POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
Instytut Informatyki

Praca dyplomowa magisterska

ROZPOZNAWANIE EMOCJI W MEDIACH SPOŁECZNOŚCIOWYCH Z WYKORZYSTANIEM GŁĘBOKICH SIECI NEURONOWYCH

inż. Jakub Zdanowski, 127239

Promotor
dr hab. inż. Agnieszka Ławrynowicz

Opiekun
mgr inż. Maksymilian Marcinowski

POZNAŃ 2020

Tutaj będzie karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wstęp

Emocje międzyludzkie są podstawą codziennych interakcji z innymi osobami, a badania naukowców pokazują, że emocje są zjawiskiem uniwersalnym dla ludzi bez względu na pochodzenie. Jednak wpływy kulturowe jak i interpersonalne odgrywają kluczową rolę w identyfikacji konkretnych nawet podstawowych emocji, takich jak radość, miłość, gniew, strach i złość. Im bardziej wyszczególnione są etykiety emocji, tym trudniej jest wykryć tę właściwą. System rozpoznawania emocji może okazać się przydatny do wzajemnego zrozumienia między osobami, poprzez dostarczenie niewykrytego sygnału emocji.

Doskonałym przykładem są emocje występujące w mediach społecznościowych, które w niektórych sytuacjach są bardzo wyraziste, lecz bywają też niejednoznaczne i przy tym mogą być wyrażane w wulgarny sposób. System wykrywania emocji w obszarze dialogów między ludźmi w postaci rozmowy na forum internetowym lub komentarzy pod postem może okazać się bardzo pomocny w poprawieniu bezpieczeństwa w Internecie dla ludzi młodych i dzieci. Przykładów zastosowań jest mnóstwo, kilka z nich to filtrowanie treści, blokada słów wulgarnych i obraźliwych oraz zdań z ukrytym podtekstem. Jest to także rekomendacja treści, grupowanie tekstu o podobnym znaczeniu, czy nawet badanie rynku. Można by wykorzystać taki system do zbadania większej liczby komentarzy, np. w przypadku koncernów samochodowych, czy firm produkujących elektronikę jakie emocje przewyższają w komentarzach pod wyświetlanymi reklamami w mediach społecznościowych. Czy jest to zachwyt, zadowolenie, czy rozczarowanie. Działania te mogły by odpowiedzieć na pytanie czy wydany właśnie przez nich produkt przyjmie się na rynku oraz co warto by poprawić. W takich przypadkach etykietowanie komentarzy przez człowieka mogłoby okazać się trudne do wykonania i niejednoznaczne oraz bardzo kosztowne i czasochłonne.

1.1 Cel i zakres pracy

Głównym celem pracy jest opracowanie modelu uczenia maszynowego opartego na głębokich sieciach neuronowych (klasyfikatora wieloklasowego) w celu detekcji emocji w tekście postów z dialogów z mediów społecznościowych. W ramach tego celu niezbędne będzie zapoznanie się z literaturą dotyczącą przetwarzania języka naturalnego i dostępnymi frameworkami do uczenia głębokiego, przeprowadzenie analizy eksploracyjnej wybranych

zbiorów danych, wstępne przetworzenie tych danych po czym nastąpi budowa i ewaluacja kilku modeli o różnych architekturach w celu przeprowadzenia analizy porównawczej.

1.2 Struktura pracy

Struktura pracy jest następująca. Rozdział 2 przedstawia podstawy teoretyczne wymagane do zrozumienia dalszych etapów pracy wraz z przeglądem literatury. Rozdział 3 jest poświęcony analizie eksploracyjnej wybranych zbiorów danych. Rozdział 4 zawiera techniki przetwarzania wstępnego zbiorów danych. Rozdział 5 ukazuje budowę modeli a rozdział 6 ich ewaluację. Rozdział 7 zawiera podsumowanie pracy.

Rozdział 2

Podstawy teoretyczne

2.1 Rozpoznawanie emocji

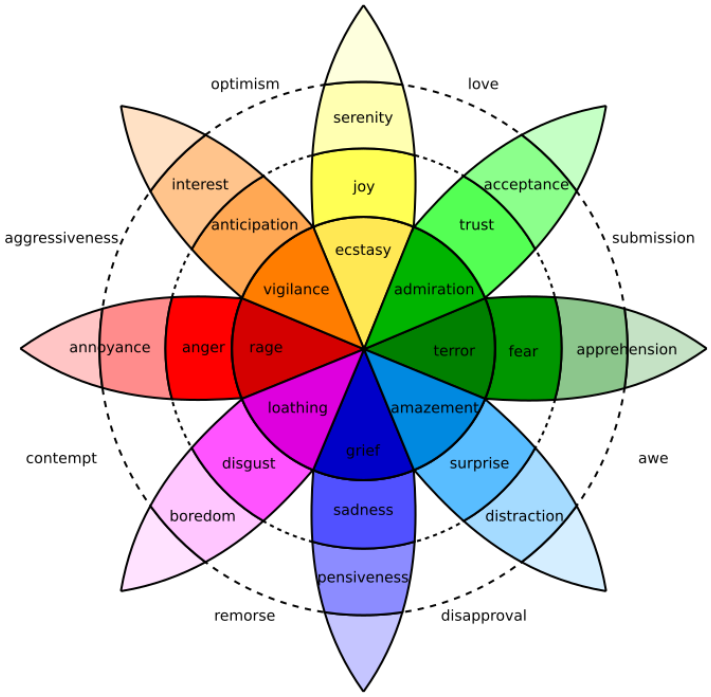
Rozpoznawanie emocji w dialogach koncentruje się na wydobyciu emocji przekazanej w rozmowie pomiędzy co najmniej dwoma rozmówcami. Problem ten stawia bardzo dużo wyzwań, takich jak obecność sarkazmu w rozmowie, przesunięcie emocji do kolejnych wypowiedzi tego samego rozmówcy oraz uchwycenie szerszego kontekstu pomiędzy wypowiedziami różnych osób. Dużym plusem w tej dziedzinie jest bardzo dobra dostępność do danych, które pochodzą z platform społecznościowych takich jak Facebook, Youtube, Reddit, Twitter [12]. Poprzez łatwy dostęp do danych rozpoznawanie emocji w rozmowie staje się coraz bardziej popularne, a trudność tego problemu stwarza coraz to bardziej odległe granice co sprowadza się do wysokiego zainteresowania tą dziedziną przetwarzania języka naturalnego (ang. *natural language processing - NLP*).

2.2 Modele emocji

Aby dobrze zrozumieć postawiony problem niezbędne będzie określenie czym są emocje. Wszyscy ludzie posiadają wrodzony zestaw podstawowych emocji, które można rozpoznać za pomocą gestów, czynów lub wypowiadanych słów. Możemy wyróżnić dyskretne emocje, aby móc odróżnić je od siebie.

2.2.1 Model emocji Ekmana

Istnieje kilka definicji różnych modeli emocji, jednym z nich jest model zaproponowany przez Paula Ekmana [5]. Paul wraz ze współpracownikami stwierdzili, że istnieje sześć podstawowych emocji: gniew, obrzydzenie, strach, szczęście, smutek i zaskoczenie, a z każdą z tych emocji związane są jakieś cechy. Dzięki temu można wyrazić emocje w różnym stopniu a każda z nich jest zdefiniowana jako dyskretna kategoria, co pozwala na dość łatwą klasyfikację konkretnej emocji.



Rysunek 2.1. Koło emocji Plutchika [11].

2.2.2 Model emocji Plutchika

Kolejną definicję modelu emocji przedstawił Robert Plutchik, który podzielił emocje na osiem podstawowych typów, z których każdy ma drobniejsze podtypy pokrewne [11], zaprezentowane na rysunku 2.1 za pomocą koła emocji. Prezentuje on emocje jako koncentryczne kręgi, gdzie wewnętrzne części odpowiadają za podstawowe emocje a te zewnętrzne za bardziej złożone. Model ten jest dyskretny, lecz widać w nim pewne zależności i podobieństwa pomiędzy sąsiadującymi częściami koła emocji. Budowa ta wynika ze złożoności emocji i możliwości wyrażania ich intensywności.

2.2.3 Porównanie modeli emocji

Podsumowując wymienione modele emocji możemy wydzielić dwa główne typy: kategoryczne oraz wymiarowe. Modele wymiarowe mapują emocję w sposób ciągły na wektory. Modele kategoryczne klasyfikują emocję do konkretnej emocji dyskretnej, np. jednej z wybranego modelu emocji Ekmana lub Plutchika. Modele kategoryczne mają pewne wady. Jedną z nich jest brak możliwości opisania innych emocji oraz utrudnione opisywanie emocji złożonej z kilku różnych podtypów zdefiniowanych w dyskretnym modelu. Drugą wadą jest brak możliwości porównywania emocji, co umożliwiłby model wymiarowy, za pomocą porównywania dwóch wektorów. Wybór odpowiedniego modelu emocji nie jest łatwy, a jednocześnie jest bardzo ważnym elementem do późniejszej klasyfikacji emocji. Decydując się na kategoryczny typ emocji, z jednej strony mamy prosty model Ekmana który nie jest w stanie zamodelować złożonych emocji. Z drugiej strony w modelu Plutchika może

być bardzo trudno rozróżnić drobnoziarniste emocje od siebie. Wybór ten należy zatem dokonać mając na uwadze wielkość oraz jakość zbioru danych.

2.3 Głębokie uczenie

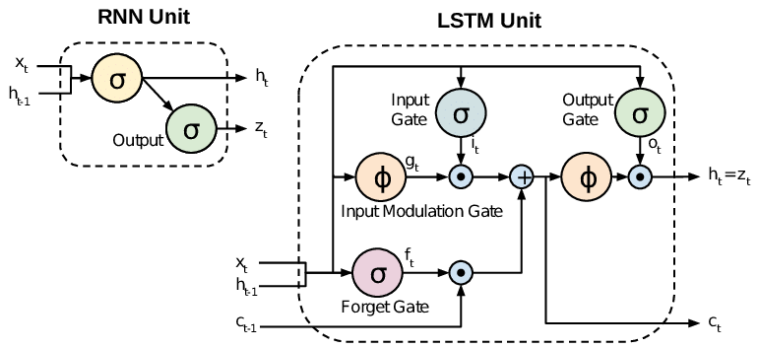
Bardzo ważnym elementem w rozpoznawaniu emocji jest możliwość zrozumienia danego przekazu w kontekście, od którego może zależeć rodzaj emocji. Szczególnie trudnym przypadkiem jest zrozumienie i zapamiętanie kontekstu w konwersacji, co jak pokazują autorzy artykułu na temat architektury głębokiego uczenia do rozpoznawania emocji w rozmowach tekstowych [20], może okazać się kluczowym czynnikiem skuteczności tych metod. Do uzyskania satysfakcjonujących wyników nie wystarczają tradycyjne metody uczenia maszynowego lub najbardziej podstawowe architektury sieci neuronowych [8]. Niezbędne jest użycie sieci rekurencyjnych oraz różnych rozszerzeń pozwalających na zapamiętanie i zrozumienie kontekstu w tekście.

