na podstawie wyznaczonych gradientów. Proces ten wykonywany jest wielokrotnie, z wykorzystaniem różnych danych, co jest odzwierciedlone przeważnie w znaczącym koszcie obliczeniowym.

Aby móc ocenić i porównać jakość architektur, potrzebna jest jednolita metryka porównawcza. W przypadku klasyfikacji obiektów jest to konkurs *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC). Polega on na klasyfikacji różnorodnych obiektów należących do 1000 klas na podstawie 1,2 miliona zdjęć treningowych. Dokładność jest testowana na 150 000 zdjęć i obliczane są metryki dokładności Top1 oraz Top5. Dokładność Top1 oznacza odsetek zdjęć sklasyfikowanych poprawnie bezpośrednio, a Top5 bierze pod uwagę 5 predykcji z największym prawdopodobieństwem. Celem jest maksymalizacja obu wyników – wraz z postępującymi innowacjami w dziedzinie proponowane są nowe architektury, osiągające coraz wyższe wyniki. Do badań w pracy wybrano przełomowe topologie, które w większości ustanawiały nowe rekordy dokładności w momencie zdefiniowania.

ImageNet nie jest jedynym znaczącym zbiorem danych w klasyfikacji obiektów. Do popularnych wykorzystywanych zbiorów należą m.in.: CIFAR-10, CIFAR-100, SIFT10M, Open Images Dataset, Microsoft Common Objects in Context (COCO). Przykładowe zdjęcia zawarte w zbiorze CIFAR-10 przedstawiono na rys. 4.1. Pobicie rekordu w konkursie ILSVRC nie zawsze przenosi się bezpośrednio na rekordowe wyniki na innych zbiorach. Zbiory mogą się różnić podstawowymi właściwościami danych – np. rozdzielczością pojedynczego obrazu, co może rzutować bezpośrednio na architekturę. Architektury ekstraktorów cech często zawierają komponenty redukujące wymiary mapy cech. Operowanie na obrazach większej rozdzielczości typowo wymaga głębszej sieci oraz

większej liczby redukcji. Z tego powodu architektury uzyskujące wysoką dokładność na zbiorze CIFAR-10, zawierającym zdjęcia o rozdzielczości 32×32, są zazwyczaj płytsze i zawierają mniej parametrów niż architektury wygrywające ILSVRC, operujące na rozdzielczości większej lub równej 224×224.

Mając na uwadze te różnice, wiele klasyfikatorów używa zbiorów danych, wzorując się na charakterystyce zdjęć ze zbioru ImageNet. Umożliwia to bezpośrednie wykorzystanie wiodących architektur, bez dodatkowych modyfikacji. Dodatkowo wiąże się to z dostępem do wytrenowanych parametrów na potrzeby konkursu ILSVRC, co otwiera możliwość wykorzystania techniki *transfer learning*. *Transfer learning* polega na wykorzystaniu wytrenowanego modelu osiągającego dobrą dokładność w jednym problemie i przetrenowaniu go do rozwiązywania innego, zbliżonego problemu. Jest to dobry punkt startowy – sieć na samym początku będzie używać gotowych parametrów ekstraktorów cech (głównie filtrów warstw konwolucyjnych) odpowiednich do rozwiązywania bardzo ogólnego problemu. Istnieje duża szansa, że wiele wykrywanych cech będzie miało odzwierciedlenie w nowym problemie – bezpośrednio lub po przetrenowaniu. Proces przetrenowywania wymaga znacznie mniej zasobów obliczeniowych niż trening od zera oraz umożliwia uzyskiwanie wysokiej dokładności nawet na niewielkich zbiorach danych, co sprawia, że ta technika cieszy się dużą popularnością w rozwiązywaniu nowych problemów klasyfikacji obiektów.



Rysunek 4.1: Przykładowe zdjęcia i etykiety ze zbioru CIFAR-10 [1]

W dalszych sekcjach rozdziału opisano architektury oraz osiągnięcia 28 topologii głębokich konwolucyjnych sieci neuronowych, należących do 7 rodzin, wraz

z wyszczególnieniem różnic pomiędzy pojedynczymi modelami. Wykorzystane sieci to:

- EfficientNet
  - EfficientNetB0
  - EfficientNetB1
  - EfficientNetB2
  - EfficientNetB3
  - EfficientNetB4
  - EfficientNetB5
  - EfficientNetB6
  - EfficientNetB7
- NASNet
  - NASNetMobile
  - NASNetLarge
- ResNet
  - ResNet50
  - ResNet50V2
  - ResNet101
  - ResNet101V2
  - ResNet152
  - ResNet152V2
- MobileNet
  - MobileNet
  - MobileNetV2
  - MobileNetV3Large
  - MobileNetV3Small
- Inception
  - InceptionV3
  - InceptionResNetV2
  - Xception
- DenseNet
  - DenseNet121
  - DenseNet169
  - DenseNet201
- VGG
  - VGG16
  - VGG19

4.2. EfficientNet

Architekturę EfficientNet zdefiniowano w 2019 roku w publikacji [16], z aktualizacją w roku 2020. Osiągają one w konkursie ILSVRC dokładności Top1 w granicach 77,1–84,3% oraz Top5 93,3–97%. Głównym obiektem badań, na podstawie których powstały modele z tej rodziny, było wydajne skalowanie głębokich konwolucyjnych sieci neuronowych.

4.2.1. Konwencjonalne metody skalowania

Autorzy wyróżniają trzy metody (wymiary) skalowania:

- skalowanie w głąb (*depth scaling*);
- skalowanie wszerz (*width scaling*);
- skalowanie rozdzielczości (*resolution scaling*).

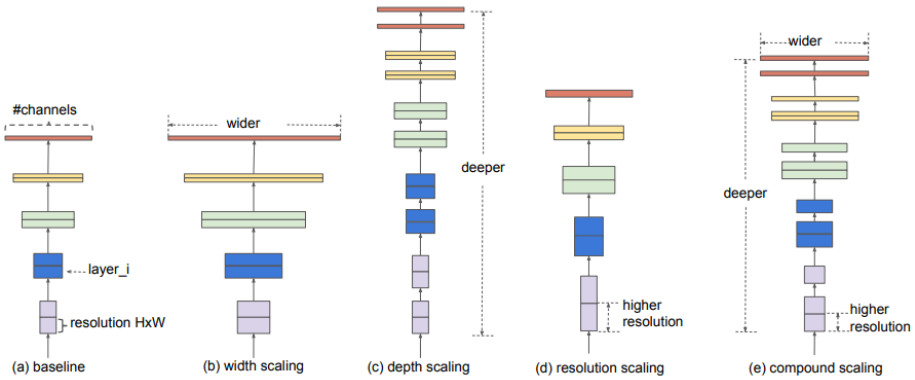
Skalowanie w głąb jest najczęściej stosowaną techniką i polega na zwiększeniu liczby warstw konwolucyjnych. Wiąże się to ze wzrostem liczby i złożoności wykrywanych cech, jednak utrudnia i wydłuża proces uczenia. Ta metoda znacznie poprawia jakość klasyfikacji do pewnej granicy, po przekroczeniu której dalsze skalowanie skutkuje zmniejszonymi zyskami. Dodatkowo w bardzo głębokich sieciach konwolucyjnych można zaobserwować problem zanikającego gradientu, lecz nawet w przypadku architektur, w których ten problem został zminimalizowany (np. ResNet), występuje punkt saturacji skalowania w głąb, powyżej którego wyniki przestają rosnąć.

Skalowanie wszerz polega na zwiększeniu liczby kanałów warstw konwolucyjnych. Jest przeważnie używane w płytkich sieciach neuronowych, w których zwiększając szerokość, odnotowano poprawę jakości klasyfikacji oraz prędkości treningu [21]. Płytsze sieci konwolucyjne przeważnie dobrze rozpoznają szczegółowe cechy, lecz słabo generalizują. Ponadto, podobnie jak w skalowaniu w głąb, występuje punkt saturacji, powyżej którego skalowanie przynosi coraz mniejsze korzyści.

Skalowanie rozdzielczości umożliwia detekcję bardziej szczegółowych cech, które mogłyby zostać utracone przy użyciu obrazów niższej rozdzielczości. Tak jak przy użyciu ww. metod, istnieje punkt, powyżej którego zwiększanie rozdzielczości nie skutkuje lepszą jakością klasyfikacji.

4.2.2. Skalowanie złożone

Autorzy architektury modelu zauważyli, że skalowanie któregokolwiek z trzech wymienionych wymiarów spowoduje wzrost jakości klasyfikacji, jednak większe modele będą charakteryzować się coraz mniejszymi zyskami. Ponadto, badania empiryczne wskazują na wzajemną zależność tych wymiarów. Twórcy odnotowują konieczność zwiększenia głębokości oraz szerokości wraz ze wzrostem rozdzielczości. Potrzeba przeskalowania głębokości wynika z faktu, że cechy znajdują się na większym polu receptywnym, przez co większa liczba konwolucji jest wymagana do obsługi cech równoważnych obrazowi niższej rozdzielczości. Analogicznie, konieczność wykrywania bardziej szczegółowych cech powoduje potrzebę przeskalowania szerokości. Bazując na tych obserwacjach, autorzy wskazują na problem wyważenia wszystkich trzech wymiarów i jako rozwiązanie proponują **skalowanie złożone** (*compound scaling*), które zostało przedstawione na rys. 4.2.



Rysunek 4.2: Wizualizacja metod skalowania przedstawionych w publikacji definiującej architekturę EfficientNet
Od lewej: a) sieć bazowa, b) skalowanie wszerz, c) skalowanie w głąb, d) skalowanie rozdzielczości, e) skalowanie złożone [16]

Skalowanie złożone zakłada istnienie współczynnika (ϕ), który służy do równomiernego skalowania głębokości, szerokości i rozdzielczości według wzoru (4.1):

$$\begin{aligned} \text{głębokość: } & d = \alpha^\phi \\ \text{szerokość: } & w = \beta^\phi \\ \text{rozdzielczość: } & r = \gamma^\phi \\ \text{takie, że: } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \quad \beta \geq 1, \quad \gamma \geq 1 \end{aligned} \tag{4.1}$$

Wzór $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ powstał z przybliżenia złożoności obliczeniowej konwolucji. Autorzy wskazują, że podwojenie głębokości poskutkuje podwojeniem liczby operacji zmiennoprzecinkowych (FLOPs[1]), natomiast podwojenie szerokości lub rozdzielczości spowoduje czterokrotny wzrost FLOPs. Zakłada się zatem, że w sieciach konwolucyjnych, gdzie przeważającą część obliczeń zajmują warstwy konwolucyjne, w sytuacji użycia skalowania złożonego o współczynniku ϕ liczba wymaganych FLOPs na całą sieć wzrośnie w przybliżeniu o $(\alpha \cdot \beta^2 \cdot \gamma^2)^\phi$. Przybliżenie do dwóch sprawia, że wiadomo *a priori*, że koszt obliczeniowy wzrośnie o około 2^ϕ.

Aby zastosować skalowanie złożone, autorzy proponują ustalić $\phi = 0$, po czym wykonać optymalizację parametrów α, β i γ metodą *grid search*, według wzoru (4.1). Następnie podstawić znalezione wartości α, β i γ jako stałe do wzoru (4.1) i utworzyć przeskalowaną sieć według dobranych wartości ϕ. Przykładowy wynik *grid search* dla EfficientNetB0 wynosi:

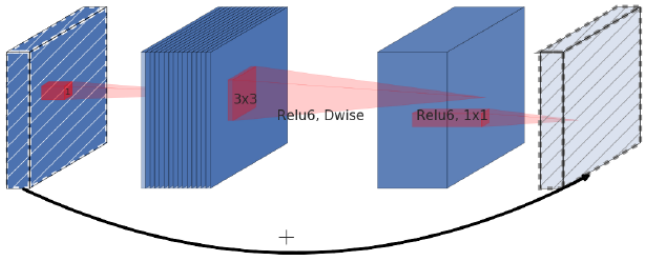
$$\alpha = 1,2, \quad \beta = 1,1, \quad \gamma = 1,15 \tag{4.2}$$

4.2.3. Model bazowy

Model bazowy jest krytyczną częścią architektury EfficientNet. Samo skalowanie złożone nie zmienia rodzajów operacji wykonywanych w modelu, więc ważne jest dobranie architektury, która zapewni dobry balans pomiędzy dokładnością a liczbą obliczeń na każdym etapie skalowania.

Autorzy prezentują sieć zbliżoną strukturą do MnasNet oraz MobileNetV2, która ma zapewnić wysoką jakość klasyfikacji przy liczbie parametrów umożliwiającej wykonanie na urządzeniach mobilnych. Głównym budulcem jest warstwa MBConv (rys. 4.3), będąca odwróconym blokiem rezydualnym (*inverted residual block*). Zawiera on trzy warstwy konwolucyjne: dwie z filtrami 1×1 na początku oraz końcu bloku oraz jedną konwolucję „depthwise" z filtrem 3×3 pomiędzy dwiema wcześniej wspomnianymi. Dodatkowo blok zawiera optymalizację *Squeeze-and-Excitation*, pozwalającą na dynamiczną rekalibrację kanałów.

[1]FLOPs – *floating point operations*, nie mylić z *floating point operations per second*



Rysunek 4.3: Odwrócony blok rezydualny [11]

4.2.4. Implementacja

Zdefiniowano aż 8 modeli EfficientNet: EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4, EfficientNetB5, EfficientNetB6, EfficientNetB7. Powstały one na bazie EfficientNetB0 z wykorzystaniem skalowania złożonego, z rosnącym współczynnikiem ϕ. Wykorzystując wzór (4.1), wartości wyznaczone w (4.2) oraz dane z modelu bazowego (rys. 4.4), można z łatwością wyznaczyć właściwości modeli od EfficientNetB0 do EfficientNetB7 (tabl. 4.1):

Model	ϕ	d (α^ϕ)	w (β^ϕ)	r (γ^ϕ)	**Rozdzielczość** $224 \cdot r$ $(224 \cdot \gamma^\phi)$
EfficientNetB0	0	1,00	1,00	1,00	224,00
EfficientNetB1	1	1,20	1,10	1,15	257,60
EfficientNetB2	2	1,44	1,21	1,32	296,24
EfficientNetB3	3	1,73	1,33	1,52	340,68
EfficientNetB4	4	2,07	1,46	1,75	391,78
EfficientNetB5	5	2,49	1,61	2,01	450,54
EfficientNetB6	6	2,99	1,77	2,31	518,13
EfficientNetB7	7	3,58	1,95	2,66	595,84

Tablica 4.1: Obliczone wartości skalowania złożonego dla EfficientNet

Paradoksalnie, mimo dużego nacisku architektury na wydajność obliczeniową, powyższe wartości mogą okazać się bardzo niewydajne. Jeden z najpopularniejszych frameworków uczenia głębokiego, TensorFlow, stosuje *zero-padding* w konwolucjach dla rozdzielczości niepodzielnych przez 8, co spowoduje marnotrawstwo zasobów obliczeniowych. Dodatkowo występuje wymóg podzielności przez 8 kanałów warstw konwolucyjnych. Z perspektywy wydajności na niskim poziomie producenci akceleratorów często tworzą wysoce zoptymalizowane implementacje określonych warstw dla konkretnych rozmiarów, co może poskutkować dużo lepszą wydajnością w konwolucjach z np. 512 kanałami niż 513. Z tego powodu implementacje w wiodących frameworkach (TensorFlow, PyTorch) używają manualnie dobranych właściwości zapewniających wysoką wydajność i do-

brą, przetestowaną jakość wyników, lecz same wartości mogą znacznie odbiegać od wyznaczonych przez wzór (4.1). Tablica 4.2 przedstawia faktyczne wartości wykorzystywane we frameworku TensorFlow.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Rysunek 4.4: Bazowy model EfficientNet [16]

Model	d (α^ϕ)	w (β^ϕ)	**Rozdzielczość** $224 \cdot r$ $(224 \cdot \gamma^\phi)$
EfficientNetB0	1,0	1,0	224
EfficientNetB1	1,1	1,0	240
EfficientNetB2	1,2	1,1	260
EfficientNetB3	1,4	1,2	300
EfficientNetB4	1,8	1,4	380
EfficientNetB5	2,2	1,6	456
EfficientNetB6	2,6	1,8	528
EfficientNetB7	3,1	2,0	600

Tablica 4.2: Zaimplementowane wartości skalowania złożonego dla EfficientNet we frameworku TensorFlow

4.3. ResNet

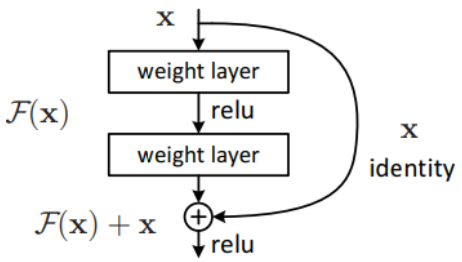
ResNet50 został zdefiniowany w 2015 roku w pracy [3] jako propozycja rozwiązania problemu zanikających i eksplodujących gradientów. Dzięki połączeniom rezydualnym topologie tej architektury są w stanie osiągnąć bezprecedensową głębokość (nawet 1000 warstw konwolucyjnych) bez napotkania na wcześniej wspomniany problem. W badaniach wykorzystano sześć modeli z tej architektury: ResNet50, ResNet101 oraz ResNet152, w wersji podstawowej oraz V2. Osiągają one w konkursie ImageNet 74,9–78% dokładności Top1 oraz 92,1–94,2% Top5, zależnie od głębokości modelu.

4.3.1. Problem zanikających i eksplodujących gradientów

W czasach powstawania architektury ResNet dominującym podejściem podnoszenia jakości klasyfikatorów było skalowanie ich głębokości. Stwarza to jednak problem na etapie propagacji wstecznej oraz optymalizacji, gdyż obliczone gradienty są propagowane przez wszystkie zależne warstwy. Celem propagacji wstecznej jest obliczenie dla każdego trenowalnego parametru $\frac{\partial L}{\partial W}$, gdzie L to funkcja kosztu, a W to parametr. Jeżeli jednak W nie poprzedza bezpośrednio funkcji kosztu, wykorzystuje się regułę łańcuchową – dla przykładów warstw sekwencyjnie połączonych z parametrami W_0 i W_1 obliczenie gradientu względem W_0 będzie wyglądać następująco: $\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial W_0} \cdot \frac{\partial W_0}{\partial W_1}$. Wraz ze wzrostem głębokości wzrasta też liczba czynników mnożenia. Jeżeli czynniki będą miały bardzo małe wartości, to gradient będzie dążył do zera, co spowoduje problem zanikającego gradientu, co z kolei sprawi, że warstwa i zależne od niej warstwy przestaną się uczyć, bo gradienty będą bliskie lub równe zeru. Jeżeli czynniki będą miały duże wartości, spowoduje to znaczną niestabilność trenowalnych parametrów w trenowanej warstwie oraz jej zależności z potencjalnymi przepełnieniami i wartościami NaN. Oba podejścia przeważnie wpływają katastrofalnie na wyniki treningu i są znacznym problemem w uczeniu sieci głębokich.

4.3.2. Połączenia rezydualne

Jako rozwiązanie problemu eksplodujących i zanikających gradientów, zaproponowano koncept połączeń rezydualnych, przedstawionych na rys. 4.5. Umożliwiają one łatwiejszy przepływ gradientów przez sieć. Można je intuicyjnie potraktować jako szynę, do której w sposób równoległy podłączono warstwy z trenowalnymi parametrami, a wyniki zsumowano. Sprawia to, że w propagacji wstecznej, w punkcie sumowania, wynik dodawania zostanie rozdzielony na dwie „gałęzie” – jedna ominie cały blok, a druga zostanie użyta do propagacji wstecznej bloku. Następnie gradienty obu „gałęzi” są dodawane. Dzięki temu bloki rezydualne mają wpływ na wynikowy gradient wymagany przez wyższe warstwy, lecz ryzyko wystąpienia zanikającego lub eksplodującego gradientu jest drastycznie zmniejszone przez wydzielenie nieliniowych funkcji aktywacji do osobnej gałęzi.



Rysunek 4.5: Schemat bloku rezydualnego [3]

Warto także zwrócić uwagę, że zastosowanie operacji sumowania staje się problematyczne przy zmianie rozmiaru przetwarzanego tensora. Bezpośrednie dodanie wejścia bloku do wyjścia staje się niemożliwe, jeżeli blok modyfikuje wymiary – np. na przykładzie warstwy *pooling* lub konwolucji z parametrem *stride* większym niż 1. W takim przypadku wymagane jest dodanie operacji pośredniej przy połączeniu rezydualnym, która przekształci tensor wejściowy do wymaganego kształtu. W przypadku architektury ResNet jest to warstwa konwolucyjna z filtrem 1×1 i odpowiednim parametrem *stride*.

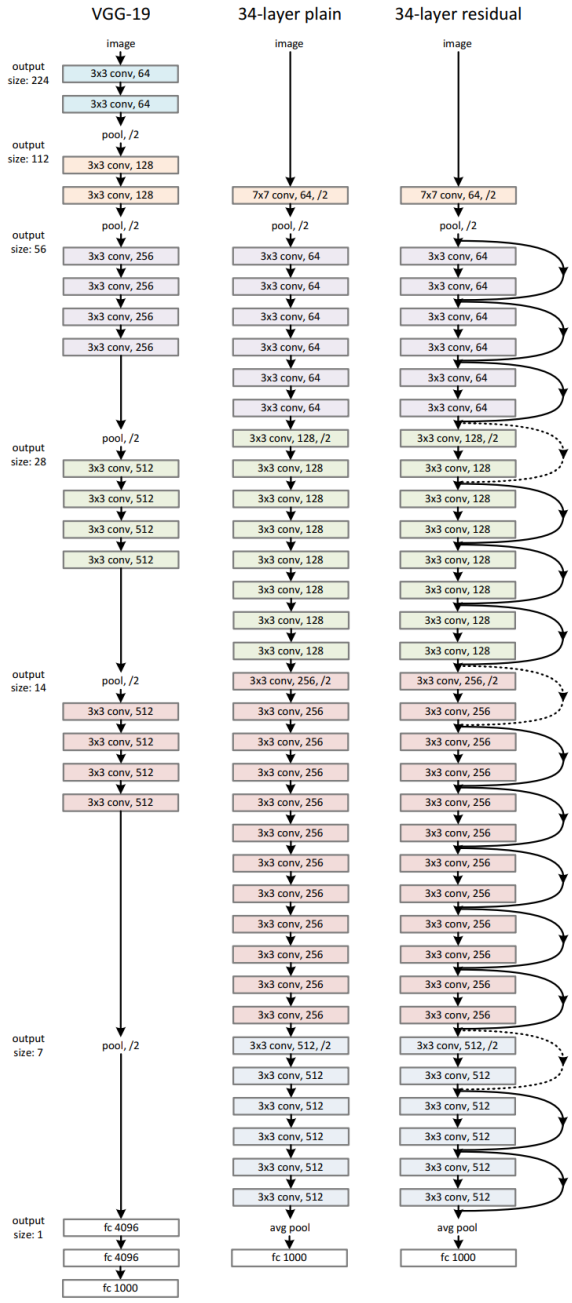
4.3.3. Architektura modelu

Architektura ResNet jest skalowalna – autorzy publikacji przedstawiają warianty ResNet-18, ResNet-34, ResNet-50, ResNet-101 oraz ResNet-152. W rzeczywistości liczba warstw jest znacznie większa, niż wskazuje to nazwa topologii. Autorzy nazywają modele zależnie od liczby warstw konwolucyjnych w blokach rezydualnych oraz ostatniej warstwy gęstej. Nie obejmuje to wielu warstw, np. *pooling*, *batch normalization* lub konwolucji umieszczonych w połączeniach rezydualnych.

We wszystkich zaproponowanych wariantach topologia składa się z pięciu bazowych bloków: *conv1*, *conv2*, *conv3*, *conv4* oraz *conv5*. Oznaczają one zmianę w wysokości i szerokości przetwarzanych danych – każdy blok redukuje je dwukrotnie, podobnie jak w VGG. Warto zauważyć, że redukcja rozmiarów w blokach *conv2*, *conv3* oraz *conv4* odbywa się inaczej niż w VGG – jest ona osiągana z użyciem *stride* 2 w ostatniej warstwie konwolucyjnej. Architekturę ResNet-34 przedstawiono na rys. 4.6, gdzie poszczególne bloki zaznaczone są różnymi kolorami, a połączenia rezydualne wymagające dodatkowej konwolucji 1×1 z parametrem *stride* równym 2. Między każdym połączeniem rezydualnym wykonywana jest także funkcja aktywacji ReLU.

W przypadku ResNet-18 oraz ResNet-34 wykorzystano w pojedynczym bloku rezydualnym dwie konwolucje z filtrem 3×3. W głębszych wariantach zostało to zmienione na trzy konwolucje, z filtrami kolejno 1×1, 3×3 oraz 1×1. Dokładną liczbę filtrów oraz wykorzystywanych warstw przedstawiono na rys. 4.7. W każdym przypadku po warstwach konwolucyjnych występuje *batch normalization* [8]. Dodatkowo po operacjach *batch normalization* niewystępujących bezpośrednio na końcu bloku rezydualnego wykonywana jest funkcja aktywacji ReLU.

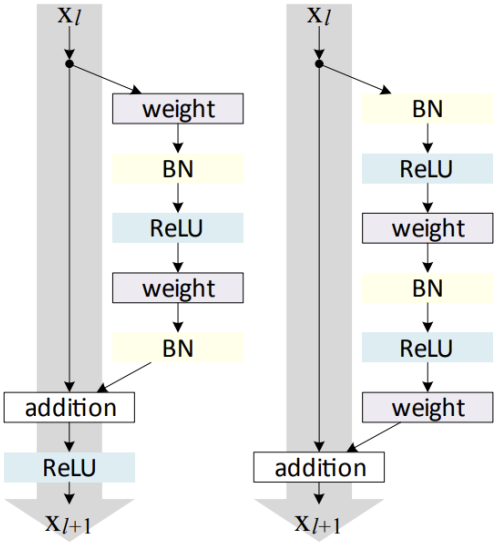
W 2016 roku opublikowano rewizję do oryginalnego modelu ([4]), którą nazwano ResNet V2. Modyfikuje ona blok rezydualny w taki sposób, aby wyeliminować występowanie funkcji aktywacji w „szynie" połączeń rezydualnych. Różnice w blokach przedstawiono na rys. 4.8.



Rysunek 4.6: Architektura ResNet34 porównana z analogiczną architekturą nierezydualną oraz VGG19 [3]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Rysunek 4.7: Warianty architektury ResNet [3]



Rysunek 4.8: Po lewej: blok rezydualny ResNet v1, po prawej: blok rezydualny ResNet v2 [4]

4.4. DenseNet

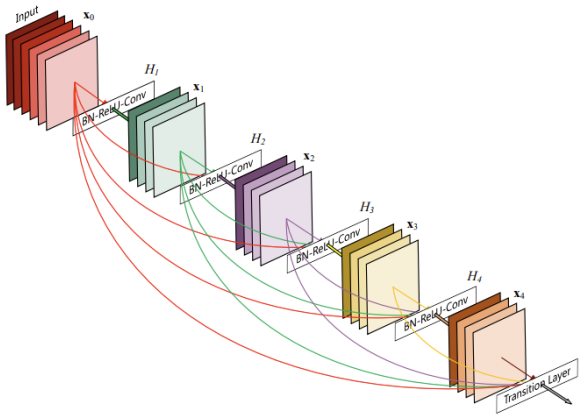
Architekturę DenseNet zdefiniowano w 2016 roku w publikacji [7]. Podobnie jak w przypadku ResNet, jej motywacją jest rozwiązanie problemu zanikających gradientów poprzez skrócenie ścieżki ich przepływu. Sposób osiągnięcia tego celu jest jednak inny – zamiast używać bloków rezydualnych, w których wejście jest dodawane do wyjścia, DenseNet używa bloków gęstych (*dense block*). Wewnątrz każdego bloku gęstego znajdują się bloki konwolucji 1×1 oraz 3×3. W obrębie pojedynczego bloku gęstego wyjście z każdego bloku konwolucyjnego jest łączone (*concatenation*) w wymiarze kanału z wejściem do każdego kolejnego. Graficzną reprezentację przykładowego bloku gęstego przedstawiono na rys. 4.9.

Gęste połączenia sprawiają, że każda warstwa w obrębie pojedynczego bloku gęstego ma bezpośredni dostęp do jego wyjścia, co znacznie skraca ścieżkę przepływu, nawet w porównaniu z ResNet50. Kolejną cechą odróżniającą DenseNet od ResNet jest liczba kanałów (szerokość) warstw konwolucyjnych. Jest ona typowo bardzo niska (np. 12 filtrów), ze względu na jawne rozróżnianie stanu poprzednich warstw oraz informacji dodanych. Każdy blok konwolucyjny zwiększa mapę cech wejściowych kolejnych warstw, jednocześnie nie modyfikując map cech z warstw poprzednich.

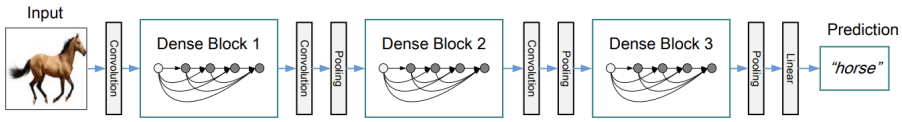
Podobnie jak w innych architekturach, w DenseNet występują redukcje wysokości oraz szerokości mapy cech. Zostały one nazwane *transition layer* i składają się z warstwy konwolucyjnej z filtrem 1×1 oraz *average pooling* z parametrem *stride* $= 2$, co skutkuje redukcją rozmiaru o połowę. Warto zauważyć, że połączenia gęste wymagają zachowania tej samej wysokości i szerokości. Oznacza to, że nie występują one pomiędzy blokami oddzielonymi warstwą *transition layer*, co przedstawiono na rys. 4.10.

Architektura DenseNet składa się z czterech bloków gęstych, trzech redukcji *transition layer*, warstwy klasyfikatora (7×7 *average pooling* oraz warstwa *fully-connected* z aktywacją *softmax*) oraz, podobnie jak w ResNet, warstwy konwolucyjnej z filtrem 7×7, *stride* $= 2$ i *max pooling*, *stride* $= 2$ na początku.

DenseNet występuje w czterech wariantach: DenseNet-121, DenseNet-169, DenseNet-201 oraz DenseNet-264. Przedstawione warianty różnią się rozmiarem ostatnich dwóch bloków gęstych, co przedstawiono na rys. 4.11. Warto wspomnieć, że w momencie zdefiniowania DenseNet osiągnęła najwyższą dokładność Top1 na zbiorze ImageNet, uzyskując lepsze wyniki niż wówczas dominująca ResNet, przy jednoczesnej redukcji parametrów. DenseNet-121 osiągnęła 75%, DenseNet-169 76,2%, DenseNet-201 77,42%, a DenseNet-264 77,85%.



Rysunek 4.9: Przykładowy blok gęsty DenseNet [7]



Rysunek 4.10: Architektura DenseNet [7]

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Rysunek 4.11: Warianty architektury DenseNet [7]

4.5. Inception

Architekturę Inception zdefiniowano w 2014 roku w pracy [13]. W tej sekcji opisano InceptionV3, InceptionResNetV2 i Xception. W odróżnieniu od większości testowanych modeli, nie są to jedynie przeskalowane wersje modelu bazowego, lecz odrębne architektury, poprawiające problemy poprzednich modeli.

4.5.1. Inception v1 i GoogLeNet

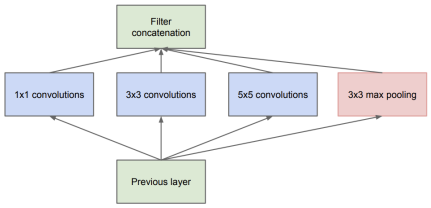
W 2014 roku dominującym podejściem w poprawianiu dokładności klasyfikatorów było zwiększanie ich głębokości oraz liczby filtrów. Autorzy zauważyli, że ta metoda skutkuje sieciami bardziej podatnymi na zjawisko przeuczenia (*overfitting*) oraz utrudnia przepływ gradientów. Ponadto zwrócono uwagę na znaczny wzrost kosztu obliczeniowego wraz ze wzrostem liczby sekwencyjnie ułożonych warstw konwolucyjnych z dużą liczbą filtrów. Kolejnym istotnym problemem jest dobór optymalnych rozmiarów filtrów w warstwach konwolucyjnych. Istotne cechy zdjęcia mogą znacząco różnić się rozmiarem (np. przez perspektywę lub odległość) i wymagać większego lub mniejszego pola receptywnego.

Aby rozwiązać powyższe problemy, zaproponowano naiwny blok Inception (rys. 4.12), który na tych samych danych wejściowych wykonuje trzy warstwy konwolucyjne z różnymi filtrami (1×1, 3×3, 5×5) oraz warstwę *max pooling* z oknem 3×3. W dalszej kolejności wyniki są łączone (*concatenation*) w wymiarze kanałów. Taka architektura bloku wiąże się jednak ze znacznym kosztem obliczeniowym – konwolucje 3×3 i 5×5 są bardzo kosztowne i wykonywanie ich bezpośrednio na tensorze wejściowym z wieloma kanałami będzie skutkować ogromną liczbą parametrów. Z tego powodu zaproponowano wariant z redukcją (rys. 4.13), w której wykorzystano dodatkowe warstwy konwolucyjne z filtrem 1×1, zmniejszające liczbę kanałów wejściowych. Warto zauważyć, że konkretne wartości redukcji oraz docelowych filtrów są ustalane osobno dla każdej warstwy w bloku i są typowo różne, przez co utworzenie bloku Inception wymaga określenia sześciu parametrów – liczby kanałów wyjściowych dla konwolucji 1×1, 3×3, 5×5 i *max pooling* oraz liczby zredukowanych kanałów wejściowych dla konwolucji 3×3 i 5×5. Przykładowe rozmiary i przepływ tensorów w dwóch blokach Inception v1 przedstawiono na rys. 4.14.

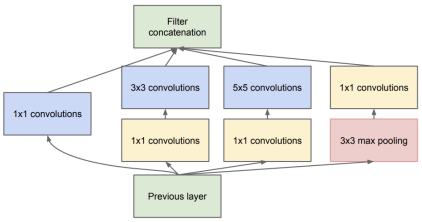
Autorzy zdefiniowali architekturę GoogLeNet, wykorzystującą dziewięć bloków Inception v1. Schemat architektury oraz parametry bloków przedstawiono na rys. 4.15. Aby przeciwdziałać problemowi zanikających gradientów i ułatwić ich przepływ, postanowiono dodać klasyfikatory pomocnicze (*auxiliary classifier*) po blokach 4a oraz 4d. Funkcja kosztu jest obliczana osobno dla każdego klasyfikatora. Następnie obliczony koszt klasyfikatorów pomocniczych jest mnożony przez *discount factor* (równy 0,3) i dodawany do całkowitego kosztu. Klasyfikatory pomocnicze biorą udział wyłącznie w treningu – podczas inferencji są one wyłączane.

4.5.2. Inception v2 i v3

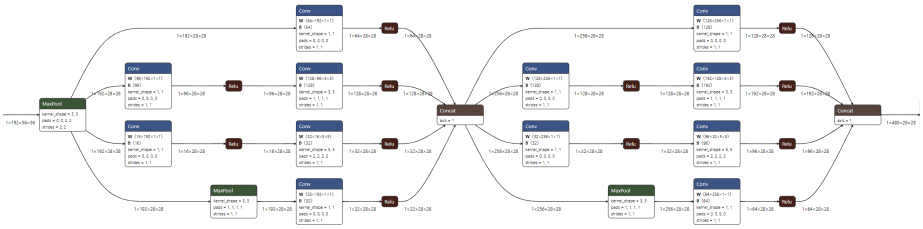
Architektury Inception v2 oraz v3 zostały zdefiniowane w 2015 roku w publikacji [14]. Stanowią one modyfikację Inception v1/GoogLeNet mającą na celu zmniejszenie utraty informacji oraz zwiększenie wydajności i złożoności obliczeniowej. Inception v3 osiąga 77,9% dokładności Top1 na zbiorze ImageNet oraz 93,7% Top5, co w 2015 roku ustanowiło nowy *state-of-the-art*. Schemat architektury Inception v2 przedstawiono na rys. 4.16.



Rysunek 4.12: Naiwny blok InceptionV1 [13]



Rysunek 4.13: Blok InceptionV1 z redukcją [13]



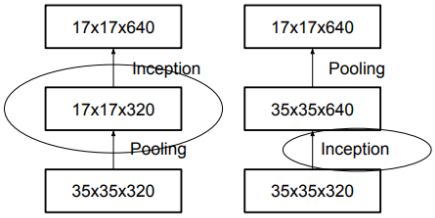
Rysunek 4.14: Przepływ tensorów w dwóch pierwszych blokach Inception v1

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

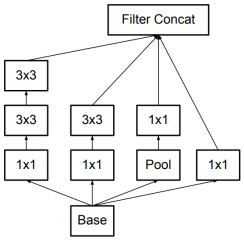
Rysunek 4.15: Architektura Inception v1 [13]

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

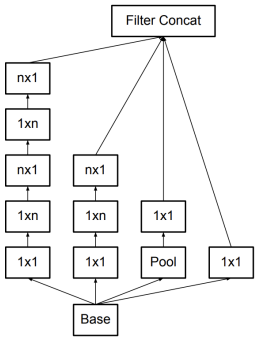
Rysunek 4.16: Architektura Inception v2 oraz Inception v3 [14]



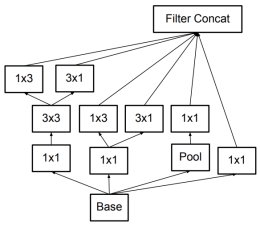
Rysunek 4.17: Po lewej: przykład zjawiska *representational bottleneck*, po prawej: droższa metoda redukcji wymiarowości, niewprowadzająca *representational bottleneck* [14]



Rysunek 4.18: Blok Inception v2 z faktoryzacją konwolucji 5×5 jako dwie konwolucje 3×3 [14]



Rysunek 4.19: Blok Inception v2 z faktoryzacją konwolucji $n \times n$ jako $1 \times n$ i $n \times 1$ [14]



Rysunek 4.20: Blok Inception v2 z rozszerzonymi filtrami wyjściowymi [14]

Jednym z przytoczonych problemów skutkujących utratą informacji jest zjawisko *representational bottleneck*. W Inception występuje ono przy zmniejszeniu rozmiaru mapy cech warstwą *pooling* z parametrem *stride* = 2, a następnie zwiększeniu liczby kanałów warstwą Inception. Przedstawiono to na lewym diagramie z rys. 4.17. Jeśli potraktować sieć jako potok danych, $17 \times 17 \times 320$ stanowi wąskie gardło. Na tym etapie następuje znaczna utrata informacji, które nie mogą być zrekonstruowane przez dalsze rozszerzenie potoku. W przypadku prawego diagramu z rys. 4.17 wąskie gardło stanowi tensor $17 \times 17 \times 640$, co umożliwia przekazanie dwukrotnie większej ilości informacji, jednak wykonywane operacje są kosztowniejsze obliczeniowo.

Jako sposób na zwiększenie wydajności obliczeniowej bloków Inception autorzy zaproponowali metody faktoryzacji warstw konwolucyjnych. Pierwszą zasto-

sowaną optymalizacją było rozbicie konwolucji 5×5 na dwie 3×3 (rys. 4.18). Autorzy odnotowali zwiększenie wydajności obliczeniowej przy pomijalnym zmniejszeniu mocy reprezentacyjnej. Kolejną optymalizacją było rozbicie konwolucji $n \times n$ na $1 \times n$ oraz $n \times 1$ (rys. 4.19). Przykładowo w miejscu splotu 3×3 blok Inception będzie zawierał konwolucję 1×3 oraz 3×1, co poskutkuje tą samą liczbą filtrów wyjściowych, przy redukcji kosztu obliczeniowego o 33%. Następną zaproponowaną modyfikacją bloku Inception było zwiększenie liczby filtrów wyjściowych, bez zjawiska *representational bottleneck*. Rozwiązano to przez redukcję głębokości rozbitych splotów i zwiększenie szerokości – konwolucje $1 \times n$ oraz $n \times 1$ operują na wspólnym wejściu (rys. 4.20).

W końcowej architekturze Inception v2 wykorzystano trzy wymienione rodzaje bloków Inception. Dla map cech o rozmiarze 35×35 wykorzystano blok z rys. 4.18, dla 17×17 blok z rys. 4.19, a dla 8×8 blok z rys. 4.20.

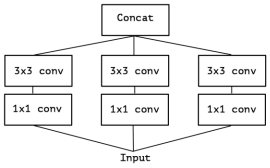
Inception v3 jest zmodyfikowanym wariantem Inception v2. Został wydzielony poprzez aplikowanie dodatkowych optymalizacji związanych z architekturą oraz procesem treningu. Zmiany w topologii są niewielkie: wprowadzono faktoryzację konwolucji 7×7 jako trzy konwolucje 3×3 oraz dodano operację *batch normalization* do klasyfikatorów pomocniczych. Proces treningu zmodyfikowano poprzez zmianę komponentu *optimizer* na RMSProp oraz zastosowanie techniki *label smoothing*.

4.5.3. Xception

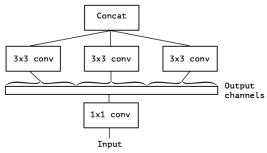
Architekturę Xception zdefiniowano w 2017 roku w publikacji [2]. Nazwa pochodzi od zaproponowanego bloku Inception – Extreme Inception. Powstał on z rozszerzenia uproszczonej wersji standardowego bloku Inception v3, w którym ograniczono wykonywane operacje do konwolucji 1×1 oraz 3×3 (rys. 4.21). Następnie przeformułowano ten blok, aby zawierał pojedynczy splot 1×1, a konwolucje 3×3 operują wyłącznie na części kanałów wyjściowych (rys. 4.22). Z tego powstała „ekstremalna" wersja bloku (rys. 4.23), w której każdemu kanałowi wyjściowemu splotu 1×1 odpowiada pojedyncza konwolucja 3×3.

Autorzy wskazują na podobieństwo „ekstremalnego" bloku Inception do *depthwise separable convolution* (opisanej w rozdziale 4.7.1) i proponują zastąpienie tą operacją konwencjonalnych bloków Inception. Splot 1×1 można interpretować jako konwolucję *pointwise*, a 3×3 jako *depthwise*. Typowo w *depthwise separable convolution* operacje te występują w odwrotnej kolejności (najpierw *depthwise*, potem *pointwise*), jednak autorzy zakładają, że nie jest to znacząca różnica. Dodatkowo w modułach Inception typowo między operacjami występuje nieliniowa funkcja aktywacji, nieobecna między konwolucjami *depthwise* oraz *pointwise*. Autorzy przeprowadzili badania, które wykazały, że brak nieliniowości między aktywacjami nie pogarsza procesu treningu oraz jakości klasyfikacji – wręcz przeciwnie, odnotowano szybsze osiągnięcie zbieżności oraz wyższą dokładność bez pośredniej funkcji aktywacji.

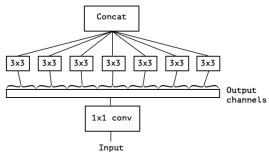
Traktując *depthwise separable convolution* jako „ekstremalny" blok Inception, autorzy zdefiniowali architekturę Xception, przedstawioną na rys. 4.24. Jest ona łatwa w definicji i modyfikacji w popularnych frameworkach, w po-



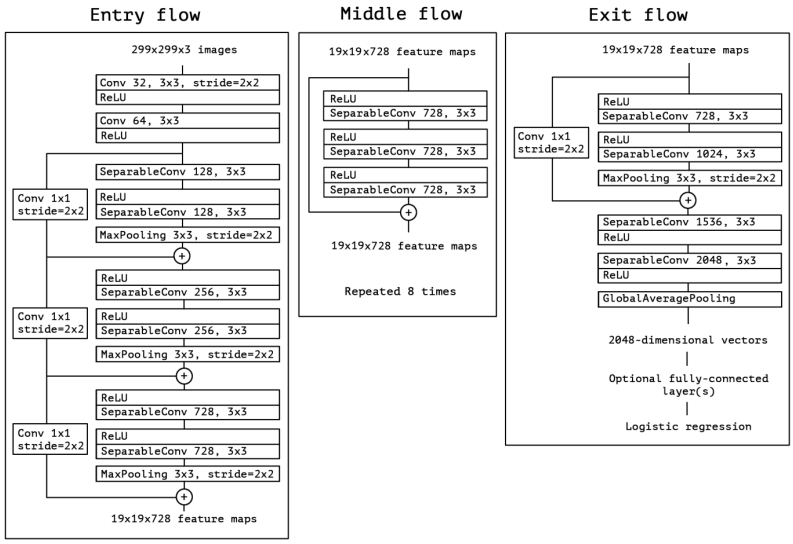
Rysunek 4.21: Uproszczony blok Inception [2]



Rysunek 4.22: Przeformułowany blok Inception z rys. 4.21 [2]



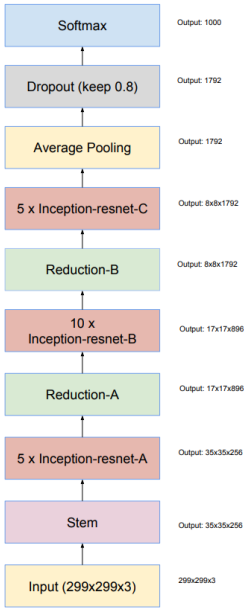
Rysunek 4.23: „Ekstremalna" wersja bloku Inception z rys. 4.22 [2]



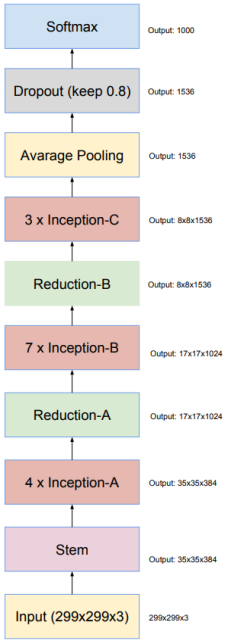
Rysunek 4.24: Architektura Xception [2]

równaniu z poprzednimi wersjami Inception. Dodatkowo Xception wykorzystuje połączenia rezydualne, podobnie jak ResNet. Wynikowy model składa się z trzech części, wykonywanych sekwencyjnie: *entry flow*, *middle flow* oraz *exit flow*. *Entry flow* modyfikuje wymiary mapy cech z $229 \times 229 \times 3$ do $19 \times 19 \times 728$. *Middle flow* zawiera powtórzony 8 razy blok rezydualny, w którym znajdują się trzy operacje *depthwise separable convolution* z funkcjami aktywacji ReLU przed każdą. Wymiary mapy cech nie ulegają zmianie. *Exit flow* redukuje mapę cech $19 \times 19 \times 728$ do wektora zawierającego 2048 elementów, na podstawie którego jest dokonywana klasyfikacja.

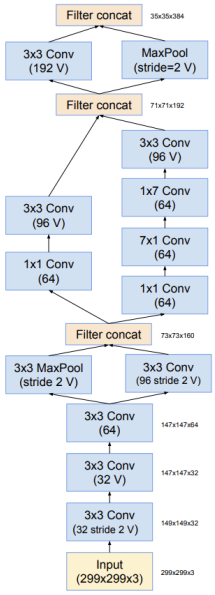
Xception uzyskuje wyniki lepsze niż Inception v3 na zbiorze ImageNet, z dokładnością Top1 równą 79% (poprawa o 1,1%) oraz Top5 94,5% (poprawa o 0,8%), przy zmniejszonej liczbie parametrów (22,86M względem 23,63M).



Rysunek 4.25: Architektura InceptionResNetV1 oraz InceptionResNetV2 [15]



Rysunek 4.26: Architektura Inception v4 [15]

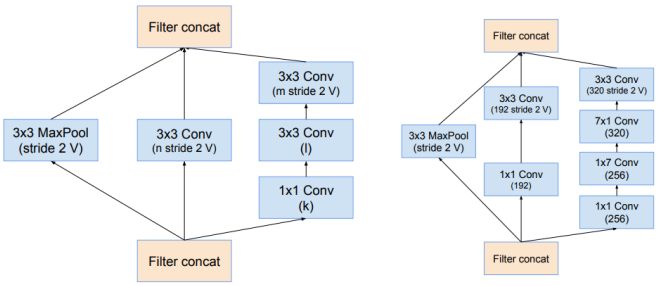


Rysunek 4.27: Moduł *stem* Inception v4 oraz InceptionResNetV2 [15]

4.5.4. Inception v4 i InceptionResNet

Architektury Inception v4, InceptionResNetV1 oraz InceptionResNetV2 zdefiniowano w 2016 roku w publikacji [15]. Głównym celem było uproszczenie i ujednolicenie architektury Inception oraz wdrożenie połączeń rezydualnych znanych z ResNet. InceptionResNetV2 osiągnęła największą dokładność z trzech przedstawionych modeli – Top1 80,3% oraz Top5 95,3% w konkursie ILSVRC. InceptionResNetV1 osiągnęło dokładności Top1 79,7% oraz Top5 94,5%, a InceptionV4 Top1 80% oraz Top5 95%.

Zaproponowane modele różnią się w znacznym stopniu – InceptionV4 jest „czystym" modelem Inception, niezawierającym połączeń rezydualnych. Jest bezpośrednim następcą Inception v3, lecz z prostszą architekturą oraz większą liczbą bloków Inception, co przekłada się na wyższą liczbę parametrów i wyższy koszt obliczeniowy. InceptionResNetV1 oraz InceptionResNetV2 to zmodyfikowane wersje architektury, zawierające bloki Inception z połączeniami rezydualnymi. InceptionResNetV1 i InceptionResNetV2 dzielą ten sam schemat architektury (przedstawiony na rys. 4.25), lecz bloki Inception oraz początkowa sekcja (*stem*) różnią się zawartością. Model InceptionResNetV1 jest zbliżony rozmiarem do Inception v3, a InceptionResNetV2 do większego Inception v4.



Rysunek 4.28: Moduły Reduction-A oraz Reduction-B dla Inception v4, InceptionResNetV1 i InceptionResNetV2 [15]

Architektura Inception v4 ma podobną strukturę, lecz różni się od InceptionResNet liczbą powtórzeń każdego bloku Inception oraz liczbą filtrów wyjściowych (rys. 4.26). Topologie zawierają jednak komponenty wspólne – moduł *stem* (rys. 4.27) jest taki sam dla InceptionResNetV2 oraz InceptionV4, a bloki Reduction-A oraz Reduction-B (rys. 4.28) są takie same dla wszystkich modeli.

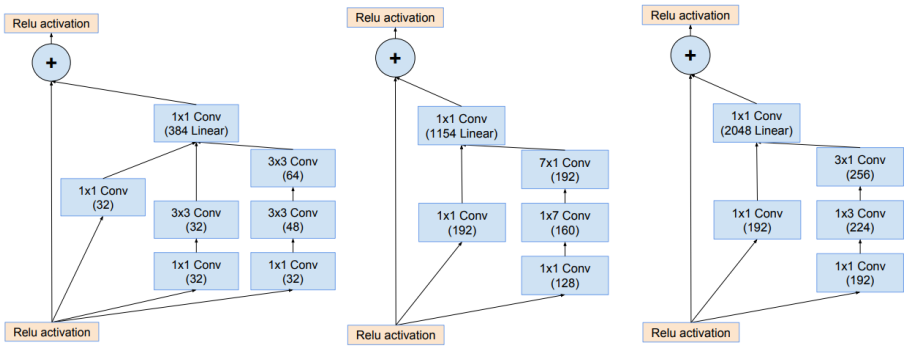
Po ustaleniu części wspólnych jedynym brakującym elementem architektur Inception v4 oraz InceptionResNetV2 są bloki Inception. Posiadają one odmienną topologię – bloki InceptionResNet (rys. 4.29) zawierają znacznie mniej warstw oraz nie wykorzystują operacji *average pooling*. Jednak przez większą liczbę powtórzeń cały model zawiera liczbę parametrów zbliżoną do Inception v4. Zaproponowana architektura bloków Inception v4 (rys. 4.30) nie odbiega znacząco od tych z Inception v3, różnią się jednak ich zastosowanie i liczba.

4.6. NASNet

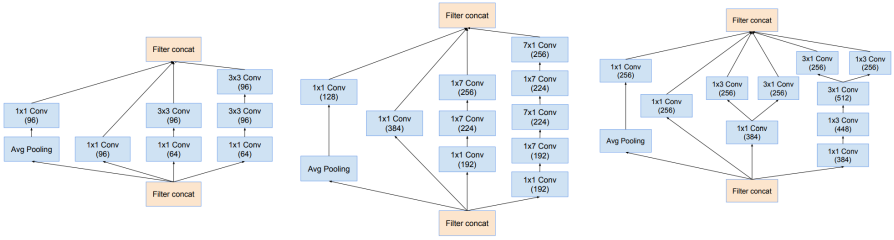
NASNet to architektura zdefiniowana w 2017 roku w publikacji [23]. Nazwa podchodzi od Neural Architecture Search – techniki umożliwiającej automatyzację projektowania architektur sieci neuronowych.

4.6.1. AutoML i Neural Architecture Search

Architektura NASNet jest ściśle powiązana z projektem stworzonym przez Google AI, znanym jako AutoML [10] lub Neural Architecture Search [22]. Polega on na potraktowaniu problemu znalezienia optymalnej architektury sieci neuronowej jako problem uczenia ze wzmocnieniem. Autorzy Neural Architecture Search zauważyli, że strukturę i połączenia w sieci neuronowej można przeważnie wyrazić w formie ciągu znaków o różnej długości. Z tego powodu zdefiniowano sieć rekurencyjną *controller*, której zadaniem jest generowanie ciągu znaków określających docelową sieć neuronową *child network*. Przykładowy kontroler przedstawiono na rys. 4.31. Następnie docelowa sieć jest trenowana na zadanym zbiorze danych, a jej osiągnięta dokładność jest traktowana jako nagroda



Rysunek 4.29: Bloki Inception dla InceptionResNetV2. Od lewej: Inception-A, Inception-B, Inception-C [15]



Rysunek 4.30: Bloki Inception dla InceptionV4. Od lewej: Inception-A, Inception-B, Inception-C [15]

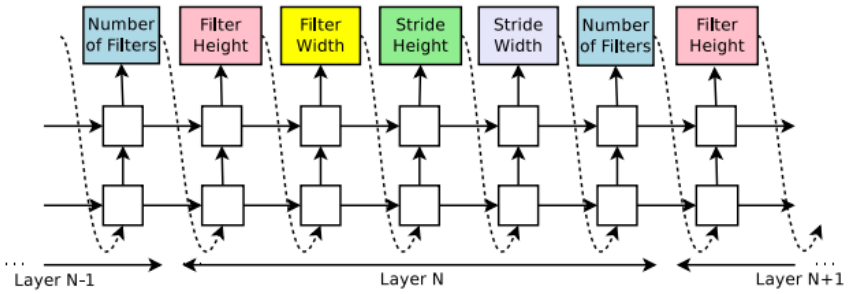
(*reward*) i wykorzystywana do aktualizacji gradientu polityki (*policy gradient*). Ten proces jest powtarzany wiele tysięcy razy, przy czym za każdym razem wymaga się trenowania *child network* na nowo.

Ze względu na bardzo wysoki koszt obliczeniowy AutoML była wykorzystywana początkowo jedynie na mniejszych zbiorach danych. Rozwiązywane problemy stanowiły klasyfikacja obrazów na zbiorze CIFAR-10 oraz tworzenie modelu językowego na zbiorze Penn Treebank, w których uzyskano dokładność na równi ze *state-of-the-art*.

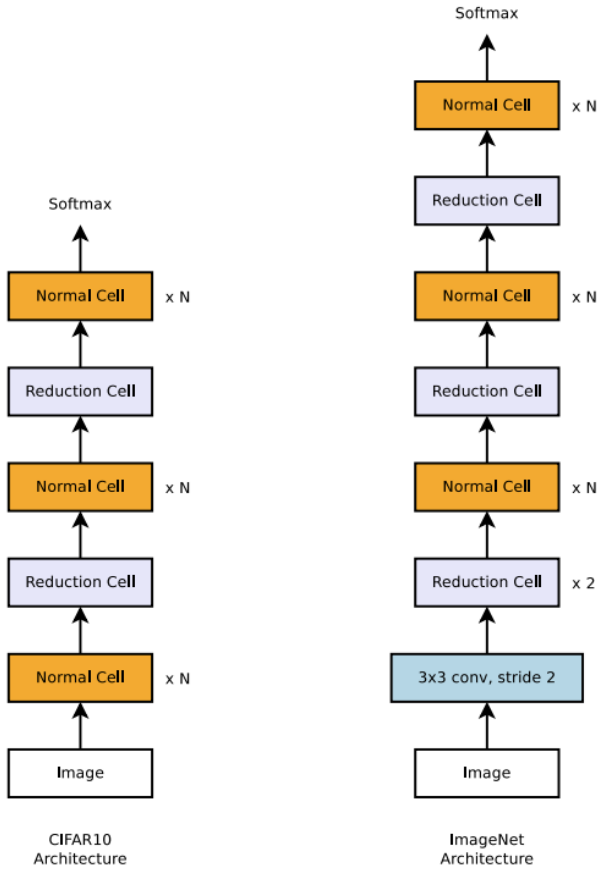
4.6.2. Neural Architecture Search dla ImageNet

NASNet powstała poprzez wykorzystanie techniki Neural Architecture Search/ AutoML na zbiorze ImageNet. Aby tego dokonać, znacznie zawężono przestrzeń wyszukiwania (*search space*). Autorzy zauważyli podobieństwa w strukturach współczesnych architektur konwolucyjnych sieci neuronowych i na tej podstawie ustalili ogólny kształt architektury, przedstawiony na rys. 4.32.

Wyróżniono dwa rodzaje bloków – *normal cell* oraz *reduction cell*. *Normal cell* oznacza blok, w którym wysokość i szerokość mapy cech są zachowane. W *reduction cell* wysokość i szerokość są redukowane do połowy źródłowego rozmiaru. Jedynie wewnętrzne struktury *normal cell* oraz *reduction cell* są



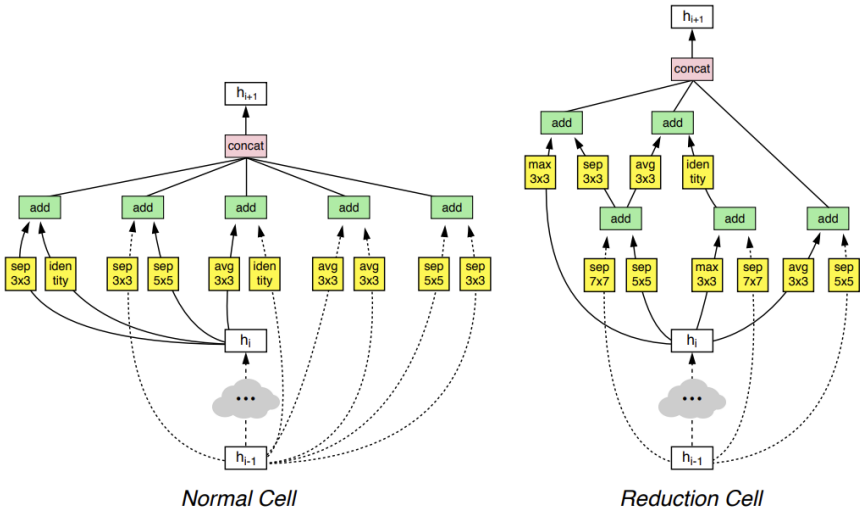
Rysunek 4.31: Przykładowy kontroler do generowania prostych sieci konwolucyjnych [22]



Rysunek 4.32: Ogólna architektura NASNet [23]

ustalane przez kontroler Neural Architecture Search/AutoML.

Finalnie wyznaczono trzy wersje architektury: NASNet-A, NASNet-B oraz NASNet-C. Strukturę bloków przedstawiono na rys. 4.33.



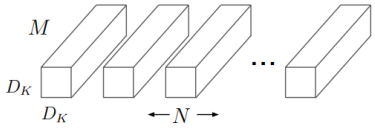
Rysunek 4.33: Wyznaczone bloki NASNet-A [23]

Zawężenie wyszukiwania do dwóch bloków 7-krotnie zredukowało czas wyznaczenia optymalnej architektury względem poprzednich prób (CIFAR-10 oraz Penn Treebank). Mimo to koszt obliczeniowy stanowi znaczącą przeszkodę w popularności tej metody – finalny czas trwania wyniósł 4 dni przy wykorzystaniu 500 GPU NVIDIA Tesla P100, co stanowi 48 000 GPU-godzin.

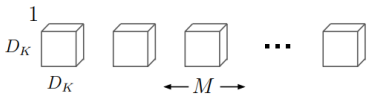
Modele NASNetMobile oraz NASNetLarge osiągają kolejno 74,4% oraz 82,5% dokładności Top1 na zbiorze ImageNet. Po uruchomieniu model NASNetLarge pobił rekord dokładności na zbiorze ImageNet. Oba modele dzielą tę samą strukturę bloków oraz podstawową architekturę modelu – różnią się jedynie współczynnikiem N oznaczającym liczbę powtórzeń bloku *normal cell* oraz liczbą cech w przedostatniej warstwie. W przypadku NASNetLarge, wynoszą one kolejno 6 i 4032, a w NASNetMobile 6 i 768.

4.7. MobileNet

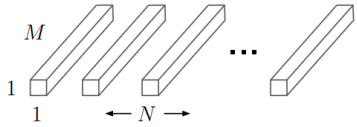
Architektura MobileNet została zdefiniowana w 2017 roku w pracy [6]. Jej głównym celem jest umożliwienie wydajnej inferencji na urządzeniach mobilnych i wbudowanych. Wykorzystanie głębokich sieci neuronowych typowo wymaga znacznych zasobów obliczeniowych. Wraz z redukcją rozmiaru modelu koszt obliczeniowy spada, jednak przeważnie wiąże się to także ze spadkiem



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Rysunek 4.34: Wizualizacja filtrów *depthwise separable convolution*: a) filtry wykorzystywane w standardowej konwolucji; b) filtry konwolucji *depthwise*; c) filtry konwolucji *pointwise* [6]

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Rysunek 4.35: Architektura MobileNet [6]

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Rysunek 4.36: Wpływ zamiany konwencjonalnych splotów na *depthwise separable convolution* na przykładzie MobileNet [6]

dokładności. Należy też rozważyć wydajność obliczeniową – zwiększanie głębokości sieci powyżej pewnego etapu będzie skutkować nieproporcjonalnie małą poprawą dokładności względem wzrostu kosztu obliczeniowego. Twórcy wskazują na ten problem i proponują architekturę zapewniającą optymalną wydajność obliczeniową, stanowiącą kompromis między dokładnością, rozmiarem oraz czasem inferencji.

4.7.1. *Depthwise separable convolution*

Wysoką wydajność obliczeniową osiągnięto za pomocą specjalnego rodzaju konwolucji, który określono jako *depthwise separable convolution*. W standardowej operacji splotu filtry są przeważnie reprezentowane jako tensory 4D, których wymiary odpowiadają wysokości (H) i szerokości (W) filtra oraz liczbie kanałów wejściowych (C) i wyjściowych (K). Filtry są wykorzystywane do przekształce-

nia obrazu wejściowego zawierającego C kanałów na nowy, K-kanałowy obraz poprzez kombinację liniową wartości w wymiarach C oraz K.

Aby zmniejszyć koszt obliczeniowy, *depthwise separable convolution* dzieli proces konwencjonalnego splotu na dwie fazy – filtrowania oraz kombinacji. Wykorzystano do tego dwie warstwy konwolucyjne (*depthwise* oraz *pointwise*) z filtrami o efektywnie niższej wymiarowości. Konwolucja *depthwise* aplikuje pojedynczy filtr na każdy kanał wejściowy, co sprawia, że można go przedstawić jako tensor 4D o przykładowym kształcie $(H, W, 1, C)$, który praktycznie jest trójwymiarowy. Etap ten aplikuje filtry na obraz wejściowy, jednak nie zmienia jego liczby kanałów. Do tego celu wykorzystuje się konwolucję *pointwise*, która jest standardowym splotem o rozmiarze filtra 1×1, z liczbą kanałów wejściowych C i wyjściowych K. Tensor filtrów będzie miał przykładowy kształt $(1, 1, C, K)$, co stanowi dwuwymiarową macierz. Wizualną reprezentację różnic w filtrach przedstawiono na rys. 4.34

Poza oszczędnością pamięciową twórcy odnotowują znaczną redukcję w obliczeniach, równą: $\frac{1}{N} + \frac{1}{K^2}$, gdzie N to liczba filtrów wyjściowych, a K to rozmiar kwadratowego filtra. W praktyce, wykorzystując *depthwise separable convolution*, autorzy odnotowali 8–9-krotną redukcję kosztu obliczeniowego względem konwencjonalnych splotów, przy niewielkim spadku dokładności Top1 na zbiorze ImageNet. Wyniki przedstawiono na rys. 4.36.

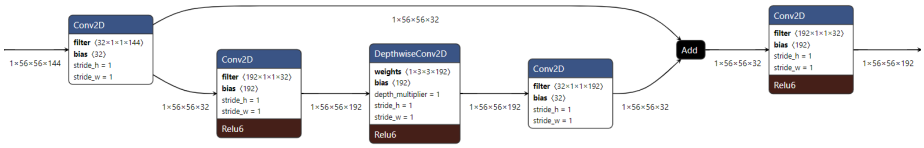
MobileNet

Oryginalny wariant MobileNet osiąga na zbiorze ImageNet dokładność Top1 70,6%. Sieć składa się z warstw konwolucyjnych *depthwise* (z filtrami 3×3) oraz *pointwise* (z filtrami 1×1), ułożonych sekwencyjnie. Nie występują połączenia rezydualne pomiędzy warstwami. Redukcja rozmiaru obrazu wyjściowego odbywa się podobnie jak w architekturze ResNet – z wykorzystaniem parametru *stride* 2. Wyjątkiem jest warstwa trzecia od końca, w której występuje redukcja rozmiaru z 7×7 do 1×1 z wykorzystaniem warstwy *average pooling*. Architekturę modelu przedstawiono na rys. 4.35.

MobileNetV2

Wariant MobileNetV2 zdefiniowano w pracy [11]. W dalszym ciągu architektura wykorzystuje warstwy *depthwise separable convolution*, jednak została dostosowana tak, aby wykorzystać połączenia rezydualne oraz możliwie zmniejszyć liczbę kanałów, przy zachowaniu wysokiej dokładności. Głównym budulcem sieci jest odwrócony blok rezydualny przedstawiony na rys. 4.3, który w MobileNetV2 jest używany jako warstwa *bottleneck*.

Odwrócony blok rezydualny (*bottleneck*) zawiera trzy konwolucje, które autorzy nazwali kolejno: *expansion*, *depthwise* oraz *projection*. *Expansion* to standardowa konwolucja 1×1, która rozszerza wymiar kanału o określony czynnik skalowania t (domyślnie równy 6). Warstwy *depthwise* oraz *projection* odpowiadają komponentom *depthwise separable convolution*, jednak z tą różnicą, że warstwa *projection* stanowi odwrotność warstwy *expansion*, redukując liczbę fil-



Rysunek 4.37: Przepływ tensorów w odwróconym bloku rezydualnym MobileNetV2 z czynnikiem skalowania $t = 6$

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

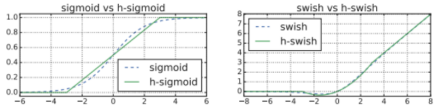
Rysunek 4.38: Architektura MobileNetV2 [11]

trów o ten sam czynnik t. Sprawia to, że rozmiary tensorów pomiędzy blokami rezydualnymi są relatywnie niewielkie, a samo filtrowanie odbywa się na wielu kanałach. Intuicyjnie można potraktować ten proces jako kompresję i dekompresję – warstwa *expansion* „dekompresuje" dane wejściowe, produkując znacznie większy tensor, *depthwise* dokonuje filtrowania, a następnie *projection* kompresuje przefiltrowany wynik z powrotem do źródłowej liczby kanałów. Przykładowy schemat działania bloku wraz z rozmiarami tensorów wejściowych i wyjściowych pokazano na rys. 4.37.

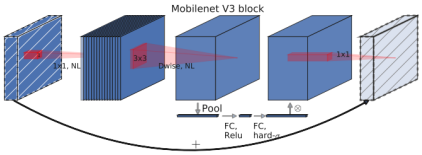
Całościowe spojrzenie na architekturę MobileNetV2 zostało przedstawione na rys. 4.38. Redukcję wymiarów obrazu wyjściowego zrealizowano podobnie jak w pierwszej wersji MobileNet – używając warstw konwolucyjnych z *stride* = 2. Wprowadzone zmiany w architekturze powodują wzrost dokładności Top1 na zbiorze ImageNet do 72,0%, przy spadku liczby trenowalnych parametrów (z 4,2M do 3,4M) oraz kosztu obliczeniowego (z 575M do 300M operacji *multiply/add*) względem pierwszej wersji.

MobileNetV3

Architekturę MobileNetV3 zdefiniowano w publikacji [5] w 2019 roku. Powstała ona dzięki wykorzystaniu techniki NAS, opisanej w rozdziale 4.6.1, oraz algorytmu NetAdapt, zdefiniowanego w pracy [19]. NetAdapt pełni istotną rolę w wyszukiwaniu MobileNetV3 – proponuje zmiany dla każdej architektury wygenerowanej przez kontroler, mające na celu zmniejszyć latencję sieci na określonym urządzeniu. W odróżnieniu od kolejnych kroków NAS, propozycje gene-



Rysunek 4.39: Aktywacje *h-swish* oraz *h-sigmoid* [5]



Rysunek 4.40: Blok *bottleneck* MobileNetV3 [5]

Input	Operator	exp size	*#out*	SE	NL	*s*
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

Rysunek 4.41: Architektura MobileNetV3Small [5]

Input	Operator	exp size	*#out*	SE	NL	*s*
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Rysunek 4.42: Architektura MobilenetV3Large [5]

rowane przez NetAdapt są jedynie modyfikacjami istniejącego modelu i używają już wytrenowanych wag. W przypadku zmian wymagających dodatkowych parametrów są one inicjalizowane losowo i dotrenowywane. Ze wszystkich propozycji modyfikacji wybierana jest najdokładniejsza. Dodatkowo po wyznaczeniu modelu z NAS + NetAdapt autorzy zmodyfikowali wynikową architekturę, zamieniając najbardziej kosztowne obliczeniowo warstwy. Ponadto wykorzystywana jest autorska, wydajna obliczeniowo i ułatwiająca kwantyzację, funkcja aktywacji *h-swish* (rys. 4.39) oraz *block bottleneck* (rys. 4.40).

Strukturę modeli MobileNetV3Small i MobileNetV3Large przedstawiono kolejno na rys. 4.41 oraz 4.42. Osiągają one dokładność Top1 na ImageNet kolejno 75,2% oraz 67,4%, przy 5,4 oraz 2,5 miliona parametrów.

4.8. VGG

VGG jest najstarszą opisaną w tym rozdziale architekturą, zdefiniowaną w 2014 roku w publikacji [12]. Określono dwie sieci wykorzystujące tę architekturę: VGG16 oraz VGG19. Oba modele zawierają bardzo dużą liczbę parametrów – kolejno 138,3 miliona oraz 143,6 miliona. Na datasecie ImageNet obie osiągają dokładność Top1 71,3%. VGG-16 osiąga dokładność Top5 90,1%, a VGG19

90%. W momencie zdefiniowania VGG ustanowiła nowe *state-of-the-art*, lecz obecnie została wyparta przez nowe architektury, osiągające większą dokładność przy znacznie niższej liczbie parametrów.

