POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Specjalizacja: Interakcja człowiek-komputer

Paulina Żak

Nr albumu s18257

Analiza systemu rozpoznającego zagnieżdżone nazwy własne przy wykorzystaniu parsowania zależnościowego

praca magisterska

dr. hab. Agnieszka Mykowiecka

Warszawa, Czerwiec, 2020r.

Spis treści

1 Wstęp **5**

2 Rozpoznawanie nazw własnych **7**

2.1 Definicja zadania 7

2.2 Zasady anotacji nazw własnych 8

2.3 Rozpoznawanie nazw własnych – stan badań 11

2.3.1 Systemy regułowe 11

2.3.2 Metody słownikowe 12

2.3.3 Uczenie maszynowe 12

3 Dane **15**

3.1 Kategorie 15

3.2 Podkorpus milionowy 16

3.3 Korpus testowy 19

4 Przetworzenie danych tekstowych **21**

4.1 Podział na zdania 21

4.2 Segmentacja 21

4.3 Analiza zależnościowa 22

4.4 Reprezentacja słów w przestrzeni wektorowej 23

5 Modele uczenia maszynowego **31**

5.1 Sieć neuronowa 31

5.2 Rekurencyjne sieci neuronowe 33

5.3 Warunkowe pola losowe (CRF) 36

5.4 LSTM-CRF 37

5.5 Tree-LSTM 38

6 Opis opracowanego rozwiązania **41**

6.1 Zbiory danych 42

6.1.1 Plik korpusu treningowego i walidacyjnego . . . 43
6.1.2 Plik korpusu testowego . . . 43
6.1.3 Przygotowanie danych . . . 44
6.2 Modele . . . 46
6.2.1 Padding . . . 46
6.2.2 Opis modelu GRU/LSTM(-CRF) . . . 46
6.2.3 Opis modelu TreeLSTM . . . 48
6.3 Wyjście . . . 50
6.3.1 Reprezentacja wyjścia . . . 50

7 Ewaluacja **53**
7.1 System porównywania zakresów . . . 53
7.2 Metryki . . . 54
7.3 Wyniki . . . 54

8 Analiza błędów **59**
8.1 Błędy ogólne . . . 59
8.2 Błędy specyficzne dla fraz zagnieżdżonych . . . 63

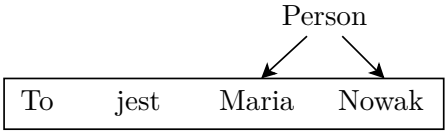
9 Podsumowanie **67**

10 Appendix **79**

Rozdział 1

Wstęp

Rozpoznawanie nazw własnych jest powszechnym problemem z dziedziny przetwarzania języka naturalnego (NLP).
Zadanie to polega na automatycznym znajdowaniu fraz, które zostaną uznane przez system za nazwy własne, a następnie oznaczeniu ich zgodnie z zadaną kategoryzacją. Na rysunku 1.1 pokazany jest przykład jak wygląda oznaczenie nazwy własnej, “Maria Nowak”, w zdaniu i zaklasyfikowanie tej frazy do kategorii osoba.
Systemy rozpoznające nazwy własne mogą być wykorzystywane jako niezależne programy. Jednakże mają też szerokie zastosowanie jako moduły do innych zadań takich jak tłumacze maszynowe czy chatboty.



Rysunek 1.1: Zdanie z nazwą własną. ’ Maria Nowak ’ jest oznaczona jako osoba

Poniższa praca skupia się na sytuacjach, gdy frazy będące częścią jednej nazwy własnej stanowią osobną nazwę innej kategorii. W zdaniu “Spotkaliśmy się na placu Jana Pawła II”, “Jan Paweł II” jest odniesieniem do osoby i jednocześnie częścią nazwy miejsca “plac Jana Pawła II”. Większość rozwiązań komputerowych rozpoznaje tylko najbardziej zewnętrzną kategorię frazy, całkowicie ignorując elementy zagnieżdżone. Przez co informacje przekazane do użytkownika są niepełne i nie odzwierciedlają stanu rzeczywistego.
Szczególnie istotną dziedziną, która wykorzystuje zagnieżdżenia jest biomedycyna, w której nazwy kolejnych struktur często budowane są przy wykorzystaniu istniejących. I tak na przykład antybiotyk “Penicylina” stał się podstawą do utworzenia nazwy enzymu wiążącego “Białka wiążace Penicylinę”(Penicillin Binding Proteins).

Celem przedstawionej pracy było zbadanie jak wyniki parsowania zależnościowego wpływają na rozpoznawanie zagnieżdżonych nazw własnych w korpusie języka polskiego.

Znajomość struktury zależnościowej w zdaniu wzbogaca informacje przekazane do systemu o powiązaniach między poszczególnymi słowami. W teorii dzięki tej wiedzy system może wydajniej przewidywać nawet skomplikowane frazy. W kolejnych rozdziałach pracy znajduje się opis systemu rozpoznającego nazwy własne[1] wraz z analizą jego wyników.

[1]Kod dostępny na stronie: https://github.com/Zuchens/ner_poleval

Rozdział 2

Rozpoznawanie nazw własnych

2.1 Definicja zadania

Zadanie rozpoznawania nazw własnych polega na znalezieniu i kategoryzowaniu fraz w tekście odnoszących się do rzeczywistych obiektów, które posiadają identyfikujące je nazwy. Dziedzina zajmująca się analizą nazw własnych w języku nazywa się onomastyka.
Rzeczowniki pospolite reprezentują pojęcia ogólne, zbiory cech lub obiektów. Natomiast nazwy własne wskazują najczęściej na pojedyncze obiekty lub ich niewielkie zbiory (w przypadku nazw niejednoznacznych). Wyrazy pospolite mają bardzo szeroki zakres, ale dość ubogą treść - niosą wyłącznie informacje wspólne dla dużej grupy obiektów. W porównaniu do nich nazwy własne mają zwykle bardzo wąski zakres, ale treść może być bogata. W zdaniach

"*Wspiąłem się na Everest*"

oraz

"*Wspiąłem się na szczyt*",

"Everest" jest nazwą własną, a słowo "szczyt" nazwą pospolitą. Mimo tego, że dla wypowiadającego te dwa zdania mogą oznaczać to samo miejsce, to odbiorca otrzymuje więcej informacji w zdaniu pierwszym. "Everest" jednoznacznie oznacza Górę Everest, która jest najwyższym szczytem na Ziemie, znajdującym się w Paśmie Górskim Himalajów. "Szczyt" natomiast może być dowolnym szczytem na Ziemi i nie jest podana żadna dokładniejsza informacja na jego temat jak np. wysokość czy położenie. Jednakże, warto zwrócić uwagę, iż nazwy własne nie muszą jednoznacznie opisywać danego obiektu. Mimo tego, że "Jan Kowalski" jest nazwą własną, może odnosić się do jednej z wielu osób o takim imieniu i nazwisku.

W zadaniach przetwarzania języka naturalnego dokładny termin określający co uznajemy za nazwę własną i jaką posiada kategorie jest mocno uzależniony od zastosowań tych

oznaczeń. Mogą na nie wpłynąć różne czynniki jak np.

- **pochodzenie tekstów** - np. twitter czy Wikipedia,
- **przeznaczenia systemu** - np. chatbot, analiza dokumentów medycznych,
- **ustaleń wewnętrznych** - to, czy fraza "król Artur" powinna być uznany w całości za nazwę własną, czy taką etykietę należy przypisać tylko słowu "Artur" zależy od przyjętych założeń.

Rozpoznanie nazw własnych stanowi jeden z ważnych elementów analizy tekstu. Systemy komputerowe zajmujące się tym zadaniem w obecnym formacie stały się popularne na początku lat dziewięćdziesiątych i od tej pory trwają prace nad poprawą ich jakości.
Prace nad poprawą systemów rozpoznawania nazw własnych przyniosły niemałe rezultaty. Aktualnie różne wersje tych programów są powszechnie wykorzystywane do różnych zadań, jak na przykład: automatyczna ekstrakcja danych użytkownika do wypełnienia formularza czy anonimizacja wrażliwych danych.
Rozpoznane typy nazw własnych często są wykorzystywane jako dodatkowa informacja w innych zadaniach przetwarzania języka naturalnego. Na przykład w tłumaczeniu maszynowym, dzięki informacji o rozpoznaniu nazwy własnej można zapobiec niepotrzebnemu przekształcaniu nazwy, która mimo przełożenia całego tekstu na inny język nie powinna zostać zmieniona. **"I went to Apple store"**, oznacza **"Poszedłem do sklepu Apple"**, a nie **"Poszedłem do sklepu Jabłko"**.

Ze względu na różnorodność zastosowań systemów identyfikujących nazwy własne w tekście zainteresowanie poprawą ich wydajności i jakości wyników nie maleje. Z tego powodu cały czas pojawiają się nowe konkursy, artykuły i zbiory danych dla tego zadania.

2.2 Zasady anotacji nazw własnych

Anotacje nazw własnych można podzielić na dwa podzadania:

- **anotacje zakresu nazwy**
- **anotacje kategorii**

Zwykle wybór zakresu i kategorii związany jest z docelowym przeznaczeniem systemu oraz technicznymi ograniczeniami przeszkadzającymi w utworzeniu jak najlepszego modelu do predykcji.
Poniżej znajduje się opis w jaki sposób zostały utworzone najbardziej popularne zbiory do tego zadania.

Zakres nazwy Jednym z najpopularniejszych korpusów do rozpoznawania nazw własnych w języku angielskim jest ten pochodzący z konferencji z MUC [4]. Posiada on obszerne reguły anotacji, które później stały się podstawą do tworzenia innych zbiorów dla tego zadania.
Anotacja nazw własnych, choć w teorii jest intuicyjna, to w praktyce ręczne oznaczenie fraz jest zadaniem czasochłonnym, a wyniki uzyskane przez różne osoby mogą być znacząco odmienne. Dodatkowy czas potrzebny jest zatem na ujednolicenie wyników anotacji.
W szczególności trudnym problemem jest uściślenie zakresu fragmentów tekstu, którym przypisywane są etykiety. Aby zminimalizować powstające różnice, przed rozpoczęciem procesu oznaczania, tworzone są dokumenty zawierające wytyczne anotacji. Oto niektóre zasady obowiązujące w korpusie pochodzącym z Seventh Message Understanding Conference:

1. Nazwy własne będące częścią frazy, która nie jest nazwą własną są oznaczane odpowiednim typem.
   "the Clinton government"
   the <ENAMEX TYPE="PERSON">Clinton</ENAMEX> government

2. tytuły lub role nie są anotowane w ramach etykiet nazw osób
   "the U.S. Vice President"
   the <ENAMEX TYPE="LOCATION">U.S.</ENAMEX> Vice President

3. znaki cytowania powinny być zawarte we frazie
   "...as 'The Godfather', was ..."
   ...as <ENAMEX TYPE="PERSON">"The Godfather"</ENAMEX>...

Dodatkowym utrudnieniem zadania jest występowanie nazw zagnieżdżonych. Ze względów technicznych często anotacja takich nazw jest upraszczana i ograniczona wyłącznie do najwyższego poziomu.

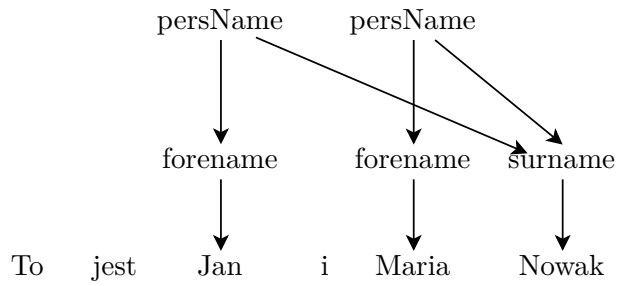
"*św. Jan*"

we frazie

"*ul. św. Jana*"

mógłby być rozpoznany i oznaczony jako odniesienie do osoby, ale często jedyną etykietą, która jest w tu używana jest etykieta oznaczająca miejsce (ulicę).

Innym trudnym przypadkiem są frazy nieciągłe, pojawiające się w szczególności w przypadku fraz skoordynowanych (Przykład rys: 2.1). Frazy te rzadko występują w korpusie, zatem nauczenie modeli takich informacji jest trudne. Dodatkowo dostosowanie architektury systemu pod takie przypadki wymaga dużego nakładu pracy i zwiększenia złożoności systemu, co niekoniecznie przekłada się na końcowe, całkowite wyniki ewaluacji.



Rysunek 2.1: Przykład zdania z frazą nieciągłą. W zdaniu można zidentyfikować dwie nazwy osób: 'Jan Nowak ' i 'Maria Nowak ', przy czym część oznaczająca nazwisko jest współdzielona i jednocześnie sama nazwa "Jan Nowak" jest nieciągła.

Kategorie W kolejnej popularnej odsłonie omawianego zadania z CONLL-2003 [1] oznaczone są cztery kategorie nazw własnych tj.

- **Osoba** (imię, nazwisko, przydomek),
- **Miejsce** (kraj, miasto, góra),
- **Organizacja** (uniwersytet, firma),
- **Inne** (wojny, tytuły książek/filmów).

Powyższa kategoryzacja została przyjęta jako podstawa w większości następnych zbiorów rozpoznawania nazw własnych, istnieją jednak także inne korpusy z bardziej złożoną kategoryzacją, która może składać się nawet z setek typów. Przykładem może być: Sekine Extended Named Entity Hierarchy [2], która posiada 150 kategorii, takie jak Bóg, Produkt czy Choroba.

Bardzo ważnym elementem przy ustaleniu kategorii, do której należy konkretne wystąpienie napisu oznaczającego nazwę własną, jest kontekst wypowiedzi. Często bowiem taka sama nazwa przypisywana jest do obiektów różnej kategorii. Przykładowo, w zależności od zdania, "Paris" może oznaczać imię bądź miasto. Zwykle kontekst użycia pozwala na ustalenie o jaki typ obiektu chodzi, ale czasem kontekst ten jest zbyt krótki i nie jest możliwe wiarygodne ocenienie jaki rodzaj powinna mieć dana fraza. Jeżeli z kontekstu nie będzie wynikało czy "Kraków" odnosi się do gminy czy do miasta, a nie są przewidziane etykiety wieloznaczne, jedynie w gestii anotatora pozostaje wybór kategorii.

Kolejnym problemem, który muszą rozwiązać anotatorzy korpusów jest decyzja jak oznaczać metonimie. W zdaniu

"*Biały Dom ogłosił konferencję*"

nazwa "*Biały Dom*"

tak naprawdę odnosi się do sekretarza prasowego Białego Domu, oznajmującego o decyzji rządu. Niektórzy twórcy korpusów próbują rozwiązać ten problem poprzez dodanie osobnych kategorii. Na przykład duński korpus nazw własnych [21] posiada takie tagi jak "Miejsce jako Człowiek" (LOC as human). Jednak najczęstszym rozwiązaniem jest po prostu oznaczenie frazy bardziej popularną kategorią.

2.3 Rozpoznawanie nazw własnych – stan badań

Dla człowieka rozpoznanie czy dane słowo jest nazwą własną będzie dość łatwym zadaniem. Nawet jeśli frazę słyszy po raz pierwszy, to jest w stanie z dość dużą pewnością stwierdzić czy jest ona nazwą własną. Decyzje podejmuje na podstawie różnych źródeł, na przykład kontekstu zdania czy struktury frazy. W zdaniu "*Pracuję w BlopTech*", "*BlopTech*" to prawdopodobnie firma z zajmująca się obszarem technologii.
Dodatkowe informacje pochodzą także z wiedzy dziedzinowej. Nawet bez znajomości kontekstu, fraza "*Znam Londyn*" prawdopodobnie będzie się odnosiła do miasta.

Dla systemów komputerowych zadanie rozpoznawania nazw własnych to nie jest oczywiste i różne podejścia do jego rozwiązania były testowane przez ostatnie lata. Wyniki, jakkolwiek bardzo dobre, nie są jednakowo dobre dla wszystkich kategorii nazw i typów tekstów.

Zainteresowanie automatycznym rozpoznawaniem nazw własnych zaczęło przybierać na sile w roku 1995 wraz z pierwszym większym wydarzeniem, konferencją Message Understanding Conference (MUC)[5]. W tamtym czasie większość systemów opierała się na regułowym lub słownikowym podejściu do problemu. Jednak wraz z upływem czasu pojawiały się coraz bardziej złożone systemy. Obecne systemy wykorzystują zwiększoną moc obliczeniową komputerów oraz nowe wyniki badań z dziedziny sztucznej inteligencji. W ostatnich latach większość rozwiązań jest oparta na głębokim uczeniu maszynowym.

