



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI
KATEDRA TELEKOMUNIKACJI

PRACA DYPLOMOWA INŻYNIERSKA

**Aplikacja do przewidywania liter w tekście wykorzystująca
uczenie maszynowe**

Application for character prediction using machine learning

Autor: **Michał Cieślak**

Kierunek studiów: **Elektronika i Telekomunikacja**

Typ studiów: **Stacjonarne**

Opiekun pracy: **dr inż. Jarosław Bułat**

Kraków, 2020

OŚWIADCZENIE STUDENTA

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz.U. z 2018 r. poz. 1191 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, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym (tj. Dz.U. z 2018 r. poz. 1668, z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.”, oświadczam, że niniejszą pracę dyplomową wykonałem osobiście, samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba, że praca jest częścią utworu zbiorowego. Ponadto Uczelnia, jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz.U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

(czytelny podpis studenta)

Spis treści

Wstęp	5
1. Sieci neuronowe.....	6
1.1. Czym są sieci neuronowe.....	6
1.2. Typy sieci neuronowych	8
1.3. Zasada działania sztucznych sieci neuronowych	12
1.4. Sieci Long short-term memory	16
2. Przewidywanie tekstu	20
2.1. Opis użytych narzędzi.....	20
2.2. Modelowanie języka	25
2.3. Implementacja modelu języka naturalnego	27
3. Opis otrzymanych wyników	39
3.1. Wyniki aplikacji przewidującej znaki.....	40
3.2. Wyniki aplikacji przewidującej wyrazy.....	44
3.3. Porównanie z Łańcuchem Markowa.....	49
4. Podsumowanie.....	52
Bibliografia	53

Wstęp

Sztuczne sieci neuronowe to przyszłość świata technicznego. Ich początek datuje się na rok 1943 (opisanie matematycznego modelu sztucznego neuronu przez McCullocha i Pittsa). Kluczowym wydarzeniem było opublikowanie artykułu przez Hintona i Salakhutdinova w magazynie „Science” pod tytułem „Reducing the Dimensionality of Data with Neural Networks [31]”. Był on inspiracją do badań dla wielu osób. Na przestrzeni ostatnich lat sztuczne sieci neuronowe stały się bardzo popularnym i skutecznym narzędziem do analizy danych. Obecnie sieci neuronowe przeżywają renesans, głównie dzięki lepszym rozwiązaniom wcześniejszych problemów, dostępności wielkich zbiorów danych oraz rozwoju techniki (większe możliwości obliczeniowe). Na sztucznych sieciach neuronowych opierają się najnowocześniejsze systemy sztucznej inteligencji. Na ich rozwój stawiają wielkie firmy takie jak Google, IBM czy Facebook.

Celem pracy jest przedstawienie możliwości zastosowania sztucznych sieci neuronowych w przetwarzaniu języka naturalnego, a w szczególności w przewidywaniu tekstu.

Praca została podzielona na cztery rozdziały. W pierwszym rozdziale opisano podstawy działania sztucznych sieci neuronowych. Drugi rozdział zawiera opis implementacji modelu języka naturalnego przy pomocy rekurencyjnych sieci neuronowych. W rozdziale trzecim zostały przedstawione wyniki aplikacji do przewidywania tekstu oraz porównanie tych wyników z łańcuchem Markowa. Ostatni rozdział jest podsumowaniem wykonanej pracy.

1. Sieci neuronowe

1.1. Czym są sieci neuronowe

Początki sztucznych sieci neuronowych sięgają 1943 roku, kiedy to Warren McCulloch i Walter Pitts napisali artykuł na temat tego, jak neurony mogą pracować i modelować ich pomysły, tworząc również prostą sieć neuronową za pomocą obwodów elektrycznych. Kilka lat później Hebb w 1949 roku odkrył, że informacja może być zapisana w strukturze sieci, jako waga połączeń. Dalszy rozwój tej dziedziny nauki zapewnił Rosenblatt, tworząc sieć neuronową zwaną perceptronem (System do określenia przynależności danych wejściowych do jednej z dwóch klas). Badania nad sieciami neuronowymi nabrały tempa, gdy w 1975 roku Kunihiko Fukushima opracowała pierwszą prawdziwą sieć neuronową składającą się z wielu warstw. W 1982 roku John Hopfield wprowadził rekurencyjną architekturę sieci neuronowej. Trzy lata później, w 1985 roku, Amerykański Instytut Fizyki zapoczątkował coroczne konferencje o nazwie „Neural Networks for Computing”. Istotną datą jest też rok 1997, w którym to Schmidhuber i Hochreiter wymyślili sieć Long short-term memory (LSTM). Pierwotnym celem w badaniach nad sieciami neuronowymi było stworzenie narzędzia obliczeniowego, które mogłoby rozwiązywać problemy analogicznie do ludzkiego mózgu. Z upływem czasu badacze zaczęli wykorzystywać sieci neuronowe do spełnienia określonych zadań, co odsunęło na bok ściśle biologiczne założenie sieci neuronowej, jako analogu mózgu [1].

Sieć neuronowa jest rodzajem uczenia maszynowego, który w uproszczony sposób sprawuje funkcje podobne do ludzkiego mózgu. Składa się z jednostek obliczeniowych pracujących równolegle, które mają zdolność do przechowywania wiedzy empirycznej i przekazywania nauczonych informacji [2]. Jednostki te, zwane neuronami, połączone w dużą sieć, wykonują proste przetwarzanie informacji. Korzystając z algorytmów neurony mogą rozpoznawać ukryte wzorce, znajdować korelacje w zbiorze danych i je grupować.

Sieć neuronowa przypomina mózg pod kilkoma względami [3]:

- Wiedza jest pozyskiwana przez sieć z jej środowiska przez proces uczenia się,

- Przetwarzanie danych wykonywane jest w sposób równoległy i wyciągane są z nich istotne informacje,
- Waga synaptyczna, czyli siła połączenia pomiędzy neuronami, służy do przechowywania zdobytej wiedzy,
- Magazynowanie informacji jest dokonywane w pamięci i odwoływanie się do nich następuje w odpowiednich sytuacjach.

Sieci neuronowe pomagają ludziom w rozwiązywaniu złożonych zadań w rzeczywistych sytuacjach. Mogą one tworzyć relacje między wejściami i wyjściami, które są złożone, oraz ujawniać ukryte wzorce i prognozy. W rezultacie sieci neuronowe odnajdują zastosowanie w wielu obszarach codziennego życia [1]:

- Transport: optymalizacja logistyki dostaw,
- Energetyka: optymalizacja dostaw, prognozowanie potrzeb energetycznych,
- Sprzedaż online: chat boty konwersacyjne, rekomendacja produktów,
- Medycyna: diagnostyka predykcyjna, obrazowanie biomedyczne i monitorowanie stanu zdrowia,
- Bankowość: wykrywanie oszustw, analizy kredytowe, prognozy finansowe dotyczące cen akcji, waluty, prognozowanie sprzedaży,
- Rząd: wspieranie tzw. inteligentnych miast, monitorowania bezpieczeństwa, rozpoznawanie twarzy, głosu,
- Serwisy społecznościowe: dostosowanie reklam do zainteresowań użytkownika, rozpoznawanie twarzy na zdjęciach (Facebook), podpowiadanie emoji do wprowadzonego tekstu (Instagram), systemy rekomendacji (Amazon),
- Poczta elektroniczna: filtry spamu,
- Serwisy filmowe: polecanie filmów dostosowanych do zainteresowań użytkownika,
- Smartfony: translacja głosu na tekst, rozpoznawanie głosu, podpowiadanie tekstu.

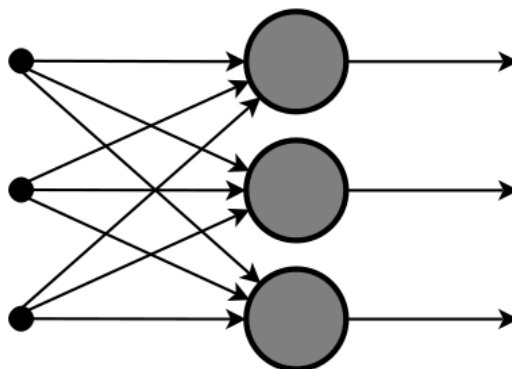
Znajdują one głównie zastosowanie tam, gdzie nie ma prostych reguł klasyfikacji. Podobnie jak człowiek, sieć neuronowa może się uczyć na przykładach, z tym, że posiada

o wiele większą szybkość uczenia się. Jednym z ciekawszych eksperymentów z realnego życia było diagnozowanie zawałów serca u pacjentów z silnymi bólami w klatce piersiowej. Celem doświadczenia było przewidzenie występowania zawału serca u pacjenta bazując na 41 danych o stanie zdrowia pacjenta. Sieć neuronowa stawiała lepszą diagnozę niż grupa lekarzy specjalistów (odpowiednio 92% i 88%) wykorzystując tylko połowę danych o pacjencie [4]. Za pomocą sieci neuronowej można uzyskać informacje, które cechy danych są najbardziej istotne do przewidzenia danego zjawiska. W przyszłości sieci neuronowe w medycynie mają służyć jako narzędzie, które oceni szanse wystąpienia choroby i pomoże lekarzowi w decyzji.

1.2. Typy sieci neuronowych

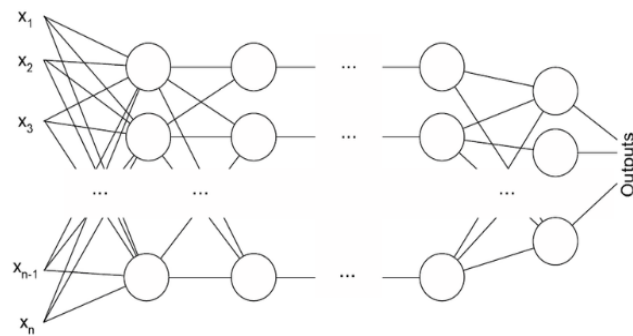
Różne typy sieci neuronowych stosują inne zasady przy ustalaniu własnych reguł. Istnieje wiele rodzajów sieci neuronowych, każdy z nich ma swoje zalety i wady, w zależności od zastosowania. Modele sieci neuronowych różnią się topologią połączeń oraz sposobem przesyłania informacji. Oto najważniejsze z nich [5]:

- Jednokierunkowe sieci neuronowe (ang. Feedforward Neural Network) – jedne z najprostszych rodzajów sieci neuronowych. W sieci dane przechodzą przez różne węzły wejściowe, aż dotrą do węzła wyjściowego. Dane przemieszczają się tylko w jednym kierunku (od pierwszej warstwy do węzła wyjściowego). Inaczej niż w bardziej złożonych sieciach neuronowych tutaj nie występuje propagacja wsteczna. Mogą mieć jedną warstwę albo dodatkowo mogą mieć warstwy ukryte. W jednokierunkowej sieci neuronowej wynikiem na wyjściu jest obliczana suma iloczynów danych wejściowych i ich wag. Schemat sieci został przedstawiony na Rys 1.1,



