Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ

Praca inżynierska

Oprogramowanie do analizy tekstów oparte o algorytmy uczenia maszynowego

Autor: *Alicja Zoń*

Kierunek studiów: *Informatyka Stosowana*

Opiekun pracy: *dr hab. inż. Tomasz Szumlak*

Kraków, 2019

Uprzedzona o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzona o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałam osobiście i samodzielnie i że nie korzystałam ze źródeł innych niż wymienione w pracy.

Ocena merytoryczna opiekuna

Ocena merytoryczna recenzenta

Spis treści

1. Wstęp

1.1 Wprowadzenie

Istotną częścią naszego codziennego życia jest pozyskiwanie informacji, choć często bardziej interesujące od faktów bywają opinie. Wraz z dynamicznym rozwojem Web 2.0, czyli sieci socjalnej, nastawionej głównie na treści generowane przez użytkowników portali, zwiększa się potrzeba kategoryzacji tychże tekstów. Wiedza o tym, jaka jest publiczna opinia na temat jakiegoś produktu, osoby bądź miejsca może znacznie ulepszyć strategie biznesowe, polityczne i marketingowe. Manualne przetworzenie tak dużej ilości danych jest bardzo czasochłonne, stąd konieczność stosowania systemów do ich automatycznej analizy.

Celem tej pracy jest stworzenie rozwiązania do analizy opinii w krótkich tekstach. W założeniu program ma określać, czy dane zdanie ma wydźwięk pozytywny, negatywny, czy neutralny. Motywacją do zbadania tego właśnie tematu jest chęć praktycznego zastosowania uczenia maszynowego w dziedzinie, która cieszy się popularnością i daje duże możliwości rozwoju.

1.2 Zawartość pracy

- **Rozdział 2:** zawiera opis poruszanego problemu wraz z nakreśleniem możliwych podejść do jego rozwiązania. Przedstawiono uzasadnienie wyboru konkretnego podejścia oraz aktualny stan badań w tej dziedzinie (tzw. *state of the art*).
- **Rozdział 3:** przedstawiono w nim źródło, z którego zostały zaczerpnięte dane do uczenia. Opisany jest sposób reprezentacji danych i etykietowania zdań.
- **Rozdział 4:** opisuje modele teoretycznie możliwe do zastosowania w badanym problemie. Przedstawiony jest krótki opis i analiza tych modeli oraz ostateczna postać wybranego modelu wraz z wzorami i uzasadnieniem, dlaczego najbardziej pasuje do problemu opisywanego w tej pracy.
- **Rozdział 5:** wymieniono w nim użyte technologie i narzędzia wraz z ich zaletami i uzasadnieniem wyboru.

- **Rozdział 6:** opisuje implementację modelu z rozdziału czwartego. Przedstawiony jest również dobór parametrów wraz z porównaniem wyników dla różnych wartości, a także regularyzacja modelu i korzyści z niej płynące.
- **Rozdział 7:** przedstawione są tu wyniki najlepszego wytrenowanego modelu wraz z wykresami poszczególnych metryk i analizą jakości klasyfikacji.
- **Rozdział 8:** zawiera analizę konkretnych przypadków wraz z wynikiem ich dopasowania. Opisane są zarówno przykłady prawidłowych dopasowań, jak i błędy pierwszego i drugiego rodzaju.
- **Rozdział 9:** przedstawia krótką dyskusję dotyczącą napotkanych problemów. Przedstawiono również możliwości rozwoju programu.
- **Rozdział 10:** wnioski z pracy w kontekście realizacji jej celu i możliwości dalszego wykorzystania uzyskanych wyników.

1.3 Aspekt twórczy i badawczy w pracy

Rozdziały od szóstego do dziewiątego stanowią część twórczo-badawczą, która opracowuje metody naukowe przedstawione w części teoretycznej pracy. Kod programu będącego przedmiotem analizy w tej pracy został stworzony w oparciu o źródła wymienione w bibliografii.

2. Opis problemu

Badanie opinii i emocji w tekstach (*ang. sentiment analysis*) jest zadaniem dość trudnym, gdyż nawet ludzie różnie odbierają wydźwięk tych samych zdań. Dlatego stworzenie precyzyjnego klasyfikatora wydźwięku tekstu to duże wyzwanie, zwłaszcza że w zdaniach istotne są nie tylko słowa, ale także ich kolejność i kontekst.

Wydźwięk tekstów może być analizowany w różnych skalach, w tej pracy przyjęte zostało podejście najbardziej podstawowe, czyli wykrywanie biegunowości (*ang. polarity detection*), będące klasyfikacją zdań na trzy grupy opinii: pozytywne, negatywne oraz neutralne.

2.1 Zastosowana metoda

Istnieje wiele technik i algorytmów, których używa się do analizy wydźwięku tekstu. Można je podzielić na dwie główne grupy: techniki oparte na regułach (leksykonie) oraz podejścia automatyczne wykorzystujące uczenie maszynowe.

Podejście regułowe polega na klasyfikowaniu tekstów na podstawie przygotowanego wcześniej słownika, który zawiera listę słów i wyrażeń oraz przypisany do nich wydźwięk[1]. Jeśli tekst posiada więcej słów pozytywnych niż negatywnych, to opinia jest klasyfikowana jako pozytywna, w przeciwnym wypadku jako negatywna. Podejście to posiada jednak kilka istotnych wad: po pierwsze stworzenie leksykonu wydźwięku jest czasochłonne i wymaga manualnego zdefiniowania reguł, a po drugie kontekst słów w zdaniu nie jest brany pod uwagę. Zatem jeśli w słowniku słowu *podoba* jest przypisana etykieta pozytywna, to zdanie *To mi się nie podoba* zostanie zaklasyfikowane jako pozytywne pomimo negacji.

Natomiast w podejściu opartym o algorytmy uczenia maszynowego budowany jest automatyczny klasyfikator, który, mając zestaw danych treningowych, uczy się kojarzyć zadany wejściowy tekst z przypisanym mu wyjściem wynikowym (etykietą)[1]. Uczenie przebiera różną formę, w zależności od wybranego modelu i stopnia jego skomplikowania.

Do rozwiązania opisywanego problemu wybrana została technika uczenia maszynowego, gdyż wymaga dużo mniejszego nakładu pracy, daje możliwość osiągnięcia znacznie lepszych wyników, a końcowy system ma większą uniwersalność i skalowalność.

2.2 *State of the art*

Wraz z rozwojem głębokiego uczenia maszynowego (*ang. deep learning*), coraz powszechniejsze stało się stosowanie go w analizie tekstów, czego dowodem są liczne publikacje w tej dziedzinie[2]. Powstawanie nowych modeli sieci neuronowych doprowadziło w ostatnich dziesięciu latach do dużego postępu w automatycznej analizie wydźwięku tekstu[2], a równocześnie zwiększyła się popularność takich badań.

Tworzone są modele analizujące teksty pod kątem ich ogólnego wydźwięku, na poziomie zdań, a także w oparciu o konkretne aspekty, gdyż jedno zdanie może oceniać dwie różne kwestie (np. *Fabuła była ciekawa, ale obsada mi się nie podobała.*). Najnowsze badania oprócz analizy tego, czy tekst jest pozytywny, czy negatywny, skupiają się również na rozpoznawaniu różnych emocji (np. gniew czy smutek), a także na wykrywaniu sarkazmu i ironii[2].

Większość tych prac analizuje teksty w języku angielskim, natomiast w języku polskim nie przeprowadzono jak dotąd wielu badań na ten temat – przykładowe publikacje warte uwagi to projekt OPTA[3] Zespołu Inżynierii Lingwistycznej Polskiej Akademii Nauk, a także prace zgłoszone do konkursu PolEval 2017[4], gdzie jednym z zadań było stworzenie modelu do analizy wydźwięku tekstu.

3. Dane uczące

Dane wykorzystane do uczenia i testowania stworzonego modelu zostały zaczerpnięte ze zbioru Treebank Wydźwięku (*ang. Polish Sentiment Treebank*)[5] w wersji 2.0. Zbiór uczący zawiera 2200 zdań o różnym stopniu złożoności, natomiast zbiór testowy składa się z 350 zdań.

3.1 Rozkład zdań

Do plików z danymi tekstowymi dołączone są pliki *parents* przedstawiające rozbiór gramatyczny zdań za pomocą drzew składniowych. Każde słowo ma przypisanego rodzica, czyli węzeł nadrzędny, natomiast korzeń drzewa posiada etykietę 0. W tabeli 3.1 przedstawiono przykładowe zdanie ze zbioru uczącego. Jak widać, w tym przypadku korzeniem drzewa jest słowo *działali*.