2.3.1 Systemy regułowe

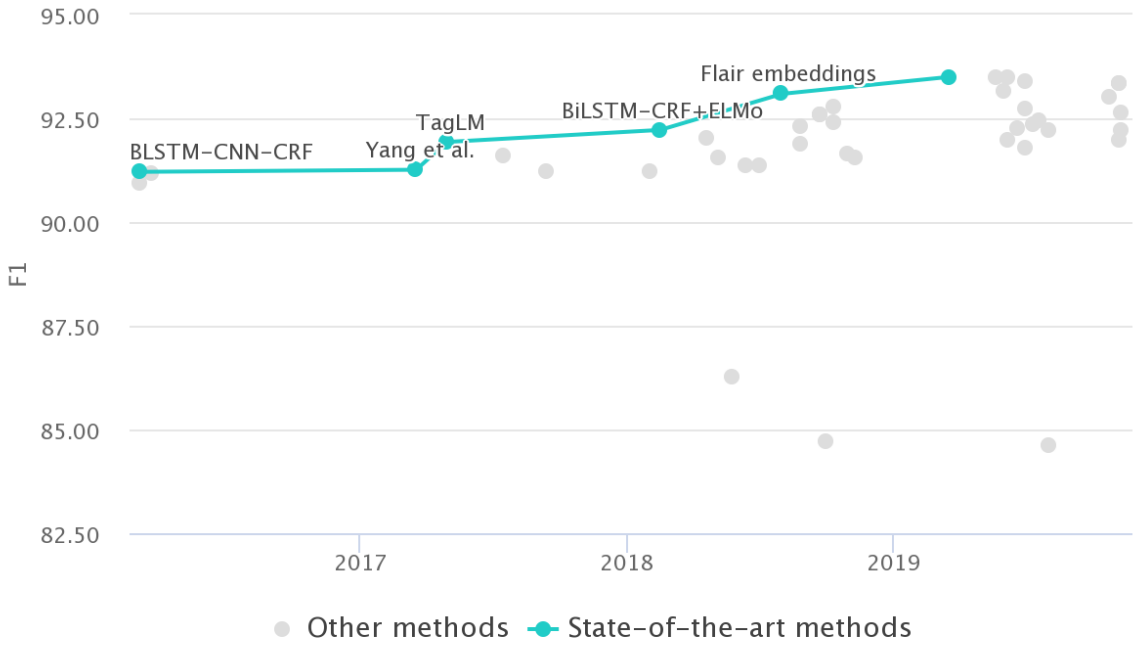
Pierwsze rozwiązania problemu automatycznego rozpoznawania nazw własnych wykorzystywały ręcznie sformułowane reguły.[6]. Reguły te były tworzone zarówno na podstawie wiedzy lingwistycznej jak i dziedzinowej. Osiągały zwykle dużą precyzję, natomiast nie rozpoznawały nazw pojawiających się w innych, niż opisane regułami, kontekstach. Obecnie systemy regułowe zostały w większości zastąpione poprzez nauczanie nadzorowane z modelami trenowanymi na korpusach. Jednak w przypadku specyficznej kategoryzacji i braku zaanotowanych zbiorów systemy regułowe nadal tworzą dobrą alternatywę do modeli uczenia maszynowego.[5].

2.3.2 Metody słownikowe

Systemy bazujące na słownikach, zwykle połączone z małą liczbą reguł, także były popularnym wyborem przy tworzeniu wczesnych wersji systemów rozpoznawania nazw własnych. Słowniki zawierają informacje o świecie zewnętrznym takie jak lista miejscowości, imion i nazwisk sławnych osób czy organizacji. Dzięki nim popularne frazy na pewno zostaną poprawnie oznaczone.
Najczęstszym sposobem tworzenia słowników jest ręczne ich zapisywanie. Jednak wymaga to dużego nakładu pracy eksperckiej. Dlatego dodatkowym sposobem jest także próba automatycznego ich zbierania za pomocą zewnętrznych korpusów lub innych źródeł.
W zależności od specyfiki zadania, informacji ze słowników można także użyć jako jednej z cech wejściowych do systemów uczenia maszynowego.[7]

2.3.3 Uczenie maszynowe

Najczęściej używaną techniką w zadaniu z CONLL-2003 [1] był model bazujący na maksymalnej entropii, a pięć kolejnych systemów także korzystało z modeli statystycznego uczenia.
W ostatnich latach większość systemów rozpoznających nazwy własne korzysta z uczenia maszynowego, a w szczególności z sieci neuronowych. Jest wiele możliwych podejść, które przewidują frazy z dużą pewnością. W przypadku małej liczby zaanotowanych dokumentów można wykorzystać częściowo nadzorowane uczenie (semi-supervised learning), dzięki czemu unikamy kosztownego ręcznego anotowania dokumentów.
Natomiast w większości przypadków wykorzystuje się systemy nadzorowanego uczenia. W tym wypadku rozpoznawanie nazw własnych sprowadza się do problemu klasyfikacji. System oznacza po kolei każde słowo, czy zostało zaklasyfikowane jako część nazwy własnej.
Obecnie najwyższe rezultaty dla korpusu CONLL-2003 osiąga sieć neuronowa o architekturze BiLSTM-CNN . Niedaleko za nią znajdują się wariacje architektur bazujących na BiLSTM (jak na przykład model Bi-LSTM-CRF [17]) lub transformerach [24].



Rysunek 2.2: Wykres SOTA dla zbioru CONLL-2003 (stan na 01.06.2020)
Źródło https://paperswithcode.com/sota/named-entity-recognition-ner-on-conll-2003

RANK	METHOD	F1	EXTRA TRAINING DATA	PAPER	CODE	RESULT	YEAR
1	CNN Large + fine-tune	93.5	✓	Cloze-driven Pretraining of Self-attention Networks			2019
2	GCDT + BERT-L	93.47	✓	GCDT: A Global Context Enhanced Deep Transition Architecture for Sequence Labeling			2019
3	I-DARTS + Flair	93.47	✓	Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition			2019
4	LSTM-CRF+ELMo+BERT+Flair	93.38	✓	Neural Architectures for Nested NER through Linearization			2019
5	Hierarchical + BERT	93.37	×	Hierarchical Contextualized Representation for Named Entity Recognition			2019

Rysunek 2.3: Wyniki SOTA dla zbioru CONLL-2003 (stan na 01.06.2020)
Źródło https://paperswithcode.com/sota/named-entity-recognition-ner-on-conll-2003

Rozdział 3

Dane

Dane, które zostały wykorzystane w tej pracy pochodzą z dwóch zbiorów z różnych źródeł:

- zbiór treningowy - podzbiór z **Narodowego Korpusu języka Polskiego** [8]
- zbiór testowy - pochodzący z konkursu **Poleval 2018** [23]

3.1 Kategorie

Każda nazwa własna wyróżniona w tych korpusach należy do jednej z kategorii opisanych poniżej:

- **persName** - do tej grupy należą nazwy indywidualnych osób i rodzin; w tej kategorii wyróżnione są następujące podkategorie:
  - **forename** – imię, pojedyncze, podwójne lub w liczbie mnogiej: Marcin, Jan Maria;
  - **surname** – nazwisko: Kowalscy, Washington;
  - **addName** – pseudonim, przydomek, dynastia, dodatkowy epitet: Rudy, Lwie Serce, Groźny, Jagiellonowie;
- **orgName** - nazwy różnych organizacji, instytucji międzynarodowych i państwowych, firm, stowarzyszeń: Amazon, Ministerstwo Pracy;
- **geogName** - obiekty geograficzne mających cechy fizyczne wyróżniające je na terytorium. Takie jak regiony geograficzne, wyspy, morza, lasy: Mazury, Bałtyk;
- **date** - daty kalendarzowe: 1 października 1999, XXI wiek;
- **time** - czas w postaci godzin, minut i ewentualnie sekund : ósma trzydzieści;
- **placeName** - nazwy geopolityczne lub nazwy obiektów wynikające z podziałów administracyjnych;

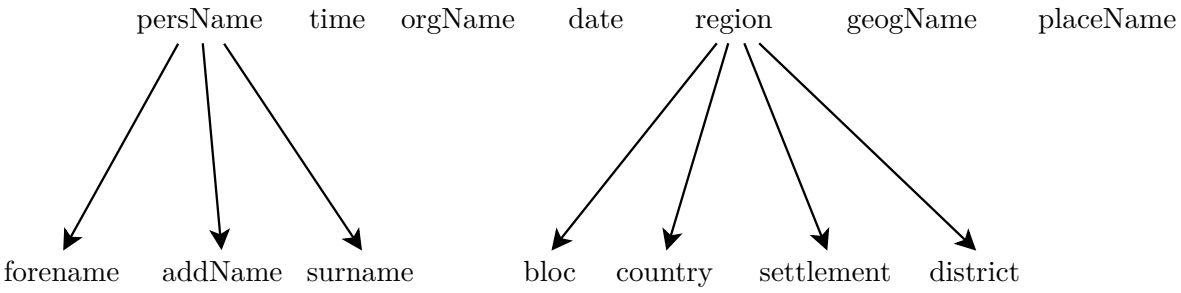
– **district** - jednostka podziału administracyjnego miasta lub innej osady. Takie jak osiedle, dzielnica: Gmina Bielany, Żoliborz;

– **settlement** - miasto, wioska lub osada: Katowice, Piaski;

– **region** - jednostka podziału administracyjnego większa niż miasto, ale mniejsza niż państwo, np. województwo, stan: województwo mazowieckie, Alabama;

– **country** - państwo, kraj lub kolonia : Polska, RPA;

– **bloc** - jednostka geopolityczna obejmująca dwa lub więcej państw: Unia Europejska, NATO;



Rysunek 3.1: Hierarchia kategorii nazw własnych w podkorpusie milionowym

3.2 Podkorpus milionowy

W najbardziej popularnych zbiorach, CONLL-2003 [1] czy Muc-7 [4], nazwy własne zostały zaanotowane w najprostszy z możliwych sposobów. Powyższe zbiory są przeznaczone do wykorzystania dla automatycznej anotacji, a każdy element dodający złożoność do zadania mocno komplikuje tworzenie dla niego systemu. Natomiast powoduje to utratę pewnych informacji.
Innym podejściem charakteryzują się natomiast zbiory użyte w tej pracy. Zostały tu wykorzystane różne reguły anotacji, które zapewniają zachowanie jak największej ilości informacji.
Podkorpus milionowy został utworzony jako podzbiór z Narodowego Korpusu języka Polskiego, NKJP, [8], w którym wszystkie oznaczenia nazw własnych, pierwotnie zidentyfikowane za pomocą zestawu reguł, zostały poddane ręcznej weryfikacji. Podkorpus ten składa się on z niewielkich próbek różnych tekstów (opis w tabeli 3.1) i posiada ponad milion segmentów (słów, znaków interpunkcyjnych).

Na potrzeby budowy NKJP opracowano dedykowane procedury anotacji, uwzględniające wiele nietypowych sytuacji. Dokładny opis zasad przygotowania korpusu znajduje się w

Typ	Procentowy udział
Dzienniki	25,5%
Pozostałe periodyki	23,5%
Książki publicystyczne	1,0%
Literatura piękna	16,0%
Literatura faktu	5,5%
Typ informacyjno-poradnikowy	5,5%
Typ naukowo-dydaktyczny	2,0%
Internetowe interaktywne (blogi, fora, Usenet)	3,5%
Internetowe nieinteraktywne (statyczne strony, Wikipedia)	3,5%
Quasi-mówione (protokoły sesji parlamentu)	2,5%
Mówione medialne	2,5%
Mówione konwersacyjne	5,0%
Inne teksty pisane	3,0%
Książka niebeletrystyczna nieklasyfikowana	1,0%

Tablica 3.1: Procentowy skład źródeł tekstów w podkorpusie milionowym

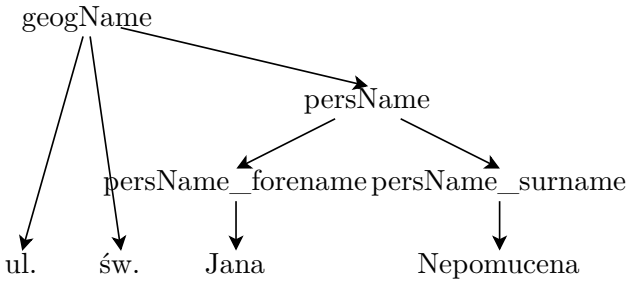
oficjalnej dokumentacji [8]. W niniejszej pracy skupimy się na następujących cechach przyjętego zbioru etykiet i typ anotowanych fraz:

1. **Hierarchiczność kategorii**: Hierarchia nazw własnych jest utworzona z dwóch poziomów i składa się 14 kategorii (Fig 3.1).

2. **Frazy zagnieżdżone**: Kategorie mogą na siebie nachodzić tworząc dwie osobne nazwy własne posiadające wspólne słowa.(Fig 3.2) .

Hierarchia

W anotacji z hierarchią nazwa własna może przyjmować kategorię nadrzędną lub podrzędną. Jeżeli złożenie nazw własnych z kategorii podrzędnych tworzy kategorię nadrzędną, to obie zostaną oznaczone. W przykładzie, cała fraza: *Maria Nowak*, będzie oznaczona jako persName, a także *Maria*, jako *forename* i *Nowak* jako *surname*.
W sytuacji gdy część nazwy własnej, która posiada kategorią nadrzędną, nie pasuje do żadnej kategorii podrzędnej to nie zostaje ona nie skategoryzowana. W przykładzie *król Maciuś Pierwszy*, cała fraza zostanie oznaczona jako osoba, *Maciuś* - imię, *Pierwszy* - addName, a słowo *król* będzie oznaczone wyłącznie kategorią *persName*.
Może też wystąpić sytuacja, w której nazwa własna została oznaczona kategorią podrzędną, a nie jest oznaczona częścią anotacji nadrzędnej. Taka sytuacja ma przykładowo miejsce , jeśli w tekście pojawi się tylko imię.



Rysunek 3.2: Fraza ul. św. Jana Nepomucena jako zaklasyfikowana lokalizacja. Jan Nepomucen jest zagnieżdżoną frazą oznaczoną jako osoba

Zagnieżdżenia

Nazwy własne w podkorpusie milionowym zostały zaanotowane w takim formacie, że każda fraza tworząca nazwę własną jest osobnym elementem. Konwencja ta różni się od takich zbiorów jak CONLL-2003 [1], gdzie anotowane zostały tylko frazy o maksymalnej długości. Oznacza to, że cała fraza "Liceum Ogólnokształcące Juliusza Słowackiego" zostanie oznaczona jako ORG, natomiast jej część opisująca osobę "Juliusza Słowackiego" zostanie zignorowana. Natomiast w korpusie NKJP, "Liceum Ogólnokształcące Juliusza Słowackiego" zostanie oznaczona jako org, a "Juliusz Słowacki" jako osoba. Dzięki takiej strukturze korpus:

- posiada dużą gęstość i różnorodność nazw własnych,
- ułatwia analizę powiązań pomiędzy frazami,
- ułatwia oznaczanie typów jednostek nazewniczych

Zagnieżdżone nazwy własne nie są popularnym elementem korpusów w językach naturalnych, głównie z powodów praktycznych [9]. Jednak niektóre zbiory, szczególnie posiadające dane biomedyczne, zostały stworzone ze zwróceniem szczególnej uwagi na ten element. Na przykład w korpusie GENIA aż 17% fraz jest zawartych w innych frazach. Natomiast w zbiorze ACE 30% zdań zawiera zagnieżdżone nazwy [10]. GENIA jest to korpus utworzony w celu ekstrakcji informacji biomedycznych z czasopism naukowych. Natomiast korpus utworzony przez program ACE (Automatic Content Extraction) pochodzi z tekstów dziennikarskich i posiada kategorie takie jak Person, Location i Organization. W przypadku podkorpusu milionowego NKJP, **3599** słów należy do zagnieżdżonych nazw własnych. Stanowią one około 4.1% wszystkich nazw własnych w całym korpusie.

3.3 Korpus testowy

Korpus testowy pochodzi z konkursu Poleval 2018 [23] i składa się z 1828 dokumentów pobranych losowo z korpusu NKJP. Anotacja tego zbioru odbyła się zgodnie z zasadami anotacji podkorpusu milionowego. Dane tekstowe bez anotacji zostały udostępnione 6 sierpnia 2018r. Natomiast anotacje zbioru testowego wraz ze skryptem ewaluującym 22 sierpnia 2018 i są publicznie dostępne[1].

[1]Dostępne na stronie: http://mozart.ipipan.waw.pl/~axw/poleval2018/

Rozdział 4

Przetworzenie danych tekstowych

Nieprzetworzone dokumenty przechowywane są w formie tekstu jako ciąg znaków. Jednak modele uczenia maszynowego nie obsługują takich danych. Konieczne jest ich przetworzenie do formatu wektorów liczb, jednocześnie maksymalizując ilość informacji przekazanych do modelu.
W poniższym rozdziale znajduje się opis kroków przetwarzania tekstu wykorzystanych w tym zadaniu.

4.1 Podział na zdania

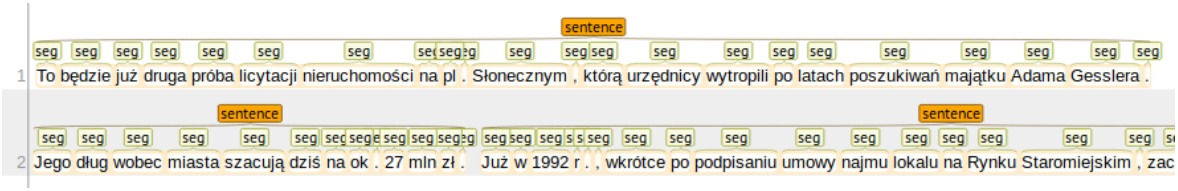
Zdanie rozumiane jest tradycyjnie jako ciąg od wielkiej litery do znaku kończącego (kropka, wielokropek, znak zapytania, wykrzyknik). Natomiast w zależności od źródła danych struktura tekstu może się nieco różnić. W danych pochodzących przykładowo z komunikatorów internetowych bardzo często trudno wyróżnić kompletne zdania i trzeba poprzestać na analizie fraz. Ponadto interpunkcja jest często pomijana, zatem podział na zdania musi się opierać na innych przesłankach.