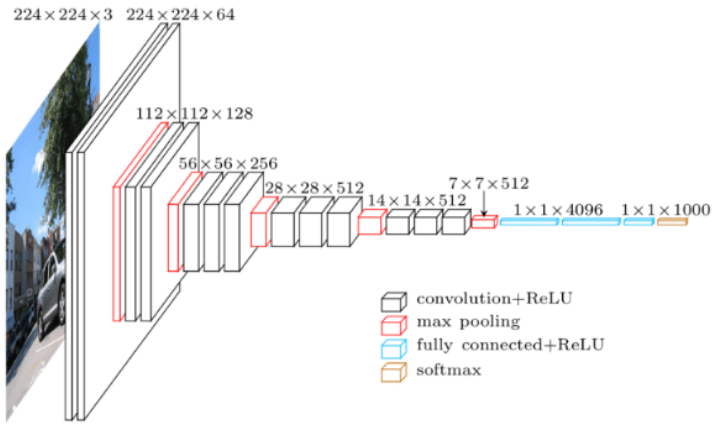
Nazwy modeli z tej rodziny wynikają z liczby trenowalnych warstw. Finalna liczba warstw jest większa – w tym przypadku tylko konwolucje oraz warstwy gęste posiadają trenowalne wagi. Model VGG16 zawiera 22 warstwy – 13 warstw konwolucyjnych, 3 warstwy gęste, 5 warstw *max pooling* oraz 1 warstwę *softmax*. VGG19 różni się od VGG16 tym, że zawiera trzy warstwy konwolucyjne więcej.

Wszystkie konwolucje wykorzystują fitr 3×3, krok (*stride*) 1 oraz używają paddingu „same" – oznacza to, że wymiary wyjściowe będą takie same jak wejściowe. Warstwy *max pooling* także funkcjonują w jednolity sposób – rozmiar każdej wynosi 2×2 i są przesuwane z krokiem 2×2. Powoduje to zmniejszenie wymiaru wysokości i szerokości o połowę.

W obu modelach dwie pierwsze konwolucje z 64 filtrami są wykonywane na obrazie wejściowym o wymiarze 224×224. Następnie wykonywana jest warstwa *max pooling*, redukując dwukrotnie wysokość i szerokość. Kolejno wykonuje się dalsze dwie konwolucje, tym razem z 128 filtrami, co skutkuje tensorem o rozmiarze $112 \times 112 \times 128$. Później występuje blok zawierający kolejną warstwę *max pooling* oraz warstwy konwolucyjne z 256 filtrami. W VGG19 jest ich cztery, a w VGG16 trzy. Ten blok jest powtarzany dwa razy, z podwojoną liczbą filtrów. Po pięciu operacjach *max pooling* obraz będzie zmniejszony 32-krotnie (pięciokrotne zmniejszenie o połowę, 2^5), co będzie skutkowało tensorem $7 \times 7 \times 512$. Następnie jest on przekazywany do dwóch warstw gęstych, każda o wymiarze wejściowym 4096 (pierwsza warstwa będzie posiadała trenowalną macierz wag o wymiarze $(7 * 7 * 512) \times 4096$, a druga 4096×4096). Ostatnia warstwa odpowiada za zmniejszenie wymiarowości do liczby docelowych klas, a następująca po niej warstwa *softmax* zamieni jej wyniki na rozkład prawdopodobieństwa przynależności do klas. Graficzną reprezentację architektury VGG można zaobserwować na rys. 4.43. Rysunek przedstawia VGG16, lecz można stąd łatwo uzyskać wizualizację VGG19, dodając po jednej dodatkowej warstwie konwolucyjnej w trzech ostatnich blokach.

4.9. Podsumowanie

Lista opisanych w tym rozdziale architektur sieci neuronowych nie jest wyczerpująca, a klasyfikacja obrazów nadal pozostaje otwartym problemem. *State-of-the-art* w konkursie ILSVRC jest stale przebijany przez nowe propozycje modeli, z których wiele stanowi ewolucję starszych architektur z zastosowaniem innowacyjnych ulepszeń bądź zapożyczeń z innych poddziedzin uczenia głębokiego. W momencie pisania tego rozdziału *state-of-the-art* w konkursie ILSVRC wynosi 91,00% dokładności Top1 z wykorzystaniem nowatorskiego modelu CoCa [20]. Rekord ten został ustanowiony 4 maja 2022 roku, przebijając tym samym poprzedni *state-of-the-art*, wynoszący 90,94%, ustanowiony 10 marca 2022 roku, z użyciem techniki „Model soups" [18]. Obecnie zauważyć można trend zmierzający ku adaptacji technik znanych z przetwarzania języka naturalnego



Rysunek 4.43: Architektura VGG16
Źródło: `https://www.researchgate.net/figure/VGG16-architecture-16_fig2_321829624`
Grafika na licencji Creative Commons Attribution 3.0 Unported, nie została zmodyfikowana

– mechanizmu uwagi oraz transformerów [17] w połączeniu z konwolucjami. Te rozwiązania stale się zmieniają, jednak innowacje często powstają na bazie wniosków wyciągniętych ze starszych architektur.

Bibliografia

[1] Alex Krizhevsky Vinod Nair G.H. *Strona główna datasetu CIFAR-10. https://www.cs.toronto.edu/~kriz/cifar.html.*

[2] Chollet F. *Xception: Deep Learning with Depthwise Separable Convolutions. arXiv:1610.02357*, 2017.

[3] He K., et al. *Deep Residual Learning for Image Recognition. arXiv:1512.03385*, 2015.

[4] He K., et al. *Identity Mappings in Deep Residual Networks. arXiv:1603.05027*, 2016.

[5] Howard A., et al. *Searching for MobileNetV3. arXiv:1905.02244*, 2019.

[6] Howard A.G., et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861*, 2017.

[7] Huang G., et al. *Densely Connected Convolutional Networks. arXiv:1608.06993*, 2018.

[8] Ioffe S., Szegedy C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167*, 2015.

[9] Papert S.A. *The Summer Vision Project. https://dspace.mit.edu/handle/1721.1/6125*, 1966.

[10] Quoc Le & Barret Zoph Research Scientists G.B.t. *Using Machine Learning to Explore Neural Network Architecture. https://ai.googleblog.com/2017/05/using-machine-learning-to-explore.html*, 2017.

[11] Sandler M., et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381*, 2019.

[12] Simonyan K., Zisserman A. *Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556*, 2015.

[13] Szegedy C., et al. *Going Deeper with Convolutions. arXiv:1409.4842*, 2014.

[14] Szegedy C., et al. *Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567*, 2015.

[15] Szegedy C., et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. arXiv:1602.07261*, 2016.

[16] Tan M., Le Q.V. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946*, 2020.

[17] Vaswani A., et al. *Attention Is All You Need. arXiv:1706.03762*, 2017. URL `https://arxiv.org/abs/1706.03762`.

[18] Wortsman M., et al. *Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. arXiv:2203.05482*, 2022. URL `https://arxiv.org/abs/2203.05482`.

[19] Yang T.J., et al. *NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. arXiv:1804.03230*, 2018.

[20] Yu J., et al. *CoCa: Contrastive Captioners are Image-Text Foundation Models. arXiv:2205.01917*. URL `https://arxiv.org/abs/2205.01917`.

[21] Zagoruyko S., Komodakis N. *Wide Residual Networks. arXiv:1605.07146*, 2017.

[22] Zoph B., Le Q.V. *Neural Architecture Search with Reinforcement Learning. arXiv:1611.01578*, 2017.

[23] Zoph B., et al. *Learning Transferable Architectures for Scalable Image Recognition. arXiv:1707.07012*, 2018.

Rozdział 5

Segmentacja obrazów medycznych przy ograniczonej liczbie adnotacji

Tomasz Gruzdzis[1] , Krzysztof Wicki[1] , Patrycja Gładkowska[1] , Tomasz Boiński[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
tomasz.boinski@pg.edu.pl

Abstrakt

W dziedzinie badań klinicznych i opieki zdrowotnej tradycyjne podejście w uczeniu głębokim polegające na wykorzystaniu dużych zbiorów danych jest trudne w realizacji. Przyczyną takiego stanu rzeczy są koszty znakowania obrazów medycznych, zwłaszcza w przypadku segmentacji obrazów medycznych. Jest to żmudna operacja, która zazwyczaj wymaga intensywnego znakowania pikseli wykonanego przez ekspertów – lekarzy. W tym rozdziale zaprezentowano podejście bazujące na wykorzystaniu metod uczenia maszynowego w semantycznej segmentacji obrazów medycznych. Celem badań była taka realizacja segmentacji, aby zminimalizować konieczną liczbę pełnych adnotacji na wybranym zbiorze danych. W tym celu zaproponowano użycie sieci nnU-Net. Skupiono się na zbadaniu, czy uczenie modelu z wykorzystaniem ograniczonej liczby adnotacji jest możliwe.

Słowa kluczowe: uczenie głębokie, segmentacja obrazów, redukcja zbioru

5.1. Wstęp

Niezależnie od zastosowań, metody uczenia głębokiego osiągają bardzo dobre rezultaty. Większość najnowocześniejszych modeli bazuje jednak w dużej mierze na treningu z wykorzystaniem dużych zestawów danych oznaczonych adnotacjami. Niestety w wielu dziedzinach dane takie często są niedostępne, np. w szeroko rozumianej dziedzinie badań klinicznych i opieki zdrowotnej [12]. Przyczyną takiego stanu rzeczy są koszty znakowania obrazów medycznych, zwłaszcza w przypadku segmentacji obrazów medycznych [16]. Jest to żmudna operacja, która zazwyczaj wymaga intensywnego znakowania pikseli wykonanego przez ekspertów – lekarzy. Silna zdolność uczenia się i uogólniania na podstawie ograniczonego nadzoru, w tym ograniczonej liczby, a do tego nielicznych i niedokładnych adnotacji, ma kluczowe znaczenie dla pomyślnego zastosowania modeli głębokiego uczenia w segmentacji obrazów medycznych. Jednakże ze względu na jego nieodłączną złożoność proces segmentacji z ograniczonym nadzorem jest trudny i potrzebne są specyficzne projekty modeli lub strategie uczenia się.

W niniejszym rozdziale zaprezentowano podejście bazujące na wykorzystaniu metod uczenia maszynowego w semantycznej segmentacji obrazów medycznych. Celem badań była taka realizacja segmentacji, aby zminimalizować konieczną liczbę pełnych adnotacji na wybranym zbiorze danych. W tym celu zaproponowano użycie sieci nnU-Net [11]. Na obecnym etapie celem nie jest wskazanie metod selekcji danych umożliwiających utworzenie reprezentacji. W rozdziale skupiamy się na zbadaniu, czy takie działanie w ogóle jest możliwe. Celem jest określenie, czy możliwe jest wyznaczenie takiego podzbioru, który zawierałby tylko reprezentatywne dane, który umożliwi utworzenie modelu sieci o parametrach porównywalnych jak model uczony pełnymi danymi.

W sekcji 5.2 opisano metody segmentacji obrazów oraz architekturę sieci nnU-Net. W sekcji 5.3 zaprezentowano zbiór danych użyty w eksperymentach. Następnie, w sekcji 5.4, zaprezentowano przeprowadzone eksperymenty i uzyskane rezultaty. Na końcu zamieszczono krótkie podsumowanie.

5.2. Segmentacja obrazów

5.2.1. Częściowo nadzorowana segmentacja

Częściowo nadzorowana segmentacja to typowy scenariusz w zastosowaniach medycznych, w którym zakłada się, że tylko niewielki podzbiór obrazów szkoleniowych zawiera pełne adnotacje w pikselach. Zazwyczaj dostępnych jest również wiele obrazów bez adnotacji, które można wykorzystać do poprawy zarówno dokładności adnotacji, jak i możliwości uogólniania systemów wykorzystywanych w medycynie. Ponieważ dane nieoznaczone nie wymagają pracochłonnych adnotacji, każdy wzrost wydajności wynikający z używania danych nieoznaczonych wiąże się z niskim kosztem. Główne wyzwanie związane z tym scenariuszem uczenia się polega na tym, jak skutecznie i dokładnie wykorzystać dużą

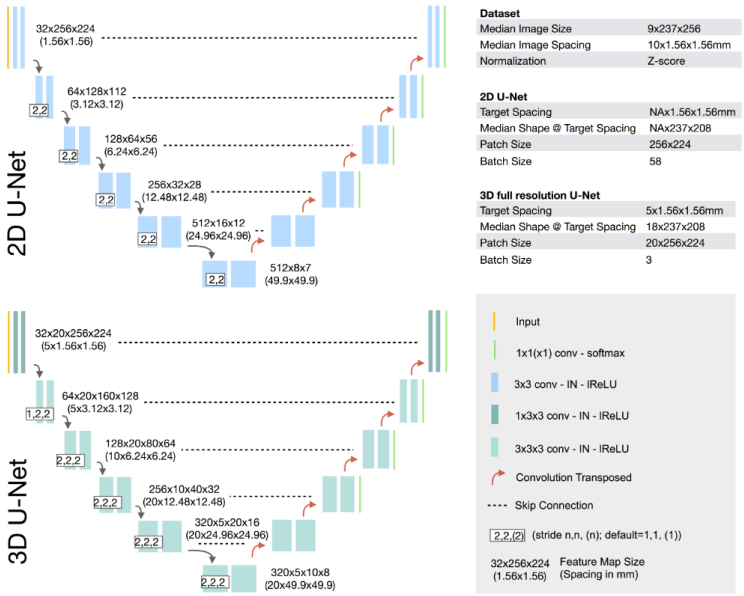
ilość nieoznakowanych danych. Najczęstsze podejścia do segmentacji częściowo nadzorowanej możemy podzielić na dwie grupy. Pierwsza to strategie ogólne, takie jak: *transfer learning*, *data augmentation*, *prior knowledge learning*, *curriculum learning*, i *few-shot learning*. Druga grupa to metody specjalistyczne, które wykorzystują dane nieoznakowane, np. *self-training* [17], regularyzacja spójności [10], *co-training*, *self-supervised learning* i *adversarial learning*.

W tej pracy skupiamy się na podejściu częściowo nadzorowanym, z wykorzystaniem reprezentacji obrazu i wyselekcjonowaniem najistotniejszych z nich w celu stworzenia nowego, pomniejszonego zbioru uczącego. Zbiór taki powinien mieć maksymalnie małą stratę względem wyników możliwych do uzyskania z wykorzystaniem pełnego zbioru. Jako reprezentacje obrazu przyjmujemy wektor cech tego obrazu [1]. Podejście to jest często stosowane w badaniach dotyczących porównywania podobieństw obrazów do reprezentacji ich cech [8, 9]. Posiadając reprezentacje obrazów, można spróbować je pogrupować w sposób nienadzorowany, tzn. poddać je klasteryzacji [4].

5.2.2. Architektura sieci nnU-Net

U-Net [11] jest narzędziem pozwalającym na automatyzację procesu uczenia sieci oraz ustalanie hiperparametrów uczenia. Narzędzie to nie wymaga dużej ingerencji w samą strukturę sieci co sprawia, że jest uniwersalne w zastosowaniu. Od momentu publikacji narzędzie to wyznaczyło nowy standard, z którym porównuje się wszystkie inne narzędzia do segmentacji obrazów medycznych. Autorzy przewidzieli trzy różne możliwości zastosowania sieci: 2D, 3D oraz działanie kaskadowe. Jednym z powodów, dla których U-Net osiąga tak dobre rezultaty, jest wykorzystanie augmentacji obrazów do zwiększania możliwości uczenia się sieci. Techniki, jakie zastosowano, to skalowanie o losową wartość (*random scaling*), losowe deformacje (*random elastic deformations*), modyfikacja korekcji gamma (*gamma correction augmentation*) oraz tworzenie odbicia lustrzanego (*mirroring*).

Architektura sieci nnU-Net opiera się na schemacie enkodera i dekodera [6] (rys. 5.1). Enkoder zmniejsza wymiary przestrzenne na każdej warstwie, jednocześnie zwiększając liczbę kanałów, natomiast dekoder zwiększa wymiary przestrzenne, zmniejszając liczbę kanałów. Tensor podawany pomiędzy enkoderem i dekoderem nazywamy *bottleneckiem*. Na koniec wymiary są przywrócone do oryginalnych, a sieć dokonuje predykcji dla każdego piksela zdjęcia podanego na wstępie. Enkoder składa się z dwóch splotów 3×3, po każdym z nich następują ReLU oraz normalizacja wsadowa (*batch normalization*). Następnie wykonywana jest operacja 2×2 *max-pooling* w celu zmniejszenia wymiaru przestrzennego. Na każdym etapie zmniejszamy o połowę wymiar przestrzenny i zwiększamy liczbę kanałów. Dekoder zawiera skalowanie w górę (*upsampling*) mapy cech (*feature map*), po którym następuje splot transponowany, który zmniejsza liczbę kanałów o połowę. Kolejną warstwą jest splot 3×3 wraz z ReLU. W skład ostatniej warstwy wchodzi splot 1×1, służący do mapowania kanałów do wybranej liczby klas.



Rysunek 5.1: Architektura sieci nnU-Net [6]

5.3. Zbiór danych ISIC

W trakcie prac wykorzystano zbiór danych ISIC 2018 [5, 15], zawierający obrazy dermatoskopowe skóry, przeznaczony do analizy zmian skórnych w kierunku wykrywania czerniaka.

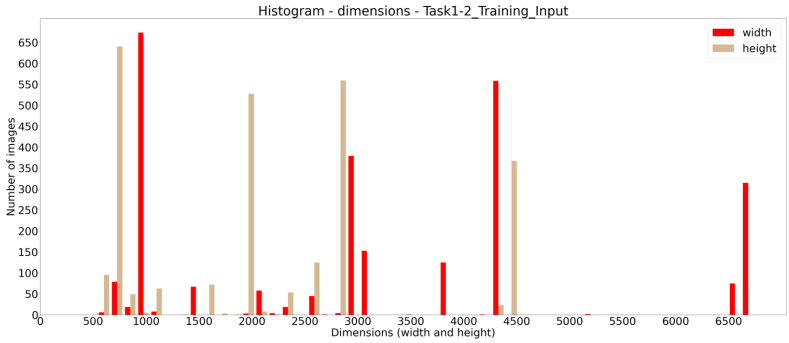
Zbiór obejmuje:

- zestaw treningowy – 2594 obrazów;
- zestaw walidacyjny – 100 obrazów;
- zestaw testowy – 1000 obrazów (bez masek segmentacji).

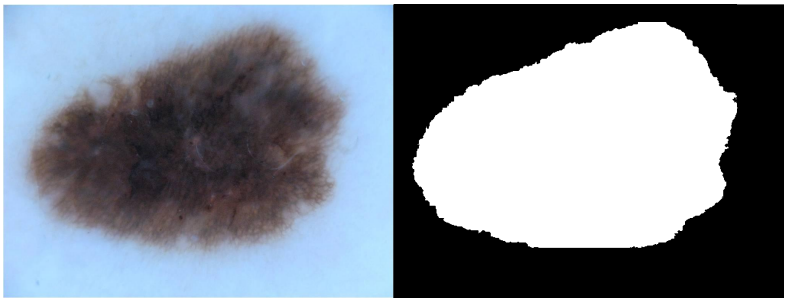
Rozkład liczebności i wymiarów zestawu treningowego przedstawiają histogramy na rys. 5.2. Przykładowa próbka widoczna jest na rys. 5.3.

5.4. Eksperymenty

W pierwszym etapie realizacji eksperymentu definiowana jest liczba obrazów n, jakie chcemy, by zawierał nasz zestaw danych. Dla wszystkich obrazów wykorzystanych w uczeniu pełnego modelu utworzono wektory cech. Wektory cech poddawane są przetwarzaniu wstępnemu, gdzie stosowane są normalizacja i redukcja wymiarowości. Wymiar redukowany jest przy użyciu algorytmu PCA z zachowaniem 99% wariancji. Następnie, przy użyciu algorytmu K-Means,



Rysunek 5.2: Wykres liczebności obrazów w zależności od ich wymiarów (wysokość i szerokość)



Rysunek 5.3: Przykład obrazu ze zbioru ISIC [5, 15] wraz z maską segmentacji

dane podlegają klasteryzacji na n grup, gdzie n to liczba obrazów określona przez użytkownika przed rozpoczęciem procesu. Po dokonanej klasteryzacji wyznaczane są centroidy utworzonych klastrów. Względem nich, przy użyciu biblioteki sklearn, wybierany jest punkt najbliżej centroida. Na bazie współrzędnych tego punktu wyznaczane są próbki będące najbliżej środka każdego z klastrów. Wybrane w ten sposób obrazy są wykorzystywane następnie jako zbiór uczący do treningu modelu nnU-Net.

nnU-Net [11] oferuje szereg metryk służących do ewaluacji uzyskanych rezultatów. W niniejszym rozdziale zdecydowano o użyciu trzech metryk [14]:

- Accuracy (dokładność) – mówi o tym, jak bardzo wyniki prognozy zgadzają się z rzeczywistą etykietą (*ground truth*, GT); jest to stosunek wszystkich prawdziwych przypadków (*true positive*, TP) do wszystkich prawdziwych i fałszywych przypadków (*true/false positive*, TP/FP);

- Dice – definiowana jako miara odtwarzalności; jest to najczęściej stosowana metryka w zagadnieniach segmentacji obrazów medycznych; definiuje się ją jako stosunek dwukrotności TP do sumy dwukrotności TP, wartości FP i wartości FN;

- Jaccard – zdefiniowany jako przecięcie dwóch zbiorów podzielone przez

ich sumę; innymi słowy, jest to stosunek TP do sumy TP i wszystkich fałszywych przypadków.

Niestety autorzy różnych opracowań korzystają z różnych miar w celu prezentacji wyników swoich badań, np.:

- [7] wskazuje Meanoverlapscore dla zestawu testowego równy 0,776 przed zerowaniem przypadków, które nie osiągnęły progu 0,65, a 0,701 po zerowaniu;
- [13] wskazuje Jaccard dla zestawu testowego o wartości 0,825 oraz Jaccard progowany na 0,65 równy 0,787;
- [2] wskazuje Meanoverlap dla zestawu testowego/walidacyjnego równy 0,735;
- [5] wskazuje Jaccard dla zestawu testowego równy 0,786;
- [3] wskazuje Jaccard progowany na 0,65 równy 0,694, 0,686 i 0,728 odpowiednio dla zbioru treningowego, testowego i walidacyjnego.

Rysunek 5.4: Przebieg treningu 1

W trakcie eksperymentów przeprowadzono łącznie pięć treningów sieci nn-UNet. W trakcie treningu wykorzystany został zestaw walidacyjny w stałym, niezmiennym rozmiarze dla wszystkich eksperymentów. Wypracowane modele zostały wykorzystane do przeprowadzenia predykcji na zestawie walidacyjnym, a więc tym, który wykorzystany był podczas treningu, co stanowić będzie różnicę względem referencyjnych publikacji (gdzie predykcja wykonywana była na

osobnym zbiorze testowym). Podyktowane jest to faktem braku dostępu do pełnego zbioru testowego – autorzy opracowań udostępniają jedynie same obrazy bez dostępu do ich masek.

W pierwszym kroku przeprowadzono trzy treningi sieci, różniące się od siebie rozmiarem zestawu treningowego. Modele trenowano całością lub częścią zbioru testowego – wykorzystano cały zbiór treningowy, 20% jego elementów i 10% jego elementów. Mniejsze zbiory utworzone zostały poprzez pobranie losowych próbek w liczbie 10% i 20% całego zestawu treningowego. Dla obu przypadków wykonano dwa podejścia do treningu – dwukrotnie została wykonana losowa selekcja właściwej liczby obrazów. Zaprezentowane dane są danymi uśrednionymi.

Trening 1, pełny zestaw danych – 100% danych. Przebieg treningu widać na rys. 5.4. Kolorami czerwonym i niebieskim opisano stratę odpowiednio dla obrazów ze zbiorów treningowego i walidacyjnego. Kolor zielony natomiast oznacza metrykę ewaluacyjną – *averageDicescore*.

Jak można zauważyć, została osiągnięta niska strata zarówno dla zbioru uczącego, jak i walidacyjnego na poziomie 0,6–0,65.

Wyniki predykcji dla zestawu walidacyjnego kształtują się następująco:

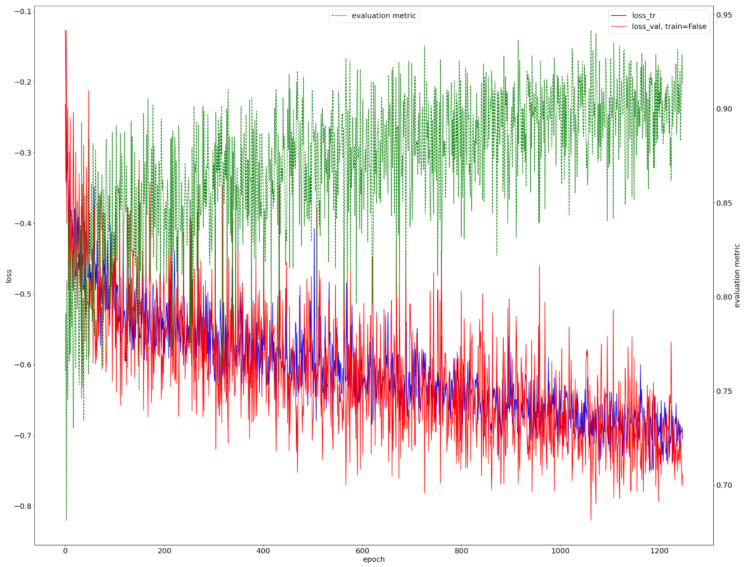
- Accuracy: 0,90;
- Dice: 0,81;
- Jaccard: 0,72.

Uzyskane wyniki są zbliżone do tych wskazywanych przez opracowania innych autorów. W celu ich dalszej weryfikacji w trakcie badań przeprowadzono trening kilkukrotnie. Wydłużanie treningu z wyjściowego wynoszącego 1000 epok nie poprawiło znacznie wyników.

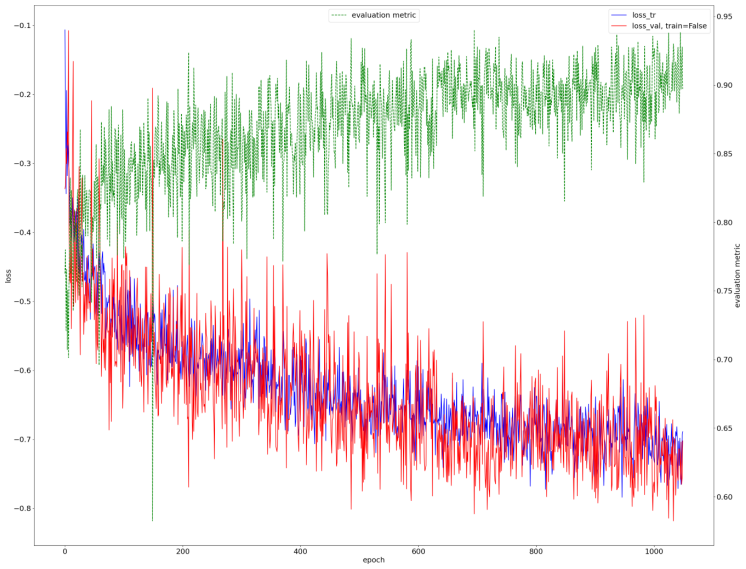
Trening 2, niepełny zestaw danych – 20% losowo dobranych danych. Przebieg treningu widać na rys. 5.5. Kolorami czerwonym i niebieskim opisano stratę odpowiednio dla obrazów ze zbiorów treningowego i walidacyjnego. Kolor zielony natomiast oznacza metrykę ewaluacyjną – *averageDicescore*. Czas treningu wyniósł tym razem 1300 epok, jednak ogólny wynik strat jest nieco wyższy względem poprzedniego eksperymentu.

Trening 3, niepełny zestaw danych – 10% losowo dobranych danych. Przebieg treningu widać na rys. 5.6. Kolorami czerwonym i niebieskim opisano stratę odpowiednio dla obrazów ze zbiorów treningowego i walidacyjnego. Kolor zielony natomiast oznacza metrykę ewaluacyjną – *averageDicescore*. Trening obejmował 1000 epok. Wyniki ponownie zbliżone są do tych uzyskanych w innych opracowaniach.

W kolejnym kroku przeprowadzono dwa treningi, analogicznie jak w treningach 2 i 3 (z taką samą liczbą obrazów w zbiorze walidacyjnym i odpowiednio taką samą liczbą elementów w zbiorze treningowym), jednakże wykorzystując



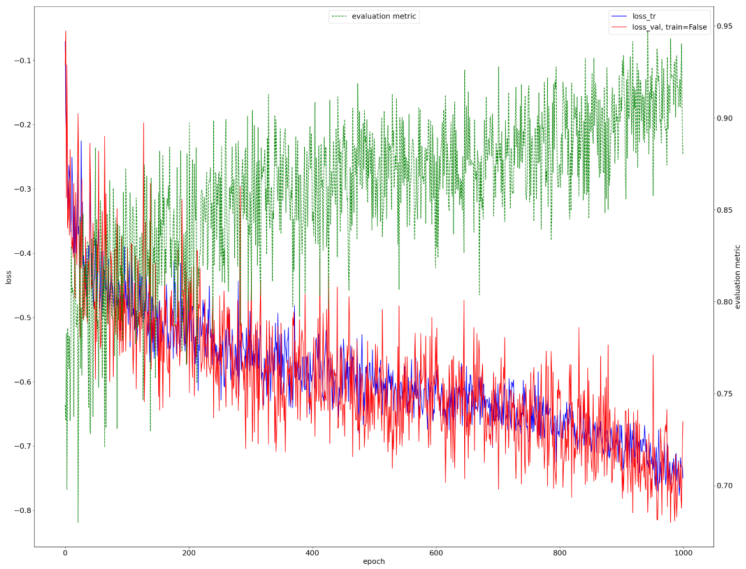
Rysunek 5.5: Przebieg treningu 2



Rysunek 5.6: Przebieg treningu 3

metodę wyboru reprezentantów do zbioru treningowego poprzez klasteryzację. Wyniki uzyskane we wcześniejszych eksperymentach traktować będziemy jako materiał porównawczy względem eksperymentów opartych na zestawach z tak wyselekcjonowanymi obrazami.

Trening 4, niepełny zestaw danych – 20% obrazów wybranych metodą klasteryzacji. Przebieg treningu widać na rys. 5.7. Kolorami czerwonym i niebieskim opisano stratę odpowiednio dla obrazów ze zbiorów treningowego i walidacyjnego. Kolor zielony natomiast oznacza metrykę ewaluacyjną – *averageDicescore*. Trening obejmował 1000 epok.



Rysunek 5.7: Przebieg treningu 4

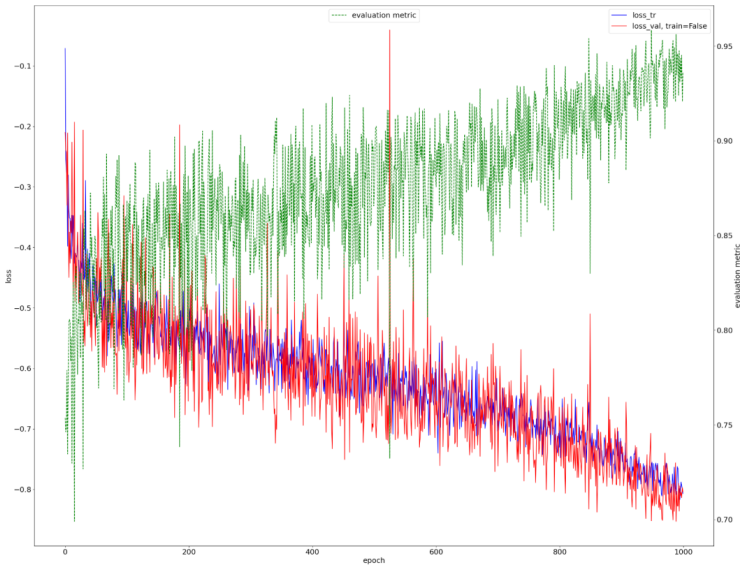
Wyniki predykcji dla zestawu walidacyjnego kształtują się następująco:

- Accuracy: 0,91;
- Dice: 0,82;
- Jaccard: 0,73.

Trening 5, niepełny zestaw danych – 10% obrazów wybranych metodą klasteryzacji. Przebieg treningu widać na rys. 5.8. Kolorami czerwonym i niebieskim opisano stratę odpowiednio dla obrazów ze zbiorów treningowego i walidacyjnego. Kolor zielony natomiast oznacza metrykę ewaluacyjną – *averageDicescore*. Trening obejmował nieco ponad 1000 epok. Zauważalne są znaczne wahania averageDicescore.

Wyniki predykcji na zestawie walidacyjnym:

- Accuracy: 0,91;
- Dice: 0,83;
- Jaccard: 0,75.



Rysunek 5.8: Przebieg treningu 5

Tablica 5.1: Wyniki eksperymentów

Eksperyment	Accuracy	Dice	Jaccard
100% danych	0,90	0,81	0,72
Losowe 20% próba 1	0,89	0,79	0,69
Losowe 20% próba 2	0,89	0,79	0,70
Średnia – losowe 20%	0,89	0,79	0,70
Losowe 10% próba 1	0,89	0,79	0,71
Losowe 10% próba 2	0,88	0,78	0,70
Średnia – losowe 10%	0,89	0,79	0,71
Wybrane 20%	0,91	0,82	0,73
Wybrane 10%	0,91	0,83	0,75

Wyniki wszystkich pięciu pomiarów zebrane są w tablicy 5.1.

Zestawiając osiągnięte średnie wyniki uzyskane na zestawach walidacyjnych, można zauważyć niewielką stratę w jakości segmentacji między wynikami uzyskanymi dla modeli trenowanych pełnym zbiorem i jego losowymi fragmentami. Można zauważyć niewielkie różnice między wynikami w treningu bazującym na podzbiorach 20% i 10%. Dice i Accuracy dla wyników bazujących na podzbiorach 20% są wyższe niż dla treningu bazującego na podzbiorze 10%, mimo że uzyskano niższą miarę Jaccard. Należy zauważyć jednak, że otrzymane wyniki są porównywalne z wynikami uzyskanymi w przytaczanych wcześniej pracach.

Trzeba zauważyć, że wyniki uzyskane dla modeli trenowanych na wybranych obrazach okazują się wyższe niż wyniki otrzymane dla modelu uczonego pełnym zbiorem treningowym. Ważne jest również to, że sam trening tych modeli był krótszy – na takie rezultaty wystarczyło 1000 cykli treningowych. Dla modelu uczonego pełnym zbiorem treningowym osiągnięcie miary Jaccard o wartości 0,72 wymagało 2000 epok.

Przy porównaniu ze sobą tych eksperymentów widać podobne zachowania, tzn. osiągnięty wynik dla 10% objętości danych w obu przypadkach był wyższy niż wynik dla 20% objętości danych. Podział bazujący na obrazach wybranych poprzez metodę klasteryzacji dał w rezultacie wyższe wyniki niż podział losowy. Uzyskane rezultaty wskazują więc na zasadność redukcji zbioru uczącego poprzez selekcję najistotniejszych próbek i wykonania treningu sieci tylko za ich pomocą.

5.5. Podsumowanie

Tradycyjnie jakość modeli sieci głębokich poprawia się poprzez zwiększenie rozmiarów modeli oraz liczby ich parametrów. Uczenie takich modeli wymaga niestety coraz większych zbiorów danych uczących. W przypadku danych medycznych takie podejście jest trudne do zastosowania z powodu braku dużej liczby dobrze opisanych danych.

Uzyskane rezultaty pokazują, że możliwe jest zachowanie jakości sieci przy uczeniu małymi zbiorami danych. Zbiory te muszą być jednak reprezentatywne. Już dla losowej selekcji próbek otrzymano zadowalające rezultaty, ale podejście takie obarczone jest wysokim ryzykiem doboru niewłaściwych próbek. Obecny stan wiedzy pozwala na inteligentną selekcję próbek, jednakże nadal nie jesteśmy w stanie uzyskać wystarczająco wysokiej pewności co do właściwego doboru próbek uczących. Dalsze prace powinny skupić się na opracowaniu algorytmów selekcji takich danych. Dzięki temu możliwe stanie się zredukowanie zarówno kosztu pozyskania danych, kosztu uczenia sieci, jak i złożoności wytrenowanych modeli.

Bibliografia

[1] Aghav-Palwe S., Mishra D. *Feature vector creation using hierarchical data structure for spatial domain image retrieval. Procedia Computer Science*, 167:2458–2464, 2020.

[2] Ali R., et al. *Skin lesion segmentation and classification for ISIC 2018 by combining deep CNN and handcrafted features. arXiv preprint arXiv:1908.05730*, 2019.

[3] Bissoto A., et al. *Deep-learning ensembles for skin-lesion segmentation, analysis, classification: RECOD titans at ISIC challenge 2018. arXiv preprint arXiv:1808.08480*, 2018.

[4] Caron M., et al. *Deep clustering for unsupervised learning of visual features.* In *Proceedings of the European conference on computer vision (ECCV)*, pp. 132–149. 2018.

[5] Codella N., et al. *Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic). arXiv preprint arXiv:1902.03368*, 2019.

[6] Gupta P., Rajmani K., Heinrich M. *nnU-Net: The no-new-UNet for automatic segmentation.* `https://medium.com/miccai-educational-initiative/nnu-net-the-no-new-unet-for-automatic-segmentation-8d655f3f6d2a`. (Dostęp 07/08/2022).

[7] Hardie R.C., et al. *Skin lesion segmentation and classification for ISIC 2018 using traditional classifiers with hand-crafted features. arXiv preprint arXiv:1807.07001*, 2018.

[8] Kekre H., Sarode M.T.K., Thepade S.D. *Image retrieval using color-texture features from DCT on VQ codevectors obtained by Kekre's fast codebook generation. ICGST-International Journal on Graphics, Vision and Image Processing (GVIP)*, 9(5):1–8, 2009.

[9] Kekre H., Thepade S.D. *Improving the performance of image retrieval using partial coefficients of transformed image. International Journal of Information Retrieval, Serials Publications*, 2(1):72–79, 2009.

[10] Li Y., et al. *Self-loop uncertainty: A novel pseudo-label for semi-supervised medical image segmentation.* In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 614–623. Springer, 2020.

[11] MIC-DKFZ. *nnU-Net.* `https://github.com/MIC-DKFZ/nnUNet`. (Dostęp 07/08/2022).

[12] Peng J., Wang Y. *Medical Image Segmentation with Limited Supervision: A Review of Deep Network Models*, 2021. URL `https://arxiv.org/abs/2103.00429`.

[13] Qian C., et al. *A detection and segmentation architecture for skin lesion segmentation on dermoscopy images. arXiv preprint arXiv:1809.03917*, 2018.

[14] Taha A.A., Hanbury A. *Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. BMC medical imaging*, 15(1):1–28, 2015.

[15] Tschandl P., Rosendahl C., Kittler H. *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Scientific data*, 5(1):1–9, 2018.

[16] Willemink M.J., et al. *Preparing medical imaging data for machine learning. Radiology*, 295(1):4–15, 2020.

[17] Yu L., et al. *Uncertainty-aware self-ensembling model for semi-supervised 3D left atrium segmentation.* In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 605–613. Springer, 2019.

Rozdział 6

Detekcja emocji w nagraniach mowy

Michał Affek[1]

[1] Katedra Systemów Decyzyjnych i Robotyki
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
michal.affek@pg.edu.pl

Abstrakt

Dziedzina rozpoznawania emocji z sygnału mowy jest w ostatnim czasie mocno rozwijana. Branża komercyjna mocno inwestuje w najnowsze metody i algorytmy, które są wykorzystywane we flagowych produktach. Coraz częściej publikowane i prezentowane są prace dotyczące podejść bazujących na aktualnie mocno rozwijanych algorytmach uczenia maszynowego. Dofinansowania komercyjne pozwalają środowisku naukowemu tworzyć pełną pasji społeczność zaangażowanych badaczy. Ten rozdział ma za zadanie podsumowanie standardowej procedury tworzenia systemu rozpoznawania emocji z dźwięku. Oprócz przykładów prezentowane są również innowacyjne i zyskujące popularność podejścia do tego skomplikowanego tematu. Dodatkowo przeprowadzono krótki przegląd dostępnych i istniejących zbiorów danych wykorzystywanych w zagadnieniach anotacji emocji z sygnału audio.

Słowa kluczowe: rozpoznawanie emocji, analiza mowy ludzkiej, uczenie maszynowe

6.1. Wprowadzenie

Mowa jest uważana za główną formę komunikacji między ludźmi. Z tego powodu potencjał, jaki drzemie w dziedzinie przetwarzania sygnału mowy, jest zauważany przez podmioty nie tylko naukowe, ale przede wszystkim komercyjne. Problemy takie jak analizowanie sygnałów mowy, próby generowania ich czy synteza z innymi danymi były przez lata gruntownie studiowane i rozwijane. Dziesięciolecia badań umożliwiły opracowanie metod i algorytmów, które pozwalają zamienić sygnał mowy na słowa, wykrywać emocje, generować sztuczne sygnały mowy czy weryfikować tożsamość osoby jedynie na podstawie cech charakterystycznych głosu. Jednak metody te często wykazują niewystarczającą dokładność, co skutkuje ciągłym rozwojem w tej dziedzinie nauki. Powstające produkty komercyjne, opierające się na wymienionych technologiach, przyspieszają postępy naukowców i inżynierów (spore budżety w działach R&D).

Jeśli chodzi o aspekt detekcji emocji, istnieją dwa ogólne modele czy szerzej podejścia opisujące to zagadnienie: *categorical* (emocje są dyskretnymi i zasadniczo różnymi konstrukcjami) i *dimensional* (emocje można scharakteryzować na podstawie wymiarów, grupując je). Pierwszy model próbuje kategoryzować wykryte cechy do jednej z predefiniowanych klas (radość, smutek, złość i inne). Natomiast podejście wymiarowania opiera się na umieszczeniu zmierzonych emocji w przestrzeni dwuwymiarowej na osiach pobudzenia oraz wartościowości [33].

Ostatecznym przeznaczeniem rozpoznawania emocji z mowy jest stworzenie systemów oraz metod odpowiedzialnych za interakcję pomiędzy człowiekiem a maszyną [1]. Taki rodzaj komunikacji uważa się za kolejny etap rozwoju technologii w porównaniu z komunikacją opartą na pozbawionych emocji zwykłych wiadomościach. Posiadając informacje na temat emocjonalnego stanu człowieka, maszyna może dostosować odpowiednio sposób przekazywanych wiadomości, aby sprostać oczekiwaniom rozmówcy, a nawet spróbować kierować przebiegiem rozmowy.

Jako uzupełnienie warto dodać, że istnieją również inne dziedziny próbujące sprostać zapotrzebowaniu na rozpoznawanie emocji. Są to metody oparte na sygnałach audio, które niekoniecznie mają na celu ewaluację sygnału dźwięku mowy. Jedna z mocno rozwijanych dziedzin związana jest z detekcją emocji, które przeważają w utworach muzycznych. Zapotrzebowanie na takie systemy jest duże, między innymi w systemach strumieniujących muzykę, a konkretnie w algorytmach kolejkowania utworów na listach odtwarzania. Takie silniki rekomendacji, które ekstrahują przeważający nastrój utworu, sprawdzają się w momencie, gdy słuchacz nie jest w stanie zidentyfikować, co mu się podobało w konkretnym utworze, ale jest w stanie nazwać emocje, jakie odczuwał podczas odsłuchiwania nagrania [23]. Zarekomendowanie muzyki o podobnych parametrach „emocjonalnych" zwiększa zadowolenie użytkownika oraz wydłuża czas, jaki spędza on na platformie, czyli pośrednio jest korzystny dla budżetowych celów firmy.

6.2. Przetwarzanie mowy

Istnieje wiele podejść do wyodrębniania cech z nieprzetworzonych sygnałów audio. Pierwsza wyróżniająca się różnica między metodami jest związana z dziedziną, którą algorytmy mają przetwarzać. W przeszłości dominującą techniką była analiza statystyczna całego sygnału audio. Było to podyktowane głównie szybkością i niską złożonością obliczeniową. W artykule [22] autorzy stwierdzili, że skupienie się na całym sygnale audio ogranicza model do rozróżniania klas jedynie na podstawie wartości pobudzenia (*arousal*).

Wzrost mocy obliczeniowej pozwolił badaczom podzielić sygnał wejściowy na pomniejsze ramki. Umożliwia to dokładniejszą niż całościowa analizę sygnału mowy. Warto jednak dodać, że starsze metody statystyczne są nadal często łączone z obecnymi rozwiązaniami. Rozmiar i wielkość nakładania się następujących po sobie ramek różnią się w różnych podejściach.

Inny sposób wyodrębniania cech z sygnału mowy jest związany z fonemami (podstawowa jednostka fonologiczna; w rzeczywistej mowie realizowana przez głoski). Dla każdego utworzonego fonemu wykonuje się obliczenie wektora cech. Autorzy artykułu [18] zaproponowali i potwierdzili słuszność wykorzystania fonemów w rozpoznawaniu emocji z dźwięku. Jednak z powodu trudnej implementacji tej metody szersze zastosowanie ma inna metoda, oparta na dźwięcznych (*voiced*) segmentach mowy. Ze względu na specyficzną charakterystykę dźwięcznych segmentów mowy (łatwo identyfikowalne oscylacje) łatwiej jest je wykryć i dokonać podziału wejściowego sygnału audio.

Możliwy jest podział cech charakterystycznych mowy na dwie główne kategorie. Pierwsza z nich to cechy prozodyczne (wysokość tonu, energia i inne). Udowodniły one swoją skuteczność w określaniu poziomu pobudzenia w sygnałach mowy. Drugie podejście opiera się na spektrogramach. Współczynniki melcepstralne (*mel-frequency cepstral coefficients*, MFCC) są obliczane i wykorzystywane w dalszej części przetwarzania. Ta metoda jest szeroko stosowana w problemach rozpoznawania emocji. Na przykład w [11] nieprzetworzony sygnał audio jest reprezentowany jako spektrogram i przeskalowywany do macierzy dwuwymiarowej o 156 wierszach i 64 kolumnach.

Podjęto wiele prób połączenia tych dwóch kategorii, z różnym powodzeniem. Autorzy [14] używali połączenia głębokiego widma rozpraszania (*deep scattering spectrum*, DSS) i niskopoziomowych deskryptorów (*low level descriptors*, LLD). Drugi składnik działa na ramkowym poziomie rozdzielczości, a pierwszy operuje na wyższej rozdzielczości informacyjnej. Dla jasności, różnica pomiędzy metodą DSS a MFCC polega na reprezentacji wysokich częstotliwości. Badacze zaprezentowali wyniki eksperymentu połączenia wyodrębnionych cech, które wskazują, że kombinacja DSS i LLD tworzy lepsze deskryptory niż każda z tych metod z osobna.

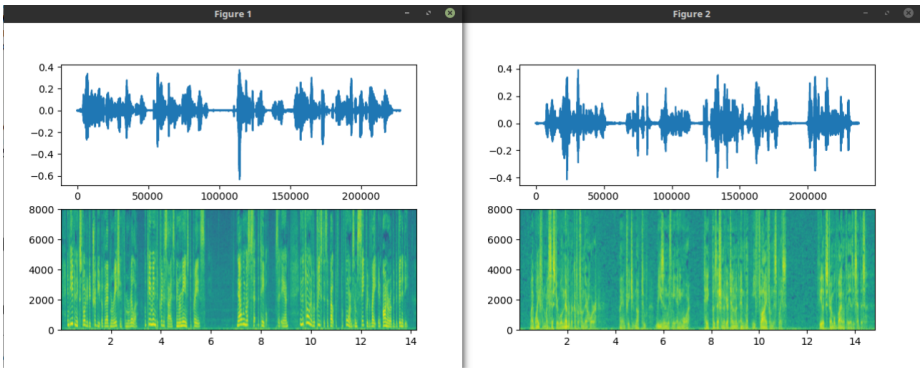
W dziedzinie rozpoznawania emocji z dźwięku można też zaobserwować podejścia oparte na niepłynnościach mowy. Zawierają one dane o pauzach związanych z wypełniaczami (na przykład „hmmm") i ciszą. W artykule [29] opisano potencjalne korelacje pomiędzy niepłynnościami a stanem emocjonalnym.

Naukowcy często łączą bazy danych podczas modelowania metody klasyfikacji emocji, jak również w trakcie testów zaimplementowanych metod. Zrozumiałe jest, że dane zebrane w każdej z kolekcji uzyskano w różnych konfiguracjach, w różnym czasie oraz z różnymi założeniami sprzętowymi. Tworzy to potrzebę wstępnego przetwarzania sygnału mowy, by zbudowany system cechował się zadowalającą dokładnością i generalizacją. Jedną z głównych zmiennych różnicujących dane z różnych źródeł są mikrofony, czyli standardowe narzędzie do przetwarzania i odbierania danych dźwiękowych. To za pomocą ich membran istnieje możliwość wychwytywania różnic ciśnień w powietrzu. Podczas rejestracji sygnału ustala się określoną częstotliwość próbkowania – to dzięki niej możemy zdigitalizować falę dźwiękową i zapisać serię danych na pamięci dysku. W ten sposób można na przykład uzyskać głos konkretnej osoby i przekierować do dalszych metod czy algorytmów. Przed zebraniem jakichkolwiek deskryptorów sygnału często należy przetworzyć surowe nagranie w pożądany sposób. Zwykle oznacza to normalizację wypowiedzi w odniesieniu do różnic środowiskowych [1]. Wyrównanie propagacji fali dźwiękowej można uzyskać, stosując filtr preemfazy, który można opisać następującą funkcją przenoszenia (w postaci transformaty Z) [27]:

$$H(z) = 1 - 0,97z^{-1} \tag{6.1}$$

Dzięki tej modyfikacji sygnał jest gotowy do dalszych przekształceń matematycznych. Podążając za obecnymi trendami w zakresie wstępnego przetwarzania danych (reprezentującymi najlepszą efektywność), powszechnym ruchem jest traktowanie sygnału audio jako obrazu (na rys. 6.1 przedstawiono porównanie zdigitalizowanego sygnału mowy i spektrogramu wyodrębnionego z niego metodą *log-mel spectrogram*). Powód jest prosty – algorytmy wizji komputerowej mają najbardziej rozwinięte sieci neuronowe i sposoby ekstrakcji cech charakterystycznych. Oznacza to, że takie podejście jest bardzo korzystne z wydajnościowego punktu widzenia. Jest ono także preferowane z powodu dostępności oraz otwartości kodu dotyczącego metod oraz wsparcia, jakie oferuje znacząca społeczność zorganizowana wokół takich rozwiązań. Warto również wspomnieć o wstępnie wytrenowanych modelach sieci, które w przypadku trenowania od zera pochłaniałyby znaczne zasoby pieniężno-czasowe.

Ważnym krokiem w przetwarzaniu sygnałów mowy jest redukcja wymiarowości. Wykonuje się ją w celu pozbycia się grupy informacji (zmiennych) wykazujących na przykład nadmierną korelację pomiędzy sobą. Umożliwia to znaczne zmniejszenie zapotrzebowania na moc obliczeniową w procesie ekstrakcji cech charakterystycznych i skrócenie czasu ewaluacji danych dla całego systemu. Redukcja wymiarowości może zostać osiągnięta poprzez zastosowanie wielu różnych metod. Analiza głównych komponentów (*principle component analysis*, PCA) należy do najbardziej popularnych i jest szeroko stosowana. Ta procedura przeprowadza redukcję wymiarowości poprzez utworzenie reprezentacji przestrzeni w mniejszym wymiarze ze zmaksymalizowaną wariancją. Stanowi to gwarancję, że informacje są nadal obecne w zredukowanym wektorze cech. Wiele prac, takich jak [5], potwierdziło zasadność stosowania PCA. Istnieją także inne me-



Rysunek 6.1: Nieprzetworzone sygnały audio i ich spektrogramy dla żeńskiego (po lewej) oraz męskiego (po prawej) lektora

tody, warto wspomnieć o liniowej analizie dyskryminacyjnej (*linear discriminant analysis*, LDA) czy analizie korelacji kanonicznej (*canonical correlation analysis*, CCA). Opracowana została również metoda, która opiera się na sieciach neuronowych [12]. Wydajność i implementację autoenkoderów w zadaniu redukcji wymiarowości można przeanalizować w [14]. Ostatecznie metoda, która powinna być stosowana w realizacji konkretnego zadania, zależy w dużej mierze od cech dostarczonych danych i architektury systemu.

Połączenie cech charakterystycznych z różnych domen jest dość częstym zjawiskiem. Dane audio można wzbogacić znajomością znaczenia wypowiadanego słowa. Taka kombinacja może w ogromnym stopniu wpłynąć na jakość klasyfikacji emocji. Wykorzystanie słowników lingwistycznych [17] jest nadal dość często implementowanym udoskonaleniem systemów wykrywania emocji. Aspekt ten jest jednak dość niepewny ze względu na kontekst kulturowy. Bardziej zdecydowane zastosowanie przedstawili autorzy [11], wskazując potencjał wykorzystania korelacji audiowizualnych w detekcji emocji. W wyniku rozwoju sieci neuronowych na przestrzeni ostatniego dziesięciolecia możliwe jest stworzenie dodatkowego sztucznego (abstrakcyjnego) wskaźnika korelacji pomiędzy audiowizualnymi domenami. Taki parametr pomógłby wypełnić luki w oznakowaniu takich danych i rozszerzyć zastosowanie baz audiowizualnych w rozpoznawaniu emocji z sygnału mowy.

6.3. Wykrywanie emocji

Każdy człowiek ma własne sposoby (czasem podświadome) ekspresji emocji. Uczeni od setek lat starali się określić zestawy ruchów mimicznych, gestykulacji czy intonacji mowy, by z wysoką dokładnością móc stwierdzić, w jakim stanie emocjonalnym znajduje się badana osoba. Zadanie klasyfikacji danych audiowizualnych czy tekstowych w zależności od emocji jest nadal rozwijane, gdyż nie jest łatwo zdefiniować uniwersalny detektor kategoryzujący. Przeszko-

dami są różnice kulturowe, warunki obserwacji oraz przede wszystkim złożoność i kontekst okazywanych emocji. Uczeni nadal prowadzą debaty dotyczące wysokopoziomowej różnicy w definicji emocji. Występują tu dwa główne podejścia: *categorical* oraz *dimensional*.

Pierwsza z teorii stwierdza, że jesteśmy w stanie rozpoznawać emocje jako kompletnie odrębne konstrukcje (podejście dyskretne). Zakłada ona również, że każdy człowiek ma wewnętrzny zestaw zdefiniowanych kategorii emocjonalnych, które są możliwe do zmapowania pomiędzy różnymi kulturami. Najpopularniejszym takim zestawem jest opracowany w 1992 roku przez zespół badawczy Paula Ekmana zbiór zawierający sześć podstawowych emocji: złość, wstręt, strach, szczęście, smutek i zaskoczenie [7]. Naukowcy stwierdzili, że każda z tych kategorii ma własny zestaw ruchów mimicznych, gestykulacji i intonacji w mowie, które u różnych osób w różnym stopniu się powtarzają. Warto dodać, że nie chodzi o predefiniowane absolutne stany emocjonalne, a bardziej o rozmyte kategorie emocji. W późniejszych latach definicje Paula Ekmana zostały podzielone na podkategorie na podstawie przyczyny wywoływania emocji (więcej informacji w artykule [8]).

Drugim z podejść jest teoria opierająca się na wymiarowej klasyfikacji emocji. Bazuje ono na przedstawieniu przestrzeni wymiarowej (dwu- lub trzywymiarowej), w której można zdefiniować obszary odpowiedzialne za poszczególne stany emocjonalne. Jest to podejście bardziej rozmyte i przeczące poprzedniej definicji. Jego popularność wynika między innymi z łatwiejszego zastosowania takiej definicji w praktyce podczas konstrukcji klasyfikatora. Większość podejść wymiarowych zawiera wymiar walencyjny i wymiar pobudzenia (lub intensywności). Ta teoria wykrywania emocji sugeruje, że za wszystkie stany afektywne odpowiada wspólny i wzajemnie powiązany system neurofizjologiczny [26]. Większość modeli w tej kategorii można podzielić na dwuwymiarowe (takie jak *circumplex model* [26], *vector model* [2] oraz *positive activation – negative activation model* [31]) i trójwymiarowe (takie jak *Plutchik's model* [25]).

Aspektem wspólnym dla wielu podstawowych teorii emocji jest to, że powinny istnieć oznaki, które odróżniają różne emocje: powinniśmy być w stanie powiedzieć, jakie emocje odczuwa dana osoba, obserwując aktywność jej mózgu czy fizjologię. Na podstawie obserwacji czynników biologicznych nie ma potrzeby analizowania tego, na czym badana osoba skupia wzrok czy w jakim konkretnym zdarzeniu bierze udział. Można stwierdzić, że taka informacja biologiczna jest informacją absolutną – niestety obecnie wymaga dość sporej liczby zaawansowanych czujników, co utrudnia wykorzystanie takiego podejścia w produktach komercyjnych czy zastosowaniach czasu rzeczywistego [7].

6.4. Zbiory danych

Bazy danych zawierające oznakowane emocjami sygnały dźwiękowe (mowy ludzkiej) są niewątpliwie trudne do zdobycia w formie gotowej do wykorzystania jako zbiór uczący dla konkretnych algorytmów. Dostępne zestawy danych to w większości prywatne zbiory, z nielicznymi wyjątkami. Wynika to z faktu, że groma-

dzenie tego typu danych wymaga długiego czasu i ogromnej ilości pieniędzy (w porównaniu z innymi dziedzinami wymagającymi zbiorów danych). Wysokopoziomowo można podzielić zbiory danych mowy z anotowanymi emocjami na te, w których tworzeniu brali udział profesjonaliści (zawodowi aktorzy), półprofesjonaliści (amatorzy), i nagrania osób wyrażających naturalne (autentyczne) emocje. Taki podział zbiorów spotyka się z wieloma dyskusjami na temat uniwersalności powstałych na ich bazie rozwiązań. Między badaczami istnieje spór dotyczący sytuacji, w której model wyszkolony na emocjach przekazywanych przez aktorów miałby być wykorzystywany (reprezentując podobną dokładność wyników) w naturalnych (autentycznych) przypadkach testowych czy realnych scenariuszach wdrożeniowych [1].

Przed rozpoczęciem poszukiwań odpowiedniego zbioru wymagane jest zdefiniowanie użytkownika powstającej aplikacji. Jednak w większości uniwersalnych systemów nie jest to możliwe. Istnieje niedobór baz danych innych niż zebrane od dorosłych ludzi. Przykładem próby analizy niestandardowych danych jest praca [24], w której badacze stworzyli własny mały zbiór danych zebranych od pięcioosobowej grupy dzieci (dwóch chłopców i trzy dziewczynki). Emocje pomierzone i przeznaczone do rozpoznania obejmowały takie stany, jak złość, zaskoczenie, strach i radość. Wypowiedzi miały częstotliwość próbkowania 8 kHz z 8 bitami. Jeszcze bardziej specjalistycznym przykładem są zbiory takie jak KISMET [3] i BabyEars [28], które zawierają wypowiedzi skierowane do niemowląt [1]. Takie dane mogą być przydatne w modelowaniu metod odpowiedzialnych za interakcje dziecko–maszyna (wychwytują cechy tego specyficznego typu mowy).

W przypadku [19] zespół wykorzystał zbiór danych „Berlin", który jest jedną z dostępnych publicznie i darmowych kolekcji. Można w nim znaleźć 535 wypowiedzi w języku niemieckim o 10 różnych treściach, wypowiedzianych przez 10 różnych profesjonalnych aktorów, wyrażających 7 emocji [6]. Jego wadą jest przede wszystkim to, że definiuje pod względem kulturowo-językowym dość ograniczony (niemiecki) obszar potencjalnych zastosowań. Autorzy nie potwierdzili możliwości przeniesienia użyteczności modelu poza język, w którym został on wytrenowany (nawet jeśli chodzi tylko o emocje). Określa to ogólne wyzwanie całej dziedziny, która musi poradzić sobie z faktem, że każdy zbiór danych będzie miał swoją charakterystykę kulturowo-językową.

Interesujące podejście do kwestii niedostatku etykietowanych emocjami danych mowy zostało przedstawione w [11]. Naukowcy zauważyli, że istnieje duża liczba zbiorów dotyczących graficznej ewaluacji wyrazu twarzy, i postanowili wykorzystać ten fakt. Podejście to opiera się na badaniu korelacji audiowizualnych w niesparowany sposób. Ta metoda zakłada również, że istnieje istotna korelacja między wyrazem twarzy a mową, jeśli chodzi o ekspresję emocji. To pokazuje, że algorytmy są na tyle dojrzałe, by próbować budować systemy, których fundamenty opierają się na danych połączonych z różnych źródeł, a nawet różnych domen. Do pomiaru dokładności rozwiązania przedstawionego w [11] zastosowano multimodalne zbiory danych CREMA-D [4] i RAVDESS [20]. Oba zbiory są publicznie dostępne i zawierają oznaczone dane zarówno dźwiękowe, jak i wizualne.

Należy pamiętać, że dane uzyskane od przedstawiciela każdej z płci mogą

znacząco się różnić. Taka informacja również musi być brana pod uwagę i analizowana podczas wyboru bazy danych wykorzystywanej w planowanej implementacji.

6.5. Klasyfikatory

Ostatnie lata przyniosły rozwój szeroko pojętych klasyfikatorów. Metody można podzielić na podstawie poszczególnych kategorii: ukryte modele Markowa (*hidden markov models*, HMM), mieszane modele Gaussa (*Gaussian mixture models*, GMM), maszyny wektorów nośnych (*support vector machine*, SVM), sztuczne sieci neuronowe (*artificial neural networks*, ANN), k-NN i wiele innych [1]. W obecnych czasach najpowszechniejsze są metody oparte na SVM i sieciach neuronowych. Rozwiązania oparte na HMM (w przeszłości standardowy klasyfikator) powoli znikają z powodu słabszych osiągów i bardziej skomplikowanej implementacji (w porównaniu z działaniem sieci neuronowych).

W artykule [14] badacze stworzyli system, w którym AE wytwarza dane wejściowe dla architektury długoterminowej pamięci krótkoterminowej (*long short-term memory*, LSTM [13]). Ten klasyfikator poprawia efektywność rekurencyjnych sieci neuronowych (*recurrent neural networks*, RNN) w przypadku długich zakresów kontekstowych. LSTM są przydatne w klasyfikowaniu emocji z mowy, gdyż dobrze sobie radzą z rozpoznawaniem ze względu na możliwość przechowywania i dostępu do danych w długich sekwencjach. Dzięki temu sieć może wyłapać kontekst z danych (rozdzielonych w oknach), co w przypadku emocji jest istotne. Dodatkowo ta sieć neuronowa zachowuje rekurencyjny charakter sieci RNN.

Zespół badawczy odpowiedzialny za pracę [11] zdecydował się na użycie generatywnych sieci współzawodniczących (*generative adversarial network*, GAN [10]) z autoenkoderem wariacyjnym (*variational autoencoder*, VAE) jako trzonu w systemie tworzenia i klasyfikowania danych ze względu na emocje. Ta złożona metoda częściowo nadzorowanej, skupionej na klasyfikacji translacji pomiędzy danymi audio i wizualnymi działa zaskakująco dobrze, biorąc pod uwagę braki w oznakowaniu niektórych danych. Wewnętrzny proces translacji pomiędzy domenami realizowany jest przez kategoryzację opartą na gęstości. Rozszerzone dane są tworzone z próbki o dużej gęstości, podczas gdy reszta jest wykorzystywana do ustawienia parametrów kontradyktoryjności.

Wiele metod opracowanych w ostatnich latach wykorzystuje konwolucyjne sieci neuronowe (*convolutional neural networks*, CNN) w swoich aplikacjach. Wynika to z faktu, że warstwy CNN są skuteczne w znajdowaniu korelacji pomiędzy danymi. Dane przekształcone za pomocą CNN działają lepiej w klasyfikatorach opartych na sieciach neuronowych.

Sieci neuronowe, takie jak RNN, CNN i głębokie sieci neuronowe (*deep neural networks*, DNN), są wykorzystywane głównie jako metody przewidujące emocje i jako metody mające na celu reprezentacje danych w domenie emocji (uczenie cech) [9].

SVM jest jednym z bardziej stabilnych klasyfikatorów pod względem po-

pularności w ostatnich dziesięcioleciach. Algorytm oparty jest na mapowaniu oryginalnych cech w przestrzeń o dużej wielkości z zamiarem sklasyfikowania danych za pomocą klasyfikatora liniowego [1]. Dodatkowo w rozpoznawaniu emocji z sygnału mowy nie ma potrzeby doskonałego separowania klas. Ta właściwość problemu klasyfikacji emocji umieszcza nie do końca perfekcyjny SVM wysoko w hierarchii klasyfikatorów. W artykule [34] można zaobserwować i prześledzić eksperymenty i działanie klasyfikatora SVM w różnych konfiguracjach i z różnymi założeniami początkowymi.

6.6. Podejścia całościowe (*end-to-end*, E2E) do detekcji emocji z mowy

Wspomniane wcześniej metody są oparte na standardowych krokach: wstępne przetwarzanie, wyodrębnianie cech, redukcja wymiarowości i klasyfikacja, jednak istnieją również metody oparte całkowicie na sieciach neuronowych i ich reprezentacyjnych zdolnościach [34]. W tej procedurze sieci neuronowe są traktowane jako czarne skrzynki. Ten rodzaj automatycznego rozpoznawania emocji oszczędza czas potrzebny na budowę i analizę metod ekstrakcji cech charakterystycznych mowy. Często dane wejściowe mają postać surowego sygnału audio (na przykład bezpośrednio format .wav), co oznacza, że również wstępne przetwarzanie jest w większości niepotrzebne. Minusem takiego rozwiązania jest czas wymagany do kompletnego i trafnego zaprojektowania architektury sieci neuronowej [6].

6.7. *State-of-the-art*

Wybór odpowiedniej, uniwersalnej metody przetwarzania sygnału dźwiękowego w celu klasyfikacji emocji w nim dominującej zależy w głównej mierze od konkretnego problemu. Różnorodność danych, jeżeli chodzi zarówno o jakość, jak i o domenę tych danych, stwarza rozmyty obszar *state-of-the-art*. Jednocześnie bardzo często naukowcy nie są w stanie przenieść modelu wstępnie wytrenowanego pomiędzy różnie zdefiniowanymi problemami. Obserwuje się znaczny wzrost znaczenia metod wykorzystujących DNN [16].

Powtarzające się w tych metodach podejście to definicja rozwiązania problemu za pomocą systemu opartego na pięciu podstawowych krokach (rys. 6.2). System ten jest stworzony do wykrywania emocji z mowy i potrzebuje klasyfikatora, czyli nadzorowanego konstruktu uczącego się, zaprogramowanego do ewaluacji nowych sygnałów audio. Taki nadzorowany system wprowadza konieczność oznakowania danych z osadzonymi w nich emocjami. Zanim jakiekolwiek przetwarzanie będzie możliwe, należy dokonać wstępnego przetworzenia danych wejściowych. Częstotliwość próbkowania we wszystkich bazach danych używanych w jednej implementacji powinna być spójna. Następnie system może zredukować dane do ich cech charakterystycznych. Mogą to być cechy łączone

Rysunek 6.2: Ogólny schemat systemu rozpoznawania emocji z mowy

z danymi z innych domen, takimi jak informacje dotyczące języka, który dominuje w zbiorze danych, czy dane leksykalne. Można powiedzieć, że wydajność klasyfikatorów zależy głównie od techniki wyodrębniania cech [32].

Najszerzej wykorzystywanymi zbiorami danych do rozpoznawania emocji z mowy są Berlin (EMO-DB), IEMOCAP (Interactive Emotional Dyadic Motion Capture), SAVEE (Surrey Audio-Visual Expressed Emotion), oraz wspomniane już CREMA-D i RAVDESS. Jeśli chodzi o ekstraktory cech charakterystycznych oraz klasyfikatory, to przykładowo w zadaniach mających na celu identyfikację stresu w nagraniach głosowych zalecane są cechy oparte na operatorach energii (*teager energy operator*, TEO) [21]. Natomiast jeśli chodzi o klasyfikowanie ekscytacji z emocji zawartych w mowie powinno się wybierać cechy akustyczne, takie jak wysokość tonu i ton podstawowy. Jednak nadal w większości przypadków najskuteczniejszymi sposobami rozpoznawania mowy są właściwości spektralne, takie jak MFCC [16]. Jako obiecujące jawią się metody łączące ekstraktory cech spektralnych i akustycznych, jednak ich pole zastosowań jest często ograniczone.

Przykładem kompetytywnego podejścia do problemu jest [15]. Autorzy przedstawiają nowy schemat cyklu pracy, który bazuje na umiejętnym połączeniu ekstraktorów cech charakterystycznych. Wyodrębniają MFCC, chromagramy, spektrogramy w skali mel, reprezentację Tonnetz oraz kontrast spektralny z nagrań audio i wykorzystują je jako wejścia do jednowymiarowej konwolucyjnej sieci neuronowej. Całość badań została przeprowadzona i przetestowana na zbiorach danych RAVDESS, Berlin (EMO-DB) i na IEMOCAP. Uzyskano rozpoznawanie emocji w nagraniu ze skutecznościami: 71,61% na zbiorze 8 klas z RAVDESS, 86,1% na 535 przykładach z 7 klas EMO-DB, 95,71% na 520 przykładach z 7 klas EMO-DB, oraz 64,3% na zbiorze 4 klas z IEMOCAP.

6.8. Podsumowanie

Głównym problemem w wykrywaniu emocji z mowy jest trudność w określeniu znaczenia i konkretnej definicji poszczególnych emocji. Emocje w rzeczywistych, nieodgrywanych nagraniach są zwykle wymieszane ze sobą i mało zrozumiałe (nawet dla człowieka). Zbiory baz danych wyraźnie odzwierciedlają brak porozumienia wśród twórców co do tych faktów. Dodatkowym wyzwaniem jest radzenie sobie z regularnie współwystępującym szumem addytywnym w danych

audio, obejmującym zniekształcenia (zepsuty lub niskojakościowy odbiornik – mikrofon) i dodatkowe informacje (dźwięki z tła).

Problemy pojawiają się również na etapie etykietowania wypowiedzi. Po nagraniu wypowiedzi dane mowy są oznaczane przez specjalnie zatrudnionych ludzi (specjaliści) lub poprzez metody *crowdsourcingu*. Rzeczywiste emocje mówiącego mogą różnić się od tych postrzeganych przez osobę zajmującą się etykietowaniem. Nawet w przypadku ludzkich adnotatorów wskaźniki rozpoznawania pozostają niższe niż 90% [30].

Kluczowym etapem systemu jest klasyfikacja. To od niej zależy zdolność powstałego rozwiązania do interpretacji wyników generowanych przez metody wyodrębniające cechy charakterystyczne nagrań. Klasyfikatory oparte na głębokim uczeniu są znacznie wolniejsze z powodu *max-poolingu*, co skutkuje wydłużeniem czasu treningu modelu. Z kolei klasyfikatory tradycyjne, takie jak kNN, drzewo decyzyjne czy SVM, wymagają więcej czasu, aby przetworzyć większe zbiory danych – co jest konieczne, jeśli autorzy chcą zwiększać dokładność powstałych systemów.

Oprócz zwiększania dokładności sporym wyzwaniem dla dziedziny rozpoznawania emocji z mowy są systemy wielojęzyczne oraz systemy potrafiące wykrywać emocje z nagrań zawierających jednoczesną mowę grupy osób [30].

Bibliografia

[1] Ayadi M.E., Kamel M.S., Karray F. *Survey on speech emotion recognition: Features, classification schemes, and databases. Pattern Recognition*, 44(3):572–587, 2011.

[2] Bradley MM Greenwald MK P.M.L.P. *Remembering pictures: pleasure and arousal in memory. J Exp Psychol Learn Mem Cogn*, 18(2):379–90, 1992.

[3] Breazeal C., Aryananda L. *Recognition of Affective Communicative Intent in Robot-Directed Speech. Autonomous Robots*, 12(1):83–104, 2002.

[4] Cao H., et al. *CREMA-D: Crowd-Sourced Emotional Multimodal Actors Dataset. IEEE Transactions on Affective Computing*, 5(4):377–390, 2014.

[5] Chuang Z.J., Wu C.H. *Emotion recognition using acoustic features and textual content.* pp. 53 – 56 Vol.1. 2004. ISBN 0-7803-8603-5.

[6] Deschamps-Berger T., Lamel L., Devillers L. *End-to-End Speech Emotion Recognition: Challenges of Real-Life Emergency Call Centers Data Recordings. CoRR*, abs/2110.14957, 2021.