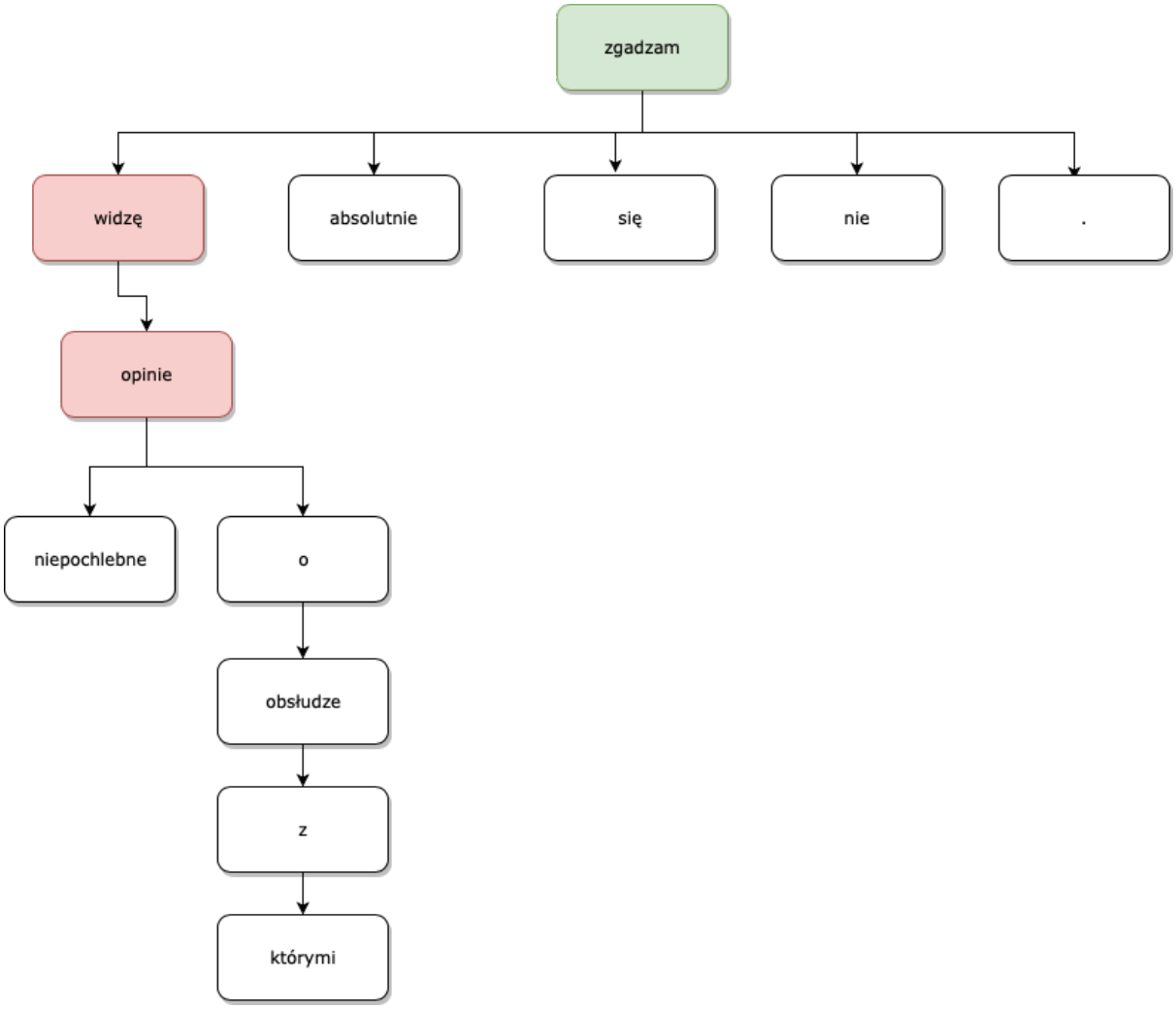
Pozycja w zdaniu	1	2	3	4	5	6	7	8	9	10
Słowo lub znak	*Zatrzymani*	*wczoraj*	*przestępcy*	*od*	*roku*	*działali*	*na*	*własną*	*rękę*	*.*
Rodzic	3	1	6	6	4	0	6	9	7	6

Tabela 3.1. Przykładowe zdanie z wykorzystanego zbioru danych wraz z przypisanymi rodzicami.

Dzięki takiemu rozbiorowi semantycznemu możliwe jest przedstawienie kontekstu słów w zdaniu w sposób zrozumiały dla sieci neuronowej.

3.2 Etykiety

Zdania w zbiorze danych są opatrzone etykietami wydźwięku (pliki *labels*), gdzie *-1* to opinia negatywna, *0* – neutralna, a *1* pozytywna. Reprezentacja etykiet w zbiorze uczącym jest następująca: 54% to etykiety pozytywne, natomiast 46% – negatywne. Ogólny wydźwięk zdania jest określany na podstawie etykiety korzenia drzewa, a także jego poddrzew. Jak widać na rys. 3.2, zdanie *Widzę niepochlebne opinie o obsłudze, z którymi absolutnie się nie zgadzam.* zawiera człon *widzę niepochlebne opinie o obsłudze*, który ma wydźwięk negatywny, dlatego został opatrzony etykietą *-1*. Jednak negacja dalszej części zdania zmienia jego wydźwięk na pozytywny, dlatego korzeń drzewa ma ostetcznie etykietę *1*.



Rys. 3.1. Drzewo zdania wraz z etykietami wydźwięku dla każdego poddrzewa.

4. Wybrany model

4.1 Rekurencyjna sieć neuronowa

Zdania to sekwencje danych, zatem warto jest wybrać taki algorytm, który podczas procesu uczenia zapamiętuje poprzednie elementy w sekwencji. Dzięki temu możliwe będzie wytrenowanie sieci w taki sposób, by kojarzyła dany wynik nie tylko z konkretnym słowem, ale także z pozostałymi słowami w zdaniu i ich kolejnością.

Rekurencyjna sieć neuronowa (RNN) rozszerza połączoną sieć neuronową (wzór 4.1) tak, by aktualny stan h_t był obliczany na podstawie akutalnego wejścia x_t oraz poprzedniego stanu h_{t-1}[6].

$$y = \sigma(Wx) \tag{4.1}$$

Najpopularniejszą postać RNN przedstawia wzór 4.2.

$$h_t = tanh(W_x x_t + W_h h_{t-1}) \tag{4.2}$$

Rozwiązanie to nie jest optymalne, gdyż w tego typu sieciach występuje problem znikającego lub ekspolodującego gradientu[7]. Oznacza to, że podczas uczenia gradient może wykładniczo maleć bądź rosnąć. W pierwszym przypadku sieć coraz gorzej aktualizuje wagi (lub nawet przestaje je aktualizować), zatem nie jest w stanie wyuczyć się zależności występujących w dłuższych sekwecjach. Natomiast rezultatem zbyt dużego gradientu są znaczne zmiany w macierzach wag, co może skutkować tym, że model będzie przeskakiwał minima i nigdy ich nie osiągnie, a w skrajnych przypadkach wartość funkcji kosztu wyniesie NaN.

4.2 LSTM

Problemowi znikającego gradientu zapobiega sieć LSTM[6] (*ang. Long Short-Term Memory*), która wprowadza strukturę c_t zwaną komórką pamięci, mającą możliwość przechowania informacji na przestrzeni czasu. Przepływ informacji z poprzednich stanów kontroluje mechanizm bramek: wejściowej i_t, zapominającej f_t oraz wyjściowej o_t. Wzory 4.3 - 4.8 przedstawiają wariant sieci LSTM użyty w tej pracy, gdzie W_x i W_h to macierze wag.

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i) \tag{4.3}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{4.4}$$

$$c_t = c_0 * f_t + i_t * tanh(W_{ci}x_t + W_{hc}h_{t-1} + b_c) \tag{4.5}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{4.6}$$

$$c_0 = c_t \tag{4.7}$$

$$h_t = o_t * tanh(c_0) \tag{4.8}$$

4.3 Struktura drzewa w LSTM

Ograniczeniem typowej architektury LSTM jest to, że pozwala ona tylko na sekwencyjną propagację informacji. Aby móc przetworzyć dane, które są przedstawione w postaci drzew zależności, konieczne jest wprowadzenie rozszerzonej sieci zwanej Tree-LSTM[6]. Od zwykłej sieci LSTM różni się tym, że aktualizacje wektorów bramek i komórki pamięci zależą nie od jednego stanu poprzedniego, ale od stanów wszystkich dzieci danego węzła.