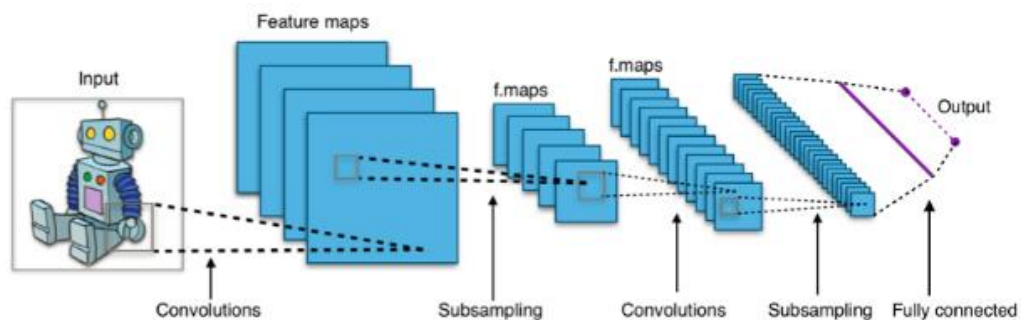
Rys. 1.1. Schemat jednokierunkowej sieci neuronowej [5]

- Perceptron wielowarstwowy (ang. Multilayer Perceptron) – ma trzy lub więcej warstw. Służy do klasyfikacji danych, których nie można rozdzielić liniowo. Każdy węzeł w warstwie jest połączony z każdym węzłem w kolejnej warstwie. Perceptron wielowarstwowy wykorzystuje nieliniową funkcję aktywacji. Ten typ sieci neuronowych jest stosowany w rozpoznawaniu mowy i tłumaczeniu maszynowym. Schemat sieci został przedstawiony na Rys 1.2,



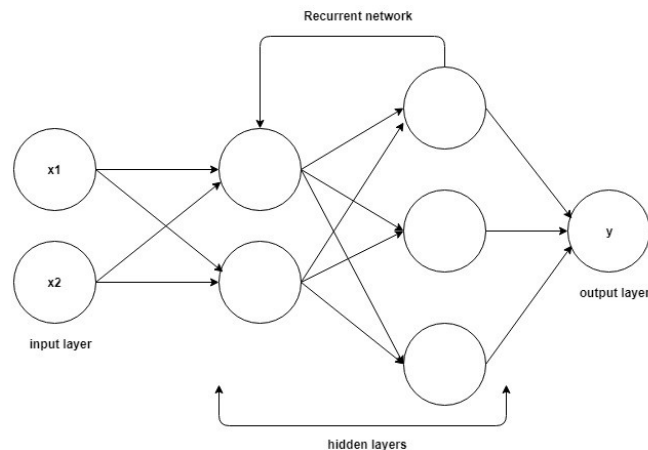
Rys. 1.2. Schemat perceptronu wielowarstwowego [5]

- Splotowe sieci neuronowe (ang. Convolutional Neural Network) – wykorzystują odmianę perceptronów wielowarstwowch. Zawierają jedną lub więcej warstw splotowych, które mogą być ze sobą całkowicie lub częściowo połączone. Wykonuje operacje splotu na wejściu zanim przekaże wynik do następnej warstwy. Dzięki tej zdolności ten rodzaj sieci jest skuteczny w rozpoznawaniu obrazów i wideo, przetwarzaniu języka naturalnego oraz w systemach rekomendacji. Dodatkowo, liczba parametrów warstw splotowych jest zdecydowanie mniejsza od liczby parametrów w warstwach w pełni połączonych (ang. fully connected), więc proces uczenia przebiega szybciej. Schemat sieci został przedstawiony na Rys 1.3,



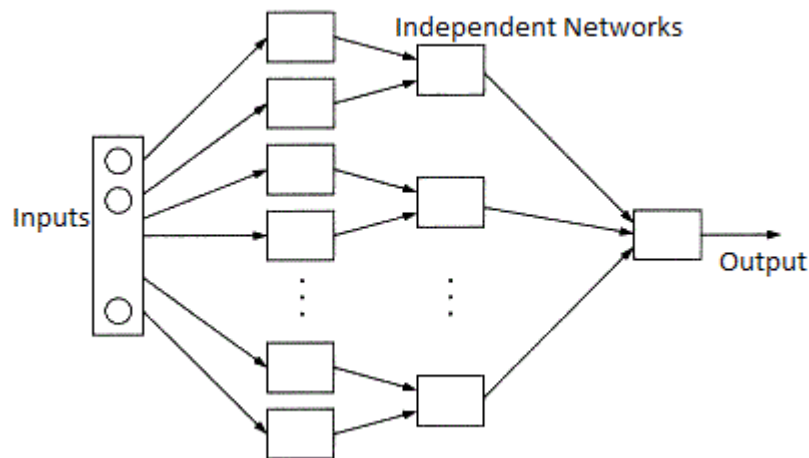
Rys. 1.3. Schemat splotowej sieci neuronowej [5]

- Rekurencyjne sieci neuronowe (ang. Recurrent Neural Network) – rodzaj sieci neuronowej, w której dane wyjściowe określonej warstwy są zapisywane i podawane z powrotem na wejście. Pierwsza warstwa jest tworzona tak samo jak w sieci jednokierunkowej, czyli jako iloczyn sumy wag i cech. W kolejnych warstwach rozpoczyna się proces cyklicznej sieci neuronowej. W każdym kroku czasowym każdy węzeł zapamiętuje informacje, które mogą być przydatne później. Jeśli przewidywanie jest niepoprawne, to w sieci następuje modyfikacja wag połączeń podczas propagacji wstecznej aby osiągnąć prawidłowy wynik. Ten typ jest bardzo skuteczny w konwersji tekstu na mowę. Rekurencyjne sieci neuronowe zostaną bardziej szczegółowo opisane w dalszej części pracy. Schemat sieci został przedstawiony na Rys 1.4,



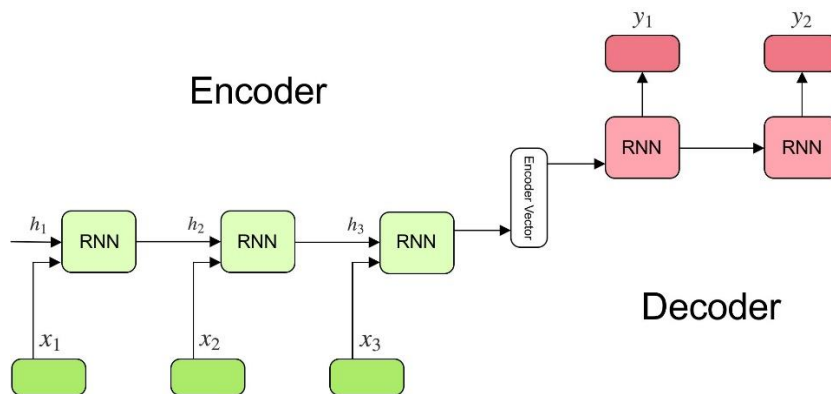
Rys. 1.4. Schemat rekurencyjnej sieci neuronowej [7]

Modułowa sieć neuronowa (ang. Modular Neural Network) – zawiera w sobie wiele różnych sieci neuronowych, które działają od siebie niezależnie i wykonują inne zadania. Różne sieci nie komunikują się ze sobą w żaden sposób, nawet podczas procesu obliczeniowego. Działają niezależnie, aby osiągnąć pożądany wynik. Złożony proces obliczeniowy zostaje podzielony na niezależne komponenty. Schemat sieci został przedstawiony na Rys 1.5,



Rys. 1.5. Schemat modułowej sieci neuronowej [5]

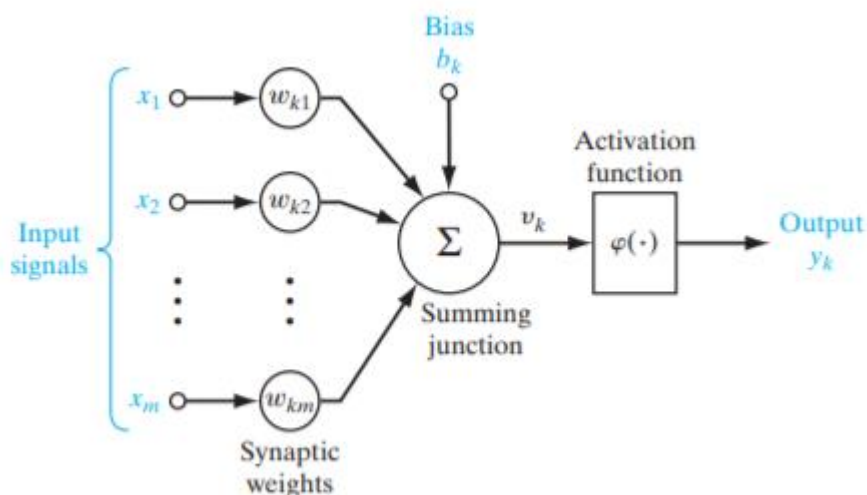
- Model sekwencyjno-sekwencyjny (ang. Sequence-To-Sequence Models) – składa się z dwóch rekurencyjnych sieci neuronowych. Jedną z nich to koder przetwarzający dane wejściowe a druga to dekodery przetwarzający dane wyjściowe. Ten model ma szczególne zastosowanie w przypadkach, gdy długość danych wejściowych jest inna niż długość danych wyjściowych. Schemat modelu został przedstawiony na Rys 1.6.



Rys. 1.6. Schemat modelu sekwencyjno-sekwencyjnego [15]

1.3. Zasada działania sztucznych sieci neuronowych

Podstawową jednostką przetwarzającą informacje jest neuron. Schemat na Rysunku 1.7 przedstawia model, który jest kluczowy do zrozumienia, w jaki sposób działa sieć neuronowa [3].



Rys. 1.7. Schemat modelu neuronu [3]

Model składa się z trzech podstawowych elementów:

1. Zestawu połączeń, z których każde ma własną wagę,
 2. Elementu sumującego, który sumuje iloczyny danych wejściowych z odpowiadającymi im wagami,
 3. Funkcji aktywacji, według której obliczana jest wartość na wyjściu neuronu.
- Najczęściej stosowane funkcje aktywacji to: ReLU (ang. Rectified Linear Unit), funkcja skokowa Heaviside'a, funkcja sigmoidalna, tangens hiperboliczny oraz funkcja Softmax.

Dodatkowo występuje także tzw. Bias, który jest stałą wartością powodującą zwiększenie lub obniżenie wejściowej funkcji aktywacji. Używając zapisu matematycznego można opisać neuron z Rys. 1.8 zależnościami (1.1) i (1.2) [3]:

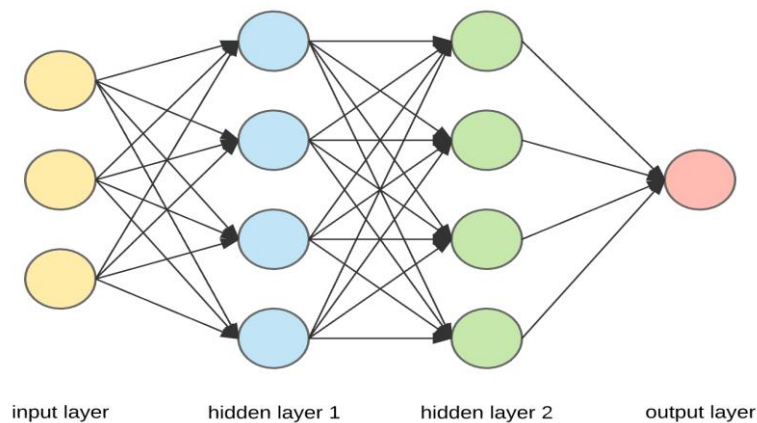
$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (1.1)$$

$$y_k = \varphi(u_k + b_k) \quad (1.2)$$

gdzie x_1, x_2, \dots, x_m są sygnałami wejściowymi; $w_{k1}, w_{k2}, \dots, w_{km}$ są wagami odpowiadających połączeń neuronu k ; u_k to suma iloczynów sygnałów wejściowych z wagami; b_k to bias; φ jest funkcją aktywacji; natomiast y_k jest sygnałem wyjściowym z neuronu.