4.2 Segmentacja

Kolejnym elementem przygotowania danych jest ich segmentacja. Segmentami nazywamy sekwencję znaków w zdaniu, które potencjalnie odpowiadają słowom języka, liczbom czy znakom interpunkcyjnym. Zwykle są one nie dłuższe niż fragmenty oddzielone białymi znakami lub znakami interpunkcyjnymi, tak jak na rys 4.1. W zależności od decyzji anotacyjnych, segmentami mogą być też fragmenty krótsze np:

- formy aglutacyjne leksemu “być” np. <długo>, <śmy>
- partykuły -by, -że, -li np. <napisała>, <by>, <m>

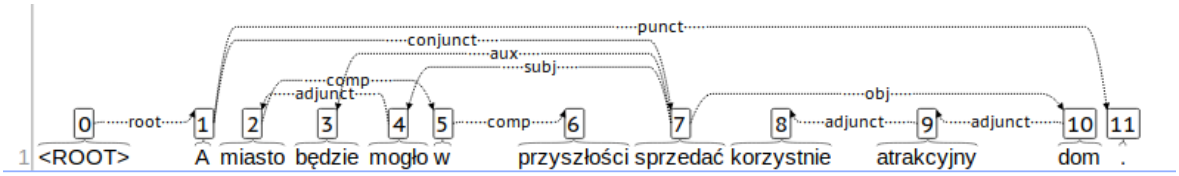
- niektóre słowa zawierające łącznik np. <polsko><-><niemiecki>



Rysunek 4.1: Przykład segmentacji zdania
Źródło http://multiservice.nlp.ipipan.waw.pl/

4.3 Analiza zależnościowa

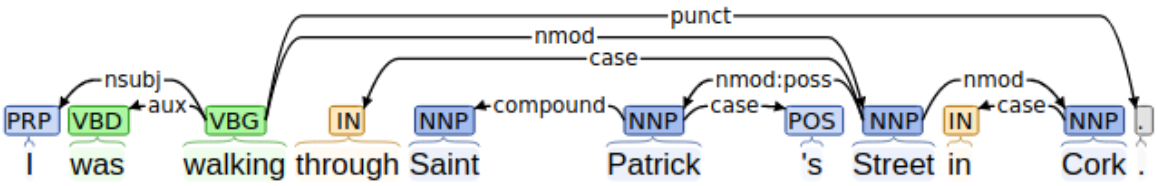
Analiza zależnościowa jest jednym z kolejnych kroków analizy języka naturalnego. Jej celem jest wyznaczenie relacji pomiędzy wyrazami w zdaniu. Przykładowe relacje to bycie podmiotem, dopełnieniem lub modyfikatorem.
Jako podstawę do przeprowadzenia analizy zależnościowej wykorzystuje się wynik tagowania. Tagowanie jest to ujednoznacznianie wyniku analizy morfosyntaktycznej, która ma na celu wyznaczenie wszystkich możliwych wartości cech morfologicznych i części mowy wszystkich segmentów.
Więcej informacji jak i sposób przeprowadzania tagowania można znaleźć na przykład w rozdziale 3.6 wydania "Narodowego Korpusu Języka Polskiego" [8].



Rysunek 4.2: Przykład drzewa zależności
Źródło http://multiservice.nlp.ipipan.waw.pl/

W wyniku analizy zależnościowej powstaje (zwykle) drzewo, w którym każdy wierzchołek ma jedną krawędź wejściową prowadzącą od elementu nadrzędnego. Wierzchołki odpowiadają segmentom w zdaniu, Rys: 4.2.
Dokładniejszy opis jak przeprowadzane jest parsowanie zależnościowe można znaleźć na przykład w rozdziale piętnastym wydania "Speech and Language Processing" autorstwa Dana Jurafsky'ego i Jamesa H. Martina. [25]
Nazwy własne stanowią zwykle kompletne frazy. Rozpoznając granice nazw własnych można więc wykorzystać informacje pochodzące z analizy zależności składniowych. Przykładowo, w

zdaniu (Rys: 4.3) “ I was walking through Saint Patrick’s Street in Cork.” (“Szłam po ulicy św. Patryka w Cork.”), można wyróżnić parę nazw własnych. Imię - “Patryk”, osobę - “św. Patryk”, miejsce - “ulica św. Patryka” i miasto “Cork”. Jak można zauważyć cała nazwa “ Saint Patrick’s Street” znajduje się wewnątrz jednego poddrzewa w zdaniu, a w osoba jest częścią poddrzewa ulicy.
Dzięki takim informacjom systemy uczące się mogą ograniczyć błędy wynikające z przypisania słów spoza takiego poddrzewa do frazy, przykładowo nazwy miasta do nazwy ulicy.



Rysunek 4.3: Parsowanie zależnościowe przykładowego zdania z nazwami własnymi
Źródło: https://corenlp.run/

4.4 Reprezentacja słów w przestrzeni wektorowej

Po podziale na zdania i segmentacji otrzymujemy sekwencje słów jak w przykładzie 4.1. Aby dane te mogły być użyte w programach maszynowego uczenia się, czy to statystycznych, czy w sieciach neuronowych, muszą zostać przetworzone. Etykiety symboliczne są zwykle zamieniane na reprezentację typu one-vs-all, w której liczba etykiet równa jest liczbie cech o wartościach równych 1 tylko wtedy gdy ta etykieta powinna być przypisana konkretnej obserwacji. W przypadku słów języka naturalnego, które nie są od siebie niezależne, udało się jednak wypracować znacznie efektywniejsze sposoby reprezentacji, które zostaną omówione poniżej.

	seg 1	seg 2	seg 3	seg 4	seg 5
Zdanie 1	To	jest	kot	.	
Zdanie 2	Mały	kot	głośno	miauczy	.

Tablica 4.1: Przykład podzielonych zdań na segmenty.

Najprostszym sposobem reprezentacji słów jest wykorzystanie słownika, w którym każdy segment jest jednym jego elementem. Tak więc w przykładzie z tabeli 4.2 możemy zapisać <kot> znajdować się pod indeksem 3.

Następnie pojawienie się takiego słowa zaznaczamy jako numer 1 na indeksie 3, w wektorze o długości słownika, której reszta elementów jest wypełniona zerami. Wektor taki jest nazywany 1 z n (one-hot) i wygląda jak na przykładzie z tabeli 4.3. W tabeli 4.4 pokazana

	seg 1	seg 2	seg 3	seg 4	seg 5
Zdanie 1	1	2	3	4	
Zdanie 2	5	3	6	7	4

Tablica 4.2: Przykład podzielonych zdań na segmenty z wykorzystaniem indeksów słownika.

jest suma wektorów z całego zdania nazywamy workiem słów (bag of words). Worek słów był początkowo dość popularnym sposobem reprezentacji zdania na wejściu systemów uczących się.

	Mały	kot	głośno	miauczy	.
1 – To	0	0	0	0	0
2 – jest	0	0	0	0	0
3 – kot	0	1	0	0	0
4 - .	0	0	0	0	1
5 – Mały	1	0	0	0	0
6 – głośno	0	0	1	0	0
7 – miauczy	0	0	0	1	0

Tablica 4.3: Przykład sekwencji wektorów jeden z 1

	1 – To	2 – jest	3 – kot	4 - .	5 – Mały	6 – głośno	7 – miauczy
To jest kot.	1	1	1	1	0	0	0
Mały kot głośno miauczy.	0	0	1	1	1	1	1

Tablica 4.4: Przykład reprezentacji workiem słów.

Jednak taka reprezentacja ma dwie zasadnicze wady:

- nie zawiera zależności semantycznych między słowami (słowa kot i kiciuś, choć są bardzo bliskie znaczeniem, mają indeksy od siebie niezależne
- jest ona bardzo złożona pamięciowo. Zwykle słowniki do przetwarzania języka naturalnego zawierają minimum 50 tysięcy słów. W przypadku, gdy długość zdania wynosi 50 słów, to na jedno zdanie musimy zaalokować $5000 \cdot 50 = 250000$ liczb.

Aby uniknąć powyższych problemów, obecnie większość systemów wykorzystuje zanurzenia słów (word embeddings). Jest to reprezentacja słowa jako wektora liczb rzeczywistych w N wymiarowej przestrzeni, gdzie N jest zwykle znacznie mniejsze niż 1000. Zaletą takiej reprezentacji jest relatywnie mała szerokość wejścia w stosunku do innych metod. Uzyskujemy tu średnio wejście o dwa rzędy wielkości mniejsze od reprezentacji

pojedynczo słów w całym słowniku. W zdaniu o 50 słowach i wektorach 300 wymiarowych potrzeba do alokacji tylko 15 000 liczb.
Jest kilka sposobów na utworzenie takiej reprezentacji. Może być to produkt uboczny zadania nadzorowanego, takiego jak klasyfikacja dokumentów. Innym podejściem jest statystyczne grupowanie słów wykorzystywanych w tym samym otoczeniu, czego przykładem są wektory Glove [19]. W tym zadaniu zostały natomiast wykorzystane 300 wymiarowe wektory fasttext dla języka polskiego, które tworzone są w trakcie trenowania sieci neuronowej w zadaniu przewidywania słów w tekście. [13]

Wektory FastText zostały wytrenowane dla 157 języków dokumentów z dwóch rodzajów źródeł dokumentów:

- Wikipedii - największej internetowej encyklopedii, która zawiera artykuły w ponad 200 językach. Struktura wysokiej jakości dokumentów jest tam dość silnie określona, a teksty są w większości poprawnie gramatyczne i stylistyczne. Zbiór, na którym wytrenowane zostały wektory słów dla języka polskiego, posiada 386,874,622 tokenów i 1,298,250 słów.

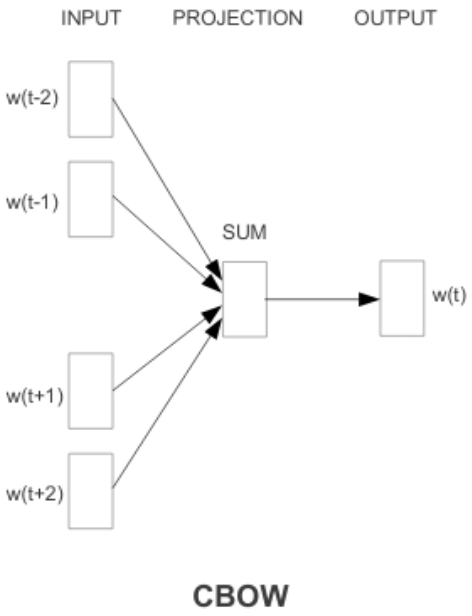
- Common Crawl - dane zbierane przez organizację non-profit o tej samej nazwie pochodzą ze ściągniętych stron internetowych. Dane te zostały udostępnione publicznie i są w formacie czystego HTML lub też plików tekstowych. Teksty wykorzystane do utworzenia wektorów pochodzą z Maja 2017 roku. Zbiór ten posiada 21,859,939,298 tokenów i 10,209,556 słów, więc jest znacząco większy od Wikpiedii. Co więcej, jakkolwiek zbiór Wikipedii jest zadowalający dla najbardziej popularnych języków (angielski, niemiecki itp.), tak dla mniej popularnych języków jest o wiele mniejszy, co znacząco utrudnia tworzenie reprezentacji słów.

Przy tworzeniu reprezentacji słów wykorzystywane są dwa modele:

- CBOW (continuous bag of words - ciągły worek słów) - model ten jest rozszerzeniem modelu z 2013.4.4
  Model CBOW przewiduje słowo bazując na kontekście wokół niego. To znaczy, celem systemu jest przewidzenie słowa w_i, jeśli na wejściu znajduje się jego kontekst zapisany jako $w_0, ..., w_{i-1}, w_{i+1}, ..., wn$. Zakładając że kontekst jest wielkości 2 to w przykładzie "Szybki pies goni małego kota", model przyjmuje więc na wejściu indeksy słów "szybki", "pies", "małego", "kota", a na wyjściu oczekiwaną wartością będzie indeks słowa "goni" zapisany za pomocą wektora 1 z n.
  W 2017r. do modelu zostały dodane dwa rozszerzenia:

  – przekazywanie informacji o częściach słów (na przykład sylabach) poprzez wykorzystanie n-gramów znakowych,

– wagi pozycyjne pozwalające na efektywną analizę informacji o pozycji słowa w zdaniu.

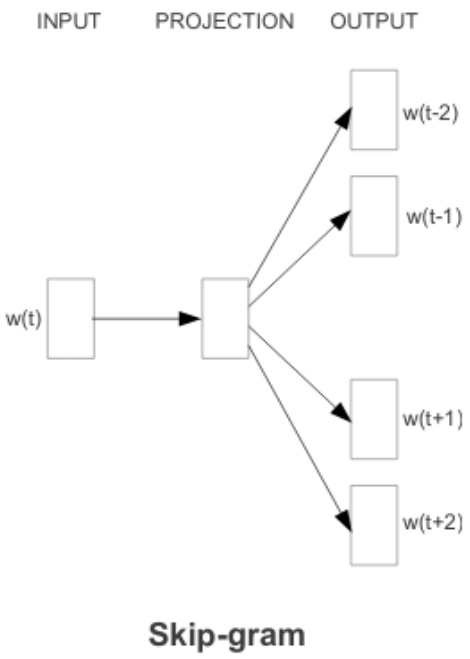


Rysunek 4.4: Model CBOW
Źródło: https://arxiv.org/pdf/1301.3781.pdf(Mikolov et. al. [13])

- skipgram - model ten jest rozszerzeniem modelu z 2013 roku.4.5 Przewiduje on kontekst bazując na słowie wejściowym. Jest to odwrotność algorytmu CBOW.
  W zdaniu "Szybki pies goni małego kota" model skipgram na wejściu będzie posiadać indeks słownika oznaczający "goni", natomiast oczekiwanymi wartościami będą indeksy słów "szybki", "pies", "małego", "kota". W modelu znajduje się jedna warstwa ukryta, która na którą nakładany jest softmax, który normalizuje wartości przewidywane.
  Do wyznaczenia zanurzeń zostało wykorzystane rozwinięcie o wykorzystanie ngramów znakowych. Reprezentacja wektorowa słowa jest utworzona poprzez poprzez sumę wektorów ze znakowych ngramów, które znajdują się w danym słowie.

Tak utworzone wektory posiadają liczne ciekawe właściwości, których nie udałoby się osiągnąć przy innych formach reprezentacji słów. Na przykład zanurzenia o podobnym znaczeniu i zastosowaniu tworzą klastry.4.6 Inną cechą jest reprezentacja rzeczywistych zależności za pomocą różnic w pozycjach słów.

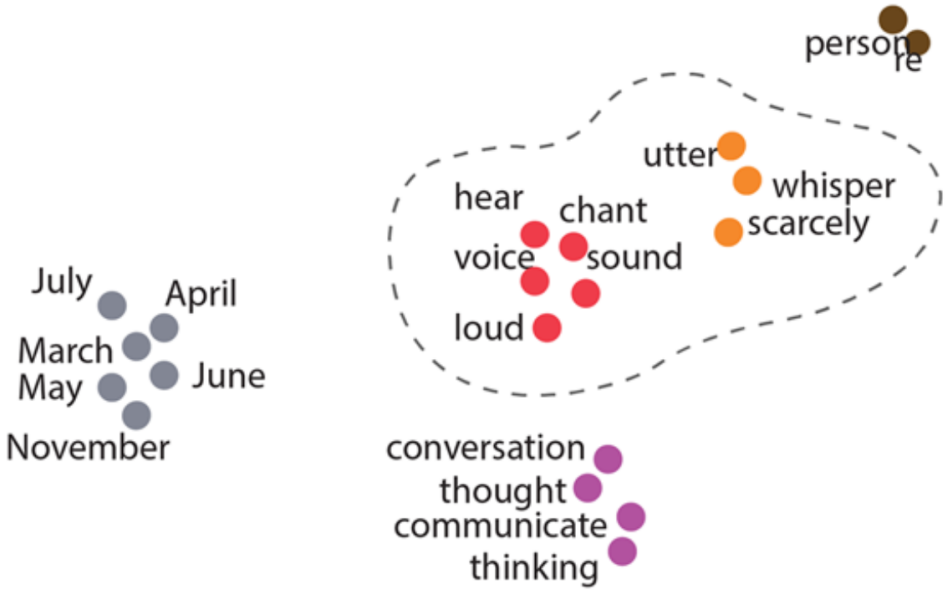
Najbardziej popularnym przykładem ilustrującym własności wektorowych zanurzeń słów jest różnica między dwoma wektorami zgodnie z relacją: *"Mężczyzna ma się tak do króla jak kobieta do królowej."*



Rysunek 4.5: Model skipgram
Źródło: https://arxiv.org/pdf/1301.3781.pdf(Mikolov et. al. [13])

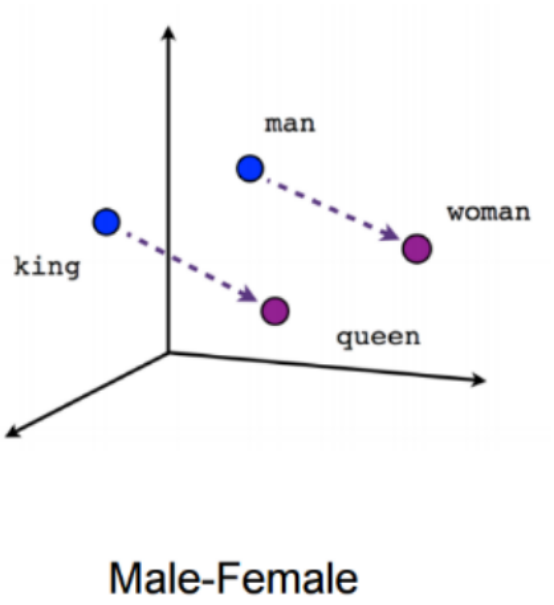
Oznacza to, że jeśli pomiędzy dwoma słowami x1,y1 zachodzi relacja X oraz jeśli pomiędzy słowami x2,y2 zachodzi relacja X' wtedy relacja X' jest podobna do X ($x1 - y1 \approx x2 - y2$)
I tak w powyższym przykładzie, jeżeli oznaczymy następująco dane słowa:
x1=mężczyzna, y1=król
x2=kobieta, y2=królowa
Oznacza to, że relacja mężczyzna-król jest podobna do relacji kobieta królowa i będzie reprezentowana podobnymi wektorami. (Rys.4.7)

W zależności od rodzaju danych, uzyskane z porównywania wektorów wyniki mogą być czasem zaskakujące, a czasem odkrywać jakąś wiedzę o konkretnych użytkownikach języka. Przykładowo, w modelu dla języka angielskiego okazało się, że relacja zachodząca między angielskimi słowami "mężczyzna" i "lekarz", dla słowa "kobieta" prowadzi do pielęgniarka. Warto jednak zwrócić uwagę, iż jest to tylko uproszczony opis niektórych właściwości. W przestrzeni o kilkuset wymiarach relacje te są bardziej skomplikowane. W artykule "Fair is Better than Sensational: Man is to Doctor as Woman is to Doctor"[20] ukazane jest, jak często taka relacja jest myląca.
Jeżeli usunięte zostaną ograniczenia nałożone na wektory słów, to w popularnym przykładzie *Mężczyzna jest podobny do lekarza, jak kobieta do pielęgniarki*, kobieta w tym przypadku jest bliższa do doktora niż pielęgniarki. Dlatego też, jakkolwiek pokazane wyżej właściwości są interesujące, to w związku z podanymi powyżej wątpliwościami nie należy opierać na nich



Rysunek 4.6: Wektory słów rzutowane na przestrzeń dwuwymiarową
Źródło: A machine learning approach to predicting psychosis using semantic density and latent content analysis

całego rozwiązania. Jednak wektory słów sprawdzają się bardzo dobrze jako element wejścia do systemu nauczania maszynowego, który zadecyduje o wartości podanej informacji.