Niniejsza praca opiera się na modelu zaproponowanym przez Michała Lwa i Piotra Pęzika[8], który nosi nazwę *Sequential Child-Combination Tree-LSTM*. Ma on taką samą postać jak model LSTM ze wzorów 4.3 - 4.8, z tą jednak różnicą, że aktualny stan h_t węzła jest wyliczany z kombinacji k stanów jego dzieci. Zatem wartość komórki pamięci c_{ik} jest wyznaczana korzystając ze stanu ukrytego h_{ck} poprzedniego dziecka w sekwencji (dzieci są ułożone w kolejności występowania w zdaniu), co przedstawiają wzory 4.9 - 4.13.

$$c_{i1} = c_0 * f_{i1} + i_{i1} * tanh(W_{ci}x_t + W_{hc}h_{c1} + b_c) \tag{4.9}$$

$$c_{i2} = c_{i1} * f_{i2} + i_{i2} * tanh(W_{ci}h_{i1} + W_{hc}h_{c2} + b_c) \tag{4.10}$$

...

$$c_{in} = c_{in} * f_{in} + i_{in} * tanh(W_{ci}h_{in-1} + W_{hc}h_{cn} + b_c) \tag{4.11}$$

$$c_t = c_{in} \tag{4.12}$$

$$h_t = h_{in} \tag{4.13}$$

4.4 Wektorowa reprezentacja słów

Do wektorowej reprezentacji słów w przestrzeni (*ang. word embedding)* służy model *Word2vec*. Przyporządkowuje on podobne słowa do punktów położonych blisko siebie w przestrzeni wektorowej[9].

Mapowanie odbywa się przy pomocy macierzy J o losowych wartościach i wymiarach MxN, gdzie M to rozmiar słownika zawierającego wszystkie słowa z danych uczących, natomiast N to przyjęty rozmiar macierzy (zazwyczaj przyjmuje się którąś z potęg liczby 2).

5. Wykorzystane narzędzia i technologie

5.1 Python 3.5

Python jest najpopularniejszym językiem programowania w uczeniu maszynowym – według raportu firmy Vision Mobile[10] używa go 57% programistów i naukowców związanych z przetwarzaniem danych. Na kolejnych miejscach znajduje się C/C++ (używany przez 43% programistów) oraz Java (41%).

Powyższy raport przytacza również konkretne najpopularniejsze obszary zastosowań danych języków. Język Python jest najbardziej powszechny w eksploracji danych z sieci (*ang. web mining*), przetwarzaniu języka naturalnego oraz w analizie wydźwięku tekstu (używany przez 44% programistów). Dzieje się tak dlatego, że Python to język o przejrzystej składni, kładący nacisk na prostotę i czytelność[11]. Zaletą są również liczne biblioteki, które można wykorzystać do uczenia maszynowego, między innymi PyTorch, TensorFlow, SciKit i Pandas.

5.2 TensorFlow

Tensorflow to biblioteka powszechnie wykorzystywana w głębokim uczeniu maszynowym. Dużą zaletą jest możliwość stworzenia całej sieci za pomocą zmiennych zastępczych (*ang. placeholder*). Umożliwiają one oddzielenie fazy konfiguracji od fazy ewaluacji, co pozwala na łatwe skalowanie sieci. Biblioteka ta jest niskopoziomowa, ma więc dużo możliwości do dostosowywania sieci do konkretnych potrzeb. Posiada również szczegółową dokumentację z przykładami immplementacji[12], a z racji dużej popularności, łatwo znaleźć wiele źródeł i tutoriali wykorzystujących TensorFlow.

Przed przystąpieniem do implementacji brana była pod uwagę również biblioteka Keras. Zapewnia ona wysokopoziomowy interfejs programowania aplikacji, co sprawia, że stworzenie sieci LSTM jest proste i nie wymaga wiele kodu. Ograniczona możliwość dostosowania poszczególnych parametrów sieci sprawia jednak, że w opisywanym w tej pracy przypadku użycie biblioteki Keras się nie sprawdza.

5.3 Scikit-learn

Jest to biblioteka przeznaczona do uczenia maszynowego. W pracy został wykorzystany jej moduł *metrics*[13]. Posiada on liczne funkcje do ewaluacji modelu oraz mierzenia jakości klasyfikacji. Na stronie biblioteki opisane są również przykłady zastosowania każdej z metryk oraz sposób interpretacji otrzymanych wyników.

5.4 Pozostałe narzędzia

- **NumPy** – biblioteka służąca do obliczeń numerycznych.
- **Pyplot** – moduł biblioteki Matplotlib służący do sporządzania wykresów.

6. Implementacja

6.1 Przygotowanie danych

Przygotowany został moduł *data*, w którym dane są wczytywane i poddawane obróbce koniecznej do przeprowadzenia prawidłowego procesu uczenia.

Każde słowo ze zbioru danych zapisano do słownika *word2idx* i przypisano mu unikalny indeks. W celu przyspieszenia działania programu zamieniono wszystkie słowa w zdaniach na ich indeksy ze słownika.

Lista etykiet określających rodziców została zastąpiona słownikiem, w którym kluczami są kolejne węzły, a wartościami – lista ich dzieci lub pusta lista w przypadku braku dzieci.

6.2 Implementacja modelu

Mamy trzy sekwencje, po których potrzebujemy przeiterować w tym samym czasie: listę słów, która zawierającą indeksy ze słownika *word2idx*, listę dzieci poszczególnych węzłów oraz listę etykiet ich wydźwięku. Potrzebna jest zatem pętla, która przejdzie przez te sekwencje. Nie znane są jednak na tym etapie wartości tych zmiennych, ani nawet ich długości, nie można więc użyć zwykłej pętli *while*. Wykorzystano w tym przypadku *tf.while_loop* z biblioteki TensorFlow (listing 1), która powtarza funkcję *pętla* dopóki *warunek* jest prawdziwy.

```
wyjście, _ = tf.while_loop(
        warunek,
        pętla,
        wartości początkowe
    )
```

Listing 1. Pętla generująca tablicę wyjść.

Warunkiem zakończenia pętli jest tutaj koniec zdania, natomiast ciało wykonywanej funkcji przedstawia listing 2. Dla każdego dziecka danego węzła obliczane są wartości poszczególnych bramek LSTM według wzorów podanych w rozdziale czwartym. W momencie gdy algorytm dojdzie do korzenia drzewa, który znajduje się na końcu listy, przeprowadzane jest mapowanie

słowa na wektor liczb rzeczywistych.

for dziecko **in** lista dzieci **do**
  if węzeł nie jest korzeniem **then**
    $h_1 = oblicz\_bramki(h_1, wyjscie(dziecko))$
  else
    $h_1 = mapuj(aktualne\ slowo)$
  end if
end for

Listing 2. Funkcja wyznaczająca stan aktualnego węzła.

Po zakończeniu pętli otrzymujemy ostateczną wyjściową tablicę. Mnożona jest ona przez wyjściową macierz wag i wyjściowy bias. Otrzymany wynik wykorzystywany jest do wyznaczenia wartości funkcji kosztu. W programie użyta została funkcja *sparse_softmax_cross_entropy_with_logits*, którą stosuje się do problemów klasyfikacji wieloklasowej.

Zasadniczo cała implementacja sprowadza się do następujących etapów:

- Wczytanie danych
- Zbudowanie modelu LSTM
- Wyznaczenie wartości kosztu
- Uruchomienie pętli uczenia
- Wyświetlenie wyników

6.3 Dobór parametrów

6.3.1. Wybór optymalizatora

Optymalizator to algorytm mający na celu znalezienie minimum zadanej funkcji kosztu. Wybór metody optymalizacji używanej w procesie uczenia miał duży wpływ na otrzymywane wyniki, co widać w tabeli 6.1. Zbadano optymalizator gradientu prostego, a także jego modyfikacje: *Momentum*, *Adagrad* oraz *Adam*. Powodem wybrania tych czterech algorytmów do analizy był fakt, że powtarzały się one najczęściej w znalezionych przykładach implementacji róznych modeli LSTM.