2.3.1 Rekurencyjne sieci neuronowe

Tradycyjne sieci neuronowe nie są w stanie podejmować decyzji na podstawie sekwencji zdarzeń, w tym przypadku kolejnych słów w zdaniu. Sieci rekurencyjne wykorzystują informacje zawarte w całej sekwencji za pomocą powiązania wyjścia z komórek do następnych obliczeń w tej samej komórce. Dodatkowo każda komórka może używać swojej pamięci wewnętrznej do przechowywania informacji o poprzednim wejściu. Sieci rekurencyjne bardzo dobrze radzą sobie w problemach przetwarzania tekstu, rozpoznawania mowy a także szeregów czasowych. Umożliwiają one zrozumienie kontekstu wypowiedzi na podstawie pozostałych słów znajdujących się w zdaniu.

2.3.2 Sieci rekurencyjne LSTM

Rozszerzeniem zastosowania rekurencyjnych sieci neuronowych jest zastosowanie bardziej złożonych komórek. Przykładem są komórki LSTM (ang. *Long Short-Term Memory*), które w przeciwieństwie do standardowych komórek sieci neuronowych, posiadają połączenia zwrotne, umożliwiające zapamiętanie sąsiednich stanów w sieci. Dodatkowo są odporne na wygaszanie długotrwałych zależności w zdaniu. Jest to sytuacja w której znaczenie danego słowa zależy od słowa znajdującego się dużo wcześniej w zdaniu. Komórki LSTM złożone są z kilku modułów umożliwiających zapamiętanie długotrwałych zależności. Głównym elementem jest komórka pamięci, która utrzymuje swój stan w czasie. Dodatkowo występują jednostki bramki (ang. *gates*), które decydują o tym jaką część informacji brać do obliczeń w obecnym kroku i regulują przepływ informacji płynących do oraz z komórek pamięci. Szczegóły budowy oraz porównanie do tradycyjnych komórek sieci rekurencyjnych (RNN) zaprezentowano na rysunku 2.2.



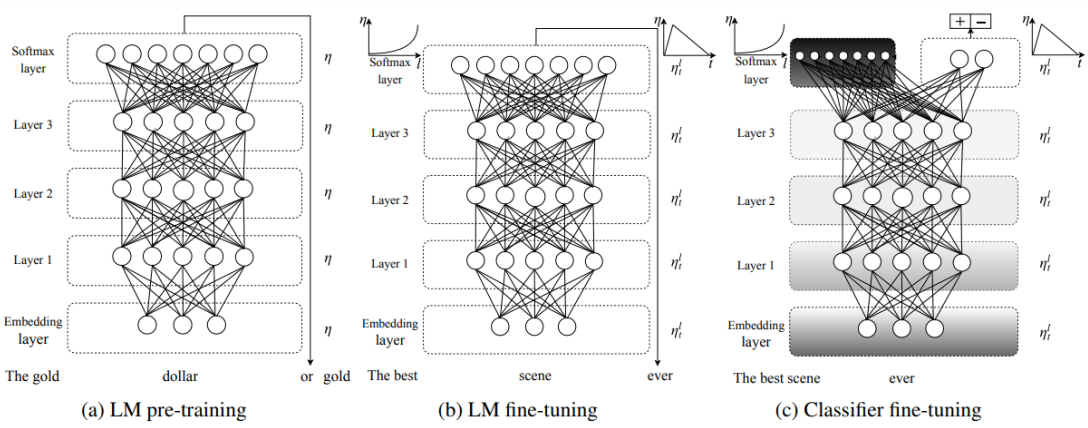
Rysunek 2.2. Porównanie komórek rekurencyjnych (RNN) z komórkami LSTM.

2.4 Transfer wiedzy lingwistycznej

W przetwarzaniu języka naturalnego z użyciem głębokich sieci neuronowych coraz częściej używane są techniki transferu wiedzy (ang. *transfer learning*) oraz adaptacji domenowej. Model języka jest kluczowym elementem do zastosowania powyższych technik. Umożliwia on przewidzenie kontekstu w jakim dane słowo znajduje się w zdaniu i na tej podstawie umożliwia odkryć jego prawdziwy sens. Jest uważany za bardzo istotny element w dziedzinie *NLP*, który stanowi podstawę do wszelkich zastosowań przetwarzania języka naturalnego. Najważniejsze jego cechy to zrozumienie długofalowych zależności i hierarchicznej struktury tekstu, a największe zalety to otwarte i wolne zasoby do jego stworzenia. Jest tworzony za pomocą nienadzorowanego procesu uczenia, który potrzebuje tylko korpusu nieoznakowanego tekstu.

2.4.1 Metoda ULMFIT

Znakomitym przykładem użycia transferu wiedzy za pomocą wielokrotnego uczenia modelu języka jest metoda *ULMFIT* [6] (ang. *Universal Language Model Fine-tuning for Text Classification*, która może być zastosowana do każdego zadania w NLP. Zastosowane są w niej techniki, które są kluczowe dla dostrojenia modelu językowego. Używane są w niej 3 warstwy sieci neuronowej wykorzystującej komórki LSTM. Dodatkowo zastosowana jest technika przerywania (ang. *dropout*) niwelująca problem przeuczania. Cały etap nauki w metodzie *ULMFIT* składa się z 3 etapów zaprezentowanych na rysunku 2.3. Na początku następuje szkolenie wstępne modelu językowego na dowolnym korpusie, następnie dostrojenie modelu językowego na zadaniu docelowym i na końcu dostrojenie klasyfikatora na zadaniu docelowym. Dzięki zastosowaniu tych technik możliwe jest wyuczenie wstępne modelu języka na dowolnych danych, (np. korpus Wikipedii) a następnie wykorzystanie tego wstępnie wyuczonego modelu na zadaniu docelowym. Na podobnej zasadzie działają dzisiejsze najbardziej wyrafinowane architektury głębokich sieci neuronowych w zastosowaniu przetwarzania języka naturalnego nazywane aktualnym stanem techniki (ang. *state of the art*).



Rysunek 2.3. 3 etapy nauki modelu języka (ang. *LM*) w metodzie ULMFIT [6].

2.4.2 Architektura transformera

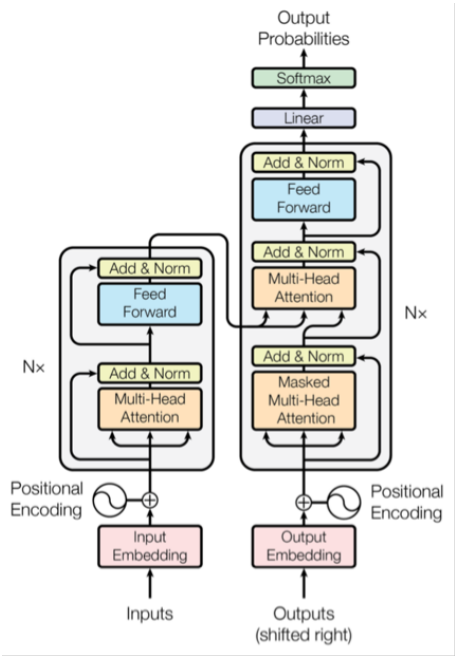
Transformer (ang. *The Transformer*) [16] jest to model który wykorzystuje mechanizm uwagi, inaczej samoobserwacji (ang. *self-attention*) do przyspieszenia procesu nauki. Główny jego cel to poprawa modelowania sekwencja do sekwencji (ang. *Seq2Seq*) poprzez samoobserwację i kodowanie pozycji (ang. *positional encoding*). Składa się on z dwóch głównych komponentów zaprezentowanych na rysunku 2.4. Jest to część kodująca (po lewej stronie) oraz część dekodująca (po prawej stronie), które są ze sobą połączone.

Elementem kodującym jest stos koderów o tej samej strukturze. Pierwszą warstwą każdego kodera jest mechanizm samoobserwacji. Umożliwia to łączenie znaczenia danego słowa z innymi słowami w zdaniu, poprzez kodowanie danego słowa. Wyjście z warstwy samoobserwacji przekazywane jest do sieci neuronowej typu *feed-forward*, która jest stosowana do każdej pozycji w zdaniu.

W części dekodującej także znajduje się stos dekoderów o tej samej liczności co koderów. Wszystkie z nich mają identyczną strukturę, podobną do struktury kodera. Dodatkowo pomiędzy warstwami występującymi w koderze znajduje się dodatkowa warstwa uwagi, która pomaga dekoderowi skupić się na odpowiednich częściach zdania wejściowego.

Zdanie jest przetwarzane na reprezentację wektorową na wejściu pierwszego kodera. Maksymalna szerokość tego wektora to 512. Wektor ten, jako reprezentacja zdania wejściowego przechodzi przez każdą z dwóch warstw kodera.

Najważniejszą częścią budowy tego modelu jest wspomniany wcześniej mechanizm samoobserwacji, który pozwala na tak dobre zrozumienie danych słów w kontekście całego zdania. Podczas gdy model przetwarza każde słowo, uwaga własna pozwala mu patrzeć na inne słowa w zdaniu w celu poszukiwania wskazówek, które mogły by pomóc w lepszym zakodowaniu tego słowa. Dzięki temu podobnie jak w rekurencyjnych sieciach neuronowych możliwe jest zrozumienie znaczenia danego słowa poprzez zapamiętanie znaczenia pozostałych słów w zdaniu.



Rysunek 2.4. Architektura modelu transformera [16].