Typowa sieć neuronowa zawiera dużą liczbę neuronów ułożonych w szereg warstw. Przykładowa sieć neuronowa została przedstawiona na Rys. 1.8. Dostępne rodzaje warstw w sieci neuronowej to [6]:

- Warstwa wejściowa (ang. Input layer) – zawiera te neurony, które mają otrzymywać informacje ze świata zewnętrznego,
- Warstwa wyjściowa (ang. Output layer) – zawiera jednostki, które uzyskują konkretny wynik na informacje wprowadzane do systemu,
- Warstwa ukryta (ang. Hidden layer) – znajduje się pomiędzy warstwą wejściową a wyjściową. Jej zadaniem jest przekształcenie danych wejściowych w dane, które warstwa wyjściowa może w jakiś sposób wykorzystać.



Rys. 1.8. Schemat modelu neuronu [8]

Z reguły wszystkie neurony w każdej warstwie są połączone z neuronami w innych warstwach. Umożliwia to pełny proces uczenia się, a uczenie jest najbardziej wydajne, jeśli wagi połączeń są aktualizowane po każdej iteracji.

Proces uczenia sieci neuronowej jest odpowiednikiem projektowania algorytmu (modyfikowanie wag połączeń). W zwykłym programowaniu istnieją ustalone zasady,

warunki, dzięki którym komputer oblicza wyniki, natomiast w sieciach neuronowych to sieć musi nauczyć się zasad dla danych informacji. Sieci neuronowe mogą się uczyć na trzy sposoby [2]:

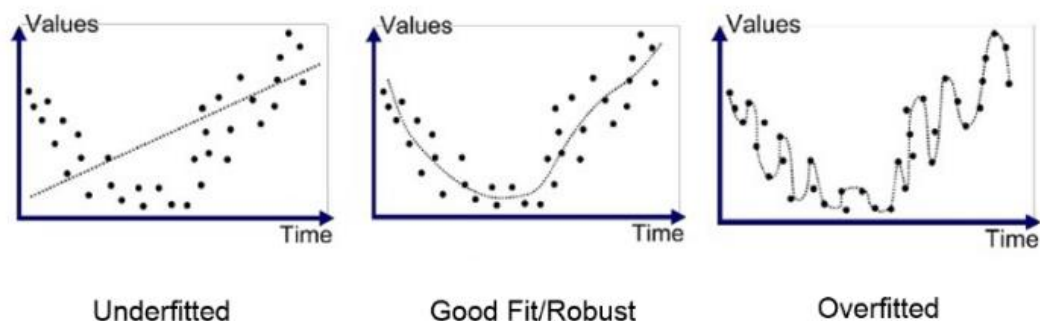
- **Uczenie nadzorowane** (ang. Supervised learning) – stosowany jest zbiór danych wejściowych oznaczonych etykietami (dobrymi odpowiedziami). Podczas nauki sieć dostraja wagi połączeń między neuronami, modyfikuje funkcje wyjścia do momentu, gdy zminimalizuje wartość między wartością oczekiwaną a nauczoną,
- **Uczenie nienadzorowane** (ang. Unsupervised Learning) – strategia wykorzystywana, gdy nie ma dostępu do zestawu danych z etykietami. W czasie trwania nauki sieć musi sama analizować dane, odnajdywać wzorce i relacje. Na podstawie funkcji kosztu, która informuje sieć neuronową jak blisko była od celu, modyfikuje swoje parametry w celu zwiększenia dokładności algorytmu,
- **Uczenie ze wzmocnieniem** (ang. Reinforcement Learning) - strategia, w której sieć otrzymuje zestaw gotowych reguł i zestaw dozwolonych działań. Można powiedzieć, że jest to metoda prób i błędów. Sieć neuronowa jest nagradzana za wyniki pozytywne i karana za wyniki negatywne. Wiedząc jak ocenione zostają decyzje, modyfikuje algorytm, aby dostosować się do pozytywnych ocen.

Sieci neuronowe zazwyczaj wykonują zadania w sposób nadzorowany, budując wiedzę z zestawów, na które z góry udzielono właściwej odpowiedzi. Następnie sieci uczą się, dostrajając wagi połączeń, aby znaleźć poprawną odpowiedź. Żeby to zrobić sieć porównuje otrzymane wyniki z poprawną odpowiedzią, odpowiada za to funkcja kosztu (opisana w dalszej części tekstu). Wartość błędu (różnica pomiędzy wartością oczekiwaną a uzyskaną) jest przepychana z powrotem przez wszystkie połączenia i neurony w celu dostosowania wartości biasów i wag połączeń. Ta metoda przechodzenia przez sieć od wyjścia do początku nazywana jest propagacją wsteczną i jest najważniejszym aspektem tego, jak sieć neuronowa uczy się określonego zadania [18].

Funkcja kosztu służy do sprawdzenia, jak bardzo trenowany model się myli. Celem w uczeniu jest znalezienie minimum funkcji kosztu. W tym celu stosowane są różne algorytmy, na przykład algorytm spadku gradientowego, który polega na iteracyjnym

podążaniu w kierunku minimum funkcji, zmieniając wagi aż osiągnie minimum. Wynik każdej iteracji jest zależny od parametru α zwanym parametrem szybkości uczenia (ang. learning rate). Upraszczając, jeśli α ma za małą wartość, to uczenie jest wolne, a jeśli za duże, to algorytm może nie znaleźć minimum [18].

Do trenowania modelu używany jest zbiór treningowy, który jest używany do dopasowania parametrów modelu (wag połączeń pomiędzy neuronami). Podczas szkolenia predykcje trenowanego modelu są używane do obserwacji, jak dobrze model się uczy używając danych ze zbioru walidacyjnego. Zestaw danych walidacyjnych zapewnia obiektywną ocenę, ponieważ zawiera dane, których nie zawiera zbiór treningowy. Walidacja może być wykorzystywana do wcześniejszego zatrzymania szkolenia modelu, gdy zacznie się zwiększać błąd w zestawie danych walidacyjnych [19]. Takie zjawisko nazywane jest przeuczeniem (ang. overfitting) i oznacza, że model za bardzo dopasowuje swoje parametry do konkretnych danych ze zbioru treningowego. Przeuczenie występuje, jeśli model zaczyna zapamiętywać przykłady ze zbioru testowego zamiast uczyć się z nich zależności – generalizować informację (Rys. 1.9). Aby sobie poradzić z przeuczeniem, można użyć regularyzacji, metody dropout (porzucania połączeń pomiędzy neuronami i zmuszanie sieci do znalezienia nowych połączeń) albo po prostu zatrzymać szkolenie wcześniej, bo może model już się nauczył pożądanych właściwości (metoda early stopping). Sposoby uporania się z przeuczeniem zostały bardziej szczegółowo opisane w rozdziale trzecim. Zbiór danych testowych służy do oceny ostatecznego dopasowania modelu na danych, których model jeszcze nie „widział” [20].

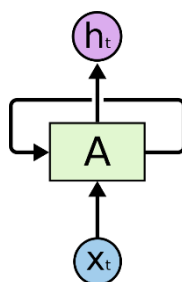


Rys. 1.9. Dopasowanie modelu do danych [20]

1.4. Sieci Long short-term memory

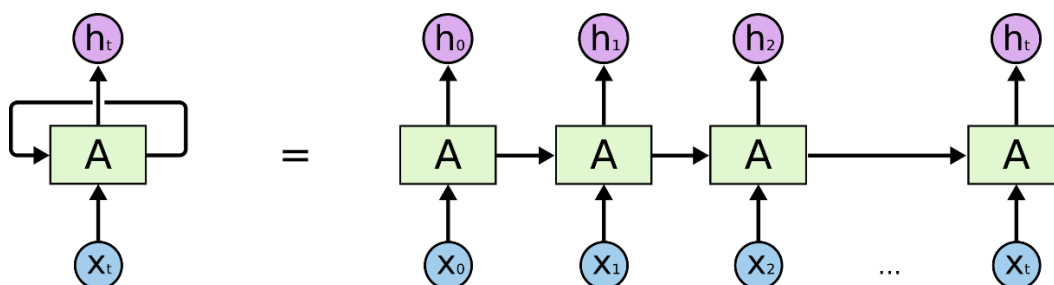
Ludzie nie zaczynają co sekundę myśleć „od zera”. Czytając tekst ludzie rozumieją każde słowo na podstawie zrozumienia poprzednich słów. Ludzkie myśli są trwałe i widząc nowe informacje ludzie nie zapominają wszystkiego, co wiedzieli dotychczas - nie zaczynają myśleć od nowa, ponieważ mają już pewien zbiór myśli i doświadczeń, który mogą wykorzystać [9].

Tradycyjne sieci neuronowe nie potrafią wykonać takich operacji i do niektórych systemów są nieodpowiednie. Przykładowo, klasyfikując jakie wydarzenie dzieje się w danym punkcie filmu, zwykła sieć neuronowa nie mogłaby wykorzystać informacji dotyczących poprzednich wydarzeń występujących w filmie. Rekurencyjne sieci neuronowe rozwiązują ten problem. Są to sieci z pętlami, które umożliwiają zachowanie poprzednich informacji [9].



Rys. 1.10. Uproszczony schemat neuronu w rekurencyjnej sieci neuronowej [9]

Na Rys. 1.10 został przedstawiony ideowy schemat neuronu w rekurencyjnej sieci neuronowej. Blok sieci A bierze dane wejściowe x_t i wyprowadza wartość h_t . Pętla natomiast umożliwia przekazywanie informacji z jednego etapu do następnych. Rekurencyjną sieć neuronową można traktować jako wiele kopii tej samej sieci rozwiniętej w czasie (Rys. 1.11), z których każda przekazuje informacje do następcy [9].



Rys. 1.11. Rozwinięty w czasie neuron w rekurencyjnej sieci neuronowej [9]

Można też myśleć o rekurencyjnych sieciach neuronowych jako o jednostkach posiadających „pamięć”, która przechowuje informacje o obliczonych wcześniej wynikach. W ciągu ostatnich lat rekurencyjne sieci neuronowe z sukcesem znajdują zastosowanie w [9]:

- Rozpoznawaniu, syntezie mowy,
- Sterowaniu robotami, przewidywaniu trajektorii samochodów autonomicznych w celu unikania wypadków,
- Kompozycji muzycznej,
- Sprawdzaniu poprawności gramatycznej,
- Rozpoznawaniu ludzkich działań,
- Przewidywaniu tekstu.

Istotne dla tych sukcesów jest zastosowanie Long short-term memory (LSTM), czyli bardzo szczególnego rodzaju rekurencyjnej sieci neuronowej, która działa znacznie lepiej w przypadku wielu zadań niż wersja standardowa. Wszystkie rekurencyjne sieci neuronowe mają pętle sprzężenia zwrotnego, która umożliwia zapamiętanie poprzednich danych. W przypadku długoterminowych zależności szkolenie RNN (rekurencyjnych sieci neuronowych) może okazać się trudne, ponieważ sieć będzie zapamiętywać wszystkie poprzednie informacje [9].