Rysunek 4.7: Zależności między wektorami rzutowanymi na przestrzeń trójwymiarową
Źródło: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

Rozdział 5

Modele uczenia maszynowego

5.1 Sieć neuronowa

Sieci neuronowe są algorytmami uczenia maszynowego inspirowanymi przez sposób uczenia się mózgu. Ich celem jest aproksymacja pewnej funkcji f* przekształcającej wejście X na wyjście Y.
Na przykład wejściem X może być zbiór pikseli z obrazka, a wyjściem wartość określająca czy ten zbiór pikseli pokazuje kota lub psa.

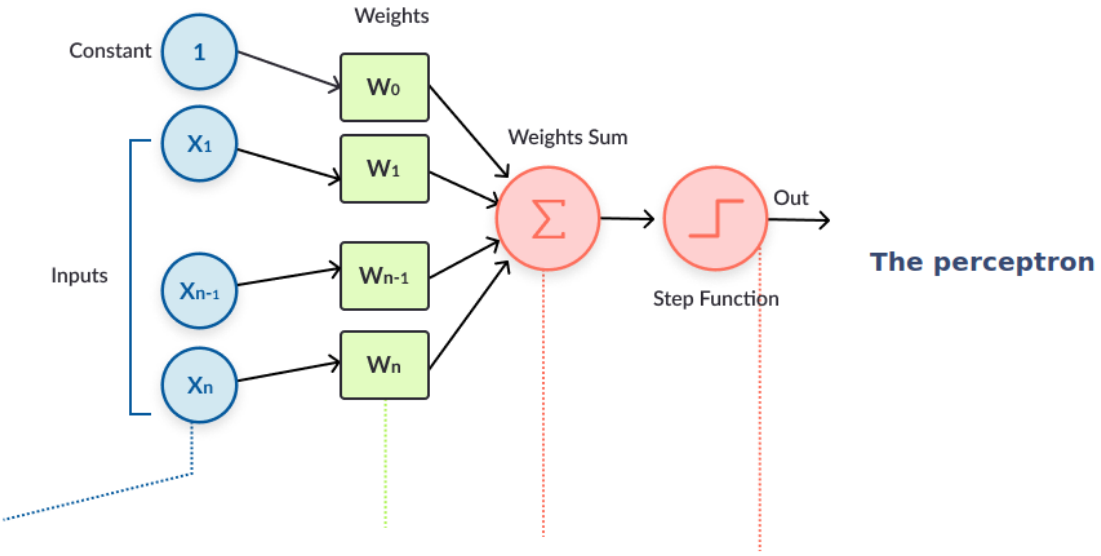
Z pojęciem sieci neuronowej wiążę się perceptron - sztuczny neuron, będący podstawową strukturą w sieci. Perceptron pobiera na wejściu sekwencję liczb x1,x2...xn, i mnożąc je poprzez wagi neuronów tworzy na wyjściu pojedynczą wartość y. Najprostszy model można zapisać wzorem:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{jeśli} \sum_{i=1}^{m} w_i x_i, + b > 0, \\ 0 & \text{w p.p.} \end{cases} \tag{5.1}$$

Funkcja aktywacji Do zmodyfikowania wyjścia perceptronu stosuje się funkcję aktywacji. Dzięki zastosowaniu takiej opcji model zyskuje takie cechy jak:

- **kontrola zakresu**: zakresy funkcji aktywacji przeciwdziałają wzrostowi wartości ku nieskończoności, co blokowałoby możliwość nauki sieci.
- **ciągłość**: umożliwiającą uczenie na podstawie spadku gradientu

Sieci wielowarstwowe Sieci wielowarstwowe (głębokie sieci jednokierunkowe) są najbardziej podstawowymi modelami głębokiego uczenia. Są one reprezentowane poprzez złożenie wielu funkcji skierowanym grafem acyklicznym, opisującym jak funkcje są ze sobą



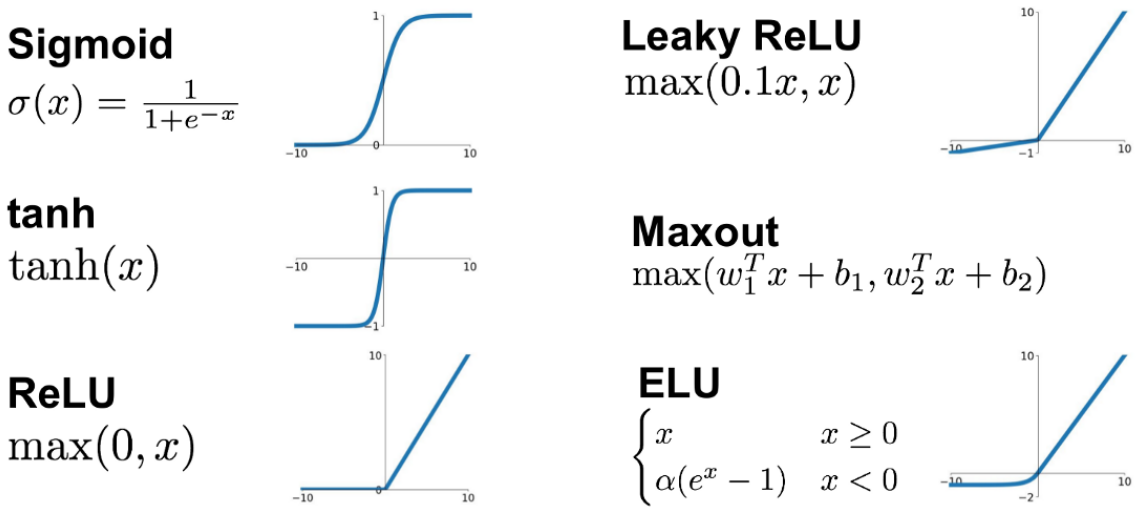
Rysunek 5.1: Perceptron

Źródło: https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/

powiązane. Funkcje te nazywamy warstwami modelu, a całkowita długość łańcucha opisuje głębokość modelu. Sieci wielowarstwowe złożone są z warstwy wejściowej, wyjściowej i warstw ukrytych.

Optymalizacja Aby sieć mogła poprawnie przewidywać wyjście Y, konieczne jest jej wytrenowanie. Podczas uczenia zmieniane są wagi wewnątrz sieci, tak aby wyjście przewidywane ($\hat{Y}$), było jak najbardziej podobne do rzeczywistego (Y). Najbardziej popularnym algorytm uczenia sieci neuronowych jest metoda stochastycznego spadku wzdłuż gradientu.
W metodzie gradientu prostego dla każdego przypadku w zbiorze treningowym przeprowadzane jest wyliczenie jego błędu. Na początku następuje aktywacja neuronów, która produkuje pewne wyjście $\hat{y}$. To wyjście $\hat{y}$ porównywane jest do wartości wzorcowej y za pomocą funkcji kosztu. Następnie wynik tej funkcji po wszystkich elementach wejściowych, jest propagowany wstecz na całą sieć. Zmiany są wyliczane po kolei dla każdej warstwy, a wagi są modyfikowane się zgodnie z wartością jaką wpływały na błąd. Algorytm ten nazywa się propagacją wsteczną.

Przy spadku gradientowym przetwarzane jest pełne wejście w każdej iteracji. Jest to rozwiązanie zapewniające maksymalną skuteczność, natomiast bardzo czasochłonne. Dlatego też w rzeczywistości korzysta się ze stochastycznego spadku wzdłuż gradientu. W tej metodzie w jednej iteracji na raz brany jest pod uwagę pewien zbiór przypadków, który nazywany nazwany jest partią (batch). Jednak w tym wypadku poszukiwanie minimum może być



Rysunek 5.2: Popularne funkcje aktywacji
Źródło: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

zaszumione i uczenie niektórych partii prowadzi do powiększenia błędu ogólnego, zamiast go pomniejszać.

Z procesem uczenia się sieci neuronowej powiązane są też następujące parametry:

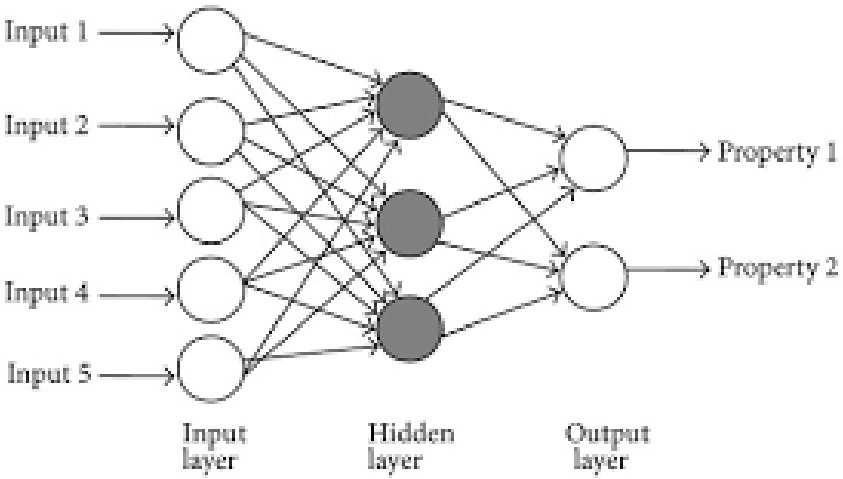
- tempo uczenia się (learning rate) - wartość określająca wielkość zmian w każdej partii,
- zanik tempa uczenia (Learning Rate Decay) - definiowany w celu zmniejszenia tempo uczenia po każdej epoce. Zmiana tempa uczenia się pozwala dokonywać większych modyfikacji na początku procesu, a potem zmian coraz precyzyjniejszych zmian w miarę zbliżania sie do rozwiązania. Zmniejsza to czas dotarcia do minimum (lokalnego).

5.2 Rekurencyjne sieci neuronowe

Do przetwarzania danych sekwencyjnych, takich jak tekst czy sygnały wykorzystujemy rekurencyjne sieci neuronowe. Takie sieci otrzymują sekwencje na wejściu, posiadają rekurencyjne połączenia między jednostkami ukrytymi i mogą generować sekwencyjne wyniki. RNN otrzymuje wartości $x^{(t)}$ na wejściu, gdzie x jest zestawem cech dla danego kroku, a t oznacza krok w sekwencji z zakresu 1 do Γ. Równanie na ukryty stan t możemy zapisać następująco

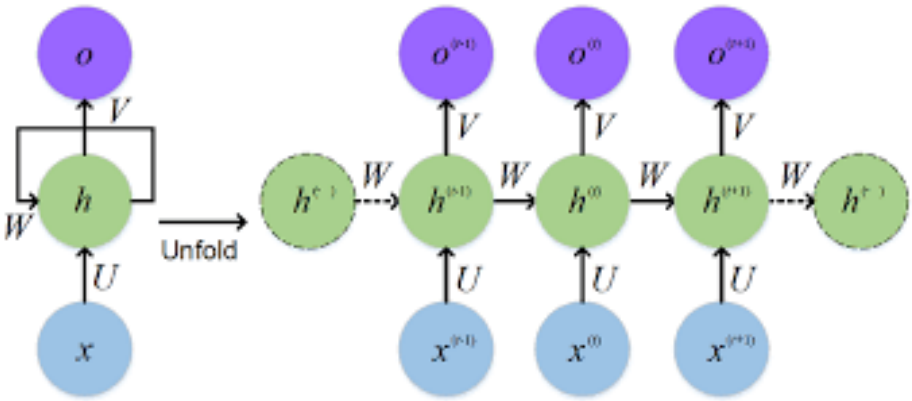
$$h^{(t)} = f(h^{(t-1)}, x_t)$$

LSTM Zwykłe sieci rekurencyjne dobrze przewidują krótkie sekwencje, natomiast mają problem z utrzymaniem informacji pomiędzy jednostkami daleko oddzielonymi w czasie.
W teorii zwykła sieć rekurencyjna jest w stanie zapamiętać dowolnie długie zależności



Rysunek 5.3: Perceptron wielowarstwowy

Źródło: https://medium.com/pankajmathur/a-simple-multilayer-perceptron-with-tensorflow-3effe7bf3466



Rysunek 5.4: Rekurencyjne sieci neuronowe

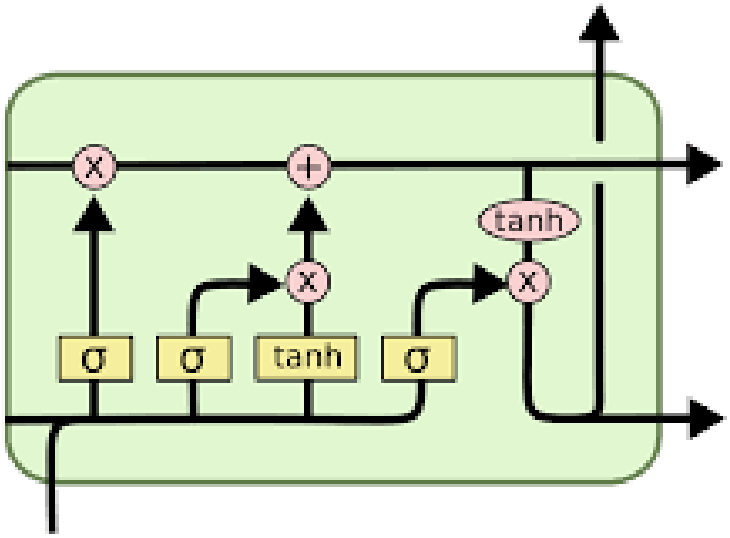
Źródło: Audio visual speech recognition with multimodal recurrent neural networks

pochodzące z wejściowego zdania. Jednak w trakcie korzystania z propagacji wstecznej gradienty mają tendencję do zanikania (zmierzania ku zeru) lub eksplodowania (zmierzania ku nieskończoności) w czasie.

Aby zapobiec takim sytuacjom obecnie zwykle stosuje się tzw. bramkowe sieci RNN jak na przykład sieci z długą pamięcią krótkoterminową (LSTM).