W każdym przypadku rozmiar partii wynosił 32, natomiast wartość wskaźnika uczenia – 0.01. W tabeli 6.1 widać bardzo dużą różnicę między wynikami osiąganymi przez optymalizator

Adam, a pozostałymi algorytmami. Nie ma więc wątpliwości, że Adam najlepiej sprawdzi się w trenowaniu zaporojektowanej tutaj sieci.

Klasyfikator	Adam	Adagrad	Gradient prosty	Momentum
Dokładność uczenia	**96,78%**	77,3%	89,47%	86,59%
Wartość funkcji kosztu	**2.56**	11,12	5,2	4,58

Tabela 6.1. Porównanie wyników po 10 epokach uczenia.

6.3.2. Wskaźnik uczenia

Wskaźnik uczenia (*ang. learning rate*) wpływa na to, jak szybko aktualizowane są wagi – zbyt wysoka wartość sprawia, że sieć może przeskakiwać minima, natomiast zbyt niska znacznie wydłuża czas uczenia. Tabela 6.2 przedstawia porównanie wyników dla 32-wymiarowej warstwy ukrytej i optymalizatora Adam. Jak widać, wysoki wskaźnik uczenia się nie sprawdza. W bibliotece TensorFlow dla optymalizatora Adam domyślnym wskaźnikiem uczenia jest wartość 0,001, jednak w opisywanym tutaj przypadku lepiej sprawdza się wartość 0,01, gdyż osiąga dobre wyniki w rozsądnym czasie.

Wskaźnik uczenia	0,1	0,01	0,001
Dokładność uczenia	77,25%	**96,78%**	96,64%
Wartość funkcji kosztu	12.05	**2.56**	4.31

Tabela 6.2. Porównanie wyników po 10 epokach uczenia.

6.3.3. Wymiary warstwy ukrytej

Działanie modelu zostało sprawdzone dla różnych wymiarów warstwy ukrytej. W każdym przypadku użyty został optymalizator Adam, natomiast wskaźnik uczenia wynosił 0.01. Jak widać w tabeli 6.2, wartości w zakresie 8-32 dają bardzo zbliżone wyniki w kwestii dokładności uczenia i wartości funkcji kosztu, z niewielką przewagą 32-wymiarowej warstwy ukrytej.

W literaturze nie znaleziono uzasadnienia wpływu tego parametru na ostateczne wyniki, można więc uznać dobór wymiarów warstwy ukrytej za kwestię indywidualną. Im większa jest ta wartość, tym dłuższy jest czas obliczeń: przy 16 wymiarach czas obliczeń jednej epoki wynosił 150 s, przy 32 – 200 s, natomiast dla 64-wymiarowego stanu ukrytego już 337 s.

Wymiary warstwy ukrytej	4	8	16	32	64
Dokładność uczenia	95,46%	97,02%	96,93%	**97,09%**	96,44%
Wartość funkcji kosztu	3.55	2.39	2.3	**2.25**	2.55%

Tabela 6.3. Porównanie wyników po 15 epokach uczenia.

6.4 Regularyzacja modelu

6.4.1. Regularyzacja funkcji kosztu

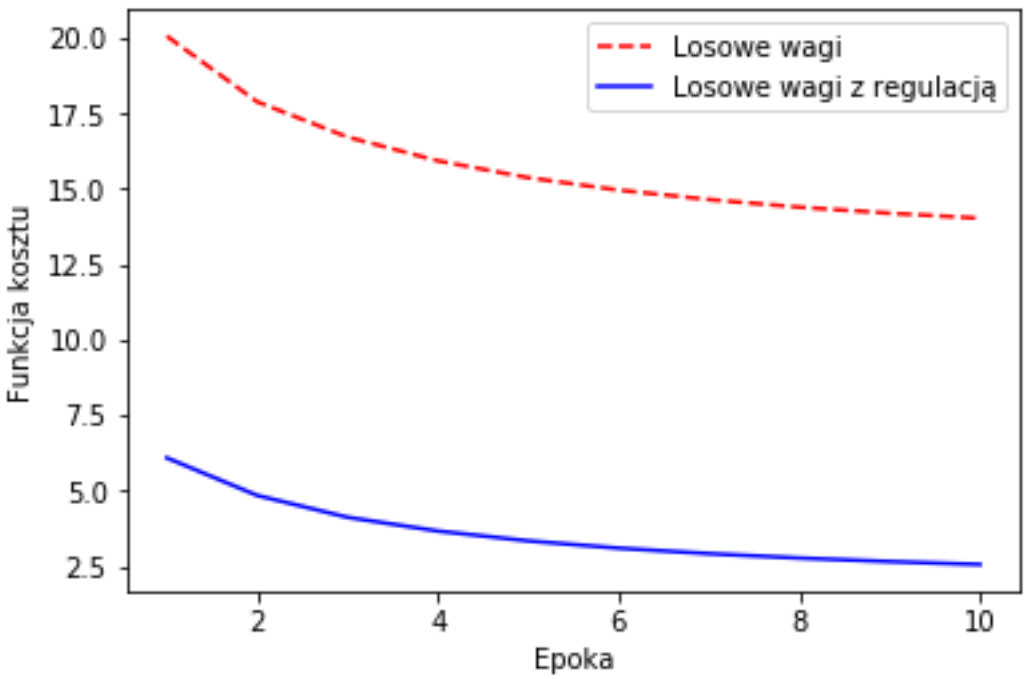
W celu zwiększenia wydajności modelu i uniknięcia przeuczenia (nadmiernego dopasowania) stosuje się różne techinki regularyzacji. Jedną z typowych metod jest regularyzacja l2, która polega na wyznaczeniu regularyzacyjnej funkcji kosztu będącej sumą normy L^2 obliczonej dla wag wejściowych W_{xi} oraz dla wag wyjściowych U, co przedstawia wzór 6.1. Wartość 0,02 jest tu dobranym parametrem optymalnym do używanego modelu. Wyznaczenie całkowitego kosztu polega na zsumowaniu wartości funkcji kosztu i wartości funkcji regularyzacji.

$$r = 0,02 * L^2(W_{xi}) + L^2(U) \tag{6.1}$$

6.4.2. Wpływ wag na proces uczenia

Samo zaincjalizowanie macierzy wag losowymi wartościami może doprowadzić do problemu znikającego lub eksplodującego gradientu, który został opisany w rozdziale 4. Aby temu zapobiec, wartości początkowe macierzy wyznaczono za pomocą heurystyki ze wzoru 6.1.

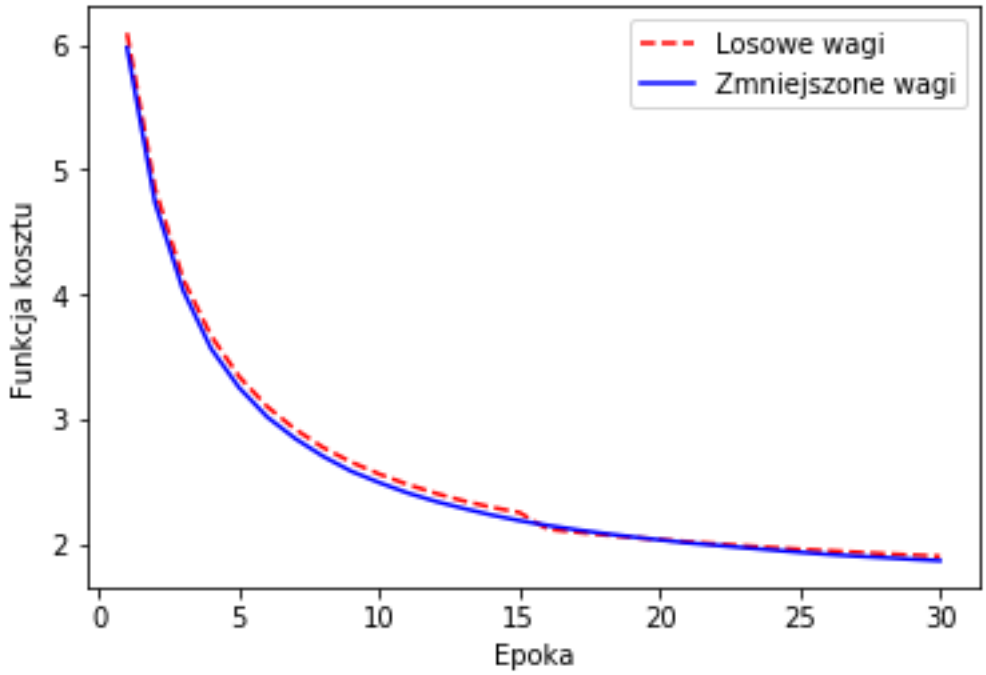
$$W = random(M, N) * \sqrt{\frac{1}{M+N}} \tag{6.2}$$



Rys. 6.1. Porównanie wyników dla macierzy wag inicjalizowanych losowo oraz losowo z regularyzacją.