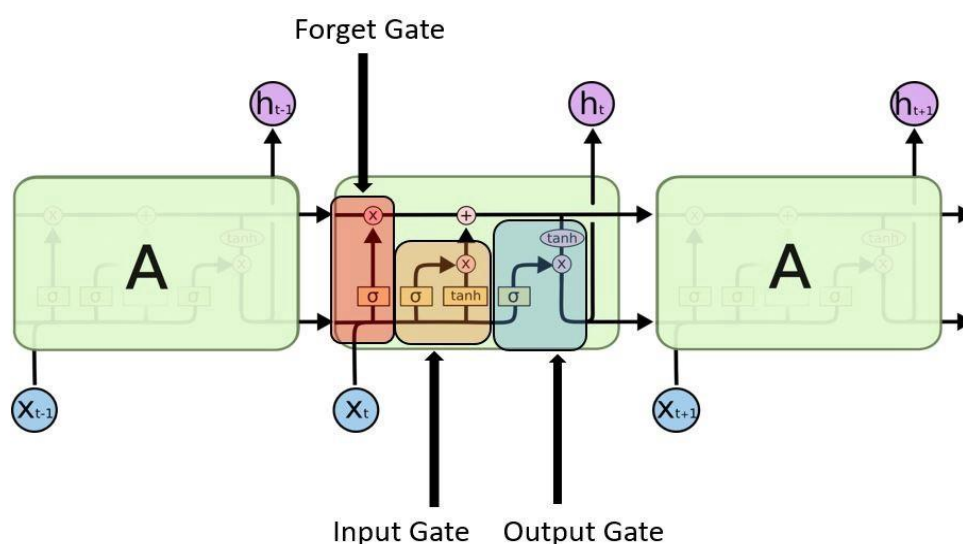
Sieci LSTM są specjalnym rodzajem RNN, zdolnym do uczenia się zależności długoterminowych. Zostały one wymyślone przez Seppa Hochreitera i Jurgena Schmidhubera w 1997 roku, a następnie dopracowane i spopularyzowane przez wiele innych osób, między innymi Feliksa Gersa, który wprowadził wraz z Schmidhuberem i Cumminsem bramkę zapominania w 1999 roku. W 2000 roku Schmidhuber i Cummins dodali do architektury połączenia komórki z bramą. Czternaście lat później, w 2014 roku, Kyunghyun Cho zaproponował uproszczony wariant o nazwie Gated Recurrent Unit (GRU), który posiada dwie bramki: bramkę resetowania i bramkę aktualizacji.

LSTM oprócz jednostek standardowych posiada również specjalne bramki (Rys. 1.12), służące do kontroli zapamiętywanych danych [10]:

1. Bramka wejściowa (ang. Input Gate) – odnajduje, którą wartość z wejścia wykorzystać do modyfikacji pamięci. Funkcja sigmoidalna decyduje, które wartości przepuścić (0 albo 1), a funkcja tanh (tangens hiperboliczny) nadaje

wagę przekazywanym wartościom, oceniając ich poziom ważności w przedziale od -1 do 1,

2. Bramka zapominania (ang. Forget Gate) – dowiaduje się, jakie szczegóły należy usunąć z bloku (decyduje o tym funkcja sigmoidalna). Sprawdza poprzedni stan h_{t-1} , wartość wejścia x_t i oblicza liczbę z przedziału od 0 (zapomnij) do 1 (zachowaj),
3. Bramka wyjściowa (ang. Output Gate) – wejście i pamięć bloku są używane do decydowania o wyjściu. Funkcja sigmoidalna decyduje, które wartości przepuścić (0 albo 1). Funkcja tanh decyduje o ważności sygnału (wartości od -1 do 1) i jest przemnożona z wyjściem funkcji sigmoidalnej.



Rys. 1.12. Architektura neuronu LSTM [10]

RNN z wykorzystaniem jednostek LSTM może być trenowana w sposób nadzorowany na zestawie sekwencji treningowych. Wartości błędów są propagowane wstecz z warstwy wyjściowej i zostają w komórkach jednostki LSTM. Błędy są wysyłane z powrotem do każdej bramki jednostki LSTM, aż bramki nie nauczą się usuwać tej wartości.

Do znaczących sukcesów sieci rekurencyjnych z jednostkami LSTM w ostatnich latach należą [11]:

- Od 2016 roku duże firmy technologiczne, w tym Google, Apple i Microsoft zaczęły stosować LSTM, jako podstawowe komponenty w nowych produktach. Na przykład Google zastosował LSTM do rozpoznawania mowy w smartfonie i Tłumacza Google. Apple używa LSTM do funkcji „Quicktype” na iPhone i Siri,
- W 2017 roku Facebook wykonywał dziennie około 4.5 miliarda automatycznych tłumaczeń przy użyciu LSTM,
- W 2018 roku boty opracowane przez OpenAI pokonały ludzi w grze Dota 2,
- W 2018 roku OpenAI przeszkolił LSTM do kontroli ręki robota podobnej do ludzkiej, która manipuluje obiektami z ogromną zręcznością,
- W 2019 roku program AlphaStar produkcji Google’a osiągnął poziom arcymistrza w skomplikowanej grze strategii czasu rzeczywistego StarCraft II (dla każdego ruchu istnieje 10^{26} możliwych wyborów). Program jest w stanie pokonać 99,8% graczy w rywalizacji.

2. Przewidywanie tekstu

2.1. Opis użytych narzędzi

Do realizacji wybranych algorytmów użyto języka Python 3.6. Jest to język programowania wysokiego poziomu ogólnego przeznaczenia stworzony przez Guida van Rossuma w 1991 roku. Głównymi zaletami Pythona są [12]:

- Wsparcie dla programowania obiektowego, proceduralnego, imperatywnego oraz funkcyjnego,
- Duży zbiór bibliotek standardowych i rozbudowana społeczność programistów,
- Prosty i czytelny kod,
- Automatyczne zarządzanie pamięcią,
- Dynamiczne typowanie zmiennych.

Przy tworzeniu aplikacji wykorzystano biblioteki zewnętrzne:

- tensorflow, keras – implementacja sieci neuronowych (opisane poniżej),
- numpy – operacje na tablicach, funkcje matematyczne,
- random – obsługa losowania,
- string – stałe, na przykład stała string.digits zawierająca wszystkie cyfry jako tekst,
- glob – wyszukiwanie nazw plików w folderach,
- sklearn – rozdzielenie danych na treningowe i walidacyjne,
- matplotlib – tworzenie charakterystyk.

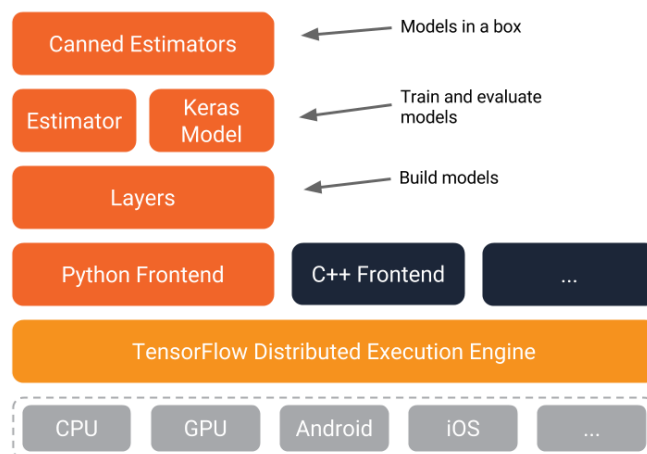
Do implementacji sieci neuronowych zastosowano Tensorflow 2.0. Jest to biblioteka programistyczna stworzona na licencji Open Source przez Google Brain Team w 2015 roku. Stosowana jest w uczeniu maszynowym i sieciach neuronowych [13]. Tensorflow może podczas działania wykorzystywać procesory (ang. CPU), karty graficzne (ang. GPU) oraz wyspecjalizowane mikroprocesory (ang. TPU).

Do zalet Tensorflow należą:

- Łatwe budowanie modeli za pomocą wysokopoziomowego interfejsu,
- Obsługa wielu języków programowania,
- Niezależność od środowiska, możliwość wdrażania modeli w chmurze.

Biblioteka składa się z kilku modułów (Rys. 2.1) [14]:

- Rozproszony silnik wykonawczy (ang. Distributed Execution Engine) znajduje się w najniższej warstwie i w celu podniesienia wydajności został stworzony w języku programowania C++,
- Nakładki (frontendy) napisane w kilku językach programowania, w tym w Pythonie oraz C++,
- Interfejs, który zapewnia prostszy sposób obsługi dla często używanych warstw w modelach głębokiego uczenia,
- Interfejsy wyższego poziomu, w tym Keras i API Estimator.



Rys. 2.1. Architektura biblioteki Tensorflow [14]

Użyty w aplikacji interfejsem Tensorflow jest Keras. Został zaprojektowany, aby umożliwić szybkie eksperymentowanie z sieciami neuronowymi i jest nastawiony na łatwą obsługę przez użytkownika. Został on opracowany w ramach projektu ONEIROS (ang. Open-ended Neuro-Electronic Intelligent Robot Operating System) przez inżyniera Google’a, Francois Cholleta. W 2017 roku został wprowadzony do podstawowej

biblioteki Tensorflow, oferując intuicyjną warstwę abstrakcji, która pozwala opracowywać modele sieci neuronowych bez wgłębiania się w zaplecze obliczeniowe. Do atrybutów interfejsu Keras należą [16]:

- Łatwość obsługi – spójny i prosty interfejs minimalizujący liczbę działań, które użytkownik musi wykonać, aby osiągnąć zamierzony efekt,
- Modułowość – warstwy neuronowe, funkcje kosztu, aktywacji, regularyzacji oraz optymalizatory są niezależnymi modułami, które można łączyć, aby stworzyć nowy model,
- Rozszerzalność – dodawanie nowych modułów w prosty sposób, jako nowe klasy i funkcje,
- Duża ilość przykładów i materiałów pomocniczych dostępnych w Internecie.

Model, który jest wykorzystywany w interfejsie Keras, to model Sekwencyjny (ang. Sequential). Jest to liniowy stos warstw. Model musi wiedzieć, jakiego kształtu danych powinien się spodziewać. Pierwsza warstwa w modelu sekwencyjnym musi otrzymać informacje o swoim kształcie wejściowym (pozostałe warstwy nie). Zanim model będzie mógł zostać wytrenowany, trzeba skonfigurować proces uczenia się. Konfiguracja obejmuje wybór optymalizatora, funkcji strat oraz metrykę i odbywa się metodą kompilacji [21]. Do metod modelu sekwencyjnego, które zostały wykorzystane w aplikacji, należą [22]:

- `compile(optimizer, loss=None, metrics=None, loss_weights=None, sample_weight_mode=None, weighted_metrics=None, target_tensors=None)`
- konfiguruje model do trenowania (dostępnych jest wiele optymalizatorów, funkcji strat oraz metryk, które można wybrać),
- `fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=None, validation_freq=1, max_queue_size=10, workers=1, use_multiprocessing=False)`
- szkoli model dla skończonej liczby iteracji określanej liczbą epok (parametr `epochs`). Do trenowania modelu należy dostarczyć zarówno dane treningowe jak i walidacyjne. Parametr `batch_size` określa liczbę danych, które są wprowadzane

do sieci w tym samym momencie. Do wywoływania funkcji wczesnego zatrzymywania szkolenia albo generowania tekstu co epokę zostało użyte tzw. wywołanie zwrotne (ang. `callback`),

- `predict(x, batch_size=None, verbose=0, steps=None, callbacks=None, max_queue_size=10, workers=1, use_multiprocessing=False)`
- generuje wynik predykcji dla ustalonych danych wejściowych.