[7] Ekman P. *An argument for basic emotions. Cognition and Emotion*, 6(3-4):169–200, 1992.

[8] Gendron M. *Reconstructing the Past: A Century of Ideas About Emotion in Psychology. Emotion review: journal of the International Society for Research on Emotion*, 1(4):316–339, 2009.

[9] Ghosh S., et al. *Representation Learning for Speech Emotion Recognition.* pp. 3603–3607. 2016.

[10] Goodfellow I., et al. *Generative Adversarial Nets.* In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger, eds., *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.

[11] He G., et al. *Image2Audio: Facilitating Semi-supervised Audio Emotion Recognition with Facial Expression Image.* In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3978–3983. 2020.

[12] Hinton G.E., Salakhutdinov R.R. *Reducing the Dimensionality of Data with Neural Networks. Science*, 313(5786):504–507, 2006.

[13] Hochreiter S., Schmidhuber J. *Long Short-term Memory. Neural computation*, 9:1735–80, 1997.

[14] Huang K.Y., et al. *Speech emotion recognition using autoencoder bottleneck features and LSTM.* In *2016 International Conference on Orange Technologies (ICOT)*, pp. 1–4. 2016.

[15] Issa D., Fatih Demirci M., Yazici A. *Speech emotion recognition with deep convolutional neural networks. Biomedical Signal Processing and Control*, 59:101894, 2020.

[16] Jahangir R., et al. *Deep learning approaches for speech emotion recognition: state of the art and research challenges. Multimedia Tools and Applications*, 80(16):23745–23812, 2021.

[17] Lee C., Narayanan S., Pieraccini R. *Combining acoustic and language information for emotion recognition.* 2002.

[18] Leinonen L., et al. *Expression of emotional–motivational connotations with a one-word utterance. The Journal of the Acoustical Society of America*, 102(3):1853–1863, 1997.

[19] Lim W., Jang D., Lee T. *Speech emotion recognition using convolutional and Recurrent Neural Networks.* In *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pp. 1–4. 2016.

[20] Livingstone S.R., Russo F.A. *The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS).* 2018.

[21] Mahalle V.S., et al. *Teager Energy Operator: A Signal Processing Approach for Detection and Classification of Power Quality Events.* In *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1109–1114. 2018.

[22] Nwe T.L., Foo S.W., Silva L.C.D. *Speech emotion recognition using hidden Markov models. Speech Communication*, 41(4):603–623, 2003. ISSN 0167-6393.

[23] Orjesek R., et al. *DNN Based Music Emotion Recognition from Raw Audio Signal.* In *2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pp. 1–4. 2019.

[24] Palo H.K., Mohanty M.N., Chandra M. *Statistical feature based child emotion analysis.* In *Michael Faraday IET International Summit 2015*, pp. 307–312. 2015.

[25] Plutchik R. *The Nature of Emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. American Scientist*, 89(4):344–350, 2001.

[26] Posner J., Russel J.A., Peterson B.S. *The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology. Development and Psychopathology*, 17(3):715–734, 2005.

[27] Rabiner L., Schafer R. *Digital Processing of Speech Signals.* Englewood Cliffs: Prentice Hall, 1978.

[28] Slaney M., McRoberts G. *BabyEars: A recognition system for affective vocalizations. Speech Communication*, 39:367–384, 2003.

[29] Vidrascu L., Devillers L. *Detection of real-life emotions in call centers.* pp. 1841–1844. 2005.

[30] Wani T.M., et al. *A Comprehensive Review of Speech Emotion Recognition Systems. IEEE Access*, 9:47795–47814, 2021.

[31] Watson D T.A. *Toward a consensual structure of mood. Psychol Bull*, 98(2):219–35, 1985.

[32] Yala N., Fergani B., Fleury A. *Towards improving feature extraction and classification for activity recognition on streaming data. Journal of Ambient Intelligence and Humanized Computing*, 8(2):177–189, 2017.

[33] Zeng Z., et al. *A Survey of Affect Recognition Methods: Audio, Visual, and Spontaneous Expressions. IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):39–58, 2009.

[34] Zhao H., Ye N., Wang R. *A Survey on Automatic Emotion Recognition Using Audio Big Data and Deep Learning Architectures.* In *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*, pp. 139–142. 2018.

Rozdział 7

Nieświadome sieci neuronowe

Stanisław Barański[1]

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

Abstrakt

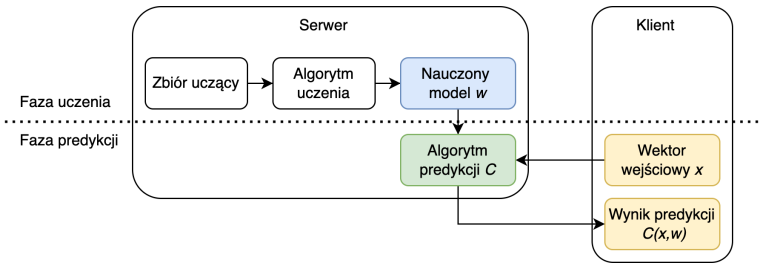
Coraz większą popularność zyskuje usługa predykcji za pomocą sieci neuronowych. Model ten zakłada istnienie serwera, który za pomocą wyuczonej sieci neuronowej dokonuje predykcji na danych otrzymanych od klienta. Model ten jest wygodny, ponieważ obie strony mogą skupić się na rozwoju w swojej specjalizacji. Wystawia on jednak na ryzyko utraty prywatności zarówno klienta, wysyłającego wrażliwe dane wejściowe, jak i serwer, udostępniający wyuczony model sieci neuronowej. Niniejszy rozdział opisuje proces dokonywania nieświadomej predykcji za pomocą sieci neuronowych. Nieświadomość pozwala na dokonanie predykcji za pomocą sieci neuronowych przy zachowaniu prywatności danych wejściowych klienta oraz modelu serwera. Umożliwia świadczenie usług, które dotychczas blokowane były przez regulacje prawne lub brak wiarygodności przetwarzających je systemów.

Słowa kluczowe: sieci neuronowe, nieświadome, prywatność, *multi-party computation*

7.1. Wstęp

Uczenie maszynowe (*machine learning*, ML) znajduje zastosowanie w wielu aplikacjach; najważniejsze z perspektywy niniejszego rozdziału to rozpoznawanie obrazów, diagnozy medyczne, ocena ryzyka udzielenia kredytu, i inne przetwarzające poufne dane osobiste.

Uczenie nadzorowane (rys. 7.1) składa się z dwóch faz: 1) faza uczenia, w której algorytm uczy model w zbiorem uczącym; 2) faza predykcji, w której algorytm predykcji C na podstawie danych wejściowych x (zwanych wektorem parametrów) oraz wcześniej wyuczonego modelu w dokonuje predykcji $C(x, w)$ wartości ciągłej (regresja) lub przypisania kategorii (klasyfikacja).



Rysunek 7.1: Diagram przedstawiający realizację usługi *Prediction as a service*

Szczególnie popularną metodą uczenia maszynowego, ze względu na ich wszechstronność, są sieci neuronowe (*neural networks*, NN).

Model biznesowy świadczenia predykcji za pomocą sieci neuronowych (*prediction as a service*) pozwala klientowi na oddelegowanie predykcji C danych wejściowych x do usługodawcy (dalej zwanego serwerem), który oferuje wytrenowany model sieci neuronowych w.

Model ten jest atrakcyjnym rozwiązaniem dla wielu biznesów, ponieważ usługodawcy mogą skupić się na optymalizacji modelu, a klienci zyskują przez ograniczenie kosztów potrzebnych na zatrudnienie specjalistów, zebranie danych treningowych, wytrenowanie modeli i utrzymanie infrastruktury.

Model ten jednak implikuje niewidoczne na pierwszy rzut oka trudności. W sytuacji, gdy dane wejściowe zawierają wrażliwe informacje, pojawiają się problemy związane z prywatnością i bezpieczeństwem przetwarzania danych (np. GDPR i HIPAA). Przykładem takich sytuacji może być laboratorium medyczne, które chciałoby dokonać rozpoznania choroby na podstawie zdjęć rentgenowskich lub oszacowania prawdopodobieństwa wystąpienia choroby na podstawie pobranych próbek. Regulacje dotyczące przetwarzania danych mogą uniemożliwić zarówno udostępnianie danych pacjenta do serwera, jak i udostępnianie przez serwer nauczonego modelu do laboratorium.

Innym przykładem może być klient, który chciałby otrzymać oszacowanie swojej zdolności kredytowej lub składek ubezpieczeniowych od instytucji finansowej. Zarówno klient nie chce przekazywać szczegółowych danych personalnych, jak i serwer nie chce udzielać szczegółów algorytmu estymacji.

W takich przypadkach oddelegowywanie predykcji staje się utrudnione, a nawet niemożliwe. Najprostszym rozwiązaniem tego problemu jest pobranie modelu w i dokonanie predykcji C lokalnie w domenie klienta. Rozwiązanie to jednak w wielu przypadkach jest niemożliwe ze względu na: 1) prawa własnościowe takich modeli; 2) utrudnioną aktualizację modelu; 3) możliwość ekstrakcji informacji o zbiorze treningowym, który również może być poufny, np. w przypadku historii danych medycznych pacjentów.

Nasuwa się więc pytanie, **czy możliwe jest dokonanie predykcji C tak, aby serwer nie dowiedział się niczego o danych wejściowych x, a klient nie dowiedział się niczego o modelu w w poza wynikiem predykcji $C(x, w)$?**

Jak się okazuje, jest to możliwe. Niniejszy rozdział prezentuje aktualny stan techniki realizacji tego typu rozwiązań. W dalszej części będziemy nazywać te techniki nieświadomymi (*oblivious*). Istnieją metody zapewniania nieświadomości wielu algorytmów uczenia maszynowego [5, 35], jak np. maszyny wektorów nośnych, naiwny klasyfikator bayesowski, Perceptron, AdaBoost. Jednak w tym rozdziale skupimy się na predykcji za pomocą nieświadomych sieci neuronowych. Pominiemy również fazę nieświadomego uczenia – zakładamy, że model został nauczony w dowolny sposób, a nieświadomość ogranicza się jedynie do fazy predykcji. W celu zrozumienia procesu unieświadamiania sieci neuronowych musimy najpierw ustalić, czym tak naprawdę jest sieć neuronowa. W sekcji 7.2 odpowiemy na to pytanie przez wizualizację oraz sformułowanie uniwersalnego modelu sieci neuronowych.

W sekcji 7.3 przejdziemy do tematu nieświadomych obliczeń. Wymagać to będzie zaczerpnięcia wiedzy z czterech dziedziny nauki: teorii obliczeń, teorii układów cyfrowych, kryptografii, oraz uczenia maszynowego.

Aby przekazać tak szeroką wiedzę w jednym rozdziale, skupimy się na jednym – najpopularniejszym i najprostszym – rozwiązaniu do budowania nieświadomych sieci neuronowych. Następnie, korzystając z nabytej intuicji, rozpatrzymy alternatywne podejścia (sekcja 7.5).

7.2. Sieci neuronowe

Sieć nieuronowa jest potokiem warstw. Każda warstwa otrzymuje sygnał wejściowy, przetwarza go i generuje sygnał wyjściowy, który staje się sygnałem wejściowym dla kolejnej warstwy. Pierwsza warstwa otrzymuje dane wejściowe x, a sygnał wyjściowy ostatniej warstwy jest wynikiem predykcji $C(x, w)$.

Typowa warstwa sieci neuronowej dokonuje transformacji liniowej (mnożenie i dodawanie macierzy), a następnie transformacji nieliniowej (funkcja aktywacji, a czasami również *pooling* w celu zredukowania rozdzielczości danych).

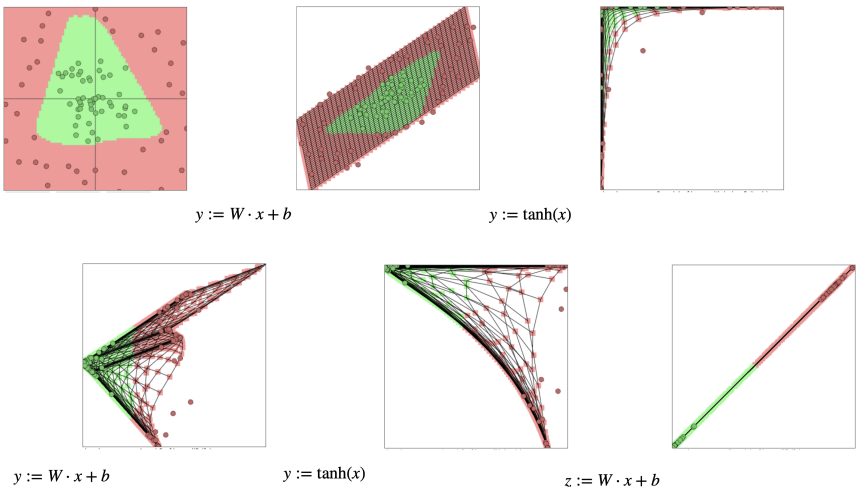
Predykcje za pomocą sieci neuronowych można przedstawić za pomocą potoku transformacji:

$$x \to f_1 \to a_1 \to ... \to f_n \to a_n \to y$$

gdzie:

- x jest wektorem wejściowym;
- f_i jest transformacją liniową warstwy i;
- a_i jest transformacją nieliniową warstwy i;
- n jest łączną liczbą warstw sieci neuronowej.

Celem transformacji jest zniekształcenie przestrzeni danych wejściowych w taki sposób, aby stały się liniowo separowalne, tj. aby dało się stworzyć linię (jeśli dane wejściowe opisane są w dwóch wymiarach), płaszczyznę (w trzech wymiarach), hiperpłaszczyznę (w $n + 1$ wymiarach), która podzieli zbiór wejściowy na dwa podzbiory. Rys. 7.2 przedstawia transformacje każdej z warstw przykładowej sieci neuronowej. Liniowa separowalność danych (na obszary zielony i czerwony) jest możliwa przez przeprowadzenie sekwencji transformacji liniowych i nieliniowych. Transformacje liniowe pozwalają na obracanie, przechylanie oraz rozciąganie przestrzeni. Transformacje nieliniowe natomiast pozwalają na deformacje przestrzeni.



Rysunek 7.2: Wizualizacja transformacji liniowych i nieliniowych w celu osiągnięcia liniowej separowalności

7.2.1. Transformacje liniowe

Mnożenie i dodawanie macierzy są najpopularniejszymi transformacjami liniowymi stosowanymi w sieciach neuronowych:

$$y := W \cdot x + b \tag{7.1}$$

gdzie:

- x jest wektorem wejściowym;
- W jest macierzą wag;
- b jest wektorem wyrazów wolnych;
- y jest wektorem wyjściowym.

Konwolucja jest transformacją liniową pozwalającą na obliczenie produktu skalarnego pomiędzy tensorem wag (filtra, zwanego kernelem) a elementem macierzy wraz z jego sąsiadującymi elementami. Proces ten jest powtarzany z przesuwaniem filtra po macierzy wejściowej. W praktyce konwolucje zamienia się na postać mnożenia i dodawania macierzy, osiągając wzrost wydajności [12], podobnie jak w równaniu (7.1), z tą różnicą, że dana wejściowa i wyraz wolny są macierzami: $Y := W \cdot X + B$.

7.2.2. Transformacje nieliniowe

Sieci neuronowe korzystają z nieliniowych transformacji w celu zamodelowania nieliniowej zależności pomiędzy przestrzeniami wejściową a wyjściową.
Funkcje aktywacji. Rozróżnić można trzy kategorie funkcji aktywacji.

- *Funkcja aktywacji przedziałami liniowymi.* Jest to funkcja, która może zostać zareprezentowana jako zbiór n funkcji liniowych $f_i(x) = a_i x + b_i$, gdzie x jest ograniczone przez zakresy dolny i górny dla danego przedziału. Funkcjami tego typu są:
  - funkcja tożsamościowa: $f(x) = x$;
  - *rectified linear units (ReLU)*: $f(x) = max(0, x)$;
  - *leaky ReLU*: $f(x) = max(0, x) + amin(0, x)$;
  - *maxout*: $f(x) = max(y_1, ..., y_n)$.
- *Gładka (regularna) funkcja aktywacji.* Jest to funkcja różniczkowalna określona pewną liczbę razy w swojej dziedzinie. Najpopularniejsze gładkie funkcje aktywacji to:
  - *sigmoid (logistic)*: $f(x) = \frac{1}{1+e^{-x}}$;
  - *hyperbolic tangent (tanh)*: $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$;
  - *softplus*: $f(x) = \log(e^x + 1)$.

  Sigmoid oraz *tanh* są funkcjami zwanymi sigmoidalnymi, pomiędzy którymi zachodzi zależność:

$$tanh(x) = 2 \cdot sigmoid(2x) - 1 \tag{7.2}$$

- *Softmax.* Funkcja ta używana jest najczęściej jako ostatnia warstwa NN w celu określenia rozkładu prawdopodobieństwa klasyfikacji. Funkcja zdefiniowana jest jako

$$softmax_i(x) = \frac{e^{x_i}}{\sum_k e^{x_k}} \tag{7.3}$$

Pooling jest operacją redukującą rozdzielczość macierzy. Nazywana również operacją złożenia, polega na organizacji danych wejściowych w podgrupy i agregacji każdej podgrupy przy zmniejszaniu wymiaru danych wejściowych. Najczęściej elementy agreguje się za pomocą funkcji uśredniania (*mean pooling*) lub maksymalizacji (*max pooling*).

7.2.3. Wnioski

Faza predykcji za pomocą sieci neuronowych sprowadza się do sekwencji transformacji liniowych oraz tranformacji nieliniowych. Transformacje liniowe sprowadzają się do mnożenia i dodawania macierzy. Transformacje nieliniowe sprowadzają się do wykonania funkcji aktywacji lub operacji *poolingu*.

Dlatego, aby unieświadomić cały proces predykcji, wystarczy unieświadomić operacje mnożenia i dodawania macierzy oraz wykonywania funkcji aktywacji i operacji *poolingu*.

7.3. Nieświadome obliczenia

Secure multi-party computation (MPC) jest techniką wykonywania dowolnego algorytmu $F(x_1, ..., x_n)$ przez n niezależnych uczestników w taki sposób, że żaden z nich nie dowiaduje się niczego o danych wejściowych pozostałych uczestników.

Dla naszych potrzeb skupimy się na przypadku, w którym tylko dwie strony (klient i serwer) biorą udział w wykonaniu algorytmu F. Ten szczególny przypadek nazywany jest *secure two-party computation* (2PC).

Najprostszym sposobem na wykonanie algorytmu F w taki sposób, aby żadna ze stron nie dowiedziała się o danych wejściowych drugiej strony, jest wprowadzenie zaufanej trzeciej strony (*trusted third party*, TTP), która otrzymuje dane wejściowe od obu uczestników, wykonuje funkcjonalność F oraz zwraca wynik obliczeń.

Wprowadzenie zaufanej trzeciej strony jest problematyczne z wielu powodów, najważniejszym z nich jest wprowadzenie – często nieakceptowalnego – założenia o faktycznej uczciwości TTP. Dlatego jest to założenie, którego za wszelką cenę staramy się uniknąć, projektując zaufane systemy informatyczne.

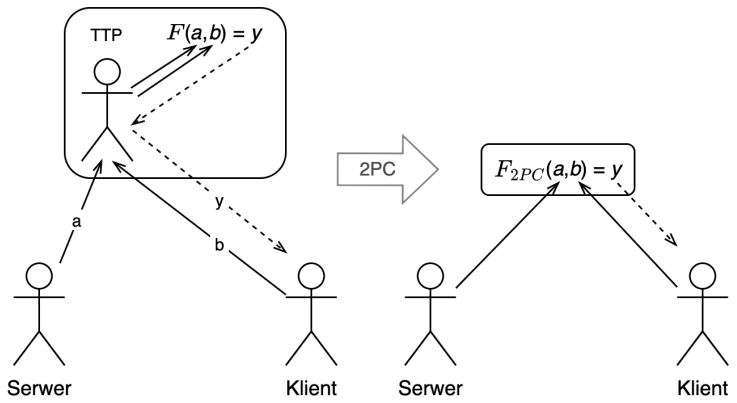
Technika 2PC rozwiązuje ten problem, pozwalając na osiągnięcie tej samej funkcjonalności bez pomocy TTP (rys. 7.3).

7.3.1. *Yao's garbled circuit* (GC)

Przez wiele lat 2PC stało w cieniu teoretycznych rozważań kryptologów. Pierwszym praktycznym rozwiązaniem jest *Yao's garbled circuit* (GC) [36].

Intuicja protokołu GC

W celu pokazania działania protokołu GC posłużymy się przykładem problemu dwóch studentów chcących dowiedzieć się, kto z nich jest bogatszy, bez ujaw-



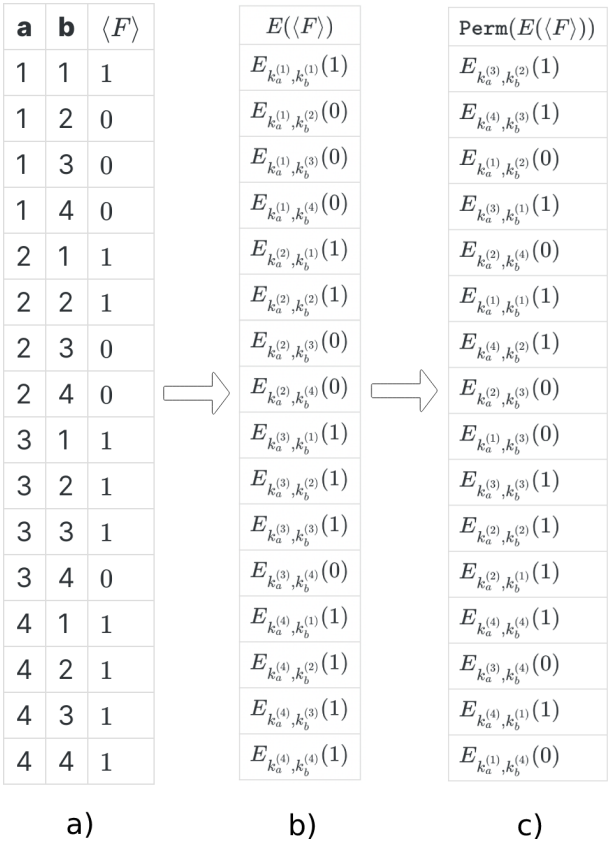
Rysunek 7.3: Transformacja bezpiecznej funkcjonalności F w modelu z zaufaną trzecią stroną (TTP) do bezpiecznej funkcjonalności bez zaufanej trzeciej strony F_{2PC} (*secure two-party computation*)

nienia swojego majątku. Problem ten można zareprezentować przez funkcję $F(a, b) = a \geq b$, gdzie a to majątek pierwszego studenta (dla spójności nazywanego serwerem), a b to majątek drugiego studenta (klienta). Dla uproszczenia przykładu załóżmy, że liczby a i b reprezentować będą liczbę cyfr majątku każdej ze stron oraz że maksymalna liczba cyfr to 4. Przykładowo, majątek o wartości 7 zł reprezentowany jest przez liczbę 1, majątek o wartości o wartości 35 zł przez liczbę 2, o wartości 786 zł przez liczbę 3, o wartości 9999 zł przez liczbę 4.

Dziedziną funkcji $F\colon X^2 \to Y$ jest $X^2 = \{1, 2, 3, 4\}^2$, a przeciwdziedziną $Y = \{1, 0\}$, gdzie 1 reprezentuje prawdę (pierwszy argument jest większy lub równy drugiemu), natomiast 0 fałsz (pierwszy argument jest mniejszy od drugiego). W pierwszym kroku jedna ze stron (załóżmy, że będzie to serwer) tworzy tablice (*lookup table*) wszystkich możliwych wywołań funkcji F, tj. $\langle F \rangle = \langle F(1,1) = 1, F(1,2) = 0, ..., F(4,4) = 1 \rangle$ (rys. 7.4a). Wywołanie funkcji reprezentowanej w formie tablicy sprowadza się do wyciągnięcia odpowiedniego wiersza z kolumny $\langle F \rangle$ dla wartości a i b.

W celu stworzenia nieświadomej wersji funkcji F reprezentowanej w formie tablicy $\langle F \rangle$ serwer szyfruje tablicę, używając losowo wygenerowanych kluczy dla każdej wartości a oraz b. Konkretniej, każdej wartości $a \in X$ oraz $b \in X$ serwer przypisuje silny pseudolosowy klucz $k_a \in \{0, 1\}^\kappa$, oraz $k_b \in \{0, 1\}^\kappa$, gdzie κ jest parametrem bezpieczeństwa i oznacza długość ciągu bitów klucza szyfrującego.

Powstaje w ten sposób łącznie $|X|+|X| = 8$ kluczy ($k_a^{(1)}$, $k_a^{(2)}$, $k_a^{(3)}$, $k_a^{(4)}$, $k_b^{(1)}$, $k_b^{(2)}$, $k_b^{(3)}$, $k_b^{(4)}$) służących do zaszyfrowania wyniku funkcji F dla każdej kombinacji argumentów a i b. Niech $E_{k_a,k_b}(\cdot)$ będzie symetryczną funkcją szyfrującą, która jest parametryzowana dwoma kluczami szyfrującymi: odpowiednim dla wartości a kluczem serwera k_a oraz odpowiednim dla wartości b kluczem klienta k_b. Szyfrowanie dwoma kluczami można implementować na wiele sposobów,



Rysunek 7.4: Transformacje tablicy protokołu *garbled circuit* (GC)

jednym z nich jest podwójne szyfrowanie, pierwszy raz kluczem serwera, drugi raz kluczem klienta; formalnie $E_{k_a,k_b}(\cdot) = E_{k_b}(E_{k_a}(\cdot))$.

Zaszyfrowaną tablicę $\langle F \rangle$ oznaczać będziemy $E(\langle F \rangle)$ (rys. 7.4b).

Tak przygotowana przez serwer tablica jest prawie gotowa do wysłania klientowi. Prawie, ponieważ otrzymując tablicę w takiej postaci, klient jest w stanie wydedukować a i b na podstawie wiersza tablicy. Dlatego ostatnią modyfikacją jest dokonanie losowej permutacji, tak aby klient nie był w stanie wydedukować wartości a. Powstaje w ten sposób tablica $\mathrm{Perm}(E(\langle F \rangle))$ widoczna na rys. 7.4c.

W tym momencie tablica jest gotowa. Serwer wysyła tablicę klientowi wraz z kluczem $k_a^{(i)}$, gdzie i jest wartością wybraną przez serwer; załóżmy, że wartość ta wynosi $i = 4$, dlatego serwer wysyła klientowi klucz $k_a^{(4)}$.

Zauważmy, że sam klucz $k_a^{(4)}$ jest losowym ciągiem bitów nienoszącym żadnej informacji. Nie pozwala na odszyfrowanie żadnego wiersza w tablicy. Nie pozwala również na określenie danej wejściowej serwera.

Nieświadomość obliczenia osiągana jest dopiero w następnym kroku, a przebiega on następująco. Serwer przeprowadza z klientem procedurę nieświadomego transferu (*oblivious transfer*, OT) [28], która pozwala klientowi wybrać jeden klucz ze zbioru oferowanych kluczy $\{k_b^{(1)}, k_b^{(2)}, k_b^{(3)}, k_b^{(4)}\}$ w taki sposób, że serwer nie dowiaduje się, który element został wybrany, a klient nie dowiaduje się niczego o pozostałych kluczach poza tym, który wybrał. Dla naszego przykładu załóżmy, że dla klienta liczba cyfr jego majątku wynosi $j = 3$, dlatego pobiera on klucz $k_b^{(3)}$.

W tym momencie klient posiada wszystkie elementy potrzebne do odtworzenia wartości funkcji $F(a, b)$. W połączeniu z otrzymanym kluczem $k_a^{(4)}$ klient może odszyfrować tylko i wyłącznie wiersz zaszyfrowany przy użyciu obu kluczy $k_a^{(4)}$ oraz $k_b^{(3)}$, czyli wiersz drugi o wartości $E_{k_a^{(4)},k_b^{(3)}}(1)$. Klient odszyfrowuje wiersz $E_{k_a^{(4)},k_b^{(3)}}(E_{k_a^{(4)},k_b^{(3)}}(1)) = 1$, otrzymując wartość pozytywną, mówiącą o większym majątku serwera $F(4, 3) = 4 \geq 3$. W zależności od uzgodnienia klient wysyła (lub nie) wynik do serwera. W celu zapewnienia wiarygodności wyniku zwróconego przez klienta wynik powinien zostać opatrzony podpisem cyfrowym wykonanym przez serwer.

Właściwy protokół GC

Uważny czytelnik może dostrzec dwa problemy wyżej opisanego protokołu. Po pierwsze, klient nie ma informacji, który wiersz odpowiada której wartości, dlatego nie wie, który wiersz powinien odszyfrować, używając otrzymanych kluczy. Naiwną metodą jest dodanie do każdej wartości wynikowej ciągu zer, które będą sygnalizowały, że to, co odszyfrował, faktycznie jest wartością wynikową, a nie losowym ciągiem bitów. Prawdopodobieństwo odszyfrowania błędnego wiersza zakończonego ciągiem zer jest bardzo niskie ($p = \frac{1}{2^\sigma}$, gdzie σ to liczba zer). Rozwiązanie to nie jest idealne, ponieważ klient musi dokonać średnio $\frac{\langle F \rangle}{2}$ odszyfrowań, zanim dotrze do poprawnego wiersza w tablicy. W praktyce stosuje się technikę *point-and-permute* [3], która pozwala na dodanie wskaźnika lokalizującego wiersz do odszyfrowania. Metoda została opisana w dalszej części tej sekcji.

Drugim problemem jest rozmiar tablicy, który rośnie kwadratowo wraz z rozmiarem argumentów $|\langle F \rangle| = |X| \times |X| = |X|^2$. Gdybyśmy chcieli zmienić dziedzinę funkcji F na liczby typu uint32, rozmiar tablicy wyniósłby $|\text{uint32}| \times |\text{uint32}| = 2^{32} \times 2^{32} = 2^{64}$, następnie każdą liczbę należałoby zakodować na 32 bitach, w wyniku czego rozmiar tablicy wyniósłby $2^{64} \times 32$ bitów, czyli zdecydowanie za dużo, jak na proste porównanie dwóch liczb.

Niemniej dla małych wartości, jak np. bramki logiczne, dla których dziedzina wynosi $|\{0, 1\}| \times |\{0, 1\}| = 4$, rozwiązanie to jest praktyczne. Dlatego **protokół GC interpretuje algorytm F w postaci układu logicznego** C, składającego się ze zbioru bramek logicznych $G = g_1, g_2, ..., g_m$, połączonych przewodami $W = w_1, w_2, ..., w_n$. Każda bramka g_i jest funkcją przyjmującą wartości dwóch przewodów wejściowych i zwracającą wartość przewodu wyjściowego. Przykładowo niech g_{OR} będzie bramką OR, której przewodami wejściowymi są w_1

oraz w_2, a przewodem wyjściowym jest w_3. Bramka g_{OR} reprezentuje funkcję $w_3 \leftarrow g_{\text{OR}}(w_1, w_2)$, taką że:

$$0 \leftarrow g_{\text{OR}}(0,0),\ 1 \leftarrow g_{\text{OR}}(0,1),\ 1 \leftarrow g_{\text{OR}}(1,0),\ 1 \leftarrow g_{\text{OR}}(1,1)$$

Z teorii obliczeń wiemy, że każdy algorytm opisany w modelu maszyny RAM (random-access machine) może zostać zareprezentowany w postaci sieci bramek logicznych i wzajemnie (dowód polega na emulacji jednego modelu w drugim w czasie wielomianowym) [15]. W praktyce wystarczy zastosowanie trzech rodzajów bramek: NOT, XOR, AND, które w ciele skończonym $\mathbb{Z}_2$ (arytmetyka binarna) odpowiadają za negacje, dodawanie i mnożenie, pozwalając na reprezentację dowolnej innej bramki logicznej – są funkcjonalnie skończone. Co więcej, zapewniają algorytmiczną skończoność Turinga (*Turing completeness*), zatem pozwalają na implementację dowolnego algorytmu obliczeniowego.

Tak jak poprzednio, serwer szyfruje każdą z tablic prawdy losowo wygenerowanymi kluczami, z tą różnicą, że do każdej wartości w tablicy dodany będzie klucz umożliwiający odszyfrowanie dalej zagnieżdżonych bramek. Ewaluacja układu przebiega podobnie jak w poprzednio opisanym przykładzie – za pomocą nieświadomego transferu (OT) serwer oferuje parę kluczy $(k_w^{(0)}, k_w^{(1)})$, z której klient wybiera klucz odpowiadający bitowi wejściowemu dla każdego przewodu $w \in W$ będącego wejściem układu C.

Konstrukcja zaszyfrowanego układu (GC) przebiega następująco:

1. Serwer generuje dla każdego przewodu $w \in W$ parę losowo wygenerowanych kluczy $(k_w^{(0)}, k_w^{(1)})$, reprezentujących semantyczną wartość 0 i 1.

2. Serwer generuje dla każdego przewodu $w \in W$ jednobitowy klucz $p_w \in \{0,1\}$, który posłuży zaszyfrowaniu faktycznej wartości przewodu. Jeśli wartość w tablicy prawdy bramki wynosi v, to po zaszyfrowaniu wynosić będzie $e_w = v_w \oplus p_w$, gdzie $\oplus$ jest operatorem XOR[1]. Szyfrowanie to przeciwdziała dedukcji wartości wejściowej drugiej strony na podstawie wartości wyjściowej. Przykładowo wiedząc, że na wyjściu bramki OR jest zero, wiemy, że na wyjściu musiały być dwa zera.

3. Dla każdej bramki w układzie $g \in G$ serwer generuje zaszyfrowaną tablicę prawdy. Wykorzystuje do tego przygotowane w poprzednich krokach zaszyfrowane wartości przewodów $e_w = v_w \oplus p_w$ oraz klucze $(k_w^{(0)}, k_w^{(1)})$ reprezentujące semantyczne wartości 0 i 1 przewodu $w \in W$. Zaszyfrowana tablica prawdy T_g dla każdej z bramek $g \in G$ ma następującą postać:

$$T_g = \begin{pmatrix} E_{k_m^{(0\oplus p_m)}, k_n^{(0\oplus p_n)}}(k_o^{(g(0\oplus p_m, 0\oplus p_n)}) || (g(0\oplus p_m, 0\oplus p_n) \oplus p_o) \\ E_{k_m^{(0\oplus p_m)}, k_n^{(1\oplus p_n)}}(k_o^{(g(0\oplus p_m, 1\oplus p_n)}) || (g(0\oplus p_m, 1\oplus p_n) \oplus p_o) \\ E_{k_m^{(1\oplus p_m)}, k_n^{(0\oplus p_n)}}(k_o^{(g(1\oplus p_m, 0\oplus p_n)}) || (g(1\oplus p_m, 0\oplus p_n) \oplus p_o) \\ E_{k_m^{(1\oplus p_m)}, k_n^{(1\oplus p_n)}}(k_o^{(g(1\oplus p_m, 1\oplus p_n)}) || (g(1\oplus p_m, 1\oplus p_n) \oplus p_o) \end{pmatrix}$$

[1]Operator XOR może zostać użyty jako (idealna) funkcja szyfrująca. Jest to możliwe dzięki właściwości $(m \oplus k) \oplus k = m$, gdzie m jest wiadomością, a k kluczem szyfrującym.

gdzie m i n są przewodami wejściowymi, o jest przewodem wyjściowym, p_w jest bitem szyfrującym wartość przewodu $w \in W$, a $||$ jest operatorem konkatenacji.

W ten sposób zaszyfrowane tablice zostają wysłane do klienta w celu **wykonania układu**, który przebiega następująco:

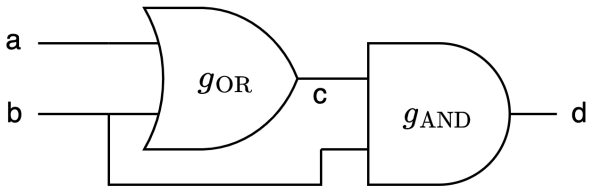
1. Serwer wysyła do klienta:
   - zaszyfrowane tablice T_g dla wszystkich bramek $g \in G$;
   - klucze szyfrujące $k_w^{(x)}$ dla każdego przewodu $w \in W$, które semantycznie odpowiadają danym wejściowym serwera, tj. jeśli dla przewodu i serwer wybrał wartość 0, wysyła on klucz $k_i^{(0 \oplus p_i)}$;
   - za pomocą nieświadomego transferu (OT) klucze szyfrujące $k_w^{(x)}$ dla każdego przewodu $w \in W$, które semantycznie odpowiadają danym wejściowym klienta;
   - bity p szyfrujące wartości przewodów wyjściowych układu.
2. Klient, korzystając z otrzymanych kluczy, dokonuje sekwencji rozszyfrowywania kolejnych bramek, a dokładniej, odpowiadających im zaszyfrowanych tablic prawdy. Ponieważ nie posiada on bitów szyfrujących p dla przewodów innych niż wyjściowe, nie może on wydedukować faktycznych wartości bramek.
3. Po wykonaniu sekwencji deszyfracji całego układu i otrzymaniu zaszyfrowanych wartości przewodów wyjściowych układu klient, korzystając z otrzymanych w pierwszym kroku kluczy p, odszyfrowuje wartości przewodów wyjściowych, ucząc się wartości wynikowej wykonanego układu.
4. W zależności od uzgodnienia klient wysyła (lub nie) wynik do serwera. W celu zapewnienia wiarygodności wyniku zwróconego przez klienta wynik bramek wyjściowych powinien zostać opatrzony podpisem cyfrowym wykonanym przez serwer.

Przykład protokołu GC

Weźmy dla przykładu funkcję $F(a, b) = (a \vee b) \wedge b$, gdzie a i b są wartościami logicznymi. Przewód a jest wejściem serwera, przewód b wejściem klienta, c jest przewodem pośrednim, a d jest przewodem wyjścia układu. Rys. 7.5 prezentuje układ C_F realizujący funkcję F. Układ składa się z przewodów $W = \{a, b, c, d\}$ oraz bramek $G = \{g_{\text{OR}}, g_{\text{AND}}\}$.

Konstrukcja zaszyfrowanego układu (GC) widocznego na rys. 7.6 przebiega następująco:

1. Serwer losowo generuje klucze szyfrujące wartości p_w dla każdego przewodu $w \in W$. Przyjmijmy $p_a = 1, p_b = 0, p_c = 1, p_d = 0$.



Rysunek 7.5: Układ C_F realizujący $F(a, b) = (a \vee b) \wedge b$

2. Serwer losowo generuje parę kluczy $(k_w^{(0)}, k_w^{(1)})$ odpowiadających semantycznie wartościom 0 i 1 dla każdego przewodu $w \in W$ oraz konkatenuje je z zaszyfrowaną (kluczem p_w) odpowiadającą wartością. Powstaje w ten sposób para $(k_w^{(0)}||(0 \oplus p_w), k_w^{(1)}||(1 \oplus p_w))$. Przykładowo dla przewodu a, gdzie klucz szyfrujący $p_a = 1$, para kluczy to $(k_a^{(0)}||(0 \oplus 1 = 1), k_a^{(1)}||(1 \oplus 1 = 0))$. Powstają w ten sposób pary kluczy:

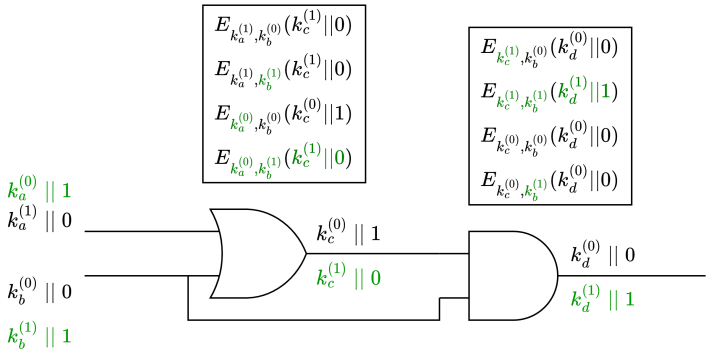
$$
\begin{gathered}
(k_a^{(0)}||1, k_a^{(1)}||0) \\
(k_b^{(0)}||0, k_b^{(1)}||1) \\
(k_c^{(0)}||1, k_c^{(1)}||0) \\
(k_d^{(0)}||0, k_d^{(1)}||1)
\end{gathered}
$$

3. Korzystając z kluczy k oraz p, serwer tworzy tablice prawdy dla każdej bramki. Przykładowo dla pierwszej bramki $T_{g_{\text{OR}}}$, gdzie przewodami wejściowymi są a i b, a wyjściowym jest c, tablica wygląda następująco:

$$
\begin{aligned}
T_{g_{\text{OR}}} &= \begin{pmatrix}
E_{k_a^{(0\oplus p_a)}, k_b^{(0\oplus p_b)}}(k_c^{(g(0\oplus p_a, 0\oplus p_b)}||(g(0 \oplus p_a, 0 \oplus p_b) \oplus p_c) \\
E_{k_a^{(0\oplus p_a)}, k_b^{(1\oplus p_b)}}(k_c^{(g(0\oplus p_a, 1\oplus p_b)}||(g(0 \oplus p_a, 1 \oplus p_b) \oplus p_c) \\
E_{k_a^{(1\oplus p_a)}, k_b^{(0\oplus p_b)}}(k_c^{(g(1\oplus p_a, 0\oplus p_b)}||(g(1 \oplus p_a, 0 \oplus p_b) \oplus p_c) \\
E_{k_a^{(1\oplus p_a)}, k_b^{(1\oplus p_b)}}(k_c^{(g(1\oplus p_a, 1\oplus p_b)}||(g(1 \oplus p_a, 1 \oplus p_b) \oplus p_c)
\end{pmatrix} \\
&= \begin{pmatrix}
E_{k_a^{(1)}, k_b^{(0)}}(k_c^{(1)}||0) \\
E_{k_a^{(1)}, k_b^{(1)}}(k_c^{(1)}||0) \\
E_{k_a^{(0)}, k_b^{(0)}}(k_c^{(0)}||1) \\
E_{k_a^{(0)}, k_b^{(1)}}(k_c^{(1)}||0)
\end{pmatrix}
\end{aligned}
$$

Gdy zaszyfrowany układ jest gotowy, możemy przejść do **fazy wykonania** (zwanej również fazą online). Przyjmijmy, że wartością wejściową serwera jest $a = 0$, klienta $b = 1$. Spodziewana wartość $F(0, 1) = (0 \vee 1) \wedge 1 = 1$.

Wykonanie układu przebiega następująco:



Rysunek 7.6: Zaszyfrowany układ (GC)

- Klient otrzymuje od serwera zaszyfrowane tablice $T_{g_{\text{OR}}}, T_{g_{\text{AND}}}$, klucz szyfrujący wyjście p_d oraz klucz odpowiadający semantycznie wartości wejściowej serwera. Serwer wybiera wartość 0 i przesyła odpowiadający klucz $k_a^0||1$. Klient widzi klucz k_a^0, który jest dla niego losowym ciągiem bitów, oraz zaszyfrowaną wartość 1, z której (bez klucza p_a) nie jest w stanie dowiedzieć się, czy faktyczna wartość to 0 czy 1.

- Korzystając z nieświadomego transferu, serwer oferuje klientowi parę kluczy $(k_b^{(0)}||0, k_b^{(1)}||1)$. Klient wybiera wartość 1 i otrzymuje klucz $k_b^1||1$.

- Posiadając wszystkie klucze wejściowe (k_a^0, $k_b^1||1$, oraz $p_d = 0$), klient rozpoczyna wykonywanie układu przez rozszyfrowywanie tablic. Klient wybiera wartość z tablicy na podstawie skonkatenowanych (uzewnętrznionych) zaszyfrowanych wartości przewodów wejściowych $e = v \oplus p$. Dla pierwszej bramki g_{OR} wartość uzewnętrzniona przewodu a wynosi 1, tak samo dla przewodu b uzewnętrzniona wartość wynosi 1, dlatego klient odszyfrowuje czwarty rząd tablicy $T_{g_{\text{OR}}}$. Na rys. 7.6 kolorem zielonym zaznaczone zostały wartości, których nauczył się klient. Wyjściem układu jest przewód d, którego wartość wynosi $k_d^{(1)}||1$. Korzystając z klucza $p_d = 0$, klient rozszyfrowuje ostateczną wartość $1 \oplus p_d = 1 \oplus 0 = 1$, która zgadza się z oczekiwaną wartością $F(0,1) = (0 \vee 1) \wedge 1 = 1$.

7.3.2. Kompilatory MPC

Funkcjonalność F, którą chcemy wykonać nieświadomie, musi zostać zamieniona na postać układu logicznego C_F, a dokładniej listy sieci (*netlist*), która następnie zostaje poddana unieświadomieniu przy pomocy protokołu GC.

Transformacja funkcjonalności F do postaci listy sieci jest metodą powszechnie stosowaną w projektowaniu układów cyfrowych. Można ją wykonywać natywnie, korzystając z języków takich jak np. Verilog, VHDL lub z syntezatorów

HLS (*high-level synthesis*) pozwalających na syntezę z języków wysokiego poziomu, np. z języka C (SPARK [18], CBMC-GC [21]) lub Python (MyHDL[2]). W rezultacie powstaje sprzętowy opis układu logicznego w formacie HDL (*hardware description language*), który pozwala na stworzenie uproszczonej listy sieci opisującej elementy układu i połączenia między nimi.

Następnie tak utworzona lista sieci może zostać uruchomiona nieświadomie przy pomocy protokołu GC.

Istnieją narzędzia umożliwiające dokonywanie wyżej wymienionych transformacji. Część z nich przyjmuje za dane wejściowe sprzętowy opis układ logicznego HDL, przykładowo: TinyGarble [33][3], ABY [13][4]. Inne dostarczają pełny zestaw narzędzi, od języka programowania do środowiska uruchomieniowego, przykładami są: MP-SPDZ [23][5], MPyC [32][6], EzPC [11][7], EMP-Toolkit [34][8].

Analiza porównawcza kompilatorów MPC dostępna jest w przeglądzie [19][9].

7.4. Nieświadome sieci neuronowe

Z sekcji 7.3 wiemy, jak zamienić dowolny algorytm na algorytm w wersji nieświadomej, korzystając z protokołu GC. Z sekcji 7.2 wiemy również, jak dokonywana jest predykcja przy użyciu sieci neuronowych. Pozostaje połączyć zdobytą wiedzę do stworzenia nieświadomej predykcji sieci neuronowej.

Predykcja przy użyciu NN może zostać zamodelowana jako rekurencyjna sekwencja mnożenia i dodawania macierzy oraz funkcji aktywacji/*poolingu*:

$$z := W_L * f_{L-1}(...f_1(W_1 * X + B_1)...) + B_L$$

gdzie:

- $W_1, W_2, ..., W_L$ są macierzami wag sygnałów pojawiających się na wejściach poszczególnych neuronów każdej z L warstw;
- $B_1, B_2, ..., B_L$ są wektorami wyrazów wolnych (*bias*) dla każdej z L warstw;
- $f_1, ..., f_{L-1}$ są funkcjami aktywacji/*poolingu* dla każdej z L warstw;
- X jest wektorem wejściowym;
- z jest wynikiem predykcji.

Celem nieświadomych sieci neuronowych jest obliczenie wartości z w taki sposób, aby klient nie dowiedział się niczego o $(W_1, W_2, ..., W_L)$ oraz $(B_1, B_2, ..., B_L)$, a serwer nie dowiedział się niczego o X oraz z.

[2]https://www.myhdl.org/
[3]https://github.com/esonghori/TinyGarble
[4]https://github.com/encryptogroup/ABY
[5]https://github.com/data61/MP-SPDZ
[6]https://github.com/lschoe/mpyc/
[7]https://github.com/mpc-msri/EzPC
[8]https://github.com/emp-toolkit
[9]https://github.com/MPC-SoK/frameworks

Aby osiągnąć taką izolację, podstawiamy dane wejściowe każdej ze stron pod model funkcjonalności $F_{2PC}(a, b)$, gdzie a jest prywatną daną wejściową jednej ze stron, a b prywatną daną wejściową drugiej ze stron. W naszym przypadku $a = X$ $b = ((W_1, W_2, ..., W_L), (B_1, B_2, ..., B_L))$. W wyniku powstaje:

$$F_{2PC}(X, ((W_1, W_2, ..., W_L), (B_1, B_2, ..., B_L))) \\ = W_L * f_{L-1}(...f_1(W_1 * X + B_1)...) + B_L$$

Funkcje aktywacji $f_1, ..., f_{L-1}$ oraz rozmiary macierzy są częścią znanego obu stronom układu logicznego, który zostaje wykonany wspólnie przez obie strony.

Tak zareprezentowana predykcja jako dwuargumentowa funkcja F zostaje zamieniona na postać listy sieci, a następnie wykonana wspólnie przez obie strony przy wykorzystaniu protokołu GC opisanego w sekcji 7.3.

Finalny przebieg protokołu został zwizualizowany na rys. 7.7 i wygląda następująco:

1. Serwer dokonuje syntezy architektury modelu sieci neuronowej do postaci netlisty, którą przesyła klientowi.
2. Klient tworzy, postępując zgodnie z protokołem GC, zaszyfrowany układ (*garbled circuit*), a dokładniej zaszyfrowane tablice (*garbled tables*) i przesyła je do serwera.
3. Klient przesyła klucze reprezentujące jego dane wejściowe oraz oferuje serwerowi klucze odpowiadające bitom reprezentującym wyuczone wagi modelu.
4. Serwer, korzystając z otrzymanych kluczy, realizuje układ (odszyfrowując kolejne tablice), a następnie wysyła zaszyfrowany wynik klientowi.
5. Klient odszyfrowuje wynik, otrzymując predykcję.

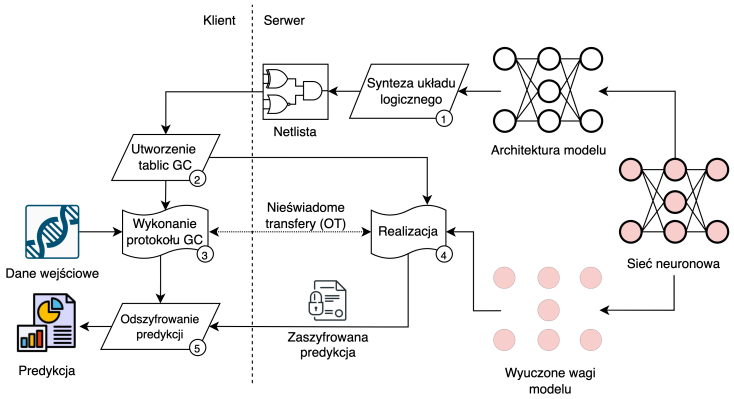
Nieświadome transformacje liniowe

Jak ustaliliśmy w sekcji 7.2.3, transformacje liniowe sprowadzają się do mnożenia i dodawania macierzy.

Mnożenie i dodawanie macierzy polegają na wykonaniu operacji dodawania i mnożenia wag, wyrazów wolnych oraz danych wejściowych.

Nieświadome transformacje nieliniowe

Funkcja aktywacji przedziałami liniowa jest prosta do unieświadomienia. Przykładowo ReLU polega na skonstruowaniu układu, który porówna y oraz 0 i na podstawie wyniku zwróci y lub 0. Wszystkie funkcje aktywacji przedziałami liniowe są możliwe do zrealizowania operacjami odejmowania, dodawania i mnożenia.



Rysunek 7.7: Przebieg nieświadomej predykcji sieci neuronowych

Gładka funkcja aktywacji jest trudniejsza do unieświadomienia, ponieważ wymaga skomplikowanych operacji dzielenia oraz potęgowania przez zmienną. Dlatego w praktyce gładkie funkcje zamienia się na przybliżone funkcje przedziałami liniowe. Dzięki temu zyskuje się wydajniejsze i prostsze w kompilacji funkcje aktywacji, kosztem nieznacznego spadku dokładności predykcji.

Pooling sprowadza się do wielokrotnego wykonania operacji porównania (*max pooling*) lub sumowania (*mean pooling*) na elementach agregowanego regionu, które również są proste w implementacji.

7.5. Stan techniki

Opisane w tym rozdziale podejście do wykonywania nieświadomych predykcji bazuje na rozwiązaniu **DeepSecure** [31] (DAC 2018), które w całości bazuje na protokole GC. Istnieje jednak wiele innych podejść korzystających z innych protokołów kryptograficznych do osiągania MPC.

Homomorficzne szyfrowanie (*homomorphic encryption*, HE) pozwala na wykonywanie obliczeń na zaszyfrowanych danych. Istnieją dwie wersje HE: pełne (*fully homomorphic encryption*, FHE) [9] oraz niepełne (*partially homomorphic encryption*, PHE) [8, 27]. FHE pozwala na wykonywanie obliczeń na zaszyfrowanych danych, jednak przy bardzo dużym narzucie obliczeniowym, co wyklucza je z większości praktycznych zastosowań. PHE natomiast ma dużo mniejszy narzut obliczeniowy, jednak pozwala na wykonywanie tylko podzbioru operacji (dodawanie lub mnożenie) lub wymaga ograniczenia głębokości układu reprezentującego algorytm. Co gorsza, funkcje nieliniowe, takie jak ReLU, nie są wspierane przez HE [30].

Współdzielenie sekretu (*secret sharing*, SS) pozwala na dokonywanie obliczeń w modelu MPC przy stosunkowo niskim narzucie obliczeniowym. SS polega na potraktowaniu danych wejściowych każdej ze stron jako sekret, który następnie zostaje podzielony pomiędzy każdą ze stron. Korzystając z właściwości SS, części sekretu mogą być poddawane operacjom arytmetycznym, a następnie zrekonstruowane, co prowadzi do uzyskania wyniku obliczeń [14]. Pierwszy protokół MPC korzystający z tej właściwości to **GMW** [17]. Pozwala on na wykonanie operacji NOT oraz XOR bez wzajemnej interakcji pomiędzy stronami; bramki AND – podobnie jak w przypadku GC – wymagają wykonania nieświadomego transferu (OT). Rozwinięciem protokołu GMW jest **BGW** [4] bazujący na schemacie współdzielenia sekretu Shamira (*Shamir secret sharing*, SSS), a dokładniej na homomorficzności wielomianów. Reprezentacja danych w postaci wielomianów pozwala na przedstawienie funkcjonalności F w postaci układu arytmetycznego, który umożliwia bardziej kompaktową reprezentację algorytmu niż układ logiczny. Przy wykorzystaniu homomorficzności wielomianów zarówno dodawanie, jak i mnożenie może zostać wykonane lokalnie – bez konieczności komunikacji między stronami. Obliczenia przy użyciu homomorficzności wielomianów w schemacie Shamira mają jednak jedną wadę. Mnożenie dwóch wielomianów stopnia t powoduje dwukrotny wzrost stopnia wielomianu, co z kolei sprawia, że rekonstrukcja sekretu wymaga użycia dwukrotnie większej liczby części sekretu. Powoduje to konieczność stosowania skomplikowanej procedury redukcji stopnia wielomianu oraz wymaga, aby co najmniej $2t + 1 = 3$ niezależne strony (brak większościowej koalicji) brały udział w protokole, a co za tym idzie, stworzenie 2PC nie jest możliwe (wymagane są co najmniej trzy strony). Z tego powodu SS często nie wspiera operacji mnożenia, a jedynie operację dodawania. Tak zawężony w swej funkcjonalności schemat nazywany jest addytywnym dzieleniem sekretu (*additive secret sharing*, ASS).

Trójki Bravera. Rozwiązaniem powyższych problemów jest zastosowanie – przełomowych w obszarze MPC – trójek Bravera [2], które pozwalają na dokonywanie operacji mnożenia w prostszy sposób. Trójki Bravera wymagają podziału protokołu na dwie fazy: i) jednorazowa faza offline, ii) każdorazowa faza online.

- W **fazie offline** dla każdej bramki mnożenia generowana jest trójka liczb (a, b, c) takich, że a i b są losowymi wartościami, a $c = ab$. Każda z liczb generowana jest w częściach, w taki sposób, aby żadna ze stron protokołu nie miała wglądu do pełnej wartości liczby, a jedynie do swojej części. Dopiero współpraca wszystkich stron protokołu i zebranie wszystkich części pozwalają na rekonstrukcję właściwej liczby. Część liczby x oznaczana jest symbolem $[x]$. Każda ze stron posiada zestaw części każdej z liczb, tj. $[a], [b], [c]$. Do wygenerowania współdzielonych trójek Bravera można wykorzystać protokół GMW [14] lub PHE [25].

- W **fazie online**, gdy w układzie pojawia się bramka odpowiedzialna za mnożenie dwóch wartości α i β, zostają one współdzielone przez obie strony jako wartości $[\alpha]$ oraz $[\beta]$. Następnie każda ze stron, używając swo-

ich części, liczy zamaskowaną wartość $[d] = [\alpha] - [a]$ oraz $[e] = [\beta] - [b]$, a następnie wysyła pozostałej stronie $[d]$ oraz $[e]$.

Jako że wartość jest zamaskowana wartością, nie niesie ona informacji o faktycznej wartości. Po otrzymaniu pozostałej części wartości d oraz e możliwa jest ich rekonstrukcja. Następnie, korzystając z równoważności:

$$\begin{aligned} & de + db + ae + c \\ & = de + db + ae + ab \\ & = (d + a)(e + b) \\ & = (\alpha - a + a)(\beta - b + b) \\ & = \alpha\beta \end{aligned}$$

każda ze stron liczy lokalnie część wyniku faktycznego mnożenia $[\alpha\beta] = de + d[b] + e[a] + [c]$.

Optymalizacja polega na tym, że dodawanie oraz mnożenie współdzielonej wartości $[x]$ przez dowolną stałą (która nie jest współdzielona) są proste i mogą zostać przeprowadzone lokalnie, bez konieczności wykonywania skomplikowanej procedury nieświadomego transferu i redukcji stopnia wielomianu [14].

MiniONN [25] (CCS 2017) w porównaniu z DeepSecure jest rozwiązaniem bardziej złożonym, ponieważ korzysta ze wszystkich wyżej wymienionych kryptosystemów: GC, HE, ASS oraz trójek Beavera. Kompilacja układu dokonywana jest za pomocą narzędzia ABY [13]. Mimo narzutów związanych z konwersjami pomiędzy kryptosystemami takie rozwiązanie (nazywane hybrydowym) osiąga dużo większą wydajność.

Chameleon [29] (AsiaCCS 2018), podobnie jak MiniONN, jest rozwiązaniem hybrydowym korzystającym z kompilatora ABY. Poprawia on wydajność protokołu MiniONN, zastępując złożoną fazę generowania trójek Bravera przez częściowo zaufaną trzecią stronę (*semi-honest third party*, STP), która odpowiedzialna jest za dostarczanie źródła niezależnej losowości dla obu stron.

Gazelle [22] (USENIX 2018) wprowadza kolejne optymalizacje konwersji przejść pomiędzy kryptosystemami oraz zoptymalizowane operacje SIMD (*single instruction multiple data*) dla dodawania oraz mnożenia macierzy, osiągając w ten sposób 20–30-krotnie większą wydajność w fazie online w porównaniu z poprzednimi rozwiązaniami (MiniONN oraz Chameleon).

XONN [30] (USENIX 2019) jest systemem stworzonym przez Microsoft, korzystającym z koncepcji binarnych sieci neuronowych (*binary neural network*, BNN), w których wagi i aktywację przyjmują tylko dwie wartości, ± 1. Dodatkowo protokół XONN kładzie nacisk na syntezę układu przy użyciu jak największej liczby bramek XNOR, które pozwalają na wykonywanie mnożenia bez

konieczności wykonania nieświadomego transferu (OT). W porównaniu z poprzednimi rozwiązaniami, gdzie liczba rund nieświadomych transferów była liniowa co do liczby warstw w NN, w XONN liczba rund jest stała, niezależnie od liczby warstw NN. Wynikiem tego jest siedmiokrotne przyspieszenie w porównaniu z Gazelle i około stukrotne przyspieszenie w porównaniu z MiniONN. XONN pozwala na interpretację wysokopoziomowych modeli NN bezpośrednio z biblioteki Keras.

CrypTFlow [24] (2019) to kolejny system stworzony przez Microsoft, składającym się z trzech komponentów. Pierwszym (Athos) jest kompilator pozwalający na zamianę dowolnego modelu NN zareprezentowanego przy użyciu Tensorflow do nieświadomego protokołu MPC. Drugim komponentem (Porthos) jest ulepszony protokół MPC pozwalający na nieświadome predykcje przy użyciu dużych sieci, jak ResNet50 lub DenseNet121, w czasie około 30 sekund. Trzecim komponentem (Aramis) jest protokół bazujący na założeniach bezpieczeństwa sprzętowego (np. Intel SGX), który zwiększa bezpieczeństwo oraz poprawia wydajność w porównaniu z rozwiązaniami czysto kryptograficznymi.

Delphi [26] (USENIX 2020) jest kolejną iteracją usprawnień w protokołach nieświadomej predykcji. W porównaniu z Gazelle Delphi zmniejsza ilość transferu danych z 560 MB do 60 MB oraz skraca czas predykcji z 82 sekund do 3,8 sekundy dla modelu ResNet-32 [20]. Jest to możliwe dzięki dwóm transformacjom: przeniesieniu większości obliczeń do fazy offline oraz zamianie nieliniowych funkcji aktywacji na bardziej MPC-przyjazne funkcje wielomianowe (kosztem około 1–5% spadku predykcji).

Przegląd rozwiązań w obszarze nieświadomych predykcji NN dostępny jest w pracy [10]. Wydajnościowe porównanie wszystkich rozwiązań jest problematyczne z powodu różnic modeli w wykonanych testach wydajnościowych oraz braku kodów źródłowych, co uniemożliwia przeprowadzenie reprodukowalności wyników.

7.6. Wnioski końcowe

Technologia umożliwiająca nieświadome obliczenia (MPC) znajduje zastosowanie w wielu obszarach, takich jak: biometryczne dopasowanie, rozpoznawanie twarzy, klasyfikacja danych, elektroniczne aukcje, internetowe wybory, zdalne diagnozy i bezpieczne wyszukiwanie.

Pomimo różnorodności protokołów używanych do unieświadamiania sieci neuronowych protokół *garbled circuit* (GC) używany jest w większości rozwiązań i nic nie wskazuje na to, aby sytuacja miała się zmienić. Dlatego w tym rozdziale skupiliśmy się właśnie na nim.

Najnowsze rozwiązania nieświadomych sieci neuronowych są rozwiązaniami hybrydowymi, tj. korzystają z wielu kryptosystemów w celu optymalizacji wykonania poszczególnych podprocedur.

W celu ułatwienia wytwarzania protokołów realizujących nieświadome obliczenia powstaje szereg kompilatorów pozwalających na wysokopoziomowy opis układu, automatyczny dobór kryptosystemów oraz konwersje pomiędzy nimi [10, 11, 13, 19].

Można zaobserwować dwa nurty rozwoju technologii nieświadomych obliczeń. Jeden z nich stara się zwiększyć wydajność, zmniejszyć obciążenie obliczeniowe, skrócić czas wykonania, przenieść jak największą ilość obliczeń do fazy offline oraz zmniejszyć narzut komunikacji. Drugi nurt stara się zwiększyć przystępność technologii, zapewnia kompilatory dla wysokopoziomowych języków oraz API do bezpośredniej integracji z bibliotekami i standardami, jak ONNX, TensorFlow, Keras i PyTorch.

O optymistycznej przyszłości protokołów nieświadomych obliczeń świadczy fakt, że są one rozwijane oraz wykorzystywane przez duże korporacje, jak Microsoft [16, 24] oraz Apple [1]. Dodatkowo najnowsze rozwiązania w obszarze *blockchain* bazują na technologii MPC w zakresie prywatnego wykonywania transakcji [6], a nawet inteligentnych kontraktów (*smart contracts*) [7].

Bibliografia

[1] *CSAM Detection - Technical Summary.* `https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf`. (Accessed on 07/08/2022).

[2] Beaver D. *Efficient multiparty protocols using circuit randomization.* In *Annual International Cryptology Conference*, pp. 420–432. Springer, 1991.

[3] Beaver D., Micali S., Rogaway P. *The round complexity of secure protocols.* In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pp. 503–513. 1990.

[4] Ben-Or M., Goldwasser S., Wigderson A. *Completeness theorems for non-cryptographic fault-tolerant distributed computation.* In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 351–371. 2019.

[5] Bost R., et al. *Machine learning classification over encrypted data. Cryptology ePrint Archive*, 2014.

[6] Bowe S., Gabizon A., Green M.D. *A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK.* In *International Conference on Financial Cryptography and Data Security*, pp. 64–77. Springer, 2018.

[7] Bowe S., et al. *Zexe: Enabling decentralized private computation.* In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 947–964. IEEE, 2020.

[8] Brakerski Z., Gentry C., Vaikuntanathan V. *(Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[9] Brakerski Z., Vaikuntanathan V. *Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on computing*, 43(2):831–871, 2014.

[10] Cabrero-Holgueras J., Pastrana S. *SoK: Privacy-preserving computation techniques for deep learning. Proceedings on Privacy Enhancing Technologies*, 2021(4):139–162, 2021.

[11] Chandran N., et al. *EzPC: programmable and efficient secure two-party computation for machine learning.* In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 496–511. IEEE, 2019.

[12] Chellapilla K., Puri S., Simard P. *High performance convolutional neural networks for document processing.* In *Tenth international workshop on frontiers in handwriting recognition.* Suvisoft, 2006.

[13] Demmler D., Schneider T., Zohner M. *ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation.*

[14] Evans D., et al. *A pragmatic introduction to secure multi-party computation. Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018.

[15] Giaro K. *Elementy kwantowego modelu obliczeń i algorytmiki kwantowej: łagodne wprowadzenie do informatyki kwantowej.* Wydawnictwo Naukowe OWSIiZ.

[16] Gilad-Bachrach R., et al. *Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy.* In *International conference on machine learning*, pp. 201–210. PMLR, 2016.

[17] Goldreich O., Micali S., Wigderson A. *How to play any mental game, or a completeness theorem for protocols with honest majority.* In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 307–328. 2019.

[18] Gupta S., et al. *SPARK: a parallelizing approach to the high-level synthesis of digital circuits.* Springer Science & Business Media, 2007.

[19] Hastings M., et al. *Sok: General purpose compilers for secure multi-party computation.* In *2019 IEEE symposium on security and privacy (SP)*, pp. 1220–1237. IEEE, 2019.

[20] He K., et al. *Deep residual learning for image recognition.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778. 2016.

[21] Holzer A., et al. *Secure two-party computations in ANSI C*. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 772–783. 2012.

[22] Juvekar C., Vaikuntanathan V., Chandrakasan A. {*GAZELLE*}*: A low latency framework for secure neural network inference*. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1651–1669. 2018.

[23] Keller M. *MP-SPDZ: A Versatile Framework for Multi-Party Computation*. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020. URL `https://doi.org/10.1145/3372297.3417872`.

[24] Kumar N., et al. *Cryptflow: Secure tensorflow inference*. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 336–353. IEEE, 2020.

[25] Liu J., et al. *Oblivious neural network predictions via minionn transformations*. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 619–631. 2017.

[26] Mishra P., et al. *Delphi: A cryptographic inference service for neural networks*. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522. 2020.

[27] Paillier P. *Public-key cryptosystems based on composite degree residuosity classes*. In *International conference on the theory and applications of cryptographic techniques*, pp. 223–238. Springer, 1999.

[28] Rabin M.O. *How to exchange secrets with oblivious transfer*. *Cryptology ePrint Archive*, 2005.

[29] Riazi M.S., et al. *Chameleon: A hybrid secure computation framework for machine learning applications*. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pp. 707–721. 2018.

[30] Riazi M.S., et al. {*XONN*}*:*{*XNOR-based*} *Oblivious Deep Neural Network Inference*. In *28th USENIX Security Symposium (USENIX Security 19)*, pp. 1501–1518. 2019.

[31] Rouhani B.D., Riazi M.S., Koushanfar F. *Deepsecure: Scalable provably-secure deep learning*. In *Proceedings of the 55th annual design automation conference*, pp. 1–6. 2018.

[32] Schoenmakers B. *MPyC–Python Package for Secure Multiparty Computation*. In *Workshop on the Theory and Practice of MPC*. 2018.

[33] Songhori E.M., et al. *Tinygarble: Highly compressed and scalable sequential garbled circuits*. In *2015 IEEE Symposium on Security and Privacy*, pp. 411–428. IEEE, 2015.

[34] Wang X., Malozemoff A.J., Katz J. *EMP-toolkit: Efficient MultiParty computation toolkit*. `https://github.com/emp-toolkit`, 2016.

[35] Wu D.J., et al. *Privately evaluating decision trees and random forests. Cryptology ePrint Archive*, 2015.

[36] Yao A.C. *Protocols for secure computations*. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160–164. IEEE, 1982.

Rozdział 8

Metody generatywne w problemach kategoryzacji

Rafał Lipiński[1] , Jakub Łuczkiewicz[1] , Arkadiusz Szamocki[1] , Julian Szymański[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
julian.szymanski@eti.pg.edu.pl

Abstrakt

W rozdziale przedstawiono zastosowanie metod generatywnych w zadaniach kategoryzacji. Wskazano podział na dwa zadania – nadzorowane (*supervised learning*) i nienadzorowane (*unsupervised learning*) oraz ich przykładowe zastosowania. W ramach metod nadzorowanych opisano klasyfikator Bayesa oraz zastosowanie wielowymiarowego rozkładu Gaussa. Do rozwiązania zadania uczenia nienadzorowanego z użyciem podejścia generatywnego przedstawiono *Gaussian mixture model*. W rozdziale opisano również generatywną sieć neuronową wykorzystującą architekturę autoenkodera w modelu *variational autoencoder*.

Słowa kluczowe: autoenkoder wariacyjny, *Gaussian mixture model*, naiwny klasyfikator Bayesa, kategoryzacja generatywna

8.1. Wstęp

Uczenie maszynowe jest dziedziną informatyki, w której algorytmy są budowane na podstawie danych wejściowych. Typowym zadaniem jest tu sytuacja, gdy program musi poprawnie przypisać wprowadzone do niego obiekty do klas lub też stworzyć pewne abstrakcyjne grupy i skojarzyć każdy obiekt z jedną z nich. Problem ten ogólnie nazywany jest kategoryzacją.

Wśród metod kategoryzacji wyróżnia się podejścia nadzorowane, gdzie zestaw obiektów uczących jest opatrzony etykietą, która zawiera informację o przynależności do danej klasy. Gdy obiekty nie posiadają żadnej informacji o swojej kategorii, kategoryzacja odbywa się poprzez automatyczny podział na grupy i mówimy wtedy o uczeniu nienadzorowanym.

Wynikiem kategoryzacji jest przypisanie obiektów do klas lub kategorii, a zadanie to może być rozwiązane z użyciem wielu podejść, w szczególności można rozróżnić algorytmy generatywne oraz dyskryminatywne [8].

Metody dyskryminatywne jako wyniki zwracają zdefiniowane powierzchnie decyzyjne, określające granice pomiędzy klasami. W najprostszym przypadku może być to linia prosta, powyżej której znajduje się klasa pierwsza, a poniżej klasa druga. W wielu sytuacjach taka metoda klasyfikacji może okazać się wystarczająca. Ponadto wynik w takiej postaci jest prosty do interpretacji i nie budzi wątpliwości, do której klasy należy dany obiekt. Niestety często występuje potrzeba bardziej szczegółowego opisu rozmieszczenia punktów w przestrzeni i do tego wykorzystywane są bardziej złożone funkcje.

Metody generatywne budują model w postaci opisu rozkładu obiektów w przestrzeni poprzez wyznaczenie funkcji gęstości prawdopodobieństwa przynależności do danej klasy. W takim wypadku uzyskujemy właściwości grup nie tylko w granicach ich istnienia, lecz także w ich środku.

8.2. Generatywne metody nadzorowane

Uczenie nadzorowane cechuje się obecnością informacji, czy prognoza modelu jest poprawna, czy nie. Dzięki temu algorytm uczy się zależności między cechami a klasami i na tej podstawie podejmuje decyzje kategoryzacyjne. Metoda ta wymaga danych, dla których oznaczone są klasy, co często wymaga kosztownego udziału czynnika ludzkiego.