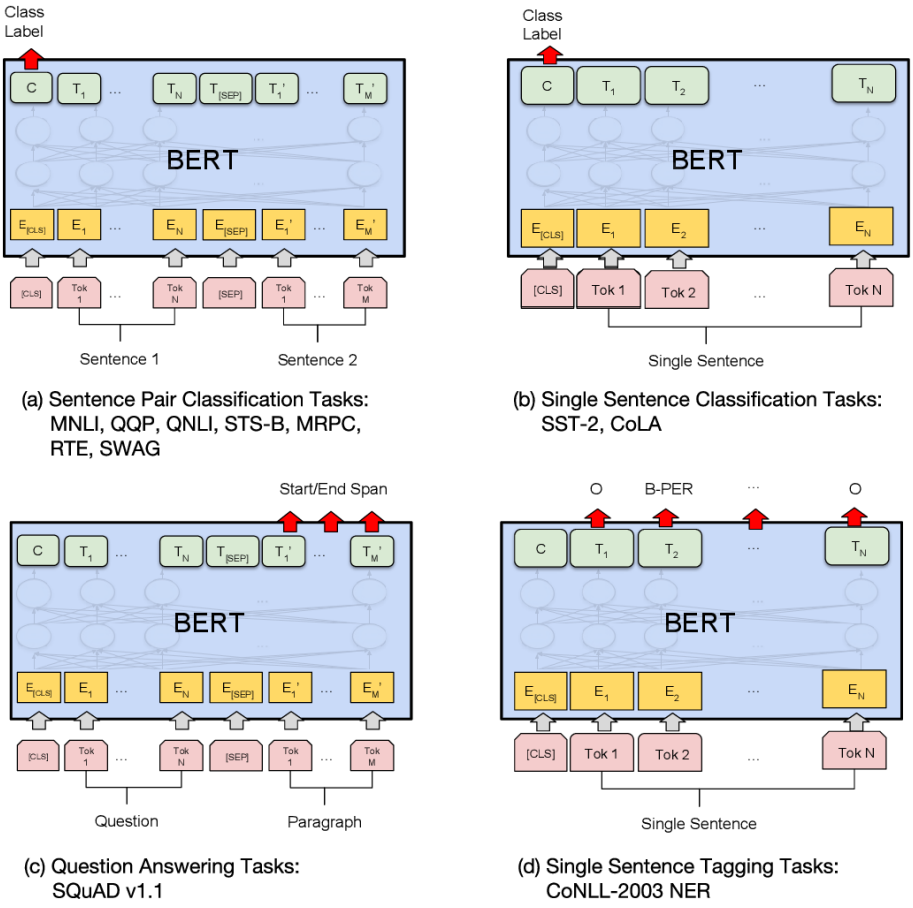
2.4.3 BERT

Rok 2018 był przełomowy dla modeli uczenia maszynowego przetwarzających tekst, a wydanie BERT (ang. *Bidirectional Encoder Representations from Transformers*) określane jest jako początek nowej ery w dziedzinie przetwarzania tekstu. Architektura ta jest niejako następcą metody ULMFIT oraz Transformera, gdyż główne jej założenia wywodzą się właśnie z tych metod. Wkrótce po wydaniu artykułu opisującego metodę [3], zespół z Google udostępnił także kod źródłowy modelu, który umożliwia użycie tej architektury w uproszczony sposób.

Architektura tej metody składa się ze stosu koderów zaprezentowanych dla Transformera. Każdy z nich zawiera warstwę samoobserwacji oraz sieć neuronową typu *feed-forward*. Dodatkowo używana jest metoda dwukierunkowej nauki, która umożliwia osiągnięcie jeszcze lepszego wyniku.

BERT opiera się na sposobie trenowania podobnym do przedstawionego w metodzie ULMFIT. Jest to częściowo nadzorowane uczenie na dużym zbiorze danych. Wykorzystane są metody pozwalające na efektywne wykorzytsanie tego, czego model uczy się podczas wstępnego treningu. Wprowadza on bowiem model językowy, który pozwala na efektywne użycie wiedzy lingwistycznej do zastosowania w problemach przetwarzania tekstu. Sam proces nauki tego modelu jest oparty o dwa zadania które umożliwiają obliczenie funkcji strat, są to problemy przewidywania następnego zdania oraz rozpoznanie ukrytego słowa w zdaniu.

Model BERT może być używany do wielu różnych zadań językowych, za pomocą drobnych modyfikacji jednej z ostatnich warstw. Bardzo dobrze sprawdza się w zadaniach rozpoznawania par zdań, rozpoznawania sentymentu, odpowiadanie na pytania czy tagowania zdań. Wszystkie te problemy przedstawione są na rysunku 2.5.



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Rysunek 2.5. Zastosowanie BERT do różnych problemów [3].

Rozdział 3

Analiza eksploracyjna

3.1 Zbiór danych - EmoContext

Zbiór danych został udostępniony przez zespół z Międzynarodowych Warsztatów Ewaluacji Semantycznej (ang. *SemEval-2019 International Workshop on Semantic Evaluation*) jako jedno z zadań konkursowych o nazwie EmoContext[1] (ang. *Contextual Emotion Detection in Text*). Celem jest odkrycie prawidłowej etykiety emocji dla danego dialogu, składającego się z trzech wypowiedzi. W zadaniu tym użyty jest uproszczony model emocji zaproponowany przez Paula Ekmana [5], składający się z czterech klas: *Happy*, *Sad*, *Angry* oraz *Others*. Przykładowe dialogi oraz odpowiadające im etykiety emocji pochądzące ze zbioru treningowego zaprezentowane są na rysunku 3.1.

	id	turn1	turn2	turn3	label
0	0	Don't worry I'm girl	hmm how do I know if you are	What's ur name?	others
1	1	When did I?	saw many times i think -_-	No. I never saw you	angry
2	2	By	by Google Chrome	Where you live	others
3	3	U r ridiculous	I might be ridiculous but I am telling the truth.	U little disgusting whore	angry
4	4	Just for time pass	wt do u do 4 a living then	Maybe	others
5	5	I'm a dog person	youre so rude	Whaaaat why	others
6	6	So whatsup	Nothing much. Sitting sipping and watching TV....	What are you watching on tv?	others
7	7	Ok	ok im back!!	So, how are u	others
8	8	Really?	really really really really really	Y saying so many times...i can hear you	others
9	9	Bay	in the bay	😘 love you	others

Rysunek 3.1. Przykładowe dialogi w danych z EmoContext wraz z etykietą emocji.

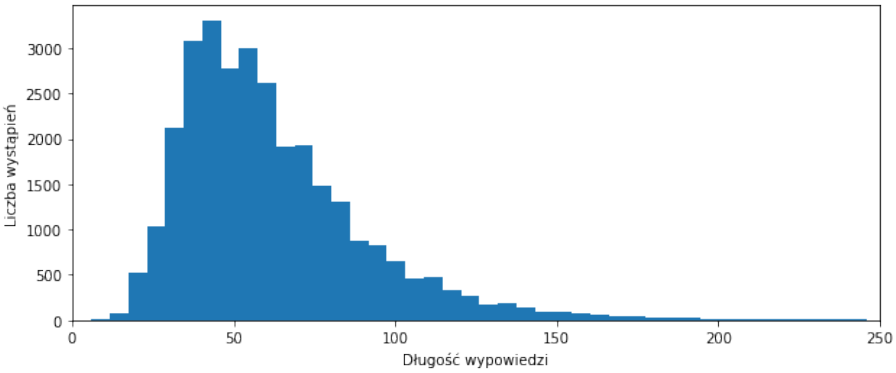
3.1.1 Szczegóły poszczególnych zbiorów

Przez organizatorów konkursu udostępnione zostały następujące zbiory do ewaluacji własnych modeli: zbiór treningowy (*train*), zbiór walidacyjny (*dev*) oraz zbiór testowy (*test*). Każdy z tych zbiorów zawiera tą samą strukturę danych, każdy przykład składa się z identyfikatora (*Id*), trzech wypowiedzi w dialogu oraz etykiety emocji. Szczegółowe informacje na temat tych zbiorów ukazuje tabela 3.1.

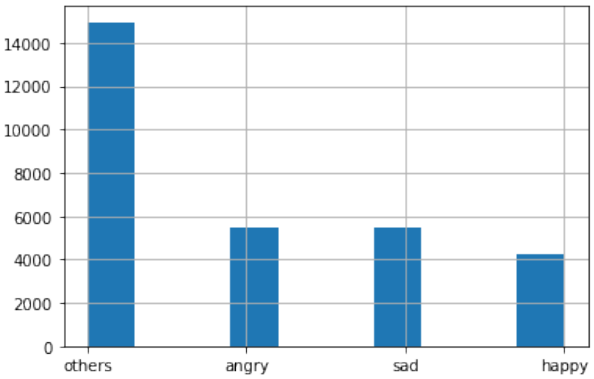
[1] `https://www.humanizing-ai.com/emocontext.html`

Tablica 3.1
Tabela prezentująca informacje o zbiorach w danych z EmoContext.

zbiór	liczba przykładów	najczęstsza klasa
train	30160	*Others* 50%
dev	2755	*Others* 84%
test	5509	*Others* 84%



Rysunek 3.2. Rozkład długości dialogu w danych z EmoContext.



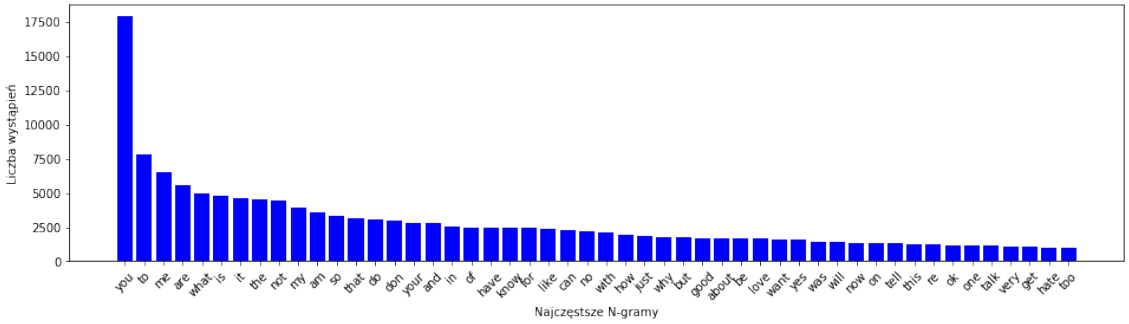
Rysunek 3.3. Niezbalansowany rozkład etykiet emocji w danych z EmoContext.