Do modelu sekwencyjnego dodawane są warstwy używając metody `add`. W argumentach metody dostępne są takie parametry jak: liczba jednostek w warstwie, funkcja aktywacji, inicjatory oraz regularyzacja. W aplikacji zostały użyte warstwy:

- `Dense` – typowa warstwa o połączeniach gęstych, czyli „każdy z każdym”,
- `LSTM` – rekurencyjna warstwa typu Long short-term memory,
- `Embedding` – przekształca liczby całkowite w wektory o ustalonym rozmiarze (musi być użyta jako pierwsza warstwa), głównym celem tej warstwy jest redukcja wymiarowości danych wejściowych,
- `BatchNormalization` – normalizuje aktywacje poprzedniej warstwy,
- `Dropout` – wykonuje regularyzację metodą dropout.

W procesie programowania wykorzystano chmurowe środowisko Google Colab. Jest to bezpłatna usługa Google’a w chmurze, która została stworzona, aby zachęcić do badań nad uczeniem maszynowym i sztuczną inteligencją, gdzie często napotykanym ograniczeniem jest ogromne zużycie mocy obliczeniowej. W ramach usługi otrzymujemy dostęp do różnych urządzeń (Tab. 2.1): procesora, pamięci RAM, układu GPU, a nawet do układu TPU (w zależności od dostępności w danym czasie).

Tab. 2.1. Lista dostępnych urządzeń w usłudze Google Colab

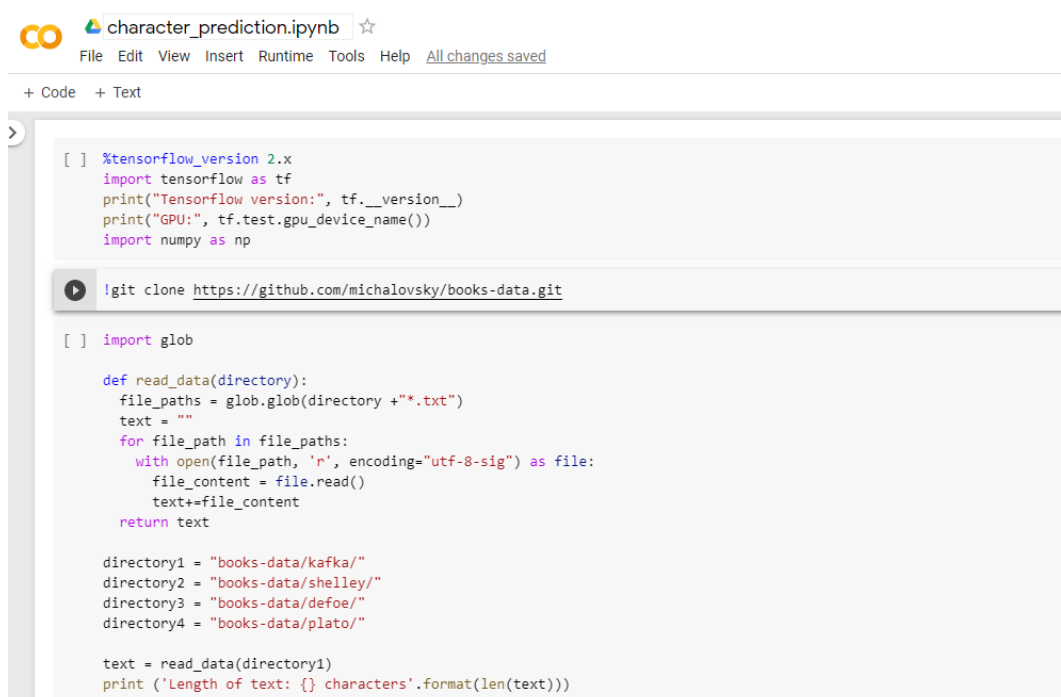
<pre>name: "/device:CPU:0" device_type: "CPU" memory_limit: 268435456 locality { } incarnation: 4144757193918400718</pre>
<pre>name: "/device:GPU:0" device_type: "GPU" memory_limit: 15956161332 locality { bus_id: 1 links { } } incarnation: 8138525966517145665 physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0"]</pre>

Do głównych powodów użycia Google Colab w procesie budowy aplikacji zaliczają się [17]:

- Integracja z GitHub – zapisywanie zmian w swoim repozytorium na GitHub,
- Wsparcie dla Pythona 2.7 i Pythona 3.7,
- Dostęp do darmowych zasobów z dużą mocą obliczeniową (uczenie modeli wykonano na przydzielonej karcie graficznej Tesla K80, która posiada 24GB pamięci GDDR5 384-bit o przepustowości 480GB/s),

- Zainstalowane biblioteki, gotowe do zaimportowania do projektu,
- Niezależność platformy od sprzętu i systemu operacyjnego – dostęp przez przeglądarkę internetową,
- Automatyczny zapis projektów na dysku Google.

Na rysunku 2.2 został przedstawiony wygląd okna głównego w otwartym programie do przewidywania znaków w usłudze Google Colab.



```
[ ] %tensorflow_version 2.x
import tensorflow as tf
print("Tensorflow version:", tf.__version__)
print("GPU:", tf.test.gpu_device_name())
import numpy as np

!git clone https://github.com/michalovsky/books-data.git

[ ] import glob

def read_data(directory):
    file_paths = glob.glob(directory + "*.txt")
    text = ""
    for file_path in file_paths:
        with open(file_path, 'r', encoding="utf-8-sig") as file:
            file_content = file.read()
            text += file_content
    return text

directory1 = "books-data/kafka/"
directory2 = "books-data/shelley/"
directory3 = "books-data/defoe/"
directory4 = "books-data/plato/"

text = read_data(directory1)
print('Length of text: {} characters'.format(len(text)))
```

Rys. 2.2. Okno główne otwartej aplikacji w usłudze Google Colab

2.2. Modelowanie języka

Języki formalne tak samo jak języki programowania można w pełni określić, definiując zastrzeżone słowa i ich prawidłowe sposoby użycia. Jednak nie można tego dokonać, jeśli chodzi o język naturalny, ponieważ języki naturalne nie zostały świadomie zaprojektowane (nie mają formalnej definicji). Języki naturalne obejmują ogromną liczbę terminów, które mogą być używane w wielu kontekstach i wprowadzają dwuznaczności - jednak ludzie potrafią je zrozumieć. Pomimo tego, że język naturalny jest żywy (zmieniają się słowa i ich użycia), to lingwiści starają się określić język za pomocą

formalnych gramatyk i struktur. Alternatywnym podejściem do określania modelu języka jest uczenie się go na przykładach [23].

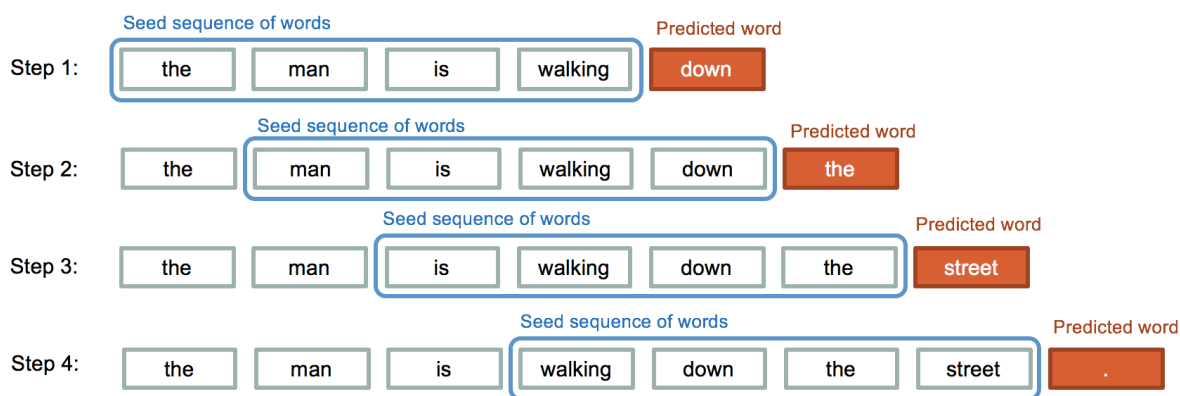
Statystyczne modelowanie języka polega na przypisywaniu prawdopodobieństwa wystąpienia następnego, konkretnego słowa do każdej sekwencji słów. Model językowy uczy się prawdopodobieństwa wystąpienia słowa na podstawie przykładów z tekstu. Proste modele działają na poziomie sekwencji słów, natomiast te bardziej złożone na poziomie zdań albo akapitów (najczęściej modele działają na poziomie słów). Modelowanie językowe jest stosowane w aplikacjach takich jak: rozpoznawanie mowy, rozpoznawanie ręcznego pisma, tłumaczenie maszynowe. Podejście oparte na sieci neuronowej (ang. Neural Language Models, skrót NLM) uzyskuje lepsze wyniki niż metody klasyczne. Szczególnie przydatnym jest mapowanie słów na wektory o wartościach rzeczywistych (ang. word embedding). Taka reprezentacja słów zdefiniowana w oparciu o ich użycie pozwala słowom o podobnym znaczeniu mieć podobną reprezentację. To uogólnienie jest czymś, co jest bardzo ciężko osiągnąć metodami statystycznymi. Ponadto word embedding pozwala na lepsze skalowanie reprezentacji wraz ze wzrostem rozmiaru słownika (liczby używanych słów). Początkowo do wprowadzenia podejścia opartego na NLM zastosowano sieci neuronowe jednokierunkowe (ang. feed-forward), później rekurencyjne sieci neuronowe, aż w końcu sieci neuronowe rekurencyjne oparte o LSTM. Opisane w poprzednim rozdziale sieci LSTM pozwalają modelom na poznanie kontekstu w dłuższych sekwencjach i odkrycie zależności długoterminowych. Sieci jednokierunkowe mogą tylko grupować podobne słowa, a sieci rekurencyjne mogą przeprowadzać grupowanie podobnych historii. Pozwala to modelom, opartym na rekurencyjnych sieciach neuronowych, na wydajną reprezentację sekwencji o zmiennej długości. W publikacji „Exploring the Limits of Language Modeling” autorzy proponują następujące wskazówki w celu opracowania wydajnych NLM [23]:

- Duży rozmiar modelu (duża liczba jednostek pamięci),
- Zastosowanie regularyzacji metodą dropout (rezygnacja z niektórych połączeń),
- Przewidywanie wyniku z wielu modeli może poprawić wydajność.