Na wykresie 6.1 widać jak dużą wpływ na osiąganą wartość funkcji kosztu ma zastosowanie powyższej regularyzacji.

Ciekawym eksperymentem okazało się również zmniejszenie wag poprzez pomnożenie każdej macierzy przez parametr o wartości 0,05. Jak widać na wykresie 6.2, nie wprowadza to znacznych zmian, choć to ulepszenie pozwala na osiągnięcie danej wartości funkcji kosztu o dwie epoki szybciej niż w przypadku, gdy nie mnożymy macierzy przez parametr.



Rys. 6.2. Porównanie wyników dla macierzy wag inicjalizowanych losowo oraz losowo ze zmniejszonymi wagami.

7. Metryki

Podstawowymi miarami obrazującymi jakość uczenia sieci neuronowej jest dokładność (*ang. accuracy*) oraz wartość funkcji kosztu (*ang. loss value*).

Dokładność to procent prawidłowych klasyfikacji, zatem wyznaczenie tej wartości dla zbioru uczącego i testowego obrazuje jakość uczenia. Można również zaobserwować, czy jest różnica między tym, jak sieć radzi sobie z danymi, które już zna i tymi, które widzi po raz pierwszy.

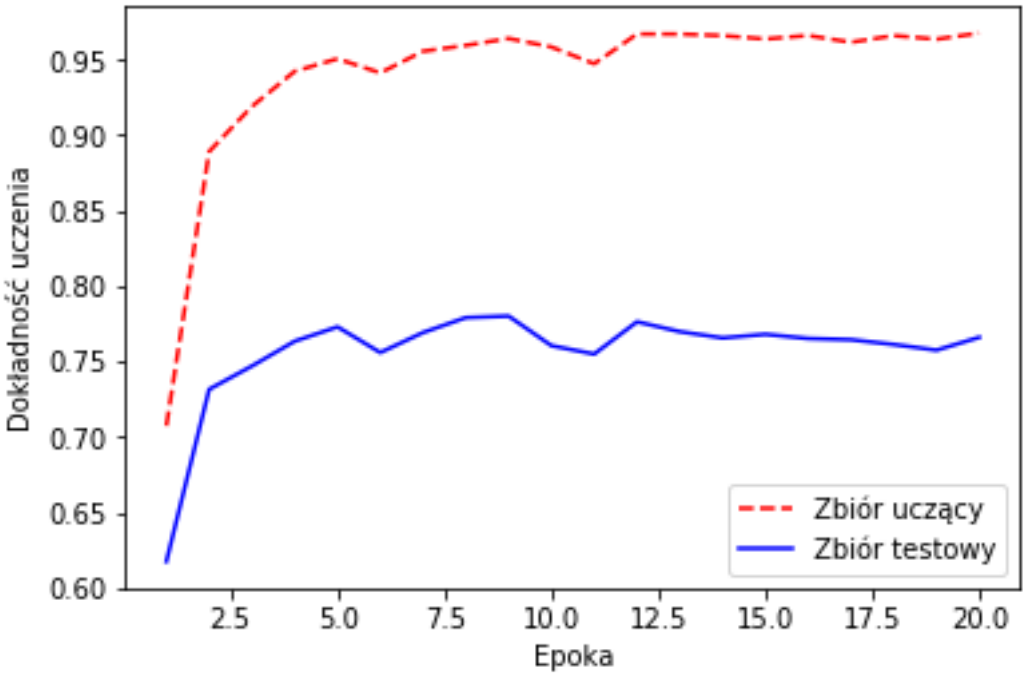
Celem procesu uczenia jest minimalizacja funkcji kosztu, zatem im niższa jej wartość, tym lepsza jest nasza sieć. Gdy wartość tej funkcji zaczyna rosnąć, mamy do czynienia z problemem przeuczenia.

7.1 Porównanie metryk

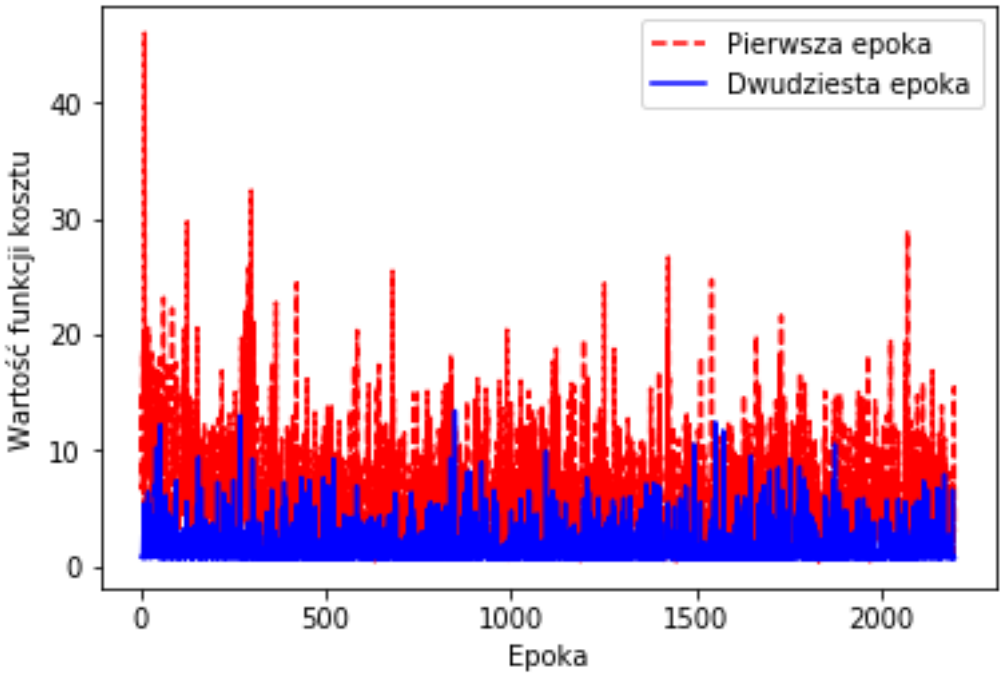
Pierwszy analizowany model ma następujące parametry:

- Optymalizator: **Adam**
- Wskaźnik uczenia: **0,01**
- Wymiary warstwy ukrytej: **16**
- Liczba epok: **20**

Jak widać na wykresie 7.1, występuje duża rozbieżność między dokładnością zbioru uczącego a dokładnością zbioru treningowego i testowego. W każdym z analizowanych podczas badań przypadków ta różnica była podobna lub nawet większa. Ani razu podczas przeprowadzanych badań nie udało się osiągnąć w zbiorze testowym dokładności większej niż 78%.m. Powodem takiej sytuacji może być zjawisko nadmiernego dopasowania, gdzie model uczy się konkretnych reguł, których nie jest w stanie rozszerzyć poza zbiór uczący. Iną przyczyną może być zbyt mała ilość danych uczących. Ta opcja jest bardziej prawdopodobna, gdyż dokładność w zbiorze testowym stabilizuje się już po kilku epokach i ma bardzo zbliżone wartości niezależnie od wyboru parametrów (dlatego w rozdziale szóstym wartość ta nie została uwzględniona w tabelach porównawczych). Końcowa dokładność klasyfikacji zbioru uczącego to 96,77%, jest to również wartość nalepsza ze wszystkich epok. Dla zbioru testowego dokładność po dwudziestu epokach wynosi 76,6%, natomiast najlepszą wartość wynoszącą 78% uzyskano w ósmej epoce.