3.1.2 Analiza zbiorów

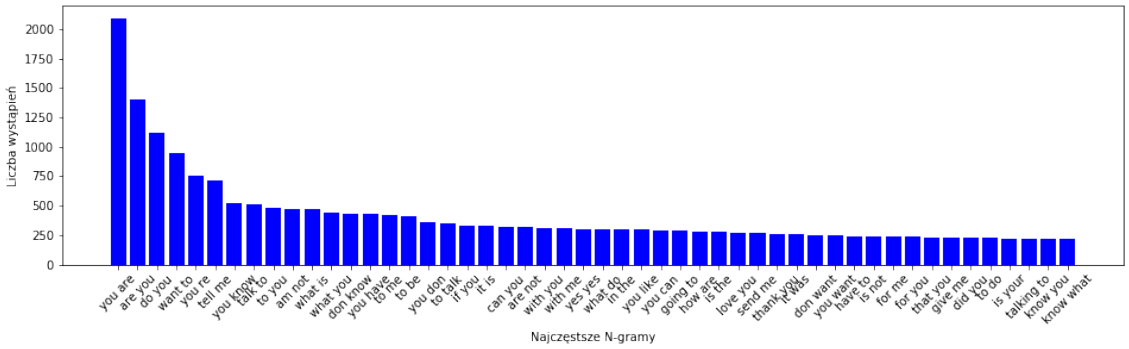
Aby lepiej zrozumieć dane treningowe przeprowadzone zostały podstawowe analizy eksploracyjne tego zbioru. Jedną z nich jest rozkład długości całego dialogu zaprezentowany na rysunku 3.2. Widzimy na nim że przeważające dialogi są dosyć krótkie, średnio 62 znaki, co może utrudnić zadanie rozpoznania właściwej emocji. Zauważyć można także występujący długi ogon w kierunku coraz dłuższych dialogów, najdłuższy z nich ma 692 znaki.

Kolejna analiza to histogram częstości występowania danej etykiety w zbiorze treningowym, dzięki temu można sprawdzić czy występuje niezbalansowanie klas. Rysunek 3.3 prezentuje dominację klasy *Others* nad pozostałymi klasami, jest średnio ponad dwukrotnie liczniejszy od każdej z pozostałych etykiet.

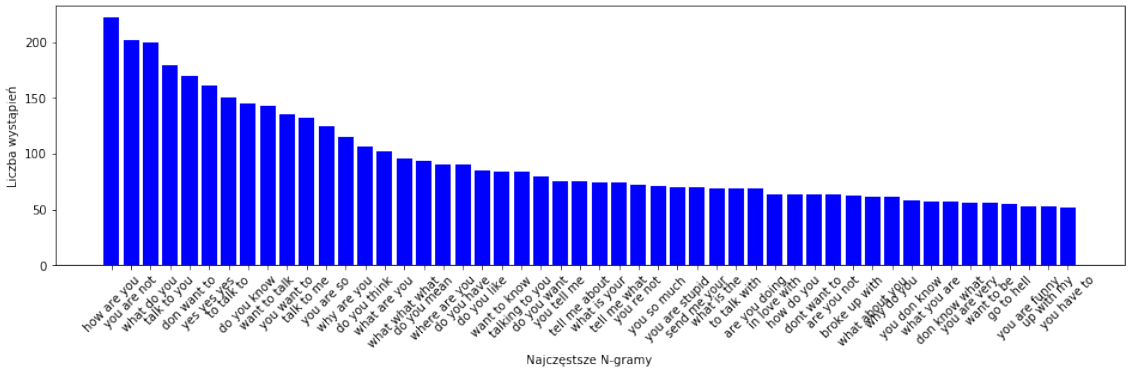
Ostatnim elementem analizy eksploracyjnej tego zbioru jest identyfikacja najczęściej występujących *N-gramów*, gdzie N to liczba zlepek słów występujących obok siebie w dia-



Rysunek 3.4. Rozkład wystąpień najczęstszych 1-gramów (wyrazów) w danych z EmoContext.



Rysunek 3.5. Rozkład wystąpień najczęstszych 2-gramów w danych z EmoContext.



Rysunek 3.6. Rozkład wystąpień najczęstszych 3-gramów w danych z EmoContext.

logach. Rysunki 3.4, 3.5 oraz 3.6 ukazują histogramy najczęstszych *N-gramów*. Można zauważyć że najczęściej występującymi słowami są słowa należące do grupy słów nie wnoszących znaczenia (ang. *stop words*), np.: słowa popularne, spójniki, przedimki, jednak często występujące słowa to także wnoszące dużo znaczenia do wypowiedzi takie jak *love*, *like*, *good*, *hate*.

Rozdział 4

Przetwarzanie danych

4.1 Wczytanie i oczyszczenie danych

Pierwszym etapem przygotowania danych do użycia w modelu jest ich wczytanie oraz przetworzenie. Zbiór danych z EmoContext zawiera pięć kolumn (rys. 3.1). W pierwszej kolumnie znajduje się identyfikator, w kolumnach od 2 do 4 zawarty jest docelowy tekst który będzie użyty jako wejście do modelu, a w ostatniej kolumnie znajduje się etykieta emocji. Po wczytaniu danych następuje połączenie trzech kolumn z tekstem w jeden ciąg znaków, oddzielone znakiem specjalnym *EOS* (ang. *end of sentence*). Jest to niezbędna operacja która umożliwi docelowemu modelowi oddzielić trzy wypowiedzi od siebie. Następnie na tak otrzymanym ciągu znaków wykonywane są operacje usuwania powtarzających się znaków interpunkcyjnych (np. *"!!!!"* zostanie zamienione na pojedynczy znak *"!"*). Na końcu wykonywane jest oddzielenie znaków interpunkcyjnych od wyrazów, usunięcie powtarzających się spacji oraz zamiana wszystkich dużych liter na małe.

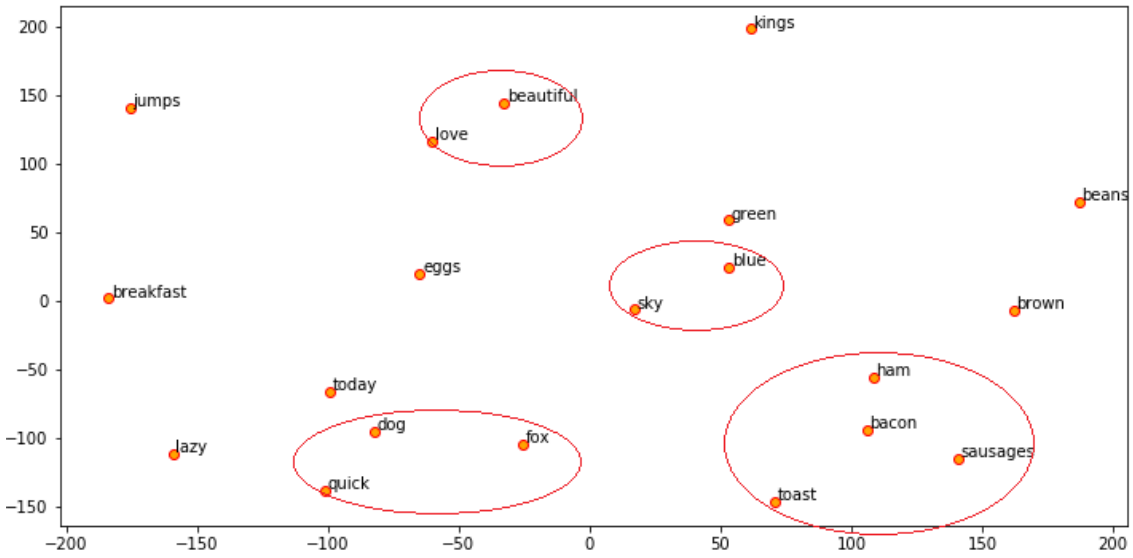
Jest to wspólny proces dla wszystkich modeli, kolejne etapy zostały podzielone na te zastosowane do architektur korzystających z rekurencyjnych komórek LSTM oraz tych korzystających z architektury BERT.

4.2 Przetwarzanie dla architektur LSTM

4.2.1 Tokenizacja i wyrównanie

Modele głębokie przetwarzania języka naturalnego nie operują na jawnym tekście w postaci ciągu znaków, tylko na reprezentacji w postaci liczbowej. Do reprezentacji wyrazów używane są słowniki które przechowują wszystkie znane słowa z korpusu uczącego. Do tej zamiany użyta została metoda `Tokenizer` z pakietu TensorFlow. Klasa ta pozwala na wektoryzację korpusu tekstu, poprzez przekształcenie każdego wyrazu w liczbę całkowitą. Każda z tych liczb odpowiada indeksowi tego słowa w słowniku.

Po przekształceniu każdego słowa w token następuje wyrównanie długości wszystkich przykładów w zbiorze. Przykładowa liczba użyta do wyrównania (ang. *padding*) to 20. Wszystkie przykłady, które mają mniejszą liczbę tokenów, wypełniane są tokenem zerowym aż do osiągnięcia długości 20, a wszystkie przykłady które mają większą liczbę

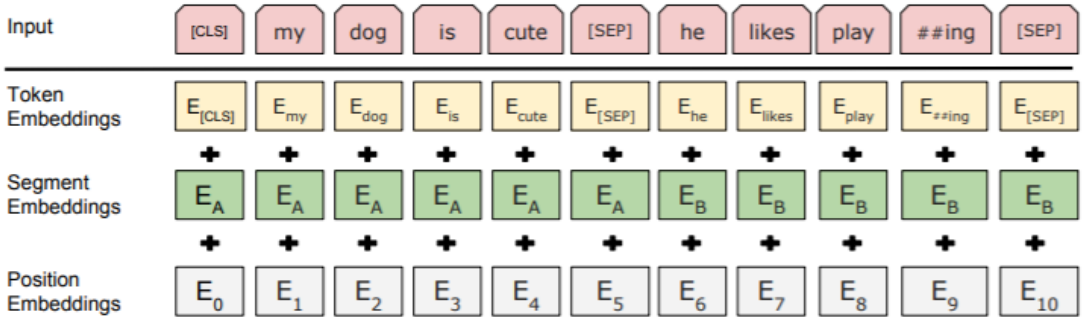


Rysunek 4.1. Przykładowe słowa w przestrzeni wektorowej GloVe [1].

tokenów są skracane od końca. W ten sposób została utworzona macierz danych która może być wykorzystana do użycia w nauce i ewaluacji modeli.