2.3. Implementacja modelu języka naturalnego

Celem aplikacji jest opracowanie modelowania języka naturalnego za pomocą sieci neuronowych. Model sieci powinien przewidywać następny element sekwencji. Aby w pełni zobrazować proces przewidywania tekstu przygotowano dwa programy: pierwszy z nich bazuje na znakach i przewiduje kolejny znak, natomiast drugi jest oparty na wyrazach i przewiduje kolejny wyraz. Zasadniczo modele bazujące na słowach wykazują większą dokładność niż modele na poziomie znaków, ponieważ mogą tworzyć krótsze reprezentacje zdań i łatwiej zachowują kontekst pomiędzy słowami (w jednym zdaniu model oparty na słowach musi przewidzieć poprawnie kilka wyrazów, natomiast model oparty na znakach kilkadziesiąt znaków). Model bazujący na znakach wymaga mniej pamięci i może zostać szybciej wytrenowany (wyjściem jest kilkadziesiąt neuronów odpowiadających liczbie unikalnych znaków, a w przypadku modelu opartego na wyrazach kilka, kilkanaście lub kilkadziesiąt tysięcy) [28].

W celu pokazania, jak model przewiduje kolejne elementy (Rys. 2.3) generowane są kolejne znaki/wyrazy tworząc sekwencje, które powinny przypominać styl pisania, w jakim zostały napisane teksty, na których przeprowadza się uczenie modelu. W budowaniu modeli wykorzystano rekurencyjne sieci neuronowe oparte o jednostki LSTM (opisane w poprzednim rozdziale). Zadaniem sieci jest przewidywanie następnego znaku/słowa patrząc na sekwencję, która występuje przed nim (rekurencyjne sieci neuronowe doskonale się sprawdzają w takich zastosowaniach, ponieważ są w stanie utrzymywać i przenosić informacje z przeszłości do następnych kroków uczenia). Wybrano jednostki LSTM zamiast zwykłych jednostek sieci rekurencyjnej, ponieważ sieć oparta na LSTM nie zapamiętuje wszystkich informacji z przeszłości, tylko te, które uzna za istotne a resztę odrzuca. Do implementacji modelu sieci neuronowej została użyta biblioteka Keras (opisana w rozdziale 2.1), która umożliwia eksperymentowanie z sieciami neuronowymi z poziomu wysokopoziomowego interfejsu.



Rys. 2.3. Proces przewidywania następnego elementu bazując na wyrazach [24]

Kod źródłowy aplikacji został napisany na podstawie publicznie dostępnych źródeł [25][26][27][28]. Oba programy do przewidywania tekstu zostały podzielone na sześć modułów:

1. Wczytywanie danych:

Użyto tekstów z książek, które nie są chronione prawami autorskimi, albo, których autor udzielił zgodę na rozpowszechnianie. Biblioteką internetową, która posiada takie teksty, głównie w języku angielskim, jest Project Gutenberg. Biblioteka pozwala na darmowe pobieranie tekstów, przetwarzanie ich oraz udostępnianie [30]. Książki, które były wczytywane podczas procesu budowy aplikacji to:

- „The Life and Adventures of Robinson Crusoe” autorstwa Daniela Defoe,
- „Metamorphosis” oraz „The Trial” autorstwa Franza Kafki,
- „The Republic” autorstwa Plato,
- „Frankenstein” autorstwa Mary Shelley,
- „The Count of Monte Cristo” autorstwa Alexandra Dumasa.

Wybrano książki, które różnią się ilością tekstu, aby szkolić model na różnych wielkościach zbiorów danych. Wyjątkiem jest książka „The Trial”, którą pobrano, aby mieć dwie książki jednego autora i próbować zmieszać obie książki, jako zbiór danych wejściowych. Pliki zawierające książki mają rozmiar:

- „Metamorphosis” – 117KB,
- „Frankenstein” – 413KB,
- „The Trial” – 442KB,

- „The Life and Adventures of Robinson Crusoe” – 610KB,
- „The Republic” – 1,2MB,
- „The Count of Monte Cristo” – 2,7MB.

Pliki zostały pobrane ze strony internetowej Project Gutenberg (<https://gutenberg.org/>) i zapisane do repozytorium na serwisie internetowym GitHub (https://github.com/michalovsky/books_data), z którego dane pobierane są do aplikacji (komenda `git clone`). W repozytorium stworzono foldery na twórczość każdego z autorów.

Funkcja odpowiadająca za wczytywanie danych (Tab. 2.2) wyszukuje wszystkie pliki tekstowe w danym folderze i zwraca ich zawartość.

Tab. 2.2. Funkcja wczytująca dane z plików tekstowych

```
def read_data(directory):
    file_paths = glob.glob(directory + "*.txt")
    text = ""
    for file_path in file_paths:
        with open(file_path, 'r', encoding="utf-8-sig") as file:
            file_content = file.read()
            text+=file_content
    return text
```

2. Preprocessing danych

Project Gutenberg dodaje specjalny nagłówek i stopkę do każdej książki, które wyznaczają, gdzie zawiera się prawdziwy tekst książki (przykładowy nagłówek i stopkę prezentuje tabela 2.3). Usunięto tekst nieznajdujący się pomiędzy nagłówkiem a stopką.

Tab. 2.3. Struktura pliku pobranego z biblioteki Project Gutenberg

START OF THE PROJECT GUTENBERG EBOOK THE LIFE AND ADVENTURES OF ROBINSON CRUSOE
Zawartość książki
END OF THE PROJECT GUTENBERG EBOOK THE LIFE AND ADVENTURES OF ROBINSON CRUSOE

Dane z wczytywanych tekstów zawierają bardzo dużo różnorodnych znaków. Aby zawęzić liczbę elementów, które model może przewidzieć, usunięto z tekstu cyfry, znaki interpunkcyjne (oprócz kropki i przecinka, ponieważ formują one szkielet zdania) oraz znaki specjalne (na przykład „&” czy „@”). W części tekstów występują wyrazy zawierające litery z innych języków, które te znaki zamieniono na angielskie odpowiedniki (na przykład „Miss Bürstner” na „Miss Burstner”). Kolejną metodą użytą, aby zmniejszyć ilość słów, jest nierozróżnianie wielkości liter (wszystkie litery zostały zamienione na małe). Wynik przetwarzania tekstu jest ukazany w tabeli 2.4.

Tab. 2.4. Unikalne znaki po preprocessingu

[' , ' , ' , ' , ' a ' , ' b ' , ' c ' , ' d ' , ' e ' , ' f ' , ' g ' , ' h ' , ' i ' , ' j ' , ' k ' , ' l ' , ' m ' , ' n ' , ' o ' , ' p ' , ' q ' , ' r ' , ' s ' , ' t ' , ' u ' , ' v ' , ' w ' , ' x ' , ' y ' , ' z ']

Metoda klasy DataProcessor (Tab. 2.5) odpowiadająca za preprocessing danych usuwa znaki niepożądane, tłumaczy znaki (na przykład „ą” na „a”) oraz zamienia wszystkie pozostałe znaki na małe. Dodatkowo usuwa wszystkie nadmierne białe znaki, pozostawiając jedną spację między słowami.

Tab. 2.5. Klasa odpowiadająca za preprocessing danych

```
class DataProcessor:
    def __init__(self, chars_to_remove, chars_to_translate, replacement_chars):
        self.chars_to_remove = chars_to_remove
```

```

self.chars_to_translate = chars_to_translate
self.replacement_chars = replacement_chars

def preprocess_data(self, text):
    removal_translator = str.maketrans("", "", self.chars_to_remove)
    special_characters_translator = str.maketrans(self.chars_to_translate, self.replacement_
chars , "")
    text = text.lower().translate(removal_translator).translate(special_characters_translator
)
    text = " ".join(text.split())
    return text

```

3. Przygotowanie zbiorów danych

- Przewidywanie znaków:

Na tekście zastosowano mapowanie unikalnych znaków na liczby (29 liczb, tabela 2.6). Tekst został podzielony na sekwencje po 40 znaków przesuwając się iteracyjnie po całym tekście znak po znaku. Każda taka sekwencja posiada swoją etykietę, którą jest znak występujący po tej sekwencji.

Tab. 2.6. Mapowanie znaków na liczby na przykładzie sekwencji po 40 znaków

one morning, when gregor samsa woke from
[17, 16, 7, 0, 15, 17, 20, 16, 11, 16, 9, 1, 0, 25, 10, 7, 16, 0, 9, 20, 7, 9, 17, 20, 0, 21, 3, 15, 21, 3, 0, 25, 17, 13, 7, 0, 8, 20, 17, 15]

- Przewidywanie wyrazów:

Podobnie jak w przewidywaniu znaków w tekście zastosowano mapowanie unikalnych wyrazów na liczby (około 5600 liczb, w zależności ile jest unikalnych wyrazów w tekście, tabela 2.7). Tekst również został podzielony na sekwencje, z tym, że w tym przypadku po 10 wyrazów, a etykietą jest następny wyraz.

Tab. 2.7. Mapowanie wyrazów na liczby na przykładzie sekwencji po 10 wyrazów

one morning , when gregor samsa woke from troubled dreams ,
[3366, 3155, 0, 5443, 2192, 4163, 5522, 2050, 5115, 1507, 0]

Następnie w obu przypadkach zastosowano konwersję każdego elementu sekwencji (liczby) na wektor w postaci One-Hot Encoding (Rys. 2.4). Kodowanie to polega na zamianie liczby na wektor zawierający same zera z jedną jedynką na miejscu unikalnym dla liczby.

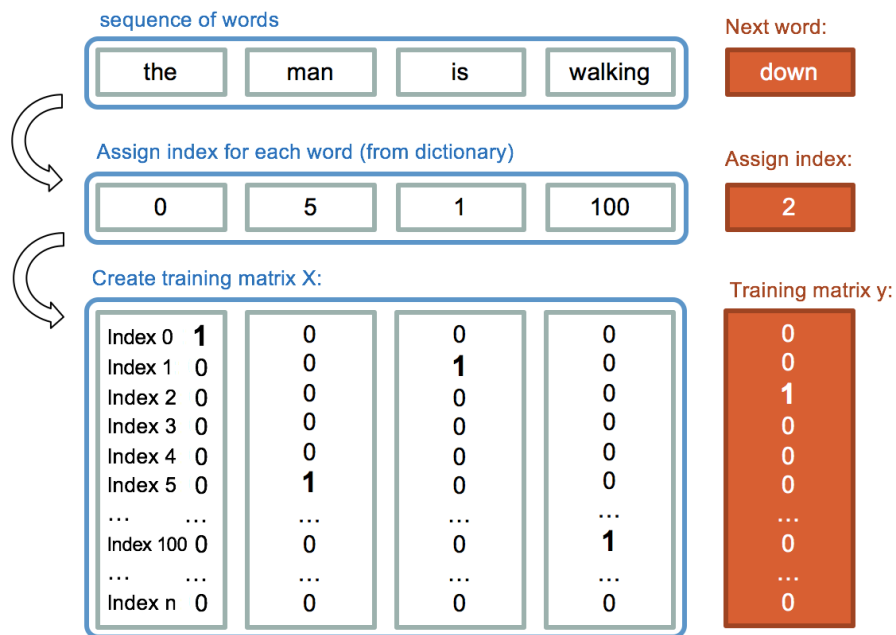
Dane wejściowe (X) mają wymiary:

1. Liczba sekwencji,
2. Liczba elementów w sekwencji,
3. Liczba znaków/unikalnych wyrazów (One-Hot Encoding).

Oznaczenia danych wejściowych (y) mają wymiary:

1. Liczba sekwencji,
2. Liczba znaków/liczba unikalnych wyrazów (One-Hot Encoding).

W następnym kroku rozdzielono dane na zbiór treningowy i walidacyjny w proporcji 5 do 1 (80% - zbiór danych treningowych, 20% - zbiór danych walidacyjnych). Dane jeszcze przed rozdzieleniem zostały pomieszane, aby zapewnić, że model nie będzie się uczyć sekwencji w takiej kolejności, w jakiej występują w tekście i zapamiętywać, że dana sekwencja występuje po innej sekwencji (model powinien się uczyć z etykiet a nie z kolejności wejścia sekwencji danych treningowych).