8.2.1. Naiwny klasyfikator Bayesa

Klasyfikator Bayesa [3] to prosty w implementacji klasyfikator probabilistyczny, oparty na twierdzeniu Bayesa, określony wzorem (8.1):

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{8.1}$$

W przypadku wektora cech $X = (x_1, x_2...x_n)$ dla danej klasy C_k można zapisać powyższy wzór w postaci formuły (8.2):

$$p(C_k|X) = \frac{p(X|C_k)p(C_k)}{p(X)} \tag{8.2}$$

Prawdopodobieństwo $p(C_k|X)$ jest nazywane prawdopodobieństwem *a posteriori*, które wyznaczane jest na podstawie obserwacji zjawiska, tu – częstości wystąpień cech w obiektach określonej klasy. Określa ono prawdopodobieństwo przynależności obiektu o cechach $x_1, x_2, ..., x_n$ do klasy C_k.

Na podstawie tej wartości dokonywana jest ostateczna klasyfikacja. Wartość *a posteriori* jest liczona dla każdej klasy i wybierana jest etykieta tej klasy, dla której prawdopodobieństwo jest największe.

Ponieważ wektor cech X jest stały, a interesuje nas, czy prawdopodobieństwo *a posteriori* jednej klasy jest większe niż innej, możemy zignorować mianownik i uprościć twierdzenie Bayesa do równania (8.3):

$$p(C_k|X) = p(X|C_k)p(C_k) \tag{8.3}$$

Zmienna $p(C_k)$ jest nazywana prawdopodobieństwem *a priori* i zależy ona od liczby wystąpień obiektów danej klasy w stosunku do liczby wszystkich obiektów, co wyraża wzór (8.4).

$$p(C_k) = \frac{\text{liczba obiektów klasy } k}{\text{liczba wszystkich obiektów}} \tag{8.4}$$

Prawdopodobieństwo $p(X|C_k)$, korzystając z reguły łańcuchowej, możemy rozwinąć do formy iloczynu prawdopodobieństw cech w klasie, co przedstawiono w równaniu (8.5) [11]:

$$\begin{aligned} p(X|C_k) =& p(x_1, x_2...x_n|C_k) = \\ & p(x_1|C_k)p(x_2, x_3...x_n|C_k) = \\ & p(x_1|C_k)p(x_2|C_k)p(x_3, x_4...x_n|C_k) = \\ & p(x_1|C_k)p(x_2|C_k)...p(x_n|C_k) = \prod_{i=1}^{n} p(x_i|C) \end{aligned} \tag{8.5}$$

Równanie (8.5) określa iloczyn prawdopodobieństw wystąpienia każdej z cech $x_1, x_2, ..., x_n$ w klasie C_k. Każdą z tych wartości można policzyć z rozkładu normalnego, gdzie μ_{ik} i σ_{ik} to średnia i odchylenie standardowe wartości cechy x_i w klasie C_k:

$$p(x_i|C_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \exp\left(\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}\right) \tag{8.6}$$

Ostatecznie otrzymujemy wzór (8.7) określający prawdopodobieństwo przynależności wektora cech X do klasy C_k:

$$p(C_k|X) = p(C_k)p(x_1|C_k)p(x_2|C_k)...p(x_n|C_k) = p(C_k)\prod_{i=1}^{n} p(x_i|C) \tag{8.7}$$

Przykład

By zobrazować działanie naiwnego klasyfikatora Bayesa, przyjmijmy zbiór obiektów przedstawionych w tablicy 8.1, opisanych czterema cechami i przynależących do jednej z dwóch klas.

Tablica 8.1: Zestaw danych treningowych

Klasa	Cecha x_1	Cecha x_2	Cecha x_3	Cecha x_4
0	2,3	1,2	4,5	0,81
0	3,3	1,8	3,8	0,32
1	1,2	2,3	3,1	0,31
1	1,8	2,1	2,6	0,13
1	2,1	2,8	3,2	0,02

Dla tych danych wyliczamy średnią i odchylenie standardowe każdej z cech w klasie. Przykładowo średnia μ_{x_1} dla klasy 0 oznacza przeciętną wartość cechy x_1 w klasie 0: $\mu_{x_1} = \frac{2,3+3,3}{2} = 2,8$. σ_{x_1} dla klasy 0 oznacza odchylenie standardowe wartości cechy x_1 w klasie 0: $\sigma_{x_1} = \sqrt{\frac{(2,3-2,8)^2+(3,3-2,8)^2}{2}} = 0,5$. Wartości średniej i odchylenia standardowego każdej z cech w obu klasach podano w tablicy 8.2.

Tablica 8.2: Dane obliczone

Klasa	μ_{x_1}	σ_{x_1}	μ_{x_2}	σ_{x_2}	μ_{x_3}	σ_{x_3}	μ_{x_4}	σ_{x_4}
0	2,8	0,5	1,5	0,18	4,15	0,245	0,565	0,120
1	1,7	0,21	2,4	0,13	2,96	0,103	0,153	0,0214

Prawdopodobieństwo *a priori* klas $p(C_k)$ można wyliczyć ze wzoru (8.4):

$$p(C_0) = \frac{\text{liczba obiektów klasy 0}}{\text{liczba wszystkich obiektów}} = \frac{2}{5} \tag{8.8}$$

$$p(C_1) = \frac{\text{liczba obiektów klasy 1}}{\text{liczba wszystkich obiektów}} = \frac{3}{5} \tag{8.9}$$

Przyjmijmy wektor cech $X = (x_1, x_2, x_3, x_4)$, przedstawiony w tablicy 8.3, który należy przypisać do jednej z klas.

Tablica 8.3: Obiekt do klasyfikacji

Klasa	Cecha x_1	Cecha x_2	Cecha x_3	Cecha x_4
?	2,5	1,9	3,2	0,1

Można policzyć prawdopodobieństwo przynależności wektora do klasy 0 za pomocą wzoru (8.7). Prawdopodobieństwo *a priori* jest już znane, a braku-

jące wartości $p(x_1|C_0)$ oraz $p(x_2|C_0)$ można policzyć, wykorzystując rozkład normalny (8.6).

$$p(x_1|C_0) = \frac{1}{\sigma_{x_1C_0}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_1C_0})^2}{2\sigma^2_{x_1C_0}}) = \frac{1}{0{,}5\sqrt{2\pi}}\exp(\frac{-(x-2{,}8)^2}{2\cdot 0{,}5^2}) = 0{,}665 \tag{8.10}$$

$$p(x_2|C_0) = \frac{1}{\sigma_{x_2C_0}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_2C_0})^2}{2\sigma^2_{x_2C_0}}) = \frac{1}{0{,}18\sqrt{2\pi}}\exp(\frac{-(x-1{,}5)^2}{2\cdot 0{,}18^2}) = 0{,}188 \tag{8.11}$$

$$p(x_3|C_0) = \frac{1}{\sigma_{x_3C_0}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_3C_0})^2}{2\sigma^2_{x_3C_0}}) = \frac{1}{0{,}245\sqrt{2\pi}}\exp(\frac{-(x-4{,}15)^2}{2\cdot 0{,}245^2}) = 8{,}85{\cdot}10^{-4} \tag{8.12}$$

$$p(x_4|C_0) = \frac{1}{\sigma_{x_4C_0}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_4C_0})^2}{2\sigma^2_{x_4C_0}}) = \frac{1}{0{,}12\sqrt{2\pi}}\exp(\frac{-(x-5{,}65)^2}{2\cdot 0{,}12^2}) = 1{,}84{\cdot}10^{-3} \tag{8.13}$$

Korzystając ze wzoru (8.7), można obliczyć prawdopodobieństwo przynależności obiektu do klasy 0.

$$\begin{aligned} p(C_0|X) &= p(C_0)p(x_1|C_0)p(x_2|C_0)p(x_3|C_0)p(x_4|C_0) = \\ &= 0{,}4\cdot 0{,}665\cdot 0{,}188\cdot 8{,}85\cdot 10^{-4}\cdot 1{,}84\cdot 10^{-3} = 8{,}12\cdot 10^{-8} \end{aligned} \tag{8.14}$$

Wszystkie powyższe kroki zostają powtórzone również dla klasy 1.

$$p(x_1|C_1) = \frac{1}{\sigma_{x_1C_1}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_1C_1})^2}{2\sigma^2_{x_1C_1}}) = \frac{1}{0{,}21\sqrt{2\pi}}\exp(\frac{-(x-1{,}7)^2}{2\cdot 0{,}21^2}) = 1{,}34{\cdot}10^{-3} \tag{8.15}$$

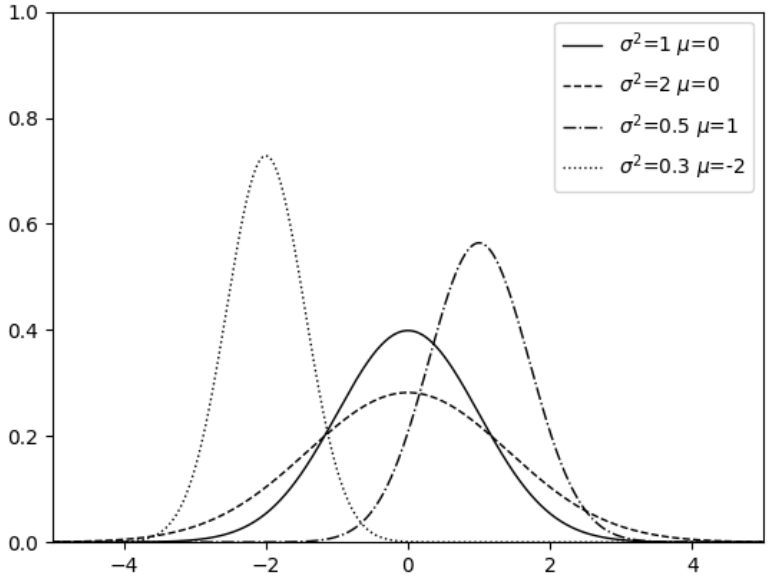
$$p(x_2|C_1) = \frac{1}{\sigma_{x_2C_1}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_2C_1})^2}{2\sigma^2_{x_2C_1}}) = \frac{1}{0{,}13\sqrt{2\pi}}\exp(\frac{-(x-2{,}4)^2}{2\cdot 0{,}13^2}) = 1{,}88{\cdot}10^{-3} \tag{8.16}$$

$$p(x_3|C_1) = \frac{1}{\sigma_{x_3C_1}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_3C_1})^2}{2\sigma^2_{x_3C_1}}) = \frac{1}{0{,}103\sqrt{2\pi}}\exp(\frac{-(x-2{,}96)^2}{2\cdot 0{,}103^2}) = 0{,}302 \tag{8.17}$$

$$p(x_4|C_1) = \frac{1}{\sigma_{x_4C_1}\sqrt{2\pi}}\exp(\frac{-(x-\mu_{x_4C_1})^2}{2\sigma^2_{x_4C_1}}) = \frac{1}{0{,}0214\sqrt{2\pi}}\exp(\frac{-(x-0{,}153)^2}{2\cdot 0{,}0214^2}) = 0{,}884 \tag{8.18}$$

$$\begin{aligned} p(C_1|X) &= p(C_1)p(x_1|C_1)p(x_2|C_1)p(x_3|C_1)p(x_4|C_1) = \\ &= 0{,}6 \cdot 1{,}34 \cdot 10^{-3} \cdot 1{,}88 \cdot 10^{-3} \cdot 0{,}302 \cdot 0{,}884 = \\ &= 8{,}12 \cdot 10^{-8} = 3{,}845742252 \cdot 10^{-7} \end{aligned} \tag{8.19}$$

Ponieważ prawdopodobieństwo przynależności obiektu X jest większe dla klasy 1, klasyfikator Bayesa na podstawie jego cech przewiduje, że obiekt przynależy do tej klasy.



Rysunek 8.1: Rozkłady normalne o zróżnicowanych parametrach

8.2.2. Wielowymiarowy rozkład Gaussa

Wielowymiarowy rozkład Gaussa [16] jest uogólnieniem rozkładu normalnego na przypadki, gdzie badane obiekty posiadają więcej niż jedną cechę. Aby zobrazować działanie tej metody, najprościej zacząć od sytuacji jednowymiarowej. Gęstość prawdopodobieństwa jednowymiarowego rozkładu normalnego opisana jest wzorem (8.20):

$$\mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{8.20}$$

Funkcja jednej zmiennej opisująca gęstość prawdopodobieństwa (rys. 8.1) posiada dwa parametry: wartość oczekiwaną μ, określającą, gdzie funkcja będzie miała maksimum, i wariancję σ^2, opisującą, jak smukła jest krzywa funkcji.

Zasady panujące w przestrzeni jednowymiarowej mogą zostać uogólnione do jej n-wymiarowego odpowiednika przedstawionego we wzorze (8.21):

$$\mathcal{N}(X, \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{8.21}$$

Jak w przypadku jednowymiarowym, funkcja opisująca gęstość prawdopodobieństwa posiada dwa parametry, które mają identyczne znaczenia. Zasadniczą różnicę stanowi przejście z wartości skalarnych na macierze. W n-wymiarowej przestrzeni parametrami będą:

- X – wektor losowy o rozmiarze $n \times 1$;
- μ – wektor wartości oczekiwanych o rozmiarze $n \times 1$;
- Σ – macierz wariancji o rozmiarze $n \times$n.

Dysponując zestawem k obiektów, w celu wyznaczenia parametrów równania korzystamy z zasady maksimum wiarygodności [1]:

$$\mu = \frac{1}{k} \sum_{i=1}^{k} X_i \tag{8.22}$$

$$\Sigma = \frac{1}{k} \sum_{i=1}^{k} (X_i - \mu)(X_i - \mu)^T \tag{8.23}$$

Przykład

Zastosowanie wielowymiarowego rozkładu Gaussa można przedstawić na przykładzie opartym na zestawie danych podanych w tablicy 8.4 i obiekcie klasyfikowanym podanym w tablicy 8.5.

Tablica 8.4: Zestaw danych treningowych

Klasa	Cecha x_1	Cecha x_2
0	2,8	2,6
0	4,0	3,1
0	1,2	2,9
0	0,7	3,3
0	5,1	2,7
1	7,8	9,6
1	5,9	8,1
1	6,6	7,9
1	7,1	6,3
1	6,1	9,7

Należy zaznaczyć, że powyższe obiekty mają tylko dwie cechy, nie ze względu na ograniczenia metody, a dla bardziej czytelnej wizualizacji rezultatów.

Tablica 8.5: Obiekt testowy do klasyfikacji

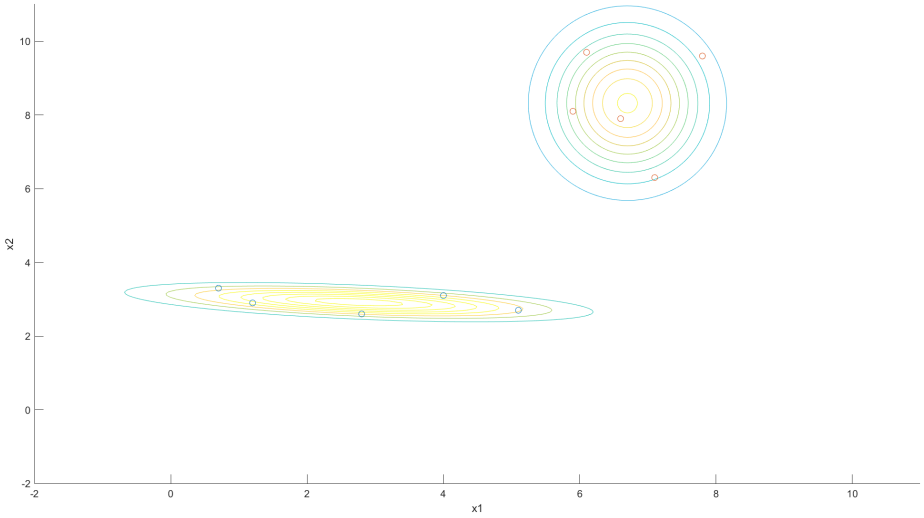
Klasa	Cecha x_1	Cecha x_2
?	3,2	2,7

W punkcie (8.24) zostaje użyty wzór (8.22) do wyznaczenia parametru μ dla klasy „0", natomiast w (8.25) zostaje użyty wzór (8.23) do obliczenia Σ dla tej samej klasy, przy wykorzystaniu już wyznaczonych wektorów $(X_i - \mu)$. Następnym krokiem jest powtórzenie powyższych czynności dla klasy „1".

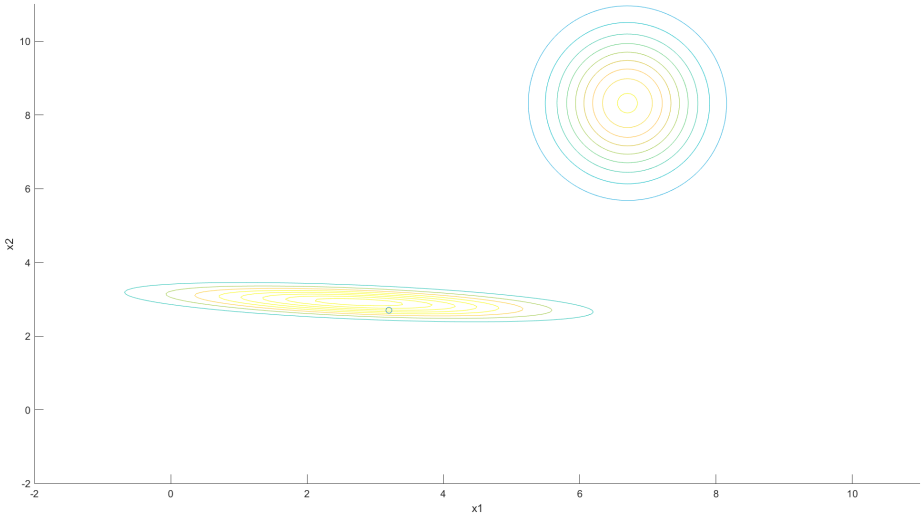
$$\mu_0 = \frac{1}{5}(\begin{bmatrix}2{,}8\\2{,}6\end{bmatrix} + \begin{bmatrix}4{,}0\\3{,}1\end{bmatrix} + \begin{bmatrix}1{,}2\\2{,}9\end{bmatrix} + \begin{bmatrix}0{,}7\\3{,}3\end{bmatrix} + \begin{bmatrix}5{,}1\\2{,}7\end{bmatrix}) = \frac{1}{5}\begin{bmatrix}13{,}8\\14{,}6\end{bmatrix} = \begin{bmatrix}2{,}76\\2{,}92\end{bmatrix} \tag{8.24}$$

$$\begin{aligned}
\Sigma_0 = \frac{1}{5}((\begin{bmatrix}0{,}06\\-0{,}32\end{bmatrix} \times \begin{bmatrix}0{,}06 & -0{,}32\end{bmatrix}) + (\begin{bmatrix}1{,}24\\0{,}18\end{bmatrix} \times \begin{bmatrix}1{,}24 & 0{,}18\end{bmatrix}) + \\
+(\begin{bmatrix}-1{,}56\\-0{,}02\end{bmatrix} \times \begin{bmatrix}-1{,}56 & -0{,}02\end{bmatrix}) + \\
+(\begin{bmatrix}-2{,}06\\0{,}38\end{bmatrix} \times \begin{bmatrix}-2{,}06 & 0{,}38\end{bmatrix}) + (\begin{bmatrix}2{,}34\\-0{,}22\end{bmatrix} \times \begin{bmatrix}2{,}34 & -0{,}22\end{bmatrix})) = \\
= \frac{1}{5}(\begin{bmatrix}0{,}0016 & -0{,}0128\\-0{,}0128 & 0{,}1024\end{bmatrix} + \begin{bmatrix}1{,}5376 & 0{,}2232\\0{,}2232 & 0{,}0324\end{bmatrix} + \\
+ \begin{bmatrix}2{,}4336 & 0{,}0312\\0{,}0312 & 0{,}0004\end{bmatrix} + \begin{bmatrix}4{,}2436 & -0{,}7828\\-0{,}7828 & 0{,}1444\end{bmatrix} + \begin{bmatrix}5{,}4756 & -0{,}5148\\-0{,}5148 & 0{,}0484\end{bmatrix}) = \\
= \frac{1}{5}(\begin{bmatrix}13{,}692 & -1{,}056\\-1{,}056 & 0{,}328\end{bmatrix} = \begin{bmatrix}2{,}7384 & -0{,}2112\\-0{,}2112 & 0{,}0656\end{bmatrix}
\end{aligned} \tag{8.25}$$

Mając obliczoną wartość średnią oraz wariancję, możemy zobaczyć, jak rozkłada się gęstość prawdopodobieństwa przynależności do badanej klasy lub samo prawdopodobieństwo, z jakim obiekt do niej należy. Aby je otrzymać, wystarczy do równania (8.21) podstawić μ, Σ oraz wektor X, zawierający cechy obiektu do klasyfikacji, co zostanie obliczone w następnym podpunkcie. Graficzny przykład rozkładu prawdopodobieństw dla danych z tablicy 8.4 pokazano na rys. 8.2.



Rysunek 8.2: Rozkład normalny stworzony na podstawie zestawu danych treningowych, obiekty klasy „0" reprezentowane są niebieskimi punktami, natomiast klasy „1" pomarańczowymi



Rysunek 8.3: Obiekt testowy na tle gęstości prawdopodobieństwa

$$\mathcal{N}_0(\begin{bmatrix}3{,}2\\2{,}7\end{bmatrix}, \begin{bmatrix}2{,}76\\2,92\end{bmatrix}, \begin{bmatrix}2{,}7384 & -0{,}2112\\-0{,}2112 & 0{,}0656\end{bmatrix}) = \frac{1}{(2\pi)^1|\begin{bmatrix}2{,}7384 & -0{,}2112\\-0{,}2112 & 0{,}0656\end{bmatrix}|^{1/2}} \cdot e^{-\frac{1}{2}\left(\begin{bmatrix}3{,}2\\2{,}7\end{bmatrix}-\begin{bmatrix}2{,}76\\2{,}92\end{bmatrix}\right)^T \begin{bmatrix}2{,}7384 & -0{,}2112\\-0{,}2112 & 0{,}0656\end{bmatrix}^{-1} \left(\begin{bmatrix}3{,}2\\2{,}7\end{bmatrix}-\begin{bmatrix}2{,}76\\2{,}92\end{bmatrix}\right)} = 0{,}294 \tag{8.26}$$

$$\mathcal{N}_1(\begin{bmatrix}3,2\\2,7\end{bmatrix}, \begin{bmatrix}6,7\\8,32\end{bmatrix}, \begin{bmatrix}0,476 & -0,002\\-0,002 & 1,569\end{bmatrix} = 1,924 \cdot 10^{-11} \tag{8.27}$$

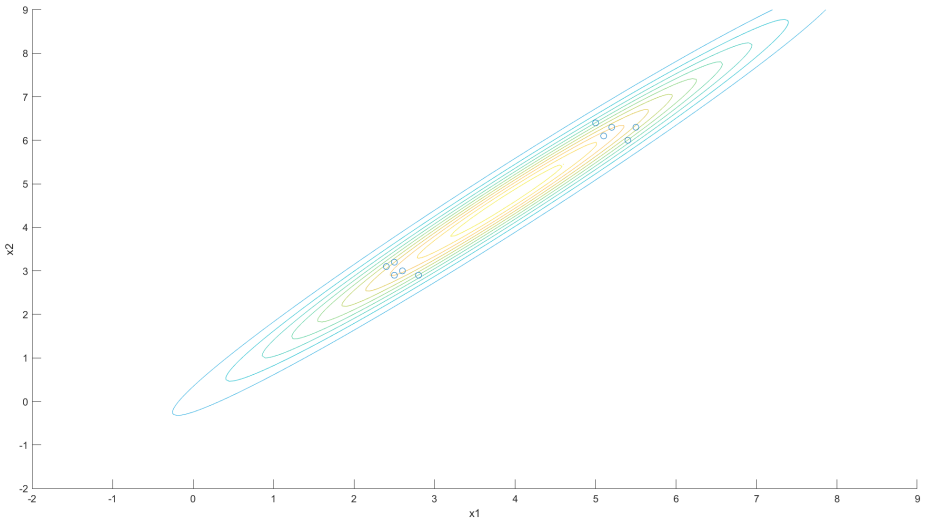
Powyższe wyniki należy interpretować jako prawdopodobieństwa, z którymi obiekt testowy należy do danej klasy. W równaniu (8.26) rezultat określa prawdopodobieństwo przynależności do klasy „0", natomiast w (8.27) do klasy „1". Obie liczby różnią się znacznie, więc z dużą dozą pewności można na tej podstawie stwierdzić że obiekt należy do klasy „1". Jak widać na rys. 8.3, graficzna interpretacja również potwierdza powyższy wniosek.

8.3. Generatywne metody nienadzorowane

Drugi typ metod – uczenie nienadzorowane – nie korzysta z etykiet. Podobnie jak w metodzie nadzorowanej, algorytm uczy się na danych wejściowych związków między punktami danych, lecz nie posiada informacji o powiązaniach obiekt–klasa. Metody tego typu pozwalają na wyznaczenie grup obiektów o podobnych cechach.

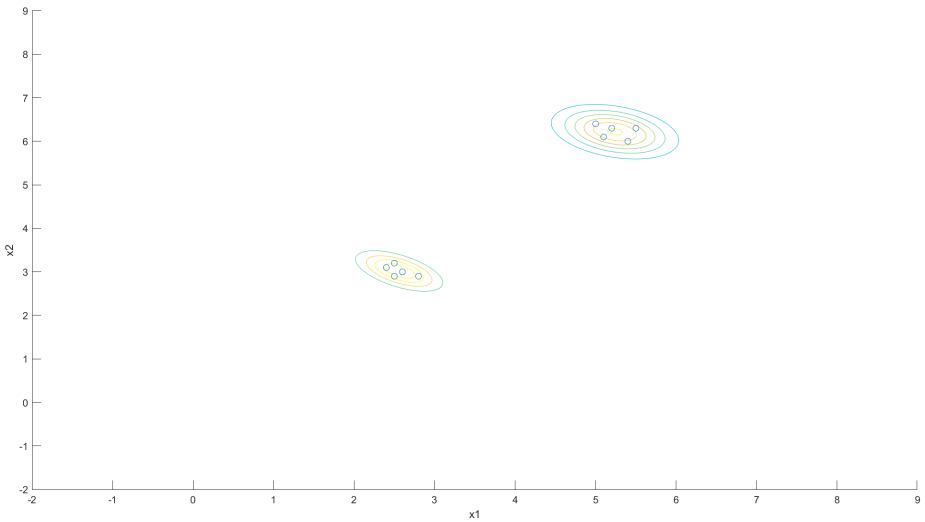
8.3.1. *Gaussian mixture model*

Wielowymiarowy rozkład Gaussa może być przydatny do opisu wielu klas, jednak w rzeczywistości zestawy danych formują się w różne układy i nie zawsze tworzą prosty rozkład normalny. Jedną z wielu takich sytuacji przedstawiono na rys. 8.4.



Rysunek 8.4: Rozwiązanie za pomocą jednego rozkładu Gaussa

Jak widać, wartość oczekiwana znajduje się pomiędzy dwoma klastrami obiektów, co jest sytuacją trudną do uchwycenia z użyciem modelu bazującego na jednym rozkładzie. Rozwiązaniem tego problemu może być opisanie powyższego modelu dwoma rozkładami normalnymi [14]. Aby uzyskać wspomniany efekt, należy podzielić obiekty na dwie grupy i obliczyć parametry rozkładu dla każdej z osobna (rys. 8.5).



Rysunek 8.5: Rozwiązanie za pomocą dwóch rozkładów Gaussa

Dodatkowo należy pamiętać o podstawowym aksjomacie rachunku prawdopodobieństwa przedstawionym wzorem (8.28).

$$P(\Omega) = 1 \tag{8.28}$$

Aby równanie (8.28) było spełnione, wystarczy przemnożyć każde równanie rozkładu Gaussa przez pewien parametr. Wynika z tego, że jeśli się chce opisać zestaw danych k równaniami, dodatkowo dochodzi k parametrów ϕ_k, spełniających warunek opisany równaniem (8.29):

$$\sum_{i=1}^{k} \phi_i = 1 \tag{8.29}$$

Podsumowując wszystkie wspomniane wiadomości, można zapisać ostateczny wzór dla metody *Gaussian mixture model* (8.30):

$$\mathcal{G}(X, \phi_n) = \sum_{i=1}^{k} \phi_i \mathcal{N}(X, \mu_i, \Sigma_i) \tag{8.30}$$

oraz jego pełniejszą wersję opisaną wzorem (8.31):

$$\mathcal{G}(X, \phi_n) = \sum_{i=1}^{k} \phi_i \frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)} \tag{8.31}$$

Algorytm *expectation–maximization*

Z uwagi na to, że GMM jest metodą nienadzorowaną, zestaw danych wejściowych nie posiada etykiet, co uniemożliwia obliczenie optymalnych parametrów modelu poprzez optymalizację funkcji ryzyka empirycznego. Do wyznaczenia etykiet maksymalizujących podobieństwo obiektów w wybranym obszarze wykorzystany może zostać algorytm *expectation–maximization* [12], [13]. Składa się on z dwóch kroków: *expectation* (E) – obliczenia parametrów dla obecnego podziału obiektów i *maximization* (M) – przypisania obiektów do aktualnie najbardziej pasującej grupy.

W kroku E wyznaczamy parametry zgodnie z równaniami opisanymi w podrozdziale 8.2.2 (wielowymiarowy rozkład Gaussa), dodatkowy parametr ϕ, który nie jest opisany we wspomnianym podrozdziale, obliczamy według wzoru (8.32), po zakończeniu głównej pętli programu.

$$\phi_i = \frac{\text{liczba obiektów w grupie } i}{\text{liczba wszystkich obiektów}} \tag{8.32}$$

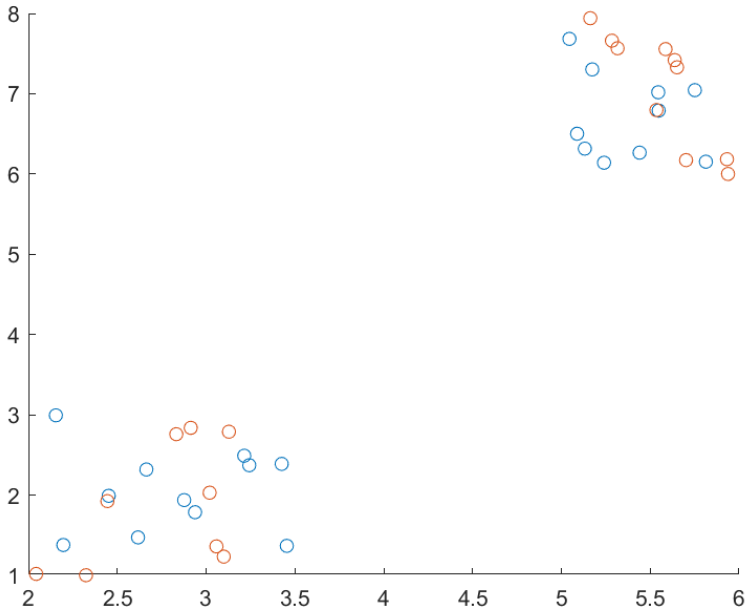
W kroku M, korzystając z nowo obliczonych parametrów, dla każdego z obiektów zostaje sprawdzone, do którego równania najlepiej pasują. Upraszczając, każdy osobny obiekt zostaje podstawiony do każdego z równań i jeśli prawdopodobieństwo przynależności jest większe dla grupy, do której obecnie obiekt nie

należy, zostaje on tam przeniesiony poprzez zmianę etykiety. W innym wypadku jego etykieta pozostaje bez zmian.

Ostatnią kwestią pozostaje warunek stopu. Najczęściej spotykanym rozwiązaniem jest zatrzymanie algorytmu w momencie, gdy podczas kolejnej iteracji nie następuje żadna zmiana w kroku M. Jednak gdy czas wykonywania jest ważniejszy niż poprawność klasteryzacji, można przyjąć warunek stopu jako maksymalną liczbę iteracji.

Przykład

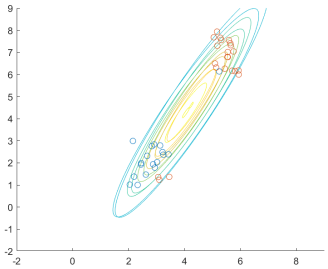
Aby zobrazować przebieg algorytmu E-M, zostanie wykorzystany zestaw danych tworzący dwie wyraźne grupy w przestrzeni dwuwymiarowej przedstawione na rys. 8.6. Początkowy przydział do grup zostanie dobrany losowo, jednak w celu optymalizacji często sytuację początkową dobiera się z wykorzystaniem np. algorytmu k-średnich [15].



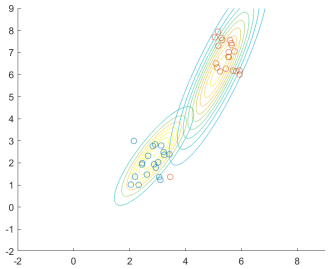
Rysunek 8.6: Rozmieszczenie punktów, przynależność do grup symbolizują kolory znaczników

Po wykonaniu początkowych kroków można przejść do głównej pętli programu. Kolejne wykresy (rys. 8.7–8.10) oznaczają sytuacje występujące po następnych iteracjach algorytmu.

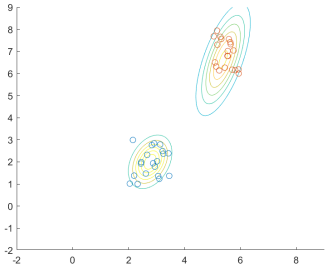
Jak widać, w czterech iteracjach algorytm był w stanie poprawnie wykryć



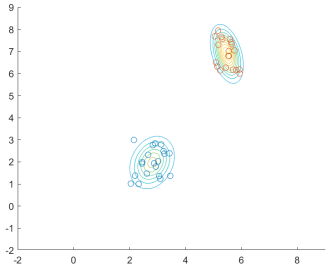
Rysunek 8.7: Pierwsza iteracja



Rysunek 8.8: Druga iteracja



Rysunek 8.9: Trzecia iteracja



Rysunek 8.10: Czwarta iteracja

i wyznaczyć rozkłady prawdopodobieństwa dla obu grup. Należy jednak zaznaczyć, że wybór punktów w przestrzeni był dopasowany do wizualizacji rezultatów. W przypadku pracy na danych rzeczywistych iteracji potrzebnych do uzyskania zadowalających wyników będzie znacznie więcej, a sam rozkład może okazać się nie tak przejrzysty jak na pokazanych wykresach. Ostatnim etapem algorytmu jest wyznaczenie parametrów ϕ. Wiedząc, że w obu grupach znajduje się po 20 obiektów, otrzymujemy:
$\phi_1 = \frac{20}{40} = 0{,}5$ i $\phi_2 = \frac{20}{40} = 0{,}5$.

Znając wszystkie parametry, można zapisać ostateczne równanie (8.33) opisujące badany model:

$$\begin{aligned} \mathcal{G}(X, \begin{bmatrix}0{,}5 & 0{,}5\end{bmatrix}) = 0{,}5\mathcal{N}_1(X, \begin{bmatrix}2{,}80\\1{,}97\end{bmatrix}, \begin{bmatrix}0{,}17 & 0{,}05\\0{,}05 & 0{,}37\end{bmatrix}) \\ +0{,}5\mathcal{N}_2(X, \begin{bmatrix}5{,}47\\6{,}89\end{bmatrix}, \begin{bmatrix}0{,}07 & -0{,}06\\-0{,}06 & 0{,}38\end{bmatrix}) \end{aligned} \tag{8.33}$$

Często spotykanym błędem podczas nauki działania *Gaussian mixture model* jest traktowanie go jako zwykłego wielowymiarowego rozkładu normalnego. Warto przeanalizować wykresy na rys. 8.2 oraz 8.5. Na obu wykresach widać dwa rozkłady normalne, jednak wykres z podrozdziału 8.2.2 przedstawia po jednym rozkładzie dla jednej klasy, natomiast jego odpowiednik z GMM ukazuje

co prawda dwa rozkłady, ale nie dla dwóch klas, lecz dla jednej. Na tym polega główna różnica pomiędzy dwiema omawianymi metodami.

Tablica 8.6: Wartości parametrów rozkładu przy kolejnych iteracjach

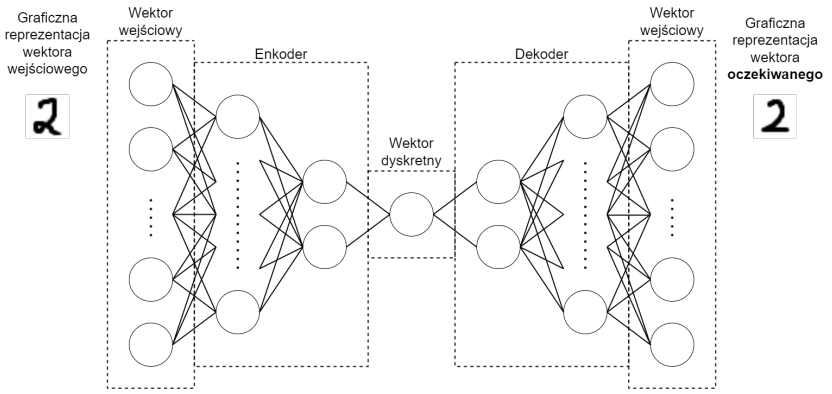
Iteracja	μ_1	Σ_1	μ_2	Σ_2
1	$\begin{bmatrix}3{,}72\\3{,}82\end{bmatrix}$	$\begin{bmatrix}2{,}17 & 3{,}58\\3{,}58 & 6{,}84\end{bmatrix}$	$\begin{bmatrix}4{,}33\\5{,}11\end{bmatrix}$	$\begin{bmatrix}2{,}04 & 3{,}47\\3{,}47 & 6{,}23\end{bmatrix}$
2	$\begin{bmatrix}2{,}98\\2{,}44\end{bmatrix}$	$\begin{bmatrix}1{,}03 & 1{,}23\\1{,}23 & 2{,}03\end{bmatrix}$	$\begin{bmatrix}5{,}12\\6{,}60\end{bmatrix}$	$\begin{bmatrix}1{,}04 & 1{,}70\\1{,}70 & 3{,}01\end{bmatrix}$
3	$\begin{bmatrix}2{,}67\\2{,}07\end{bmatrix}$	$\begin{bmatrix}0{,}20 & 0{,}01\\0{,}01 & 0{,}27\end{bmatrix}$	$\begin{bmatrix}5{,}53\\7{,}18\end{bmatrix}$	$\begin{bmatrix}0{,}08 & 0{,}07\\0{,}07 & 0{,}32\end{bmatrix}$
4	$\begin{bmatrix}2{,}67\\2{,}07\end{bmatrix}$	$\begin{bmatrix}0{,}19 & 0{,}01\\0{,}01 & 0{,}28\end{bmatrix}$	$\begin{bmatrix}5{,}53\\7{,}18\end{bmatrix}$	$\begin{bmatrix}0{,}08 & 0{,}07\\0{,}07 & 0{,}32\end{bmatrix}$

8.3.2. Autoenkoder wariacyjny

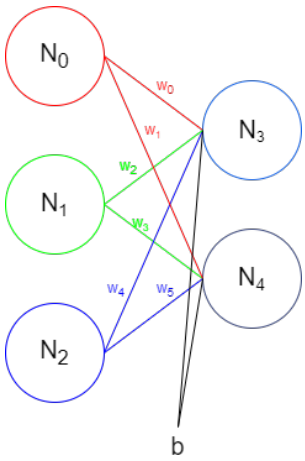
Autoenkoder [2] jest modelem sieci neuronowej, w którym typowo liczba neuronów wejścia i wyjścia jest taka sama, a sama struktura jest symetryczna, składająca się z bloków kodującego i dekodującego. Dzięki temu model ma możliwość wykonania rekonstrukcji danych dostarczonych na wejściu. W środku sieci znajduje się wektor krótszy niż wektor wejściowy, realizujący tzw. *bottle neck* skutkujący redukcją wymiarowości danych wejściowych.

Autoenkodery oraz ich warianty mają szeroki wachlarz zastosowań: tłumaczenie języków obcych w czasie rzeczywistym [4], wykrywanie anomalii w danych [5] czy usuwanie szumów z obrazów [17]. W tym podrozdziale omówiona zostanie zasada działania odmiany autoenkodera, który pozwala na generację nowych danych na podstawie zestawu trenującego, czyli autoenkodera wariacyjnego (*variational autoencoder*, VAE) [10].

Aby przedstawić działanie VAE, najpierw zostanie wyjaśnione, jak działa zwykły autoenkoder. Za przykład posłuży klasyfikacja obrazów cyfr (0–9), które mają wymiar 28×28 pikseli, przy czym każdy z nich przyjmuje wartość od 0 (kolor biały) do 1 (kolor czarny). Taki obraz można przedstawić w formie wektora o długości 784 elementów (28*28), gdzie każdy element wektora odpowiada wartości koloru każdego poszczególnego piksela, co przedstawiono na rys. 8.11.



Rysunek 8.11: Przykładowa architektura sieci typu autoenkoder



Rysunek 8.12: Uzyskiwanie nowych wag na podstawie prostej sieci neuronowej

W autoenkoderze, tak jak w klasycznej sieci neuronowej, każde połączenie między neuronami posiada wagi, które na początku treningu sieci ustawiane są na losowe wartości. Wartości neuronów w kolejnych warstwach są równe sumie iloczynów poszczególnych wartości neuronów oraz wag z odpowiedniego połączenia plus dodatkowo stronniczość (*bias*). Poniższy przykład zobrazowany został na rys. 8.12.

Dla danych:

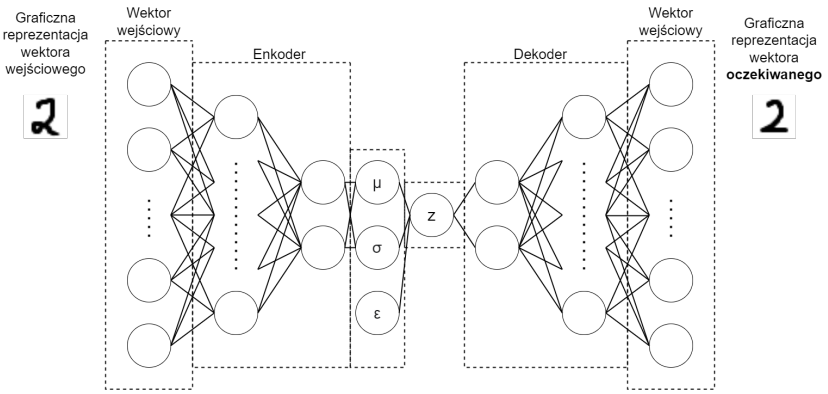
$$\begin{array}{lll} N_0 = 0{,}9 & w_0 = 0{,}1 & w_1 = 0{,}25 \\ N_1 = 0{,}5 & w_2 = 0{,}4 & w_3 = 0{,}3 \\ N_2 = 0{,}25 & w_4 = 0{,}9 & w_5 = 0{,}5 \\ b = 0{,}25 & & \end{array}$$

wartości N_3 oraz N_4 można obliczyć, stosując poniższe równania:

$$N_3 = N_0 * w_0 + N_1 * w_2 + N_2 * w_4 + b \tag{8.34}$$

$$N_4 = N_0 * w_1 + N_1 * w_3 + N_2 * w_5 + b \tag{8.35}$$

Aby uzyskać ostateczne wartości odpowiedzi neuronów w kolejnych warstwach, należy użyć funkcji aktywacji, np. sigmoidalnej $N_n = \frac{1}{1+e^{-N_n}}$, gdzie n jest indeksem odpowiedniego neuronu, dla przykładu wartość neuronu N_3 wyniesie $\frac{1}{1+e^{0,9*0,1+0,5*0,4+0,25}} \approx 0,37$. Na rys. 8.11 przedstawiono architekturę VA, cały opisany proces jest działaniem enkodera oraz dekodera, pośrodku całego układu znajduje się zredukowany wektor. Po nauczeniu sieci w tym miejscu można wpisać dowolny wektor o odpowiednim rozmiarze, jednakże jedyne, co można tu uzyskać, to klasyfikacja obiektu ze względu na fakt, że testowany jest jedynie pojedynczy obiekt. Aby wygenerować nowy obiekt, zamiast jasno ustalonego wektora w środku układu wybierany jest rozkład jego parametrów; w rezultacie schemat sieci wygląda nieco inaczej, tak jak przedstawiono to na rys. 8.13.

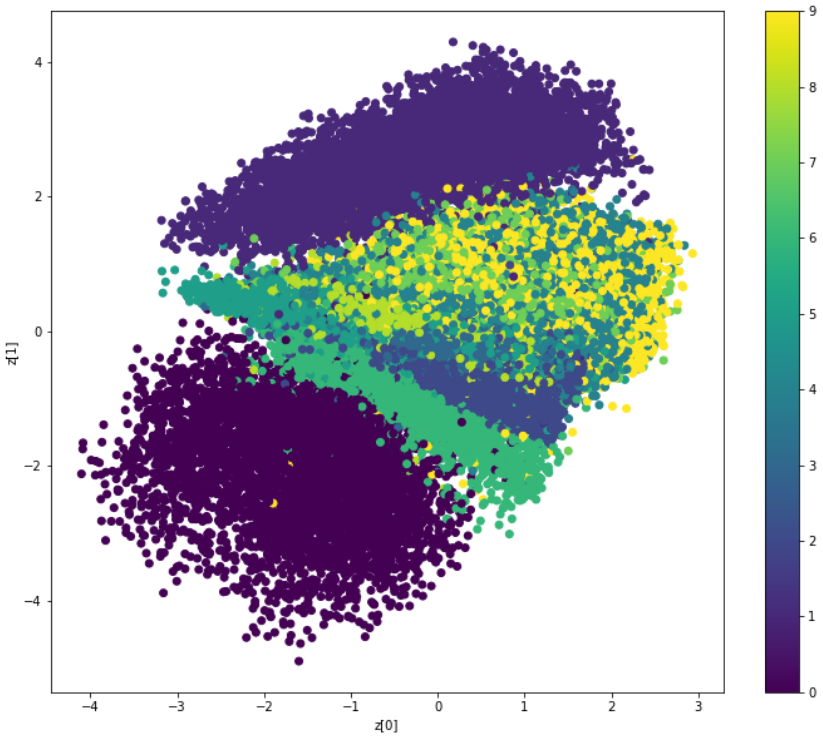


Rysunek 8.13: Schemat VAE

W przeciwieństwie do klasycznego autoenkodera pośrodku układu zostanie zastosowany tzw. trik reparametryzacji (*reparametrization trick*) [7]. Pozwala on na zmianę dekodowanego wektora z formy deterministycznej do formy losowej poprzez pozostawienie w nim części deterministycznej w celu uzyskania możliwości wygenerowania i wykrywania obiektów ze spektrum dyskretnego parametrów obiektów. μ oraz σ są parametrami rozkładu normalnego elementów znajdujących się w warstwie enkodera, na podstawie sumy średnich wartości oraz losowo wybranego odchylenia standardowego dla każdego elementu tworzymy jawny wektor z określony wzorem (8.36):

$$z = \mu + \sigma \odot \epsilon \tag{8.36}$$

gdzie:
μ – wektor uśrednionych wartości parametrów zredukowanego wektora;
$\sigma \odot \epsilon$ – wektor odchyleń standardowych danych parametrów przemnożonych przez losową wartość z standardowego rozkładu normalnego $\mathcal{N}(0, 1)$.



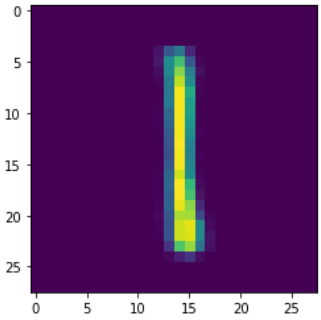
Rysunek 8.14: Rozkład klas obiektów

Dzięki faktowi, że trenujemy układ za pomocą dystrybuanty wartości parametrów, można nauczyć sieć wytwarzać nowe, zbliżone, ale całkowicie sztuczne obiekty.

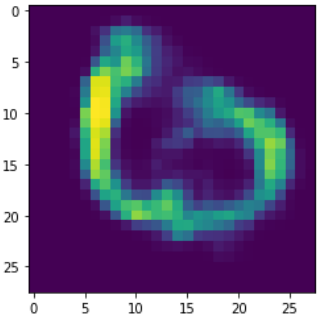
We wcześniej wspomnianym przykładzie tworzenia nowych obrazów ręcznie pisanych cyfr dla lepszej wizualizacji wektor z posiada dwa parametry.

Rys. 8.14 przedstawia rozkład wygenerowanych obrazów cyfr. Na rys. 8.15 i 8.16 widoczne są przykładowe obiekty. Pierwszy z nich (rys. 8.15) jest w klastrze jedynek, dzięki czemu kształtem wyraźnie przypomina cyfrę 1. Drugi obiekt (rys. 8.16) nie leży w pobliżu żadnego klastra, przez co jego kształt wyglądem nie przypomina żadnej cyfry.

Klasyczny autoenkoder jest uczony algorytmem propagacji wstecznej, co typowo umożliwia wykrywanie danej klasy obiektów, tak jak już wcześniej wspomniano, przez fakt, że wybierany jest jeden wektor. Dzięki trikowi reparametryzacji można przeprowadzić trening VAE do tworzenia nowych obiektów, z tą różnicą, że błąd między wektorami wyjściowym a uczącym jest obliczany z uży-



Rysunek 8.15: $z = [1, 4]^T$, jest w klastrze jedynek i wyraźnie widać, że jest to cyfra 1



Rysunek 8.16: $z = [2, -4]^T$, obszar, w którym nie ma żadnego klastra, wyglądem najbardziej przypomina 0

ciem dywergencji Kullbacka–Leiblera [9], która określa, jak bardzo dany rozkład odbiega od standardowego rozkładu normalnego porównywanego obiektu:

$$\sum_{i=1}^{n} p_\theta(i) \log_2 \frac{p_\theta(i)}{p_\omega(i)} \tag{8.37}$$

gdzie:
n – liczba parametrów nauczanego wektora;
$p_\theta(i)$ – prawdopodobieństwo modelu trenującego;
$p_\omega(i)$ – prawdopodobieństwo modelu uczonego.

Przyjmijmy przykład dwóch wektorów – wektora wyjściowego $p_\omega = \begin{bmatrix} 0{,}35 \\ 0{,}56 \end{bmatrix}$ oraz wektora trenującego $p_\theta = \begin{bmatrix} 0{,}28 \\ 0{,}72 \end{bmatrix}$. Korzystając ze wzoru (8.37), możemy obliczyć dywerencję Kullbacka–Leiblera, podstawiając kolejne wartości z obu wektorów:

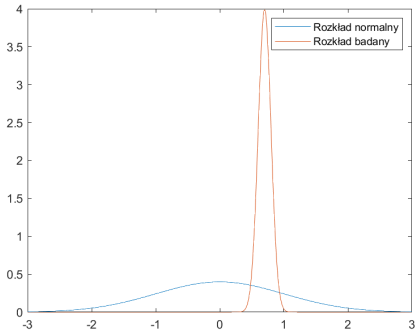
$$\sum_{i=1}^{n} p_\theta(i) \log_2 \frac{p_\theta(i)}{p_\omega(i)} = 0{,}28 * \log_2 \frac{0{,}28}{0{,}35} + 0{,}72 * \log_2 \frac{0{,}72}{0{,}56} \approx 0{,}17 \tag{8.38}$$

W przypadku VAE przyrównywany jest wektor z (a konkretniej jego parametry rozkładu normalnego) ze standardowym rozkładem normalnym $\mathcal{N}(0, 1)$:

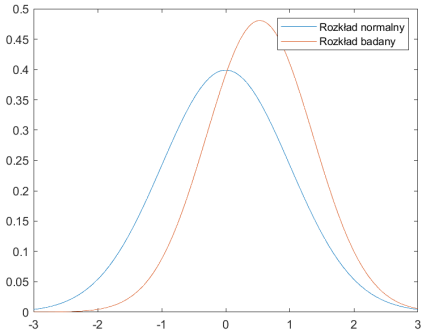
$$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - log(\sigma_i) - 1 \tag{8.39}$$

gdzie:
n – liczba parametrów nauczanego wektora;
σ_i – i-te odchylenie standardowe parametru;
μ_i – i-ta średnia wartość parametru.

Wpływ zmiennych na kształt rozkładów badanego i normalnego jest widoczny na rys. 8.17 i 8.18. Przedstawiają one wykresy rozkładów dla różnych wartości średniej, odchylenia standardowego i dywergencji Kullbacka–Leiblera.



Rysunek 8.17: $\sigma = 0{,}7$, $\mu = 0{,}1$, dywergencja K–L = 0,5



Rysunek 8.18: $\sigma = 0{,}53$, $\mu = 0{,}83$, dywergencja K–L = 0,05

Jak widać na rys. 8.17 i 8.18, im bardziej zbliżony jest rozkład wektora zakodowanego do rozkładu normalnego, tym mniejsza jest dywergencja, stanowiąca element funkcji celu treningu sieci neuronowej.

8.4. Podsumowanie

W rozdziale przedstawiono zastosowanie modeli generatywnych do zadań kategoryzacji. Na wstępie omówiono podział na dwa zadania uczenia nadzorowanego i nienadzorowanego a następnie opisano podstawowy generatywny algorytm nadzorowany – naiwny klasyfikator Bayesa. Przykład ten opisuje podstawowe zasady działania metod nadzorowanych – naukę rozkładów klas – oraz służy do zobrazowania, jak klasyfikatory generatywne określają przynależność nowego obiektu do klasy. Dzięki szybkiemu działaniu i prostej implementacji w przypadkach, gdy cechy klas są niezależne, w praktyce algorytm Bayesa jest często stosowany w klasyfikatorach związanych z tekstem, takich jak filtry antyspamowe. Kolejny podrozdział przedstawił bardziej złożony model – wielowymiarowy rozkład Gaussa. Korzysta on z własności odkrytej przez niemieckiego naukowca Carla Friedricha Gaussa, mówiącej, że każda wielkość, która jest sumą wielu niezależnych czynników, tworzy rozkład zbliżony do krzywej dzwonowej, potocznie nazywanej rozkładem normalnym. Przykład liczbowy przedstawiony w tym rozdziale pokazał, że mając zestaw danych testowych, jesteśmy w stanie stworzyć jego model probabilistyczny oparty na rozkładzie normalnym, co pozwala nam z dużym prawdopodobieństwem poprawnie skojarzyć nowo dodany punkt z klasą.

W drugiej części rozdziału opisano modele nienadzorowane. W pierwszej sekcji opisano *Gaussian mixture model* – rozszerzenie wcześniej omawianego wielowymiarowego rozkładu Gaussa; ich zastosowania praktyczne są bardzo podobne, jednak GMM pozwala na o wiele lepsze przybliżenie rozkładu do sytuacji rzeczywistej poprzez rozbicie go na wiele rozkładów normalnych. Wadą tej metody jest brak możliwości analitycznego wyznaczenia parametrów równania. Dlatego został również opisany algorytm *expectation–maximization* (EM) pozwalający na iteracyjne przybliżenie tych wartości. Zaznaczyć należy, że algorytm EM jest kosztowny obliczeniowo, a gdy zdecydujemy się na ograniczenie liczby iteracji, uzyskane wyniki mogą być niezadowalające. Dlatego bardzo ważne są analiza danych wejściowych i decyzja, z której metody należy skorzystać: szybszej, ale mniej dokładnej (wielowymiarowy rozkład Gaussa), czy wolniejszej, lecz dokładniejszej (GMM).

Ostatnia część rozdziału została poświęcona autoenkoderowi wariacyjnemu, czyli wariantowi sieci neuronowej typu autoasocjacyjnego. Zdecydowanymi zaletami tego rozwiązania są szeroki wachlarz zastosowań, redukcja szumów, tworzenie nowych obiektów, interpolacja czy usuwanie znaków wodnych. Implementacja jest stosunkowo prosta, między innymi dzięki bibliotekom języka Python Keras, oraz Tensorflow. Wadą autoenkodera wariacyjnego jest, w porównaniu z innymi metodami (np. GAN [6]), generowanie mniej dokładnych wyników. VAE jest przydatny do generowania danych do nauki algorytmu uczenia maszynowego, nie jako generator sam w sobie. Dla przykładu obrazy wygenerowane przez VAE będą rozmyte, w porównaniu z obrazami wygenerowanymi przez GAN.

Bibliografia

[1] Anderson T.W. *Maximum likelihood estimates for a multivariate normal distribution when some observations are missing. Journal of the american Statistical Association*, 52(278):200–203, 1957.

[2] Bank D., Koenigstein N., Giryes R. *Autoencoders. arXiv preprint arXiv:2003.05991*, 2020.

[3] Berrar D. *Bayes' theorem and naive Bayes classifier. Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, 403, 2018.

[4] Chandar AP S., et al. *An autoencoder approach to learning bilingual word representations. Advances in neural information processing systems*, 27, 2014.

[5] Chen Z., et al. *Autoencoder-based network anomaly detection.* In *2018 Wireless telecommunications symposium (WTS)*, pp. 1–5. IEEE, 2018.

[6] Creswell A., et al. *Generative adversarial networks: An overview. IEEE signal processing magazine*, 35(1):53–65, 2018.

[7] Fabius O., Van Amersfoort J.R. *Variational recurrent auto-encoders. arXiv preprint arXiv:1412.6581*, 2014.

[8] Jebara T. *Machine learning: discriminative and generative*, vol. 755. Springer Science & Business Media, 2012.

[9] Joyce J.M. *Kullback-leibler divergence.* In *International encyclopedia of statistical science*, pp. 720–722. Springer, 2011.

[10] Kingma D.P., Welling M., et al. *An introduction to variational autoencoders. Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[11] Leung K.M. *Naive bayesian classifier. Polytechnic University Department of Computer Science/Finance and Risk Engineering*, 2007:123–156, 2007.

[12] Moon T.K. *The expectation-maximization algorithm. IEEE Signal processing magazine*, 13(6):47–60, 1996.

[13] Ng S.K., Krishnan T., McLachlan G.J. *The EM algorithm.* In *Handbook of computational statistics*, pp. 139–172. Springer, 2012.

[14] Reynolds D.A. *Gaussian mixture models. Encyclopedia of biometrics*, 741(659-663), 2009.

[15] Sinaga K.P., Yang M.S. *Unsupervised K-means clustering algorithm. IEEE access*, 8:80716–80727, 2020.

[16] Tong Y.L. *The multivariate normal distribution.* Springer Science & Business Media, 2012.

[17] Yasenko L., Klyatchenko Y., Tarasenko-Klyatchenko O. *Image noise reduction by denoising autoencoder.* In *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pp. 351–355. IEEE, 2020.

Rozdział 9

Przegląd systemów rekomendacyjnych: techniki, zastosowania, zalety, ograniczenia

Karolina Selwon[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
karolina.sala@pg.edu.pl

Abstrakt

Systemy rekomendacyjne powstały w celu rozwiązywania problemów wynikających z dynamicznego generowania informacji przez portale internetowe. Stosując techniki analizy danych, dostarczają użytkownikom spersonalizowane treści i usługi. W rozdziale przedstawiono charakterystykę, zastosowania, zalety i ograniczenia różnych metod rekomendacji.

Słowa kluczowe: filtrowanie grupowe, filtrowanie oparte na treści, filtrowanie hybrydowe, systemy rekomendacji

9.1. Wstęp

Przedsiębiorstwa prowadzące działalność z wykorzystaniem kanałów online gromadzą coraz więcej danych od swoich klientów. Natomiast klienci stają przed wyborem produktów i usług spośród szerokiej oferty wielu dostawców. Motywacją systemu rekomendacji jest rozwiązanie problemu przeciążenia ilością informacji. System ma na celu dostarczanie najważniejszych i odfiltrowanie nieistotnych z punktu widzenia konkretnego użytkownika. Wykorzystując dane historyczne dotyczące preferencji użytkowników, system stosuje techniki filtrowania danych w celu wskazania rekomendowanych elementów.

Systemy rekomendacyjne wspomagają proces podejmowania decyzji. Znalazły zastosowanie w sektorze biznesowym. W przypadku serwisów internetowych e-commerce są skutecznym sposobem sprzedaży coraz większej liczby produktów poprzez możliwość dotarcia ze spersonalizowaną ofertą do klientów. Inne szerokie zastosowania istnieją dla serwisów do przeszukiwania katalogów treści.

Tradycyjne podejścia konstruowania systemów rekomendacji obejmują filtrowanie oparte na treści oraz filtrowanie zespołowe. Podejście oparte na treści bazuje na opisach produktów i profilu preferowanych wyborów użytkownika. Natomiast metody filtrowania zespołowego wykorzystują dane o aktywności wielu użytkowników oraz podobieństwa i różnice między profilami użytkowników. Obie metody zostały skutecznie zaimplementowane w różnych domenach, jednak w wielu sytuacjach nie sprawdzają się dobrze ze względu na pewne charakterystyczne ograniczenia. Z tego względu powstały metody hybrydowe, łączące obie techniki. Łączenie wielu podejść pozwala wykorzystać ich mocne strony, jednocześnie niwelując wady.

Cele tego rozdziału obejmują dokonanie przeglądu klasycznych metod w tej dziedzinie, zidentyfikowanie ograniczeń oraz wskazanie kierunków zastosowań.

9.2. Rodzaje systemów rekomendacyjnych

System rekomendacji uzyskuje informacje o preferencjach użytkownika poprzez analizę historii jego zachowań, a następnie wykonuje modelowanie spersonalizowanych poleceń przedmiotów. Na przykładzie sklepów internetowych model może uczyć się preferencji klientów na podstawie danych jawnych, czyli wprowadzonych bezpośrednio przez użytkownika, takich jak oceny produktów, oraz danych niejawnych, takich jak historia przeglądania, zawartość koszyka zakupowego, transakcje itp., które można pozyskać poprzez narzędzia do śledzenia aktywności użytkowników portalu. Ze względu na rodzaj danych oraz wymagania systemu można zdefiniować podział metod budowy systemu rekomendacyjnego, które dzielą się na trzy główne typy: filtrowanie oparte na treści (*content-based filtering*), filtrowanie zespołowe (*collaborative filtering*) i systemy hybrydowe (*hybrid recommendation systems*) [1].

9.2.1. Filtrowanie oparte na treści

Technika oparta na treści jest algorytmem zależnym od domeny, a wyniki przewidywań są zależne od analizy atrybutów produktu. Metoda dostarcza rekomendacje dokonywane na podstawie profili użytkowników z wykorzystaniem cech wyodrębnionych z opisu elementów, które użytkownik oceniał w przeszłości. Generowane są profil produktu i profil preferowanych wyborów użytkownika. Podczas generowania profilu produktu analizowane są jego cechy, które mogą być ustrukturyzowanymi lub nieustrukturyzowanymi danymi. Zazwyczaj dane o produktach są podawane z góry. Na etapie generowania profilu użytkownika modelowane są jego preferencje na podstawie wcześniej wybranych przez użytkownika elementów [8]. Algorytm rekomenduje produkty podobne do tych, które użytkownik wybrał w przeszłości. Idea filtrowania opartego na treści obejmuje trzy główne etapy:

- **profil produktu** – wyodrębnienie cech produktu w celu utworzenia jego ustrukturyzowanej reprezentacji;
- **profil użytkownika** – tworzony na podstawie wybranych produktów. Modelowane są preferencje, czyli funkcja, która dla dowolnego elementu wyznacza prawdopodobieństwo, że użytkownik może być nim zainteresowany;
- **generowanie rekomendacji** – zwracana jest lista elementów rekomendowanych użytkownikowi poprzez porównanie profili produktów z profilem użytkownika. Do listy dodawane są elementy, które z największym prawdopodobieństwem będą odpowiednie dla użytkownika.

Zastosowanie

Tego typu podejście sprawdza się szczególnie w przypadku polecania elementów zawierających informacje tekstowe, takich jak strony internetowe, publikacje, filmy, muzyka, a także witryny e-commerce, które wyświetlają stronę z listą produktów [8]. Algorytm zaimplementowano w obszarze rekrutacji do analizy życiorysów zawodowych i rekomendacji posortowanej listy najlepszych kandydatów [3].

Zalety

- Użytkownik otrzymuje rekomendacje zgodne z jego indywidualnymi preferencjami.
- W przeciwieństwie do metody filtrowania zespołowego, profil użytkownika nie jest porównywany z profilami innych użytkowników, nie są wykorzystywane informacje o preferencjach innych użytkowników lub podobieństwa z innymi użytkownikami.
- Możliwe jest polecanie nowych przedmiotów, nawet jeśli nie mają ocen użytkowników.

- System jest skalowalny pod względem liczby użytkowników.
- Użytkownicy mogą otrzymywać rekomendacje bez udostępniania swojego profilu, co zapewnia prywatność.
- System pozwala zapobiegać nieodpowiedniemu dla użytkownika marketingowi i zapewnia ochronę przed reklamowaniem złośliwych przedmiotów.

Ograniczenia

- Techniki są zależne od metadanych produktów. System nie może zwrócić dobrych rekomendacji, jeśli opis produktu nie zawiera wystarczającej ilości informacji, aby odróżnić przedmioty, które odpowiadają użytkownikowi, od przedmiotów, którymi nie będzie on zainteresowany.
- Duża liczba produktów może wymagać kosztownej analizy. Zadanie jest podatne na błędy i czasochłonne.
- Użytkownicy otrzymują rekomendacje podobne do elementów już zdefiniowanych w ich profilach. System często poleca użytkownikowi pozycje, które są podobne do tych, które już wcześniej nabył. Stąd też inne rodzaje przedmiotów, z którymi użytkownik nie jest zaznajomiony, zwykle nie są mu przedstawiane.
- Profile użytkowników są budowane na podstawie statycznych charakterystyk. W rezultacie istnieje duże prawdopodobieństwo, że różni użytkownicy mają podobne profile, nawet jeśli mają różne preferencje wśród tych samych pozycji. Mogą odwiedzać te same produkty lub pozostawiać komentarze, mimo że ich opinie są odmienne.
- Występuje problem nowego użytkownika. Ze względu na brak wcześniejszych danych system nie jest w stanie dawać wiarygodnych sugestii nowym użytkownikom.

9.2.2. Filtrowanie zespołowe

Celem filtrowania zespołowego jest przewidywanie preferencji użytkownika na podstawie zachowań innych użytkowników, którzy w przeszłości wykonali podobne aktywności. Metoda polega na analizowaniu informacji o zachowaniach użytkowników. Filtrowanie zespołowe opiera się na założeniu, że użytkownicy, którzy wybierali te same przedmioty, są skłonni do podobnych wyborów. Zatem biorąc pod uwagę aktywność danego użytkownika, algorytm odnajduje wszystkich użytkowników, którzy wykonali podobne aktywności w przeszłości. Następnie dokonywana jest predykcja oceny produktów, których dany użytkownik nie widział, na podstawie ocen użytkowników podobnych pod względem zainteresowań. Predykcje mogą być reprezentowane jako macierz użytkowników oraz elementów, w której każda komórka reprezentuje ocenę użytkownika dla określonego elementu. Metody filtrowania zespołowego można podzielić na dwie klasy:

oparte na modelu i oparte na pamięci [5]. **Typ oparty na modelu** [9] polega na wykorzystaniu uczenia maszynowego lub technik eksploracji danych. W dotychczasowych pracach wykorzystywano techniki między innymi: faktoryzacji macierzy, redukcji wymiarów macierzy, uzupełniania macierzy, grupowania, analizy skupień, reguł asocjacyjnych, regresji, klasyfikatory bayesowskie, drzewa decyzyjne, sztuczne sieci neuronowe. **Typ oparty na pamięci** [1] przewiduje ocenę produktu na podstawie ocen innych użytkowników. Typ ten można podzielić na dwie metody: opartą na użytkownikach i opartą na przedmiotach.

- Technika filtrowania zespołowego oparta na użytkownikach oblicza podobieństwo między użytkownikami, porównując ich oceny tego samego elementu. Następnie oblicza przewidywaną ocenę elementu dokonaną przez wybranego użytkownika jako średnią ważoną ocen elementu dokonanych przez użytkowników o podobnych preferencjach. Algorytm wymaga obliczenia informacji o każdej parze użytkowników.

- Techniki filtrowania oparte na elementach obliczają predykcje na podstawie podobieństwa między elementami, a nie podobieństwa między użytkownikami. Nawiązują do założenia, że użytkownik prawdopodobnie preferuje przedmioty podobne do tych, które wybrał w przeszłości. Prognozę wykonuje się na podstawie średniej ważonej ocen użytkowników dla elementów podobnych. Najpopularniejsze wykorzystywane miary do obliczania podobieństwa między elementem a użytkownikiem to współczynnik korelacji Pearsona oraz podobieństwo cosinusowe. Algorytm wymaga znacznie mniej zasobów niż filtrowanie oparte na użytkownikach, ponieważ nie wykorzystuje wszystkich wyników podobieństwa między użytkownikami.

Zastosowania

Podejście sprawdza się dla serwisów, w których istnieje możliwość śledzenia aktywności dużej liczby użytkowników, a informacje o produktach mogą nie być ustrukturyzowane lub być trudne do analizy. Na przykład serwis Amazon.com zbiera dane o ocenach i zakupach użytkowników. System przewiduje zainteresowanie użytkowników na podstawie ocenianych przez nich pozycji. Jest skalowalny niezależnie od liczby użytkowników i produktów [7].

Zalety

- System może działać w domenach, w których nie ma zbyt wielu treści związanych z produktami lub treść jest trudna do przeanalizowania.

- Metoda nie opiera się na wstępnym przetwarzaniu danych. Nie wymaga analizowania cech elementów. Może ominąć związane z tym trudności, które pojawiają się w przypadku metody filtrowania opartego na treści.

Ograniczenia

- W przypadku filtrowania zespołowego wymagane są dane o interakcjach użytkowników względem produktów. Skuteczność podejścia zależy od ich dostępności.
- Rzadkie informacje o użytkowniku powodują ograniczenie skuteczności rekomendacji, ponieważ algorytm nie może generować dokładnych rekomendacji dla użytkowników bez wystarczającej liczby danych o aktywności.
- System napotyka problem zimnego startu podczas modelowania rekomendacji dla nowych użytkowników lub przedmiotów, o których system nie posiada żadnych informacji.
- Liczba użytkowników i ich interakcji może szybko rosnąć. W rezultacie system wymaga skalowalności zasobów obliczeniowych. Problem można minimalizować, stosując techniki redukcji wymiarów macierzy.
- W podejściach opartych na ocenach użytkowników istnieje dodatkowe wyzwanie interpretacji ocen.

9.2.3. Hybrydowe systemy rekomendacji

Hybrydowe systemy rekomendacji łączą metody filtrowania zespołowego i filtrowania opartego na treści. Podejście powstało w celu optymalizacji i uniknięcia wad pojedynczych algorytmów. Stosowane jest połączenie kilku algorytmów według wybranej strategii [4]. W sytuacji wyboru podejścia rekomendacji hybrydowych ważne jest zbadanie celów projektowych dla systemu (dokładność, zimny start itp.). Kolejną kwestią do rozważenia przy wyborze technik hybrydyzacji jest ocena wydajności każdego komponentu hybrydy – szczególnie wydajności w czasie wykonywania, ponieważ użytkownicy oczekują szybkiej odpowiedzi systemu. Wybrane strategie scharakteryzowano poniżej:

- **kombinacja mieszana** – wyniki różnych metod rekomendacji są łączone w celu uzyskania ostatecznych rekomendacji;
- **kombinacja ważona** – w początkowym etapie wyniki różnych metod rekomendacji otrzymują równe wagi. Następnie wyniki są łączone, a wagi dostosowywane w zależności od stopnia, w jakim ocena przedmiotu dokonana przez użytkownika pokrywa się z oceną przewidzianą przez system rekomendacji;
- **kombinacja sekwencyjna** – początkowo cechy użytkownika są tworzone przez metody filtrowania oparte na treści. Gdy istnieje wystarczająca liczba ocen użytkowników, metody te można zastąpić metodami filtrowania zespołowego w celu uzyskania ostatecznych rekomendacji [2];
- **przełączanie metod** – rekomendacje pochodzące z dwóch systemów działają na tym samym obiekcie danych. Techniki są przełączane przez

system na podstawie o określonego kryterium przełączania. Podejście zwykle zwiększa liczbę parametrów w systemie;

- **łączenie cech** – funkcje pochodzące z wielu systemów rekomendacji są łączone w jeden system rekomendacji;
- **augmentacja cech** - wybrana technika rekomendacji jest używana do obliczania funkcji lub zestawu funkcji, które następnie stają się częścią danych wejściowych dla innej techniki;
- **metoda kaskadowa** – rekomendacje są udoskonalane w sposób iteracyjny. Pierwszy system rekomendacji ma wyższy priorytet niż inne i może zawęzić wyniki zwrócone przez system rekomendacji o niższym priorytecie;
- **poziom meta** – dane wyjściowe wybranej techniki rekomendacji są stosowane jako dane wejściowe dla innej [4].