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Sieci LSTM w strukturze są dość podobne do sieci RNN. Natomiast różnią się w niej elementy

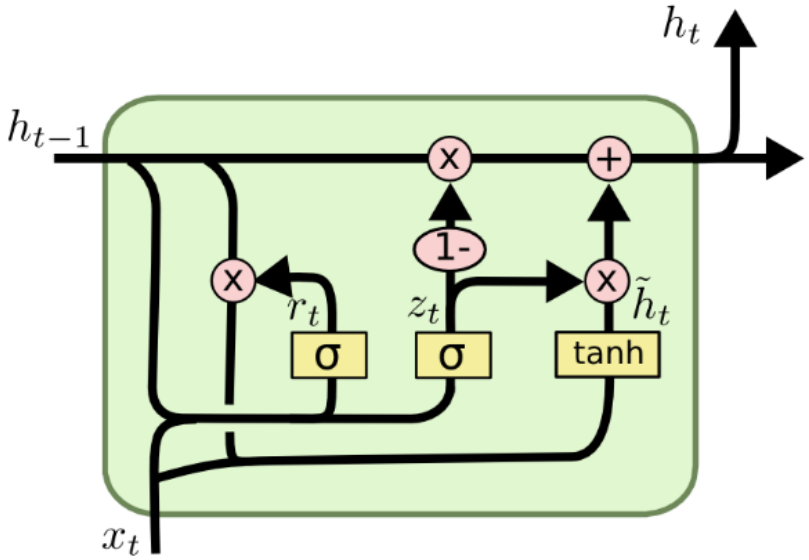


Rysunek 5.5: wygląd jednostki LSTM
Źródło: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

warstwy ukrytej i sposób przekazywania informacji. Zwykła jednostka LSTM posiada jedną komórkę pamięci i trzy bramki, które regulują przepływ informacji w warstwie. Znajduje się tam bramka wejściowa, wyjściowa i zapomnienia. W dalszej części rozdziału zostanie opisana struktura jednostki GRU, także opiera się na bramkach, ale o innej strukturze.
Komórka pamięci jest odpowiedzialna za przepływ informacji w sekwencji. Bramka wejściowa kontroluje ilość informacji, która przepływa do komórki. Bramka zapomnienia kontroluje ilość informacji, która zostaje zatrzymana w komórce. Natomiast bramka wyjściowa kontroluje ilość informacji przekazanej do nowej jednostki warstwy ukrytej.

GRU Innym rodzajem rekurencyjnej sieci neuronowej jest gated recurrent unit. Zasada działania jest taka sama jak LSTMu, natomiast różni się nieco jego wewnętrzna struktura.

Jednostka GRU posiada tylko dwie bramki, bramkę aktualizacji i regulacji, przez co ma mniejszą liczbę parametrów od LSTM.
W zależności od zadania GRU może być lepszym wyborem niż LSTM, dlatego często testuje się obie opcje.

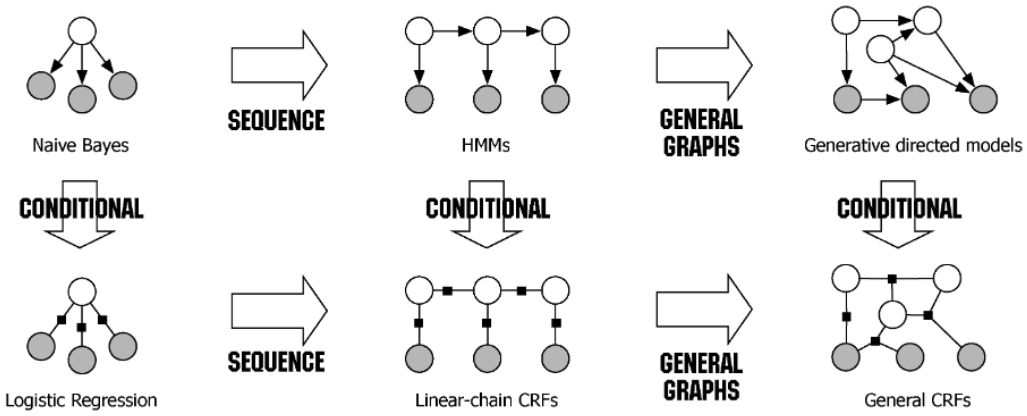


Rysunek 5.6: wygląd jednostki GRU
Źródło: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

5.3 Warunkowe pola losowe (CRF)

Warunkowe pola losowe (Conditional random fields - CRF) są statystyczną metodą uczenia maszynowego. Są one popularną alternatywą dla sieci neuronowych, szczególnie dla mniejszych zbiorów treningowych. Główną zaletą tych systemów jest możliwość analizy i predykcji sekwencji. Dlatego też są jedną z najpopularniejszych metod płytkiego nauczania maszynowego dla przetwarzania języka naturalnego.
CRF można reprezentować w postaci dyskryminowanego nieskierowanego, probablistycznego modelu grafu, który zapisuje związek pomiędzy zmiennymi. Ważną cechą tych modeli jest bezpośrednie połączenie między wyjściem a wejściem. Dzięki tym właściwościom CRF potrafi



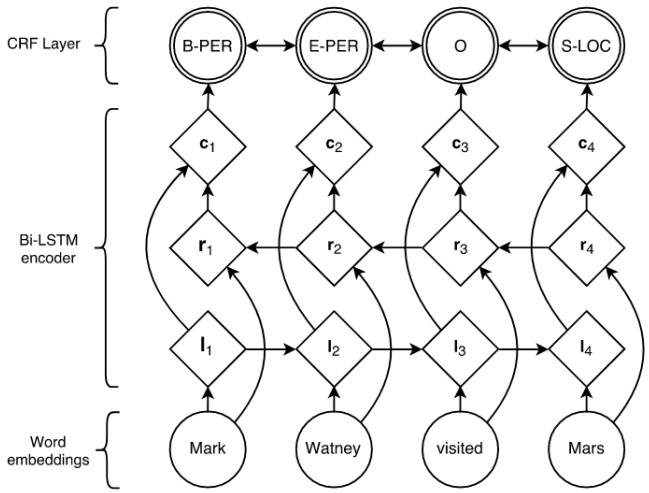
Rysunek 5.7: Warunkowe pola losowe

Źródło: https://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf

utworzyć spójną interpretacje wyników, co trudniej osiągnąć w sieciach neuronowych.

5.4 LSTM-CRF

Połączenie dwukierunkowej sieci LSTM z modelem CRF (rys 5.8) zostało zaproponowane w 2015 roku [16] i osiągnęło SOTA (State of the Art) w rozpoznawaniu nazw własnych. System ten potrafi wydajnie wykorzystywać informacje o całym kontekście słowa dzięki warstwie BILSTM, a także informacje na poziomie zdania dzięki CRF. W tym modelu model CRF jest podłączony do wyjść z warstwy LSTM i przewiduje najbardziej prawdopodobną sekwencję tagów ze wszystkich możliwości.



Rysunek 5.8: LSTM-CRF

Źródło: https://arxiv.org/abs/1603.01360

5.5 Tree-LSTM

Modele rekurencyjne takie jak LSTM i GRU są liniowe. To znaczy słowa przeglądane są w kolejności zgodnej z sekwencją segmentów idąc w przód lub w tył. Jednak takie rozwiązania bywają nieefektywne. Na przykładzie systemu odpowiadającego na pytanie, jeśli na wejściu znajdowałoby się następujące zdanie:

"Komunikatory, takie jak, WhatsApp, FB Messenger czy Discord, przetrzymują dużo informacji o użytkownikach"

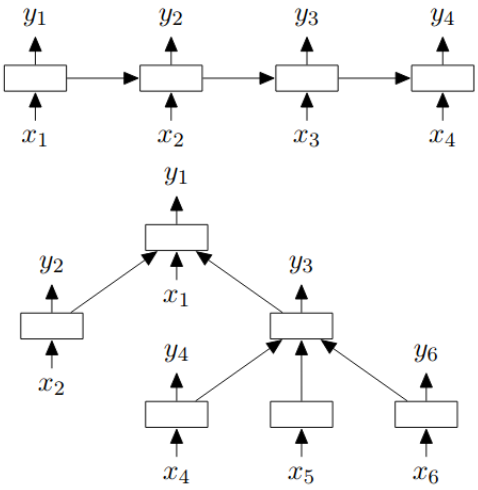
Pytaniem natomiast byłoby:

"Jakie systemy przetrzymują informację o użytkownikach?"

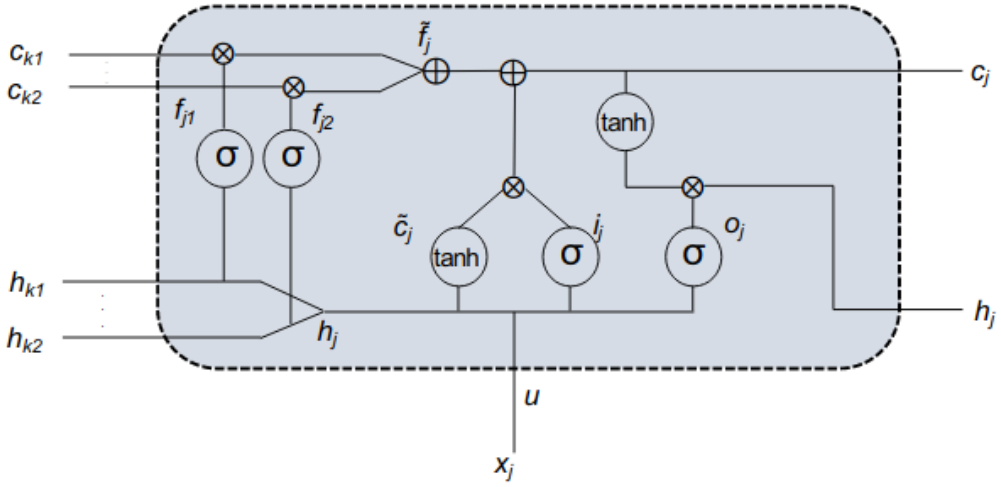
Jeśli system QA bazuje na liniowym modelu, to bardzo prawdopodobne, że wybierze odpowiedź najbliższą indeksem do słów zawartych w pytaniu. Czyli w tym wypadku byłaby to odpowiedź, "Discord". W tym przypadku odległość między słowami "przetrzymują", a "Komunikatory" wynosi aż 11 segmentów.
Natomiast jeżeli system bazuje na modelach zależnościowych to istnieje bezpośrednie połączenie między tymi segmentami. Zwiększa to znacznie prawdopodobieństwa wybrania bardziej ogólnej odpowiedzi.

Powyższe zdanie to tylko jeden z przykładów informacji strukturalnej, która jest tracona gdy wykorzystywane są modele sekwencyjne. Dlatego też powstał inny rodzaj sieci nazwany Tree-LSTM [18].

Na przykładzie 5.9 można zauważyć, że każda jednostka w LSTMie przyjmuje pewien wektor x_i na wejściu, informacje, z poprzedzającej ją liniowo komórki h_i oraz tworzy wektor y_i. Tak samo jednostka Tree-LSTM, przyjmuje na wejściu x_i i tworzy y_i, natomiast różni się przekazywanie informacji między jednostkami. Jednostka Tree-LSTM otrzymuje informacje od wszystkich swoich dzieci, a wyjście y_i jest przekazywane do rodzica. Aktualizacja komórki pamięci odbywa się poprzez przetworzenie stanów wszystkich dzieci, w przeciwieństwie do tylko stanu poprzednika w zwykłym LSTM.



Rysunek 5.9: Różnice między liniowymi, a drzewiastymi modelami
Źródło: https://arxiv.org/pdf/1503.00075.pdf



Rysunek 5.10: Wygląd jednostki Tree-LSTM
Źródło: https://arxiv.org/pdf/1901.00066.pdf

Rozdział 6

Opis opracowanego rozwiązania

Celem pracy było opracowanie metody do rozpoznawania nazw własnych w tekście poprzez wytrenowanie modułu na podstawie pliku zawierającego zbiór treningowy, a następnie przewidzenie nazw własnych na tekstach ze zbioru testowego.
Rozwiązanie jest przeprowadzone w następujących krokach

- Ujednolicenie struktury danych tekstowych i treningowych
- Utworzenie drzew zależności dla tekstu przy wykorzystaniu systemu COMBO [22]
- Ładowanie danych treningowych i testowych oraz wektorów słów
- Przetworzenie wstępne wejścia na format obsługiwany przez systemy uczenia maszynowego.
- Wydzielenie zbioru walidacyjnego ze zbioru treningowego
- Uczenie modelu sieci neuronowej za pomocą zbioru treningowego oraz sprawdzanie jego poprawności zbiorem walidaycjnym
- Przetworzenie wyjścia z systemu uczenia maszynowego na format zawierający frazy.
- Zapisanie wyjścia do pliku ewaluacyjnego.
- Ewaluacja wyników

Plik konfiguracyjny

W celu łatwej modyfikacji konfiguracji modelu i systemu został utworzony plik konfiguracyjny config.py, który zawiera parametry systemu, które opisywały zmieniające się dane w systemie jak np. liczbę epok do trenowania. Do tych parametrów należały :

- **emb_path** - ścieżka do pliku zawierającego wektory zanurzenia w formacie fasttext

- **train_dataset_path** - ścieżka do pliku treningowego
- **test_dataset_path** - ścieżka do pliku testowego
- **validation-size** - proporcja wielkości zbioru walidacyjnego do treningowego
- **epochs** -liczba epok uczenia

Słownik parametrów wykorzystanych do wyszukania najlepszych hiper-parametrów:

- **type** - simple lub tree, czy zostaje wykorzystany TreeLSTM [simple]
- **lowercase** - czy słowa na wejściu do sieci będą przetworzone na małe litery. np. “To jest Janek” na “to jest janek” [True]
- **rnn** - klasa wartstwy RNN [GRU]
- **dropout** Procentowa wartość przy warstwie dropoutu [0.5]
- **trainable_embeddings** - douczalne wektory słów [True],
- **optimizer** Optymalizator [adam]
- **is_crf** - czy wykorzystujemy CRF, opcja działa tylko w ypadku typu “simple” [True]

Dodatkowo utworzony został plik requirements.txt z listą bibliotek Pythonowych, które należy zainstalować w celu przeprowadzenia eksperymentów .

6.1 Zbiory danych

Korpus przechowywany jest w standardzie JSON. Jest to format elastyczny, łatwy do przetworzenia, a jednocześnie czytelny dla człowieka. Składa się z listy elementów lub krotek klucz i wartość.
Dane tekstowe z anotacjami nazw własnych są przetrzymywane w dwóch plikach. Pierwszy zawiera zbiór podkorpusu milionowego (którego dokumenty zostały później podzielone na zbiór treningowy i walidayjny). Na drugi składają się teksty zbioru testowego, udostępnionego z zadania Poleval 2018[23]. Z racji różnego pochodzenia oba zbiory zostały oryginalnie utworzone w różnych formatach, które zostały ujednolicone na potrzeby tego projektu.
W dodatkowych dwóch plikach w formacie “.txt” były przechowywane informacje o zależnościach w zdaniu wygenerowane poprzez model parsera zależnościowego COMBO [22].

6.1.1 Plik korpusu treningowego i walidacyjnego

Jako plik korpusu treningowego został wykorzystany przetworzony plik narododwego korpusu języka polskiego w formacie TEI do pliku JSON, który zawiera listę dokumentów.

Każdy z dokumentów jest zestawem zaanotowych tekstów i zawiera pola:

- **tokens** - lista słów w paragrafie.
- **offsets2Entities** - mapa, gdzie kluczami jest indeks początkowy danego słowa w tekście. Natomiast wartością lista krotek opisująca nazwy własne, w których zawarte jest powyższe słowo:
  - **text** - tekst nazwy własnej (pełen)
  - **type** - kategoria podstawowa
  - **subtype** (opcjonalne) - podkategoria
  - **offsets** - lista z polami "from" i "to" opisującymi start i zakończenie każdego słowa w nazwie własnej
  - **entities** - wartości z mapy offsets2Entities opisane jako lista elementów posortowana po indeksie słowa w tekście

6.1.2 Plik korpusu testowego

Zbiór testowy pochodził z konkursu Poleval 2018 z zadania rozpoznawania nazw własnych i składa się on z 1827 dokumentów. Dane zostały przygotowane w formacie JSON jako lista tekstów wraz z anotacjami. Każdy dokument składał się z kilku pól, którego uproszczony opis znajduje się poniżej:

- **tekst** - paragraf z jednego dokumentu w formacie tekstowym.
- **id** - nazwa pliku źródłówego
- **brat** - pole odpowiedzi w formacie tekstowym, które zostało dodane po ewaluacji wyników konkursu. Każda fraza była oddzielona znakiem nowej linii. Frazy składa się z elementów tekstowych oddzielonych tabulatorem:
  - T-#{numer tagu} - unikalny number tagu w dokumencie
  - Typ
  - Start<spacja>koniec - indeks rozpoczęcia i zakończenia nazwy własnej. W przypadku nazw nieciągłych anotacja wyglądała następująco start1<spacja>koniec1;start2<spacja>koniec2

– Pełen tekst nazwy własnej W przypadku fraz zagnieżdżonych każda fraza była traktowana jako osobna nazwa własna z nakładającymi się indeksami ale innym tekstem i kategorią.

Przykład nazw własnych w formie nieprzetworzonej:
"T1 \t persName 0 16 \t Rafał Wieczyński \n T2 \t persName_forename 0 5 \t Rafał \n T3 \t persName_surname 6 16 \t Wieczyński"
Przygotowany został skrypt przeznaczony do konwersji danych w formacie testowym do formatu zbioru treningowego.

6.1.3 Przygotowanie danych

Aby przygotować dane do trenowania sieci zostały poczynione następujące kroki:

Podział tekstu

Podział tekstu w zbiorze treningowym był dostępny z poziomu podkorpusu milionowego i te informacje zostały wyciągnięte z pliku TEI.
Za podział tekstu na zdania i segmenty w zbiorze testowym odpowiadała biblioteka NLTK. Wykorzystana została klasa TreebankWordTokenizer z modyfikacjami zapewniającymi informację o zakresie segmentów w tekście.
System COMBO [22] posiada własną anotację segmentów, przez co konieczne było utworzenie skryptu, który wyrównywał różnicę w segmentacji między plikami korpusu, a plikami z parsowaniem zależnościowym.