Rys. 2.4. Tworzenie zbiorów danych w postaci One-Hot Encoding [24]

Metoda `make_dataset` klasy `Dataset` (Tab. 2.8) jest odpowiedzialna za rozdzielenie tekstu w postaci listy słów na sekwencje o odpowiedniej długości. Oprócz tego dzieli te sekwencje na zbiory treningowe i walidacyjne (80% zbiór treningowy, 20% zbiór walidacyjny). Klasy odpowiedzialne za przygotowanie zbiorów danych zostały zaprojektowane analogicznie przy przewidywaniu znaków jak i wyrazów.

Tab. 2.8. Klasa odpowiedzialna za przygotowanie zbiorów danych

```
class Dataset:
    def __init__(self):
        self.text_sequences = list()
        self.X_train = list()
        self.y_train = list()
        self.X_val = list()
        self.y_val = list()

    def make_dataset(self, words, input_sequence_length=10, output_sequence_length=1):
        sequence_length = input_sequence_length + output_sequence_length
        tokens = self.__create_tokens(words, sequence_length)
```

```

self.__create_training_and_validation_set(tokens)

return self

def __create_tokens(self, words, sequence_length):
    encoded_words = [word_to_indices[word] for word in words]
    tokens = list()
    for i in range(sequence_length, len(words)):
        line = ' '.join(words[i-sequence_length:i])
        self.text_sequences.append(line)
        tokens.append(encoded_words[i-sequence_length:i])
    return tokens

def __create_training_and_validation_set(self, tokens):
    data = np.asarray(tokens)
    X, y = data[:, :-1], data[:, -1]
    y = np_utils.to_categorical(y, num_classes=vocab_size)
    self.X_train, self.X_val, self.y_train, self.y_val = train_test_split(X, y, test_size=0.2, s
uffle=True)

```

4. Stworzenie modelu sieci neuronowej

- Przewidywanie znaków:

Utworzono sieć neuronową w oparciu o rekurencyjne jednostki LSTM. Użyto dwóch warstw LSTM po 90 jednostek z funkcją aktywacji tanh (tangens hiperboliczny). Jako warstwę wyjściową użyto zwykłej warstwy sieci neuronowej Dense z liczbą jednostek równą liczbie unikalnych znaków i funkcją aktywacji Softmax. Funkcja Softmax normalizuje wynik wyjściowy podając go jako prawdopodobieństwo. Podsumowanie modelu wraz z liczbą parametrów jest pokazane na Rys. 2.5.

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 40, 90)	43200
lstm_3 (LSTM)	(None, 90)	65160
dense_1 (Dense)	(None, 29)	2639
Total params: 110,999		
Trainable params: 110,999		
Non-trainable params: 0		

Rys. 2.5. Informacje o modelu do przewidywania znaków

- Przewidywanie wyrazów:

Podobnie jak przy przewidywaniu znaków utworzono sieć neuronową opartą o LSTM. W tym przypadku, oprócz dwóch warstw LSTM po 100 jednostek, dodano też na wejście warstwę embedding (została opisana w rozdziale 2.2). Pomiędzy warstwami LSTM użyto warstwy BatchNormalization, która normalizuje wartości w ukrytych warstwach. Na warstwę wyjściową wybrano warstwę Dense z liczbą jednostek odpowiadającą liczbie unikalnych wyrazów oraz funkcją aktywacji Softmax. Podsumowanie modelu wraz z liczbą parametrów jest pokazane na Rys. 2.6.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 10, 50)	280550
lstm (LSTM)	(None, 10, 100)	60400
batch_normalization (BatchNo	(None, 10, 100)	400
lstm_1 (LSTM)	(None, 100)	80400
batch_normalization_1 (Batch	(None, 100)	400
dense (Dense)	(None, 5611)	566711
Total params: 988,861		
Trainable params: 988,461		
Non-trainable params: 400		

Rys. 2.6. Informacje o modelu do przewidywania wyrazów

Klasa Model (Tab 2.9) jest nakładką na model z interfejsu Keras implementującą konkretny model potrzebny do przewidywania tekstu. Model uzyskuje się przez stworzenie instancji tej klasy. W obu programach użyto do kompilacji modelu funkcji strat categorical cross entropy, która mierzy wydajność modelu, którego wyjściem jest prawdopodobieństwo (model określa prawdopodobieństwa wystąpienia każdego ze znaków/wyrazów). Do trenowania sieci użyto optymalizator Adam, który wykorzystuje zmieniający się learning rate, zmniejszany wraz ze wzrostem liczby iteracji uczenia modelu (epok). Optymalizator Adam wykorzystuje również algorytm Momentum, odpowiedzialny za unikanie minimów lokalnych podczas szukania globalnego minimum funkcji strat.

Tab. 2.9. Klasa odpowiedzialna za stworzenie i szkolenie modelu

```
class Model:
    def __init__(self):
        self.model = Sequential()
        self.__build_model()
        self.__compile_model()
        self.model.summary()

    def __build_model(self):
        self.model.add(Embedding(vocab_size, 50, input_length=input_sequence_length))
        self.model.add(LSTM(100, return_sequences=True, recurrent_initializer='glorot_uniform', kernel_constraint=max_norm(3)))
        self.model.add(BatchNormalization())
        self.model.add(LSTM(100, recurrent_initializer='glorot_uniform', kernel_constraint=max_norm(3)))
        self.model.add(BatchNormalization())
        self.model.add(Dense(vocab_size, activation='softmax'))

    def __compile_model(self):
        self.model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
def fit_model(self, X_train, y_train, validation_data, epochs, batch_size, callbacks):
    return self.model.fit(
        X_train,
        y_train,
        validation_data=validation_data,
        epochs=epochs,
        batch_size=batch_size,
        verbose=2,
        callbacks=callbacks
    )
```

5. Szkolenie sieci neuronowej

Do szkolenia sieci neuronowej wykorzystano przygotowane wcześniej zbiory treningowe i walidacyjne. Zbiór walidacyjny został użyty, aby sprawdzić czy model się uczy ogólnych reguł czy dostosowuje się i zapamiętuje dane treningowe (ang. overfitting). Zastosowano funkcje wywołań zwrotnych (ang. callbacks), które są wywoływane w każdej iteracji uczenia modelu:

- EarlyStopping – funkcja wbudowana w interfejs Kera, która sprawdza jakość uczenia się modelu patrząc na daną metrykę i zatrzymuje uczenie, jeśli nie ma lepszych rezultatów (jeśli EarlyStopping chce zatrzymać uczenie, ale jest ustawiony parametr patience, to funkcja poczeka podaną liczbę epok - jeśli nadal nie odnotuje poprawy wyników, to wtedy zatrzyma proces uczenia się modelu),
- LambdaCallback – funkcja wbudowana w interfejs Keras, która przyjmuje inną funkcję i wywołuje ją w danym momencie. W tym przypadku w LambdaCallback została użyta funkcja generująca tekst (Tab. 2.10), wywoływana w celu wizualnego sprawdzenia, jak model generuje nowy tekst.

Funkcją odpowiedzialną za szkolenie modelu jest metoda fit_model klasy Model (Tab. 2.9). Istotnym parametrem podczas procesu szkolenia sieci jest rozmiar partii danych (ang. batch size). Określa, ile danych jest propagowanych na raz do sieci. Dla małej partii danych istnieje duża możliwość przeuczenia, natomiast dla dużej partii

danych występuje uśrednienie metryki i mniejsza szansa na przeuczenie (wolniejsze uczenie).

6. Generowanie tekstu

Funkcja generująca tekst (Tab. 2.10) otrzymuje wartość początkową, od której model ma zacząć generowanie sekwencji. Model zwraca najbardziej prawdopodobny element i jest on dopisywany do wyniku oraz do danych wejściowych następnej iteracji.

Tab. 2.10. Funkcja odpowiadająca za generowanie tekstu (wyrazów)

```
def generate_text(model, seed, words_amount):
    result = list()
    input_text = seed
    for _ in range(words_amount):
        encoded_text = [word_to_indices[word] for word in input_text.split()]
        encoded_text = pad_sequences([encoded_text], maxlen=input_sequence_length, truncating='pre')
        predictions = model.predict_classes(encoded_text, verbose=0)
        predicted_word = indices_to_word[predictions[0]]
        input_text += ' ' + predicted_word
        result.append(predicted_word)
    result = ''.join(result).replace(" ,", ",").replace(" .", ".\n")
    return result
```

3. Opis otrzymanych wyników

Sprawdzając proces uczenia się modeli głównie brany był pod uwagę efekt wizualny tzn. generowany tekst w każdej iteracji nauki. Obserwując wyniki z kolejnych epok można się dowiedzieć, jak długi musi być proces uczenia sieci neuronowej, aby zaczęła przewidywać sensowny tekst. Szukając modelu sieci neuronowej, który spełni założenia projektowe, próbowano wielu kombinacji różnych parametrów, które wpływają, na jakość i szybkość uczenia modelu:

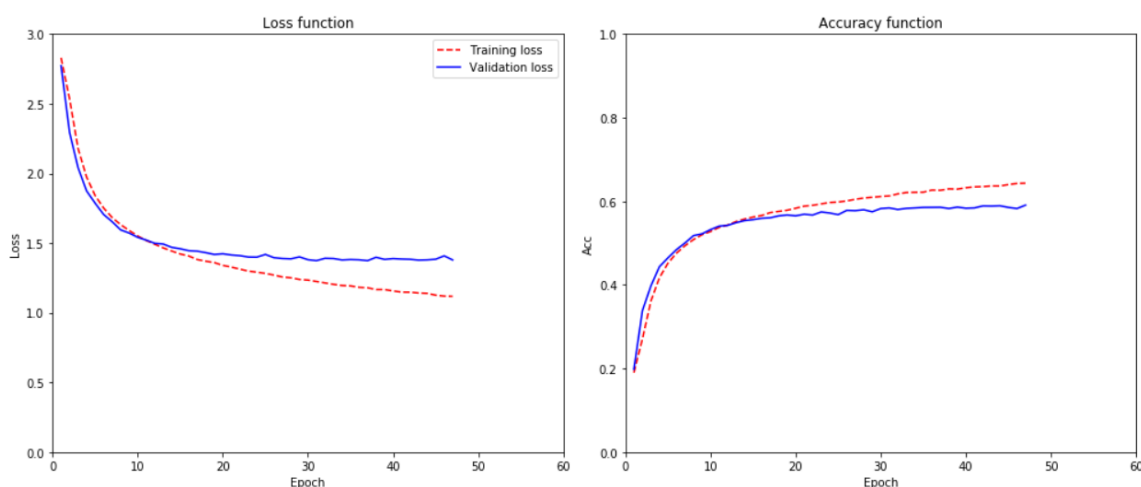
- Ilość warstw sieci neuronowej,
- Ilość jednostek w warstwie,
- Funkcje aktywacji neuronów,
- Parametr szybkości uczenia (ang. learning rate),
- Rozmiar partii (ang. batch size),
- Rozmiar sekwencji wejściowej,
- Ilość danych wejściowych.