4.2.2 Zagłębienia słów

W przypadku głębokich sieci neuronowych z rekurencyjnymi komórkami LSTM wysoce efektywne jest użycie zagłębień słów (ang. *words embeddings*) jako wejścia do pierwszej warstwy sieci neuronowej. Osadzanie słów jest przeprowadzone za pomocą mapowania tych słów na wektory liczb rzeczywistych. Reprezentacja słów jako wektory liczb jest rozszerzeniem reprezentacji *jeden z N* (ang. *one hot encodings*), która jest używana w celu zwiększenia wydajności modeli NLP. Jest to także możliwość użycia transferu wiedzy w postaci wstępnie wytrenowanych zagłębień słów jako reprezentacji danych wejściowych. Jednym z algorytmów, który umożliwia stworzenie takiej reprezentacji jest metoda GloVe [1] (ang. *Global Vectors for Word Representation*), stworzona przez zespół z Uniwersytetu Stanford. Udostępniona przez nich macierz umożliwia użycie tej reprezentacji, jako odwzorowanie słów na wektory liczb rzeczywistych. Słowa zmapowane w tej przestrzeni mają zachowane pewne właściwości, odległość między nimi jest powiązana z podobieństwem semantycznym. Na rysunku 4.1 ukazane są przykładowe słowa i zachowane podobieństwo np. między słowami love i beautiful. Sposób uzyskania tej reprezentacji bazuje na współwystępowaniu danych słów w korpusie w otoczeniu które je definiuje. Główną intuicją tego modelu jest założenie, że proporcje prawdopodobieństwa współwystępowania słów mają potencjał do kodowania jakiejś formy znaczenia. Wynikiem tej metody są wektory słów które bardzo dobrze radzą sobie z zadaniami opartymi o podobieństwo, analogię, a także odkrywaniu semantyki emocjonalnej słów i wiele innych wymienionych w pakiecie *word2vec* [10].



Rysunek 4.2. Przykładowe zdanie zamienione na reprezentację BERT [3].

4.3 Przetwarzanie dla architektur BERT

Do utworzenia reprezentacji wejściowej dla architektury BERT niezbędne było dostosowanie się do wymagań dla tego modelu. Do przetworzenia tekstu na tokeny użyto klasę `BertTokenizer` z pakietu `transformers`. Umożliwia ona zamianę reprezentacji ciągu znaków na tokeny które są mapowane dzięki wbudowanemu słownikowi w architekturę BERT za pomocą algorytmu *WordPiece* [18].

4.3.1 Tokenizacja, dodatkowe tokeny i wyrównanie

Tokenizacja rozkłada się na kilka etapów, jednym z nich jest dodanie specjalnych tokenów oznaczających początek (*[CLS]*) oraz separację (*[SEP]*) kolejnych wypowiedzi, widocznych jako Input na rysunku 4.2. Następnie wydzielone tokeny mapowane są na identyfikatory za pomocą wbudowanego słownika. W ten sposób otrzymane reprezentacje zdań zostają wyrównane do danej szerokości i przekazane do wygenerowania maski uwagi o tej samej szerokości co wyrównane zdanie. Ma to na celu wskazanie na których miejscach w zdaniu występują tokeny należące do zdania za pomocą liczby 1. Te tokeny w zdaniu które były sztucznie dodane dla celów wyrównania otrzymują liczbę 0. Tak otrzymana reprezentacja zostanie przekazana na wejście modelu korzystającego z architektury BERT.

Rozdział 5

Budowa modeli

5.1 Jednowarstwowa architektura LSTM

Pierwszy model bazuje na zaproponowanej przez organizatorów konkursu jednowarstwowej architekturze sieci rekurencyjnej z komórkami LSTM. Zastosowanie tej budowy modelu zapewnia zdolność do uczenia się długoterminowych zależności przy jednoczesnym unikaniu długotrwałego problemu uzależnienia.

5.1.1 Przygotowanie wejścia modelu

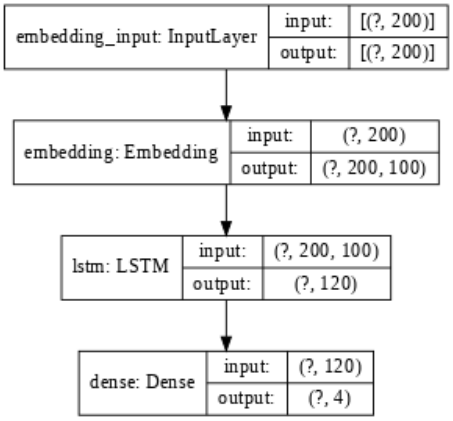
Przygotowawczym etapem modelu jest przedstawienie wymagań dla danych, które mogą być przetwarzane w warstwie wejściowej (ang. *Input Layer*). Każdy przykład jest przetworzony zgodnie z opisem w rozdziale 4. Jednym z ostatnich etapów przetwarzania jest wyrównanie wszystkich przykładów do równej długości w tym przypadku 200, zgodnie z rysunkiem 5.1 przedstawiającym budowę jednowarstwowej architektury LSTM.

Kolejną warstwą jest zamiana reprezentacji słowa na gęstą reprezentację wektorową opisaną w sekcji 4.2.2. Do uzyskania tej reprezentacji użyto gotowe macierze udostępnione przez autorów metody na stronie internetowej[1]. Do wyboru były macierze przygotowane na różnych zbiorach danych oraz o różnych szerokościach wektorów. Testy przeprowadzono z użyciem zagłębień słów wytrenowanych na zbiorze danych pochodzących z Wikipedii oraz z archiwum danych tekstowych Gigaword 5 (ang. *English Gigaword Fifth Edition*). Wybór szerokości wektorów podyktowany był złożonością modelu i czasem nauki, z pośród zbioru 50, 100, 200, 300 zdecydowano się na szerokość 100. Podsumowując złożoność tego etapu zawiera on prawie 2 miliony wewnętrznych parametrów (wag), które są zamrożone na czas uczenia. Rysunek 5.2 przedstawia szczegóły budowy jednowarstwowej architektury LSTM, wraz z tą warstwą o nazwie *Embedding*.

5.1.2 Rekurencyjne warstwy LSTM

Do stworzenia kolejnej warstwy użyto gotową reprezentację LSTM z pakietu tensorflow o nazwie `keras.layers.LSTM`, która udostępnia możliwość definiowania konkretnych szczegółów implementacyjnych. W warstwie tej użyto domyślną funkcję aktywacji jakim

[1] `https://nlp.stanford.edu/projects/glove/`



Rysunek 5.1. Graf przedstawiający budowę jednowarstwowej architektury LSTM.

```
Model: "Jednowarstwowa architektura LSTM"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 100)          1683200
_________________________________________________________________
lstm (LSTM)                  (None, 120)               106080
_________________________________________________________________
dense (Dense)                (None, 4)                 484
=================================================================
Total params: 1,789,764
Trainable params: 106,564
Non-trainable params: 1,683,200
_________________________________________________________________
```

Rysunek 5.2. Tabela przedstawiająca szczegóły budowy jednowarstwowej architektury LSTM.

jest tangens hiperboliczny. Po obserwacji krzywej uczenia reprezentującej dokładność oraz wartość funkcji straty zauważono zjawisko przeuczenia. Dokładność dla zbioru uczącego rosła wraz z malejącą dokładnością dla zbioru walidacyjnego. Z tego powodu zdecydowano się na użycie metody przerywania (ang. *dropout*), zdefiniowanej w tej warstwie. Pomogło to zmniejszyć wpływ zjawiska przeuczania się modelu.

5.1.3 Warstwa gęsta

Ostatnim etapem jest wybór odpowiedniej etykiety emocji za pomocą warstwy gęstej. Wyjście z poprzedniej warstwy nazywane stanem ukrytym (ang. *hidden state*) o szerokości 120 jest gęsto połączone z 4 neuronami decydującymi odpowiednio o każdej z kolejnych etykiet emocji. Jako funkcję aktywacji tej warstwy, zgodnie z sugestią organizatorów konkursu użyto funkcję sigmoidalną. Zauważona jednak została niekompatybilność tej funkcji z konkretnym zastosowaniem dla klasyfikacji wieloklasowej co zostało poprawione w kolejno zaprezentowanych modelach.

5.1.4 Funkcja straty oraz algorytm optymalizacyjny

Końcowym elementem budowy tego modelu jest wybór algorytmu optymalizacyjnego. Wybrano algorytm RMSprop [14] (ang. *Root Mean Square Propagation*), który dobrze radzi sobie z wygasającymi wskaźnikami uczenia oraz przeciwdziała obliczeniowym pro-

blemom numerycznym. Jako funkcję straty użyto *Categorical Cross-Entropy loss*, która bardzo dobrze radzi sobie w problemach klasyfikacji wielu klas.

5.2 Głęboka architektura LSTM

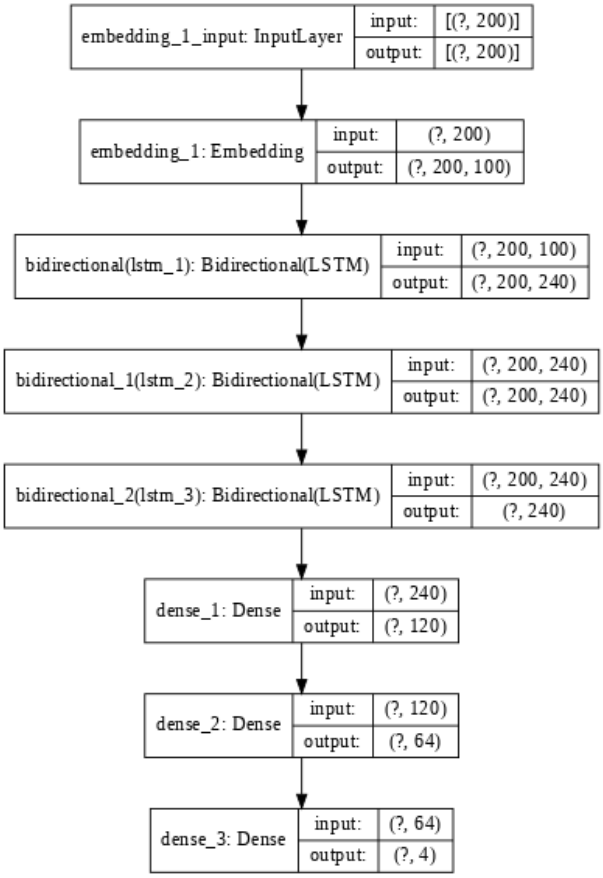
Głęboka architektura LSTM jest rozszerzeniem architektury jednowarstwowej opisanej w punkcie 5.1. Głównym celem było porównanie złożoności architektury płytkiej i głębokiej oraz wpływ rozszerzenia modelu o kolejne warstwy na wynik. Modyfikacje polegały przede wszystkim na dodaniu kolejnych warstw oraz lekkie modyfikacje parametrów oraz funkcji wewnątrz modelu. Pierwsze dwie warstwy, wejściowa (*Input Layer*) oraz mapująca słowa na gęstą reprezentację wektorową (*Embedding*) zostały bez zmian.