Utworzenie parsowania zależnościowego

Generacja anotacji parserem zależnościowym odbyła się poprzez wykorzystanie systemu COMBO [22].
Po wyznaczeniu zakresów zdań został zapisany tymczasowy plik tekstowy. W nim każde zdanie stanowiło osobną linię, co zostało przedstawione na rys 6.1.
Następnie uruchomiony został skrypt przewidujący zależności w zdaniu, przy wykorzystaniu

```
Czy to prawda , że ma się pan przenieść do Łodzi ?
Jestem po rozmowach z działaczami ŁKS , ale na razie nie mogę nic więcej powiedzieć
Nic nie jest przesądzone .
Mówi się również , że może wrócić pan na Cichą .
Też była taka możliwość i ona w dalszym ciągu jest .
```

Rysunek 6.1: Przykładowe zdanie z pliku testowego przygotowanego do anotacji zależnościami

modelu **191107_COMBO_PDB_semlab_parseonly.pkl** pochodzącego z Githuba[1] projektu COMBO [22]. Produkuje on plik typu .conll, którego przykład widoczny jest na rys. 6.2. Część z kolumn posiada tylko puste elementy _. Oznacza to, że nie przeprowadzona

```
1       Jak     _       _       _       _       3       adjunct_mod     _       _
2       nie     _       _       _       _       3       neg             _       _
3       dać     _       _       _       _       0       root            _       _
4       się     _       _       _       _       3       refl            _       _
5       oszukać _       _       _       _       3       comp_inf        _       _
6       ?       _       _       _       _       3       punct           _       _
```

Rysunek 6.2: Przykładowe zdanie z pliku testowego zawierającego parsowanie zależnościowe

została dodatkowa analiza tekstu, jak na przykład tagowanie części mowy, które wypełniłoby te kolumny. Kolumny niepuste składają się z:

- indeksu zdania
- segmentu
- indeksu rodzica
- nazwy zależności

Wydzielenie cech binarnych

Dodatkowym elementem na wejściu do modelu są cechy binarne do każdego słowa. Reprezentacja wektorowa sama w sobie zawiera bardzo dużo informacji, natomiast pewne elementy zostają utracone przy takiej transformacji. Jeśli posiadamy słowo “kot” i “Kot” i utworzymy ich reprezentacje wektorową, to będziemy posiadali dwa różne wektory. Możliwe, że będą znajdowały się blisko siebie, ale jeśli słowo “Kot” będzie rzadkie w z zbiorze uczącym, to ta relacja nie musi być zachowana. Można też zignorować wielkość liter i utworzy tylko jeden wektor słowa "kot". Tracimy natomiast wtedy informacje o wielkiej literze. Wielkość pierwszej litery jest szczeńólnie ważna przy nazwach własnych, bo stanowi istotny wyznacznik faktu, że jakieś słowo stanowi nazwę własną lub jej element.
Dlatego też w tej pracy została podjęta decyzja o zastosowaniu kilku binarnych cech. Ignorujemy wielkość liter przy tworzeniu wektorów słów, a następnie te, które zaczynają się od wielkiej litery, otrzymują dodatkową cechę. Wykorzystane cechy binarne to:

- Czy słowo zaczyna się z wielkiej litery?
- Czy słowo zawiera kropkę?
- Czy słowo zawiera liczbę?

[1] Adres do kodu i modeli projektu COMBO: https://github.com/360er0/COMBO

6.2 Modele

6.2.1 Padding

Dane wejściowe do modelu (sekwencja wektorów słów, cech binarnych, indeksów rodziców) zostały przetworzone tak, aby każda sekwencja wejściowa posiadała określoną długość N. Do tego celu została wykorzystana metoda pad_sequence z biblioteki Keras. Metoda ta przyjmuje na wejściu pewien zbiór wektorów, a następnie wypełnia je zerami, tak aby każdy z wektorów miał długość N. Wektory, które posiadały więcej niż N elementów, zostały skrócone. Elementy o indeksie większym niż N-1 zostały usunięte tak jak na przykładzie 6.1.

Zdanie	idx 0	idx 1	idx 2	idx 3
Rozumiem to.	Rozumiem	to	.	0
Mały kot bardzo głośno miauczy.	Mały	kot	bardzo	głośno

Tablica 6.1: Przykład wykorzystania metody pad_sequences

Przy takim odcięciu model nie będzie klasyfikował żadnego wyrazu w sekwencji o indeksie większym niż N-1. Słowa te będą ignorowane przez skrypty przetwarzające wyjście. Może to doprowadzić do postawiania fraz oznaczonych jako fałszywie nieprawdziwe w sytuacji, w której nazwa własna będzie znajdować się na końcu bardzo długiego zdania.

6.2.2 Opis modelu GRU/LSTM(-CRF)

Wykorzystany model korzysta z danych składających się z trzech sekwencji wektorów o długości zgodnej z długością ustaloną jako maksymalna długość wejścia i są to:

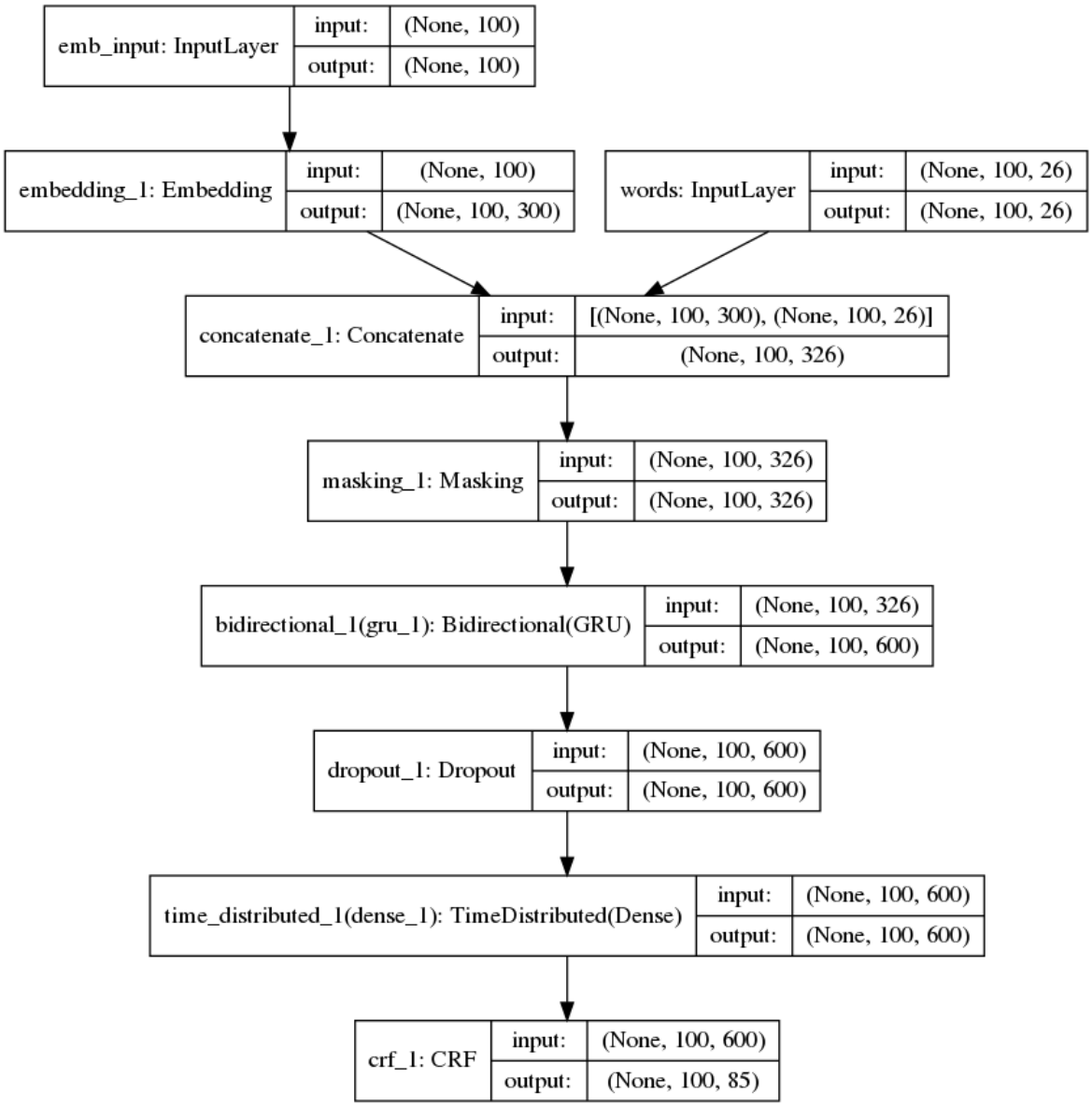
- sekwencja indeksów wektorów słów, gdzie każdy indeks odpowiada słowu ze zdania
- sekwencje:
  - cech binarnych dla każdego ze słów
  - cech parsowania zależnościowego

Sekwencja indeksów jest następnie przetwarzana na macierz wektorów słów o długości zgodnej z długością wytrenowanych wektorów słów. Następnie następuje złączenie wszystkich warstw tak aby powstał jeden tensor, który to jest wejściem do warstwy dwukierunkowego RNN. W zależności od wybranych hiperparametrów może to być GRU lub LSTM. Sekwencja wyjść z LSTM jest poddawana losowemu przesiewowi (dropout). Dropout ma za zadanie uniknięcie przeuczenia sieci neuronowej. Losowo, zgodnie z pewnym prawdopodobieństwem określonym

przed trenowaniem wyłącza część neuronów podczas trenowania. Następnym krokiem jest generowanie wyjścia, która jest tworzone na dwa sposoby:

- warstwa dense z aktywacją softmax
- warstwa CRF

Warstwy te produkują sekwencję wektorów o długości obciętego zdania i szerokości zgodnej z liczbą kategorii wyjściowych. W każdym z tych wektorów element o maksymalnej wartości oznacza najbardziej prawdopodobną (wg. modelu) kategorię słowa. Łączenie tych kategorii w oznaczenie nazw własnych zostało opisane w następnym rozdziale.



Rysunek 6.3: Przykład modelu (na rysunku model Bi-GRU-CRF)

Listing 6.1: Tworzenie modelu LSTM/GRU/ CRF-LSTM, CRF-GRU

```
def create_model(embeddings, emb_features, feature_size, maxlen,
                 output_size, model_parameters,class_weights):
    features = Input(shape=(maxlen, feature_size,), name='words')
    emb_input = Input(shape=(maxlen,), name='emb_input')
    embedding = Embedding(embeddings.shape[0], emb_features,
                 input_length=maxlen, weights=[embeddings],
                 trainable=model_parameters['trainable_embeddings'])(emb_input)
    concat = Concatenate(axis=-1)([embedding, features])
    mask = Masking()(concat)  # insert this
    lstm = Bidirectional(
        model_parameters['rnn'](output_dim=model_parameters['output_dim_rnn'],
                                activation=model_parameters['activation_rnn'],
                                return_sequences=True))(mask)
    dropout = Dropout(model_parameters['dropout'])(lstm)
    if model_parameters['is_crf']:
        part1 = TimeDistributed(Dense(600, activation="relu"))(dropout)
        crf = CRF(output_size, sparse_target=True, learn_mode="marginal")(part1)
        model = Model(inputs=[emb_input, features], outputs=[crf])
        model.compile(optimizer=model_parameters['optimizer'], loss=crf_loss,
                 metrics=[crf_viterbi_accuracy],class_weights=class_weights)
    else:
        part1 = TimeDistributed(Dense(600, activation="relu"))(dropout)
        final_dense = Masking()(part1)
        out = Dense(output_size, activation='softmax')(final_dense)
        model = Model(inputs=[emb_input, features], outputs=[out])
        model.compile(optimizer=model_parameters['optimizer'],
                 loss='binary_crossentropy', metrics=['acc'])
    model.summary()
    from keras.utils import plot_model
    plot_model(model, to_file='model.png', show_shapes=True)
    return model
```

6.2.3 Opis modelu TreeLSTM

Model TreeLSTM został napisany przy wykorzystaniu kodu przygotowanego do zadania analizy sentymentu [2].

Model ten na wejściu przyjmuje listę drzew, gdzie każdy węzeł jest segmentem zawierającym informacje o rodzicu, dzieciach, rodzaju zależności i indeksie wektora słów. System ten został napisany przy wykorzystaniu biblioteki Pytorch, która pozwala na dynamiczne dostosowanie długości zdania. Dzięki czemu w tym wypadku nie został użyty padding.

Sam model składa się z trzech modułów:

[2] Źródło https://github.com/tomekkorbak/treehopper

- Embedding: odpowiedzialny za przetworzenie indeksu segmentów na wektory słów, wykorzystujący klasę *torch.nn.Embedding*
- SequenceModule: odpowiedzialny za obliczenia przeprowadzane na sekwencji
- ChildSumTreeLSTM: odpowiedzialne za obliczenia przeprowadzane na jednostce, posiada metody:
  - get_children_states: wyliczające stan ukryty i stan komórki danej węzła na podstawie własnych inforacji i jej dzieci
  - forward: propagacja w przód na węzłach w sekwencji
  - node_forward: propagacja w przód na węźle

Listing 6.2: Tworzenie modelu TreeLSTM

```
class ChildSumTreeLSTM(nn.Module):
    def __init__(self, args, criterion, output_module):
        super(ChildSumTreeLSTM, self).__init__()
        self.cuda_flag = args['cuda']
        self.in_dim = args['input_dim']
        self.mem_dim = args['mem_dim']
        self.recurrent_dropout_c = args['recurrent_dropout_c']
        self.recurrent_dropout_h = args['recurrent_dropout_h']
        self.commons_mask = args['common_mask']
        self.zoneout_choose_child = args['zoneout_choose_child']

        self.ix = nn.Linear(self.in_dim, self.mem_dim)
        self.ih = nn.Linear(self.mem_dim, self.mem_dim)

        self.fh = nn.Linear(self.mem_dim, self.mem_dim)
        self.fx = nn.Linear(self.in_dim, self.mem_dim)

        self.ux = nn.Linear(self.in_dim, self.mem_dim)
        self.uh = nn.Linear(self.mem_dim, self.mem_dim)

        self.ox = nn.Linear(self.in_dim, self.mem_dim)
        self.oh = nn.Linear(self.mem_dim, self.mem_dim)

        self.criterion = criterion
        self.output_module = output_module

class SequenceModule(nn.Module):
    def __init__(self, args, dropout=0.5):
        super(SequenceModule, self).__init__()
        self.cuda_flag = args['cuda']
        self.mem_dim = args['mem_dim']
```

```
        self.num_classes = args['num_classes']

        self.dropout = dropout
        self.linear_layer = nn.Linear(self.mem_dim, self.num_classes)
        self.logsoftmax = nn.LogSoftmax()
        self.softmax = nn.Softmax()
        if self.cuda_flag:
            self.linear_layer = self.linear_layer.cuda()


class TreeLSTM(nn.Module):
    def __init__(self, args, criterion, embeddings):
        super(TreeLSTM, self).__init__()
        self.output_module = SequenceModule(args, dropout=0.5)
        self.tree_module = ChildSumTreeLSTM(args, criterion,
                                            output_module=self.output_module)
        self.embeddings = embeddings
```

Na wyjściu struktura drzewa zostaje zmodyfikowana, tak aby przybrała postać liniową. Postać liniowa jest przetwarzana tak samo jak wyjście z LSTMa i będzie opisana w następnej sekcji.

6.3 Wyjście

Wyjście z sieci neuronowej jest reprezentowane jako zbiór liczb dla każdego słowa. Przekazują one informacje z jakim prawdopodobieństwem słowo należy do każdej z kategorii wyjścia. Natomiast taka reprezentacja jest nieczytelna dla człowieka i zawiera dużo nieistotnych informacji. Konieczna jest więc odpowiednia modyfikacja wyjścia na inny format.

6.3.1 Reprezentacja wyjścia

Według zasad anotacji nazwy własne składają się z jednego lub więcej słów. Natomiast system rozpoznaje daną kategorię dla każdego słowa ze zdania osobno. Konieczne jest więc wykorzystanie odpowiednich metod do rozdzielenia frazy na słowa przy przetwarzaniu wstępnym i łączenia je w ewaluacji.
Przy ekstrakcji fraz w tekście najpopularniejszy jest format IOB. Gdzie każde ze słów przyjmuje jedną z trzech kategorii

- B (begin) - słowo tworzące początek frazy,
- I (inside) - słowo będące środkiem lub końcem frazy,
- O (outside) - słowo niebędące częścią żadnej frazy.

Istnieje też kilka alternatyw dla IOB na przykład:

- IO (gdzie I jest odpowiednikiem I + B. Daje nam mniej klas wyjściowych, ale nie obsługuje sytuacji, gdy frazy z tej samej klasy są nieoddzielone od siebie.
- IOBSE (gdzie E oznacza słowo kończące frazę, a S oznacza frazę zawierająca tylko jeden token. Tutaj natomiast znacznie zwiększamy liczbę możliwych klas)
  Jednak żaden z tych formatów nie obsługuje fraz zagnieżdżonych i nieciągłych.
  Frazy zagnieżdżone mają więcej niż jeden tag na słowo. Dlatego też została wykorzystana konkatenacja tagów. Tak więc słowo będące częścią dwóch nazw własnych może mieć adnotację BI, BB, II lub O ale nigdy IO czy BO.