Zastosowania

Hybrydowy system rekomendacji łączy wiele technik w celu osiągnięcia synergii między nimi. Przykładowe zastosowanie: techniki oparte na treści mogą wspomagać filtrowanie zespołowe w zrekompensowaniu problemu zimnego startu poprzez dostarczanie nowym użytkownikom rekomendacji opartych na informacjach o produktach. Metoda mieszana rekomenduje elementy użytkownikowi, integrując dane z historii użytkownika, które są zbierane po uruchomieniu systemu.

Zalety

- Metody hybrydowe pozwalają na opanowanie problemów typowych dla technik filtrowania grupowego i opartego na treści.
- Nawet algorytmy rekomendacji ze słabymi źródłami danych mogą mieć silny pozytywny wpływ na wydajność, jeśli zostaną połączone w odpowiednią hybrydę [4].

Ograniczenia

- Wysoka złożoność obliczeniowa. Wdrożenie jest kosztowne i wymaga zbadania wydajności wielu typów strategii.
- Wyprowadzenie hybrydyzacji metod nie zawsze jest możliwe ze względu na specyfikę domeny i danych [2].

9.3. Wyzwania związane z systemami rekomendacji

Zimny start

Początkowym problemem systemu jest ustalenie rekomendacji dla nowych klientów, o których system nie posiada żadnych informacji. Odnosi się to w szczególności do systemów filtrowania zespołowego. W przypadku filtrowania opartego na treści do rekomendacji można wykorzystać informacje o wybranym przez użytkownika produkcie. Jednak problem zimnego startu może wystąpić również w przypadku wprowadzenia danych o nowym produkcie. Oba te problemy można rozwiązać za pomocą podejścia hybrydowego.

Rzadkość danych

W przypadku baz danych o dużej liczbie produktów i małej liczbie aktywności użytkowników względem produktów może wystąpić problem rzadkości macierzy. System może generować wiele podobnych profili użytkowników i polecać te same produkty, ignorując tym samym produkty o ograniczonej ilości danych. Ponadto niepopularne produkty są rzadko zauważane przez użytkowników. Systemy powinny zwracać uwagę na tego typu pozycje.

Synonimizacja

W przypadku filtrowania zespołowego system może nie być w stanie odróżnić blisko powiązanych elementów. W szczególnych przypadkach filtrowania opartego na treści wymagane są wiedza specjalistyczna i taksonomie dla określonej domeny.

Nadmierna specjalizacja

Filtrowanie oparte na treści może ignorować rekomendowanie użytkownikowi nowych elementów. Jeżeli zainteresowania użytkownika zaczną się zmieniać, system nie będzie w stanie dopasować rekomendacji. Rozwiązaniem problemu może okazać się hybrydyzacja z zastosowaniem metod zespołowych.

Dryft danych

Preferencje użytkowników mogą zmieniać się ze względu na ewolucję osobistych zainteresowań, ogólnych trendów, wpływu kulturowego, popularności produktów i usług. W miarę gromadzenia się nowych rekordów historii użytkownika, starsze informacje mogą być nieaktualne. W celu rozwiązania tego problemu powstały koncepcje implementacji zarządzania dynamiką czasową [10].

Skalowalność

Problemy związane ze skalowalnością algorytmiczną i systemową narastają w miarę dostarczania większej ilości danych. System powinien być dostosowany do przetwarzania informacji i generowania rekomendacji wraz ze wzrostem liczby klientów i produktów.

Prywatność i bezpieczeństwo

Ważnym wyzwaniem dla twórców systemów rekomendacyjnych jest znalezienie sposobów wykorzystania danych do ulepszania algorytmów przy jednoczesnym zachowaniu prywatności. Systemy analizujące opinie użytkowników mogą posiadać informacje o zainteresowaniach, poglądach politycznych, orientacji i preferencjach osobistych. Mogą to być informacje wrażliwe, co prowadzi do obaw o prywatność. Ponadto wyniki rekomendacji mogą stać się stronnicze w wyniku celowego działania użytkowników [6]. Pomimo coraz lepszej skuteczności systemów rekomendujących użytkownikom trudno jest im zaufać ze względu na obawy związane z prywatnością.

9.4. Podsumowanie

Systemy rekomendacyjne umożliwiają złagodzenie problemu przeciążenia informacjami poprzez odnajdywanie spersonalizowanych treści, usług, produktów. Przy wyborze metody rekomendacji należy uwzględnić wymagania biznesowe systemu oraz zwrócić uwagę na wyzwania związane z wdrożeniem wybranej metody. W tym rozdziale przedstawiono tradycyjne techniki uczenia wykorzystywane do generowania modeli rekomendacji oraz omówiono wyzwania z nimi związane. Zawarta wiedza może stanowić podstawę do dalszych badań i ulepszania klasycznych technik rekomendacji.

Bibliografia

[1] AL-Ghuribi S.M., Noah S.A.M. *A Comprehensive Overview of Recommender System and Sentiment Analysis. arXiv preprint arXiv:2109.08794*, 2021.

[2] Albadvi A., Shahbazi M. *A hybrid recommendation technique based on product category attributes. Expert Systems with Applications*, 36(9):11480–11488, 2009.

[3] Almalis N.D., Tsihrintzis G.A., Karagiannis N. *A content based approach for recommending personnel for job positions.* In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, pp. 45–49. IEEE, 2014.

[4] Burke R. *Hybrid web recommender systems. The adaptive web*, pp. 377–408, 2007.

[5] Chen L., Chen G., Wang F. *Recommender systems based on user reviews: the state of the art. User Modeling and User-Adapted Interaction*, 25(2):99–154, 2015.

[6] Kumar S., et al. *Survey on Personalized Web Recommender System. International Journal of Information Engineering & Electronic Business*, 10(4), 2018.

[7] Linden G., Smith B., York J. *Amazon. com recommendations: Item-to-item collaborative filtering. IEEE Internet computing*, 7(1):76–80, 2003.

[8] Pazzani M.J., Billsus D. *Content-based recommendation systems.* In *The adaptive web*, pp. 325–341. Springer, 2007.

[9] Su X., Khoshgoftaar T.M. *A survey of collaborative filtering techniques. Advances in artificial intelligence*, 2009, 2009.

[10] Zhang Q., Lu J., Jin Y. *Artificial intelligence in recommender systems. Complex & Intelligent Systems*, 7(1):439–457, 2021.

Rozdział 10

Wydobywanie wiedzy z Wikipedii

Jarosław Kuchta[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
qhta@eti.pg.edu.pl

Abstrakt

Wikipedia jest olbrzymim źródłem wiedzy encyklopedycznej gromadzonej przez ludzi i przeznaczonej dla ludzi. W systemach informatycznych odpowiednikiem takiego źródła wiedzy są ontologie. Ten rozdział pokazuje, w jaki sposób Wikipedia jest transformowana w ontologię i jak wydobywać z niej pojęcia, ich właściwości i relacje między nimi.

Słowa kluczowe: Wikipedia, DBpedia, SPARQL, ontologia, Semantic Web

10.1. Wprowadzenie

Pierwsze próby wydobywania wiedzy z Wikipedii sięgają początków XXI wieku. W 2001 r. Woodward [15] stworzył na bazie parsera Stanford NLP[1] narzędzie do

[1] https://nlp.stanford.edu/

wydobywania i wizualizacji informacji czasowych i związanych z nimi nazwanych encji. Denoyer i Gallinari w 2006 r. [4] pokazali korpus lingwistyczny XML zbudowany na bazie Wikipedii i wykorzystujący kategorie Wikipedii do klasyfikowania jednostek lingwistycznych. W 2014 r. Flickinger, Oepen i Ytrestøl [7] zaprojektowali system WikiWoods do wprowadzania syntaktyczno-semantycznych adnotacji do drzew parsowania, który analizuje infoboksy Wikipedii dla pozyskania podstawowych informacji o analizowanych jednostkach. Ich wysiłki powtórzyli Margaretha i Lüngen w 2014 r. [10]. W 2011 r. Exner i Nugues [5] podjęli próbę wydobywania informacji o zdarzeniach z Wikipedii.

Istnieje możliwość pozyskiwania informacji z Wikipedii poprzez zapytania kierowane do odpowiednich „punktów końcowych" (*endpoints*) Wikipedii. Każda wersja językowa Wikipedii ma punkt dostępowy Dla przykładu angielska wersja Wikipedii ma punkt końcowy pod adresem `https://en.wikipedia.org/w/api.php`. Usługi akceptują zapytania w formie REST, a wyniki dają w formacie JSON lub XML[2].

Wśród licznych usług kluczowe dla automatycznej analizy są *query* i *parse*. Pierwsza z nich podaje szereg metadanych dotyczących wybranych stron, jak np. lista kategorii, do których należy dana strona, lista hiperłączy do innych stron, lista właściwości przypisanych do danej strony. Usługa *parse* podaje treść określonej strony (lub jej fragmentów) w oryginalnym formacie Wiki, przetworzoną do formatu HTML albo półskompilowaną do formatu XML. Wśród tej treści często można znaleźć Infobox, czyli tabelę zawierającą najważniejsze dane dotyczące tematu artykułu (rys. 10.1). Dla przykładu infoboksy artykułów opisujących języki świata zawierają takie dane, jak oryginalna nazwa języka, taksonomia, system pisma (alfabet), nazwy ludów posługujących się tym językiem itp. Odpowiednio napisany program może wydobyć informacje z tabeli infoboksu przez jej parsowanie. Jest to stosunkowo łatwe, ale niestety czasochłonne. A co jeśli poszukujemy tych informacji dla setek lub tysięcy artykułów, np. chcemy uzyskać listę wszystkich języków świata, które używają pisma niełacińskiego?

Tu z pomocą przychodzą takie rozwiązania, jak Wikidata [14], DBpedia [8], Freebase [3], OpenCyc [11] czy YAGO [9]. W tym rozdziale skoncentrujemy się na Wikidanych i DBpedii, jako że oba rozwiązania są ściśle związane z Wikipedią. Freebase była bazą wiedzy tworzoną przez Google od 2010 r, ale porzuconą w 2015 r. na rzecz Wikidata. OpenCyc była kontynuacją projektu sztucznej inteligencji Cyc, rozpoczętego w 1984 r., którego celem było stworzenie zaawansowanej ontologii i bazy wiedzy. Projekt został odnowiony w 2002 r. i ostatecznie zaniechany w 2017 r. YAGO (Yet Another Great Ontology) opracowano w Instytucie Maxa Planka w Saarbrücken. Pierwsze wydanie było w 2008 r., ale w 2020 r. baza jeszcze działała. Porównanie wymienionych tu baz wiedzy można znaleźć w pracy [11].

[2]Szczegóły w rozdziale 11.

Rysunek 10.1: Infobox

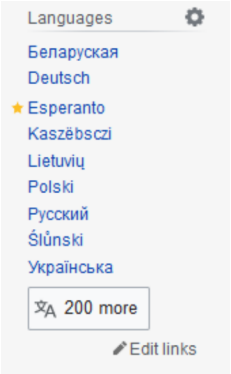
10.2. Wikidata

Koncepcja potraktowania Wikipedii jako bazy danych legła u podstaw projektu Wikidata (po polsku Wikidane). Projekt ten narodził się w 2012 r. w niemieckim oddziale fundacji Wikimedia i jego pierwotnym celem była centralizacja powiązań między różnymi wersjami językowymi Wikipedii [2]. Tu trzeba sobie uświadomić, że istnieje ponad 300 wersji językowych Wikipedii[3]! Wiele artykułów ma swoje odpowiedniki w różnych wersjach językowych, a na stronie artykułu w danym języku publikowane są powiązania (hiperłącza) do artykułów w innych językach na ten sam temat (rys. 10.2).

Ponieważ ten sam temat artykułu może mieć różne tytuły w różnych językach, dla ujednoznacznienia wprowadzono unikatowy, niezmienny identyfikator będący dodatnią liczbą całkowitą poprzedzoną wielką literą Q. Przykładowo temat „esperanto" ma identyfikator Q143. W bazie Wikidata zostały umieszczone tytuły artykułów z poszczególnych wersji językowych jako „etykiety". Wpisano też etykiety zastępcze na podstawie zarejestrowanych w Wikipedii artykułów przekierowanych danego tematu oraz krótkie opisy. Tym samym każdy temat (*subject*) zawiera:

- identyfikator unikatowy (QID) – wspólny dla wszystkich języków;

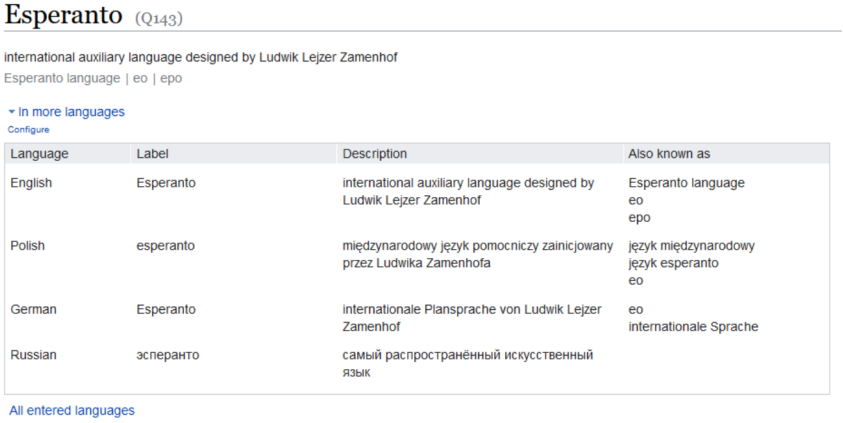
[3]Maj 2022 – **325** edycji językowych Wikipedii (315 czynnych i 10 zamkniętych) [`https://pl.wikipedia.org/wiki/Wikipedia:Lista_wersji_językowych`]



Rysunek 10.2: Powiązania międzyjęzykowe

- jedną lub wiele etykiet (*labels*) – po jednej w każdym języku;
- opcjonalnie wiele etykiet zastępczych (*aliases*) – wiele w każdym języku;
- opcjonalnie wiele opisów (*descriptions*) – po jednym w każdym języku.

Przykład dla języka esperanto przedstawiono na rys. 10.3.



Esperanto (Q143)

international auxiliary language designed by Ludwik Lejzer Zamenhof

Esperanto language | eo | epo

In more languages

Configure

Language	Label	Description	Also known as
English	Esperanto	international auxiliary language designed by Ludwik Lejzer Zamenhof	Esperanto language eo epo
Polish	esperanto	międzynarodowy język pomocniczy zainicjowany przez Ludwika Zamenhofa	język międzynarodowy język esperanto eo
German	Esperanto	internationale Plansprache von Ludwik Lejzer Zamenhof	eo internationale Sprache
Russian	эсперанто	самый распространённый искусственный язык	

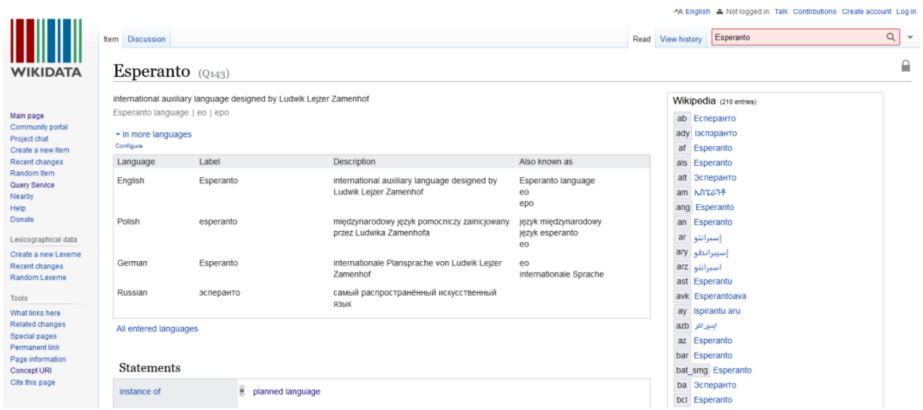
All entered languages

Rysunek 10.3: Przykładowa rejestracja języka esperanto w Wikidanych (`https://www.wikidata.org/wiki/Q143`)

Oczywiście sformułowanie „każdy język” nie oznacza, że każdy temat został zarejestrowany w ponad 300 językach, lecz „każdy język Wikipedii, w którym opisano dany temat”.

10.2.1. Przeglądarka Wikidata

Wikidane mają swoją oficjalną przeglądarkę pod adresem `https://www.wikidata.org/wiki/`. Można się nią posługiwać podobnie jak Wikipedią, wpisując hasło w pole oznaczone symbolem wyszukiwania (rys. 10.4).



Rysunek 10.4: Okno przeglądarki Wikidanych

Część główna okna przedstawia etykiety (w tym etykiety zastępcze) i krótkie opisy encji oraz listę stwierdzeń. Osobno, w prawym panelu znajduje się lista linków do różnojęzycznych wersji Wikipedii. Lewy panel umożliwia dodatkową nawigację.

10.2.2. Wikibase

Przeglądarka Wikidanych jest obsługiwana przez oprogramowanie Wikibase, które jest rozszerzeniem MediaWiki. Wikibase składa się z dwóch części: scentralizowanego repozytorium przechowującego częściowo ustrukturalizowane dane oraz oprogramowania klienckiego umożliwiającego wyszukiwanie i pobieranie danych z repozytorium, a także wpisywanie do niego nowych, uporządkowanych danych.

Wikibase zostało wydane na licencji GNU General Public License 2.0 i jest też używane w innych projektach, np. Lingua Libre[4], Europeana Eagle[5] czy OpenStreetMap[6].

10.2.3. Model danych

Wikibase jest nierelacyjną bazą danych (NoSQL). Jednostką danych jest „encja" (*entity*), unikatowy identyfikator, etykieta (pochodząca z języka naturalnego), właściwość (*property*) opisująca cechę danego podmiotu oraz wartość właściwości. Encją jest klasa lub instancja klasy. Identyfikatory są dwóch rodzajów: identyfikatory zaczynające się od Q są przypisane do encji, a zaczynające się od P – do właściwości. Właściwości łączą ze sobą encje (wyrażają relacje między encjami) lub przypisują wartości do encji. Wartości mogą być typu teksto-

[4] `https://lingualibre.org/wiki/LinguaLibre:Main_Page`
[5] `https://pro.europeana.eu/project/eagle`
[6] `https://www.openstreetmap.org`

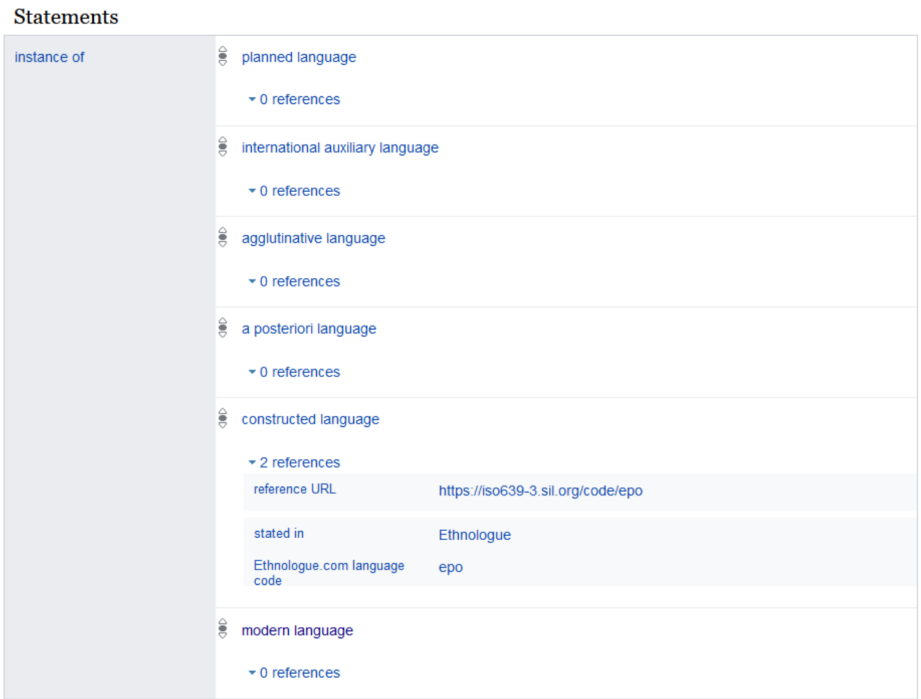
wego, liczbowego, logicznego (istnieje/nie istnieje), czasowego lub przestrzennego (współrzędne geograficzne).

Relacje między encjami i przypisanie wartości do encji są w Wikibase wyrażane przez poprzez „semantyczne stwierdzenia" (*semantic statements*), z których każde jest trójką postaci:

<podmiot> <predykat> <przedmiot>

gdzie podmiotem (*subject*) jest jednostka (encja), predykatem (*predicate*) jest relacja między encjami lub właściwość przypisująca wartość, a przedmiotem (*object*) jest inna encja lub wartość właściwości. W ten sposób dane Wikibase mogą być wyrażone w schemacie RDF (*Resource Description Framework*).

Dla przykładu encja „esperanto" zawiera sześć stwierdzeń dotyczących relacji <instance of> (rys. 10.5). Można się z nich dowiedzieć, że esperanto jest przykładem języka planowanego (*planned language*), międzynarodowego języka pomocniczego (*international auxiliary language*), języka aglutynacyjnego (*agglutinative language*), języka aposteriorycznego (*a posteriori language*), języka sztucznego (*constructed language*) oraz języka współczesnego (*modern language*).



Rysunek 10.5: Stwierdzenia <instance of> dla esperanto

W rzeczywistej implementacji użycie schematu RDF jest bardziej skomplikowane, gdyż stwierdzenia mogą być opatrzone adnotacjami, takimi jak ranga

(*rank*), odsyłacze (*references*) czy kwalifikatory (*qualifiers*).

Ranga (*rank*) służy do określenia stopnia ważności stwierdzenia. W Wikibase stosuje się trzystopniowe rangi: podstawową (*normal*), podwyższoną (*preferred*) i obniżoną (*deprecated*). Rangi są istotne, gdy spośród wielu stwierdzeń z pewnych względów chcemy niektóre wyróżnić. Dla przykładu do filmu może się odnosić wiele stwierdzeń dotyczących obsady, ale niektóre stwierdzenia dotyczą obsady ról głównych – i tym należałoby nadać rangę podwyższoną.

Odsyłacze (*references*) zawierają informacje o źródłach stwierdzenia spoza Wikipedii, np. hiperłącza do serwisów informacyjnych. Odsyłacze mogą mieć złożoną strukturę wewnętrzną.

W przykładzie z rys. 10.5 każde z tych stwierdzeń ma rangę normalną (wyrażoną przez znacznik), a stwierdzenie o języku sztucznym ma dwa odsyłacze do odpowiednich źródeł.

Kwalifikatory (*qualifiers*) dostarczają innej informacji o stwierdzeniu. Mogą precyzować, kiedy dana informacja została wprowadzona, np. przy stwierdzeniu o ludności danego kraju kwalifikator może podawać, do którego roku się to odnosi. Przy stwierdzeniach o sprawowaniu rządów (np. przez króla, przez prezydenta) mogą określać datę rozpoczęcia i zakończenia rządów. Kwalifikatory można uznać za właściwości z wartościami przypisane nie do encji, a do samego stwierdzenia.

10.2.4. Stwierdzenia Wikidanych

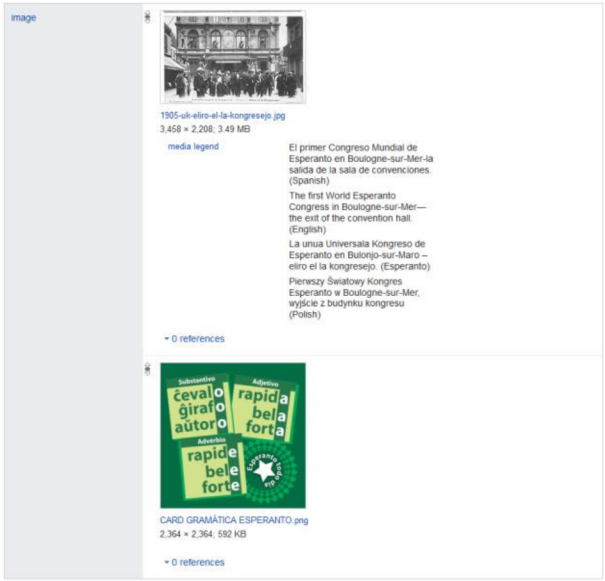
Na rys. 10.5 przedstawiono stwierdzenia Wikidanych dotyczące relacji <instance of> encji „esperanto". Oprócz nich encja ta ma szereg innych stwierdzeń, które zostaną przeanalizowane poniżej.

Na rys. 10.6 zaprezentowano stwierdzenia dotyczące obrazów powiązanych z encją „esperanto". Są tam dwa obrazy, z których jeden ma legendę (*media legend*) w kilku językach.

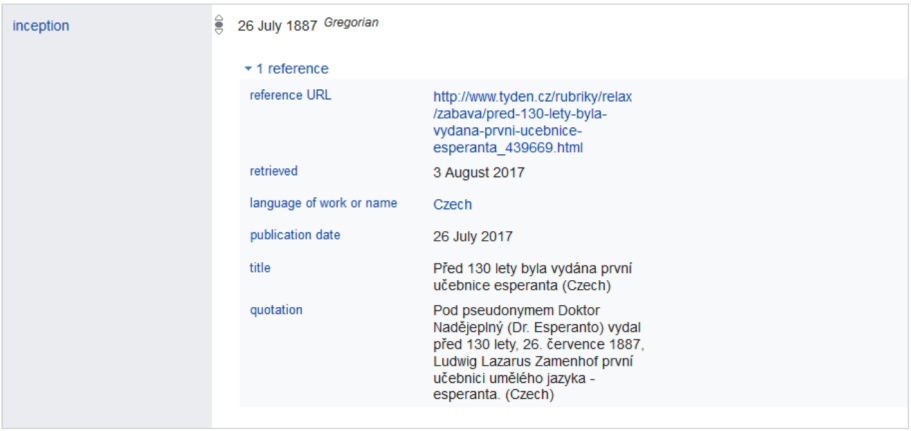
Datę wprowadzenia esperanto uwidoczniono na rys. 10.7. Jest ona opatrzona nie tylko rozbudowanym odsyłaczem, ale też oznaczeniem kalendarza (gregoriański).

Z kolei rys. 10.8 przedstawia stwierdzenie dotyczące oryginalnej nazwy (*native label*) esperanto. Ta właściwość jest charakterystyczna dla języków naturalnych. Do właściwej nazwy są dołączone jej transkrypcja w notacji IPA oraz plik dźwiękowy, w którym zarejestrowano wymowę nazwy. Pliki dźwiękowe z próbkami wymowy esperanto są też dołączone do samej encji przez właściwość <spoken text audio>(rys. 10.9).

Pozostałe właściwości esperanto to: <named after>, <movement>, <anthem>, <culture>, <location of formation>, <studied by>, <foundational text>, <follows>, <creator>, <country of origin>, <influenced by>, <significant event>, <audio>, <linguistic typology>, <has grammatical case>, <has tense>, <has grammatical mood>, <writing system>, <uses capitalization for>, <language regulatory body>, <signed form>, <number of speakers>, <Ethnologue language status>, <discoverer or inventor>, <has use>, <hashtag>, <flag>, <flag image>, <seal image>, <distribution map>, <used



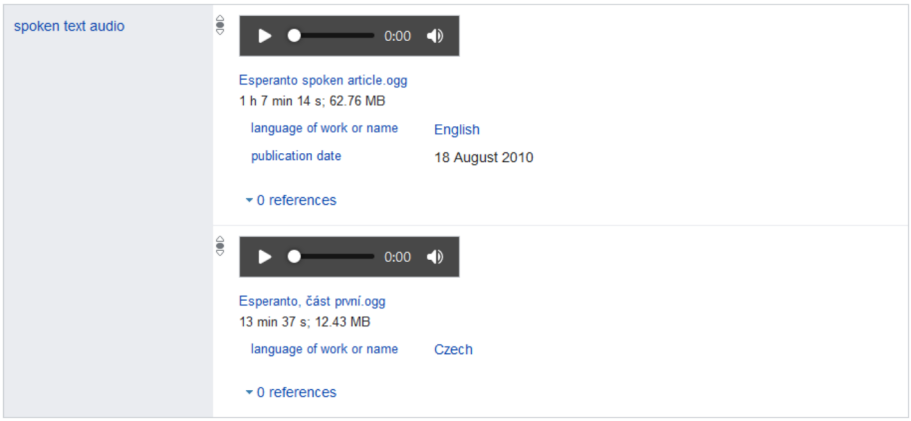
Rysunek 10.6: Stwierdzenia dotyczące obrazów powiązanych z esperanto



Rysunek 10.7: Stwierdzenie dotyczące daty wprowadzenia esperanto

by>, <history of topic>, <described by source>, <on focus list of Wikimedia project>, <Wikimedia outline>, <category for films in this language>, <maintained by WikiProject>, <related category>, <dedicated heritage entity>, <IRC channel>, <Stack Exchange site>, <ISOCAT id>, <OpenStreetMap tag or key>, <Wikimedia language code>, <exact match>, <different from>, <Commons category>, <topic's main Wikimedia portal>, <topic's main category>, <category for maps>, <catalog> – w sumie 51 właściwości. Porównu-

Rysunek 10.8: Stwierdzenie dotyczące oryginalnej nazwy esperanto



Rysunek 10.9: Próbki wymowy esperanto

jąc listę właściwości podawanych przez Wikidane z listą informacji zawartych w infoboksach Wikipedii w różnych językach, można od razu zauważyć, że Wikidane zawierają więcej danych. Jest tak dlatego, że rozwój Wikidanych nie ogranicza się do parsowania Wikipedii, ale jest wspierany osobno przez społeczność. Podobnie jak każdy użytkownik Wikipedii po zarejestrowaniu się może wprowadzać i edytować artykuły, tak też każdy użytkownik serwisu Wikidanych może wprowadzać i edytować dane. Nasuwa się pytanie o wiarygodność informacji pobieranych z Wikidanych (i z Wikipedii). Na to pytanie spróbujemy odpowiedzieć w sekcji 10.5.

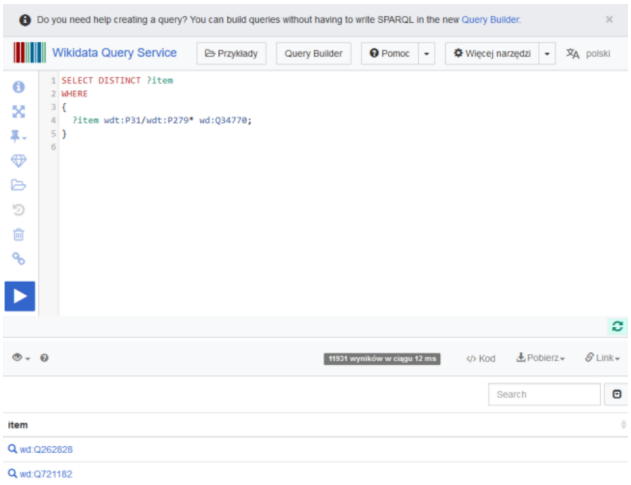
10.3. SPARQL

Przeglądarka Wikidata pozwala wyszukiwać encje po różnojęzycznych nazwach i nawigować pomiędzy nimi. Co jednak, gdy użytkownik chce wyszukać szereg encji spełniających określony warunek, np. wszystkie języki świata, które posługują się pismem niełacińskim? Do tego celu od 2015 r. fundacja Wikimedia udostępnia internetową usługę Wikidata Query Service, która umożliwia zapytania o dane z wykorzystaniem SPARQL.

SPARQL jest językiem zapytań podobnym do SQL, ale nastawionym na

wyszukiwanie stwierdzeń w bazie danych trójek zgodnych ze specyfikacją RDF. Na podobieństwo SQL język SPARQL zawiera takie klauzule, jak SELECT, WHERE, SORT, ale też inne, jak CONSTRUCT (do tworzenia encji), ASK (do zapytania, czy wzór zapytania ma rozwiązanie w bazie trójek), DESCRIBE (do pobrania „opisu" encji).

Przykłady SPARQL znajdują się pod adresem `https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples`, a punkt końcowy usługi WDQS pod adresem: `https://query.wikidata.org`. Strona dostępowa (rys. 10.10) umożliwia wprowadzanie i przesyłanie zapytań oraz przeglądanie odpowiedzi. Dostęp dla klientów softwarowych jest możliwy poprzez interfejs REST. Dla początkujących strona udostępnia narzędzie Query Builder, które ułatwia konstruowanie zapytań, chociaż jego możliwości w porównaniu ze SPARQL są bardzo ograniczone.



Rysunek 10.10: Strona dostępu do Wikidanych przez SPARQL

Strona `https://query.wikidata.org` udostępnia również szereg narzędzi (różnych autorów) umożliwiających:

- edycję elementów;
- podawanie danych do zapytań;
- rozszerzanie interfejsu użytkownika;
- wizualizację danych;
- przeglądanie listy właściwości;
- wyszukiwanie danych leksykograficznych (leksemów);
- wiele innych działań przydatnych dla programistów.

10.3.1. Konstruowanie zapytania – przykład

Najprostsze zapytanie o języki świata ma formę jak poniżej. Zapytanie to wyszukuje instancje klasy „język".

```
SELECT DISTINCT ?item
WHERE
{
  ?item wdt:P31 wd:Q315;
}
```

Rozpoczyna się od klauzuli SELECT (z opcją DISTINCT dla wyeliminowania powtarzania się wyników) podającej nazwę wyszukiwanej zmiennej (?item). Następnie klauzula WHERE podaje warunek wyszukiwania w postaci trypletu RDF. Pierwszy element trypletu (?item) jest podmiotem i określa nazwę zmiennej, do której będą przypisywane wyniki. Drugi element (wdt:P31) to predykat, który określa relację łączącą podmiot z przedmiotem określonym przez trzeci element (wd:Q34770)

Zarówno predykat, jak i przedmiot są podane z użyciem identyfikatorów Wikidanych. Identyfikatory rozpoczynające się literą P (tutaj P31) odnoszą się do właściwości, a rozpoczynające się literą Q (Q34770) – do encji. Prefiksy wdt: i wd: wskazują na przestrzenie nazw, w których identyfikatory są zdefiniowane – odpowiednio <http://www.wikidata.org/prop/direct/> oraz <http://www.wikidata.org/entity/> [7].

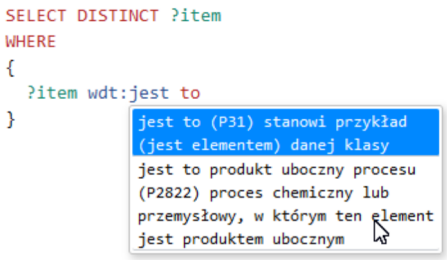
Posługiwanie się identyfikatorami w zapytaniach zapewnia jednoznaczność zapytań i uniezależnia zapytania od języka, którym posługuje się użytkownik, ale oczywiście jest niewygodne. Dla ułatwienia składania zapytań strona dostępowa WQS ma specjalne narzędzie edycyjne, które podpowiada identyfikatory przy wprowadzaniu nazw w języku naturalnym użytkownika. Dla przykładu, jeśli użytkownik wpisze „wdt:" i naciśnie Ctrl+spacja, otworzy mu się okienko podpowiedzi (rys. 10.11). Dalej, jeśli napisze „jest to" (po polsku), to w okienku podpowiedzi pojawią się dwie możliwości do wyboru: „jest to (P31) stanowi przykład (jest elementem) danej klasy" oraz „jest to produkt uboczny procesu (P2822) proces chemiczny lub przemysłowy, w którym ten element jest produktem ubocznym". Oczywiście opcja pierwsza jest tą właściwą.

Dalsze wprowadzenie „wd:język" podaje jeszcze więcej opcji (rys. 10.12) – nie dość, że termin „język" ma wiele znaczeń, to Wikidane podają wszystkie encje, których etykiety zaczynają się od słowa „język". Zwróćmy uwagę, że chociaż podajemy termin w języku polskim, to Wikidane wyszukują znaczenia tego terminu również w innych językach (np. w angielskim).

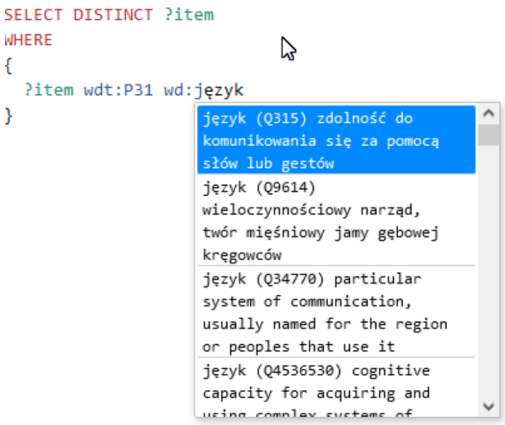
Tak sformułowane zapytanie o języki daje prostą listę identyfikatorów encji, jak pokazano na rys. 10.13. Nie są to wyniki czytelne dla człowieka, gdyż w zapytaniu zabrakło żądania etykiety.

Aby pojawiły się etykiety, trzeba do listy pól dodać zmienną „?itemLabel" i zażądać jej wypełniania przez usługę Wikibase:Label. Obowiązkowym parametrem usługi jest wybór języka. Podając parametr „[auto_language]", żądamy

[7] Lista predefiniowanych prefiksów jest dostępna pod adresem: `https://www.wikidata.org/wiki/EntitySchema:E49`



Rysunek 10.11: Podpowiadanie identyfikatorów właściwości



Rysunek 10.12: Podpowiadanie identyfikatorów encji

Rysunek 10.13: Najprostsze zapytanie o języki świata

etykiet w automatycznie wykrywanym języku. W wynikach pojawia się wówczas druga kolumna (rys. 10.14), chociaż zamiast czytelnych dla człowieka nazw występują w niej w dalszym ciągu nieczytelne identyfikatory.

Jest tak dlatego, że automatycznie wykrywanym językiem w tym przypadku jest język polski, w którym nie ma etykiet dla wybranych encji. Warto spróbować wymusić wyszukiwanie w języku angielskim, zmieniając „[auto_language]" na „en". Pojawiają się etykiety w języku angielskim (rys. 10.15), chociaż nie dla

```
SELECT DISTINCT ?item ?itemLabel
WHERE
{
  ?item wdt:P31 wd:Q315;
  SERVICE wikibase:label {
    bd:serviceParam
    wikibase:language "[auto_language]".
  }
}
```

item	itemLabel
wd:L652971	L652971
wd:Q11269170	Q11269170
wd:Q12974392	Q12974392
wd:Q12974537	Q12974537
wd:Q12980369	Q12980369
wd:Q12980982	Q12980982
wd:Q12982112	Q12982112

Rysunek 10.14: Etykiety języków w języku wykrywanym automatycznie

wszystkich encji.

```
SELECT DISTINCT ?item ?itemLabel
WHERE
{
  ?item wdt:P31 wd:Q315;
  SERVICE wikibase:label {
    bd:serviceParam
    wikibase:language "en".
  }
}
```

item	itemLabel
wd:Q85760486	Fanafo
wd:Q106483542	Occultec
wd:Q106554800	Tjupan dialect
wd:L652971	L652971
wd:Q11269170	Q11269170
wd:Q12974392	Q12974392
wd:Q12974537	Q12974537
wd:Q12980369	Q12980369

Rysunek 10.15: Etykiety języków w języku angielskim

```
SELECT DISTINCT ?item
?itemLabel
  ?itemDescription
WHERE
{
  ?item wdt:P31 wd:Q315;
  SERVICE wikibase:label {
    bd:serviceParam
    wikibase:language "en".
  }
}
```

item	itemLabel	itemDescription
wd:Q106554800	Tjupan dialect	
wd:L652971	L652971	
wd:Q85760486	Fanafo	Oceanic language spoken in Vanuatu
wd:Q11269170	Q11269170	
wd:Q111346251	Tamazight (Arzew)	Language of South of Oran and the Bay of Arzew
wd:Q12974392	Q12974392	

Rysunek 10.16: Dodana kolumna z krótkim opisem encji

Okazuje się, że nawet w języku angielskim nie wszystkie encje mają etykiety. Jest tak dlatego, że część encji reprezentuje pewne sztuczne klasy, które mają znaczenie dla klasyfikowania Wikidanych, ale niekoniecznie dla człowieka. Można zażądać podawania krótkich opisów (poprzez podanie zmiennej „?itemDescription"), ale to niewiele daje (rys. 10.16).

Strona WDQS podaje, że znaleziono 56 wyników będących przykładami języka. To znacznie za mało. Jest tak dlatego, że predykatem jest P31 – „stanowi przykład (jest elementem) danej klasy". Dla tego predykatu wyszukiwane są tylko encje będące bezpośrednimi instancjami klasy „język". Dla rozwiązania tego problemu Wikidane zalecają stosować ścieżkę właściwości „wdt:P31/wdt:P279*".

Predykat ten oznacza wszystkie instancje danej klasy lub dowolnej klasy pochodnej na dowolnym poziomie. Po zmianie zapytania otrzymujemy już 2737 wyników (rys. 10.17).

```
SELECT DISTINCT ?item ?itemLabel
  ?itemDescription
WHERE
{
  ?item wdt:P31/wdt:P279* wd:Q315;
  SERVICE wikibase:label {
    bd:serviceParam
    wikibase:language "en".
  }
}
```

item	itemLabel	itemDescription
wd:Q337922	CORC	programming language
wd:Q381247	QBasic	programming language
wd:Q206040	Mercury	functional logic programming language
wd:Q219959	Zonnon	programming language
wd:Q319268	D	programming language
wd:Q244627	Brainfuck	esoteric, minimalist programming language
wd:Q383994	J	programming language

Rysunek 10.17: Dodanie instancji klas pochodnych do zapytania

Okazuje się jednak, że spora część uzyskanych encji to języki programowania. Zachodzi uzasadnione podejrzenie, że encja Q315 oznaczająca „zdolność do komunikowania się za pomocą słów i gestów" nie jest właściwą definicją dla wyszukiwania języków świata. Lepsza byłaby chyba encja Q34770 – „szczególny system komunikacji, zwykle nazwany od regionu lub ludu, który go używa". Encja ta ma opis tylko w języku angielskim, stąd w pierwszym podejściu została pominięta. Nowe zapytanie daje 11914 wyników (rys. 10.18).

```
SELECT DISTINCT ?item
  ?itemLabel
  ?itemDescription
WHERE
{
  ?item wdt:P31/wdt:P279*
    wd:Q34770;
  SERVICE wikibase:label {
    bd:serviceParam
    wikibase:language "en".
  }
}
```

item	itemLabel	itemDescription
wd:Q35037	Ndyuka	creole language of Suriname
wd:Q721182	Kwinti	language
wd:Q18463598	Indo-Portuguese Creole of Bombay	dead creole language of India
wd:Q33954	Sango	language spoken in the Central African Republic
wd:Q7842493	Trinidadian Creole	English-based creole language spoken in Trinidad
wd:Q2669169	Kristang	creole language spoken by the Kristang people
wd:Q33779	Saramaccan	creole spoken near the Saramacca and Suriname rivers
wd:Q35785	Macanese Patois	Portuguese-based creole spoken originally in Macau
wd:Q35939	Jamaican Creole	an English-based creole spoken in and around Jamaica; it additionally takes influence from various African languages, particularly Akan

Rysunek 10.18: Właściwe zapytanie o języki świata

W podobny sposób możemy uzyskać listę systemów pisma. Tu strona WDQS podpowiada identyfikator Q8192 jako „system pisma". Otrzymujemy 1908 rekordów (rys. 10.19).

Jak uzyskać zestawienie języków świata z ich systemami pisma? Trzeba odwołać się do właściwości „script" encji zwracanych w poprzednim zapytaniu. Strona WDQS podpowiada identyfikator właściwości „P282". Zmienimy nazwę zmiennej encji na „?lang" i wprowadzimy nową zmienną „?script". Zmienna ta będzie przechowywać wartość właściwości „P282", która jest referencją do encji podawanej w zapytaniu o systemy pisma. Oczywiście oprócz identyfikatora encji „script" potrzebujemy jeszcze jej etykiety, więc do rekordu wynikowego dodajemy pole „scriptLabel", które będzie rozpoznawane przez usługę Wiki-

```
SELECT DISTINCT ?item
   ?itemLabel
   ?itemDescription
WHERE
{
   ?item wdt:P31/wdt:P279*
      wd:Q8192;
   SERVICE wikibase:label {
      bd:serviceParam
      wikibase:language "en".
   }
}
```

item	itemLabel	itemDescription
wd:Q364727	Q364727	
wd:Q1260436	Rasm	Arabic writing script
wd:Q1285080	Roman cursive	form of handwriting used in ancient Rome and to some extent into the Middle Ages
wd:Q2442696	Russian orthography before 1918	
wd:Q4069703	Q4069703	
wd:Q5399940	Q5399940	
wd:Q12031948	curial	script
wd:Q711487	gugyeol	system for rendering texts written in Classical Chinese into understandable Korean

Rysunek 10.19: Zapytanie o systemy pisma

base:Label. (Dla przejrzystości usunęliśmy z zapytania opis języka). W wyniku uzyskaliśmy 2414 rekordów (rys. 10.20).

```
SELECT DISTINCT ?lang
   ?langLabel
   ?script
   ?scriptLabel
WHERE
{
   ?lang wdt:P31/wdt:P279* wd:Q34770;
         wdt:P282 ?script.
   SERVICE wikibase:label {
      bd:serviceParam
      wikibase:language "en".
   }
}
```

lang	langLabel	script	scriptLabel
wd:Q5146	Portuguese	wd:Q8229	Latin script
wd:Q5111	Abkhaz	wd:Q8209	Cyrillic script
wd:Q7918	Bulgarian	wd:Q8209	Cyrillic script
wd:Q7930	Malagasy	wd:Q576484	Sorabe alphabet
wd:Q7737	Russian	wd:Q187846	Russian alphabet
wd:Q5885	Tamil	wd:Q1992000	Vatteluttu alphabet
wd:Q4627	Aymara	wd:Q41670	Latin alphabet

Rysunek 10.20: Zapytanie o języki i ich systemy pisma

Wróćmy do problemu postawionego we wprowadzeniu – jak uzyskać listę wszystkich języków świata, które używają pisma niełacińskiego. Trzeba nałożyć filtr na ostatnie zestawienie, eliminując te encje, których pismo to „Latin script" (Q8229) lub „Latin alphabet" (Q41670). Służy do tego klauzula FILTER, która zawiera odpowiednie wyrażenie filtrujące. Tym razem uzyskaliśmy 1151 rekordów (rys. 10.21).

Jak widać, skonstruowanie poprawnego zapytania w języku SPARQL nie jest łatwe i wymaga czasami wielu prób. Wymaga nie tylko posługiwania się wewnętrznymi (nienaturalnymi) identyfikatorami encji i właściwości, ale też pewnej znajomości wewnętrznej struktury encji Wikidanych.

10.3.2. Wybrane implementacje SPARQL

Wikidane używają magazynu danych i grafowej bazy danych Blazegraph. Wiele innych firm ma własne implementacje języka SPARQL. Zestawienie dużych magazynów trójek jest dostępne na stronie organizacji W3C[8] (tabl. 10.1). Charakterystyczne jest to, że dużą popularność mają bazy danych wielomodelowe,

[8] `https://www.w3.org/wiki/LargeTripleStores`

```
SELECT DISTINCT ?lang
  ?langLabel
  ?script
  ?scriptLabel
WHERE
{
  ?lang wdt:P31/wdt:P279* wd:Q34770;
        wdt:P282 ?script.
  SERVICE wikibase:label {
    bd:serviceParam
    wikibase:language "en".
  }
  FILTER (!(?script=wd:Q8229
    ||?script=wd:Q41670))
}
```

lang	langLabel	script	scriptLabel
wd:Q7495	Aari	wd:Q257634	Ethiopic
wd:Q27567	Abaza	wd:Q8209	Cyrillic script
wd:Q5111	Abkhaz	wd:Q8209	Cyrillic script
wd:Q27683	Acehnese	wd:Q1828555	Arabic script
wd:Q4699526	Achomi	wd:Q744068	Persian alphabet
wd:Q1750889	Adûnaic	wd:Q473725	Tengwar
wd:Q27776	Adyghe	wd:Q4058023	Adyghe alphabet
wd:Q27776	Adyghe	wd:Q1828555	Arabic script
wd:Q27776	Adyghe	wd:Q8209	Cyrillic script
wd:Q35952	Aegean Macedonian	wd:Q8216	Greek alphabet
wd:Q35952	Aegean Macedonian	wd:Q8209	Cyrillic script

Rysunek 10.21: Zapytanie o języki i ich systemy pisma różne od łacińskiego

tzn. łączące w sobie mechanizmy relacyjnej bazy danych (RDBMS), obiektowo-relacyjnej bazy danych (ORDBMS), bazy danych opartej na dokumentach (zwłaszcza XML i JSON), grafowej bazy danych (GDB) i bazy danych opartej na modelu klucz–wartość.

Tablica 10.1: Zestawienie wybranych implementacji SPARQL

Projekt	**Krótki opis**
Blazegraph[9]	Bardzo wydajna grafowa baza danych wykorzystywana przez Wikidata. Obsługuje do 50 miliardów węzłów na pojedynczej maszynie. Dostępna na licencji GNU GPL v2.
Amazon Neptune[10]	Zarządzana grafowa baza danych opublikowana przez Amazon jaku usługa webowa będąca częścią AWS (Amazon Web Services). Częściowo oparta na bazie Blazegraph.
AllegroGraph[11]	Magazyn trójek RDF o zamkniętym kodzie źródłowym. Stosowany komercyjnie i przez Departament Obrony Stanów Zjednoczonych. Komponent projektu TwitLogic, który importuje dane Semantic Web do Twittera.
Eclipse RDF4J[12]	Magazyn trójek, zwany dawniej OpenRDF Sesame, stworzony w ramach projektu Eclipse dla potrzeb aplikacji w Javie. Dostępny na licencji wolnej EPL.

[9] `https://blazegraph.com/`
[10] `https://aws.amazon.com/neptune/`
[11] `https://allegrograph.com/`
[12] `https://rdf4j.org/`

Tablica 10.1: Zestawienie wybranych implementacji SPARQL

Projekt	**Krótki opis**
Apache Jena[13] z ARQ	Platforma stworzona w firmie Apache, podobna do RDF4J, jednak zapewnia wsparcie dla OWL (Web Ontology Language).
Cray Urika-GD	Aplikacja na komputery Cray do wykrywania i analizowania wzorów i zależności danych w ogromnych zasobach informacji (*Big Data*).
KAON2	Karlsruhe ontology – infrastruktura ontologii opracowana na Uniwersytecie w Karlsruhe (Niemcy).
MarkLogic	Wielomodelowa baza danych NoSQL opracowana w amerykańskiej firmie MarkLogic Corporation do przechowywania, zarządzania i przeszukiwania dokumentów w formacie XML i JSON oraz danych semantycznych w formie trójek RDF.
Mulgara	Magazyn trójek RDF na licencji otwartej opracowany jako odprysk projektu Kowari w firmie Tucana Inc.
NitrosBase	Rosyjska wielomodelowa baza danych opracowana w Centrum Innowacji Skolkowo.
Ontotext GraphDB	Komercyjna baza danych NoSQL zbudowana na podstawie architektury EDF4J.
Oracle DB Enterprise Spatial & Graph	Wielomodelowa baza danych wspierana przez Oracle Corporation. Jako Oracle Spatial & Graph wspiera przeszukiwanie trójek RDF.
RDFLib Python library	Implementacja RDF dla języka Python.
Virtuoso	Wielomodelowa baza danych opracowana w języku C przez firmę OpenLink Software, dostępna na licencji GPLv2. Używana w projekcie DBpedia.

10.4. DBpedia

Wikidata to nie jedyna baza danych Wikipedii. Jedną z bardziej znanych alternatywnych implementacji jest DBpedia [11]. Projekt ten powstał we współpracy Wolnego Uniwersytetu Berlińskiego i Uniwersytetu Lipskiego z firmą OpenLink Software, a obecnie jest wspierany przez społeczność zrzeszoną w DBpedia.org[14]. Organizacja ta publikuje swój własny zbiór danych (pierwsze wydanie w 2007 r.) na licencjach wolnych (CC-BY-SA).

Co kilka miesięcy wydawana jest najmniejsza wersja zbioru danych *DBpedia Latest Core Release*, zwana również *Tiny Diamond*. Zawiera ona ok. 1 miliarda

[13] `https://jena.apache.org/`
[14] `https://www.dbpedia.org/`

trójek, w tym dane zebrane z artykułów i infoboksów Wikipedii w angielskiej wersji językowej oraz etykiety i streszczenia z największych edycji językowych Wikipedii. Ponadto DBpedia uruchomiła bota o nazwie Marvin, który nieustannie przegląda 140 wydań językowych Wikipedii (pełnotekstowe artykuły), dane udostępniane przez Wikidata oraz repozytorium otwartych zasobów Wikipedia Commons, znajdując 5,5 tys. trójek na sekundę. W wersji prototypowej BETA działa baza danych ontologii DBpedia Archivo, w której można oceniać w czterogwiazdkowej skali ponad 1700 ontologii OWL. Docelowym wydaniem ma być *Largest Diamond*, który obecnie opisuje 220 milionów encji za pomocą 1,45 miliarda tripletów.

Wszystkie zbiory danych są udostępniane przez „szynę danych" (`https://databus.dbpedia.org`) i mogą być przez zainteresowanych ładowane do własnych repozytoriów w postaci plików w formatach TXT, JSON, OWL (RDF/XML), TTL (*Turtle – Terse RDF Triplet Language*) i NT (*N-Triples*).

10.4.1. Ontologia DBpedia (DBO)

„Sercem" DBpedii jest ontologia, która wyróżnia ponad 700 klas (takich jak miejsca, osoby, praca, gatunki, organizacje) tworzących hierarchię podporządkowania i zawiera ponad 4 miliony instancji (tabl. 10.2). „Silnikiem" ontologii jest baza danych Virtuoso firmy OpenLink.

Tablica 10.2: Zawartość ontologii DBpedii
[za `https://www.dbpedia.org/resources/ontology/`]

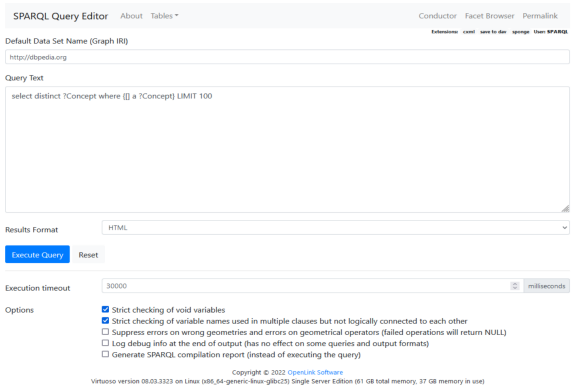
Klasy	**Instancje**
Miejsca	967 491
Osoby	1 592 912
Praca	552 115
Gatunki	190 369
Organizacje	317 867
Inne	1 207 664
Razem	4 828 418

Ontologię DBpedii można załadować do lokalnego repozytorium przez „szynę danych" (tak jak i inne zbiory danych DBpedii) lub korzystać bezpośrednio z jej zasobów poprzez punkt końcowy SPARQL (`https://dbpedia.org/sparql`). Strona dostępowa (rys. 10.22) umożliwia edycję własnego zapytania SPARQL. Wyniki są wyświetlane na osobnej stronie.

10.4.2. DBpedia SPARQL

Dialekt SPARQL używany przez bazę danych Virtuoso dla potrzeb DBpedii jest nieco odmienny od SPARQL Wikidanych. Po pierwsze zamiast nieczytelnego predykatu „wdt:31|wdt:P31/wdt:P279*" stosuje się prosty predykat „a". Po

drugie zamiast identyfikatorów Pxxx i Qxxx podaje się nazwy właściwości i nazwy klas poprzedzone definiowalnymi prefiksami. W tym dialekcie zapytanie o wszystkie języki (instancje klasy „Language") ma postać następującą:



Rysunek 10.22: Strona dostępowa ontologii DBpedii

```
PREFIX owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>
SELECT ?name
WHERE {
  ?item a owl:Language.
  ?item dbpprop:name ?name .
FILTER(lang(?name) = "en")}
```

Nałożenie filtru (lang(?name) = "en") wymusza tu podawanie tylko angielskich nazw języków. Oznaczenie języka pojawia się w wyświetlanych wynikach po znaku @ (rys. 10.23). Brak tego filtra dałby wyniki we wszystkich obsługiwanych przez DBpedię językach świata.

Aby sprawnie formułować zapytania SPARQL trzeba znać przynajmniej strukturę klas i listę właściwości klas. Strukturę klas można przeglądać na stronie: `http://mappings.dbpedia.org/server/ontology/classes/` (rys. 10.24). Wchodząc w głąb struktury, można zobaczyć opis każdej klasy (rys. 10.25) i przez to odczytać, jakie ma dostępne właściwości.

Listę wszystkich właściwości stosowanych we wszystkich klasach (rys. 10.26) można uzyskać przez zapytanie:

```
SELECT DISTINCT ?pred WHERE {
  ?pred a rdf:Property
} ORDER BY ?pred
```

Rysunek 10.23: Wyniki zapytania o wszystkie języki (tylko nazwy angielskie)

Rysunek 10.24: Struktura klas ontologii DBpedii

Employer (Show in class hierarchy)

Label (ja): 雇用者
Label (de): Arbeitgeber
Label (fr): Employeur
Label (es): Empleador
Label (ur): آجر
Label (ga): fostóir
Label (nl): werkgever
Label (en): Employer
Label (da): arbejdsgiver
Label (el): Εργοδότης
Comment (en): a person, business, firm, etc, that employs workers.
Comment (de): Arbeitgeber ist, wer die Arbeitsleistung des Arbeitnehmers kraft Arbeitsvertrages fordern kann und das Arbeitsentgelt schuldet.
Comment (el): άτομο, επιχείρηση, οργανισμός, κλπ που προσλαμβάνει εργαζόμενους.
Comment (ur): آجر وہ ہے جو ملازمت کے معاہدے کی وجہ سے ملازم سے کام کا مطالبہ کر سکتا ہے اور جس پر اجرت واجب الادا ہے۔
Super classes: Agent

Properties on *Employer*:

Name	Label	Domain	Range	Comment
age (edit)	age	Agent	*xsd:integer*	
artPatron (edit)	patron (art)	Agent	Artist	An influential, wealthy person who supported an artist, craftsman, a scholar or a noble.. See also
championships (edit)	championships	Agent	*xsd:nonNegativeInteger*	
denomination (edit)	denomination	Agent	owl:Thing	Religious denomination of a church, religious school, etc. Examples: Haredi_Judaism, Sunni_Islam, Seventh-day_Adventist_Church, Non-Denominational, Multi-denominational, Non-denominational_Christianity
discipline (edit)	discipline	Agent	owl:Thing	
generalCouncil (edit)	general council	Agent	TermOfOffice	
hometown (edit)	home town	Agent	Settlement	
ideology (edit)	ideology	Agent	Ideology	
juniorSeason (edit)	junior season	Agent	owl:Thing	
managerSeason (edit)	manager season	Agent	owl:Thing	
nationalSelection (edit)	national selection	Agent	owl:Thing	
owns (edit)	owns	Agent	Thing	Used as if meaning: has property rights over

Rysunek 10.25: Opis klasy w ontologii DBpedii wraz z listą właściwości tej klasy

10.5. Wiarygodność danych Wikipedii

„Wikipedia nie jest wiarygodnym źródłem dla cytowania w innych miejscach Wikipedii". Takie sformułowanie można znaleźć na stronach Wikipedii[15]. Ponieważ

[15] `https://en.wikipedia.org/wiki/Wikipedia:Wikipedia_is_not_a_reliable_source`

pred
http://dbpedia.org/ontology/aSide
http://dbpedia.org/ontology/abbeychurchBlessing
http://dbpedia.org/ontology/abbeychurchBlessingCharge
http://dbpedia.org/ontology/abbreviation
http://dbpedia.org/ontology/ableToGrind
http://dbpedia.org/ontology/absoluteMagnitude
http://dbpedia.org/ontology/abstentions
http://dbpedia.org/ontology/abstract
http://dbpedia.org/ontology/academicAdvisor
http://dbpedia.org/ontology/academicDiscipline
http://dbpedia.org/ontology/academyAward

Rysunek 10.26: Alfabetyczna lista wszystkich właściwości DBpedii

każdy może edytować artykuły Wikipedii, więc można w niej znaleźć wiele błędów, nieścisłości lub wręcz fałszywych informacji, które pozostają tam dostępne przez tygodnie, miesiące, a nawet lata[16]. Dlatego dobrze napisany artykuł musi podawać źródła oryginalne, spoza Wikipedii. Jeżeli czytelnik chce dalej podawać informacje znalezione w Wikipedii, to powinien je sprawdzić u źródeł[17].

Artykuły Wikipedii nie są poddawane procesowi formalnej recenzji, ale każdy użytkownik może być zarówno edytorem, jak i recenzentem. Wikipedia ma zdefiniowanych wiele szablonów, które są przeznaczone do wstawiania uwag (np. „Citation needed") do tekstu. Uwagi te są widoczne dla czytelników i często pełnią rolę ostrzeżeń o potencjalnych brakach wiarygodności.

Fundacja MediaWiki otworzyła projekt dotyczący wiarygodności Wikipedii[18]. Na liście uczestników tego projektu znajduje się już ponad 230 użytkowników[19]. Prowadzą oni dyskusje nad wiarygodnością poszczególnych artykułów i oznaczają wykryte w nich problemy (zwłaszcza w zakresie cytowania w nich wiarygodnych źródeł).

Powstało wiele narzędzi do automatycznego sprawdzania wiarygodności Wikipedii [13]. Analizują one wprowadzane poprawki i usunięcia treści pod kątem czasu wprowadzenia poprawki, stopień zaufania do edytorów etc. Ciekawym narzędziem jest WikiScanner, stworzony przez Virgila Griffitha, dzięki któremu można sprawdzić, spod jakiego adresu IP wprowadzono daną poprawkę, czy nie były to przypadkiem adresy komputerów służbowych firm [6].

Zarówno Wikidata, jak i DBpedia opierają się na Wikipedii. Zautomatyzowane narzędzia przeglądają zawartość Wikipedii, parsują treść artykułów i wydobywają z niej wiedzę. Najczęściej parsowaniu poddawane są szablony infoboks. Tworzone są trójki <encja> <właściwość> <wartość>, które są prze-

[16] `https://en.wikipedia.org/wiki/Wikipedia:List_of_hoaxes_on_Wikipedia`

[17] Dobry artykuł na temat wiarygodności Wikipedii: `https://en.wikipedia.org/wiki/Reliability_of_Wikipedia`

[18] `https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Reliability`

[19] Dane z 30 czerwca 2022

kształcane w stwierdzenia <podmiot> <predykat> <przedmiot> i zapisywane w bazie danych.

Oczywiste jest, że wiarygodność tak wyodrębnionych stwierdzeń zależy od wiarygodności źródła, czyli Wikipedii, jednak proces zastosowany przez Wikidata i DBpedia może zwiększyć tę wiarygodność. Stosuje się kilka mechanizmów kontrolnych:

- Jako źródła są wykorzystywane artykuły na ten sam temat z różnych wersji językowych [1]. Infoboksy sugerują, do jakiej klasy należy nowo tworzona encja. W przypadku różnych wskazań przeprowadzane jest głosowanie.
- Z tekstu artykułu wyodrębniane są frazy identyfikujące encję i dopasowywane do nazwy klasy kandydackiej [5].
- Bada się stabilność i spójność nazw właściwości, sprawdza się zgodność typów danych wartości z ustalonym schematem [12].

Wiarygodność Wikipedii jest tylko jednym z aspektów jej jakości. Inne aspekty to m.in. aktualność, spójność, odporność na wandalizm. Jest to temat na osobne opracowanie.

10.6. Podsumowanie

W tym rozdziale pokazano, jak Wikipedia z wielojęzycznego zbioru artykułów stała się bazą wiedzy dla ontologii. Uwagę skupiono na bazie wiedzy Wikidata i ontologii DBpedia. Pokazano na rozbudowanym przykładzie, jak konstruować zapytania w języku SPARQL do tych dwóch centrów wiedzy dostępnych w Internecie. Poruszono też problem wiarygodności tej wiedzy.

W żadnym wypadku treść rozdziału nie wyczerpuje bogactwa poruszanej tematyki, ale ma stanowić zachętę do głębszego zapoznania się z tą formą Semantic Web. W przekonaniu autora zautomatyzowane pozyskiwanie klas i encji z Semantic Web może podnieść procesy analizy problemu i projektowania systemów informatycznych na wyższy poziom jakości.

Bibliografia

[1] Aprosio A.P., Giuliano C., Lavelli A. *Automatic mapping of Wikipedia templates for fast deployment of localised DBpedia datasets.* In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, pp. 1–8. 2013.

[2] Barrón-Cedeno A., et al. *A comparison of approaches for measuring cross-lingual similarity of wikipedia articles.* In *European Conference on Information Retrieval*, pp. 424–429. Springer, 2014.

[3] Bollacker K., et al. *Freebase: a collaboratively created graph database for structuring human knowledge.* In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250. 2008.

[4] Denoyer L., Gallinari P. *The wikipedia xml corpus.* In *International Workshop of the Initiative for the Evaluation of XML Retrieval*, pp. 12–19. Springer, 2006.

[5] Exner P., Nugues P. *Using Semantic Role Labeling to Extract Events from Wikipedia.* In *DeRiVE@ ISWC*, pp. 38–47. 2011.

[6] Fallis D. *Toward an epistemology of Wikipedia. Journal of the American Society for Information science and Technology*, 59(10):1662–1674, 2008.

[7] Flickinger D., Oepen S., Ytrestøl G. *Wikiwoods: Syntacto-semantic annotation for english wikipedia.* 2010.

[8] Hellmann S., Lehmann J., Auer S. *Learning of OWL class descriptions on very large knowledge bases. International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):25–48, 2009.

[9] Mahdisoltani F., Biega J., Suchanek F. *Yago3: A knowledge base from multilingual wikipedias.* In *7th biennial conference on innovative data systems research.* CIDR Conference, 2014.

[10] Margaretha E., Lüngen H. *Building linguistic corpora from Wikipedia articles and discussions. Journal of Language Technology and Computational Linguistics. Special issue on building and annotating corpora of computer-mediated communication. Issues and challenges at the interface between computational and corpus linguistics*, 29(2):59–82, 2014.

[11] Matuszek C., et al. *An introduction to the syntax and content of Cyc. UMBC Computer Science and Electrical Engineering Department Collection*, 2006.

[12] Pellissier Tanon T., Kaffee L.A. *Property label stability in wikidata: evolution and convergence of schemas in collaborative knowledge bases.* In *Companion Proceedings of the The Web Conference 2018*, pp. 1801–1803. 2018.

[13] Sarabadani A., Halfaker A., Taraborelli D. *Building automated vandalism detection tools for Wikidata.* In *Proceedings of the 26th International Conference on World Wide Web Companion*, pp. 1647–1654. 2017.

[14] Vrandecic D. *The rise of Wikidata. IEEE Intelligent Systems*, 28(4):90–95, 2013.

[15] Woodward D. *Extraction and Visualization of Temporal Information and Related Named Entities from Wikipedia*, 2001.

Rozdział 11

Metody ekstrakcji ustrukturalizowanej treści z Wikipedii

Jarosław Kuchta[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
qhta@eti.pg.edu.pl

Abstrakt

Wikipedia jest od dawna przedmiotem zainteresowania badaczy. Jednym z obszarów zainteresowania jest pozyskiwanie wiedzy z treści Wikipedii, a to wymaga parsowania tekstu artykułów. W tym rozdziale przedstawiono analizę porównawczą różnych możliwości parsowania treści Wikipedii, wykazując problemy, z jakimi muszą się mierzyć autorzy parserów. Dzięki temu można zrozumieć, dlaczego proces wydobywania wiedzy z Wikipedii jest trudny.

Słowa kluczowe: Wiki, Wikipedia, parser, format, problem

11.1. Wprowadzenie

Wikipedia jest największym, wielojęzycznym, otwartym źródłem wiedzy encyklopedycznej. Ale cóż to oznacza? Do niedawna największą encyklopedią

w historii ludzkości była chińska encyklopedia cesarza Yongle, powstała w XV w., za czasów dynastii Ming. Zajmowała ona prawie 23 tys. zwojów zgromadzonych w 11 tys. woluminów [2] i podzielonych na 900 tys. stron. Wikipedia pobiła ten rekord w 2007 r. [7]. W 2020 r. Wikipedia z sumą 6 231 239 artykułów w języku angielskim i 55 632 716 haseł w 315 językach została ogłoszona największą encyklopedią online w księdze rekordów Guinnessa. Dla porównania druga pod względem wielkości chińska encyklopedia Baidu Baike w 2022 r. zawierała 25 mln haseł[1]. Otwartość Wikipedii oznacza, że każdy człowiek na świecie, który tylko ma dostęp do Internetu, może wprowadzać i edytować hasła Wikipedii. Taka otwartość mogłaby zakwestionować wiarygodność informacji, ale z drugiej strony stanowi też siłę Wikipedii – informacje niewiarygodne są szybko usuwane przez innych autorów, a wiarygodność polega na konsensusie społeczności. I znowu dla porównania Baidu Baike jest kontrolowana przez rząd ChRL i oskarżana o cenzurę [9, 13].

Wielojęzyczność oznacza nie tylko wspomniane już 315 języków, w których edytowana jest Wikipedia, ale także to, że każda edycja lokalna to osobny zbiór artykułów. Istnieją liczne powiązania pomiędzy artykułami na ten sam temat w różnych językach, ale nie ma gwarancji, że dany temat jest opisany we wszystkich językach. A nawet jeśli jest, to nie ma identyczności przekazywanych treści w różnych językach. Stąd wynika pewien problem dla pozyskiwania wiedzy – trzeba się zastanowić, które wydanie Wikipedii uznać za wiodące. Problem ten jest rozwiązywany w DBpedii, ale to materiał na osobny artykuł. W tym rozdziale przyjmiemy, że badaniu podlega anglojęzyczne wydanie Wikipedii.

11.1.1. Trochę historii

Najszersze zainteresowanie wydobywaniem wiedzy bezpośrednio z Wikipedii przypada na lata 2006–2014, chociaż trzeba przyznać, że opublikowano niewiele prac na ten temat. W 2006 r. Denoyer i Gallinari [3] zasygnalizowali stworzenie korpusu XML stworzonego z 659 388 dokumentów przez wyodrębnienie jednostek na podstawie kategorii artykułów. Analizie poddali 7 wersji językowych Wikipedii: angielską, francuską, niemiecką, holenderską, hiszpańską, chińską i japońską. W latach 2008–2010 Ytrestøl, Flickinger i Oepen [8, 14] wykonali eksperyment z dodawaniem adnotacji do korpusu lingwistycznego PropBank[2]. Dla potrzeb eksperymentu stworzyli narzędzie WikiWoods wykorzystujące parser Athena[3]. W 2010 r. Lange, Böhm i Naumann [10] przedstawili wydobywanie ustrukturyzowanej informacji z Wikipedii w celu wypełniania infoboksów, głównych tabelek na stronach zawierających podstawowe dane dotyczące tematu artykułu. Dohrn i Riehle w 2011 r. [5] opisali tworzenie korpusów lingwistycznych XML przy użyciu parsera Sweble. Ten sam parser zastosowali Margaretha i Lüngen w 2014 r. [11]. Sweble jest otwartoźródłowym parserem tekstu Wiki napisanym w Javie i udostępnionym w repozytorium GitHub[4].