5.2.1 Rozszerzenie warstw LSTM

Do rozszerzenia jednowarstwowej części sieci rekurencyjnej zamiast zwykłych komórek LSTM użyto dwukierunkowych komórek LSTM. Z założenia rozszerzenie to powinno poprawić wydajność modelu przy problemach z klasyfikacją sekwencji [4]. W tym przypadku można było użyć tego rozszerzenia ponieważ od razu są dostępne wszystkie sekwencje czasowe sekwencji wejściowej. W tym momencie dwukierunkowe komórki LSTM łączą dwie ukryte warstwy o przeciwnych kierunkach. Dzięki tej formie uczenia warstwa wyjściowa może jednocześnie uzyskiwać informacje z przeszłych jak i przyszłych stanów, co nie było możliwe przy użyciu podstawowych komórek LSTM. Zabieg ten pomaga lepiej zrozumieć kontekst gdyż znaczenie danego słowa może zależeć także od słów, które są przed danym słowem. Dodatkowo zamiast jednej warstwy sieci rekurencyjnej użyto łącznie trzy warstwy używające dwukierunkowego LSTM. Pierwsze dwie warstwy na wyjściu oprócz ukrytego stanu zwracają także sekwencję o długości 200, którą przekazują na wejście kolejnej warstwy sieci LSTM. Widać to na rysunku 5.3, który przedstawia graf głębokiej architektury LSTM.

5.2.2 Rozszerzenie warstw gęstych

Kolejną modyfikacją było dodanie kolejnych warstw sieci gęstej. Wcześniej użytą jedną warstwę zawierającą 4 neurony wyjściowe rozszerzono o kolejne dwie warstwy gęste. Pierwsza z nich zawiera 120 neuronów a druga zawiera 64 neurony. Szczegóły tej budowy przedstawiono na rysunku 5.4, który przestawia tabelę prezentującą szczegóły budowy głębokiej architektury LSTM. Do dodanych warstw użyto innej funkcji aktywacji jaką jest jednostronnie obcięta funkcja liniowa (ang. *ReLU*), która stała się standardem do użycia w wewnętrznych warstwach gęstej sieci neuronowej [19]. W ostatniej warstwie także zamieniono funkcję aktywacji na funkcję *softmax*, która lepiej nadaje się do klasyfikacji wieloklasowej.



Rysunek 5.3. Graf przedstawiający budowę głębokiej architektury LSTM.

Tablica 5.1
Tabela porównująca szczegóły budowy poszczególnych model.

model	parametry trenowalne	parametry stałe	SUMA
Jednowarstwowy LSTM	106,564	1,683,200	1,789,764
Głęboki LSTM	942,204	1,683,200	2,625,404

5.3 Porównanie modeli LSTM

Porównując dokonane rozszerzenia modelu z jedną warstwą LSTM, oraz wykonując opisane modyfikacje warstw zwiększyła się złożoność modelu. Prostym sposobem na porównanie modeli jest sprawdzenie liczby parametrów, które definiują zachowanie sztucznych neuronów, inaczej nazywane wagami neuronu. W tabeli 5.1 przedstawione są liczby parametrów podzielonych na dwie grupy. Parametry trenowalne modelu to są wagi, które ulegają modyfikacji, liczba tych parametrów zwiększyła się około dziewięciokrotnie co ilustruję skalę zmian. Parametry stałe to są wagi użyte do generacji gęstych reprezentacji wektorowych, które były zamrożone na czas nauki modeli. Zwiększona złożoność modelu głębokiego powinna w jakimś stopniu przełożyć się na większe możliwości przechowywania i odkrywania wiedzy zawartej w danych co powinno umożliwić osiągnięcie lepszego wyniku dla tego modelu.

```
Model: "Głęboka architektura LSTM"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 200, 100)          1683200
_________________________________________________________________
bidirectional (Bidirectional (None, 200, 240)          212160
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200, 240)          346560
_________________________________________________________________
bidirectional_2 (Bidirection (None, 240)               346560
_________________________________________________________________
dense_1 (Dense)              (None, 120)               28920
_________________________________________________________________
dense_2 (Dense)              (None, 64)                7744
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 260
=================================================================
Total params: 2,625,404
Trainable params: 942,204
Non-trainable params: 1,683,200
_________________________________________________________________
```

Rysunek 5.4. Tabela przedstawiająca szczegóły budowy głębokiej architektury LSTM.

5.4 Głęboka architektura BERT

Do zaprojektowania architektury korzystającej z BERT użyto modelu udostępnionego przez zespół o nazwie Hugging Face [17]. Udostępnione modele są wstępnie przeszkolone na masowych zbiorach danych. Umożliwia to wykorzystanie tej metody jako łatwo dostępnego komponentu, oszczędzając czas i zasoby niezbędne do wytrenowania tak dużego modelu od podstaw. Do użycia tego modelu w problemie klasyfikacji tekstu należało wprowadzić kilka modyfikacji. Główną zmianą jest dodanie ostatniej warstwy klasyfikatora w postaci metody softmax. Następnie możliwe było użycie procesu szkoleniowego jakim jest *Fine-Tuning* [15], który polega na dostrajaniu modelu przy jednoczesnej nauce klasyfikatora przy użyciu własnego zbioru danych.

5.4.1 Rozmiar modelu BERT

Model BERT występuje w dwóch rozmiarach, udostępnionych przez twórców tej metody. Są to rozmiary *BASE* (12 koderów) oraz *LARGE* (24 kodery). Pomimo faktu że modele są wstępnie wytrenowane posiadają bardzo dużo parametrów (wag), w przypadku rozmiaru *BASE* jest to 110 milionów parametrów, a dla *LARGE* jest to 340 milionów parametrów.

Na tak dużą liczbę parametrów w modelu *LARGE* składa się dwa razy większa liczba koderów oraz większe rozmiary warstw poszczególnych koderów. Sieć wewnętrzna posiada 1024 jednostki ukryte, zamiast 768 oraz porównywalnie większy wektor reprezentujący zdanie, nazwany zagłębieniami. Z powodu ograniczeń zasobowych oraz czasowych zdecydowano się na mniejszą z zaprezentowanych architektur, która i tak dała satysfakcjonujące wyniki. Zastosowanie większego modelu nie było możliwe z powodu ograniczeń pamięciowych. Szczegóły i porównanie tych modeli ukazuje tabela 5.2.

Tablica 5.2
Tabela porównująca rozmiary poszczególnych modeli BERT.

model	parametry	warstwy	ukryte	zagłębienia
BERT BASE	110M	12	768	768
BERT LARGE	340M	24	1024	1024

```
(embeddings): BertEmbeddings(
  (word_embeddings): Embedding(30522, 768, padding_idx=0)
  (position_embeddings): Embedding(512, 768)
  (token_type_embeddings): Embedding(2, 768)
  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
```

Rysunek 5.5. Szczegóły budowy pierwszej warstwy wejściowej BERT.

5.4.2 Przygotowanie wejścia do modelu

Pierwsza warstwa modelu przyjmuje na wejście specjalnie przygotowaną reprezentację tekstu przedstawioną w sekcji 4.3. Szerokość wektora reprezentującego każde słowo to 768, jest on używany w każdym z 12 koderów dla poszczególnych słów. W ten sposób uzyskana jest macierz rozmiaru (512, 768), która reprezentuje wejściowe zdanie w każdej z kolejnych warstw.

Blok ta zawiera element normalizacji warstwy dla każdej wejściowej grupy (batch), który umożliwia skrócenie czasu treningu. Dodatkowo warstwa ta zawiera metodę przerywania (*dropout*), która zapobiega przetrenowaniu modelu. Szczegóły reprezentacji tej warstwy pokazuje rysunek 5.5, który przedstawia blok o nazwie `BertEmbeddings` zawierający przedstawione elementy.

5.4.3 Wewnętrzne warstwy kodera

Koder zastosowany w BERT to architektura przetwarzania języka naturalnego zaprezentowana w modelu Transformera. Łącznie 12 bloków kodera jest połączonych w jeden duży blok aby wygenerować na wyjściu wektor z zakodowaną informacją. Cały ten blok odpowiedzialny jest za znalezienie relacji pomiędzy reprezentacjami wejściowymi i kodowanie ich na swoim wyjściu.

Model ten reprezentuje każdy token jako reprezentacja wektorowa o szerokości 768, dodatkowo zawarta jest także informacja o położeniu każdego tokena w zdaniu. Każdy koder zawiera mechanizm samoobserwacji `BertAttention`, który oblicza uwagę dla dla każdego tokenu a następnie łączy te wyniki przekazując je do sieci neuronowej `BertOutput`. Podczas tego kroku wektorowe reprezentacje tokenów nie mają wpływu na siebie nawzajem, występuje to tylko w mechanizmie samoobserwacji.

Pomiędzy dwoma głównymi elementami kodera występuje element normalizujący o nazwie `BertSelfOutput`. Dla każdej z tych warstw stosowana jest także metoda przerywania (*dropout*) o wartości 0.1. Normalizacja obliczana jest na podstawie średniej i odchylenia standardowego każdego wiersza macierzy, robiona jest w celu poprawy stabilności sieci neuronowej. Budowa tego bloku zaprezentowana jest na rysunku 5.6.

```
(layer): ModuleList(
  (0): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
```

Rysunek 5.6. Szczegóły budowy kodera BERT.

```
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=4, bias=True)
```

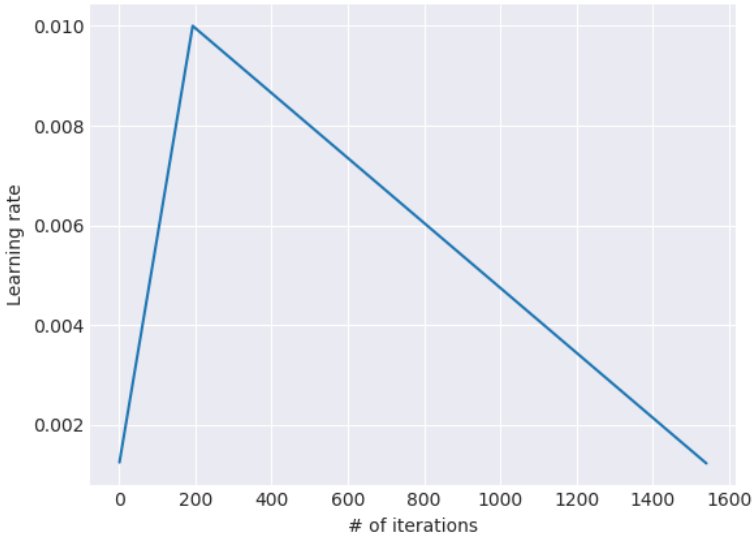
Rysunek 5.7. Szczegóły budowy ostatniej warstwy (klasyfikatora) BERT.