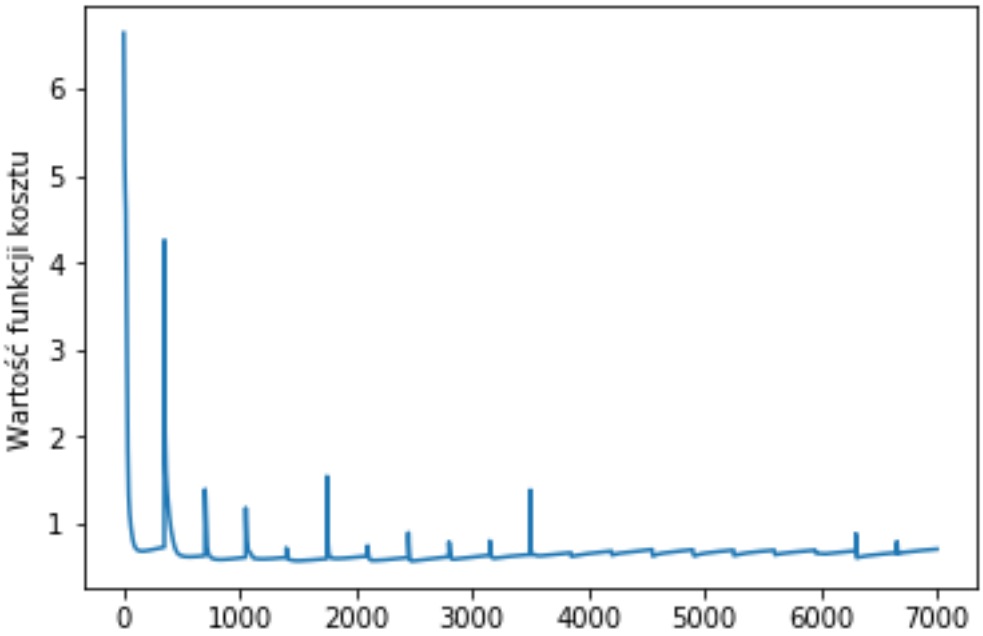
Rys. 7.1. Porównanie dokładności uczenia dla zbioru treningowego i testowego.



Rys. 7.2. Porównanie wartości funkcji kosztu dla pierwszej i dwudziestej epoki uczenia.

Wykres 7.2 obrazuje różnicę w wartościach funkcji kosztu osiąganych w pierwszej i ostatniej epoce treningu. Widzimy, że proces uczenia przebiega prawidłowo – średni koszt w pierwszej epoce wynosi 6,12, natomiast w dwudziestej 1,63.

Wartość funkcji kosztu dla danych testowych szybko się stabilizuje osiągając minimum, co pokazuje wykres 7.3. Średni koszt w ostatniej epoce wynosi 0,69.

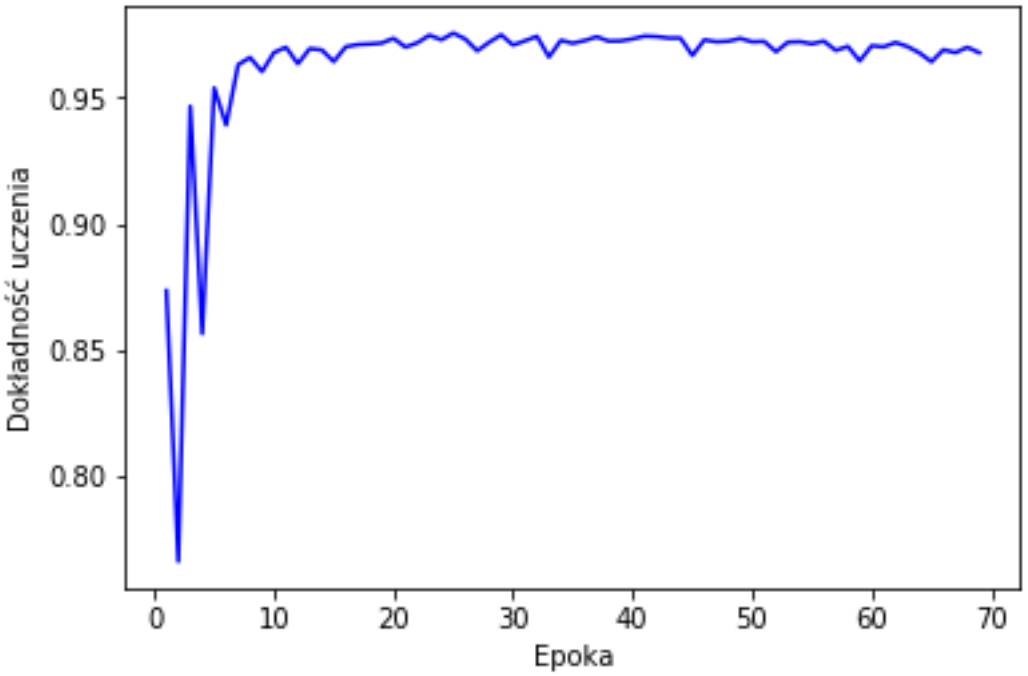


Rys. 7.3. Wartość funkcji kosztu dla zbioru testowego.

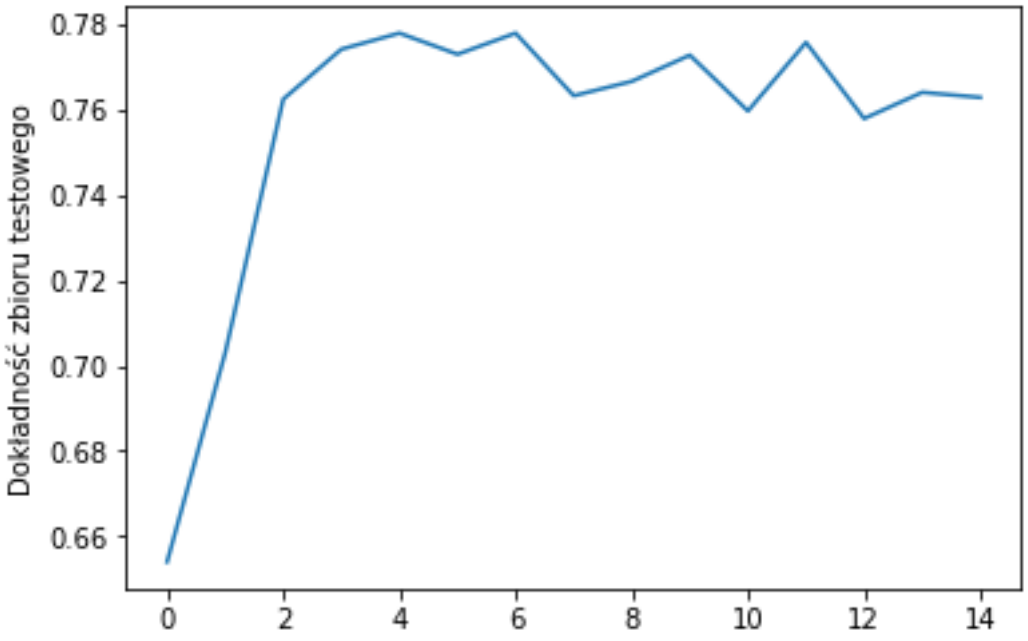
Dla porównania przeanalizowano również wykresy modelu z następującymi parametrami:

- Optymalizator: **Adam**
- Wskaźnik uczenia: **0,01**
- Wymiary warstwy ukrytej: **32**
- Liczba epok: **70**

Jak widać na wykresie 7.4, dokładność klasyfikacji zbioru uczącego osiąga stabilność po ósmej epoce, a wartości są zbliżone do tych z wykresu 7.1. Wynik uzyskany po siedemdziesięciu epokach to 96,8%, natomiast najlepsza uzyskana wartość to 97,8%.