[1] `https://meta.wikimedia.org/wiki/List_of_largest_wikis`
[2] `https://propbank.github.io/`
[3] Brak dokładniejszej informacji co do tego parsera.
[4] `https://github.com/sweble/sweble-wikitext`

11.1.2. Istniejące parsery

Na stronie MediaWiki[5] można znaleźć listę parserów tekstu Wikipedii analizujących znaczniki Wiki. W tablicy 11.1 przedstawiono te parsery, które są w pełni funkcjonalne i cały czas podtrzymywane. Spośród nich największą popularność osiągnęły dwa pierwsze: Parsoid i Sweble.

Tablica 11.1: Parsery Wikipedii

Nazwa	**Język implementacji**	**Strona projektu**
Parsoid	PEG/PHP	Tłumaczy tekst Wiki na HTML. Współpracuje z edytorem wizualnym Wikipedii[6].
Sweble	Java	Bardzo dokładny. Opisywany w artykułach naukowych.
MWParserFromHell	Python	Łatwa w użyciu i „skandalicznie" silna biblioteka Pythona[7].
wtf_wikipedia	JavaScript	Tłumaczy tekst wiki na JSON[8].
wikitextparser	Python	Udostępnia metody nawigacji do elementów konstrukcyjnych, takich jak sekcje, tabele, łącza[9].
mediawiki-parser	Haskell	Stanowi podstawę potoku ekstrakcji informacji dla projektu NIST *TREC Complex Answer Retrieval* [4].
Wikiapi	JavaScript	Podaje listę członków kategorii i rozwija szablony[10].

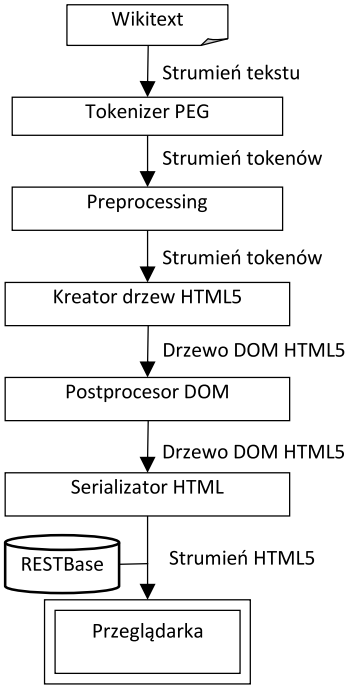
Parsoid

Parsoid to biblioteka systemu MediaWiki wykorzystywana do tłumaczenia tekstu Wiki na HTML i z HTML na Wiki. Pierwsza, starsza wersja Parsoidu została napisana w języku JavaScript z wykorzystaniem Node.js i została wdrożona na serwerach Wikimedii w 2012 r. W 2019 r. Parsoid został przepisany na PHP i zintegrowany z rdzeniem MediaWiki.

Schemat działania Parsoidu przedstawiono na rys. 11.1. Tekst Wiki jest przetwarzany przez tokenizer oparty na PEG (*parsing expression grammar*) na strumień tokenów, który w preprocessingu jest podawany wielu transformacjom, np. wykrywane są listy Wiki, znaczniki kursywy i pogrubienia, rozwijane są

[5] `https://www.mediawiki.org/wiki/Alternative_parsers`
[6] `https://www.mediawiki.org/wiki/Parsoid`
[7] `https://mwparserfromhell.readthedocs.io/en/latest/`
[8] `https://www.npmjs.com/package/wtf_wikipedia`
[9] `https://github.com/5j9/wikitextparser`
[10] `https://github.com/kanasimi/wikiapi`

szablony. Po tych transformacjach strumień tokenów jest podawany na wejście kreatora drzew HTML5. Tutaj następuje rozwiązanie zagnieżdżenia konstrukcji Wiki (np. parowanie znaczników początku i końca tabeli). Tworzone drzewo HTML5 jest poddawane postprocessingowi (w trakcie którego np. są formowane akapity), a następnie serializowane do tekstu HTML5. Tekst HTML5 jest udostępniany dla przeglądarki Wikipedii lub innego klienta Parsoidu. W celu przyspieszenia działania przeglądarki tekst HTML5 jest buforowany w RESTBase.



Rysunek 11.1: Schemat działania Parsoidu

Aplikacja Parsoid oferuje własny interfejs API, który jest wewnętrznie wykorzystywany w MediaWiki (nie jest wprost udostępniany na zewnątrz). Jest bezpośrednio wywoływany przez edytor wizualny Wiki (VisualEditor[11]), projekt Structured Discussions[12] (zwany wcześniej Flow) ułatwiający prowadzenie dyskusji przez edytorów Wikipedii, narzędzie Content Translation[13] ułatwiające tłumaczenie artykułów między wersjami językowymi Wikipedii i inne aplikacje.

[11] `https://www.mediawiki.org/wiki/VisualEditor`
[12] `https://www.mediawiki.org/wiki/Structured_Discussions`
[13] `https://www.mediawiki.org/wiki/Content_translation`

Sweble

Sweble jest alternatywnym parserem Wikitekstu opracowanym na Uniwersytecie Fryderyka–Aleksandra w Erlangen i Norymberdze. Jego oryginalnymi autorami są Hannes Dohrn i Dirk Riehle. Kod Parsera w języku Java został przez nich udostępniony jako otwarty na GitHubie[14]. Sweble w odróżnieniu od Parsoidu daje wynik w postaci drzewa składni abstrakcyjnej (AST – *abstract syntax tree*). Podobnie jak w Parsoidzie, proces tokenizacji został oparty na gramatyce PEG. Kod źródłowy parsera został w dużej części wygenerowany z gramatyki PEG przy wykorzystaniu generatora parserów Rats! [12]. Z założenia struktura AST musi przechowywać nie tylko Wikitext Object Model [1], ale też informację o tekście wejściowym, aby można było dokonać konwersji wstecznej.

Innym założeniem jest to, aby parser działał (nie zawieszał się i nie przerywał pracy) przy błędnie sformatowanym tekście źródłowym (np. niesparowanych znacznikach kursywy i wytłuszczenia). Dohrn i Riehle [6] zwrócili uwagę na problemy błędów w tekście źródłowym, Problemy te mogą się nie ujawnić przy bezpośrednim tłumaczeniu z Wikikodu na HTML – Parsoid może po prostu generować błędny HTML (np. z niesparowanymi znacznikami), a przeglądarka sobie poradzi. Jeśli jednak wynikiem parsowania jest drzewo AST, to potrzebny jest zaawansowany preprocessing dla korygowania błędnego kodu źródłowego.

W niniejszym rozdziale rozwiniemy ten problem w kontekście możliwości skorzystania z innych form tekstu artykułów Wikipedii niż kod Wiki (a konkretnie drzewa półskompilowanego oraz tekstu HTML). Przedstawimy też inny problem – ogromnej liczby szablonów Wikipedii w kontekście analizy semantycznej artykułów.

Najpierw jednak trochę podstawowych informacji kontekstowych o Wikipedii i innych projektach Wikimedii.

11.1.3. Wiki, Wikipedia, Wikimedia, MediaWiki

Podstawą Wikipedii jest format Wiki, który został wynaleziony przez Warda Cunninghama i po raz pierwszy zastosowany w WikiWikiWeb[15] w 1995 r. Jest to format alternatywny do HTML, przeznaczony do łatwego pisania tekstu dla WWW. Składnia Wiki jest opisana m.in. na stronie Wikipedii Help: Wikitext. Najważniejsze cechy *wikitekstu* to:

- oznaczanie wytłuszczenia i kursywy poprzez sekwencje apostrofów;
- oznaczanie odrębnych akapitów za pomocą podwójnych znaków nowej linii;
- oznaczanie hiperłączy w tekście ujętym w podwójne nawiasy kwadratowe;
- stosowanie różnorodnych szablonów ułatwiających włączanie określonej treści.

Przykład jest pokazany w tablicy 11.2.

[14] `https://github.com/sweble/sweble-wikitext`
[15] `http://wiki.c2.com/?WikiHistory`

Tablica 11.2: Przykład formatowania Wiki i HTML
[z `https://en.wikipedia.org/wiki/Wiki%23Editing`]

Składnia Wiki	**Składnia HTML**	**Prezentowany tekst**
"Take some more [[tea]]," the March Hare said to Alice, very earnestly.	"Take some more tea," the March Hare said to Alice, very earnestly.	"Take some more tea," the March Hare said to Alice, very earnestly.
"I've had '''nothing''' yet," Alice replied in an offended tone, "so I can't take more."	"I've had nothing yet," Alice replied in an offended tone, "so I can't take more."	"I've had **nothing** yet," Alice replied in an offended tone, "so I can't take more."
"You mean you can't take "less"," said the Hatter. "It's very easy to take "more" than nothing."	"You mean you can't take less," said the Hatter. "It's very easy to take more than nothing."	"You mean you can't take *less*," said the Hatter. "It's very easy to take *more* than nothing."

Artykuły Wikipedii to nie tylko Wiki, lecz także treść multimedialna zarządzana przez fundację Wikimedia Foundation[16] zwaną w skrócie Wikimedia. Fundacja Wikimedia prowadzi kilka projektów, z których najważniejsze to:

- Wikipedia – wielojęzyczna encyklopedia online;
- Wikimedia Commons – repozytorium multimediów;
- Wiktionary – wielojęzyczny słownik i tezaurus online;
- Wikiquote – zbiór cytatów w różnych językach;
- Wikisource – biblioteka cyfrowa;
- Wikidata – baza wiedzy;
- Wikibooks – repozytorium darmowych podręczników;
- Wikiversity – repozytorium darmowych ćwiczeń i szkoleń;
- Wikinews – otwarty serwis informacyjny;
- Wikispecies – taksonomiczny katalog gatunków;
- Wikivoyage – przewodnik turystyczny;
- Meta-Wiki – strona do koordynacji wszystkich projektów Wikimedii;

Fundacja Wikimedia zarządza treścią tekstową i multimedialną za pomocą oprogramowania zwanego Mediawiki. Mediawiki jest oprogramowaniem typu *open-source*, napisanym w języku PHP i wydanym na licencji GPL-2.0. Jego podstawowe funkcje są obejmują:

- wsparcie lokalizacji w różnych językach;
- wstawianie grafik i szablonów do artykułów;
- prowadzenie stron dyskusji;
- tworzenie list stron;
- zapamiętywanie ostatnich zmian i historii wraz z rejestrami;

[16] `https://en.wikipedia.org/wiki/Wikimedia_Foundation`

- wprowadzanie rozszerzeń API (m.in. Interwiki);
- możliwość stosowania skórek i interfejsu edycyjnego;
- wsparcie dla przestrzeni nazw i kategorii;
- możliwość nadawania uprawnień grupowych.

Wykorzystanie API Mediawiki stanowi najprostszy sposób pobierania i przetwarzania treści Wikipedii.

11.2. API Mediawiki

Każdy projekt i repozytorium Mediawiki udostępnia pewne usługi webowe przez punkt końcowy (*endpoint*) dostępny jako "…/w/api.php". Przykłady przedstawiono w tablicy 11.3.

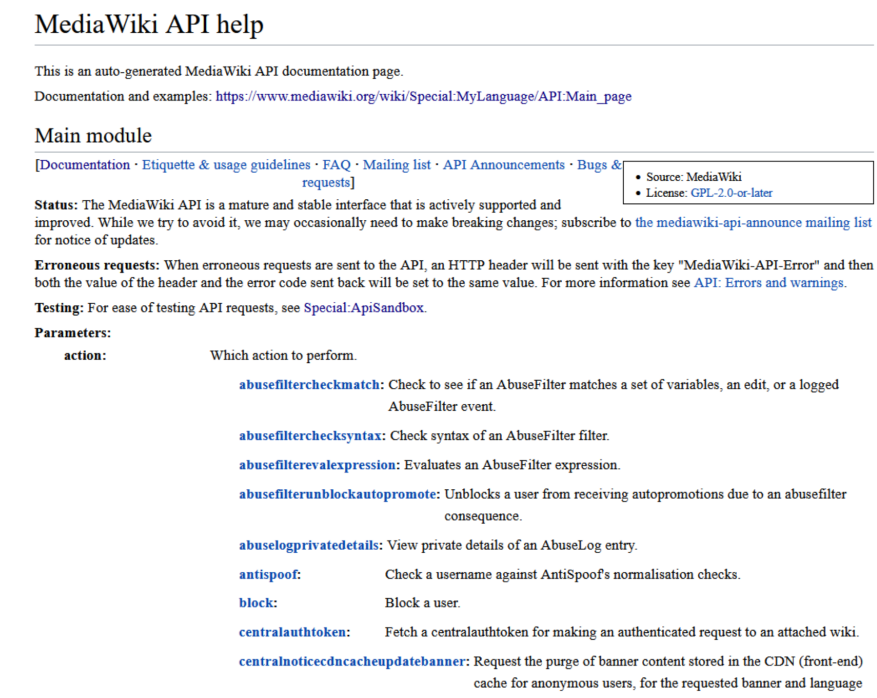
Tablica 11.3: Przykłady punktów końcowych projektów Wikimedia [z `https://www.mediawiki.org/wiki/API:Main_page/pl`]

API Wiki	**API Endpoint**
MediaWiki API	`https://www.mediawiki.org/w/api.php`
Meta-Wiki API	`https://meta.wikimedia.org/w/api.php`
English Wikipedia API	`https://en.wikipedia.org/w/api.php`
Polish Wikipedia API	`https://pl.wikipedia.org/w/api.php`
Wikimedia Commons API	`https://commons.wikimedia.org/w/api.php`
Test Wiki API	`https://test.wikipedia.org/w/api.php`

Proste wywołanie punktu końcowego w przeglądarce powoduje wyświetlenie automatycznie wygenerowanej strony dokumentującej API danego punktu (przykład na rys. 11.2). Dokumentacja pokazuje, jakie akcje (usługi) można wykonać w danym punkcie. Przykładowo w punkcie końcowym Wikipedii można wykonać akcje kwerendy (*query*) i parsowania (*parse*). Wchodząc głębiej w tę stronę, dochodzi się do dokumentacji poszczególnych akcji. Można się dowiedzieć np., że akcja *query* umożliwia pozyskanie m.in.:

- listy kategorii, do których należy dana strona;
- wszystkich plików multimedialnych zawartych na danej stronie;
- wszystkich hiperłączy wychodzących z danej strony;
- wszystkich przekierowań do danej strony;
- wszystkich szablonów użytych na danej stronie.

Na dole strony dokumentującej akcję zazwyczaj znajdują się przykłady wykorzystania wraz z hiperłączami do strony piaskownicy (*sandbox*), gdzie można przetestować zarówno te przykłady, jak i inne, dowolnie wymyślone (rys. 11.3). *API sandbox* (rys. 11.4) umożliwia ustawienie parametrów wywołania akcji, a po wysłaniu żądania odczytanie wyniku odesłanego w formacie JSON (rys. 11.5).



MediaWiki API help

This is an auto-generated MediaWiki API documentation page.

Documentation and examples: https://www.mediawiki.org/wiki/Special:MyLanguage/API:Main_page

Main module

[Documentation · Etiquette & usage guidelines · FAQ · Mailing list · API Announcements · Bugs & requests]

- Source: MediaWiki
- License: GPL-2.0-or-later

Status: The MediaWiki API is a mature and stable interface that is actively supported and improved. While we try to avoid it, we may occasionally need to make breaking changes; subscribe to the mediawiki-api-announce mailing list for notice of updates.

Erroneous requests: When erroneous requests are sent to the API, an HTTP header will be sent with the key "MediaWiki-API-Error" and then both the value of the header and the error code sent back will be set to the same value. For more information see API: Errors and warnings.

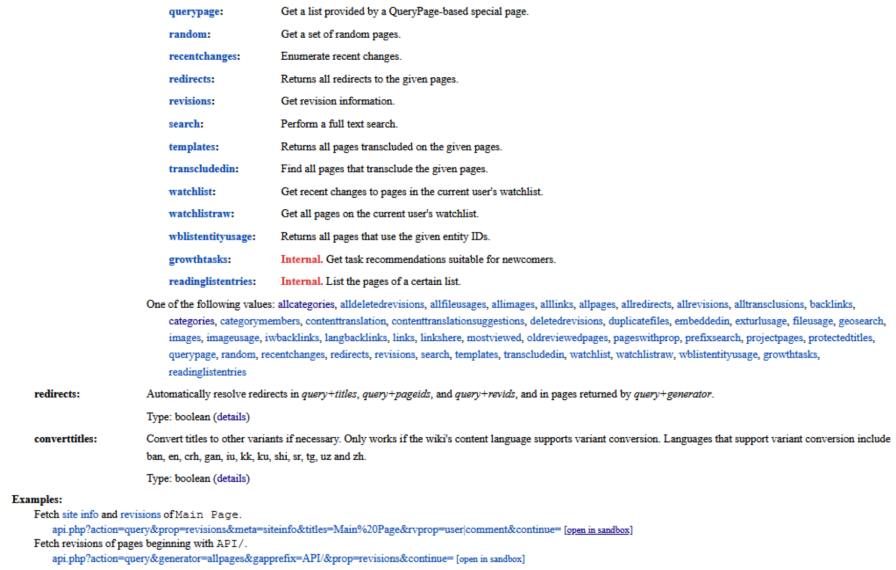
Testing: For ease of testing API requests, see Special:ApiSandbox.

Parameters:

action: Which action to perform.

- **abusefiltercheckmatch:** Check to see if an AbuseFilter matches a set of variables, an edit, or a logged AbuseFilter event.
- **abusefilterchecksyntax:** Check syntax of an AbuseFilter filter.
- **abusefilterevalexpression:** Evaluates an AbuseFilter expression.
- **abusefilterunblockautopromote:** Unblocks a user from receiving autopromotions due to an abusefilter consequence.
- **abuselogprivatedetails:** View private details of an AbuseLog entry.
- **antispoof:** Check a username against AntiSpoof's normalisation checks.
- **block:** Block a user.
- **centralauthtoken:** Fetch a centralauthtoken for making an authenticated request to an attached wiki.
- **centralnoticecdncacheupdatebanner:** Request the purge of banner content stored in the CDN (front-end) cache for anonymous users, for the requested banner and language

Rysunek 11.2: Przykład strony dokumentującej API Wikipedii



- **querypage:** Get a list provided by a QueryPage-based special page.
- **random:** Get a set of random pages.
- **recentchanges:** Enumerate recent changes.
- **redirects:** Returns all redirects to the given pages.
- **revisions:** Get revision information.
- **search:** Perform a full text search.
- **templates:** Returns all pages transcluded on the given pages.
- **transcludedin:** Find all pages that transclude the given pages.
- **watchlist:** Get recent changes to pages in the current user's watchlist.
- **watchlistraw:** Get all pages on the current user's watchlist.
- **wblistentityusage:** Returns all pages that use the given entity IDs.
- **growthtasks:** Internal. Get task recommendations suitable for newcomers.
- **readinglistentries:** Internal. List the pages of a certain list.

One of the following values: allcategories, alldeletedrevisions, allfileusages, allimages, alllinks, allpages, allredirects, allrevisions, alltransclusions, backlinks, categories, categorymembers, contenttranslation, contenttranslationsuggestions, deletedrevisions, duplicatefiles, embeddedin, exturlusage, fileusage, geosearch, images, imageusage, iwbacklinks, langbacklinks, links, linkshere, mostviewed, oldreviewedpages, pageswithprop, prefixsearch, projectpages, protectedtitles, querypage, random, recentchanges, redirects, revisions, search, templates, transcludedin, watchlist, watchlistraw, wblistentityusage, growthtasks, readinglistentries

redirects: Automatically resolve redirects in *query+titles*, *query+pageids*, and *query+revids*, and in pages returned by *query+generator*.

Type: boolean (details)

converttitles: Convert titles to other variants if necessary. Only works if the wiki's content language supports variant conversion. Languages that support variant conversion include ban, en, crh, gan, iu, kk, ku, shi, sr, tg, uz and zh.

Type: boolean (details)

Examples:

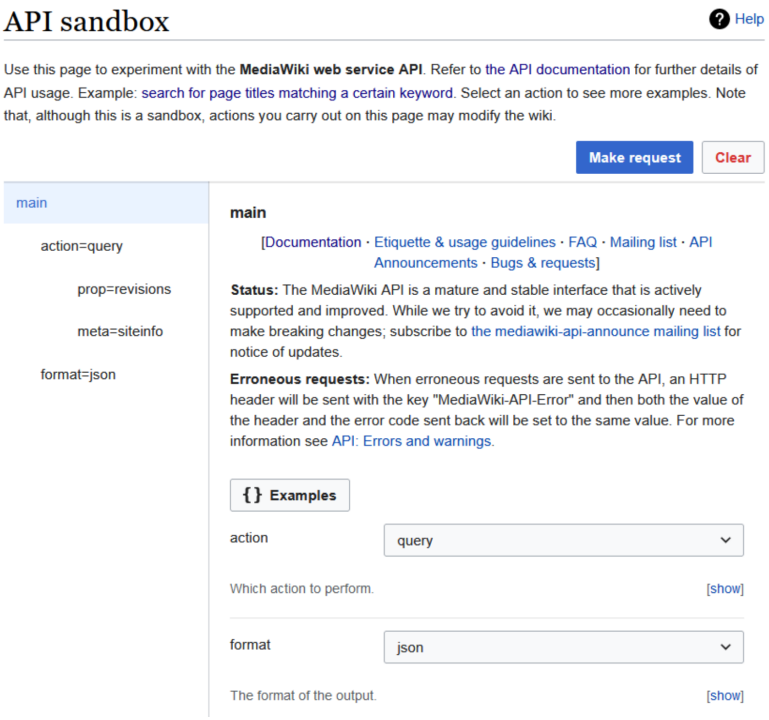
Fetch site info and revisions of `Main Page`.

api.php?action=query&prop=revisions&meta=siteinfo&titles=Main%20Page&rvprop=user|comment&continue= [open in sandbox]

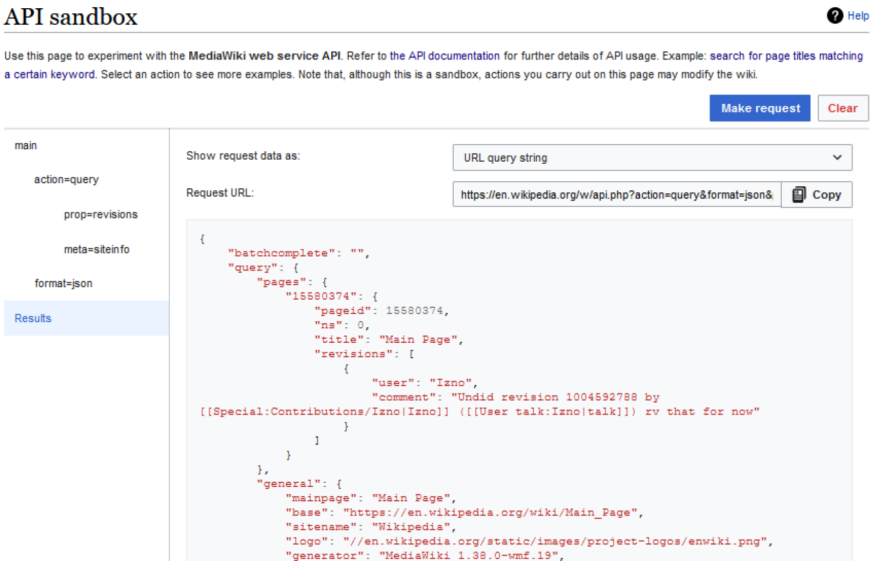
Fetch revisions of pages beginning with `API/`.

api.php?action=query&generator=allpages&gapprefix=API/&prop=revisions&continue= [open in sandbox]

Rysunek 11.3: Przykłady wykorzystania API z hiperłączami do piaskownicy



Rysunek 11.4: Piaskownica API – ustawienie parametrów



Rysunek 11.5: Piaskownica API – odczytanie odpowiedzi

Usługa *parse* punktu końcowego Wikipedii umożliwia pobranie zawartości strony i jej przetworzenie. Można również przetwarzać tekst Wiki podany przez użytkownika (wymaga to podania dodatkowo tytułu strony). Tą usługą zajmiemy się dalej jako kluczową do przetwarzania tekstu Wikipedii.

11.2.1. Parsowanie wybranych elementów strony

Domyślnie usługa *parse* podaje tytuł i identyfikator strony (przykład na rys. 11.6). Można do tego dołączyć np. właściwości *revid*, *displaytitle* i *subtitle* (przykład pokazano na rys. 11.7).


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248
    }
}
```


Rysunek 11.6: Podstawowe właściwości strony
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "revid": 1095040211,
        "displaytitle": "Esperanto",
        "subtitle": ""
    }
}
```


Rysunek 11.7: Dodatkowe właściwości strony
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto
&prop=revid%7Cdisplaytitle%7Csubtitle)

Za pomocą odpowiednich parametrów parsowania można otrzymać listy:

- *categories* – kategorii, do których należy dana strona (rys. 11.8);
- *links* – łączy wewnętrznych, tzn. prowadzących do innych stron Wikipedii (rys. 11.9);
- *externallinks* – łączy zewnętrznych, tzn. prowadzących do stron poza Wikipedią (rys. 11.10);
- *langlinks* – łączy językowych, tzn. prowadzących do innych wersji językowych Wikipedii dotyczących tego samego tematu (rys. 11.11);
- *iwlinks* – łączy interwiki, tzn. prowadzących do innych projektów Wikimedii i organizacji zaprzyjaźnionych (rys. 11.12);
- *images* – obrazów i plików dźwiękowych umieszczonych na tej stronie (rys. 11.13);
- *templates* – szablonów wykorzystywanych na tej stronie (rys. 11.14),
- *modules* – modułów ładowanych przez *ResourceLoader* (rys. 11.15). To ostatnie zapytanie wymaga żądania *jsconfigvars* lub *encodedjsconfigvars*.

Podane poniżej przykłady mogą być przydatne wówczas, gdy interesują nas wybrane metadane strony.


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "categories": [
            {
                "sortkey": "",
                "hidden": "",
                "*": "CS1_Esperanto-language_sources_(eo)"
            },
            {
                "sortkey": "",
                "hidden": "",
                "*": "Webarchive_template_wayback_links"
            },
            {
                "sortkey": "",
                "hidden": "",
                "*": "CS1_maint:_unfit_URL"
            },
```


Rysunek 11.8: Lista kategorii, do których należy dana strona
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=categories)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "links": [
            {
                "ns": 0,
                "exists": "",
                "*": "Esperanto (disambiguation)"
            },
            {
                "ns": 0,
                "exists": "",
                "*": "History of Esperanto"
            },
            {
                "ns": 0,
                "exists": "",
                "*": "Esperanto phonology"
            },
```


Rysunek 11.9: Lista łączy wewnętrznych – prowadzą do innych stron Wikipedii
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=links)

```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "externallinks": [
            "https://books.google.com/books?id=5DLpDAAAQBAJ",
            "//lccn.loc.gov/2015018907",
            "http://www.steloj.de/esperanto/paroloj/kongr3a.html",
            "https://www.gutenberg.org/files/20006/20006-h/20006-
```


Rysunek 11.10: Lista łączy zewnętrznych – prowadzą do stron poza Wikipedią
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=externallinks)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "langlinks": [
            {
                "lang": "vep",
                "url": "https://vep.wikipedia.org/wiki/Esperanto",
                "langname": "Veps",
                "autonym": "veps\u00e4n kel\u2019",
                "*": "Esperanto"
            },
            {
                "lang": "pt",
                "url": "https://pt.wikipedia.org/wiki/Esperanto",
                "langname": "Portuguese",
                "autonym": "portugu\u00eas",
                "*": "Esperanto"
            },
```


Rysunek 11.11: Lista łączy do innych wersji językowych Wikipedii
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=langlinks)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "iwlinks": [
            {
                "prefix": "eo",
                "url": "https://eo.wikipedia.org/wiki/Vikipedio:
%C4%88efpa%C4%9Do",
                "*": "eo:Vikipedio:\u0108efpa\u011do"
            },
            {
                "prefix": "eo",
                "url": "https://eo.wikipedia.org/wiki/",
                "*": "eo:"
            },
            {
                "prefix": "eo",
                "url": "https://eo.wikipedia.org/wiki/eksklavo",
                "*": "eo:eksklavo"
            },
```


Rysunek 11.12: Lista łączy interwiki – prowadzą do innych projektów Wikipedii
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=iwlinks)

```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "images": [
            "Eo-Esperanto.ogg",
            "Eo-drako-re\u011do.ogg",
            "Eo-saluton.ogg",
            "Eo-jes.ogg",
            "Eo-ne.ogg",
            "Eo-bonan_matenon.ogg",
            "Eo-bonan_vesperon.ogg",
            "Eo-bonan_nokton.ogg",
            "Eo-\u011dis_la_revido.ogg",
            "Eo-kio_estas_via_nomo.ogg",
            "Eo-mia_nomo_estas_marko.ogg",
            "Eo-kiel_vi_fartas.ogg",
            "Eo-mi_fartas_bone.ogg",
            "Cu_vi_parolas_esperanton.ogg",
```


Rysunek 11.13: Lista wykorzystywanych obrazów i plików dźwiękowych
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=images)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "templates": [
            {
                "ns": 10,
                "exists": "",
                "*": "Template:Short description"
            },
            {
                "ns": 10,
                "exists": "",
                "*": "Template:Pagetype"
            },
            {
                "ns": 10,
                "exists": "",
                "*": "Template:Main other"
            },
```


Rysunek 11.14: Lista szablonów używanych na danej stronie
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=templates)


```
{
    "warnings": {
        "parse": {
            "*": "Property \"modules\" was set but not
\"jsconfigvars\" or \"encodedjsconfigvars\". Configuration
variables are necessary for proper module usage."
        }
    },
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "modules": [
            "ext.cite.ux-enhancements",
            "ext.tmh.player",
            "ext.scribunto.logs"
        ],
        "modulescripts": [],
        "modulestyles": [
            "ext.cite.styles",
            "ext.tmh.player.styles"
        ]
    }
}
```


Rysunek 11.15: Lista modułów i zmiennych konfiguracyjnych
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=modules)

11.2.2. Parsowanie treści strony

Podstawowa funkcjonalność usługi *parse* obejmuje podawanie, parsowanie i kompilowanie treści strony Wiki. Stosując odpowiednie parametry, można uzyskać:

- *wikitext* – oryginalny tekst wiki (rys. 11.16);
- *text* – tekst skompilowany do HTML (rys. 11.17);
- *parsetree* – tekst półskompilowany do postaci drzewa (rys. 11.18).


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "wikitext": {
            "*": "{{Short description|Most widely spoken international auxiliary language}}\n{{About|the
language}}\n{{Use mdy dates|date=May 2019}}\n{{Infobox language\n| name            = Esperanto\n| altname         =
{{lang|eo|Lingvo internacia}}<ref name=\"Dua Libro\">{{cite book |last1=Zamenhof |first1=Lazaro Ludoviko |title=Dua
Libro de l' Lingvo Internacia |date=1888 |publisher=Project Gutenberg |edition=2006 |url=https://www.gutenberg.org
/files/20006/20006-h/20006-h.htm |access-date=15 March 2022 |language=eo |chapter=Aldono al la Dua Libro de l' Lingvo
Internacia |quote=Rakontinte mallonge la tutan konstruon de l' \"Lingvo internacia\" kaj \u011dian gramatikon,[...]}}
```


Rysunek 11.16: Pobrany tekst strony w formacie Wiki
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=wikitext)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "text": {
            "*": "<div class=\"mw-parser-output\"><div class=\"shortdescription nomobile noexcerpt noprint searchaux\"
spoken international auxiliary language</div>\n<style data-mw-deduplicate=\"TemplateStyles:r1033289096\">.mw-parser-out
parser-output div.hatnote{padding-left:1.6em;margin-bottom:0.5em}.mw-parser-output .hatnote i{font-style:normal}.mw-par
.hatnote+link+.hatnote{margin-top:-0.5em}</style><div role=\"note\" class=\"hatnote navigation-not-searchable\">This ar
other uses, see <a href=\"/wiki/Esperanto_(disambiguation)\" class=\"mw-disambig\" title=\"Esperanto (disambiguation)\"
</a>.</div>\n<p class=\"mw-empty-elt\">\n</p>\n<style data-mw-deduplicate=\"TemplateStyles:r1066479718\">.mw-parser-out
```


Rysunek 11.17: Tekst strony Wiki w formacie HTML
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=text)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "parsetree": {
            "*": "<root><template><title>Short description</title><part><name index=\"1\"/><value>Most widely spoken international auxiliary
language</value></part></template>\n<template lineStart=\"1\"><title>About</title><part><name index=\"1\"/><value>the language</value></part
</template>\n<template lineStart=\"1\"><title>Use mdy dates</title><part><name>date</name><equals>=</equals><value>May 2019</value></part>
</template>\n<template lineStart=\"1\"><title>Infobox language\n</title><part><name> name            </name><equals>=</equals><value>
Esperanto\n</value></part><part><name> altname         </name><equals>=</equals><value> <template><title>lang</title><part><name index=\"1
\"/><value>eo</value></part><part><name index=\"2\"/><value>Lingvo internacia</value></part></template><ext><name>ref</name><attr> name=&
quot;Dua Libro&quot;</attr><inner>{{cite book |last1=Zamenhof |first1=Lazaro Ludoviko |title=Dua Libro de l' Lingvo Internacia |date=1888
```


Rysunek 11.18: Tekst Wiki półskompilowany do postaci drzewa
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=parsetree)

Połączeniem parsowania i przeglądania strony Wiki jest możliwość pozyskania nagłówków sekcji, czyli rozdziałów i podrozdziałów strony w formacie JSON (rys. 11.19). Mając takie informacje, można zażądać parsowania tylko wybranej sekcji (rys. 11.20).

```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "sections": [
            {
                "toclevel": 1,
                "level": "2",
                "line": "History",
                "number": "1",
                "index": "1",
                "fromtitle": "Esperanto",
                "byteoffset": 8440,
                "anchor": "History"
            },
            {
                "toclevel": 2,
                "level": "3",
                "line": "Creation",
                "number": "1.1",
                "index": "2",
                "fromtitle": "Esperanto",
                "byteoffset": 8485,
                "anchor": "Creation"
            },
```


Rysunek 11.19: Wydobyte nagłówki sekcji
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto&prop=sections)


```
{
    "parse": {
        "title": "Esperanto",
        "pageid": 9248,
        "wikitext": {
            "*": "=== Creation ===\n[[File:Unua Libro ru 1st ed.pdf|thumb|upright|The first
Esperanto book by L. L. Zamenhof, published in 1887 in [[Russian language|Russian]]. The
title translates to: ''International Language: Preface and Complete Tutorial''.]]\n
\nEsperanto was created in the late 1870s and early 1880s by [[L. L. Zamenhof]], a [[History
of the Jews in Poland|Polish-Jewish]] [[Ophthalmology|ophthalmologist]] from
[[Bia\u0142ystok]], then part of the [[Russian Empire]], but now part of [[Poland]]. In the
1870s, just a few years before Zamenhof created Esperanto, [[Polish language|Polish]] was
banned in public places in Bia\u0142ystok.<ref>{{Cite web|last=Antonetti|first=Gian
```


Rysunek 11.20: Wydobyta treść jednej z sekcji
(https://en.wikipedia.org/w/api.php?action=parse&format=json&page=Esperanto
&prop=wikitext&section=2)

11.3. Parsowanie różnych formatów tekstowych

Trzy różne formaty wyników usługi *parse* mają swoje zalety i wady w aspekcie dalszego przetwarzania strony Wiki.

11.3.1. WikiText

Format WikiText zapewnia dostęp do oryginalnej, semantycznej treści strony. Zapis jest zwięzły, zapewniający niezależność informacji od sposobu jej prezentowania. Przykład artykułu pokazano na rys. 11.21. Widoczne są na nim elementy opisane poniżej.

- Znaczniki szablonów w postaci sekwencji dwóch nawiasów klamrowych otwierających {{ i zamykających }}. Po kresce pionowej występuje parametr szablonu.
  - *Short description* – zazwyczaj występuje na początku artykułu i zawiera krótki opis strony.
  - *About* – wyświetla komunikat uściślający, o czym jest ten artykuł, i przekierowanie do strony ujednoznaczniającej:

    This article is about the language. For other uses, see Esperanto (disambiguation).

  - *Use mdy dates* – określa sposób wyświetlania dat w formacie miesiąc – dzień – rok (typowy dla Stanów Zjednoczonych), parametr określa datę ostatniego sprawdzenia artykułu pod tym względem przez edytora lub bota.
  - *Infobox language* – tabela opisowa tematu artykułu, zawiera podstawowe informacje o opisywanym elemencie (tutaj: języku), umożliwia łatwe wydobycie informacji dzięki formatowaniu dwukolumnowemu właściwość – wartość.
  - *Esperanto sidebar* – wyświetla z boku strony pole nawigacyjne specyficzne dla języka esperanto (rys. 11.22).
- Treść wprowadzenia do artykułu w formacie Wiki zaczyna się od '''Esperanto''' i rozciąga się aż do spisu treści wstawionego za pomocą szablonu TOC. W treści występują:
  - znaczniki wytłuszczenia w postaci sekwencji trzech apostrofów (''');
  - szablony wprowadzane znacznikami {{ i }}, przykładowo szablon IPAc-en pokazuje wymowę w notacji fonetycznej w języku angielskim;
  - znaczniki przypominające HTML ujęte w nawiasy trójkątne ostre – w tym wypadku <ref> tworzy odsyłacz do cytowania;
  - znaczniki łączy wewnętrznych do innych artykułów Wikipedii w postaci sekwencji dwóch nawiasów kwadratowych otwierających [[i zamykających]],

- Szablon spisu treści {{TOC limit|3}}.
- Nagłówki sekcji drugiego i trzeciego poziomu otoczone z obu stron podwójnymi lub potrójnymi znakami równości (np. == History == i === Creation ===).
- Szablon {{Main|History of Esperanto}} wiodący do głównego artykułu.

Main article: History of Esperanto

- Odsyłacz do pliku z ilustracją wraz z podpisem [[File:Unua Libro ru 1st ed.pdf|...]]. Ilustracja jest wyświetlana z boku strony (rys. 11.23).

- Sekcja === See also === zawierająca odsyłacze do innych źródeł. Tutaj:
  - Szablon {{Portal|Constructed languages|Language}} powoduje wyświetlenie z boku strony pola nawigacyjnego do portali (rys. 11.24).
  - Szablon {{columns-list|colwidth=35em|...}} wprowadza formatowanie zawartej w nim listy w jednej kolumnie o ustalonej szerokości. Elementy listy rozpoczynają się od znaczników * (pierwszy poziom) i ** (drugi poziom).

- Sekcja == Notes == wraz z szablonem {{Notelist}} wprowadza listę przypisów do artykułu.
- Sekcja == References == wraz szablonem {{Reflist|...}} wprowadza listę odsyłaczy do źródeł cytowanych w artykule.
- Sekcja == Further reading == wraz z szablonem {{refbegin}} wprowadza listę odsyłaczy do innych źródeł.
- Sekcja == External links == wprowadza linki zewnętrzne przez szablony:
  - Szablon {{InterWiki|code=eo}} wyświetla link do wydania Wikipedii w języku esperanto (rys. 11.25).
  - Szablon {{Spoken Wikipedia|Esperanto spoken article.ogg|...}} wyświetla pole boczne z linkiem do artykułu dźwiękowego w języku esperanto (rys. 11.26).
  - Szablon {{Sister project links|wikt=Category:Esperanto...}} wyświetla pole boczne z linkami do siostrzanych projektów Wikipedii w kategorii esperanto (rys. 11.27)

- Szablon {{Constructed languages}} wstawia rozwijalne pole nawigacyjne prowadzące do różnych języków sztucznych (rys. 11.28).

- Szablon {{Authority control}} wprowadza rozwijalne pole pokazujące źródła kontroli autorytatywnej (rys. 11.29).

- Sekwencja linków [[Category:...] jest umieszczana w polu pokazującym kategorie, do których należy artykuł (rys. 11.30).

{{Short description|Most widely spoken international auxiliary language}}

{{About|the language}}

{{Use mdy dates|date=May 2019}}

{{Infobox language

| name = Esperanto

| altname = {{lang|eo|Lingvo internacia}}<ref name=\"Dua Libro\">{{cite book |last1=Zamenhof |first1=Lazaro Ludoviko |title=Dua Libro de l' Lingvo Internacia |date=1888 |publisher=Project Gutenberg |edition=2006 |url=https://www.gutenberg.org/files/20006/20006-h/20006-h.htm |access-date=15 March 2022 |language=eo |chapter=Aldono al la Dua Libro de l' Lingvo Internacia |quote=Rakontinte mallonge la tutan konstruon de l' \"Lingvo internacia\" kaj \u011dian gramatikon,[...]}}</ref>

...

}}

{{Esperanto sidebar}}

'''Esperanto''' ({{IPAc-en|\u02cc|\u025b|s|p|\u0259|\u02c8|r|\u0251\u02d0|n|t|o\u028a|}} or {{IPAc-en|\u02cc|\u025b|s|p|\u0259|\u02c8|r|\u00e6|n|t|o\u028a|}})<ref>{{Citation|last=Jones|first=Daniel|title=English Pronouncing Dictionary|year=2003|editor=Peter Roach|orig-year=1917|place=Cambridge|publisher=Cambridge University Press|isbn=3-12-539683-2|author-link=Daniel Jones (phonetician)|editor2=James Hartmann|editor3=Jane Setter}}</ref><ref>{{citation|last=Wells|first=John C.|title=Longman Pronunciation Dictionary|year=2008|edition=3rd|publisher=Longman|isbn=978-1-4058-8118-0}}</ref> is the world's most widely spoken [[Constructed language|constructed]] [[international auxiliary language]]. Created by [[Warsaw]]-based [[ophthalmologist]] [[L. L. Zamenhof]] in 1887, it was intended to be a universal [[second language]] for international communication, or \"the international language\" ({{Lang|eo|la lingvo internacia}}). Zamenhof first described the language in ''[[Dr. Esperanto's International Language]]'' ({{Lang-eo|Unua Libro}}), which he published under the pseudonym {{Lang|eo|[[Doctor (title)|Doktoro]] Esperanto}}. Early adopters of the language liked the name ''Esperanto'' and soon used it to describe his language. The word {{Lang|eo|esperanto}} translates into English as \"one who hopes\".<ref>{{cite web |title=Doktoro Esperanto, Ludwik Lejzer Zamenhof |url=https://global.britannica.com/biography/L-L-Zamenhof |url-status=unfit |archive-url=https://web.archive.org/web/20200529085951/https://global.britannica.com/biography/L-L-Zamenhof |archive-date=May 29, 2020 |access-date=December 3, 2016 |website=Global Britannica.com |publisher=Encyclop\u00e6dia Britannica Inc}}</ref>

...

{{TOC limit|3}}

== History ==

{{Main|History of Esperanto}}

=== Creation ===

[[File:Unua Libro ru 1st ed.pdf|thumb|upright|The first Esperanto book by L. L. Zamenhof, published in 1887 in [[Russian language|Russian]]. The title translates to: ''International Language: Preface and Complete Tutorial''.]]

...

== See also ==

{{Portal|Constructed languages|Language}}

{{columns-list|colwidth=35em|

* [[Outline of Esperanto]]

* [[Arcaicam Esperantom]]

* [[Comparison between Esperanto and Ido]]

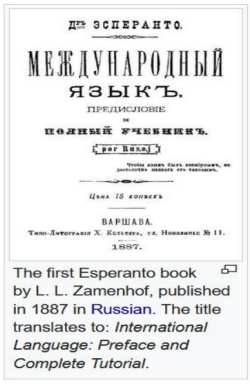
** [[Ido]]

...

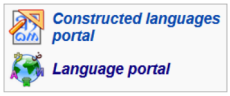
}}

Rysunek 11.21: Przykładowa treść artykułu w formacie Wiki

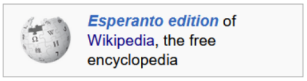
Rysunek 11.22: Pole nawigacyjne z boku strony (*sidebar*)



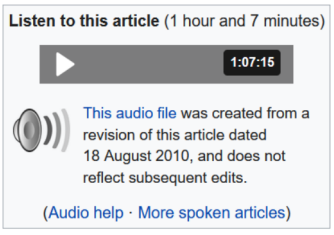
Rysunek 11.23: Ilustracja z boku strony



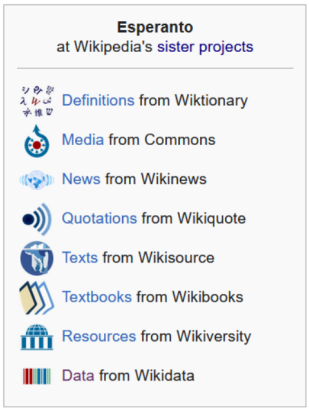
Rysunek 11.24: Pole nawigacyjne do portali



Rysunek 11.25: Link boczny do Wikipedii w języku esperanto

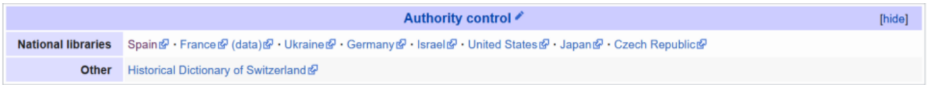


Rysunek 11.26: Pole boczne z linkiem do pliku dźwiękowego



Rysunek 11.27: Pole boczne z linkami do projektów siostrzanych

Rysunek 11.28: Rozwijalne pole nawigacyjne



Rysunek 11.29: Rozwijalne pole kontroli autorytatywnej

Rysunek 11.30: Pole kategorii

Ważnym, często występującym elementem, który ze względów redakcyjnych nie został pokazany na rys. 11.21, jest tabela. Przykładową tabelę prezentuje rys. 11.31. W formacie Wiki znacznikiem rozpoczynającym tabelę jest {|, a kończącym tabelę jest |}. Znacznikiem tytułu (*caption*) tabeli jest |+, separatora wiersza |-, wiersza nagłówkowego !, a wiersza zwykłego |. Z kolei separatorem komórek w wierszu nagłówkowym jest !!, a w wierszu zwykłym ||. Znaczniki te (z wyjątkiem !! i ||) muszą się pojawić na początku linii tekstu. Atrybut class="wikitable" oznacza standardowy styl formatowania tabeli, w którym stosowane są linie siatki, a nagłówki są wytłuszczone i na nieco ciemniejszym tle.

```
{| class=\"wikitable\" style=\"margin: 0 auto;\"
|-
!
! colspan=\"2\" | [[Labial consonant|Labial]]
! colspan=\"2\" | [[Alveolar consonant|Alveolar]]
! colspan=\"2\" | [[Palatal consonant|Palatal]]
! colspan=\"2\" | [[Velar consonant|Velar]]
! colspan=\"2\" | [[Glottal consonant|Glottal]]
|- style=\"text-align:center;\"
! style=\"text-align:left;\"| [[nasal consonant|Nasal]]
| colspan=\"2\" |{{IPA link|m}}
| colspan=\"2\" |{{IPA link|n}}
| colspan=\"2\" |  
| colspan=\"2\" |  
| colspan=\"2\" |  
|- style=\"text-align:center;\"
! style=\"text-align:left;\"| [[stop consonant|Stop]]
```

	Labial		Alveolar		Palatal		Velar		Glottal
Nasal	m		n						
Stop	p	b	t	d			k	g	
Affricate			t͡s	(d͡z)	t͡ʃ	d͡ʒ			
Fricative	f	v	s	z	ʃ	ʒ	(x)		h
Approximant			l		j		(w)		
Trill			r						

Rysunek 11.31: Tabela Wikipedii: po lewej Wikitekst, po prawej widok na stronie

Jak widać, artykuł ma złożoną strukturę, w której bardzo ważną rolę odgrywają szablony. Szablon w Wikipedii jest specjalną konstrukcją, która umożliwia wstawianie sparametryzowanej treści do artykułów. Szablony są porównywane do makr [8] i rzeczywiście wiele szablonów jest tak zaimplementowanych. Przykład takiej definicji pokazuje rys. 11.32. W definicji oprócz zwykłego tekstu (np. „References") występują znaczniki HTML (np. <h2>), funkcje wyrażeniowe (np. #if, #tag) i wywołania innych szablonów (np. {{NAMESPACE}}, {{Hatnote|...}}). Inne szablony zawierają tylko wywołanie modułu napisanego w języku Lua. Przykład podano na rys. 11.33. Implementacja szablonu w języku Lua może być bardzo złożona. Przykładowy fragment pokazano na rys. 11.34.

```
{#if:{{NAMESPACE}}|<h2>References</h2>

{{Hatnote|These references will appear in the article, but this list appears only on this page.}}

{{#tag:references|{{{refs|}}}|responsive=1}}

{{#if:{{{1|}}}|

{{Hatnote|The references of group "{{{1}}}":}}

{{#tag:references|{{{refs|}}}|group={{{1}}}|responsive=1}}}}

{{#if:{{{2|}}}|

{{Hatnote|The references of group "{{{2}}}":}}

{{#tag:references|{{{refs|}}}|group={{{2}}}|responsive=1}}}}

{{#if:{{{3|}}}|

{{Hatnote|The references of group "{{{3}}}":}}
```

Rysunek 11.32: Przykładowy szablon zdefiniowany jako makro

```
{{#invoke:Gallery|gallery}}<noinclude>{{Documentation}}</noinclude>
```

Rysunek 11.33: Przykładowy szablon zdefiniowany jako wywołanie modułu Lua

```lua
-- This module implements {{gallery}}

local p = {}

local templatestyles = 'Module:Gallery/styles.css'
local yesno = require('Module:Yesno')

local function trim(s)
    return mw.ustring.gsub(mw.ustring.gsub(s, '%s', ' '), '^%s*(.-)%s*$', '%1')
end

local tracking, preview

local function checkarg(k,v)
    if k and type(k) == 'string' then
        if k == 'align' or k == 'state' or k == 'style' or k == 'title' or
            k == 'width' or k == 'height' or k == 'lines' or k == 'whitebg' or
            k == 'mode' or k == 'footer' or k == 'perrow' or k == 'noborder' or
            k:match('^alt%d+$') or k:match('^%d+$') then
            -- valid
        elseif k == 'captionstyle' then
            if not v:match('^text%-align%s*:%s*center[;%s]*$') then
                table.insert(tracking, '[[Category:Pages using gallery with the captionstyle parameter]]')
            end
        else
            -- invalid
            local vlen = mw.ustring.len(k)
            k = mw.ustring.sub(k, 1, (vlen < 25) and vlen or 25)
            k = mw.ustring.gsub(k, '[^%w\-_ ]', '?')
            table.insert(tracking, '[[Category:Pages using gallery with unknown parameters|' .. k .. ']]')
            table.insert(preview, '"' .. k .. '"')
        end
    end
end
```

Rysunek 11.34: Implementacja szablonu w module Lua (fragment)

Po zagłębieniu się w treść artykułu, po usunięciu szablonów przypisów i znaczników referencji otrzymujemy tekst dość dobrze czytelny dla człowieka i łatwy do parsowania (rys. 11.35).

```
'''Esperanto''' ({{IPAc-en|...}} or {{IPAc-en|...}}) is the world's most widely spoken [[Constructed language|constructed]]
[[international auxiliary language]]. Created by [[Warsaw]]-based [[ophthalmologist]] [[L. L. Zamenhof]] in 1887, it was
intended to be a universal [[second language]] for international communication, or \"the international language\"
({{Lang|eo|la lingvo internacia}}). Zamenhof first described the language in ''[[Dr. Esperanto's International Language]]''
({{Lang-eo|Unua Libro}}), which he published under the pseudonym {{Lang|eo|[[Doctor (title)|Doktoro]] Esperanto}}.
Early adopters of the language liked the name ''Esperanto'' and soon used it to describe his language. The word
{{Lang|eo|esperanto}} translates into English as \"one who hopes\".
```

Rysunek 11.35: Tekst Wiki po usunięciu szablonów przypisów i znaczników referencji

Znaczniki łączy do innych artykułów wyodrębniają pojęcia, do których odwołuje się autor artykułu, a jednocześnie zapewniają możliwość szybkiego sprawdzenia tych pojęć przez hiperłącza. Tekst prezentuje się jak na rys. 11.36.

Esperanto (/ˌɛspəˈrɑːntoʊ/ or /ˌɛspəˈræntoʊ/) is the world's most widely spoken constructed international auxiliary language. Created by Warsaw-based ophthalmologist L. L. Zamenhof in 1887, it was intended to be a universal second language for international communication, or "the international language" (*la lingvo internacia*). Zamenhof first described the language in *Dr. Esperanto's International Language* (Esperanto: *Unua Libro*), which he published under the pseudonym *Doktoro Esperanto*. Early adopters of the language liked the name *Esperanto* and soon used it to describe his language. The word *esperanto* translates into English as "one who hopes".

Rysunek 11.36: Prezentacja tekstu Wiki

11.3.2. Problemy parsowania tekstu Wiki

Format Wiki ma jednak pewne wady utrudniające parsowanie. Najważniejsze scharakteryzowano poniżej.

Niejednoznaczność znaczników kursywy i wytłuszczenia. W obu przypadkach stosowane są apostrofy – znaczniki kursywy to dwa apostrofy (''*italics*''), a znaczniki wytłuszczenia to trzy apostrofy ('''**bold**'''). Jednak przy łączeniu wytłuszczenia i pochylenia pojawia się tekst, w którym występują sekwencje pięciu (!) apostrofów przeplatanych z sekwencjami dwóch i trzech. Dla przykładu tekst:

> The *general rule* is that stress falls on the vowel before the last consonant (e.g., *li**n**gua*, 'language', *ess**e**r*, 'to be', *requirim**e**nto*, 'requirement') ignoring the final plural *-(e)s* (e.g. *li**n**guas*, the plural of *lingua*, still has the same stress as the singular), and where that is not possible, on the first vowel (*v**i**a*, 'way', ***i**o* cr**e**a, 'I create').

jest zapisywany jako:

> The ''general rule'' is that stress falls on the vowel before the last consonant (e.g., ''l'''i'''ngua'', 'language', ''ess'''e'''r'', 'to be', ''requirim'''e'''nto'', 'requirement') ignoring the final plural ''-(e)s'' (e.g.

> ''l'''i'''nguas'', the plural of ''lingua'', still has the same stress as the singular), and where that is not possible, on the first vowel (''v'''i'''a'', 'way', ''''i'''o cr'''e'''a'', 'I create').

Oczywiście pojedyncze apostrofy oznaczają właśnie znaki apostrofu, ale znaki podwójnego apostrofu muszą już być wyrażane jako encje HTML ("), np.:

> title=Tunisian Arabic Corpus: Creating a written corpus of an " unwritten " language

Sekwencje czterech apostrofów też występują i wówczas oznaczają pojedynczy apostrof i znacznik wytłuszczenia. Reguły składni Wiki nie określają wówczas, czy apostrof leży wewnątrz wytłuszczenia, czy na zewnątrz, np. czy tekst '''mat'''' ma być parsowany do HTML jako: mat' czy też jako: mat'.

Niejednoznaczność znaczników w tabelach. Znaczniki tabel Wiki nie są tak jednoznaczne i łatwe w parsowaniu jak znaczniki HTML (<table>, <tr>, <td>). Problemy pojawiają się np., gdy w komórkach tabeli mają być pokazane znaki kreski pionowej. Trzeba wówczas użyć znacznika <nowiki>, który wyłącza interpretację zawartości zgodnie z regułami wiki. Można też użyć encji HTML | lub &x7C;. Przykład pokazano na rys. 11.37.

```
{| class=wikitable
|-
|<nowiki>|</nowiki>Pipes34:<nowiki>||</nowiki>
|-
|&#124;Pipes34:&#124;&#x7C;
|}
```

Pipe3:|
Pipe3:|

Rysunek 11.37: Kreska pionowa w tabeli

Inne mechanizmy zastępcze dla kreski pionowej, jak pusty znacznik <nowiki/> i szablon zastępczy {{!}} (rys. 11.38), nie zawsze działają zgodnie z oczekiwaniem.

```
{| class=wikitable
|-
|<nowiki />|Pipe3:|
|-
|Pipe2:{{!}}Pipe3:{{!}}
|}
```

Pipe3:|
Pipe3:|

Rysunek 11.38: Mechanizmy zastępcze dla kreski pionowej

Tabele zagnieżdżone wymagają specjalnych znaczników zastępczych: {{{!}} dla rozpoczęcia tabeli, {{!}}} dla zakończenia tabeli, {{!}}- dla separatora wierszy, {{!}} dla rozpoczęcia wiersza zwykłego oraz {{!}}{{!}} dla separatorów komórek zwykłych. (Wiersze nagłówkowe i komórki nagłówkowe nie mają znaczników zastępczych). Trzeba zauważyć, że znaczniki zastępcze {{{!}}, {{!}}} naruszają ogólną zasadę parowania nawiasów klamrowych Wiki (rys. 11.39).

```
{{{!}}class="wikitable" style="background-color:#f8f4ea"
!A!!B
{{!}}-
{{!}}
{{{!}}class="wikitable"
!a0!!b0
{{!}}-
{{!}}A{{!}}{{!}}A
{{!}}}xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
{{!}}
{{{!}}class="wikitable"
!a1!!b1
{{!}}-
{{!}}B{{!}}{{!}}B
{{!}}}
{{!}}}
```

A		B	
a0	b0	a1	b1
A	A	B	B
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx			

Rysunek 11.39: Tabele zagnieżdżone

Z powodu tych i innych problemów występujących w praktyce z formatowaniem tabel Wiki zaleca się obecnie stosowanie znaczników HTML: <table>, <tr>, <th> i <td>, jednak wielu autorów Wikipedii stosuje opisane powyżej formatowanie standardowe dla Wiki.

Ogromna liczba szablonów. Istnieje kilkaset (!) szablonów często używanych przez wielu autorów. Wiele z tych szablonów ma swoje nazwy zastępcze (aliasy), co potęguje zamieszanie.

Część szablonów po prostu określa sposób formatowania argumentów, ale część dodaje do treści swoje elementy, przez co następuje częściowe ukrycie semantyki. Dla przykładu szablon {{ELL2}} reprezentuje cytowanie książki Keitha Browna pt. *Encyclopedia of Language and Linguistics*, wyd. 2 z 2005 r. Inny przykład to szablon {{Life in Egypt}}, który zawiera kilkadziesiąt (!) odsyłaczy do innych artykułów Wikipedii powiązanych tematycznie z życiem codziennym w Egipcie.

Wielu autorów chętnie tworzy własne szablony i wykorzystuje je na niewielkiej liczbie stron. W ten sposób powstało kilkaset tysięcy (!) szablonów[17] i przy parsowaniu tekstu Wikipedii trzeba zawsze uwzględnić sytuację, gdy na stronie pojawi się odwołanie do nieznanego szablonu. Więcej informacji o szablonach Wikipedii podano w sekcji 11.4 tego rozdziału.

Złożona składnia szablonów. Treść szablonu można pobrać w usłudze *parse*, tak samo jak treść strony Wikipedii. Różnica polega na poprzedzeniu nazwy szablonu przedrostkiem "Template:". Dla przykładu, aby pobrać treść szablonu {{Life in Egypt}}, trzeba podać żądanie:

> https://en.wikipedia.org/w/api.php?action=parse&format=json
> &page=Template%3ALife%20in%20Egypt&prop=wikitext

Jednak interpretacja treści szablonu jest bardziej skomplikowana niż treści zwykłej strony Wikipedii. Po pierwsze w szablonie stosowane są znaczniki <includeonly> i <noinclude>, które wyróżniają tekst odpowiednio interpretowany tylko w czasie wstawiania treści szablonu na stronę oraz niewstawiany na

[17] 20 czerwca 2022 kwerenda zwróciła 923253 szablony.

stronę. Znaczniki te są rozpoznawane przez Mediawiki w czasie preprocessingu, czyli przed właściwą interpretacją tekstu Wiki. W konsekwencji mogą obejmować fragment składni, np. tylko część nagłówkową tabeli, co zaburza budowanie struktury elementów szablonu.

Po drugie w szablonach stosuje się funkcje wyrażeniowe oznaczone znakiem #. Te wyrażenia, to np. #if, #ifeq, #switch. Są one obliczane w czasie wstawiania treści szablonu na stronę i również mogą zaburzać tworzenie struktury elementów szablonu.

Przykładem dla wymienionych dwóch problemów jest definicja szablonu {{Bar percent}} pokazana na rys. 11.40.

```
<noinclude><table>
</noinclude><tr>
<td colspan="2" style="padding-left: 0.4em; padding-right: 0.4em; min-width: 8em;
{{#if:{{{bg|}}}|background:{{{bg}}};}}">{{{1}}}</td>
<td style="width: {{{barwidth|100px}}}; border-left: solid 1px silver; border-right:
solid 1px silver;{{#if:{{{bg|}}}|background:{{{bg}}};}}"><div style="background:
{{{2|gray}}}; width:{{{3|0}}}%; overflow: hidden;"> </div></td>
<td colspan="{{#if:{{{note|}}}|1|2}}" style="padding-left: 1.2em; padding-right:
0.4em; text-align: right;{{#if:{{{bg|}}}|background:{{{bg}}};}}">{{{4|{{{3|0}}}%}}}
</td>
{{#if:{{{note|}}}|<td style="padding-left: 0.4em; padding-right: 0.4em; text-
align:right;{{#if:{{{bg|}}}|background:{{{bg}}};}}">{{{note}}}</td>}}
</tr><noinclude>
</table>
{{Documentation}}
</noinclude>
```

Rysunek 11.40: Przykładowa treść szablonu

Znacznik <noinclude> na początku treści szablonu zawiera tylko znacznik <table> rozpoczynający tabelę. Z kolei <noinclude> na końcu treści szablonu zawiera zarówno znacznik HTML </table> kończący tabelę, jak i szablon {{Documentation}} tworzący odsyłacz do strony dokumentacji szablonu.

Największym problemem przy rozpoznawaniu treści szablonu jest funkcja parsera #invoke. Powoduje ona wywołanie modułu rozszerzającego treść Wiki napisanego w języku Lua. Jest to lekki język skryptowy służący do rozszerzania aplikacji, a w świecie Wikimedii do rozszerzania tekstu Wiki o takie treści, których wyrażenie w postaci szablonów byłoby trudne.

Pobranie treści modułu jest również możliwe przez usługę *parse* – nazwę modułu trzeba poprzedzić przedrostkiem "Module:" (patrz rys. 11.34), ale interpretacja tekstu modułu wymagałaby użycia wyspecjalizowanego kompilatora języka Lua.

Błędy formatowania popełniane przez autorów. Niejednoznaczna składnia Wiki (przynajmniej w zakresie kursywy i wytłuszczenia oraz tabel) powoduje, że autorzy treści mogą łatwo popełnić błąd. Najczęściej spotykane błędy to brak lub nadmiar apostrofów na końcu linii tekstu, błędy parowania nawiasów klamrowych (oznaczających szablony) i kwadratowych (oznaczających łącza do innych stron Wikipedii), a także umieszczanie średnika kończącego atrybut "style" tabel i fragmentów tekstu poza cudzysłowem zamykającym, np.

zamiast: style="text-align:center;" może wystąpić: style="text-align:center";. Błędy te, tolerowane przez człowieka w tekście, zaburzają pracę parsera i powodują konieczność stosowania opartej na heurystyce korekty tekstu przed jego właściwym przetworzeniem.

Wymienione powyżej problemy sprawiają, że rozpoznawanie treści artykułów Wikipedii przez parsowanie oryginalnego tekstu Wiki jest trudne do automatyzacji i obarczone dużym ryzykiem niepowodzenia.

11.3.3. ParseTree

Usługa *parse* może podawać tekst półskompilowany do postaci drzewa DOM. Wbrew oczekiwaniom wynikającym z nazwy tej usługi tekst wiki nie jest w pełni sparsowany, zamiast tego elementy DOM reprezentują głównie szablony i sporo nierozpoznanego tekstu w formacie WikiText. Głównym elementem drzewa DOM jest <root>. Element ten zawiera wszystkie inne elementy, co sprawia, że drzewo DOM może być rozpoznawane jako XML (rys. 11.41). Tekst Wiki jest przeplatany elementami XML (rys. 11.42), co powoduje, że do jego odczytania nie wystarczy standardowy deserializator XML, chociaż można go czytać przez dokument XDocument (z biblioteki System.Xml.Linq).

```
<root><template><title>short description</title><part><name index="1"/>
<template lineStart="1"><title>Other uses</title><part><name index="1",
<template lineStart="1"><title>pp-vandalism</title><part><name>small</n
<template lineStart="1"><title>pp-move-indef</title></template>
<template lineStart="1"><title>Use dmy dates</title><part><name>date</n
<template lineStart="1"><title>Use British English</title><part><name>c
```

Rysunek 11.41: Początek drzewa DOM z szablonami

```
Modern English relies more on [[auxiliary verb]]s and [[word order]] for th
[[grammatical mood|mood]], as well as [[passive voice|passive constructions
languages by total number of speakers|most spoken language]] in the world (
<ext>
  <name>ref</name>
  <attr/>
  <inner>{{e22|eng|English}}</inner>
  <close>&lt;/ref&gt;</close>
</ext>
and the [[List of languages by number of native speakers|third-most spoken
<template>
  <title>sfn</title>
  <part>
    <name index="1"/>
    <value>Ethnologue</value>
  </part>
  <part>
    <name index="2"/>
    <value>2010</value>
  </part>
</template>
```

Rysunek 11.42: Tekst Wiki przeplatany elementami XML

Elementy <template> mają sparsowane poszczególne części (rys. 11.43), co zdecydowanie upraszcza ich rozpoznawanie i zmniejsza podatność parsera na błędy formatowania Wiki. Dodatkowo niektóre elementy DOM zawierają takie atrybuty, jak "lineStart" i "index", które już częściowo rozpoznają, czy szablon rozpoczyna linię i czy parametr szablonu jest numerowany, czy nazwany. Inne elementy tekstu Wiki, takie jak tabele, referencje do stron Wikipedii czy kursywa i wytłuszczenie, muszą jednak być rozpoznawane samodzielnie.

```
<template>
  <title>sfn</title>
  <part>
    <name index="1"/>
    <value>Crystal</value>
  </part>
  <part>
    <name index="2"/>
    <value>2003b</value>
  </part>
  <part>
    <name>pp</name>
    <equals>=</equals>
    <value>108–109</value>
  </part>
</template>
```

Rysunek 11.43: Sparsowane części szablonu

Ten model parsowania częściowo rozwiązuje problem ogromnej liczby szablonów. Szablony w modelu DOM nie są rozwinięte, lecz występują w postaci nazwy (element <title>) i listy parametrów (elementy <part>). Częściowe rozwiązanie polega na tym, że parser nie musi znać składni konkretnego szablonu. Jeśli nie zna, to może po prostu pominąć cały element <template>. Nie jest to jednak kompletne rozwiązanie, bo brak rozwinięcia szablonu powoduje, że informacje zawarte w jego definicji pozostają ukryte. Dla przykładu wiele artykułów zawiera na końcu ustrukturalizowaną listę odsyłaczy do innych artykułów podawaną w postaci wywołania jakiegoś szablonu i te odwołania pozostaną nieodkryte.

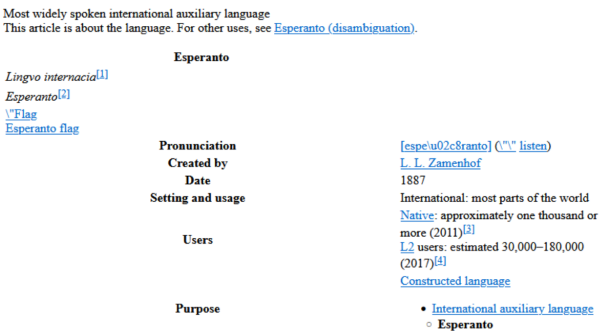
Przeprowadzony eksperyment wykazał, że zastosowanie drzewa DOM do dalszego parsowania tekstu Wikipedii nie jest praktyczne. Zbyt dużo elementów musi być samodzielnie rozpoznawanych w tekście, aby można było się oprzeć wyłącznie na strukturze XML. Parser drzewa DOM musi być rozbudową parsera oryginalnego tekstu Wiki. W dodatku przemieszanie elementów tekstowych i elementów XML powoduje konieczność zbudowania dodatkowego mechanizmu rozpoznawania hybrydowego, co znacznie komplikuje implementację.

11.3.4. HtmlText

Całkowite rozwinięcie szablonów następuje wówczas, gdy treść strony jest parsowana do tekstu HTML. Nazwanie tej usługi "parsowaniem" stanowi pewne nieporozumienie, gdyż nie jest rozpoznawana gramatyka, a w wyniku nie jest

podawane drzewo elementów, lecz wygenerowany z tego drzewa tekst gotowy do wyświetlenia w przeglądarce. Przyznają to nawet autorzy tego rozwiązania[18].

Usługa *parse* podaje osobno nagłówek HTML (opcja *headhtml*), a osobno ciało dokumentu (opcja *text*). To umożliwia zignorowanie nagłówka i przejście od razu do analizy treści artykułu. Z drugiej jednak strony w treści HTML zawarte są odwołania do styli (zdefiniowanych w nagłówku), więc tekst HTML pozyskany przez usługę *parse* i prezentowany w przeglądarce (rys. 11.44) wygląda inaczej niż strona Wikipedii przeglądana bezpośrednio (rys. 11.45).



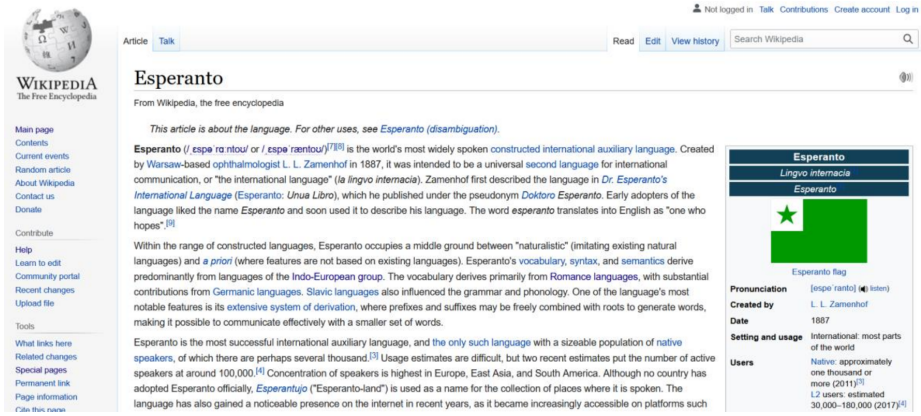
Most widely spoken international auxiliary language
This article is about the language. For other uses, see Esperanto (disambiguation).