Prócz oznaczenia zakresu nazwy, na wyjściu musi być także zawarta jej kategoria. W takim wypadku przeprowadzane jest łączenie tagu IOB i kategorii na poziomie przetwarzania wstępnego i tak, np. Frazy “Jan” i “Rysy” będą mieć dwie odrębne tagi na wyjściu: “B-person” i “B-geogname”. W takim wypadku wielkość wyjścia do sieci neuronowej wynosi 2*liczba kategorii +1 (odpowiadający za tag O).
Dodatkowo w przypadku sieci neuronowych opartych na Tensorflow należy utworzyć jeszcze jedną cechę na wyjściu - padding, opisanej w osobnym rozdziale. To zwiększa liczbę cech na wyjściu do 2*liczba kategorii +2.

W przypadku kiedy słowa należące do jednej frazy zostaną oznaczone różnymi kategoriami należy podjąć decyzję, która z kategorii zostanie jako kategoria frazy. Mamy kilka możliwości, możemy:

- Przypisać etykietę ostatniego słowa,
- Przypisać etykietę pierwszego słowa
- Przypisać etykietę, dla której uzyskano największe prawdopodobieństwo

Rozdział 7

Ewaluacja

Do ewaluacji został wykorzystany zmodyfikowany plik ewaluacyjny pochodzący z konkursu Poleval 2018.
Skrypt ten przyjmuje na wejściu dwa pliki:

- plik sprawdzający, zawierający ręczne anotacje
- plik zawierający przewidywania systemu

W oryginale format tych dwóch plików różnił się między sobą, co zostało ujednolicone w tej modyfikacji.

```
answers:    "T\tpersname_forename 0 5\tRafał\nT\tpersname_surname 6 14\tGuzowski\nT\tpersname 0 14\tRafał Guzowski"
text:       "Rafał Guzowski mówi , że władze miasta"
```

Rysunek 7.1: Przykład tekstu z pliku wyjściowego

7.1 System porównywania zakresów

W części przypadków systemy radzą sobie dobrze z zaznaczeniem frazy, ale nie w pełni jej zakresu. Jeśli w zdaniu

"*Polsko-Japońska Akademia Technik Komputerowych*"

system oznaczy "Polsko-Japońska Akademia" jako frazę, ale już "Technik Komputerowych" pozostanie oznaczona jako część niebędącą frazą, to pojawia się wtedy pytanie, czy taki błąd powinien być liczony z taką samą wagą jak nie zaznaczenie całkiem tej frazy.

W metrykach opisanych w innych artykułach i konkursach, liczenie poprawności frazy odbywa się per słowo. W przypadku przedstawionym powyżej 4 słowa zostały oznaczone

prawdziwie dodatnie a 2 fałszywie ujemne.

W konkursie Poleval zdecydowano się na inna metodę zliczania poprawnych i niepoprawnych fraz, dokładna (exact), bądź cząstkowa(overlap). W przypadku przedstawionym:

- łączenie dokładne: fraza "Polsko-Japońska Akademia Technik Komputerowych" będzie uznana za fałszywie ujemną - FN, a "Polsko-Japońska Akademia" za jako fałszywie dodatnią - FP
- łączenie cząstkowe: cała fraza będzie uznana za prawdziwie dodatnią - TP

W niniejszej pracy opisane są zawsze obie te metryki.
Niezależnie od systemu porównywania zakresów przypisanie złej etykiety zawsze jest traktowane jako fałszywie dodatnie i fałszywie ujemne.

7.2 Metryki

Do ewaluacji zostały wykorzystane standardowe metryki w rozpoznawaniu nazw własnych, to znaczy precyzja, czułość i f1.
Precyzja (precision) jest to liczba, która opisuje ile z oznaczonych klas zostało oznaczonych poprawnie względem anotacji.
Czułość (recall) jest to liczba, która opisuje ile z zaanotowanych fraz zostało poprawnie oznaczonych.
F1 jest średnią harmoniczną precyzji i czułości.

Metryki te oblicza się zgodnie ze wzorem:

$$precision = \frac{tp}{tp + fp} \tag{7.1}$$

$$recall = \frac{tp}{tp + fn} \tag{7.2}$$

$$f1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{7.3}$$

7.3 Wyniki

Poniżej przedstawione są 5 wyników doboru hiperparametrów z modelu prostego:

- **trainable_embeddings**: czy wektory słów można dotrenowywać podczas uczenia się

optimizer	is_crf	rnn	dependencies	f1-częściowe	f1-dokładne
adam	true	LSTM	true	0.0	0.0
adagrad	true	GRU	true	0.0	0.0
adagrad	false	GRU	true	0.0	0.0
adagrad	true	GRU	false	0.0	0.0
adagrad	false	GRU	false	0.721	0.655
adam	true	GRU	false	0.695	0.751
adam	false	GRU	false	0.691	0.751
adam	false	GRU	true	0.705	0.766
adam	true	GRU	true	0.704	0.759

Tablica 7.1: Przykłady wyników przetestowanych do wyboru najlepszych parametrów

- **is_crf**: czy wykorzystano warstwę CRF
- **rnn**: rodzaj warstwy rnn [GRU, LSTM]
- **optimizer**: optymalizator [adam, adagrad]
- **dependencies**: czy wykorzystano parsowanie zależnościowe

Poniżej przedstawione są wyniki wytrenowanych systemów na zbiorze testowym w dwóch rodzajach porównań zakresu: dokładnego i częściowego dopasowania.

Podczas doboru hiperparametrów dokonano następujących spostrzeżeń:

- Przy pierwszych testach modele wykorzystujące dotrenowywane wektory słów (f1-dokładne 0.655, f1-częściowe:0.721) odnosiły znacząco lepsze wyniki od niedotrenowywanych (f1-dokładne 0.578, f1-częściowe: 0.655). W celu zmniejszenia liczby modeli (i tym samym czasu trenowania) zdecydowano się na użycie tylko dotrenowanych wektorów przy doborze innych hiperparametrów.

- Przy pierwszych eksperymentach LSTM osiągnął wynik 0%. System oznaczył wszystkie słowa jako “Outside”. Z racji długiego czasu trenowania został on wyłączony z dalszego przeszukiwania hiperparametrów. Możliwe, że przy odpowiednio dobranych parametrach osiągnąłby porównywalne rezultaty jak GRU.
- Optymalizator adagrad okazał się niestabilny. W większości przypadków sieć nie

nauczyła się poprawnie i nie oznaczyła żadnych wyników.

Trudno określić z jakiego dokładnie powodu optymalizator adagrad i LSTM nie utworzyły zadowalających wyników. Możliwe, że tak duże niezbalansowanie korpusu, sprawiło, że najbardziej opłacalną opcją zostało oznaczenie słów jako “Outside”.

	Precyzja	Czułość	F1
GRU	0.729	0.657	0.691
GRU-crf	0.719	0.672	0.695
GRU-dependencies	**0.747**	0.668	0.705
GRU-crf-dependencies	0.739	**0.683**	**0.710**
TreeLSTM	0.614	0.496	0.549

Tablica 7.2: Ewaluacja dokładnego dopasowania

Z analizy wyników osobno dla każdej kategorii (7.4, 7.5) wynika, że najlepiej rozpoznawalnymi kategoriami są frazy pochodzące z placename_country, persname_forename i date. Prawdopodobnie wynika to z faktu, że kategorie te są dobrze zdefiniowane, krótkie oraz pochodzą z ograniczonego zbioru. W tych kategoriach system rzadko będzie musiał przewidzieć nazwę własną, której słów nie widział w zbiorze treningowym.
Natomiast najgorzej plasują się frazy placename, placename_bloc i placename_district. System wyuczył się, aby nie rozpoznawać kategorii placename_bloc i placename_district, co skutkowało bardzo niską czułością wyników. Natomiast dość często niepoprawnie oznaczał placename, przez co w tym wypadku kategoria ma bardzo niską precyzję.

Przeprowadzona została też ewaluacja fraz zagnieżdżonych Rys. 7.6. Wybrano tutaj tylko te nazwy własne, które posiadały zagnieżdżenia w zbiorze ewaluacyjnym jak i w zbiorze anotacji systemowych. W tym przypadku wyniki mogą być częściowo zaniżone, bo jeśli fraza była w części poprawnie oznaczona, ale system nie oznaczył zagnieżdżenia to nie była ona brana pod uwagę.

	Precyzja	Czułość	F1
GRU	0.792	0.714	0.751
GRU-crf	0.776	0.726	0.751
GRU-dependencies	**0.811**	0.726	**0.766**
GRU-crf-dependencies	0.795	**0.735**	0.764
TreeLSTM	0.767	0.619	0.685

Tablica 7.3: Ewaluacja częściowego dopasowania

Kategoria	TP	FN	FP	Czułość	Precyzja	F1
date	1501	277	495	84.42%	75.20%	79.54%
geogname	931	811	1574	53.44%	37.17%	43.84%
orgname	2572	1988	2529	56.40%	50.42%	53.25%
persname	7147	2774	1418	72.04%	83.44%	77.32%
persname_addname	77	690	721	10.04%	9.65%	9.84%
persname_forename	5216	1296	720	80.10%	87.87%	83.80%
persname_surname	3899	1831	820	68.05%	82.62%	74.63%
placename	2	1	90	66.67%	2.17%	4.21%
placename_bloc	0	197	1	0.00%	0.00%	0.00%
placename_country	2753	625	310	81.50%	89.88%	85.48%
placename_district	2	136	4	1.45%	33.33%	2.78%
placename_region	113	246	100	31.48%	53.05%	39.51%
placename_settlement	2254	1136	1296	66.49%	63.49%	64.96%
time	120	163	148	42.40%	44.78%	43.56%

Tablica 7.4: Ewaluacja dokładnego dopasowania dla każdej kategorii w BiLSTM-CRF

Kategoria	TP	FN	FP	Czułość	Precyzja	F1
date	1720	58	303	96.74%	85.02%	90.50%
geogname	1129	613	1382	64.81%	44.96%	53.09%
orgname	3285	1275	1878	72.04%	63.63%	67.57%
persname	7594	2327	986	76.54%	88.51%	82.09%
persname_addname	88	679	710	11.47%	11.03%	11.25%
persname_forename	5367	1145	569	82.42%	90.41%	86.23%
persname_surname	3990	1740	729	69.63%	84.55%	76.37%
placename	2	1	90	66.67%	2.17%	4.21%
placename_bloc	0	197	1	0.00%	0.00%	0.00%
placename_country	2808	570	255	83.13%	91.67%	87.19%
placename_district	2	136	4	1.45%	33.33%	2.78%
placename_region	150	209	64	41.78%	70.09%	52.36%
placename_settlement	2347	1043	1204	69.23%	66.09%	67.63%
time	185	98	89	65.37%	67.52%	66.43%

Tablica 7.5: Ewaluacja częściowego dopasowania dla każdej kategorii w BiLSTM-CRF

Z ewaluacji nazw zagnieżdżonych wynika, że system potrafi je lepiej rozpoznać niż

Zagnieżdżenia	typ	Precyzja	Czułość	F1
brak zagnieżdżeń	częściowe	0.811	0.726	0.766
brak zagnieżdżeń	dokładne	0.747	0.668	0.705
brak zagnieżdżeń	częściowe, bez kategorii Person	0.780	0.667	0.719
brak zagnieżdżeń	dokładne, bez kategorii Person	0.677	0.579	0.624
zagnieżdżenia	częściowe	0.825	0.718	0.768
zagnieżdżenia	dokładne	0.784	0.682	0.730
zagnieżdżenia	częściowe, bez kategorii Person	0.721	0.328	0.451
zagnieżdżenia	dokładne, bez kategorii Person	0.623	0.284	0.390

Tablica 7.6: Ewaluacja zagnieżdżeń dla najlepszego modelu GRU-crf-dependencies

frazy niezagnieżdżone. Jest to jednak niepełna prawda. Ze względu na strukturę anotacji, na większość zagnieżdżeń składają się nazwy osób, które są proste do rozpoznania. W szczególności przypadek słowa “Marek” będzie oznaczony jako fraza zagnieżdżona, gdyż posiada dwie kategorię “persname” i “persname_forname”, Dopiero przy wyłączeniu wszystkich fraz z kategorii osoba, można zauważyć, że w rzeczywistości system gorzej radzi sobie we frazach zagnieżdżonych.

Rozdział 8

Analiza błędów

W systemach rozpoznawania nazw własnych z racji swojej złożoności pojawia się wiele różnych rodzajów błędów. W niektórych przypadkach można zmniejszyć ich liczbę na przykład poprzez dodanie dodatkowych cech, które zapewniają informacje ze świata zewnętrznego. Natomiast w części przypadków nie jest możliwa poprawna anotacja frazy bez zmiany podstawowych założeń modelu. Na przykład fraza *WicemarszalekAndrzejChronowski* nigdy nie będzie poprawnie rozpoznana przez model ponieważ jest rozumiana jako jedno słowo, a nie połączenie ze sobą wielu wyrazów. Poniżej znajdują się przykładowe błędy, które pojawiły się przy predykcji zbioru testowego wraz z krótkim opisem.

8.1 Błędy ogólne

Anotacja na podstawie samego słowa

Jednym z dość popularnych błędów jest oznaczanie tekstu tylko na podstawie najbardziej popularnego znaczenia frazy, bez uwzględniania kontekstu. Zwykle pojawia się on w mniejszych zbiorach tekstów, gdzie system w dużej mierze opiera się na informacji pochodzącej z wektorów słów zamiast z kontekstów.

Dębski teczki miał u siebie przez ponad 1,5 miesiąca - wspomina Maj.

- anotacje systemu
  Maj: date
- anotacje ekspertów
  Maj: persName
  Maj: persName_surname

Jego twarz jest znana, bo nie raz, nie dwa pokazywali Kubę w tutejszych gazetach, kiedy

organizowano turnieje sportowe i zbierano pieniądze na jego leczenie

- anotacje systemu
  Kubę: placename_country
- anotacje ekspertów
  Kubę: persname_forename

Znaczeniowo poprawne

Niektóre frazy oznaczone przez system jak i ekspertów oznaczają to samo w rozumieniu przez człowieka, mimo że nie nakładają się na siebie w całości. W języku angielskim błąd ten występuje bardzo często w przypadku przedimków. Niekiedy zadanie rozpoznania czy dany przedimek powinien stanowić część nazwy własnej jest bardzo trudne nawet dla człowieka.

Na przykład, na Starym Mieście jest ulica Freta.

- anotacje systemu
  ulica Freta: geogname
- anotacje ekspertów
  Freta: geogname

Przez 24 godziny na dobę było otwarte społeczne centrum operacyjne oraz biuro Straży Miejskiej

- anotacje systemu
  biuro Straży Miejskiej: orgname
- anotacje ekspertów
  Straży Miejskiej: orgname

Podobną można dostać w plastikowych kubeczkach na nieodległym Dworcu Centralnym

- anotacje systemu
  nieodległym Dworcu Centralnym: geogname
- anotacje ekspertów
  Dworcu Centralnym: geogname

Listowanie

Bardzo trudna jest do rozpoznania nazwa własna podana bez kontekstu jako listowanie pewnych elementów. System wtedy musi polegać na informacjach zawartych w wektorach słów lub ewentualnie w podobnej strukturze dokumentów.

(...) Stokłosa (58 Błaś) , Marion (70 Wojtyga) , Wrześniak (87 Wróbel) (...)

- anotacje systemu
  87: persname_forename
  87 Wróbel: persname
- anotacje ekspertów
  Wróbel: persname_surname

Przerwanie nazwy

Systemy rozpoznawania nazw własnych bardzo silnie polegają na wielkości liter w słowach niebędącymi pierwszymi wyrazami w zdaniu. Dlatego też małe litery, jak chociażby łączniki potrafią nie być wyłapane przez system. Może to skutkować rozerwaniem poprawnej nazwy na dwie części lub ucięciem nazwy przed jej końcem. W przypadku podziału nazwy na dwie części będzie to oznaczać utworzenie dwóch błędnych nazw.

złożymy wniosek do Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej o dofinansowanie

- anotacje systemu
  Narodowego Funduszu Ochrony: orgname
- anotacje ekspertów
  Narodowego Funduszu Ochrony Środowiska i Gospodarki Wodnej: orgname

Błędy związane z brakiem informacji zewnętrznych

Jeśli zdanie zawiera bardzo krótki kontekst, z którego nie wynika jakim typem jest nazwa własna, systemy najczęściej będą oznaczać najbardziej prawdopodobną kategorią. Takie błędy są trudne do uniknięcia. Jeśli pojawiają się w większej liczbie, można rozważyć podanie zewnętrznych informacji jako cechę na wejściu do modelu, jak na przykład słowniki miejsc czy popularnych osób. W przypadku, w którym dostępne są dokumenty posiadające wiele

zdań, system może tą informację próbować otrzymać na podstawie kontekstu innych zdań.

sp1 : na Krupski Młyn. a najlepiej bo Lena ma ząbka

- anotacje systemu
  Krupski Młyn: placename_settlement
- anotacje ekspertów
  Krupski Młyn: geogname

Specjalne teksty

Wykorzystany zbiór danych pochodzi z różnorodnych źródeł, o różnej trudności analizy. Tak jak artykuły naukowe czy notki prasowe są bardzo dobrym źródłem do znalezienia nazw własnych, teksty pochodzące z rozmów czy innych źródeł komunikacji bardzo często są ciężkie do rozszyfrowana. Składa się na to inny format tekstu, błędy ortograficzne czy słowa spoza słownika

WicemarszalekAndrzejChronowski: Kto się wstrzymał od gosu

- anotacje systemu
- anotacje ekspertów
  AndrzejChronowski: persname
  Chronowski: persName_surname
  Andrzej: persName_forename

Nazwy własne nienależące do analizowanych kategorii.