5.4.4 Budowa klasyfikatora

Ostatnim blokiem jest warstwa mająca na celu klasyfikację odpowiedniej etykiety emocji. Wejściem do klasyfikatora jest wektor wyjściowy z kodera zawierającego informacje o tekście zebrane w fazie kodowania. Pierwszym elementem tego bloku jest warstwa `BertPooler`, mająca na celu pobranie reprezentacji wyjściowej i wykorzystanie jej do zadania klasyfikacji. Następnie warstwa liniowa jest odpowiedzialna za wybranie odpowiedniej emocji. Szczegóły tej budowy przedstawia rysunek 5.7.

Algorytmem optymalizacyjnym wykorzystanym w tym modelu jest rozszerzony algorytm Adam [7] o nazwie `AdamW`. Jest to algorytm stochastyczny oparty na adaptacyjnych oszacowaniach momentów pierwszego rzędu, rozszerzony o mechanizm korekty rozpadu masy (ang. *weight decay*).

Dodatkowo użyty został mechanizm skośnych trójkątnych współczynników uczenia (*STLR*). Pozwala on na dynamiczny wzrost wartości wskaźnika uczenia wraz z kolejnymi iteracjami po czym następuje liniowy spadek tej wartości. Zabieg ten ma na celu lepszą eksplorację przestrzeni w początkowych fazach treningu oraz zapobiega utracie informacji w końcowych fazach treningu. Wykres wartości wskaźnika uczenia przedstawiono na rysunku 5.8.



Rysunek 5.8. Skośne trójkątne wskaźniki uczenia w BERT.

Rozdział 6

Ewaluacja modeli

6.1 Dobór parametrów

Proces nauki poszczególnych modeli rozpoczęto od zdefiniowania *hiper-parametrów* [13]. Są to parametry, których wartości są ustawione przed rozpoczęciem procesu uczenia się i definiują one zachowanie poszczególnych warstw modelu. Początkowe wartości tych parametrów zdefiniowano zgodnie z powszechnymi wartościami używanymi w podobnych architekturach oraz na podstawie załączonej literatury. Dalsze kroki polegały na dostosowaniu tych parametrów do osiągnięcia najlepszego wyniku. Dobrym sposobem na znalezienie optymalnego doboru parametrów są metody typu wyszukiwanie w siatce (ang. *grid search*) oraz wyszukiwanie losowe (ang. *random search*) [9]. Polegają one na porównywaniu działania modelu z przyjętymi parametrami wybranymi z przestrzeni parametrów. Podjęto próby użycia tych metod, jednak ze względu na ograniczone zasoby obliczeniowe oraz długi czas przetwarzania ograniczono przestrzeń parametrów do minimum. Pozwoliło to na przeszukanie bardzo małego zbioru wartości co i tak dało satysfakcjonujące wyniki.

6.1.1 Parametry modeli LSTM

Dobrane wartości dla modeli jednowarstwowego LSTM 5.1 oraz głębokiego LSTM 5.2 przedstawiono w tabeli 6.1. Założeniem użycia tych samych hiper-parametrów dla obu modeli było sprawdzenie wpływu głębokości (liczby warstw) modelu na wynik. Rozmiar LSTM dotyczy liczby komórek LSTM w jednej warstwie, dobór wartości 120 zależał od wielkości zbioru danych oraz od ograniczeń zasobowych. Parametr przerywania (ang. *dropout*) użyty do przeciwdziałania przeuczania się modelu został wybrany z pośród dwóch wartości 0.2 oraz 0.5, okazało się że 0.2 ma lepszy wpływ na wynik. Rozmiar wektora słów dotyczy szerokości gęstej reprezentacji wektorowej każdego słowa, czym większa wartość tym więcej informacji przenoszonej przez te wektory, jednak ograniczenia zasobów pozwoliły na szerokość 100, gdzie maksymalna dostępna szerokość to 300. Pozostałe parametry dotyczą sposobu nauki modeli oraz szybkości i długości tego procesu, wartości te pozostały domyślne.

Tablica 6.1
Tabela prezentująca wybrane *hiper-parametry* modeli korzystających z LSTM.

hiper-parametr	wartość
rozmiar LSTM	120
wskaźnik uczenia się	0.003
przerywanie (ang. *dropout*)	0.2
liczba epok	100
rozmiar wektora słów	100
wielkość grupy (ang. *batch*)	200

Tablica 6.2
Tabela prezentująca wybrane *hiper-parametry* modeli korzystających z BERT.

hiper-parametr	wartość
Max. długość zdania	128
wskaźnik uczenia się	0.00002
przerywanie (ang. *dropout*)	0.1
liczba epok	4
wielkość grupy (ang. *batch*)	32

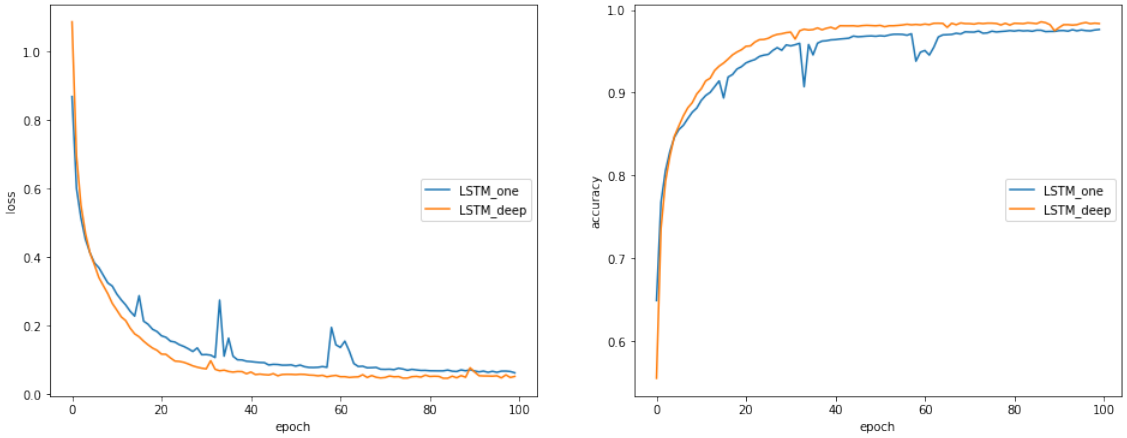
6.1.2 Parametry modeli BERT

Wybór parametrów dla architektury korzystającej z BERT opierał się głównie na sugestiach autorów tego modelu oraz doświadczeniach prowadzonych przez innych naukowców [15]. Na czas nauki parametry wewnętrzne modelu pozostały bez zmian, konfigurowano jedynie niektóre z nich do uzyskania jak najlepszego wyniku. Wartości te przedstawiono w tabeli 6.2. Maksymalna szerokość wejściowego zdania dla BERT to 512, jednak ze względu ograniczeń zasobów wybrano 256. Najdłuższe zdanie występujące w korpusie miało szerokość 168 co pozwalało ograniczyć w pewien sposób zasoby pamięciowe. Kolejne dwa parametry, wskaźnik uczenia się (0.00002) oraz przerywanie (0.1) pozostawiono domyślne. Liczba epok została wybrana poprzez obserwację krzywej uczenia się oraz ewaluacji modelu na zbiorze testowym po każdej epoce. Najlepszy wynik uzyskano dla liczby epok równej 4, była to także sugerowana wartość przez autorów modelu. Po większej liczbie epok zauważono zjawisko przeuczania się, metryki dla zbioru uczącego były coraz lepsze jednak jakość dla zbioru walidacyjnego spadała. Ostatnim parametrem jest wielkość grupy (ang. *batch size*), parametr ten został wybrany z pośród dwóch sugerowanych: 16 oraz 32. Dla 32 osiągnięto lepszy wynik dlatego został on wybrany.

6.2 Przebieg nauki

Do procesu nauki użyto platformę *Colab*[1], która daje możliwość korzystania z zasobów karty graficznej (GPU) umożliwiającej dużo wydajniejszy proces nauki modeli. Dla jednowarstwowego LSTM czas przetwarzania jednej epoki to około 9 sekund, a dla modelu głębokiego LSTM to 64 sekundy. Przy użyciu laptopa czasy te były prawie dziesięciokrotnie większe co uniemożliwiło by tak sprawne projektowanie i dostrajanie modeli.

[1] `https://colab.research.google.com/`



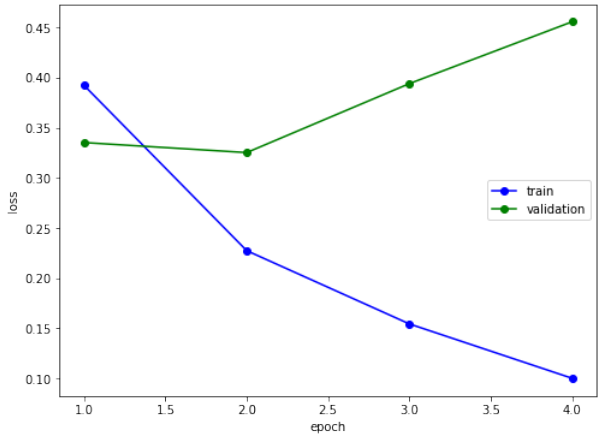
Rysunek 6.1. Wykresy przedstawiające przebieg nauki dla modeli jednowarstwowego LSTM oraz głębokiego LSTM.

6.2.1 Nauka modeli LSTM

Proces nauki rozpoczęto od podstawowego modelu jednowarstwowego LSTM. Po doborze hiper-parametrów związanych z architekturą modelu dobrane były odpowiednie wartości dla liczby epok oraz wskaźnika uczenia się. Zauważono że po dwudziestej epoce dokładność przekroczyła 90%, a po setnej epoce poprawa wyników była na tyle niewielka że przerwano proces nauki. Ostateczne wyniki są przedstawione na wykresach 6.1 przedstawiających przebieg nauki dla modeli jednowarstwowego LSTM oraz głębokiego LSTM. W początkowych etapach nauki wartości funkcji straty (ang. *loss*) oraz dokładności (ang. *accuracy*) były bardzo podobne, jednak po dziesiątej epoce widać przewagę modelu głębokiego LSTM oznaczonego kolorem pomarańczowym. Wynikać to może z dużo większej złożoności tego modelu oraz możliwości zapamiętania przez ten model większej liczby cech przechowywanych w tekście.