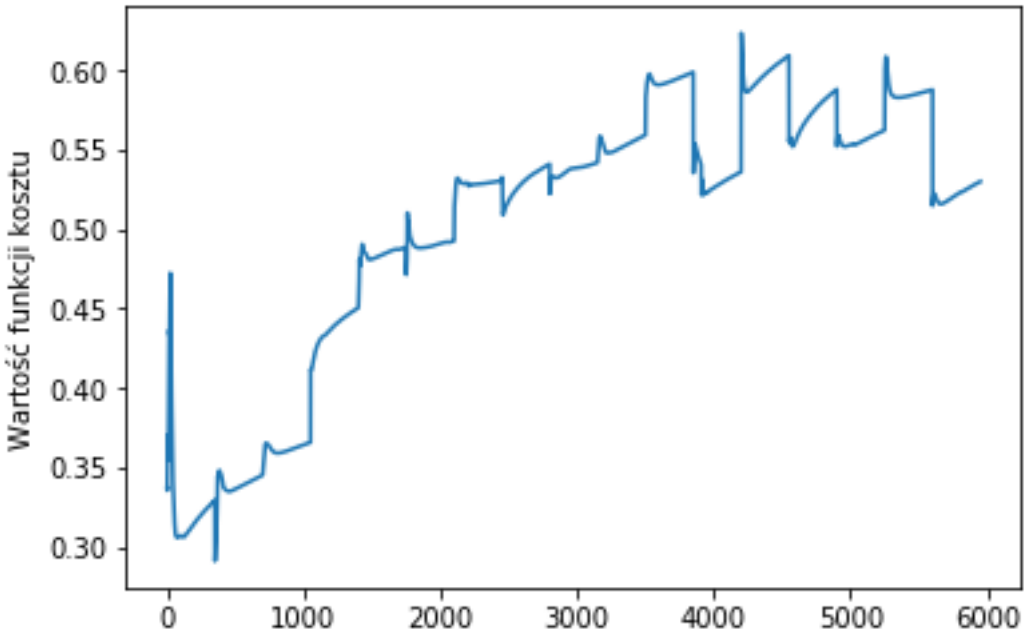
Rys. 7.4. Dokładność uczenia dla zbioru treningowego.



Rys. 7.5. Dokładność uczenia dla zbioru testowego.

Dokładność dopasowań w zbiorze testowym (wykres 7.5) po siedemdziesięciu epokach wynosi 76,4%, natomiast najlepsza uzyskana to 77,79%.

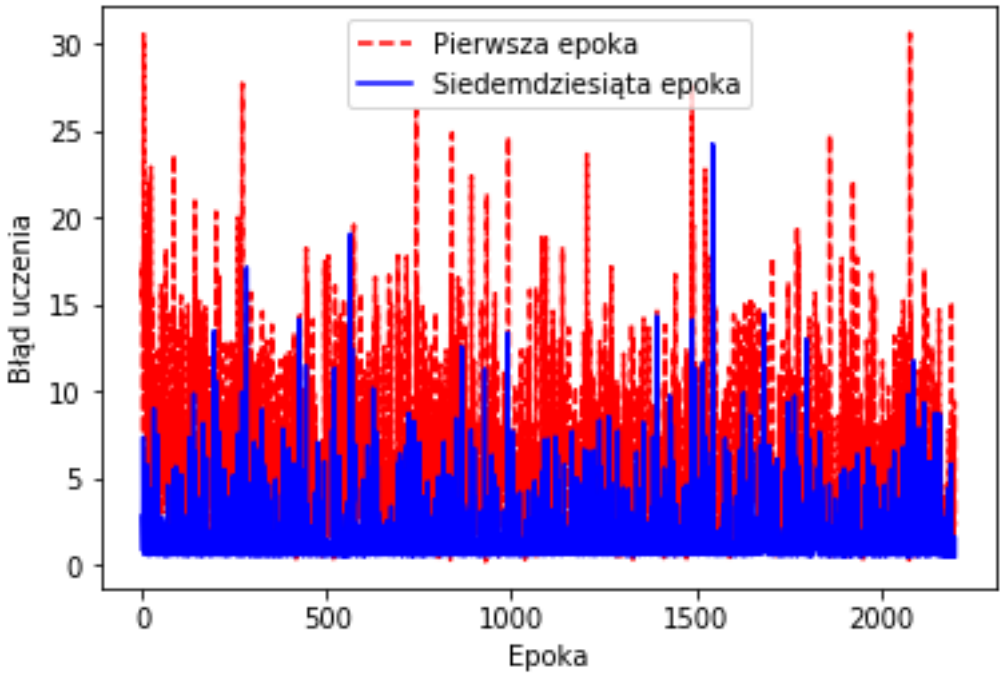
Zastanawiający jest przebieg wykresu 7.6, przedstawiającego zmianę wartości funkcji kosztu dla danych testowych. Wartość ta rośnie w miarę uczenia, co wskazuje na to, że mamy do czynienia z przeuczeniem. Jednak na wykresie 7.5 widzimy, że dokładność uczenia wraz z kolejnymi epokami utrzymuje się na tym samym poziomie. Może to oznaczać, że mimo iż model nie popełnia nowych błędów, to coraz bardziej utwierdza się w błędnych dopasowaniach.



Rys. 7.6. Wartość funkcji kosztu dla zbioru testowego.

Dla zbioru uczącego wartość funkcji kosztu w ostatniej epoce jest znacznie niższa niż w siedemdziesiątej (co przedstawia wykres 7.7), więc w tym przypadku sieć zachowuje się prawidłowo.

Do dalszej analizy metryk w tym rozdziale oraz przypadków w rozdziale 8 wybrano drugą z opisywanych w tym podpunkcie sieci.



Rys. 7.7. Porównanie wartości funkcji kosztu dla pierwszej i siedemdziesiątej epoki uczenia.

7.2 Raport klasyfikacji

	precyzja	**czułość**	**miara F-1**	**liczba wystąpień**
neutralne	0,84	0,90	0,87	3666
pozytywne	0.70	0,40	0,51	1016
negatywne	0,29	0,41	0,34	365

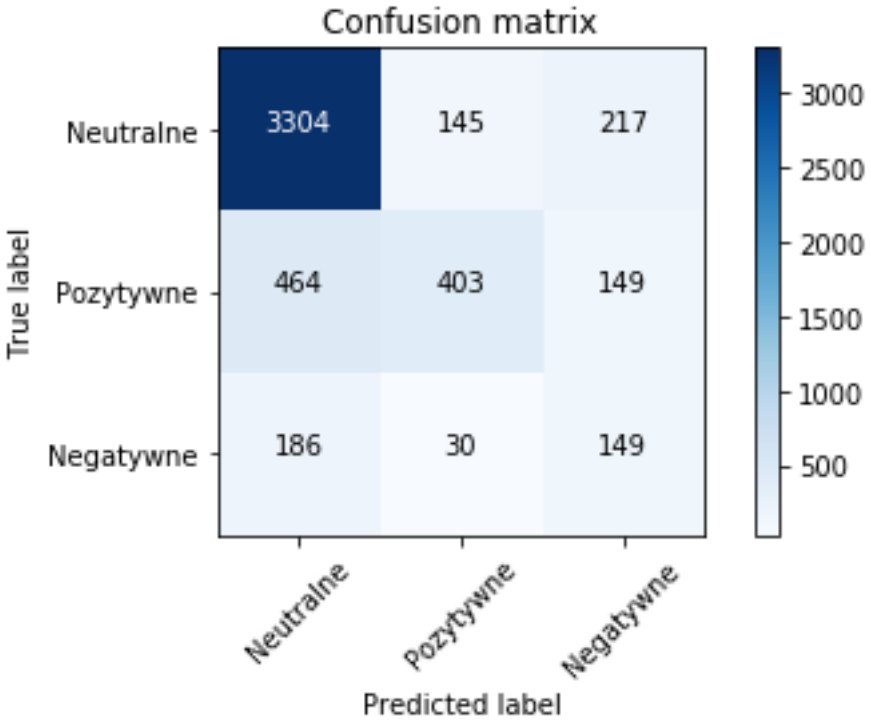
Tabela 7.1. Raport klasyfikacji modelu po 70 epokach uczenia.

Precyzja to stosunek liczby prawdziwie pozytywnych dopasowań do wszystkich dopasowań. Jak widać sieć radzi sobie dobrze z dopasowaniem etykiet neutralnych i całkiem dobrze rozpoznaje wydźwięk pozytywny. Natomiast w przypadku etykiet negatywnych mamy bardzo niską precyzję, widać, że w większości przypadków sieć źle dopasowuje wydźwięk negatywny.

Czułość to stosunek liczby dopasowań prawdziwie pozytywnych do ogólnej liczby możliwych dopasowań. W tym przypadku widzimy, że tylko 40% etykiet pozytywnych zostało dopasowanych.

Miara F-1 jest średnią harmoniczną precyzji i czułości, zatem obrazuje ogólną jakość modelu. Widzimy, że dla etykiet neutralnych klasyfikacja przebiega prawidłowo, natomiast dla etykiet pozytywnych i negatywnych nie działa zbyt dobrze.

Powodem takich wyników może być znaczna przewaga etykiet neutralnych w zbiorze uczącym, dzięki czemu sieć mogła się lepiej wyuczyć związanych z nimi zależności. Najmniej liczne były etykiety negatywne, co przekłada się na wyniki klasyfikacji. Dokładną liczbę dopasowań konkretnych etykiet do danych testowych obrazuje macierz konfuzji z rys. 7.8.