W zależności od zbioru danych kategoryzacja nazwy własnej nie musi ograniczać się do osób, lokacji i organizacji. Systemy w dużej mierze opierają się na informacji o wielkich literach, dlatego też zdarza się, że oznaczają nazwy własne, które w rozumieniu tego zadania nie są szukaną nazwą własną.

Przy jego wykorzystaniu możemy podejrzeć zawartość każdej bazy, do której posiadamy sterownik ODBC, a nie tylko JDataStore

- anotacje systemu
  ODBC: placename_settlement
  JDataStore: placename_settlement

- anotacje ekspertów

8.2 Błędy specyficzne dla fraz zagnieżdżonych

Przy analizie błędów specyficznych dla fraz zagnieżdżonych można zauważyć, że system nie ma problemów z rozpoznaniem fraz o stałej strukturze jak na przykład "ul. św. Marka" lub "I Liceum Ogólnokształcące w Krakowie"
Jednak w przypadku bardziej nietypowych fraz zaczynają się pojawiać błędy.
Jednym z przykładowych fraz sprawiających trudność do rozpoznania są nazwy własne zawierające myślnik. Prawdopodobnie przez niestandardową strukturę zdarza się, że system niepoprawnie oznacza zakres fraz wewnętrznych oraz ich kategorię.
Kolejnym typem frazy sprawiającej trudność do rozpoznania są nazwy zawierające małe litery. W takich sytuacjach mała litera rozdziela frazę, sprawiając trudność w określeniu jej pełnego zakresu. System wtedy traktuje frazę, która powinna być zagnieżdżona jako kilka niezależnych nazw własnych.
Poniżej znajduje się przykład poprawnie oznaczonych fraz zagnieżdżonych:

W parafii św. Kazimierza w Toronto właśnie odbyła się msza,

- Toronto: placename_settlement
- Kazimierza: persname_forename
- parafii św. Kazimierza: geogname
- św. Kazimierza: persname

(...)budynek dawnego szpitala obywatelskiego przy ul. Słowackiego

- ul. Słowackiego: geogname
- Słowackiego: persname
- Słowackiego: persname_surname

Rozdzielenie nazw własnych

W błędach tego rodzaju najczęściej system omija zagnieżdżenie i traktuje nazwy własne jako niezależne od siebie. Najczęściej jest to spowodowane poprzez niepoprawne oznaczenie przyimka we frazie.

DANE URZĘDU MIEJSKIEGO W RZYMIE

- anotacje systemu
  URZĘDU MIEJSKIEGO: orgname
  RZYMIE: placeName_settlement
- anotacje ekspertów
  URZĘDU MIEJSKIEGO W RZYMIE: orgname
  RZYMIE: placeName_settlement

(...) przyznaje Wioletta Janas , wicedyrektor Gimnazjum nr 5 w Elblągu.

- anotacje systemu
  Elblągu: placename_settlement
  Gimnazjum nr 5 w: orgname
- anotacje ekspertów
  Elblągu: placename_settlement
  Gimnazjum nr 5 w Elblągu: orgname

Frazy z myślnikiem

Ze względu na nietypową strukturę we frazach z myślnikiem często pojawiają się błędy. Zdarza się, że system nie jest w stanie wyróżnić granic frazy ani jej typu.

Nie musimy zakładać, jak Lévy-Bruhl, że ludzie (...)

- anotacje systemu
  Lévy-Bruhl: persname
  Lévy-Bruhl: persname_addname
- anotacje ekspertów
  Lévy: persname_surname
  Bruhl: persname_surname
  Lévy-Bruhl: persname

Henri de Toulouse-Lautrec na pewno by tutejszej kawy nie wypił.

- anotacje systemu
  Henri: orgname
  Toulouse-: persname_forename
  de Toulouse-Lautrec : persname
  Lautrec : persname_surname

- anotacje ekspertów
  Henri: persname_forename
  Henri de Toulouse-Lautrec: persname
  de Toulouse-Lautrec: persname_surname

Nie oznaczenie wewnętrzej frazy

Błąd w którym cała fraza zewnętrzna jest poprawnie oznaczona, ale wewnątrz niej znajduje się zagnieżdżona i nieoznaczona fraza. Zwykle pojawia się przy niestandardowych frazach zagnieżdżonych.

Znajdziecie ich przy ulicy Reduta Miś

- anotacje systemu
  ulicy Reduta Miś: geogname

- anotacje ekspertów
  ulicy Reduta Miś: geogname
  Reduta Miś: orgname

Nie oznaczenie zewnętrzej frazy

Błąd w którym pewna fraza została poprawnie oznaczona, ale jest częścią większej nazwy własnej, która została pominięta. Zwykle pojawia się przy niestandardowych frazach zagnieżdżonych. *(...) w drodze do Pani Jasnogórskiej, do Królowej Polski*

- anotacje systemu
  Polski: placename_country

- anotacje ekspertów
  Królowej Polski: persname
  Polski: placename_country

Rozdział 9

Podsumowanie

Idea wykorzystania parsowania zależnościowego do rozpoznawania nazw własnych wydaje się interesująca gdyż bierze pod uwagę strukturę tych fraz.
W podanej pracy został przedstawiony opis i analiza modeli sieci neuronowej o różnych strukturach:

- model rekurencyjnej dwukierunkowej sieci neuronowej GRU lub LSTM,
- model rekurencyjnej dwukierunkowej sieci neuronowej GRU lub LSTM z warstwą CRF
- model TreeLSTM

Zostało też przetestowane czy sekwencyjne modele rekurencyjne osiągają lepsze wyniki, jeżeli wykorzystują informacje z parsera zależnościowego.
W tym eksperymencie system GRU-CRF z informacjami parsowania zależnościowego osiągnął najlepsze wyniki. Na drugim miejscu znajduje się wynik tego samego modelu, ale niewykorzystujący warstwy CRF. Wynik ten tylko nieznacznie ustępuje pierwszemu miejscu. Na trzecim i czwartym miejscu stanęły modele niewykorzystujące zależności. Znacznie niższe wyniki osiągnął model TreeLSTM.
Najwyższe wyniki osiągnęły modele wykorzystujące parsowanie zależnościowe. Oznacza to, że sieć neuronowa wykorzystuje przynajmniej część z informacji przekazywanych im na wejściu.

TreeLSTM osiągnął najgorsze rezultaty. Ponieważ nie analizuje on kontekstu zdania, więc taki wyniki nie jest zaskakujący. W nazwach własnych kontekst jest szczególnie ważny. Szukane słowa dość często nie znajdują się w słowniku i nie były widziane przez zbiór trenujący. Niekiedy bazują więc tylko na informacji ze zdania, której brakuje w tym przypadku. Jedną z możliwości kontynuacji eksperymentu mogłoby być wykorzystanie warstw ukrytych pochodzących z TreeLSTM jako cechę w sekwencyjnych modelach.

W podanej pracy zostało pokazane, że parsowanie zależnościowe wspomaga rozpoznawanie nazw własnych i może z powodzeniem być wykorzystywana jako informacja w różnych modelach uczenia maszynowego.

Bibliografia

[1] Erik F. Tjong Kim, Fien De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition". arXiv:cs/0306050, june 2003

[2] Sekine, Satoshi; et.al. Ęxtended Named Entity Hierarchy". november 2002.

[3] Yosef, Mohamed Amir, et.al, "HYENA: Hierarchical Type Classification for Entity Names", 2012, "Proceedings of COLING 2012: Posters"

[4] Chinchor, Nancy. Message Understanding Conference (MUC) 7 LDC2001T02. Web Download. Philadelphia: Linguistic Data Consortium, 2001.

[5] Nadeau, David & Sekine, Satoshi. (2007). A Survey of Named Entity Recognition and Classification. Lingvisticae Investigationes. 30. 10.1075/li.30.1.03nad.

[6] Hridoy Jyoti Mahanta. A study on the approaches of developing a named entity revognition tool, IJRET: International Journal of Research in Engineering and Technology

[7] Data Community DC. “A Survey of Stochastic and Gazetteer Based Approaches for Named Entity Recognition - Part 2”. ,

[8] Agata Savary, Marta Chojnacka-Kuraś, Anna Wesołek, Danuta Skowrońska, Paweł Śliwiński. Anotacja jednostek nazewniczych. In: Narodowy Korpus Języka Polskiego. Editors: Adam Przepiórkowski and Mirosław Bańko and Rafał L. Górski and Barbara Lewandowska-Tomaszczyk. Wydawnictwo Naukowe PWN, Warsaw. 2012. pp 129-165

[9] Finkel, Jenny Rose, i Christopher D. Manning. “Nested Named Entity Recognition”. Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing Volume 1 - EMNLP ’09, t. 1, Association for Computational Linguistics, 2009, s. 141. Crossref, doi:10.3115/1699510.1699529.

[10] Katiyar, Arzoo, i Claire Cardie. “Nested Named Entity Recognition Revisited”. Proceedings of the 2018 Conference of the North American Chapter ofthe Association for Computational Linguistics: Human LanguageTechnologies, Volume 1

(Long Papers), Association for Computational Linguistics, 2018, s. 861–71. Crossref, doi:10.18653/v1/N18-1079.

[11] Palshikar, Girish. (2012). Techniques for Named Entity Recognition: A Survey. Bioinformatics: Concepts, Methodologies, Tools, and Applications. 1. 191-. 10.4018/978-1-4666-3604-0.ch022.

[12] Chiu, Jason P. C. & Eric Nichols. “Named Entity Recognition with Bidirectional LSTM-CNNs”. arXiv:1511.08308 [cs], November 2015. arXiv.org,

[13] Bojanowski Piotr, et. al. “Enriching Word Vectors with Subword Information”. arXiv:1607.04606 [cs], july 2016. arXiv.org, http://arxiv.org/abs/1607.04606.

[14] Cho, Kyunghyun; et al (2014). Łearning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". arXiv:1406.1078

[15] Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun; Bengio, Yoshua (2014). Ęmpirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling"

[16] Lample Guillaume, et.al “Neural Architectures for Named Entity Recognition”. arXiv:1603.01360 [cs], march 2016.

[17] Huang, Zhiheng, i in. "Bidirectional LSTM-CRF Models for Sequence Tagging”. arXiv:1508.01991 [cs], august 2015. arXiv.org,

[18] Tai, Kai Sheng, Richard Socher, and Christopher D. Manning, 2015. Improved semantic representations from tree-structured long short-term memory networks.CoRR, abs/1503.00075

[19] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

[20] Malvina Nissim. Rik van Noord, Rob van der Goot, “Fair is Better than Sensational:Man is to Doctor as Woman is to Doctor”

[21] Desmet, Bart & Hoste, Véronique. (2010). "Towards a Balanced Named Entity Corpus for Dutch."

[22] Rybak, Piotr and Wróblewska, Alina. “Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies” (October 2018)

[23] Aleksander Wawer, Estera Małek. “Results of the PolEval 2018 Shared Task 2:Named Entity Recognition” 2018

[24] Yijin Liu, Fandong Meng. “GCDT: A Global Context Enhanced Deep Transition Architecture for Sequence Labeling” 2019

[25] Daniel Jurafsky James H. Martin. “Speech and Language Processing” Chapter 15,2019

Spis rysunków

1.1 Zdanie z nazwą własną. ' Maria Nowak ' jest oznaczona jako osoba 5

2.1 Przykład zdania z frazą nieciągłą. W zdaniu można zidentyfikować dwie nazwy osób: 'Jan Nowak ' i 'Maria Nowak ', przy czym część oznaczająca nazwisko jest współdzielona i jednocześnie sama nazwa "Jan Nowak" jest nieciągła. . . 10
2.2 Wykres SOTA dla zbioru CONLL-2003 (stan na 01.06.2020) 13
2.3 Wyniki SOTA dla zbioru CONLL-2003 (stan na 01.06.2020) 13

3.1 Hierarchia kategorii nazw własnych w podkorpusie milionowym 16
3.2 Fraza ul. św. Jana Nepomucena jako zaklasyfikowana lokalizacja. Jan Nepomucen jest zagnieżdżoną frazą oznaczoną jako osoba 18

4.1 Przykład segmentacji zdania 22
4.2 Przykład drzewa zależności 22
4.3 Parsowanie zależnościowe przykładowego zdania z nazwami własnymi 23
4.4 Model CBOW 26
4.5 Model skipgram 27
4.6 Wektory słów rzutowane na przestrzeń dwuwymiarową 28
4.7 Zależności między wektorami rzutowanymi na przestrzeń trójwymiarową . . . 29

5.1 Perceptron 32
5.2 Popularne funkcje aktywacji 33
5.3 Perceptron wielowarstwowy 34
5.4 Rekurencyjne sieci neuronowe 34
5.5 wygląd jednostki LSTM 35
5.6 wygląd jednostki GRU 36
5.7 Warunkowe pola losowe 37
5.8 LSTM-CRF 37
5.9 Różnice między liniowymi, a drzewiastymi modelami 39
5.10 Wygląd jednostki Tree-LSTM 39

6.1 Przykładowe zdanie z pliku testowego przygotowanego do anotacji zależnościami 44
6.2 Przykładowe zdanie z pliku testowego zawierającego parsowanie zależnościowe 45
6.3 Przykład modelu (na rysunku model Bi-GRU-CRF) 47

7.1 Przykład tekstu z pliku wyjściowego 53

Spis tablic

3.1 Procentowy skład źródeł tekstów w podkorpusie milionowym 17

4.1 Przykład podzielonych zdań na segmenty. 23
4.2 Przykład podzielonych zdań na segmenty z wykorzystaniem indeksów słownika. 24
4.3 Przykład sekwencji wektorów jeden z 1 24
4.4 Przykład reprezentacji workiem słów. 24

6.1 Przykład wykorzystania metody pad_sequences 46

7.1 Przykłady wyników przetestowanych do wyboru najlepszych parametrów . . 55
7.2 Ewaluacja dokładnego dopasowania 56
7.3 Ewaluacja częściowego dopasowania 56
7.4 Ewaluacja dokładnego dopasowania dla każdej kategorii w BiLSTM-CRF . . 57
7.5 Ewaluacja częściowego dopasowania dla każdej kategorii w BiLSTM-CRF . . 57
7.6 Ewaluacja zagnieżdżeń dla najlepszego modelu GRU-crf-dependencies 58

10.1 Zestaw bibliotek użytych do utworzenia systemu 79

Słownik terminów

anotacja Wzbogacanie tekstów o informacje. 75

Bi-LSTM-CRF Model dwukierunkowej sieci LSTMpołączony z warstwą CNN. 12, 75

BiLSTM Model dwukierunkowej sieci LSTM. 12, 75

BiLSTM-CNN Model dwukierunkowej sieci LSTM połączony z warstwą CRF. 12, 75

CNN Convolutional neural network. 75, 77

CONLL Conference on Natural Language Learning. 75, 77

CONLL-2003 Konferencja CONLL, do której było przygotowane zadanie i zbiór rozpoznawania nazw własnych. 10, 12, 16, 18, 75

CRF Conditional Random Fields. 75, 77

korpus Zbiór tekstów służący badaniom lingwistycznym. 75

LSTM Long Short Term Memory (Network). 75, 77

MUC Message Understanding Conference. 9, 11, 75, 77

muc-7 Konferencja MUC, do której było przygotowane zadanie i zbiór rozpoznawania nazw własnych. 16, 75

NKJP Narodowy Korpus Języka Polskiego. 18, 19, 75

NLP Natural Language Processing (Przetwarzanie języka naturalnego). 5, 75

poleval Konkurs organizowany przez Polską Akademię Nauk, mający na celu wzbogacenie narzędzi dla przetwarzania języka naturalnego w języku polskim. 75

Rozdział 10

Appendix

Tablica 10.1: Zestaw bibliotek użytych do utworzenia systemu

Nazwa	Wersja	Opis
gensim	2.0.0	biblioteka do przetwarzania języka naturanego. W projekcie wykorzystana do ładowania zanurzeń słów z pliku do pamięci
h5py	2.9.0	biblioteka wykorzystywana do zapisywania modeli w formacie HDF5
Keras	2.2.2	Biblioteka będąca wysokopoziomową nakładką na niskopoziomowe biblioteki do tworzenia sieci neuronowych (w tym wypadku tensorflow)
nltk	3.4.4	zestaw bibliotek i programów do symbolicznego i statystycznego przetwarzania języka naturalnego.
pydot	1.4.1	interfejs do biblioteki graphviz do rysowania wizualizacji danych
scikit-learn	0.19.0	narzędzie do eksploracji i analizy danych
scipy	0.19.0	narzędzie rozszerzające opcje bibliotek naukowych takich jak numpy i pandas
tensorboard	1.10.0	tworzy wizualizacje eksperymentów wykonanych w tensorflow
tensorflow	1.10.0	Nisko poziomowa biblioteka do tworzenia sieci neuronowych
torch	0.4.0	Biblioteka do tworzenia sieci neuronowych
tqdm	4.14.0	biblioteka do wypisywania pasku postępu przy iterowaniu
numpy	1.17.1	narzędzie zajmujące się przetwarzaniem wielowymiarowych danych
matplotlib	3.1.2	biblioteka do rysowania wykresów w dwóch wymiarach
keras-contrib	2.0.8	Rozszerzenie biblioteki Keras o nowe warstwy, funkcje aktywacji i straty