Esperanto

Lingvo internacia[1]
Esperanto[2]
\"Flag
Esperanto flag

Pronunciation	[espe\u02c8ranto] (\"\" listen)
Created by	L. L. Zamenhof
Date	1887
Setting and usage	International: most parts of the world
Users	Native: approximately one thousand or more (2011)[3] L2 users: estimated 30,000–180,000 (2017)[4]
	Constructed language
Purpose	• International auxiliary language ◦ **Esperanto**

Rysunek 11.44: Pobrany tekst strony w HTML renderowany w przeglądarce



Not logged in Talk Contributions Create account Log in
Article Talk Read Edit View history Search Wikipedia
WIKIPEDIA The Free Encyclopedia
Esperanto
From Wikipedia, the free encyclopedia
This article is about the language. For other uses, see Esperanto (disambiguation).
Esperanto (/ˌɛspəˈrɑːntoʊ/ or /ˌɛspəˈræntoʊ/)[7][8] is the world's most widely spoken constructed international auxiliary language. Created by Warsaw-based ophthalmologist L. L. Zamenhof in 1887, it was intended to be a universal second language for international communication, or "the international language" (*la lingvo internacia*). Zamenhof first described the language in *Dr. Esperanto's International Language* (Esperanto: *Unua Libro*), which he published under the pseudonym *Doktoro Esperanto*. Early adopters of the language liked the name *Esperanto* and soon used it to describe his language. The word *esperanto* translates into English as "one who hopes".[9]
Within the range of constructed languages, Esperanto occupies a middle ground between "naturalistic" (imitating existing natural languages) and *a priori* (where features are not based on existing languages). Esperanto's vocabulary, syntax, and semantics derive predominantly from languages of the Indo-European group. The vocabulary derives primarily from Romance languages, with substantial contributions from Germanic languages. Slavic languages also influenced the grammar and phonology. One of the language's most notable features is its extensive system of derivation, where prefixes and suffixes may be freely combined with roots to generate words, making it possible to communicate effectively with a smaller set of words.
Esperanto is the most successful international auxiliary language, and the only such language with a sizeable population of native speakers, of which there are perhaps several thousand.[3] Usage estimates are difficult, but two recent estimates put the number of active speakers at around 100,000.[4] Concentration of speakers is highest in Europe, East Asia, and South America. Although no country has adopted Esperanto officially, *Esperantujo* ("Esperanto-land") is used as a name for the collection of places where it is spoken. The language has also gained a noticeable presence on the internet in recent years, as it became increasingly accessible on platforms such
Esperanto
Lingvo internacia
Esperanto
Esperanto flag
Pronunciation [espeˈranto] listen
Created by L. L. Zamenhof
Date 1887
Setting and usage International: most parts of the world
Users Native: approximately one thousand or more (2011)[3] L2 users: estimated 30,000–180,000 (2017)[4]

Rysunek 11.45: Strona Wikipedii w wersji widzianej przez użytkownika

Przy rozpoznawaniu treści strony niektóre znaczniki HTML mają znaczenie, a inne mogą być pomijane. Całość treści jest zawarta w znaczniku z atrybutem class="mw-parser-output" (rys. 11.46). Inne elementy mogą zawierać wieloczłonowe atrybuty *class*, które określają semantykę zawartych w nich treści. Dla przykładu element HTML:

[18] https://www.mediawiki.org/wiki/Manual:Parser

West Germanic language

odpowiada elementowi Wiki:

{{short description|West Germanic language}}

i oznacza krótki opis strony.

```
<div class="mw-parser-output">
  <div class="shortdescription nomobile noexcept noprint searchaux"
  style="display:none">West Germanic language</div>
  <style data-mw-deduplicate="TemplateStyles:r1033289096">.mw-parser-output .hatnote{font-
  i{font-style:normal}.mw-parser-output .hatnote+link+.hatnote{margin-top:-0.5em}</style>
  <div role="note" class="hatnote navigation-not-searchable">For other uses, see
   <a href="/wiki/English_(disambiguation)"
    class="mw-redirect mw-disambig"
    title="English (disambiguation)">English (disambiguation)</a>
    .
  </div>
  <p class="mw-empty-elt"/>
  <style data-mw-deduplicate="TemplateStyles:r1066479718">.mw-parser-output
  .infobox-subbox{padding:0;border:none;margin:-3px;width:auto;min-width:100%;font-size:10
  .navbar{font-size:100%}body.skin-minerva .mw-parser-output .infobox-header,body.skin-min
  .infobox-title,body.skin-minerva .mw-parser-output .infobox-image,body.skin-minerva .mw-

</div>
```

Rysunek 11.46: Znaczniki w tekście HTML

Znaczniki <style> oraz atrybuty "style" reprezentujące style prezentacji tekstu mogą być pomijane (rys. 11.47).

```
<div class="mw-parser-output">
  <div class="shortdescription nomobile noexcept noprint searchaux">West Germanic language</div>
    <div role="note" class="hatnote navigation-not-searchable">For other uses, see
    <a href="/wiki/English_(disambiguation)" class="mw-redirect mw-disambig"
       title="English (disambiguation)">English (disambiguation)</a>
    .
  </div>
```

Rysunek 11.47: Tekst HTML z usuniętymi nadmiarowymi znacznikami

Wadę parsowania tekstu HTML pobieranego z Wikipedii stanowi niepełna informacja semantyczna. Nie do końca wiadomo, jakie jest znaczenie poszczególnych elementów HTML. Jeśli taka informacja jest istotna, to trzeba stosować inżynierię wsteczną. Dla przykładu, aby odróżnić referencje wewnętrzne (do innych stron Wikipedii) od referencji do wymowy (do strony IPA), trzeba analizować URL tych referencji (rys. 11.48).

Odzyskanie oryginalnej informacji drogą inżynierii wstecznej jest często niemożliwe. Przykład stanowi element infoboks, który w tekście Wiki artykułu zawiera zestawienie podstawowych informacji dotyczących przedmiotu artykułu. W HTML ten element jest przekształcony w tabelę, która niekoniecznie zawiera wszystkie informacje oryginalnego elementu (rys. 11.49).

```
<span class="IPA nopopups noexcerpt">
  <a href="/wiki/Help:IPA/English" title="Help:IPA/English">/
    <span style="border-bottom:1px dotted">
      <span title="/ˈ/: primary stress follows">ˈ</span>
      <span title="/ɪ/: &#39;i&#39; in &#39;kit&#39;">ɪ</span>
      <span title="/ŋ/: &#39;ng&#39; in &#39;sing&#39;">ŋ</span>
      <span title="/g/: &#39;g&#39; in &#39;guy&#39;">g</span>
      <span title="&#39;l&#39; in &#39;lie&#39;">l</span>
      <span title="/ɪ/: &#39;i&#39; in &#39;kit&#39;">ɪ</span>
      <span title="/ʃ/: &#39;sh&#39; in &#39;shy&#39;">ʃ</span>
    </span>
    /
  </a>
</span>
```

Rysunek 11.48: Przykład referencji do strony IPA

(a)

```
{{Infobox language
| name            = English
| pronunciation   = {{IPAc-en|ˈ|ɪ|ŋ|g|l|ɪ|ʃ}}{{sfn|Oxford Learner's Dictionary|20
| region          =
| speakers        = 360{{ndash}}400{{nbsp}}million
| ethnicity       = [[English people]] <br/> [[Anglo-Saxons]] (historically)
| date            = 2006
| ref             = {{sfn|Crystal|2006|pp=424–426}}
| speakers2       = [[Second language|L2 speakers]]: 750{{nbsp}}million;<br/>as a
| familycolor     = Indo-European
| fam2            = [[Germanic languages|Germanic]]
| fam3            = [[West Germanic languages|West Germanic]]
| fam4            = [[Ingvaeonic languages|Ingvaeonic]]
| fam5            = [[Anglo-Frisian languages|Anglo-Frisian]]
| fam6            = [[Anglic languages|Anglic]]
| ancestor        = [[Old English]]
| ancestor2       = [[Middle English]]
| ancestor3       = [[Early Modern English]]
| script          = {{plainlist|
* [[Latin script|Latin]] ([[English alphabet]])
* [[Anglo Saxon runes]] (historically)
* [[English Braille]], [[Unified English Braille]]
}}
```

(b)

```
<tr>
  <th scope="row" class="infobox-label" style="white-space:nowrap;padding-right:0.65em;">
    <div style="display:inline-block; padding:0.1em 0;line-height:1.2em;">Native speakers</div>
  </th>
  <td class="infobox-data" style="line-height:1.3em;">360–400
    <span class="nowrap"> </span>
    million (2006)
    <sup id="cite_ref-FOOTNOTECrystal2006424–426_2-0" class="reference">
      <a href="#cite_note-FOOTNOTECrystal2006424–426-2">&#91;2&#93;</a>
    </sup>
    <br/>
    <a href="/wiki/Second_language" title="Second language">L2 speakers</a>
    : 750
    <span class="nowrap"> </span>
    million;
    <br/>
    as a
    <a href="/wiki/Foreign_language" title="Foreign language">foreign language</a>
    : 600–700 million
    <sup id="cite_ref-FOOTNOTECrystal2006424–426_2-1" class="reference">
      <a href="#cite_note-FOOTNOTECrystal2006424–426-2">&#91;2&#93;</a>
    </sup>
  </td>
</tr>
```

Rysunek 11.49: Element infoboks w tekście Wiki (a) oraz fragment tego elementu w HTML (b)

11.4. Szablony Wikipedii

Jak już wspomniano w sekcji 11.3.2, jednym z problemów parsowania jest ogromna liczba szablonów Wikipedii. Każda wersja językowa może mieć swoje szablony. Nawet w ramach wersji anglojęzycznej wiele szablonów ma po kilka nazw zastępczych (aliasów). Większość szablonów należy do przestrzeni nazw Template, ale część do innych przestrzeni nazw. Zestawienie przedstawiono w tablicy 11.4.

Tablica 11.4: Zestawienie szablonów według przestrzeni nazw (C – liczba kategorii, P – liczba stron)

Przestrzeń nazw	**Tematyka**
Article (C=11)	Szablony udostępniania treści w artykule, diagramy i wykresy, linki wewnętrzne i zewnętrzne, formatowanie grafiki i funkcji, wiadomości dotyczące artykułów
Book (P=2)	Szablony książek, testowanie PDF
Category (C=9, P=151)	Nagłówki kategorii, kategorie chronologii, klasyfikacja Wikipedii
Draft (C=2, P=20)	Szablony wersji wstępnych
File (C=5, P=36)	Włączanie plików audio i innych plików, szablony praw autorskich, symbole
Help (C=2, P=11)	Szablony pomocy
MediaWiki (P=11)	Sprawdzenie uprawnień do usunięcia pliku/strony, skórki, widoczności
Module (C=1, P=4)	Szablony wewnętrznych łączy modułu, szablony piaskownicy i testów
Multiple namespace (C=1)	Szablony chronologii wielu przestrzeni nazw
Portal (C=2, P=25)	Szablony portali bieżących wydarzeń, portalu Wikipedii i innych portali
Talk (C=12, P=82)	Szablony przestrzeni nazw dyskusji o artykułach
Template (C=9, P=124)	Większość szablonów
TemplateTalk (C=1)	Szablony nagłówków dyskusji
User (C=14, P=232)	Szablony nagród Wikipedii, szablony informacyjne użytkownika
UserTalk (C=2)	Szablony dyskusji użytkowników
Wikipedia (C=6, P=34)	Szablony nagłówkowe Wikipedii, szablony związane z akwizycją Wikipedii

Szablony mogą też być uporządkowane według kategorii, do których należą, co przedstawiono w tablicy 11.5.

Tablica 11.5: Zestawienie szablonów według kategorii
(C – liczba kategorii, P – liczba stron)

Kategoria	**Tematyka**
Szablony nagród (C=15, P=12)	Szablony nawigacyjne nagród, szablony infoboksów nagród
Szablony cytowań (C=22, P=22)	Szablony cytatów w stylu harwardzkim, w stylu *bluebook*, szablony cytujące książki, szablony dotyczące przeglądania cytowań i ich weryfikowalności, szablony przypisów itp.
Szablony wielokolumnowe (P=24)	Szablony formatujące tekst na wiele kolumn
Szablony praw autorskich (C=5, P=1)	Szablony dotyczące ochrony i stosowania praw autorskich
Szablony flag (C=7, P=4)	Szablony flag krajów, regionów, miast
Szablony usuwania (C=16, P=16)	Szablony ostrzeżeń, znaczniki usuwania, powiadomienia o usunięciu
Szablony komunikatów (P=36)	Szablony wyświetlające komunikaty o błędach na stronie
Szablony importowania infoboksów (P=18)	Szablony przeznaczone do importowania infoboksów z innych wydań językowych Wikipedii
Szablony infoboksów(C=19, P=200)	Szablony infoboksów wg kontynentów, kraju, szablony infoboksów dotyczące sztuki i kultury; zdrowia i kondycji; historii i wydarzeń; ludzi i osób; miejsc; religii i przekonań; nauki i przyrody; społeczeństwa i nauk społecznych; technologii i nauk stosowanych
Szablony współrzędnych (C=2, P=43)	Szablony konwersji współrzędnych, szablony z polami współrzędnych
Szablony map (C=2, P=17)	Mapy elektrowni jądrowych, mapy drużyn National Football League, inne szablony map
Szablony medyczne (C=26, P=9)	Szablony leków według systemów, szablony paska bocznego medycyny, szablony anatomiczne, szablony procedur medycznych etc.
Szablony nawigacyjne (C=3)	Pola nawigacyjne, szablony paska bocznego
Szablony przekierowań (C=7, P=300)	Szablony przekierowań do określonych typów artykułów
Szablony tras (C=1, P=74)	Szablony katalogu diagramów tras
Szablony szachownicy (P=21)	Diagramy szachownicy w różnych rozmiarach i typach
Szablony userboksów (C=27)	Szablony rozmieszczenia userboksów, pola nawigacyjne userboksów

Liczby przedstawione w tablicach 11.4 i 11.5 stanowią jedynie „wierzchołek góry lodowej". Nie odzwierciedlają bieżącego stanu zasobów, lecz stan uchwycony w pewnym momencie przez edytorów Wikipedii. Badanie własne autora niniejszego rozdziału przeprowadzone w 2022 r. ujawniło np. 147 507 szablonów pól nawigacyjnych i 152 055 przekierowań. Stan ten nieustannie się zmienia.

11.5. Wnioski

Oparcie parsowania na oryginalnym tekście w formacie Wiki daje dostęp do podstawowych informacji semantycznych, które w formacie HTML mogą być utracone. Ta informacja jest zakodowana w nazwach szablonów, jednak szablony są przeznaczone do ułatwienia pracy autorów Wikipedii, a nie do analizy semantycznej. Do takich szablonów należą {2x}, {3x}...{5x}, które skracają zapisy. Część szablonów, takich jak {align}, {center}, służy do formatowania tekstu. Inne, jak {efn}, {refn}, służą do wstawiania przypisów na końcu strony. Wiele szablonów, takich jak {cite}, {harv}, służy do wstawiania przypisów bibliograficznych. Jednak ogromna liczba szablonów zdecydowanie utrudnia takie parsowanie. Szablony są rozwijane przy translacji tekstu Wiki na HTML, co w rezultacie zmniejsza liczbę różnych elementów do rozpoznania. Częściowo semantyka jest wyrażana w formacie HTML poprzez atrybut *class* takich elementów jak i <table>.

Inne problemy formatu Wiki, takie jak podatność na błędy składniowe popełniane przez autorów czy niejednoznaczność formatowania, są częściowo (ale tylko częściowo!) rozwiązane w modelu DOM, czyli w formacie ParseTree. Jednak problem ogromnej liczby szablonów pozostaje nierozwiązany.

Powyższe problemy są rozwiązywane w formacie HTML, ale pojawia się wówczas problem odtworzenia semantyki drogą inżynierii wstecznej.

Dlatego wybór formatu tekstu Wikipedii do dalszej analizy jej treści zależy od celu takiej analizy. Jeśli celem jest analiza języka naturalnego, to wygodniejszym formatem jest HTML. Jeśli jednak celem jest pozyskanie oryginalnej informacji semantycznej artykułu, to mimo trudności parsowania trzeba pobierać tekst w formacie Wiki lub drzewa DOM.

Bibliografia

[1] Buffa M., et al. *SweetWiki: A semantic wiki. Journal of Web Semantics*, 6(1):84–97, 2008.

[2] Dauben J.W. *Suan Shu Shu a book on numbers and computations. Archive for history of exact sciences*, 62(2):91–178, 2008.

[3] Denoyer L., Gallinari P. *The wikipedia xml corpus.* In *International Workshop of the Initiative for the Evaluation of XML Retrieval*, pp. 12–19. Springer, 2006.

[4] Dietz L., et al. *TREC Complex Answer Retrieval Overview.* In *TREC*. 2017.

[5] Dohrn H., Riehle D. *Design and implementation of the sweble wikitext parser: unlocking the structured data of wikipedia.* In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, pp. 72–81. 2011.

[6] Dohrn H., Riehle D. *Design and implementation of wiki content transformations and refactorings.* In *Proceedings of the 9th international symposium on open collaboration*, pp. 1–10. 2013.

[7] Flanagin A.J., Metzger M.J. *From Encyclopaedia Britannica to Wikipedia: Generational differences in the perceived credibility of online encyclopedia information. Information, Communication & Society*, 14(3):355–374, 2011.

[8] Flickinger D., Oepen S., Ytrestøl G. *Wikiwoods: Syntacto-semantic annotation for english wikipedia.* 2010.

[9] Jiang M. *The business and politics of search engines: A comparative study of Baidu and Google's search results of Internet events in China. New media & society*, 16(2):212–233, 2014.

[10] Lange D., Böhm C., Naumann F. *Extracting structured information from Wikipedia articles to populate infoboxes.* In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pp. 1661–1664. 2010.

[11] Margaretha E., Lüngen H. *Building linguistic corpora from Wikipedia articles and discussions. Journal of Language Technology and Computational Linguistics. Special issue on building and annotating corpora of computer-mediated communication. Issues and challenges at the interface between computational and corpus linguistics*, 29(2):59–82, 2014.

[12] Sloane A.M., Cassez F., Buckley S. *The sbt-rats parser generator plugin for Scala (tool paper).* In *Proceedings of the 2016 7th ACM SIGPLAN Symposium on Scala*, pp. 110–113. 2016.

[13] Woo E. *Baidu's Censored Answer to Wikipedia. Bloomberg Business*, 2007.

[14] Ytrestøl G., Flickinger D., Oepen S. *Extracting and annotating Wikipedia sub-domains—Towards a new eScience community resource. LOT Occasional Series*, 12:185–197, 2008.
https://www.overleaf.com/project/625fe9baf0335643cb17472e

Rozdział 12

Synchronizacja wiedzy w systemach agentowych

Mariusz Matuszek[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
Mariusz.Matuszek@pg.edu.pl

Abstrakt

Agenty inteligentne są jednym z komponentów stosowanych w projektowaniu rozproszonych inteligentnych systemów obliczeniowych. W rozdziale wskazano istotne aspekty systemów agentowych, a następnie omówiono wybrane metody synchronizacji wiedzy między agentami będącymi częścią systemu agentowego. Omówiono podejście właściwe dla agentów zaufanych oraz jego modyfikację dla agentów, które mogą celowo próbować wprowadzać inne agenty w błąd.

Słowa kluczowe: systemy agentowe, systemy rozproszone, synchronizacja wiedzy

12.1. Agent i system agentowy

Systemy agentowe reprezentują jeden z nurtów projektowania rozproszonych inteligentnych systemów obliczeniowych. Nurt ten stawia przed projektantem

specyficzne wyzwania, związane z naturą agenta inteligentnego. Niniejszy rozdział ma na celu przybliżenie tych wymagań czytelnikowi. Pojęcie agenta jest powszechnie wykorzystywane w informatyce i obejmuje szerokie spektrum zastosowań. Na jednym jego końcu pod pojęciem agenta może kryć się wyspecjalizowany fragment oprogramowania wbudowanego w urządzenie (*firmware*), którego przykładem może być moduł umożliwiający zdalne zarządzanie urządzeniem sieciowym (SNMP agent). Na przeciwnym krańcu spektrum znajduje się agent zdolny do samodzielnego działania i realizowania swojej misji w ramach środowiska agentowego. Ten typ agenta jest czasem nazywany agentem inteligentnym i opisywany pojęciami zbliżonymi do opisu bytów wirtualnych. W szczególności agent taki posiada następujące cechy: autonomia, inteligencja (zdefiniowana jako racjonalność akcji i decyzji), proaktywność i opcjonalnie mobilność.

Niniejszy rozdział zajmuje się agentami inteligentnymi. Aby lepiej skonkretyzować to pojęcie, warto zapoznać się z często spotykaną definicją agenta: *jednostka obliczeniowa (program, robot), która jest zdolna do postrzegania stanu środowiska, w którym się znajduje, oraz oddziaływania na to środowisko, wyposażona w zdolność autonomicznego i inteligentnego decydowania o swoich akcjach w ramach tego środowiska, w celu spełnienia swojej misji.*

Spośród wymienionych cech agentów inteligentnych [1], [6] na szczególną uwagę zasługuje autonomia, ponieważ w decydujący sposób wpływa na dobór i projektowanie algorytmów działania zarówno pojedynczego agenta, jak i aplikacji agentowych, w których wiele agentów współpracuje przy realizowaniu wspólnego celu. Rozwiązania informatyczne (aplikacje, z reguły rozproszone), w których logika działania została w całości lub w zdecydowanej większości zaimplementowana przy użyciu współpracujących agentów, nazywane są systemami agentowymi.

Autonomia oznacza, że w projektowaniu algorytmów dla agentów inteligentnych należy położyć szczególny nacisk na samodzielne decydowanie przez agenta o wyborze i podejmowaniu działania, jak również niewyróżnianie żadnego z agentów w przypadku algorytmów koordynujących współdziałanie grupy agentów. Przykładowo, wskazanie jednego agenta jako koordynatora (algorytmy z grupy *master-slave*) jest podejściem zdecydowanie niepożądanym. Z reguły o wiele lepszym wyborem będą algorytmy z rodziny *peer-to-peer*.

Drugą nie mniej istotną cechą jest proaktywność, czyli zdolność agenta do samodzielnego inicjowania czynności przybliżających agenta do realizacji celu (misji). Jest to cecha zdecydowanie odróżniająca agenty od innych składowych systemów informatycznych, które – co do zasady – są reaktywne, czyli oferują jedynie interfejs usług (API).

12.1.1. Środowisko agentowe

Pojęcie środowiska agentowego ma dwa znaczenia. Pierwsze ma charakter techniczny i oznacza zestaw usług wspierających działanie agentów. Są to m.in. usługi nazewnicze, komunikacyjne oraz migracyjne, pozwalające agentom na przemieszczanie się pomiędzy kontenerami (węzłami środowiska) i wykonywa-

nie swojego kodu. Środowisko w znaczeniu technicznym może mieć wpływ na dobór sprzętu (mocy obliczeniowej) oraz języka programowania agentów, ale nie wpływa na dobór algorytmów ich działania, więc w dalszej części rozdziału przyjęto założenie, że takie środowisko istnieje i dostarcza niezbędnych usług, jako warunek konieczny do działania agentów.

Drugie znaczenie tego pojęcia oznacza zestaw cech środowiska, w którym agenty wykonują swoje zadania (misję). W tym przypadku cechy środowiska mogą odgrywać wręcz kluczową rolę w projektowaniu i doborze algorytmów działania agenta.

12.1.2. Taksonomia cech środowisk agentowych

Ponieważ cechy środowiska, w którym agent wykonuje swoją misję, wywierają istotny wpływ na dobór algorytmów, warto zapoznać się ze standardowo stosowaną taksonomią cech środowisk agentowych. Analizy środowiska dokonuje się według następującego klucza:

- dostępność: środowisko może być dostępne albo niedostępne. Środowisko dostępne to takie, o którego stanie agent może uzyskać pełną i aktualną wiedzę. Należy zauważyć, że większość złożonych środowisk, w szczególności świat człowieka czy sieć Internet, są przykładami środowisk niedostępnych;

- determinizm: środowisko może być deterministyczne albo niedeterministyczne. Środowisko deterministyczne to takie, w którym dana akcja podejmowana przez agenta przynosi zawsze taki sam efekt, niezależnie od liczby powtórzeń tej akcji. Inaczej ujmując, nie występuje niepewność co do stanu środowiska osiąganego w wyniku wykonanej przez agenta czynności. Podobnie jak w przypadku dostępności, świat człowieka jest przykładem środowiska niedeterministycznego;

- kontekstowość: ta cecha występuje, gdy istnieje ciąg przyczynowo-skutkowy między reakcjami środowiska na bieżącą akcję agenta a jego poprzednimi akcjami lub gdy agent wybiera kolejną akcję na podstawie wiedzy zgromadzonej podczas wcześniejszych interakcji z środowiskiem. Bezkontekstowość (epizodyczność) występuje, gdy agent wybiera akcję wyłącznie na podstawie bieżącego stanu środowiska, bez uwzględniania wyników wcześniejszych interakcji;

- statyczność: środowisko jest uznawane za statyczne wtedy, gdy nie zachodzą w nim zmiany inne niż będące następstwem działań agenta. Jeżeli takie zmiany zachodzą, samoistnie bądź w następstwie nieskoordynowanych działań innych agentów, środowisko określane jest mianem dynamicznego;

- ciągłość: środowisko, w którym do opisu jego stanu wymagane są funkcje ciągłe, określamy mianem środowiska ciągłego. Jego przeciwieństwem jest środowisko dyskretne – mogące przyjmować skończoną liczbę stanów. Dobrym przykładem środowiska ciągłego jest świat człowieka (rzeczywisty),

natomiast przykładem środowiska dyskretnego może być gra planszowa (np. warcaby).

12.1.3. Pożądane cechy algorytmów agentowych

Uwzględniając wcześniej wymienione cechy agentów inteligentnych oraz kryteria analizy środowiska, w którym agenty mają współdziałać, można sporządzić listę pożądanych cech algorytmów agentowych:

- symetria: algorytm nie powinien faworyzować, jak również wyróżniać żadnego z agentów;
- prostota: niskie zapotrzebowanie algorytmu na zasoby (moc obliczeniową, pasmo, pamięć);
- stabilność: w przypadku współpracy (kooperacji) agentów algorytm powinien być indywidualnie racjonalny dla każdego agenta, to znaczy, że każdy agent odnosi większą korzyść ze współpracy niż z unikania współpracy;
- rozproszenie: koordynacja procesu współpracy (decyzji, uzgodnień itp.) nie powinna być skoncentrowana w jednym agencie;
- wydajność: pożądana cecha, o ile udało się spełnić wcześniej wymienione. Algorytm nie powinien przeciążać dostępnych zasobów.

12.2. Obserwacja a stan wewnętrzny agenta

Klasyczny cykl działania agenta składa się z trzech faz, ponawianych aż do osiągnięcia celu misji agenta; są to: obserwacja otoczenia, wnioskowanie oraz wybór akcji optymalnej dla osiągnięcia celu.

W ramach fazy obserwacji agent, posługując się dostępnymi (zaimplementowanymi) metodami (sensorami), mierzy istotne z punktu widzenia osiągnięcia celu parametry (stan) otoczenia (środowiska)[1]. Następnie na podstawie uzyskanych pomiarów stanu (faktów o stanie środowiska) przeprowadzany jest proces wnioskowania, pozwalający na wybór właściwej akcji agenta. Akcja może dotyczyć próby wpłynięcia na stan środowiska albo skutkować jedynie zmianą stanu wewnętrznego agenta. Istotne będzie rozróżnienie pojęć faktu oraz wniosku:

- fakt (inaczej: obserwacja) – mierzalna wartość dotycząca cech otoczenia (środowiska agentowego)[2];

[1] Warto zauważyć, że w środowiskach ciągłych niektóre parametry można mierzyć, kwantyfikując wyniki pomiaru (np. mierzyć temperaturę otoczenia z zaokrągleniem do pełnych stopni), co sprowadza postrzeganie tej cechy środowiska do postaci dyskretnej, ale nie zmienia natury środowiska. Taki zabieg może być wystarczający w niektórych zastosowaniach agentów.

[2] W niektórych przypadkach fakty mogą dotyczyć również stanu wewnętrznego danego agenta, na przykład: *naładowanie mojego akumulatora wynosi 30%* albo: *pamięć zapełniona w 99%*.

- wniosek – element stanu wewnętrznego agenta, będący produktem subiektywnego procesu wnioskowania.

Należy podkreślić, że różne agenty mogą dochodzić do różnych wniosków na podstawie tych samych faktów. Na przykład agent może zmierzyć temperaturę otoczenia: *temperatura zewnętrzna = 230 K* i na tej podstawie wnioskować: *jest zimno*, ale inny agent na podstawie takiego samego pomiaru może wnioskować: *jest za ciepło*, przy czym, o ile pierwszy z wniosków jest zwykłą klasyfikacją mierzonej wartości, o tyle drugi wniosek jest oceną, która implikuje obecność w agencie jakiejś reguły wnioskowania, która w kolejnym kroku może skutkować podjęciem akcji. W tym przypadku można oczekiwać, że taką akcją będzie próba obniżenia temperatury otoczenia albo migracja do innej części otoczenia[3].

Opisany proces odbywa się indywidualnie wewnątrz każdego agenta, jednakże w systemach agentowych rzadko mamy do czynienia z pojedynczym agentem realizującym misję samodzielnie. O wiele częściej zachodzi sytuacja, w której logika aplikacji agentowej realizowana jest przez zbiór współpracujących agentów, nierzadko rozproszonych [2]. W tego typu realizacjach często korzystne jest, aby agenty okresowo wymieniały się informacją o stanie środowiska. Z punktu widzenia pojedynczego agenta taka synchronizacja może pozwolić zarówno zweryfikować wiedzę już posiadaną, jak też pozyskać nową. Z punktu widzenia oceny działania aplikacji wieloagentowej indywidualna weryfikacja posiadanych przez agenty faktów i pozyskiwanie nowych faktów wpływają korzystnie na jakość działania aplikacji[4]. Należy rozpatrzyć sytuacje, w których współpracujące agenty są w pełni zaufane (np. pochodzą od jednego dostawcy) oraz gdy agenty wchodzą w interakcje z agentami niezaufanymi (np. w środowiskach otwartych, w których współistnieją agenty z różnych źródeł). Każda z tych sytuacji wymaga innego podejścia w konstruowaniu algorytmu synchronizacji stanu.

12.3. Synchronizacja stanu między agentami zaufanymi

Skoro synchronizacja stanu jest indywidualnie korzystna dla każdego z agentów wchodzących w skład aplikacji agentowej oraz wpływa korzystnie na jakość samej aplikacji (agenty wnioskują na podstawie zaktualizowanej i uspójnionej wiedzy), to spełniony jest wymóg stabilności i można przyjąć, że w ramach proaktywności agent będzie (na przykład co jakiś czas, mierzony od ostatniej synchronizacji) rozsyłał zaproszenie do wspólnej synchronizacji do innych agentów. W tym celu agent, korzystając z usług środowiska agentowego, może skonfigurować przestrzeń wymiany informacji – na przykład grupę komunikacyjną – oraz rozesłać zaproszenie. Należy podkreślić, że jest to zaproszenie, a nie żądanie,

[3] W zależności od algorytmu/misji agenta i przy założeniu, że takie akcje są możliwe do wykonania.

[4] Przy milczącym założeniu, że agenty posiadają reguły wnioskowania pozwalające wykorzystać pozyskaną wiedzę.

ponieważ w ramach swojej autonomii każdy agent ma prawo takie zaproszenie zignorować. W języku nieformalnym takie zaproszenie może przyjąć formę:

```
Tu agent A z aplikacji XYZ, chcę zsynchronizować stan.
Zapraszam do grupy komunikacyjnej G.
```

Zakładając, że inne agenty uznają, że taka synchronizacja jest dla nich indywidualnie korzystna, mogą odpowiedzieć:

```
Tu agent B z aplikacji XYZ, przyjmuję zaproszenie do grupy
G.
Tu agent C z aplikacji XYZ, przyjmuję zaproszenie do grupy
G.
...
```

i dołączyć do grupy. Jeżeli żaden agent nie odpowiedział na zaproszenie, algorytm się kończy, a zaproszenie zostanie ponowione za jakiś czas, ponieważ jest to działanie indywidualnie racjonalne.

W przypadku sformowania grupy agenty rozpoczynają wymianę faktów. Sygnałem do rozpoczęcia wymiany może być na przykład upłynięcie z góry określonego czasu od ostatniego dołączenia agenta do grupy. Od tej chwili algorytm składa się z trzech faz: rozgłoszenia znanych faktów, integracji otrzymanych faktów, wysłania informacji o zakończeniu transmisji własnych faktów. Przy założeniu, że grupa komunikacyjna umożliwia komunikację rozgłoszeniową (*multicast*), rozgłoszenie faktów przez agenta polega na wysyłaniu do grupy kolejnych zdań:

```
Tu agent <id>, <opis faktu>
```

gdzie opis faktu jest stwierdzeniem wyrażonym w ontologii właściwej dla danej aplikacji. Tu zakładamy, że opis ten jest dla innych agentów zrozumiały. Na przykład:

```
Tu agent B, temperatura_otoczenia = 240K
```

Po wypowiedzeniu wszystkich znanych sobie faktów agent informuje pozostałe agenty, np:

```
Tu agent B, koniec transmisji.
```

jednak pozostaje w grupie tak długo, aż nie odbierze znaczników końca transmisji od wszystkich pozostałych uczestników grupy. Dopiero po ich otrzymaniu agent może uznać wymianę faktów za zakończoną i opuścić grupę. Gdyby mechanizm komunikacji nie pozwalał na komunikację grupową, a jedynie komunikację dwustronną, (*peer to peer*), algorytm co do istoty pozostaje bez zmian, natomiast każdy z faktów musi zostać indywidualnie wysłany do każdego z pozostałych członków grupy. To samo dotyczy znacznika końca transmisji.

Jak widać, algorytm od strony komunikacyjnej nie jest skomplikowany. Natomiast faza integracji otrzymanych faktów z własnym stanem agenta jest silnie

zależna od własności środowiska agentowego. W przypadku prostego środowiska statycznego (dla przypomnienia: to takie środowisko, w którym nie zachodzą zmiany samoistne) każdy agent może porównać otrzymane fakty z własną wiedzą i na przykład dokonać arytmetycznego uśrednienia albo zmienić swoją wiedzę, korzystając z mechanizmu głosowania większościowego.

Gdyby do cech rozpatrywanego środowiska dodać dynamizm, do opisu rozgłaszanego opisu faktu powinien zostać dodany stempel czasowy, np:

```
Tu agent B, temperatura_otoczenia = 240K, <time>
```

W takim przypadku agent może przyjąć strategię uznania faktu najmniej odległego w czasie za wiedzę własną albo zastosować uśrednianie ważone, w którym pomiary odległe w czasie mają niską wagę.

Jeszcze innej strategii integracji wymaga środowisko kontekstowe (dla przypomnienia: agent może wnioskować na podstawie historii zmian zachodzących w środowisku). W takim przypadku również niezbędny będzie stempel czasowy, ale agent w swojej wiedzy będzie gromadził historię zmian danego elementu stanu, opcjonalnie dokonując uśrednienia pomiarów sobie bliskich.

Osobnej strategii wymaga integracja z wiedzą własną faktów wcześniej agentowi nieznanych. Możliwe spektrum działań obejmuje wszystkie możliwości, od przyjęcia każdego nowego faktu, nawet jeśli pochodzi od jednego agenta w grupie, przez przyjęcie warunkowe, w którym taki fakt otrzymuje jakąś miarę wiarygodności (np. tak twierdzi 5 z 30 agentów), po zupełne odrzucenie faktu. Przyjęcie właściwej strategii integracji, przy uwzględnieniu możliwego skomplikowania reguł wnioskowania agenta, jest silnie zależne od aplikacji.

12.4. Synchronizacja stanu przy braku zaufania

W otwartych środowiskach agentowych, w których może dochodzić do interakcji z agentami pochodzącymi z różnych źródeł, należy liczyć się z tym, że część agentów może z różnych powodów być nieuczciwa i próbować wprowadzać inne agenty w błąd. W szczególności jeden bądź więcej agentów może próbować wysyłać różne wartości tego samego faktu do różnych odbiorców. W związku z istnieniem takiego zagrożenia należy wyposażyć agenty w dodatkowe umiejętności pozwalające na weryfikację otrzymanych faktów.

W omówionym wyżej scenariuszu, w którym agenty korzystają z komunikacji rozgłoszeniowej, tego typu zagrożenie nie może zaistnieć, ponieważ każdy agent transmituje swoją wartość danego faktu tylko raz i ta transmisja jest widziana przez wszystkie agenty tworzące grupę komunikacyjną. Niestety, nie wszystkie środowiska pozwalają wykorzystać ten model komunikacji i do dyspozycji pozostaje jedynie komunikacja dwustronna. W takim przypadku opisana próba oszustwa może stanowić realne zagrożenie. Rozsyłanie różnych wartości tego samego faktu przez jednego agenta może być też następstwem błędu w kodzie agenta lub jego uszkodzenia.

Aby uodpornić rozproszony system agentowy na tego typu sytuacje (awarie komponentów lub ich celowo błędne działanie), można posłużyć się algoryt-

mem znanym jako algorytm bizantyjskich generałów albo problem bizantyjskich generałów [3], [4], [5], którego dokładne omówienie wykracza poza ramy tego rozdziału. W pewnym uproszczeniu algorytm ten zakłada rozbicie komunikacji między stronami na dwie fazy. W pierwszej fazie agenty komunikują sobie swoje fakty jak dotąd. W drugiej fazie agenty komunikują sobie wzajemnie, co im zakomunikowały inne agenty, i na tej podstawie próbują wykryć rozbieżności. Przykładowo, w fazie pierwszej:

```
Agent A do B: jest zimno
Agent A do C: jest ciepło
Agent B do C: jest ciepło
Agent B do A: jest ciepło
Agent C do A: jest ciepło
Agent C do B: jest ciepło
```

W takim przypadku w fazie drugiej perspektywa agentów B i C wygląda tak:

```
Agent C do B: agent A powiedział, że jest ciepło
Agent B do C: agent A powiedział, że jest zimno
```

W efekcie agenty B i C mogą wykryć, że agent A wysyła niespójny komunikat.

Bibliografia

[1] Alkhateeb F., Maghayreh E.A., Doush I.A. *Multi-Agent Systems – Modeling, Control, Programming, Simulations and Applications*. IntechOpen, Rijeka, 2011. ISBN 978-5-533-07174-9.

[2] Capezzuto L., Tarapore D., Ramchurn S.D. *Large-Scale, Dynamic and Distributed Coalition Formation with Spatial and Temporal Constraints*. In A. Rosenfeld, N. Talmon, eds., *Multi-Agent Systems*, pp. 108–125. Springer International Publishing, Cham, 2021. ISBN 978-3-030-82254-5.

[3] Leslie L., Robert S., Marshall P. *The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst.*, 4(3), 1982. ISSN 0164-0925.

[4] Pease M.C., Shostak R.E., Lamport L. *Reaching Agreement in the Presence of Faults. J. ACM*, 27(2):228–234, 1980.

[5] Wikipedia. *Problem bizantyjskich generałów – Wikipedia, The Free Encyclopedia*. `http://pl.wikipedia.org/w/index.php?title=Problem%20bizantyjskich%20genera%C5%82%C3%B3w&oldid=55547901`, 2022. [Online; accessed 01-June-2022].

[6] Wooldridge M. *An Introduction to Multiagent Systems*. Wiley, Chichester, UK, 2 ed., 2009. ISBN 978-0-470-51946-2.

Rozdział 13

Algorytm mrówkowy do zarządzania zasobami sprzętowymi chmury obliczeniowej w przypadku różnych kategorii usług

Henryk Krawczyk[1,2] , Piotr Orzechowski[2]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
hkrawk@eti.pg.edu.pl

[2] Centrum Informatyczne TASK
Politechnika Gdańska
piotr.orzechowski@pg.edu.pl

Abstrakt

Zarządzanie chmurą obliczeniową odbywa się na dwóch poziomach: zarządzanie żądaniami klientów chmury oraz zarządzanie jej infrastrukturą, na której te usługi są realizowane. Analizując standardy dotyczące zarządzania usługami, w niniejszym rozdziale skoncentrowano się na drugim poziomie zarządzania, którego głównym celem jest efektywne wykonanie wskazanej usługi (lub usług) na dostępnych zasobach sprzętowych, tak by spełnione zostały przyjęte kryteria optymalizacyjne. Dotyczy to przede wszystkim algorytmów przydziału (alokacji) niezbędnych zasobów dla każdej z usług

oczekujących na wykonanie, przy uwzględnieniu wymagań wynikających z zawartych umów pomiędzy użytkownikiem a dostawcą takich usług (SLA). Zaproponowano algorytm mrówkowy, który jest akceptowalny dla różnych kategorii zgłaszanych przez użytkowników żądań. Jego działanie zilustrowano na trzech konkretnych przykładach wykonywanych aplikacji. Dalsze eksperymentalne badania jakości proponowanego algorytmu są przewidywane po jego implementacji w systemie zarządzania lokalnej chmury TASKcloud.

Słowa kluczowe: chmura obliczeniowa, umowa SLA, zarządzanie zasobami, alokacja usług, algorytm mrówkowy

13.1. Wprowadzenie

Usługi IT przetwarzane w chmurze obliczeniowej są usługami chmurowymi. Zbiór zasad i praktyk dotyczących wdrażania i dostarczania usług informatycznych oraz administrowania nimi dla użytkowników końcowych w sposób, który spełni ich określone potrzeby oraz wyznaczone cele biznesowe, jest określany jako zarządzanie usługami informatycznymi (*Information Technology Service Management*, ITSM [17]). Istnieją 34 praktyki zarządzania zgodne z ITSM, przy czym organizacje stosują najczęściej następujące spośród nich: zarządzanie incydentami (reagowanie na incydent w celu przywrócenia usługi); zarządzanie problemami (identyfikowanie i usuwanie pierwotnej przyczyny incydentu, dodatkowo też czynników wywołujących taką przyczynę źródłową, a także określanie najlepszego sposobu ich wyeliminowania); zarządzanie zmianami (wdrażanie zmian, które minimalizują zakłócenia w świadczeniu usług); zarządzanie zasobami i konfiguracją (autoryzacja, monitorowanie i dokumentowanie konfiguracji zasobów oprogramowania i sprzętu wykorzystywanych do świadczenia usług); zarządzanie zleceniami usług (obsługa zleceń nowych usług od poszczególnych użytkowników); zarządzanie wiedzą (generowanie i udostępnianie wiedzy związanej z usługami IT); zarządzanie poziomem świadczonych usług (uzgadnianie wymaganych lub pożądanych poziomów usług dla różnych grup użytkowników); centrum wsparcia IT (biuro pomocy – helpdesk – zarządzania wszystkimi incydentami, problemami i żądaniami dotyczącymi wykonywanych i dostępnych usług). Rozwiązania ITSM zmieniają się wraz z potrzebami i technologiami organizacji, które z nich korzystają. Pojawiają się też nowe propozycje dotyczące obsługi Internetu rzeczy (*Internet of Things*, IoT [12]). Urządzenia tego typu nieustannie gromadzą cenne dane, w tym status pracy, lokalizację i zmiany. Zatem rozwiązania ITSM wskazują sposoby monitorowania i wykorzystywania tych danych w celu usprawnienia rozwiązywania bieżących problemów i podejmowania decyzji biznesowych. Z uwagi na to, że coraz większą rolę odgrywają systemy społecznościowe [10], narzędzia ITSM ukierunkowane są na dostosowy-

wanie kanałów komunikacji do użytkownika, dzięki czemu postępuje integracja z mediami społecznościowymi, co pozwala, przy wykorzystaniu sztucznej inteligencji, usprawniać i rozwijać procesy zarządzania klientami, np. precyzyjniej oceniać nastroje klientów i włączyć to wymaganie do systemów zarządzania.

Z kolei ITIL (*Information Technology Infrastructure Library* [16]) to biblioteka najlepszych praktyk stosowanych w zarządzaniu usługami IT. Wyróżniamy pięć kluczowych etapów ITIL (składających się z 26 procesów): strategia usługi (cykl życia usług ITIL, tzn. jak zaprojektować, opracować i wdrożyć zarządzanie usługami IT); projektowanie usług (metody wytwarzania usług i procesów); zmiana usług (sposoby zarządzania przejściem na nową lub zmienioną usługę); funkcjonowanie usług (zagwarantowanie dostawy usług oraz ich płynne i niezawodne funkcjonowanie); ciągłe doskonalenie usług (dostosowanie usługi IT do zmieniających się potrzeb biznesowych – CSI, *Continual Service Improvement*).

Bardzo istotną rolę przy alokacji i wykonaniu usług odgrywają również zapisy określonej umowy o poziomie świadczonych usług – SLA (*Service Level Agreement* [13]). Jest to rodzaj umowy pomiędzy użytkownikiem usług a dostawcą usług, szczegółowo określający zakres oferowanych usług, w sposób umożliwiający ich weryfikację poprzez wskazane ich parametry. SLA stanowi standardowy zapis głównie w przypadku usług dostarczanych przez dostawców serwerów, chmur i platform hostingowych. Dostawca określa w niej dostępność, rozumianą jako zdolność infrastruktury bądź systemu do realizacji przydzielonych usług w określonej jednostce czasu (zwykle miesiąc/rok – zazwyczaj oscylująca wokół 99% czasu użytkowania). Na tej podstawie określa się sposób monitorowania systemu oraz raportowania działań i zdarzeń. Oprócz tego w SLA zawarte powinny być zapisy dotyczące zgłaszania problemów, funkcjonowania pomocy technicznej, limitu transferów oraz przepustowości łączy, a także estymacji czasu reakcji na zdarzenia, napraw i/lub obejścia błędu czy przywracania kopii zapasowych. Zagwarantowane powinny być parametry jakościowe sieci (dostępność, przepustowość, dostępność protokołów, opóźnienia, liczba możliwych błędów), zasilania (liczba i długość przerw), chłodzenia, czas reakcji na zgłoszenie, czas skutecznej wymiany sprzętu na sprawny czy możliwość zlecania usług dodatkowych zarówno dla wszystkich właścicieli firm, jak i dla poszczególnych klientów. SLA specyfikuje więc wymagania jakościowe (QoS) pomiędzy dostawcą serwisów oraz ich klientem, zależną od poziomu jakości cenę za dostarczane i realizowane usługi. Z reguły uwzględniają one interesy dostawców, w mniejszym stopniu biorą pod uwagę oczekiwania potencjalnych użytkowników. Tylko dla najważniejszych klientów istnieje możliwość zawarcia kompromisu poprzez przeprowadzenie negocjacji. Najczęściej uwzględnia się takie parametry jakościowe, jak: minimalny czas odpowiedzi na żądanie, minimalny koszt obsługi żądania, minimalne zużycie energii, zapewnienie wysokiego poziomu wiarygodności funkcjonowania chmury czy minimalizacja naruszenia porozumienia SLA.

W tablicy 13.1 przedstawiono w kolejności malejącej charakterystykę najbardziej popularnych chmur obliczeniowych, których wykorzystanie na rynku waha się od 2% do 33%. Spośród wielu istniejących rozwiązań chmurowych zasygnalizowano jedynie niektóre, chcąc podkreślić, że ich liczba i zakres możliwości

stale wzrastają. Pojawiają się też wyspecjalizowane narzędzia dotyczące zarządzania chmurą, bazujące na sztucznej inteligencji czy analityce Big Data lub systemach IoT. Jednak wiele rozwiązań praktycznych poprzedzonych było badaniami teoretycznymi. Dotyczy to zwłaszcza zakresu problemów zarządzania chmurą, w którym nadal wiele problemów jest sprawą otwartą [8].

W niniejszym rozdziale ograniczamy się jedynie do problemów zarządzania infrastrukturą chmurową, zakładając, że sprawna obsługa żądań użytkowników powinna być szczególnie brana pod uwagę. Z reguły zarządzanie chmurą obliczeniową umożliwia zachowanie wysokiej dostępności zasobów, optymalnego poziomu bezpieczeństwa oraz łatwej skalowalności zasobów. Koncentruje się ono na monitorowaniu kluczowych elementów infrastruktury i usług czy aplikacji działających w środowisku chmurowym oraz ich konfiguracji. Wskazuje na kontrolowanie obciążenia aplikacji oraz sposobu przechowywania danych. Wszystko to przekłada się na efektywniejsze wykorzystanie możliwości chmury przy jednoczesnym zachowaniu maksymalnego bezpieczeństwa jej zasobów.

W niniejszym rozdziale zaproponowano heurystyczny algorytm alokacji, który spełnia wyżej wymienione oczekiwania. W sekcji 13.2 zaprezentowano problemy zarządzania na poziomie maszyn wirtualnych i kontenerów oraz podstawowe praktyczne pakiety zarządzania chmurą. W sekcji 13.3 zdefiniowano problem alokacji i podano kryteria optymalizacji. W sekcji 13.4 zaproponowano uogólniony algorytm mrówkowy oraz jego wykorzystanie do rozwiązania problemu alokacji. W sekcji 13.5 zaprezentowano i skomentowano przykład wykorzystania tego algorytmu, by w uwagach końcowych zaakcentować uniwersalność przedstawionego rozwiązania.

13.2. Praktyka zarządzania chmurą i zasobami chmurowymi

Samo wdrożenie chmury może przynieść już sporo korzyści: zapewnia elastyczność i skalowalność zasobów, poprawę dostępności i ciągłości biznesu, redukcję kosztów i złożoności, przesunięcie wydatków na IT z modelu CAPEX (inwestycje) na rzecz OPEX (utrzymanie), przyspieszenie procesów dostarczania zasobów oraz wdrożeń usług i aplikacji, a także zwiększenie elastyczności i zwinności działania. Nie oznacza to jednak zaniechania dalszych usprawnień w zakresie architektury chmury, zarządzania klientami i zasobami, jak też ich wzajemnej współpracy. I tak istotne cechy przydziału zasobów w chmurze obliczeniowej można określić w kilku płaszczyznach dotyczących architektury chmury (maszyny wirtualne, kontenery), poziomu oferowanych usług (IaaS, PaaS, SaaS) czy zawartych umów SLA.

Tablica 13.1: Charakterystyka popularnych chmur obliczeniowych [1]

Nazwa chmury	Usługi obliczeniowe	Usługi magazynu danych	Usługi sieciowe	Narzędzia wytwarzania	Narzędzia zarządzania	Analityka Big Data	Sztuczna Inteligencja	IoT
AWS (Amazon)	Amazon Elastic Kubernetes Service	Amazon Glacier	Amazon Route Traffic Flow	Amazon DevOps Guru	AWS Management Console	Amazon Kinesis	Amazon Kendra	AWS Greengrass
Azure (Microsoft)	Azure VMware Solution	Archival Storage	Azure Traffic Manager	Azure SDK Visual Studio	Resource Manager	Azure Data Factory	Azure Machine Learning	Azure IoT Edge
Google Cloud	Google Cloud VMware Engine	Cloud Storage	Cloud DNS	Cloud Tools for Android Studio	Cloud Shell	Cloud Data Fusion	Cloud Video Intelligence	Google Cloud IoT
IBM Cloud	VMware Solutions Shared	Object Storage-ColdVault	DNS Services	BM Cloud Developer Tool	Schematics	Watson studio	Watson OpenScale	Internet of Things Platform
Oracle Cloud	Container Engine for Kubernetes (OKE)	Oracle File Storage Services	Oracle DNS	Oracle Developer Tools for Visual Studio	Oracle Management Cloud	Oracle Business Intelligence Cloud Service	Oracle Machine Learning	IoT Cloud Service
Alibaba Cloud	Container Service for Kubernetes	Hybrid Cloud Storage Array	Alibaba Content Delivery Network	Alibaba Cloud CLI	Resource Orchestration Service	E-MapReduce Service	Machine Learning Platform For AI	IoT Platform

Obecnie istnieją dwie podstawowe architektury chmury obliczeniowej związane z wykorzystaniem jednostek wirtualnych: maszyn wirtualnych (*virtual machine* [9]) oraz kontenerów (*container* [14]). Dzięki wirtualizacji jest możliwe tworzenie wielu maszyn wirtualnych na jednej maszynie fizycznej. Każda maszyna wirtualna posiada własny system operacyjny (OS) i aplikacje, które są przechowywane jako obrazy. Maszyna wirtualna nie może wchodzić w bezpośrednią interakcję z fizycznym komputerem. Zamiast tego potrzebuje lekkiej warstwy oprogramowania (tzw. *hypervisor*), która koordynuje komunikację między maszyną a komputerem. Ta warstwa przydziela fizyczne zasoby obliczeniowe – procesory, pamięć operacyjną i pamięć masową – poszczególnym maszynom wirtualnym. Dzięki temu każda z maszyn wirtualnych działa niezależnie, nie zakłócając pracy pozostałych. Innymi słowy, umożliwia odizolowanie systemu operacyjnego i aplikacji tego komputera od sprzętu. Co więcej, takie rozwiązanie zapewnia uruchamianie różnych systemów operacyjnych na tym samym serwerze, jak również wydajne i ekonomiczne wykorzystanie zasobów fizycznych. Często maszyny wirtualne (emulacja komputera fizycznego) są nazywane gośćmi (guests), podczas gdy komputer fizyczny, na którym działają, jest określany jako gospodarz (host). Goście traktują zasoby maszyny wirtualnej, takie jak procesor czy pamięć RAM, urządzenia sieciowe i dyskowe, jako pulę procesów obliczeniowych, którą można łatwo organizować i nią zarządzać. Operatorzy mogą efektywnie kontrolować takie wirtualne zasoby, a goście otrzymują je tylko wtedy, kiedy ich potrzebują. Użytkownik nie widzi żadnych różnic pomiędzy maszyną wirtualną a sprzętem, jaki mógłby mieć fizycznie obok siebie. W przypadku przetwarzania w chmurze maszyny wirtualne są zazwyczaj oferowane w wariantach zarówno dla jednego podmiotu użytkującego, jak i dla wielu takich podmiotów. W tym drugim przypadku infrastruktura fizyczna jest współdzielona przez kilku użytkowników. Jest to najbardziej korzystne cenowo i skalowalne rozwiązanie, ale brak mu niekiedy niektórych cech zapewnianych przez izolację maszyn, bardzo cenionych przez wiele organizacji oraz kładących nacisk na bezpieczeństwo i zapewnianie zgodności.

W przypadku uwzględnienia zasobów wielochmurowych oraz urządzeń Internetu Rzeczy lepszym rozwiązaniem niż maszyny wirtualne (VM) są kontenery. Rozwiązania kontenerowe opakowują kod i wszystkie zależne biblioteki, pliki binarne czy konfiguracyjne, tworząc tzw. obraz. Zamiast wirtualizować podstawowy sprzęt, wirtualizują system operacyjny i tym samym umożliwiają właśnie pakowanie i wdrażanie aplikacji w niezmienną infrastrukturę, którą można testować jako pewną jednostkę i wdrażać w systemie operacyjnym hosta w różnych środowiskach jako wystąpienie obrazu kontenera. Kontenery korzystają z systemu operacyjnego hosta, dlatego nie trzeba na nich uruchamiać systemu operacyjnego ani ładować bibliotek. Eliminuje to różnego rodzaju zależności pomiędzy systemem operacyjnym i aplikacją. Standardowy format kontenerów umożliwia pakowanie i przechowywanie wszystkich składników wymaganych do uruchomienia danej aplikacji. Dlatego każdy kontener zawiera nie tylko aplikację, ale też jej powiązania i biblioteki. Brak systemu operacyjnego gościa sprawia, że kontenery są lekkie, a tym samym szybkie i przenośne. Kontener jest uruchomioną instancją naszego obrazu, ale z jednego obrazu możemy uru-

chomić wiele instancji – kontenerów, które będą działać równolegle. Jest to coś bardzo ulotnego, ponieważ po wyłączeniu procesu, pracującego wewnątrz kontenera, kontener ten też przestaje istnieć. Dlatego dane przechowywane są na zewnątrz kontenera, na przykład w katalogu. Katalog ten jest podmapowany wewnątrz kontenera jako wolumen. Zmiany wykonywane w katalogu (na zewnątrz) oraz w wolumenie (wewnątrz) są automatycznie synchronizowane. Po wyłączeniu kontenera dane nie są usuwane. Obraz kontenera to zatem prosty, samodzielny, wykonywalny pakiet oprogramowania, który zawiera wszystkie komponenty niezbędne do uruchomienia aplikacji: kod źródłowy, środowisko uruchomieniowe (*runtime*), narzędzia systemowe, biblioteki systemowe i pliki konfiguracyjne. W ten sposób deweloperzy dostają do dyspozycji narzędzie do zwinnego wdrażania aplikacji, które pozwala zespołom na wdrożenie procesu CI/CD – ciągłej integracji i ciągłego wdrażania (*Continuous Integration / Continuous Deployment*). W połączeniu z systemem orkiestracji, takim jak Kubernetes, można zarządzać uruchamianiem tego samego kontenera aplikacji na dowolnym systemie operacyjnym Linux. Dzięki temu możemy zapewnić dokładnie takie samo środowisko uruchomieniowe i zagwarantować spójność aplikacji oraz jej łatwe przenoszenie pomiędzy środowiskami: deweloperskim, testowym, akceptacyjnym i produkcyjnym. Dodatkowym atutem kontenera jest szybkość jego uruchomienia. Zyskujemy przez to większą efektywność i wydajność pracy klastra, w którym nasza aplikacja jest rozproszona horyzontalnie na wielu węzłach, gdy zależy nam na tym, aby skalowała się ona automatycznie w zależności od bieżącego obciążenia (poprzez włączanie i wyłączanie mikroserwisów działających w kontenerach). Kontenery stają się coraz bardziej powszechne w hybrydowym środowisku chmurowym, ponieważ mogą w spójny sposób pracować na laptopach, w chmurze i tradycyjnie, w środowisku lokalnym. W organizacjach, w których ze względów bezpieczeństwa dla określonych aplikacji wymagane jest zapewnienie ich separacji na poziomie jądra systemu, lepszym rozwiązaniem będą wirtualne maszyny. Aczkolwiek już dzisiaj w świecie kontenerów pojawiają się możliwości izolacji na poziomie hiperwizora, więc ten argument powoli przestaje mieć znaczenie, przynajmniej dla kontenerów linuksowych [15].

Wszystkie węzły obliczeniowe (*workers*) czy przechowujące dane (*storage*) pracują pod kontrolą dodatkowych węzłów zarządzających (*management*). To one kontrolują działanie wszystkich usług uruchomionych w chmurze oraz uruchamiają nowe instancje usługi w ramach potrzeby. Wykorzystują do tego np. oprogramowanie Kubernetes. Dodatkowym typem węzła mogą być węzły monitorujące oraz węzeł agregujący logi dotyczące realizowanych usług i aplikacji. Bez centralnego mechanizmu zbierania logów systemowych praktycznie niemożliwy byłby monitoring zarówno logów rozproszonych, jak i zreplikowanych usług użytkownika.

Realizacja usług inicjowana jest na żądanie użytkowników chmury posiadających umowę z jej dostawcą (SLA). Klient uzyskuje możliwości obliczeniowe (dostęp do usług chmurowych, czasu serwera oraz określonej wielkości pamięci) bez interakcji za każdym razem z dostawcą chmury. W tym celu wykorzystuje standardowe mechanizmy, takie jak przeglądarka, zaś dostęp do zasobów chmurowych po zalogowaniu jest niezależny od typu urządzenia wejściowego i jego

lokalizacji. Z kolei elastyczne użytkowanie zasobów, wirtualne i fizyczne zasoby chmury obliczeniowej oferowane przez dostawców są wykorzystywane wielokrotnie w sposób dynamiczny, w pewnych przypadkach automatycznie, poprzez przydzielanie i zwalnianie zasobów zgodnie z potrzebami użytkowników. Klient nie ma żadnej kontroli ani wiedzy na temat lokalizacji fizycznych zasobów, ale może specyfikować ich lokalizację na wyższym poziomie abstrakcji, takim jak miasto, województwo czy kraj. Dostarczanie zasobów może odbywać się w dowolnym czasie, podobnie jak dostarczana jest energia elektryczna. Systemy chmurowe automatycznie kontrolują, optymalizują i monitorują wykorzystanie zasobów na bieżąco, a także dokonują rozliczenia w zależności od zakresu ich wykorzystania oraz dostarczają odpowiednie raporty dla użytkowników. Trudno jest jednak przewidzieć wielkość strumieni żądań użytkowników czy zakres zasobów wymaganych dla ich realizacji w danym momencie czasu. Dostępne są konsole zarządzania, które zapewniają jeden specjalistyczny punkt dostępu dotyczący korzystania z usług, śledzenia stanu przetwarzania czy uzyskania miesięcznych rozliczeń. Użytkownik może też wybrać system operacyjny, jakiego chce używać, wdrażać interfejs API czy powstałe wersje kodu usługi bądź aplikacji, a także dodawać i usuwać różnego typu usługi. Można określać też poziom bezpieczeństwa dla danych czy aplikacji. Oferowany jest szeroki zestaw materiałów pomocy technicznej oraz praktycznych rad i studiów przypadku.

Zarządzanie zasobami wymaga również odpowiednich procedur, które są implementowane przez specjalistyczne oprogramowanie jako odpowiednie usługi systemowe wykonujące się na różnych węzłach chmury obliczeniowej. Pakiety zarządzania pozwalają na uproszczone równoważenie obciążeń w serwerach aplikacji biznesowych. Równoważenie obciążenia między serwerami pozwala użytkownikowi wydłużyć czas dostępności i z łatwością skalować aplikacje przez dodawanie lub usuwanie serwerów, a przy tym tylko minimalnie zakłócać ruch danych. Często pozwala to też eliminować ruch odbywający się za pośrednictwem protokołu SSL i prowadzić komunikację przy użyciu zwykłego tekstu, aby zmniejszyć obciążenie serwerów aplikacji zaplecza. Dzięki redukcji obciążenia SSL serwery mogą spędzać większość swoich cykli pracy na rzeczywistym przetwarzaniu aplikacji. Regularnie monitorowany jest stan serwera, by uniknąć kierowania ruchu do unieruchomionych (uszkodzonych) serwerów.

W rozwiązaniu Kubernetes wyróżnić można cztery zasadnicze warstwy dotyczące zarządzania. Najniżej położona jest warstwa fizyczna odpowiadająca wykorzystanemu sprzętowi chmury. Kolejna to warstwa systemowa, w której uruchomione są kontenery dockerowe wymagane przez oprogramowanie Kubernetes, zdolne do działania. Następną warstwą jest warstwa wsparcia, w której uruchomione są aplikacje użytkownika wspomagane przez działanie usług dotyczących systemu plików, np. GlusterFS, oraz sieci wirtualnej Weave. Ostatnia warstwa to warstwa aplikacji, w której uruchamiane są usługi użytkownika. Wszystkie funkcjonalności aplikacji powinny być podzielone na skalowalne usługi, które mogą być publikowane na wielu węzłach. Jest to właśnie architektura mikroserwisów, która jest w takiej sytuacji naturalnym rozwiązaniem [3]. Należy jednak spełnić pewne dodatkowe wymagania, aby zapewnić efektywne i niezawodne wykorzystanie tej architektury. Usługi powinny być pogrupowane w logiczne,

spójne kontrolery. Każdy kontroler powinien posiadać własny punkt dostępu o unikatowym adresie URL dotyczącym metody REST. Każda usługa jest bezstanowa i odpowiedzialna za jedną logiczną i atomową operację użytkową. Po zapewnieniu kontroli nad tym, które dane są odbierane przez usługi, konieczne jest jeszcze sprawdzenie, kto wysyła żądania. Dotyczy to bezpieczeństwa, które musi być zapewnione na dwóch poziomach. Pierwszy poziom oznacza, że system musi sprawdzić, czy nadawca żądania jest zaufany i nie będzie możliwy nieautoryzowany dostęp. Drugi oznacza, że system musi sprawdzić, czy ten zaufany nadawca ma prawa dostępu do tej wybranej usługi. Autoryzacja żądania powinna sprawdzać nie tylko, czy autor żądania jest właściwy, ale także to, czy to żądanie jest wciąż aktualne oraz czy nie został podmieniony żaden parametr żądania podczas jego przesyłania. Dlatego w każdym żądaniu należy wysłać identyfikator nadawcy, datę wysłania, sumę kontrolną generowaną z całego żądania oraz token. Tajny klucz do generacji tokenu powinien być dostarczony klientowi poza systemem i najlepiej regularnie zmieniany. W przypadku, gdyby ktoś przechwycił żądanie i chciał je wysłać jeszcze raz, zabezpieczenie w postaci walidacji daty ważności tokenu nie pozwoli na dostęp do usługi. Gdyby przechwycone żądanie zostało zmienione (data ważności zostałaby przedłużona lub zamieniony zostałby dowolny parametr żądania), to suma kontrolna nie będzie się zgadzać i nie będzie możliwe wykonanie usługi, ponieważ bez tajnego klucza nie można wygenerować prawidłowej sumy kontrolnej. Tak więc elementy bezpieczeństwa są zapewniane również poprzez odpowiednie zarządzanie dostępem do usług [2]. W umowie SLA powinien zatem być uwzględniony wymagany poziom bezpieczeństwa.

Praktyczne zarządzanie chmurą jest dużym wyzwaniem związanym z zapewnieniem uzgodnionych wymagań SLA, w tym określeniem właściwych globalnych procedur zarządzania chmurą, przyjęciem skalowalnych inżynierskich metod zarządzania jej zasobami, predykcją wydajności wykonywanych aplikacji chmurowych, zrozumieniem ekonomicznych oczekiwań i właściwej wyceny usług chmurowych, a nawet opracowaniem paradygmatów rozwoju oraz budowy usług mobilnych w chmurach. Ciągle istnieje problem, jak alokować i zarządzać efektywnie zasobami chmurowymi dla różnych architektur czy różnych ich implementacji. Z powodu rozwoju różnych architektur oraz chmurowych sieci i modeli obliczeń (chmury sfederowane, migracja VM, wielu pośredników wykorzystania chmury) złożoność procesów zarządzania dramatycznie rośnie. Dostawcy chmury i konsumenci jej usług wykorzystują różnych pośredników i określenie, kto bezpośrednio odpowiada za zarządzanie zasobami chmury (klienci, pośrednicy, dostawcy centrów danych czy usług), nie jest wcale łatwe. Zwłaszcza w przypadkach naruszeń SLA trudno ustalić winowajcę, jeszcze trudniej w przypadku architektury wielochmurowej. Istotne stają się więc zmniejszenie operacyjnego ryzyka niedostarczenia usług na czas, zapewnienie wysokiego bezpieczeństwa oraz zapewnienie profesjonalnego wsparcia. Praktyka zarządzania chmurą jest zatem dużym wyzwaniem, które wymaga wykorzystania badań teoretycznych, głównie dotyczących modeli i optymalnych procedur zarządzania.

13.3. Zarządzanie zasobami chmury jako problem optymalizacyjny

Zarządzanie zasobami w środowisku chmurowych jest trudnym problemem z uwagi na skalę współczesnych centrów danych, różnorodność kategorii zasobów i ich współzależności, a także rozmaitość i zmienność oraz nieprzewidzialność obciążeń, jak też zakres celów różnych użytkowników korzystających z rozwiązań chmurowych. Dodatkowe wymagania dotyczą automatyzacji procedur zarządzania oraz szerokiego wykorzystania narzędzi zarządzających. Problemem rozpatrywanym w tym rozdziale będą algorytmy przydziału zasobów fizycznych chmury do maszyn wirtualnych czy kontenerów realizujących aplikacje złożone z usług lub mikrousług. Przyjmijmy następujące oznaczenia:

$U = \{u_1, u_2, \ldots, u_m\}$ – zbiór usług do wykonania na chmurze obliczeniowej;

$Z = \{z_1, z_2, \ldots, z_n\}$ – zbiór zasobów będących w dyspozycji chmury obliczeniowej;

δ – wektor bieżącego zaangażowania zasobów chmury, np. $\delta = \{\delta_1, \delta_2, \ldots, \delta_n, \delta_i \in \langle 0, 1\rangle\}$;

ψ – macierz wykorzystania zasobów niezbędnych dla wykonania usług ze zbioru U;

$\psi = [\psi_{ij}]$, gdzie $i = 1, 2, \cdots, .m; j = 1, 2, \cdots, n$; zaś:

$$\psi_{ij} = \begin{cases} 1 - \text{gdy usługa } u_i \text{ jest przypisana do zasobu } z_j \\ 0 - \text{w przeciwnym wypadku} \end{cases} \tag{13.1}$$

Macierz $\gamma = [\gamma_{ij}]$ określa stopień zaangażowania zasobów przy wykonaniu usług ze zbioru U, $\gamma_{ij} \in \langle 0, 1\rangle$.

T to macierz czasów wykonania usług ze zbioru U przy wykorzystaniu zasobów ψ ze zbioru Z:

$$T = [t_{ij}] \tag{13.2}$$

gdzie t_{ij} jest czasem wykonania usługi u_i na dostępnym zasobie z_j, tzn. gdy $\gamma_{ij} \leq 1 - \delta_j$. W przeciwnym przypadku czas wykonania usługi t_{ij} zwiększa się zgodnie ze wzorem:

$$t'_{ij} = (\delta_j + \gamma_{ij}) t_{ij} \tag{13.3}$$

Czas realizacji usług ze zbioru U na zbiorze Z określono jako $\tau(U, Z)$, dla sekwencyjnej realizacji usług określa się go w sposób następujący:

$$\tau(U, Z) = \sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij} \cdot \psi_{ij} \tag{13.4}$$

W przypadku równoległego wykonania usług czas ten może być znacznie krótszy, co zależy od liczby usług wykonywanych równolegle. Natomiast wykorzystanie zasobów $\delta(U)$ zmienia się w sposób następujący:

$$\delta'_j = \delta_j + \sum_{i=1}^{m} \psi_{ij} \cdot \gamma_{ij} \tag{13.5}$$

Przyjmując wymaganie zrównoważenia, można zdefiniować następujące zadanie optymalizacyjne.

Szukamy takiego ψ, dla którego:

$$\begin{aligned} \tau(U, Z) &= min \\ \delta(U) &\leq \delta_{max} \end{aligned} \tag{13.6}$$

przy założeniu, że każda usługa jest lokowana na różne zasoby $\sum_{i=1}^{m} x_{ij} = 1$, które umożliwiają jej w miarę szybkie wykonanie, tzn. $\gamma_{ij} \leq 1 - \delta_j$.

Czas realizacji pojedynczej usługi może być określony poprzez trzy składowe: czas ulokowania odpowiedniego zadania na zasobach wirtualnych chmury obliczeniowej, czas transmisji danych do tej jednostki wirtualnej oraz czas wykonania tego zadania na wskazanych zasobach fizycznych węzła chmury. W przypadku minimalizacji zużytej energii tego typu czasy muszą być pomnożone przez odpowiednie współczynniki zużycia energii. W celu zapewnienia wymaganej niezawodności działania w kryteriach optymalizacji należy uwzględnić zarówno intensywność błędów pojawiających się w jednostkach wirtualnych, jak też intensywność uszkodzeń węzłów. Należy także wprowadzić procedury zwielokrotnienia wykonania tych samych usług na różnych węzłach oraz procedury migracji jednostek wirtualnych na inne zasoby fizyczne chmury w przypadku uszkodzenia węzła lub nadmiernego jego obciążenia. W tym ostatnim przypadku należy też odpowiednio zmodyfikować wzrost czasów wykonania usług (wzór (13.3)) o czas uruchomienia nowego węzła, jeśli nie jest aktywny, oraz czas transmisji jednostek wirtualnych pomiędzy dwoma węzłami: rozpatrywanym i docelowym, a także czas wyboru i posadowienia wskazanej usługi na węźle docelowym. Przedstawione kryterium optymalizacji wymaga więc pozyskania dodatkowych danych, dotyczących z jednej strony wykonania usług, z drugiej zaś potencjalnych zasobów i możliwości węzłów. Dane te mogą być dostarczone dzięki specjalnym eksperymentom wykonywanym na badanej chmurze obliczeniowej albo estymowane na podstawie dostępnych symulatorów modelujących zachowanie się wskazanej chmury obliczeniowej. Mogą być też uzyskane poprzez budowanie i analizowanie odpowiednich modeli teoretycznych opisujących zachowanie badanej chmury. W dalszych rozważaniach przyjęto, że istnieją realne możliwości dostarczenia tego typu danych [4].