6.2.2 Nauka modeli BERT

Proces nauki architektury korzystającej z BERT rozpoczęto od wczytania wstępnie wyuczonych modeli, był to `BertTokenizer` służący do przygotowania wejścia do modelu `BertForSequenceClassification`. Po przetworzeniu danych wejściowych oraz doborze hiper-parametrów proces nauki wyglądał następująco. Dla każdej epoki wprowadzano model w tryb treningu (`model.train()`), a podczas walidacji wyniku w tryb ewaluacji (`model.eval()`). Umożliwiało to przełączanie niektórych warstw w tryb nauki oraz dostosowanie działania przerywania w modelu. Następnie dla każdej grupy obliczano stratę (ang. *loss*), oraz wykonywano obliczenie gradientów dla wszystkich parametrów (wag), po czym następował krok optymalizacyjny. Przebieg nauki zaprezentowano na rysunku 6.2, gdzie zauważalny jest proces przeuczenia, stąd wybrana została liczba epok równa 4.



Rysunek 6.2. Wykres przedstawiający przebieg nauki dla modeli BERT.

6.3 Metryki

6.3.1 Macierz pomyłek

W dziedzinie sieci neuronowych oraz oceny jakości modeli uczenia maszynowego, a w szczególności w problemach klasyfikacji często pomocna jest macierz pomyłek, inaczej nazywana macierzą błędów. Jest to specyficzny układ tabel, który pozwala na wizualizację wydajności algorytmu. Składa się on z dwóch wymiarów: rzeczywistym i przewidywanym. Podaje on liczbę wyników prawdziwie dodatnich (TP), prawdziwie ujemnych (TN), fałszywie dodatnich (FP) oraz fałszywie ujemnych(FN).

Tablica 6.3
Tabela ukazująca macierz pomyłek.

		Prawdziwa etykieta	
		Positive	Negative
Przewidziana etykieta	Positive	TP	FP
	Negative	TN	FN
	Łącznie	P	N

Pozwala to na bardziej szczegółową analizę oraz formułowanie wielu metryk na podstawie wartości z tej macierzy. Niektóre z tych miar zastosowano do ewaluacji modeli i zaprezentowano w kolejnych podsekcjach.

6.3.2 Miara jakości nauki

Główną metryką analizowaną w trakcie nauki modelu była dokładność (ang. *accuracy*), zdefiniowana w poniższym równaniu 6.1.

$$dokladnosc = \frac{TP + TN}{P + N} \tag{6.1}$$

Miara ta odwzorowuje efektywność nauki. Podczas jej obserwacji można stwierdzić czy każda kolejna epoka nauki poprawia wynik, czy też go pogarsza. Zbiór danych treningowy miał zrównoważony rozkład klas na tyle, że zastosowanie tej metryki miało sens. Niestety jeśli zestaw danych jest mocno niezbalansowany, tak jak w przypadku zbioru testowego

miara ta może wprowadzić w błąd. Z tego powodu zastosowanie tej metryki ograniczono tylko do obserwacji procesu nauki, a ostateczną ocenę modelu wykonano na podstawie innych miar.

6.3.3 Ostateczna ocena modelu

Ostateczną ocenę modelu dokonano za pomocą obliczenia mikro średniego wyniku F1 dla trzech klas emocji *Happy*, *Sad*, *Angry*. Wybór ten wynika z niezbalansowania klas w zbiorze testowym oraz dominacją występowania etykiety *Others*, którą pominięto w ostatecznej ocenie. Wynik F1 jest średnią harmoniczną precyzji i czułości, zdefiniowanych w poniższych wzorach. Końcowy wynik uzyskany jest poprzez uśrednianie w skali mikro, na podstawie częstotliwości klas występujących w zbiorze treningowym. Poniższe wzory przedstawiają poszczególne składniki oraz samą funkcję F1.

$$precyzja = \frac{TP}{FP + TP} \tag{6.2}$$

$$czulosc = \frac{TP}{FP + TP} \tag{6.3}$$

$$F1 = \frac{2 \cdot precyzja \cdot czulosc}{precyzja + czulosc} \tag{6.4}$$

6.4 Wyniki

Jakość wyuczonych modeli na zbiorze treningowym porównano poprzez ewaluację predykcji tych modeli na zbiorze testowym. Głównymi założeniami porównań było sprawdzenie wpływu liczby warstw sieci neuronowej na wynik oraz porównanie tradycyjnych rekurencyjnych warstw do bardziej złożonych architektur jakim jest BERT.

Porównanie jednowarstwowej architektury LSTM z głęboką architekturą LSTM pokazuje przewagę dla modelu głębokiego. Większa liczba warstw pomogła osiągnąć lepszy wynik miary mikro F1, jednak jest to niewielka poprawa. Można spekulować że jednowarstwowa sieć LSTM dla tego zbioru danych była wystarczająca aby osiągnąć maksimum możliwości tej architektury. Z drugiej strony użycie gęstej reprezentacji wektorowej każdego słowa mogło ograniczyć możliwości warstw LSTM i spowodować, że użycie kolejnych warstw sieci pomogło na zniwelowanie tylko małych błędów. Niestety analiza wpływu na wynik poszczególnych elementów sieci neuronowej jest utrudniona co uniemożliwia wyciągnięcie jasnych i sprawdzonych wniosków.

Porównanie architektury BERT z pozostałymi daje dużą przewagę użycia tej architektury do zadań, w których niezbędne jest przetwarzanie tekstu. Dużą przewagą oraz zaletą tej architektury jest fakt że może być ona użyta to różnych problemów, zmieniając tylko jedną z pierwszych lub ostatnich warstw. W obecnym zadaniu klasyfikacji architektura BERT sprawdziła się bardzo dobrze osiągając najlepszy wynik miary F1 (0.67).

Tablica 6.4
Tabela pokazująca wyniki poszczególnych modeli.

model	dokładność	mikro precyzja	micro czułość	**mikro F1**
Jednowarstwowy LSTM	0.85	0.49	0.70	0.58
Głęboki LSTM	0.87	0.51	0.71	0.60
BERT	**0.90**	**0.57**	**0.81**	**0.67**

Rozdział 7

Podsumowanie

Celem pracy było opracowanie modelu uczenia maszynowego opartego na głębokich sieciach neuronowych w celu detekcji emocji w dialogach. W jego ramach zostały zaprojektowane oraz przetestowane trzy różne architektury sieci neuronowych oraz została przeprowadzona analiza porównawcza tych modeli. Ponad to wszystkie zadania, które były niezbędne do zrealizowania tego celu także zostały wykonane. Są to między innymi zadania zapoznania się z literaturą dotyczącą głębokich sieci neuronowych i modeli emocji, przegląd oraz zapoznanie się z dostępnymi frameworkami do uczenia głębokiego, wstępna analiza wybranego zbioru danych oraz ewaluacja stworzonych modeli.

Rozpoznawanie emocji z tekstu jest zadaniem, które wymaga wielu czynników do poprawnego zrozumienia danego przekazu. Aby móc odkryć właściwą etykietę emocji należy spojrzeć na przekaz jako na całość, a nie na pojedyncze wyrazy, które mogą nieść inne znaczenie w różnych kontekstach. W podobny sposób objawia się działanie przedstawionych architektur do zrozumienia języka pisanego. Tradycyjne sieci neuronowe, proste metody uczenia maszynowego lub metody zliczające nie są w stanie odkryć znaczenia wypowiedzi jako całości. Bardzo ważnym elementem jest zrozumienie kontekstu oraz poradzenie sobie z problemami takimi jak występowanie sarkazmu lub ukrytego znaczenia.

Architektury korzystające z rekurencyjnych komórek LSTM są skonstruowane do rozumienia długotrwałych zależności i połączeń między wyrazami występującymi w zdaniu. Dzięki nim można było odkryć znaczenie przekazu, które zależało od kilku poprzednich wypowiedzi. Jeszcze lepsza okazała się architektura korzystająca z modelu BERT. Zastosowane w niej mechanizmy samoobserwacji oraz złożona budowa pozwoliła na jeszcze dokładniejsze zrozumienie tekstu.

Największym problemem w trakcie realizacji projektu okazały się ograniczenia zasobowe oraz długi czas przetwarzania modeli. Korzystanie z lokalnego laptopa do przeprowadzenia obliczeń oraz nauki modeli okazał się procesem zbyt długotrwałym oraz powodującym dużą zajętość zasobów takich jak pamięć oraz procesor. Pomocne okazały się serwisy udostępniające moce obliczeniowe w chmurze. Platforma Colab, użyta do trenowania modeli udostępniała dodatkowo dostęp do kart graficznych umożliwiających jeszcze efektywniejsze trenowanie modeli.

Do realizacji pracy użyto tylko niektórych z najbardziej popularnych metod głębokiego uczenia. W trakcie realizacji pracy wydano najnowszą architekturę określaną jako

nowy stan techniki w przetwarzaniu języka naturalnego *GPT-3* [2]. Do uzyskania jeszcze lepszych wyników warto by przetestować działanie podobnych architektur oraz poświęcić więcej czasu na lepszym doborze parametrów. Istnieje także możliwość rozwijania obecnych architektur o inne zbiory danych oraz dostosowanie ich do użycia w praktyce w prawdziwych systemach informatycznych.

Literatura

[1] Robin Brochier, Adrien Guille, and Julien Velcin. Global vectors for node representations, 2019.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, and Melanie Subbiah. Language models are few-shot learners, 2020.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[4] Zixiang Ding, Rui Xia, Jianfei Yu, Xiang Li, and Jian Yang. Densely connected bidirectional lstm with applications to sentence classification, 2018.

[5] Paul Ekman. Facial expression and emotion, 1993.

[6] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[8] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura E. Barnes, and Donald E. Brown. Text classification algorithms: A survey, 2019.

[9] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas, 2019.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[11] Robert Plutchik. A psychoevolutionary theory of emotions, 1982.

[12] Soujanya Poria, Navonil Majumder, Rada Mihalcea, and Eduard Hovy. Emotion recognition in conversation: Research challenges, datasets, and recent advances, 2019.

[13] Philipp Probst, Bernd Bischl, and Anne-Laure Boulesteix. Tunability: Importance of hyperparameters of machine learning algorithms, 2018.

[14] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[15] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification?, 2019.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[17] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, and Clement Delangue. Huggingface's transformers: State-of-the-art natural language processing, 2019.

[18] Yonghui Wu, Mike Schuster, Zhifeng Chen, and Quoc V. Le. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[19] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[20] Peixiang Zhong, Di Wang, and Chunyan Miao. Knowledge-enriched transformer for emotion detection in textual conversations, 2019.

Dodatek A

Płyta CD

Płyta CD z elektroniczną wersją pracy, projektem oraz danymi wejściowymi.

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu LaTeX na platformie Overleaf.