Rys. 7.8. Macierz konfuzji modelu po 70 epokach uczenia.

8. Analiza przypadków

8.1 Prawidłowe dopasowania

W zbiorze testowym znajdują się zdania o różnym stopniu skomplikowania. Na początek do analizy wybrano proste zdanie o pozytywnym wydźwięku:

Zdanie	*Flakon*	*dość*	*ładny*	.
Etykiety rzeczywiste	0	0	1	0
Etykiety przewidziane	0	0	1	0

Tabela 8.1. Zdanie wraz z etykietami wydźwięku. Dopasowanie: 100%

Jak widać w tabeli 8.1, sieć prawidłowo skojarzyła słowo *ładny* z wydźwiękiem pozytywnym, zatem zdanie jest prawidłowo sklasyfikowane. Tak samo dobrze dopasowane zostały etykiety do kolejnego prostego zdania (tabela 8.2), tym razem mającego wydźwięk neutralny.

Zdanie	*Opakowanie*	*pasuje*	*do*	*zapachu*	.
Etykiety rzeczywiste	0	0	0	0	0
Etykiety przewidziane	0	0	0	0	0

Tabela 8.2. Zdanie wraz z etykietami wydźwięku. Dopasowanie: 100%

Można więc sądzić, że sieć dobrze sobie radzi z prostymi zdaniami. Jednak przeanalizujmy przypadek z tabeli 8.3. Tutaj mamy do czynienia jednocześnie z błędem pierwszego i drugiego rodzaju, choć ostateczny negatywny wydźwięk zdania został wyznaczony prawidłowo.

Zdanie	*Dla*	*mnie*	*nieodpowiedni*	*model*
Etykiety rzeczywiste	0	0	-1	0
Etykiety przewidziane	0	0	0	-1

Tabela 8.3. Zdanie wraz z etykietami wydźwięku. Dopasowanie: 50%

Dla bardziej skomplikowanych zdań sieć dokonuje większej liczby pomyłek, co widać w tabeli 8.4. Korzeniem drzewa w tym zdaniu jest słowo *może*, zatem według etykiet rzeczywistych zdanie

to ma wydźwięk neutralny, a sieć neuronowaw zaklasyfikowała je jako negatywne. Mimo błędnej klasyfikacji trafność dopasowania etykiet wynosi 75%.

Zdanie	*Minusem*	*może*	*być*	*jego*	*popularność*	,	*chociaż*	*mi*	*to*	*nie*	*przeszkadza*	.
Etykiety rzeczywiste	-1	0	-1	0	1	0	-1	0	0	0	0	0
Etykiety przewidziane	-1	-1	-1	0	1	0	0	0	0	-1	0	0

Tabela 8.4. Zdanie wraz z etykietami wydźwięku. Dopasowanie: 75%

W tabeli 8.5 mamy kolejne zdanie, którego wydźwięk pozytywny został poprawnie wykryty. Sieć jednak nie zaklasyfikowała słowa *fantastyczny* jako pozytywne. Powodem tego może być fakt, że słowo to nie występowało w danych uczących.

Zdanie	*Fantastyczny*	*zapach*	,	*napełniający*	*optymizmem*	.
Etykiety rzeczywiste	1	0	0	1	1	0
Etykiety przewidziane	0	0	0	1	1	0

Tabela 8.5. Zdanie wraz z etykietami wydźwięku. Dopasowanie: 83%

Raport z rozdziału 7 pokazał, że sieć nie radzi sobie z dopasowaniem etykiet negatywnych. Często wydźwięk negatywny zdania jest prawidłowo rozpoznawany, jednak złe jest przyporządkowanie etykiet do konkretnych słów. Jak widać w tabeli 8.6, problem z dopasowaniem etykiet mógłby mieć również człowiek. O ile słowo *mdli* można uznać za jednoznacznie negatywne, to już przyporządkowanie członowi *i głowa boli* wydźwięku neutralnego jest dyskusyjne.

Zdanie	*Mnie*	*od*	*niego*	*mdli*	i	*głowa*	*boli*	.
Etykiety rzeczywiste	0	0	0	-1	0	0	0	0
Etykiety przewidziane	0	0	0	0	-1	0	-1	0

Tabela 8.6. Zdanie wraz z etykietami wydźwięku. Dopasowanie: 62%

9. Dyskusja

Otrzymane wyniki nie są dokładne, ale po przeanalizowaniu konkretnych przypadków można je uznać za zadowalające. Widać, że duży wpływ na klasyfikację miała nadreprezentacja etykiet neutralnych. Możliwe więc, że błędem był podział na trzy klasy, gdyż nie da się przygotować takich danych, by wszystkie trzy etykiety występowały w równym stopniu, bo w zdaniach, szczególnie złożonych, jest wiele słów, które nie wpływają na ich wydźwięk. Zatem w dalszych badaniach warto będzie sprowadzić klasyfikację do problemu binarnego, gdzie uwzględniamy wyłącznie etykiety pozytywne i negatywne.

Problemem napotkanym podczas badań był długi czas obliczeń, co ograniczało w zakresie doboru parametrów, gdyż nie można było wykonać zbyt wielu prób.

W kwestii dalszego rozwoju programu warto przeprowadzić eksperymenty z modyfikacją poszczególnych parametrów, można się również zastanowić nad zmianą modelu sieci. Dużą poprawę może przynieść zwiększenie i zbalansowanie zbioru uczącego.

10. Wnioski

Cel pracy został osiągnięty. Otrzymany program klasyfikuje tekst za pomocą uczenia maszynowego pod kątem pozytywnym, negatywnym i neutralnym. Z racji przedstawienia zdań w postaci drzew składniowych, jest w stanie przyjąć wyłącznie tekst, który będzie miał wyodrębnione te zależnośći. Zatem aby móc otrzymać program o potencjale użytkowym, należy połączyć go z parserem zależnośći, który będzie zamieniał zdania na strukturę drzewa. Taki parser udostępnia Zespół Inżnierii Lingwistycznej[5].

Model zaproponowany w niniejszej pracy może być dobrą bazą do dalszych badań, zarówno pod kątem naukowym, jak i biznesowym. Uzyskane wyniki są dobrą bazą do analizy tego tematu, zarówno w zakresie wydźwięku tekstu, jak i w innych dziedzinach przetwarzania języka naturalnego.

Bibliografia

[1] S.M. Vohra i J.B. Teraiya. „A comparative Study of Sentiment Analysis Techniques". W: *Journal JIKRCE* 2 (2013), s. 313–317.

[2] Lei Zhang, Shuai Wang i Bing Liu. „Deep Learning for Sentiment Analysis : A Survey". W: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2018).

[3] Aleksander Wawer. „Towards Domain-Independent Opinion Target Extraction". W: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)* (2015), s. 1326–1331.

[4] *PolEval 2017. http://2017.poleval.pl*.
[online; stan na 18.01.2019]. 2017.

[5] *Treebank Wydźwięku. http://zil.ipipan.waw.pl/TreebankWydzwieku*.
[online; stan na 18.01.2019]. 2018.

[6] Kai Sheng Tai, Richard Socher i Christopher D. Manning. „Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". W: *CoRR* abs/1503.00075 (2015).

[7] Shenxiu Liu i Qingyun Sun. „Conquering vanishing gradient: Tensor Tree LSTM on aspect-sentiment classification". W: (2015).

[8] Michał Lew i Piotr Pęzik. „A Sequential Child-Combination Tree-LSTM Network for Sentiment Analysis". W: *LTC 2017* (2017), s. 397–401.

[9] *Vector Representations of Words. https://www.tensorflow.org/tutorials/representation/word2vec*.
[online; stan na 21.01.2019].

[10] *Developer Economics: State of the Developer Nation Q1 2017. https://www.visionmobile.com/free-resources/*.
[online; stan na 18.01.2019]. 2017.

[11] Tim Peters. *The Zen of Python. https://www.python.org/dev/peps/pep-0020/*.
[online; stan na 18.01.2019]. 2004.

[12] *TensorFlow. https://www.tensorflow.org/*.
[online; stan na 18.01.2019].

[13] *Model evaluation: quantifying the quality of predictions. https://scikit-learn.org/stable/modules/model_evaluation.html.*
[online; stan na 20.01.2019].