13.4. Możliwości wykorzystania algorytmu mrówkowego

Kolonia mrówek w celu zdobycia pożywienia wykorzystuje proste instynkty, które pozwalają im je znaleźć, nawet jeśli znajduje się bardzo daleko od ich

zasiedlenia. Mrowisko wypuszcza poruszających się mniej więcej losowo zwiadowców, którzy mają pewne poczucie kierunków i gdy natrafią na coś interesującego, wracają do kolonii, znacząc swoją trasę feromonami. Pozostawione na trasie feromony przyciągają więcej mrówek, które zaczynają podążać mniej więcej wyznaczoną przez feromony trasą i transportują pożywienie do miejsca pobytu, również zaznaczając swoją drogę powrotną feromonami. Możliwość odchyłki od wyznaczonej trasy czy losowego wyboru jednej z alternatywnych tras sprawia, że ścieżki są eksploatowane w różny sposób. Większa liczba feromonów przyciąga większą liczbę mrówek, tym samym trasy krótsze są częściej wykorzystywane, a te najdłuższe są znacznie częściej ignorowane. Dodatkowo feromony z czasem parują, co coraz bardziej zwiększa różnicę w ich natężeniu pomiędzy różnymi trasami, sprawiając, że bardziej preferowana trasa staje się jeszcze bardziej popularna [5]. Zatem dzięki feromonom mrówki same się organizują i wybierają zalecane trasy, które wskazują sposób zrealizowania założonego celu, tzn. zdobycie odpowiedniej ilości pożywienia jak najmniejszym kosztem. Istotne przy tym są siła działania posiadanego feromonu, sposób jego pozostawiania na trasie oraz stosowane zasady kierowania się nim przez mrówki. Ważne są także takie parametry, jak liczba obserwowanych kroków działania grupy mrówek o zadanej liczności czy liczba iteracji obserwacji zachowania się nowych grup mrówek wychodzących kolejno z kolonii, a także zapamiętywanie i ocena znalezionych rozwiązań dla zrealizowanych iteracji [7]. W zależności od przyjętych powyższych zasad i parametrów mamy różne algorytmy mrówkowe, które rozwiązują różne zadania optymalizacyjne [11]. Co więcej, algorytmy mrówkowe można łączyć z algorytmami ewolucyjnymi, gdzie pewne parametry mogą ewaluować (na zasadzie krzyżowania i mutacji) w zależności od jakości znajdowanych rozwiązań, co prowadzi do jeszcze bardziej interesujących rozwiązań [6]. Poniżej wykorzystamy odpowiedni algorytm mrówkowy do rozwiązania zdefiniowanego powyżej optymalizacyjnego zadania alokacji zadań. Istotną cechą proponowanego algorytmu jest możliwość jego wykorzystania w sytuacjach, w których poszukiwane rozwiązanie uwzględnia różne kategorie aplikacji zgłaszanych do wykonania oraz różne sposoby wykonania aplikacji (sekwencyjnie, równolegle bądź sekwencyjno-równolegle).

Problem przydziału zasobów chmurowych może być modelowany w różnorodny sposób w zależności od żądań użytkowników. Może to być wykonanie tylko jednej aplikacji złożonej ze zbioru niezależnych usług. W przypadku scenariuszy usług zależnych należy rozróżnić dwa przypadki aplikacji: aplikacja zawiera różne scenariusze złożone z alternatywnych sekwencji usług realizujących to samo zadanie bądź aplikacja składa się z zależnych usług, które wszystkie muszą być wykonane dla realizowania tego zadania. We wszystkich przypadkach chodzi o odpowiednie przypisanie do tych usług właściwych zasobów znajdujących się na różnych węzłach chmury, takie, które spełnia odpowiednie zadanie optymalizacyjne. Tablica 13.2 podaje podstawowe własności rozpatrywanych aplikacji.

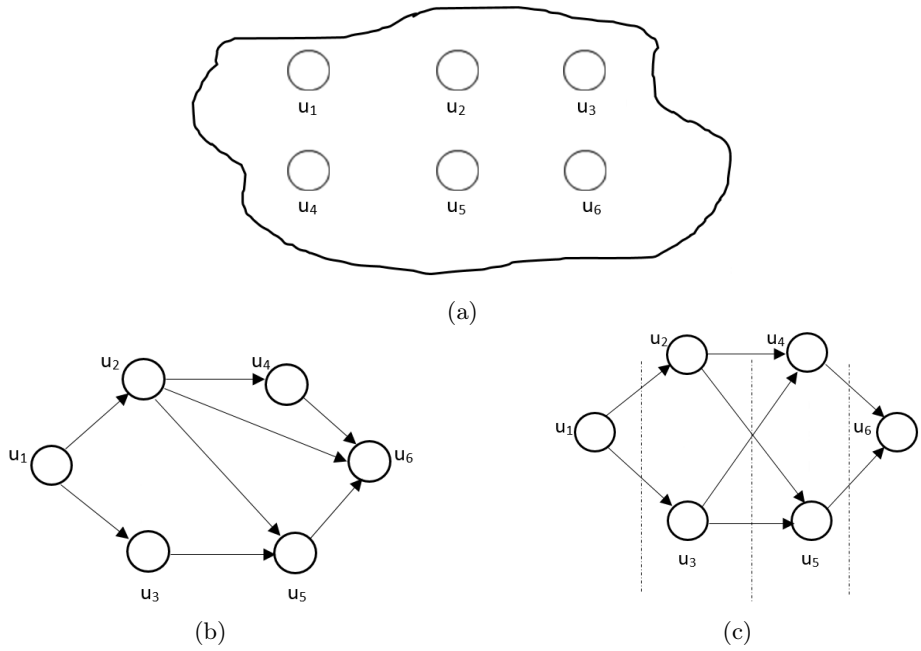
Żądania użytkowników można przedstawić jako graf $G = (U, A)$, gdzie U jest zbiorem węzłów reprezentujących usługi, A jest zaś zbiorem łuków reprezentujących zależności (np. czasowe) między usługami. Rys. 13.1 ilustruje modele gra-

Tablica 13.2: Reprezentatywne przypadki kategorii usług oraz zasady ich alokacji na infrastrukturze chmurowej

Kategoria usług	Opis żądania	Uwagi
Wykonanie zbioru niezależnych usług	Rozpatrzenie wszystkich możliwych alokacji usług ze zbioru U do zasobów ze zbioru Z (podejście równoległe) oraz wybór przyporządkowania najlepiej spełniającego kryteria optymalizacji	Przy sekwencyjnej metodzie alokacji usług kolejność ich rozpatrzenia może wpłynąć na optymalne rozwiązanie
Wykonanie odpowiedniej sekwencji usług ze zbioru sekwencji alternatywnych	Alternatywne sekwencje usług reprezentuje graf $G(U, A)$, gdzie U jest zbiorem usług do wykonania zaś A określa relację następstwa usług	Problem optymalnego wyboru infrastruktury sprowadza się do wskazania ścieżki w grafie spełniającej najbardziej kryteria optymalizacyjne
Wykonanie aplikacji będącej scenariuszem zależnych usług	Aplikację taką reprezentuje graf $G(U, A)$ jak wyżej, przy czym wykonanie aplikacji oznacza wykonanie wszystkich usług w tym grafie	Metoda alokacji wymaga uwzględnienia podejść odpowiednich dla dwóch powyższych kategorii żądań

fowe dla trzech wskazanych kategorii aplikacji. W pierwszym przypadku A jest zbiorem pustym. Usługi do zrealizowania są niezależne i mogą być rozlokowane pojedynczo na zasobach chmury oraz wykonane równolegle. W przypadku drugim aplikacja nie wymaga wykonania wszystkich usług, realizujemy tylko jedną z alternatywnych ścieżek, która w danej sytuacji jest najlepszym rozwiązaniem. W przypadku trzecim wymagane jest wykonanie wszystkich usług przy zachowaniu ich odpowiednich uwarunkowań. Taki graf może być zastosowany bezpośrednio do algorytmu mrówkowego tylko w przypadku drugiej kategorii aplikacji chmurowych.

Dlatego w proponowanym algorytmie mrówkowym nie bazujemy na grafie $G(U, A)$, a operujemy na grafie $G(D, L)$, definiowanym w sposób następujący. W zbiorze D wyróżniamy dwa specjalne wierzchołki: wejściowy (start) i wyjściowy (stop), w których inicjujemy i kończymy obliczenia. Pozostałe wierzchołki określają wszystkie możliwe decyzje przyporządkowania usług ze zbioru U należących do rozpatrywanej aplikacji do istniejących zasobów ze zbioru Z przy-



Rysunek 13.1: Przykład żądania wykonania usług w chmurze obliczeniowej: zbioru niezależnych usług (a), jednej z możliwych sekwencji usług (b), wszystkich usług w porządku określonym relacją następstw (c)

należących do chmury. Na przykład $d_1(u_1, x)$ oznacza podjęcie pierwszej decyzji, w której usługa u_1 będzie przypisana do jednego z możliwych zasobów (x) (rys. 13.2). Innymi słowy, są to stany decyzyjne, w których rozpatruje się możliwości podjęcia kolejnych decyzji, aż do uwzględnienia wszystkich usług. Kolejność analizy usług może być dowolna, zaś długość ścieżki decyzyjnej wynosi m. Zbiór L reprezentuje łuki ukazujące możliwości przejścia z danego stanu do następnego stanu, inaczej podjęcie kolejnej decyzji dotyczącej przyporządkowania następnej usługi u do wybranego zasobu z. Optymalne podejście wymaga rozpatrzenia wszystkich możliwych wariantów przyporządkowania oraz wszystkich możliwych sekwencji stanów. Liczbę takich możliwych przyporządkowań (P) określają odpowiednie wzory:

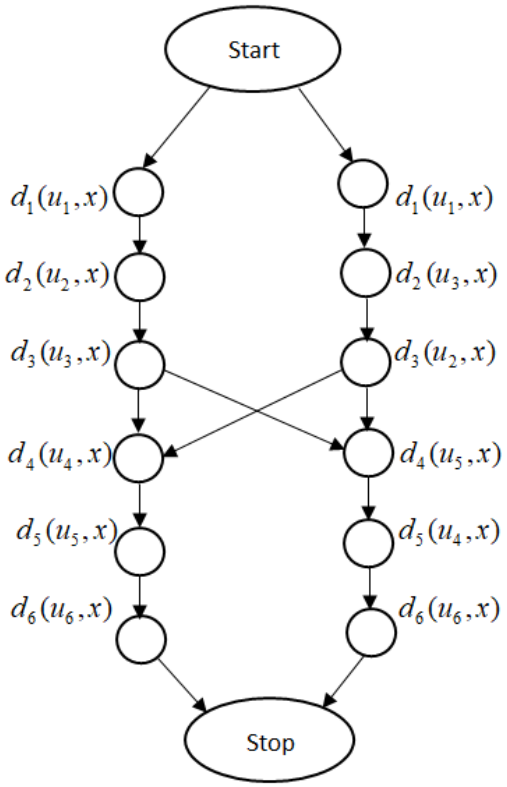
– w przypadku wszystkich możliwości przyporządkowania:

$$P = m^n \tag{13.7}$$

– w przypadku rozlokowania każdej usługi na innym zasobie:

$$P = \binom{m}{n} \cdot n! \tag{13.8}$$

Z uwagi na dużą liczbę wierzchołków podajemy jedynie uproszczony graf decyzyjny dla grafu z rys. 13.1c, gdzie $m = 6$, $n = 3$. Analiza wszystkich



Rysunek 13.2: Uproszczony graf decyzyjny dla aplikacji z rys. 13.1c.

wariantów przyporządkowań dla tak małego grafu G wynosi 216, zaś liczba możliwych przyporządkowań różnych usług na różnych zasobach wyniesie już tylko 20. Z kolei liczba wszystkich ścieżek decyzyjnych wyniesie $(m!)$ 720. Dla wielu usług i wielu zasobów wygenerowanie grafu D zajmuje dużo czasu, dlatego w algorytmie mrówkowym nie jest to konieczne. Kolejne mrówki odkrywają tylko pewne istotne fragmenty grafu decyzyjnego. Odpowiednia liczba mrówek w każdym cyklu poszukiwania oraz odpowiednia liczba cykli zapewniają wykrycie najlepszej sekwencji decyzji oraz wyznaczenie macierzy ψ.

13.5. Proponowana postać algorytmu

Dane są graf decyzji D oraz możliwe warianty przyporządkowań zasobów ze zbioru Z. Każda ze sztucznych mrówek musi przejść po grafie D od wierzchołka początkowego do wierzchołka końcowego tak, by odwiedzić dokładnie jeden wierzchołek grafu dla każdej usługi ze zbioru U oraz zachować ograni-

czenia kolejnościowe wynikające z grafu G. W ten sposób problem przydziału zostaje przekształcony w problem przeszukiwania grafu w celu znalezienia optymalnej ścieżki w grafie D z wierzchołka początkowego do wierzchołka końcowego. W celu zabezpieczenia przed powstawaniem rozwiązań niedopuszczalnych w każdym kroku mrówka musi posiadać informację o dostępnych dla niej krawędziach grafu. Wędrujące początkowo w sposób losowy mrówki w przypadku odnalezienia pokarmu powracają do mrowiska, znacząc swoją drogę feromonami. Kolejne osobniki nie wędrują już w sposób losowy, lecz często podążają śladem pozostawionym przez poprzedników. Podczas drogi każdej mrówki od wierzchołka startu do wierzchołka stopu grafu następuje selekcja następnego wierzchołka spośród decyzji dopuszczalnych (odpowiadających usługom dotąd nieodwiedzonym i jednocześnie niełamiącym ograniczeń kolejnościowych). W tym celu losowana jest liczba rzeczywista q z przedziału $\langle 0, 1\rangle$ i porównywana do liczby q_0 (parametr algorytmu z tego samego przedziału). Jeżeli $q \leq q_0$, to spośród dopuszczalnych wybierany jest ten łuk grafu, którego wierzchołek końcowy ma największą wartość spośród $a_j = \tau_j \cdot (f_j)^\beta$, gdzie f_j to wartość feromonu dla i-tego wierzchołka, zaś τ_j to dla niego wartość heurystyczna – pomocnicza, β to parametr algorytmu sterujący wpływem wartości heurystycznej na wybór wierzchołka. Wartość heurystyczną wierzchołka w grafie D definiujemy jako liczbę jego następników odpowiadającej mu czynności, powiększoną o 1; tym samym preferuje się czynności, które mają większą liczbę następników.

Algorytm mrówkowy ogranicza rozpatrzenie wszystkich możliwości przyporządkowania, ale stwarza szansę wyboru rozwiązania znajdującego się w pewnym otoczeniu rozwiązania optymalnego. Każda mrówka która znalazła dobre rozwiązanie, dzieli się tą wiedzą z innymi, dzięki temu istnieje możliwość stopniowego udoskonalania rozwiązania. Algorytm mrówkowy zakłada zaangażowanie wielu mrówek w jednym cyklu jego działania. Zarówno liczbę mrówek (la) w cyklu, jak i liczbę cyklu (lc) ustala się heurystycznie. W naszym przypadku przyjmiemy, że $la \geq m \cdot n$, $lc \geq sk$. Poza tym przyjmiemy określone zasady opisujące działanie mrówek w k-tym kroku postępowania.

W systemie mrówkowym każda mrówka na określonym etapie rozwiązania problemu podejmuje decyzję dotyczącą wyboru odpowiednich zasobów z dla zrealizowania usługi u. Decyzja przyporządkowania może być podjęta w sposób losowy, przez wylosowanie usługi do rozpatrzenia oraz wybranie dla niej zasobu również w sposób losowy. Taka metoda jest nazywana metodą eksploracji. Można też dokładniej przeanalizować sytuację w tworzonym grafie decyzyjnym i wybrać zarówno usługę, jak i odpowiedni zasób w sposób bardziej przemyślany. Taki sposób postępowania nazywamy eksploatacją.

W tablicy 13.3 podano oznaczenia wykorzystane w algorytmie mrówkowym. Poza tym istotne są wzory związane z zachowaniem się mrówki odnośnie do uaktualnienia śladu feromonu, w zależności od tego, w jakiej sytuacji się ona znajduje, oraz odnośnie do jakości znalezionego rozwiązania. Istotna jest również ocena jakości podejmowanych decyzji uwzględnianych przez każdą mrówkę. Z reguły przejście od decyzji x do decyzji y dla mrówki m określa się według wzoru:

$$y = \begin{cases} \arg\max_{y \in S_m(x)} F(x,y) \cdot [Q(x,y)]^{\beta} \text{ dla } q \leq q_o \\ \arg\max_{y \in S_m(x)} P_m(x,y) \text{ dla } q > q_o \end{cases} \tag{13.9}$$

gdzie:

$$y = \begin{cases} \frac{[F(x,y)] \cdot [Q(x,y)]^{\beta}}{\sum_{y \in S_m(x)} [F(x,y)] \cdot [Q(x,y)]^{\beta}} \text{ dla } y \in D_m(x) \\ 0 \text{ dla } y \notin D_m(x) \end{cases} \tag{13.10}$$

Z kolei uaktualnienie śladu feromonu określa się według następującego wzoru:

$$F(x,y) = (1-\rho)F(x,y) + \rho \sum_{k=1}^{K} \Delta F_k[Q^*(x,y)] \tag{13.11}$$

gdzie: $Q^*(x,y)$ – jakość najlepszego rozwiązania dla pełnej ścieżki decyzyjnej (x,y), uwzględniającej wszystkie usługi do przyporządkowania, K – liczba mrówek wskazująca to rozwiązanie.

Proponowany algorytm przyjmuje postać przedstawioną poniżej, a wszystkie oznaczenia podane są w tabl. 13.3.

1. Dane wejściowe:
   graf G, zbiory U, Z oraz macierze T, γ, a także wektor δ, R – zbiór rozwiązań, $R = \Phi$
2. Stałe algorytmu: la, lc, q_o, β, Q
3. Dla grafu G określ podzbiory usług, które mogą być wykonywane jednocześnie (równolegle), $U = U_1^*, U_2^*, \ldots, U_{m^*}^*$
4. **for** $l = 1$ **to** m^* **do**
5. **if** $|U_l^*| > n$ podziel losowo U_i^* tak, żeby $|U_{il}^*| \leq n$, każdej usłudze u z U_{il}^* przypisano inny zasób ze zbioru Z
6. **end for**
7. Algorytm mrówkowy
8. **for** $i = 1$ **to** lc **do**
9. Wybierz zestaw mrówek A_i, $a \in A_i$, $la = |A_i|$
10. **for** $j = 1$ **to** la **do**
11. Wybierz usługę $u \in U$ zgodnie z relacją kolejności (p. 5) oraz wzorami (13.9) i (13.10)
12. Określ dla niej zasoby $z \in Z$ zgodnie z kryterium optymalizacji (13.6)
13. Aktualizuj feromon na drodze decyzyjnej zgodnie ze wzorem (13.11)
14. **end for**
15. Zapamiętaj przyporządkowanie $\psi(u,z)$ w macierzy ψ_j
16. Oceń rozwiązanie ψ_j zgodnie ze wzorem (13.6)
17. Dodaj ψ_j do zbioru rozwiązań R
18. Aktualizuj feromon dla otrzymanego rozwiązania zgodnie ze wzorem (13.11)
19. **end for**
20. Wybierz najlepsze rozwiązanie ze zbioru R zgodnie z kryteriami (13.6)
21. **end**

Tablica 13.3: Podstawowe oznaczenia

Pojęcie	Symbol	Opis
Usługi	$u \in U$	Usługa u do wykonania spośród wszystkich usług zgłoszonych (zbiór U) przez klientów
Zasoby chmury	$z \in Z$	Zasób z, na którym może być wykonana jedna z usług wybrana z dostępnego zbioru zasobów chmury Z
Odwzorowanie	$P(U, Z)$	Przyporządkowanie usług ze zbioru U do zasobów chmury ze zbioru Z
Rodzaje zasobów k-węzła	$z_k = \{z_{1k}, z_{2k}, z_{3k}, z_{4k}\}$	Zasób węzła k określają: moc procesora (z_{1k}), pojemność pamięci (z_{2k}), pojemność dysku (z_{3k}) oraz przepustowość sieci (z_{4k})
Czas wykonania usług	$T = [t_{uz}] u \in U, z \in Z$	Macierz czasów wykonania usług $u \in U$ na wszystkich możliwych dostępnych zasobach chmury
Konkretne przyporządkowanie usług do zasobów	$\psi = [\psi_{u,z}]$	$\psi_{u,z} = \begin{cases} 1, \text{ jeśli usługa } u \text{ jest przypisana do zasobu } z \\ 0 \text{ przeciwnie} \end{cases}$
Mrówki	lm	Liczba mrówek zdolnych do podejmowania decyzji związanych z optymalizacją kryteriów optymalizacji
Iteracje algorytmu	la	Liczba cykli algorytmu dla różnych zbiorów mrówek
Decyzje	s_m $d_m(s)$ $D_m(s)$ $S_m(s)$	Stan mrówki m oznacza możliwość podjęcia przez mrówkę będącą w stanie s następnej decyzji $d_m(s)$ Zbiór możliwych decyzji mrówki m w stanie s Zbiór wszystkich stanów, które może rozpatrzyć mrówka m będąca w stanie s
Parametry algorytmu	q_o $f[d_m(s)]$ ρ	Wartość graniczna parametru $q \in \langle 0, 1 \rangle$ określająca wybór reguły postępowania mrówki Ilość feromonu (stopień użyteczności) związana z decyzją $d_m(s)$ Współczynnik parowania feromonu $\rho \in \langle 0, 1 \rangle$
Ocena algorytmu	$F(x, y)$ $Q(x, y)$ β $\Delta F_m(Q)$ $P_m(x, y)$	Ilość feromonu (jakość decyzji) na drodze decyzyjnej od stanu x do stanu y Jakość podjętych decyzji na drodze decyzyjnej (x, y) Ważność pomiędzy śladem feromonu a powrotnym dystansem Przyrost feromonu wytwarzanego przez mrówkę m związany ze wzrostem jakości rozwiązania Prawdopodobieństwo przejścia mrówki m od x do y

13.6. Przykład działania algorytmu

Rozpatrzmy aplikacje przedstawione na rys. 13.1. Zbiór usług oznaczmy przez $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, zaś zbiór zasobów przez $Z = \{z_1, z_2, z_3\}$. Wektor bieżącego zaangażowania zasobów chmury $\delta = [0{,}6; 0{,}4; 0{,}7]$. Macierz czasów wykonania oraz stopień zaangażowania zasobów przy wykonaniu usług ze zbioru U przy wykorzystaniu zasobów ze zbioru Z określają następujące macierze:

$$Z = \begin{bmatrix} 3 & 3 & 4 \\ 6 & 7 & 5 \\ 4 & 2 & 2 \\ 3 & 3 & 3 \\ 6 & 5 & 5 \\ 4 & 4 & 3 \end{bmatrix} \qquad \gamma = \begin{bmatrix} 0,2 & 0,2 & 0,3 \\ 0,4 & 0,5 & 0,4 \\ 0,4 & 0,2 & 0,2 \\ 0,4 & 0,3 & 0,3 \\ 0,4 & 0,3 & 0,2 \\ 0,3 & 0,3 & 0,2 \end{bmatrix}$$

Wykorzystując algorytm mrówkowy, otrzymujemy uporządkowania ψ_x dla aplikacji z rys. 13.1, gdzie $x = a$, $x = b$, $x = c$. Będą to następujące przyporządkowania:

$$\psi_a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \psi_b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \psi_c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

W tablicy 13.4 podano czasy wykonania dla tych wyznaczonych przyporządkowań. W algorytmie uwzględniono równoległość wykonania usług zgodnie z rys. 13.1c.

Tablica 13.4: Trzy wykonania aplikacji przedstawione na rys. 13.1 dla znalezionych odwzorowań ψ_x

Odwzorowanie	ψ_a	ψ_b	ψ_c
$T(\psi, x)$	8	12	15

Wektor zaangażowania zasobów na czas realizacji tych aplikacji wzrasta do następujących wartości: $\delta_a = [0,9; 0,7; 0,9]$, $\delta_b = [0,8; 0,8; 0,9]$, $\delta_c = [0,9; 0,8; 0,9]$. W międzyczasie może on jednak zmieniać się z uwagi na zakończenie wykonywania pewnych aplikacji oraz podjęcie się wykonania innych aplikacji.

13.7. Uwagi końcowe

W rozdziale przedstawiono uogólniony algorytm mrówkowy umożliwiający efektywny przydział optymalizowanych zasobów do wykonania usług zgłaszanych

przez użytkowników. Rozpatrzono przy tym trzy kategorie aplikacji: złożonej z całkowicie niezależnych usług wykonywanych równolegle na zasobach chmury, złożonej z alternatywnych scenariuszy usług, które są wykonywane w sposób sekwencyjny, oraz złożonej z usług uwarunkowanych relacją poprzedzania i wykonywanych w sposób sekwencyjno-równoległy. Istniejące tego typu algorytmy mrówkowe dotyczyły w zasadzie drugiej kategorii aplikacji.

Proponowany algorytm heurystyczny, jak każde tego typu rozwiązanie, wymaga przyjęcia pewnych arbitralnych parametrów i strojenia ich w trakcie jego testowania. Eksperymenty wykazały, że wraz ze wzrostem liczby iteracji oraz liczby mrówek wykorzystywanych w każdej iteracji jakość rozwiązania, rozumiana w sensie przyjętego kryterium optymalizacji, poprawia się, ale tylko do osiągnięcia pewnych wartości granicznych. Inne parametry algorytmu dla ustalonych wartości powyższych wielkości, takie jak współczynnik parowania czy zwiększania ilości feromonu, prawdopodobieństwo wyboru następnej decyzji, też wpływają na jakość rozwiązania, ale nie są silnie powiązane z kategorią aplikacji. Ten fakt potwierdza cechę uniwersalności proponowanego algorytmu.

Proponowany algorytm był testowany na teoretycznych przykładach w celu sprawdzenia poprawności jego działania. Wstępne badania wskazują na wysoką skuteczność otrzymania optymalnego rozwiązania (87%) oraz wysoką efektywność działania algorytmu, charakteryzującą się liniową zależnością czasową dla wzrastającej liczby przetwarzanych usług. Zakłada się też jego kompleksowe przetestowanie w środowisku rzeczywistym, jakim jest lokalna chmura obliczeniowa TASKcloud, funkcjonująca już od kilku lat w CI TASK.

Bibliografia

[1] *Essential Characteristics of Cloud Computing.* `https://comparecloud.in`, 2011. [dostęp: 14 grudnia 2022].

[2] Almutairi A., Sarfraz M.I., Ghafoor A. *Risk-aware management of virtual resources in access controlled service-oriented cloud datacenters. IEEE Transactions on Cloud Computing*, 6(1):168–181, 2015.

[3] Balalaie A., Heydarnoori A., Jamshidi P. *Microservices architecture enables devops: Migration to a cloud-native architecture. Ieee Software*, 33(3):42–52, 2016.

[4] Bhavani B., Guruprasad H. *Resource provisioning techniques in cloud computing environment: a survey. International Journal of Research in Computer and Communication Technology*, 3(3):395–401, 2014.

[5] Farahnakian F., et al. *Using ant colony system to consolidate VMs for green cloud computing. IEEE Transactions on Services Computing*, 8(2):187–198, 2014.

[6] Gao Z. *The allocation of cloud computing resources based on the improved Ant Colony Algorithm.* In *2014 Sixth International Conference on Intelli-*

gent Human-Machine Systems and Cybernetics, vol. 2, pp. 334–337. IEEE, 2014.

[7] Gill S.S., et al. *BULLET: particle swarm optimization based scheduling technique for provisioned cloud resources. Journal of Network and Systems Management*, 26(2):361–400, 2018.

[8] Jennings B., Stadler R. *Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management*, 23(3):567–619, 2015.

[9] Li W., Kanso A. *Comparing containers versus virtual machines for achieving high availability.* In *2015 IEEE International Conference on Cloud Engineering*, pp. 353–358. IEEE, 2015.

[10] Lin K., Herrmann P. *Social quality theory: A new perspective on social development.* Berghahn Books, 2015.

[11] Lin M., et al. *Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. IEEE access*, 7:83088–83100, 2019.

[12] Milenkovic M. *Internet of Things: Concepts and System Design.* Springer, 2020.

[13] Odun-Ayo I., Udemezue B., Kilanko A. *Cloud service level agreements and resource management. Adv. Sci. Technol. Eng. Syst.*, 4(2):228–236, 2019.

[14] Pahl C., et al. *Cloud container technologies: a state-of-the-art review. IEEE Transactions on Cloud Computing*, 7(3):677–692, 2017.

[15] Suri J., et al. *Aprowizowanie hosta lub klastra Hyper-V z komputerów bez systemu operacyjnego.* `https://docs.microsoft.com/pl-pl/system-center/vmm/hyper-v-bare-metal`, 2022. [dostęp: 14 grudnia 2022].

[16] Zach M. *What is the ITIL certification and why do you need it?* `https://www.itpro.co.uk/strategy/29244/what-is-the-itil-certification-and-why-do-you-need-it`, 2021. [dostęp: 14 grudnia 2022].

[17] Zhang Y., Chulkov N. *Information and communication technology (ICT) governance in the United Nations system organizations. Joint Inspection Unit*, 9:1–39, 2011.

Rozdział 14

Badanie wpływu przydziału rdzeni procesora na wydajność w środowisku skonteneryzowanym oparte na wybranym serwerze warstwy pośredniej w IoT – obserwacje i rekomendacje

Robert Kałaska[1]

[1] Katedra Architektury Systemów Komputerowych
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska
robkalas@pg.edu.pl

Abstrakt

Internet Rzeczy cieszy się coraz większym zainteresowaniem. Zagadnienie to jest szeroko omawiane zarówno w środowisku naukowym, jak i w przemyśle. Ze względu na jego wielowymiarowość jest wiele aspektów, które wymagają zbadania i obserwacji. Jednym z nich jest efektywne wdrożenie i uruchomienie aplikacji w kontekście wykorzystania zasobów sprzętowych. Innym, równie istotnym, zagadnieniem jest konteneryzacja platform IoT. Jednym ze wspólnych

obszarów dla obu tych zagadnień jest analiza wydajności w środowisku skonteneryzowanym dla różnych konfiguracji przydziału rdzeni procesora. To zagadnienie zostało zbadane w niniejszym rozdziale na podstawie z otwartych platform dla IoT – DeviceHive. Zmierzone zostały skalowalność i stabilność rozwiązania w różnych konfiguracjach węzłów i przypisanych im rdzeni. Najciekawsze wnioski zostały wyciągnięte z obserwacji tych samych konfiguracji w kontekście liczby węzłów, a różniących się sposobem przypisania rdzeni. Przedstawione wyniki mogą być przydatne w zakresie planowania architektury rozwiązań skonteneryzowanych platform IoT.

Słowa kluczowe: wdrożenie platform IoT, wydajność w skonteneryzowanym środowisku, konteneryzacja w HPC

14.1. Wprowadzenie

W ostatnich latach prace wielu badaczy skupiały się na Internecie Rzeczy. Najnowsze badania pokazują, że około 50 miliardów urządzeń jest podłączonych do Internetu Rzeczy (stan na koniec 2020 r.) [24], a przyszłe prognozy wskazują na 500 miliardów urządzeń podłączonych do Sieci [31]. Wiele artykułów naukowych z dziedziny Internetu Rzeczy jest poświęconych zagadnieniom *cloud computing* [27], przetwarzania *on edge* [18], [25], [26] i konteneryzacji platform IoT [1], [3]. Wnioski płynące z tych artykułów wskazują, że konteneryzacja jest preferowaną techniką uzyskiwania izolowanego środowiska. Ma znacznie mniejszy narzut niż wirtualizacja. Jednym z najczęściej używanych narzędzi do konteneryzacji jest Docker [7]. Interesującym zagadnieniem wydaje się zbadanie, czy wydajność rozwiązania z taką samą liczbą rdzeni przypisaną w różny sposób do kontenerów będzie się różnić, a jeśli tak to w jakim stopniu. W badaniu opisanym w niniejszym rozdziale przeanalizowano różne konfiguracje tej samej platformy Internetu Rzeczy, różniące się liczbą węzłów (kontenerów) oraz sposobem przydziału rdzeni procesora. Głównym celem było określenie, czy konfiguracja z przypisanym ograniczonym i unikalnym zestawem rdzeni dla każdego kontenera uzyska lepsze wyniki niż konfiguracja z przypisaniem wszystkich rdzeni do wszystkich kontenerów, czyli współdzieleniem ich. Co istotne – suma rdzeni dostępna dla całej platformy w każdej konfiguracji była identyczna.

Sekcja 14.2 opisuje dotychczasowe badania w podobnym zakresie. W sekcji 14.3 przedstawione zostały różne techniki uzyskiwania izolowanego środowiska – wirtualizacja i konteneryzacja oraz różnice między nimi. W sekcji 14.4 zawarto informacje dotyczące sposobu kontroli przypisania rdzeni procesora przez jądro systemu i algorytmu, który tym zarządza. Sekcja 14.5 zawiera opis środowiska testowego i przyjętych wskaźników. W sekcji 14.6 opisano i przeanalizowano otrzymane wyniki i ostatecznie w sekcji 14.7 podsumowano rezultaty badań.

14.2. Powiązane prace

Poprawa wydajności poprzez odpowiednie dopasowanie przydziału rdzeni procesora była już opisywana w literaturze naukowej. W [4] autorzy zmierzyli wydajność architektury mikroserwisowej przy użyciu oprogramowania TeaStore, które jest otwartym oprogramowaniem do testowania rozwiązań mikroserwisowych. Przeanalizowano wyniki w kontekście posiadanego sprzętu. Po analizie badacze zaproponowali optymalizację przy wykorzystaniu odpowiedniego przypisania rdzeni do serwisów, która skutkowała poprawą do 22% w zakresie przepustowości, a opóźnienia zostały zredukowane o około 18%. Inne badanie [5] koncentrowało się na odpowiednim przypisaniu rdzeni w przypadku wysyłki pliku do systemu Hadoop – otwartej platformy do efektywnego przetwarzania i przechowywania dużych zestawów danych. Autorzy uruchomili przygotowany przypadek testowy w konfiguracjach z różnym przydziałem rdzeni i przeanalizowali wyniki. Po dogłębnej analizie otrzymanych wyników zaproponowali *framework* do dynamicznego sterowania przypisaniem rdzeni do zadania. Przedstawione rozwiązanie zwiększyło o 42% przepustowość dla przypadku wgrywania pliku do systemu Hadoop. Kolejny artykuł [10] poświęcony został skalowalności serwera webowego (Lightppd). Badacze przetestowali różne obciążenia przy różnych przydziałach rdzeni procesora. Po przeanalizowaniu wyników zaobserwowali poprawę wydajności do 45% w wybranych przypadkach.

Wielu naukowców skupiło się również na badaniu różnic wydajności pomiędzy rozwiązaniami skonteneryzowanymi, wirtualnymi oraz natywnymi. W [28] autorzy zaproponowali porównanie wydajności dwóch technologii – wirtualizacji przy użyciu oprogramowania Xen oraz konteneryzacji przy użyciu technologii LXC. Otrzymane wyniki pokazały, że wydajność jest lepsza w przypadku LXC, ale w teście dotyczącym izolacji zasobów Xen uzyskał wynik 200 razy lepszy niż LXC. Autorzy przedstawili wyniki, w których stwierdzają, że rozwiązania skonteneryzowane mogą być lepiej dopasowane do zastosowania w chmurach PaaS, podczas gdy wirtualizacja może być lepsza dla chmur IaaS. Inna dogłębna analiza różnych technologii konteneryzacji i wirtualizacji z wykorzystaniem Xen w środowisku HPC została przeprowadzona w artykule [29]. Każde z przetestowanych rozwiązań kontenerowych (LXC, OpenVZ i VServer) uzyskało znacząco lepsze wyniki niż wirtualizacja przy użyciu Xen. Otrzymane wyniki dla kontenerów były bardzo zbliżone do wyników dla natywnego oprogramowania. W badaniu [28] autorzy również wykazali, że izolacja zasobów nie działa zbyt sprawnie w środowisku kontenerowym, w efekcie czego odnotowano spadek wydajności do 89,6% w sytuacji, kiedy inny kontener w ramach tego samego zasobu sprzętowego jest mocno obciążony. W innym artykule [22] naukowcy porównali różne technologie konteneryzacji, takie jak LXD, Docker i LXC. Porównania dokonano, przeprowadzając różne testy. Średnia wydajność wynosiła 81,7% dla Dockera, 90,5% dla LXD oraz 86,5% dla LXC.

Są również badania poświęcone porównaniu różnych rozwiązań oprogramowania *middleware* dla IoT. W pracy [12] autorzy porównali stabilność i skalowalność platform SiteWhere i ThingsBoard. Po analizie wyników stwierdzono,

że SiteWhere ma lepszą wydajność dla MQTT, ale gorszą stabilność. ThingsBoard ma za to lepszą wydajność w REST API. Podobne zagadnienie zostało zbadane w [6]. Badacze przeprowadzili porównanie platform Konker, InatelPat, Orion-STH, SiteWhere. Testy zostały wykonane dla 10 000 równoległych użytkowników i różnych rozmiarów pakietów reprezentowanych przez różną liczbę parametrów. Najlepsze wyniki zarejestrowano dla platformy SiteWhere.

Można również znaleźć badania poświęcone konteneryzacji we wdrożeniach rozwiązań z zakresu Internetu Rzeczy. Możliwość wdrożenia skonteneryzowanych mikroserwisów w ramach sieci Internetu Rzeczy została przeanalizowana w [11]. Autorzy zaproponowali optymalizację, na skutek której rozmiar kontenera spadł do kilkudziesięciu MB, a równocześnie osiągnięto czas inicjalizacji całego nanoserwisu poniżej minuty. Inny artykuł [19] dotyczy zaprojektowania mobilnej chmury, która jest zbudowana z kontenerów osadzonych na urządzeniach IoT. Zaproponowana architektura – IoTDoc – została porównana z rozwiązaniem komercyjnym Amazon EC2. Test bazował na komponentach *Swarm Manager* oraz *Node Manager* wdrożonych na obu chmurach. Rozwiązanie komercyjne wypadło lepiej, ale zaproponowana architektura w niektórych przypadkach nastawionych na wykorzystanie zasobów (test zapisu przy użyciu DD) uzyskała porównywalne wyniki. Interesujące rozwiązanie zrealizowane w celu zwiększenia skalowalności poprzez poprawne rozłożenie obciążenia zostało przedstawione w [2]. Autorzy przeanalizowali przypadek, w którym sieć Internetu Rzeczy zbudowana jest z trzech warstw: czujniki, bramki komunikacyjne i chmura obliczeniowa. W przedstawionym rozwiązaniu na każdej bramce zostały uruchomione trzy serwisy – odczyt i zapis z czujników, przetwarzanie danych i wysyłka do chmury. Dzięki skonteneryzowanej architekturze i podziałowi na mikroserwisy badacze uzyskali właściwe rozłożenie obciążenia na całości zaproponowanej chmury.

Żadne ze wspomnianych badań nie analizuje wpływu przydziału rdzeni w skonteneryzowanym środowisku. Takie badania mogą przynieść znaczący wkład w zakresie poprawnego wdrożenia i uruchomienia różnych rozwiązań, a szczególnie rozwiązań IoT w architekturach HPC.

14.3. Konteneryzacja i wirtualizacja

Są różne podejścia do uzyskania izolowanego środowiska – konteneryzacja i wirtualizacja. Różnią się one architektonicznie i wydajnościowo. Czas uruchomienia oraz narzut na zasoby są różne w każdym z nich. W tej sekcji krótko opisano podstawowe założenia każdego z podejść i wskazano różnice między nimi.

14.3.1. Wirtualizacja

W klasycznej wirtualizacji komponent zwany *system hypervisor* pozwala na równoległe uruchomienie dodatkowych systemów operacyjnych na pojedynczej stacji fizycznej. *System hypervisor* może mieć różne implementacje, np. stanowić część systemu operacyjnego, na którym uruchomiony jest *hypervisor* (rozwiąza-

nia takie jak np. KVM), lub być uruchomiony bezpośrednio na maszynie fizycznej (np. ESXi). Każdy z równolegle dziających systemów operacyjnych uzyskuje swoje komponenty do obliczeń, przechowywania danych czy sieci. Dodatkowy system operacyjny ma swoją własną szynę ISA (*Instruction Set Architecture*), czyli podstawowy zestaw instrukcji opisujący sposób komunikacji z procesorem. Translacja instrukcji do szyny hosta może być realizowana w różny sposób (np. wirtualizacja sprzętowa lub translacja binarna) [8].

14.3.2. Konteneryzacja

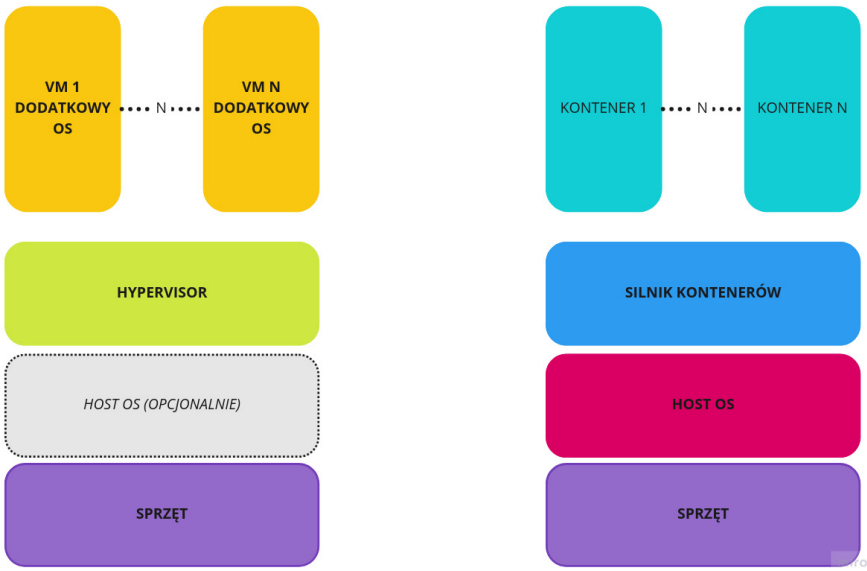
Konteneryzacja to lekki pomysł na wirtualizację. Podstawowe założenie bazuje na uruchomieniu każdego kontenera w ramach jednej instancji jądra systemu. Całość jest oparta na koncepcji *cgroups* i przestrzeni nazw [23], które pozwalają ograniczyć widoczność zasobów pomiędzy procesami, podczas gdy *cgroups* pozwalają tworzyć grupy o określonych priorytetach, ograniczonych zasobach (np. CPU), z uwzględnieniem monitoringu ich działania. Takie rozwiązanie umożliwia wyizolowanie pojedynczego procesu, podczas gdy wszystkie zasoby są kontrolowane przez jądro systemu hosta.

14.3.3. Porównanie obu technologii

Wirtualizacja jest realizowana na warstwie sprzętu, a co za tym idzie, jej czas uruchomienia jest znacząco dłuższy (mierzony w minutach), podczas gdy konteneryzacja do uruchomienia wymaga jedynie rozpoczęcia nowego procesu, zatem jej czas startu jest znacznie krótszy (mierzony w sekundach). Wiele badań zostało poświęconych porównaniu wydajności między poszczególnymi technologiami. W [21] autorzy porównali Dockera i KVM przy użyciu takich narzędzi jak Sysbench, Phoronix i Apache. W zaprezentowanych wynikach Docker uzyskał lepsze rezultaty we wszystkich testach. Inne badanie [9] prezentuje podobne wyniki – Docker uzyskuje również znacząco lepsze wydajności przy znacznie mniejszym narzucie na zasoby. Artykuł [17] przedstawia porównanie pomiędzy KVM, Dockerem i systemem natywnym. Uzyskane wyniki także wskazują na lepszą wydajność Dockera względem KVM. Ponadto wydajność rozwiązania kontenerowego jest zbliżona do tej osiągniętej na natywnym systemie operacyjnym. Nie został zaobserwowany żaden widoczny spadek wydajności. Z drugiej strony korzystając z konteneryzacji, nie można uruchomić równolegle innego systemu operacyjnego niż ten, na którym bazuje jądro systemu operacyjnego hosta, podczas gdy w przypadku wirtualizacji nie ma takich ograniczeń. Na rys. 14.1 zaprezentowano obie architektury.

14.4. Przydział rdzeni procesora

W [14] można znaleźć informację, że przydział rdzeni procesora to zdolność do przypisania jednego lub większej liczby procesów do jednego lub większej liczby



Rysunek 14.1: Architektura wirtualizacji (lewa strona) i konteneryzacji (prawa strona)

procesorów. Przy użyciu takiej funkcjonalności można kontrolować, który proces jest uruchomiony na którym rdzeniu procesora. Użycie takiego rozwiązania pozwala poprawić wydajność dostarczonego oprogramowania, ale musi to być zrobione ostrożnie i umiejętnie, gdyż niewłaściwe ustawienie może prowadzić do znacznego pogorszenia wydajności. Wewnątrz jądra systemu operacyjnego jest zaimplementowany algorytm odpowiedzialny za właściwy przydział rdzeni do procesów. Należy mieć na uwadze, że algorytm domyślnie zaimportowany w systemie Linux stara się w naturalny sposób utrzymywać wybrany proces do pracy na tym samym rdzeniu. Takie działanie pozwala uniknąć narzutu związanego z czasem przełączania i przenoszeniem danych pomiędzy różnymi rdzeniami. Linux i Docker wykorzystują ten sam mechanim przydziału, nazwany CFS (*Completely Fair Scheduler*) [16]. Jego główne zadanie to symulacja idealnego procesora, który jest w stanie podzielić zasoby równo na wszystkie zadania. Bazując na tym założeniu, jego algorytm przelicza, jak długo wybrany proces oczekiwał na CPU, a w kolejnym kroku przydział jest nadawany temu procesowi, który czekał najdłużej. Szczegółowy opis tego rozwiązania można znaleźć w [20]. W przedstawionych pomiarach symulacja różnego przydziału rdzeni procesora została zrealizowana z wykorzystaniem komendy *taskset* [15]. Pozwala ona na przypisanie wybranego CPU do procesu wybranego na podstawie PID lub do nowo wystartowanego procesu.

14.5. Model aplikacji i opis środowiska testowego

W rozwiązaniach Internetu Rzeczy możemy wyróżnić trzy warstwy:

1. czujniki – samodzielne urządzenia elektroniczne;
2. bramki/*middleware* – pośrednicy zdolni do analizy i przetwarzania danych przed podjęciem ich dalszej wysyłki;
3. platformy chmurowe – zbierające dane oraz realizujące złożoną logikę biznesową.

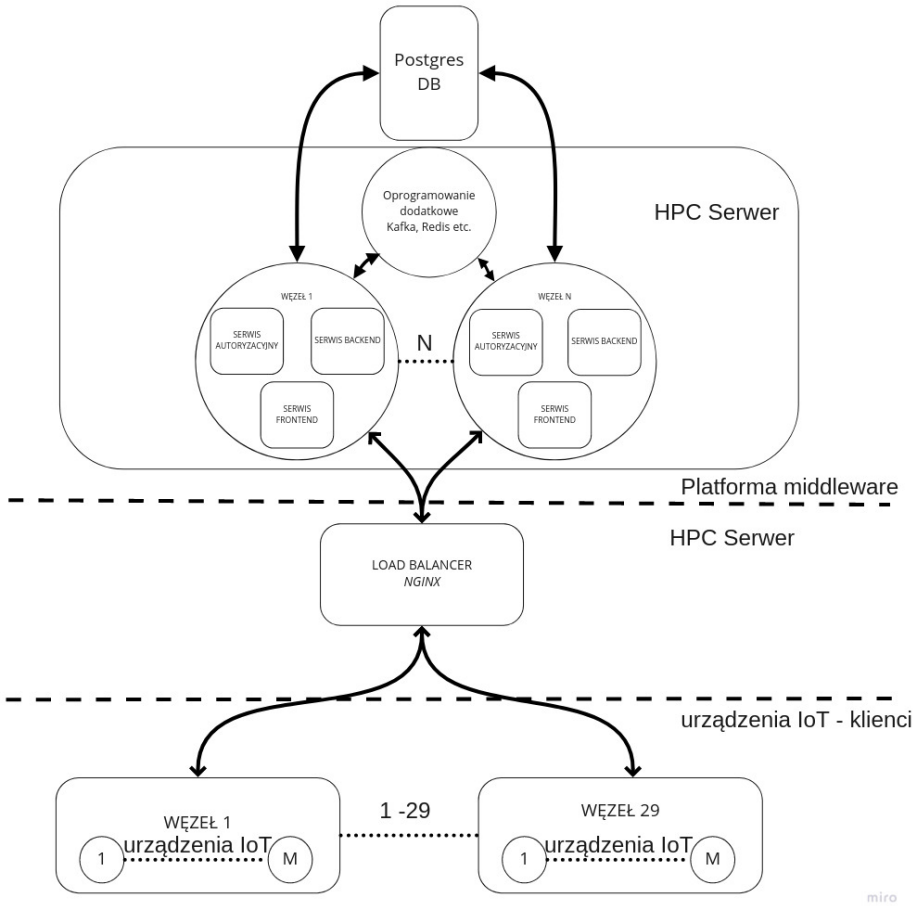
Zaproponowany model aplikacyjny angażuje dwie pierwsze warstwy. Czujniki są symulowane przy użyciu klastra komputerów, podczas gdy *middleware* jest uruchomiony na serwerze HPC. Czujniki w pętli wysyłają powiadomienia do *middleware*, który odbiera je i przechowuje w szybkiej, działającej w czasie działania aplikacji, ulotnej bazie danych. Zebrane dane mogłyby później być przetwarzane przez platformy chmurowe, jednak jest to poza zakresem przeprowadzonych badań. Podobny model aplikacyjny przyjęto w [30]. Autorzy zaprezentowali rozwiązanie IoT, którego celem było mierzenie poziomu zanieczyszczeń. Odczyty czujników były cyklicznie przesyłane do serwera, który je przetwarzał. Autor niniejszego rozdziału zdecydował się na wysyłanie danych w sposób ciągły, bez przerwy między kolejnymi pakietami, tak by zasymulować możliwie najwyższe obciążenie po stronie warstwy *middleware*.

14.5.1. Przegląd architektury

W przedstawionym badaniu architektura składała się z symulatorów urządzeń, łączących się przez *load balancer* do warstwy *middleware*. W zależności od testowanej konfiguracji liczba węzłów po stronie warstwy *middleware* może być różna, ponadto każdy testowy przypadek został uruchomiony z innym przydziałem rdzeni dla węzłów warstwy *middleware*. Funkcjonalności warstwy *middleware* są zduplikowane przy pomocy węzłów (kontenerów) tylko w zakresie mikroserwisów. Wszystkie wspólne elementy systemu, jak dodatkowe oprogramowanie (np. Kafka, Redis), zostały uruchomione na niezależnym kontenerze. Pozwoliło to na zapis danych z różnych kontenerów obsługujących żądania do jednej wspólnej kolejki i bazy. Również konfiguracja mikroserwisów zapisana została w jednej wspólnej bazie danych uruchomionej na Postgres poza testowanym środowiskiem. Zaproponowaną architekturę przedstawiono na rys. 14.2.

14.5.2. Symulator urządzeń

Symulację urządzeń uruchomiono w klastrze złożonym z 29 komputerów wyposażonych w procesory Intel Xeon E5345 @2.33GHz, 8GB RAM i pracujących pod kontrolą CentOS 6. Na każdej z maszyn uruchomionych było kilkadziesiąt symulatorów klientów (urządzeń) – testy zostały rozpoczęte z 30 urządzeniami na każdej stacji, a najbardziej obciążające konfiguracje uruchamiały 90 urządzeń na stację. W ten sposób uzyskano całkowite obciążenie w przedziale od 870 do



Rysunek 14.2: Architektura testowanego rozwiązania

2610 urządzeń pracujących równolegle. Symulator klienta to aplikacja w języku Java, która w pętli generuje 36 losowych bajtów danych (symulacja odczytu czujnika) i wysyła je poprzez REST API szyfrowanym połączeniem (HTTPS) do serwera *middleware*. Każdy klient działa w trakcie testu w określonym oknie czasowym trwającym 10 minut i wszyscy klienci uruchamiani są w tym samym czasie.

14.5.3. Warstwa *middleware*

Dzisiejsze aplikacje webowe bazują na mikroserwisach. *Middleware* dla IoT z tej perspektywy zawiera podobne funkcjonalności (np. kontrolery dostępne przez REST API), zatem również powinien być złożony z mikroserwisów. Ponadto jego architektura nie powinna być zbyt rozległa i złożona, by czas uruchomienia

poszczególnych testów nie był zbyt długi. W związku z tym autor niniejszego rozdziału zdecydował się wybrać rozwiązanie DeviceHive, składające się z trzech mikroserwisów:

- *frontend* – wystawia API komunikacyjne przez REST oraz WebSocket;
- *authorization* – realizuje uwierzytelnienie i autoryzacje użytkowników z wykorzystaniem tokenów JWT do realizacji dostępu do serwisu *frontend*;
- *backend* – serwis odpowiedzialny za realizację logiki biznesowej.

Platforma korzysta również z dodatkowego oprogramowania, takiego jak baza danych Postgres, w której przechowuje konfigurację i użytkowników. W celu zapewnienia małych opóźnień i skalowalności rozwiązania przetwarzane dane w trakcie działania aplikacji są kolejkowane z użyciem oprogramowania Kafka i zapisywane do szybkiej bazy w pamięci – Redis. Powyższe rozwiązanie zostało uruchomione na kontenerach Docker na stacji HPC wyposażonej w dwa procesory Intel Xeon E5-2680 v2 @2.8 Ghz z ogólną liczbą 40 logicznych rdzeni (z technologią HT). Stacja posiadała 128 GB pamięci RAM i pracowała pod kontrolą systemu operacyjnego Ubuntu.

14.5.4. *Load balancer*

Równoważenie obciążenia zrealizowano na oprogramowaniu nginx. Zostało skonfigurowane w taki sposób, by wystawiać dwa serwery – jeden dla autoryzacji i jeden dla serwisu *frontend*. Oba wykorzystywały algorytm *round-robin* w celu rozłożenia ruchu na poszczególne kontenery. Liczba połączeń i konfiguracja *backlog* zostały ustawione na wartość 9000, by zapobiec zakleszczeniu na poziomie *load balancera*. Ponadto terminacja SSL odbywa się na *load balancerze*, a dalsza komunikacja jest realizowana nieszyfrowanym HTTP. *Load balancer* został uruchomiony na maszynie wyposażonej w 20-rdzeniowy Intel Xeon CPU i 64 GB RAM.

14.5.5. Opis testu

Test rozpoczyna się startem wszystkich kontenerów i ich serwisów. Następnie wysyłane są trzy różne wiadomości, by rozgrzać *middleware* – załadować niezbędne klasy do JVM oraz zainicjalizować pozostałe oprogramowanie. Te wiadomości nie są brane pod uwagę podczas dalszej analizy. Po poprawnym przetworzeniu wiadomości rozgrzewających rozpoczyna się główna część testu. Wszyscy klienci zaczynają wysyłać swoje wiadomości. Ta część trwa 10 minut. Po tym czasie następuje wyłączenie klientów i oprogramowania *middleware*. Zebrane dane są odczytywane z kontenerów i klientów i zapisywane w oddzielnym katalogu. Dostarczają one następujących informacji:

- liczba wiadomości przetworzonych w czasie testu;
- czas przetwarzania logiki biznesowej, licząc od pierwszego serwletu w oprogramowaniu *middleware*;

- poprawność przetwarzania – informacja po stronie oprogramowania *middleware*.

Powyższy test został powtórzony dla następujących konfiguracji warstwy *middleware*:

- dwa węzły z przydziałem rdzeni od 0 do 9 dla węzła nr 1 i od 10 do 19 dla węzła nr 2;
- dwa węzły z przydziałem rdzeni od 0 do 19 dla obu węzłów;
- pięć węzłów z przydziałem rdzeni od 0 do 3 dla węzła nr 1, od 4 do 7 dla węzła nr 2, od 8 do 11 dla węzła nr 3, do 12 do 15 dla węzła nr 4, od 16 do 19 dla węzła nr 5;
- pięć węzłów z przydziałem rdzeni od 0 do 19 dla wszystkich węzłów;
- siedem węzłów z przydziałem rdzeni od 0 do 2 dla węzła nr 1, od 3 do 5 dla węzła nr 2, od 6 do 8 dla węzła nr 3, od 9 do 11 dla węzła nr 4, od 12 do 14 dla węzła nr 5, od 15 do 17 dla węzła nr 6 i od 18 do 20 dla węzła nr 7;
- siedem węzłów z przydziałem rdzeni od 0 do 20 dla wszystkich węzłów.

Przydział rdzeni został zrealizowany z użyciem komendy *taskset* wewnątrz kontenera. Testowana stacja była wyposażona w dwa procesory, każdy po 10 fizycznych rdzeni. Rdzenie o numerach od 0 do 9 to fizyczne rdzenie procesora 0, a o numerach od 10 do 19 to fizyczne rdzenie procesora 1.

Każda konfiguracja została przetestowana siedem razy. Po statystycznej analizie wyników, bazując na medianie czasów przetwarzania, odrzucono najgorszy i najlepszy wynik. Następnie wyznaczono z pozostałych pięciu testów medianę czasu przetwarzania, średnią liczbę przetworzonych wiadomości i liczbę błędnie przetworzonych wiadomości.

14.6. Przyjęte wskaźniki

Badanie miało na celu porównanie wydajności pomiędzy wielowęzłowymi konfiguracjami, w których ściśle określona liczba rdzeni jest przypisana do wszystkich węzłów wspólnie lub taka sama liczba rdzeni jest przypisana indywidualnie w ramach podgrup do jednego węzła. Do określenia charakteru różnic pomiędzy konfiguracjami przyjęto następujące wskaźniki:

1. liczba poprawnie przetworzonych wiadomości;
2. wskaźnik błędów;
3. czas przetwarzania przez serwis *frontend*.

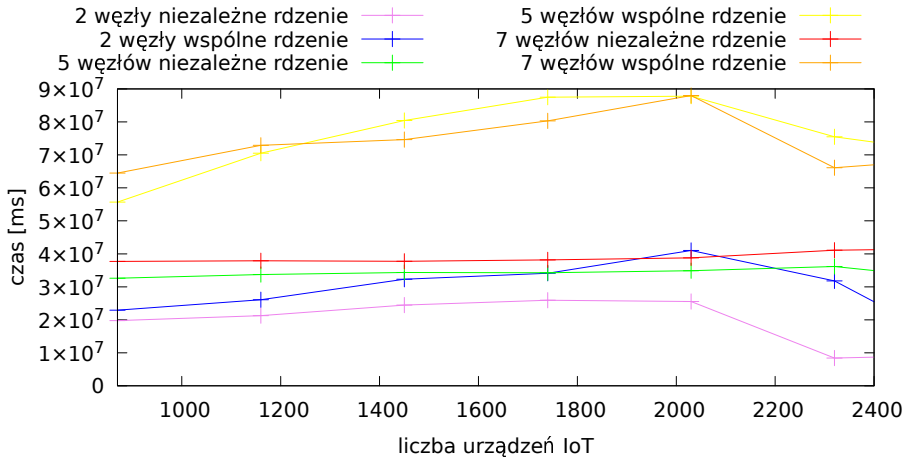
Pierwszy element reprezentuje liczbę poprawnie przetworzonych wiadomości w czasie całego testu – 10-minutowego okna czasowego. „Poprawnie“ w tym wypadku oznacza, że *middleware* odpowiedział potwierdzeniem odbioru na odebraną wiadomość. Drugi parametr określa procentową liczbę błędów (odpowiedzi przetworzonych niepoprawnie). Ostatni element przedstawia czas przetwarzania po stronie serwera, mierzony wewnątrz pierwszego serwletu w serwisie *frontend*. Należy zaznaczyć, że całkowity czas przetwarzania po stronie serwera będzie zawierał również inne elementy związane z przetwarzaniem żądania przed i po jego obsłużeniu przez sam serwis *frontend*.

14.7. Analiza i prezentacja wyników

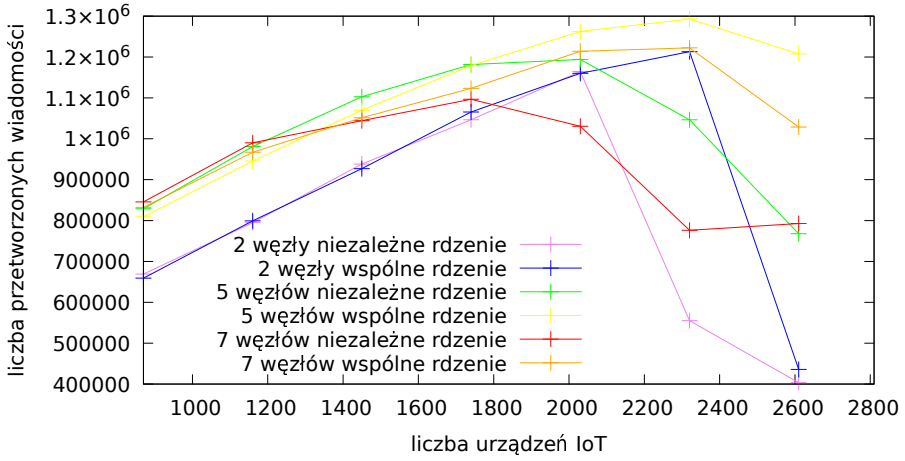
Czas przetwarzania przez serwis *frontend* platformy *middleware* został przedstawiony na rys. 14.3. Można poczynić dwie obserwacje. W sytuacji, gdy więcej rdzeni jest zaangażowanych do przetwarzania, czas przetwarzania jest znacząco krótszy (konfiguracja z dwoma węzłami ma znacząco krótsze czasy niż inne) i jest to zachowanie zgodne z oczekiwaniami. Zwiększanie liczby rdzeni poprawia skalowalność rozwiązania. Ta obserwacja została dokładnie przeanalizowana w artykule [13]. Druga, równie istotna obserwacja dotyczy przetwarzania w tych samych konfiguracjach węzłów, ale różnych przydziałach rdzeni. W sytuacji, gdy każdy węzeł ma przypisany indywidualny zestaw rdzeni, oprogramowanie przedstawia znacznie stabilniejsze wyniki i czas przetwarzania jest znacząco krószy. Oznacza to, że JVM, który realizuje logikę biznesową aplikacji, jest bardziej wydajny, kiedy rdzenie są przypisane niezależnie do kontenera. Należy tu jednak wskazać, że mierzony czas dotyczy wyłącznie przetwarzania w ramach serwisu *frontend* w oprogramowaniu *middleware*. Nie uwzględnia on opóźnień serwera, warstwy systemu operacyjnego i innych operacji wymaganych do poprawnego przetworzenia żądania HTTP. Ma to istotny wpływ na wyniki, gdyż jak widać na rys. 14.4, ogólna liczba poprawnie przetworzonych żądań jest mimo wszystko większa dla konfiguracji z większą liczbą rdzeni w ramach kontenera, ale współdzielonych między kontenery. Oznacza to, że całość przetwarzania poza samym serwisem *frontend* radzi sobie znacznie lepiej, kiedy jest dostępnych więcej rdzeni, nawet jeśli są współdzielone. Z drugiej strony biorąc pod uwagę rys. 14.5, który prezentuje poziom błędów przetwarzania, widać, że ta konfiguracja poza lepszą wydajnością ma również znacząco większy poziom błędów, co najbardziej widoczne jest dla konfiguracji pięciu i siedmiu węzłów. W przypadku konfiguracji z przydzielonymi niezależnymi rdzeniami poziom błędów zawsze jest w okolicy 0% a w tych samych przypadkach ze wspólnymi rdzeniami oscyluje w zakresie 2–4%.

14.8. Podsumowanie i kierunek dalszych badań

Optymalizacja w celu najbardziej efektywnego wykorzystania sprzętu jest ważnym tematem. W badaniu zaprezentowanym w niniejszym rozdziale przeana-

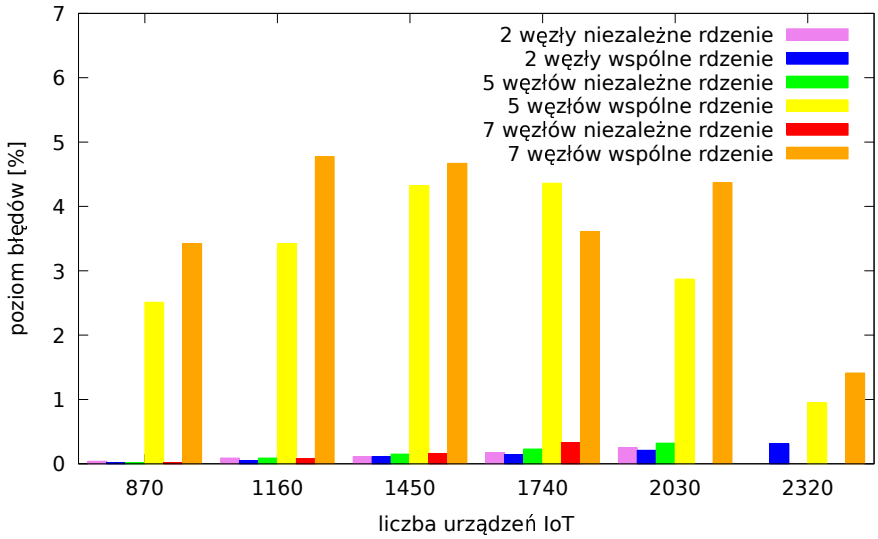


Rysunek 14.3: Czas przetwarzania w serwisie *frontend*



Rysunek 14.4: Liczba poprawnie przetworzonych żądań

lizowano, czy przypisywanie określonych i niewspółdzielonych między kontenerami zestawów rdzeni procesora wpłynie na uzyskaną wydajność rozwiązania. Przygotowany testowy przypadek oparty na aplikacji rzeczywistej pokazał, że pod dużym obciążeniem konfiguracje z większą liczbą współdzielonych rdzeni uzyskały około 10% lepsze wyniki wydajnościowe, jednak odbyło się to kosztem



Rysunek 14.5: Poziom błędów

jakości – zaobserwowano znaczący wzrost liczby błędów. Konfiguracja z zestawem niewspółdzielonych rdzeni na kontener uzyskała znacząco stabilniejsze rezultaty i niższe czasy samego przetwarzania po stronie aplikacyjnej. Biorąc pod uwagę przedstawione obserwacje, można stwierdzić, że w zależności od tego, jaki jest oczekiwany cel, zastosowanie obu rozwiązań może okazać się właściwe. W przypadku, gdy oczekiwana jest wysoka niezawodność, lepiej przydzielić niezależne rdzenie dla każdego z kontenerów planowanego rozwiązania, natomiast gdy oczekiwana jest duża wydajność, lepiej wszystkie posiadane rdzenie współdzielić pomiędzy wiele kontenerów. Rozważając kierunek dalszych badań, należałoby przeprowadzić analizę wydajności na poziomie serwera i systemu operacyjnego, by stwierdzić, czy optymalizacja poprzez różne przypisanie rdzeni pozwoli osiągnąć lepsze wyniki na tym poziomie.

Bibliografia

[1] Abdul M.S., et al. *Docker Containers Usage in the Internet of Things: A Survey. Open International Journal of Informatics*, 7(Special Issue 2):208–220, 2019. URL `https://oiji.utm.my/index.php/oiji/article/view/95`.

[2] Ahmed B., et al. *Container Based Resource Management for Data Processing on IoT Gateways. Procedia Computer Science*, 155:234–241, 2019. ISSN 1877-0509. URL `https://www.sciencedirect.com/science/article/pii/S1877050919309482`, the 16th International Conference on

Mobile Systems and Pervasive Computing (MobiSPC 2019),The 14th International Conference on Future Networks and Communications (FNC-2019),The 9th International Conference on Sustainable Energy Information Technology.

[3] Alam M., et al. *Orchestration of Microservices for IoT Using Docker and Edge Computing. IEEE Communications Magazine*, 56(9):118–123, 2018.

[4] Caculo S., Lahiri K., Kalambur S. *Characterizing the Scale-Up Performance of Microservices using TeaStore.* In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 48–59. 2020.

[5] Cho J.Y., et al. *On the Core Affinity and File Upload Performance of Hadoop.* In *Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems*, DISCS-2013, p. 25–30. Association for Computing Machinery, New York, NY, USA, 2013. ISBN 9781450325066. URL https://doi.org/10.1145/2534645.2534651.

[6] da Cruz M.A., et al. *Performance evaluation of IoT middleware. Journal of Network and Computer Applications*, 109:53–65, 2018. ISSN 1084-8045. URL https://www.sciencedirect.com/science/article/pii/S108480451830064X.

[7] da Silva V.G., Kirikova M., Alksnis G. *Containers for Virtualization: An Overview. Applied Computer Systems*, 23(1):21–27, 2018.

[8] Dua R., Raja A.R., Kakadia D. *Virtualization vs Containerization to Support PaaS.* In *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614. 2014.

[9] Felter W., et al. *An updated performance comparison of virtual machines and Linux containers.* In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172. 2015.

[10] Hashemian R., et al. *Improving the Scalability of a Multi-Core Web Server.* ICPE '13, p. 161–172. Association for Computing Machinery, New York, NY, USA, 2013. ISBN 9781450316361. URL https://doi.org/10.1145/2479871.2479894.

[11] Islam J., et al. *Docker Enabled Virtualized Nanoservices for Local IoT Edge Networks.* In *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 1–7. 2019.

[12] Ismail A.A., Hamza H.S., Kotb A.M. *Performance Evaluation of Open Source IoT Platforms.* In *2018 IEEE Global Conference on Internet of Things (GCIoT)*, pp. 1–5. 2018.

[13] Kałaska R., Czarnul P. *Investigation of Performance and Configuration of a Selected IoT System - Middleware Deployment Benchmarking and Recommendations. Applied Sciences*, 12(10), 2022. ISSN 2076-3417. URL https://www.mdpi.com/2076-3417/12/10/5212.

[14] Love R. *CPU Affinity*. URL https://dl.acm.org/doi/fullHtml/10.5555/860375.860383.

[15] Love R.M. *taskset(1) — Linux manual page*. URL https://man7.org/linux/man-pages/man1/taskset.1.html.

[16] Lynch K. *Understanding Linux Container Scheduling*. URL https://engineering.squarespace.com/blog/2017/understanding-linux-container-scheduling.

[17] Mazaheri S., et al. *Cloud benchmarking in bare-metal, virtualized, and containerized execution environments*. In *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pp. 371–376. 2016.

[18] Morabito R., et al. *Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge. IEEE Internet of Things Journal*, 4(4):1019–1030, 2017.

[19] Noor S., et al. *IoTDoc: A Docker-Container Based Architecture of IoT-Enabled Cloud System*, pp. 51–68. Springer International Publishing, Cham, 2020. ISBN 978-3-030-24405-7. URL https://doi.org/10.1007/978-3-030-24405-7_4.

[20] Pabla C.S. *Completely Fair Scheduler*. URL https://dl.acm.org/doi/fullHtml/10.5555/1594371.1594375.

[21] Potdar A.M., et al. *Performance Evaluation of Docker Container and Virtual Machine. Procedia Computer Science*, 171:1419–1428, 2020. ISSN 1877-0509. URL https://www.sciencedirect.com/science/article/pii/S1877050920311315, third International Conference on Computing and Network Communications (CoCoNet'19).

[22] Putri A.R., Munadi R., Negara R.M. *Performance analysis of multi services on container Docker, LXC, and LXD. Bulletin of Electrical Engineering and Informatics*, 9(5):2008–2016, 2020. 10.11591/eei.v9i4.1953.

[23] Rosen R. *Namespaces and Cgroups – the basis of Linux Containers*. URL https://www.andrew.cmu.edu/course/14-712-s20/applications/ln/Namespaces_Cgroups_Conatiners.pdf.

[24] S. Rethinavalli R.G. *BOTNET ATTACK DETECTION IN INTERNET OF THINGS USING OPTIMIZATION TECHNIQUES. International Journal of Electrical Engineering and Technology (IJEET)*, 11(4), 2020. ISSN 0976-6545. URL https://iaeme.com/Home/issue/IJEET?Volume=11&Issue=10.

[25] Samie F., et al. *Computation offloading and resource allocation for low-power IoT edge devices*. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 7–12. 2016.

[26] Savaglio C., et al. *Data Mining at the IoT Edge*. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–6. 2019.

[27] Truong H.L., Dustdar S. *Principles for Engineering IoT Cloud Systems. IEEE Cloud Computing*, 2(2):68–76, 2015.

[28] Virtualization, of ApplicationInfrastructure: A Comparison C. *Completely Fair Scheduler*. URL `https://thijs.ai/papers/scheepers-virtualization-containerization.pdf`.

[29] Xavier M.G., et al. *Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments*. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 233–240. 2013.

[30] Xiaojun C., Xianpeng L., Peng X. *IOT-based air pollution monitoring and forecasting system*. In *2015 International Conference on Computer and Computational Sciences (ICCCS)*, pp. 257–260. 2015.

[31] Zikria Y.B., et al. *Next-Generation Internet of Things (IoT): Opportunities, Challenges, and Solutions. Sensors*, 21(4), 2021. ISSN 1424-8220. URL `https://www.mdpi.com/1424-8220/21/4/1174`.