Ze względu na ilość zmiennych i czas uczenia się sieci neuronowej, odpowiedni dobór powyższych parametrów był trudnym i czasochłonnym zadaniem. Aby sprawdzić poprawność jednej konfiguracji, trzeba było czekać co najmniej około godziny, aby zobaczyć rezultaty (dla małej ilości danych), a sprawdzanych konfiguracji było około siedemdziesięciu. W celu obserwacji wyników, oprócz predykcji wywoływanej co iterację uczenia modelu, użyto też danych walidacyjnych. Dane walidacyjne wprowadzone do procedury uczenia modelu pomagają sprawdzić, czy sieć uczy się w założony sposób. Sekwencje wejściowe, tak samo jak i odpowiadające im etykiety, są podzielone na dwa zbiory. Pierwszy z nich to zbiór treningowy i służy do faktycznego uczenia sieci. Drugi to zbiór walidacyjny. Podczas każdej iteracji model próbuje przewidzieć z sekwencjami walidacyjnymi rzeczywistą wartość (znak/słowo), a następnie porównuje je do etykiety odpowiadającej sekwencji walidacyjnej (29 różnych etykiet w przypadku modelu bazującym na znakach i kilka tysięcy w przypadku modelu bazującym na wyrazach). Taki mechanizm sprawdzania uczenia nazywany jest metryką. Metryka powinna mówić o rzeczywistym postępie nauki sieci neuronowej. O ile w przypadku sieci neuronowej użytej w przewidywaniu znaków, bezpośrednie porównywanie etykiety (znaku) odpowiadającej sekwencji walidacyjnej do rzeczywistej

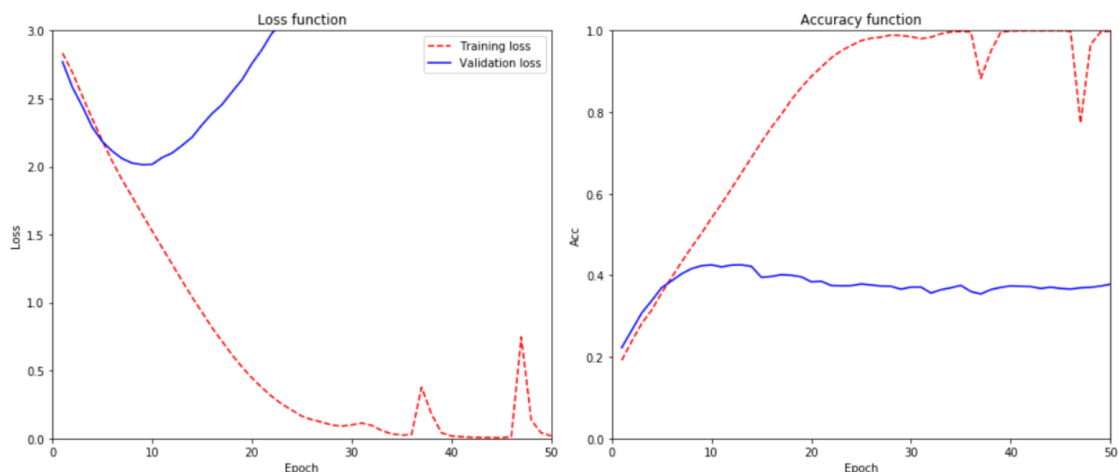
przewidzianej litery, ma uzasadnienie (odpowiednia metryka), to w przypadku przewidywania wyrazów, takie porównywanie jest nieodpowiednie. Tworząc jedno zdanie o takim samym znaczeniu, można użyć wielu różnych sposobów (zamiana kolejności wyrazów, inna konstrukcja gramatyczna). Sprawdzanie wystąpienia konkretnego słowa po sekwencji elementów może wskazywać, że model nie wykazuje postępów podczas nauki, gdy w rzeczywistości przewiduje coraz to bardziej sensowny tekst. Do otrzymania rzeczywistych wyników na zbiorze walidacyjnym powinna zostać użyta inna metryka (na przykład BLEU). Ze względu na ograniczenia czasowe nie udało się jej zaimplementować do aplikacji.

3.1. Wyniki aplikacji przewidującej znaki

Proces budowy projektu rozpoczęto od budowania aplikacji przewidywania znaków, zwracając uwagę głównie na dokładność predykcji na danych walidacyjnych. Pierwszy sensowny wynik osiągnięto dla sieci neuronowej składającej się z 3 warstw po 256 jednostek LSTM i 30 znakach wejściowych (Rys. 3.1). Znacznie zmniejszając ilość danych wejściowych możemy zaobserwować przeuczenie (Rys. 3.2). W tym przypadku model dopasowuje się do konkretnych danych wejściowych. Jak można zauważyć na funkcji strat na Rys. 3.2, gdy model uzyskuje coraz lepsze wyniki na danych treningowych, rezultaty na danych walidacyjnych są coraz gorsze. Oznacza to, że predykcje modelu w czasie uczenia są coraz mniej ogólne i nie sprawdzają się na innych danych.



Rys. 3.1. Funkcji strat i dokładności na modelu 3 warstwy po 256 jednostek neuronów



Rys. 3.2. Przykład charakterystyk wskazujących na przeuczenie modelu

Aby zlikwidować przeuczenie, można spróbować:

- Zwiększyć rozmiar danych wejściowych,
- Zmniejszyć liczbę warstw/jednostek w sieci neuronowej,
- Wprowadzić dane z wielu różnych zbiorów,
- Zmienić optymalizator,
- Wprowadzić regularyzację,
- Zwiększyć rozmiar partii danych,
- Pomieszać dane, aby sekwencje wprowadzane do sieci neuronowej nie występowały w tej samej kolejności, co w danych.

Zastosowany model był duży objętościowo (1,3 miliona parametrów) i trenowanie go zajmowało kilka godzin lub więcej (w zależności od ilości danych wejściowych). Docelowym modelem (opisanym w poprzednim rozdziale) okazał się model o wiele mniejszy, mający 2 warstwy po 100 jednostek LSTM. Na początku uczenia możemy zaobserwować regularność i powtarzalność danych (Tab. 3.1). Generowany tekst podczas procesu uczenia miał jedną wartość początkową (ang. seed) dla wszystkich iteracji, aby zobaczyć, jak bardzo się polepszy w czasie szkolenia. W czasie uczenia modelu (Tab 3.2) można zaobserwować zanikanie błędów w obrębie jednego wyrazu (nadal się zdarzają) oraz niezapętlanie się predykcji.

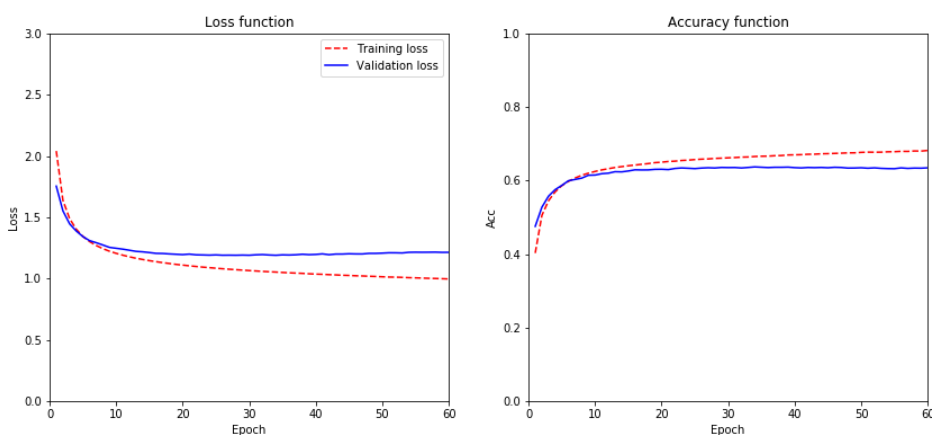
Tab. 3.1. Generowany tekst na początku uczenia modelu opartego na znakach

Epoch 1/80 aken it will do him no harm but the man simpen as out uf that have go nit not as fay penialy biked a cany uatr. but k himsalle that staided
Epoch 2/80 aken it will do him no harm but the man of cout ruaten. k draad to see still, is that if from but congectuped dont dost the buch be and if s
Epoch 3/80 aken it will do him no harm but the man afpeareffle, you that, what it, and poseinital, indiscisiately had been. say we looded. youre dowing

Tab. 3.2. Generowany tekst podczas uczenia modelu opartego na znakach

Epoch 57/80 Generated text: aken it will do him no harm but the man if you are on it, he would see, look at her assection asked k, so great enough to his sister well do
Epoch 58/80 Generated text: aken it will do him no harm but the man had explained to speak well with you any easier. before her pervers instoul was no longer very surpr
Epoch 59/80 Generated text: aken it will do him no harm but the man was the whole in this part of your understanding to humilistone a very key town he could only proped

Proces uczenia się sieci neuronowej można także zaobserwować na wykresach (Rys 3.3), gdzie na czerwono oznaczono wartości funkcji na danych treningowych, a na niebiesko wartości funkcji na danych walidacyjnych. Osiągnięto dokładność około 64% na danych walidacyjnych (losowa szansa 1 z 29 to 3.4%). Przykładowy wygenerowany tekst przedstawiono w Tabeli 3.3. Można zauważyć, że w obrębie słów rzadko występują błędy i kolejne słowa pasują do siebie, ale całościowo nie ma ten tekst jakiegos określonego znaczenia.



Rys. 3.3. Funkcje strat i dokładności w procesie uczenia sieci neuronowej opartej na znakach

Tab. 3.3. Wygenerowany z początkowym tekstem zaznaczonym na zielono

sible, interwoven babble of shouts and read how it was a secret of the floor, one of the man stretched her hand to go home on the same time, he was not behaved the money about the policemen was shocked on the gentlemen who was left him he would really need to start the while at the steps to see how the room to go to himself.

it was all the most of the stairs, he had expected the lawyer was clearly tired of the door of the steps to the lawyer took his face of them.

the light of the man was a superfisition has so that he was talking at the bed thing to be received a chair in his hand with the sort of the evening and the court officials.

3.2. Wyniki aplikacji przewidującej wyrazy

Kolejnym krokiem w celu udoskonalenia generowanego tekstu, było opracowanie aplikacji, w której sieć neuronowa uczy się na wyrazach. Struktura opracowanej sieci neuronowej jest podobna do struktury poprzedniej aplikacji. Istotną zmianą jest wprowadzenie warstwy word embedding (opisanej w poprzednim rozdziale). Skutkiem tej zmiany jest zwiększenie liczby parametrów do wyszkolenia w sieci neuronowej. Wzrosła ona prawie 20 krotnie, a wraz z tym czas szkolenia podczas każdej iteracji. Pierwszym podejściem było ustawienie liczby wyrazów w sekwencji wejściowej na 10. Na początku uczenia sieci neuronowej można zauważyć, że predykcje bardzo często się powtarzają i zapętłają (Tab. 3.4). Po około 90 iteracjach uczenia (Tab. 3.5) tekst generowany przez sieć neuronową nadal nie jest idealny, ale jest znacznie wyższej jakości niż w początkowej fazie uczenia.

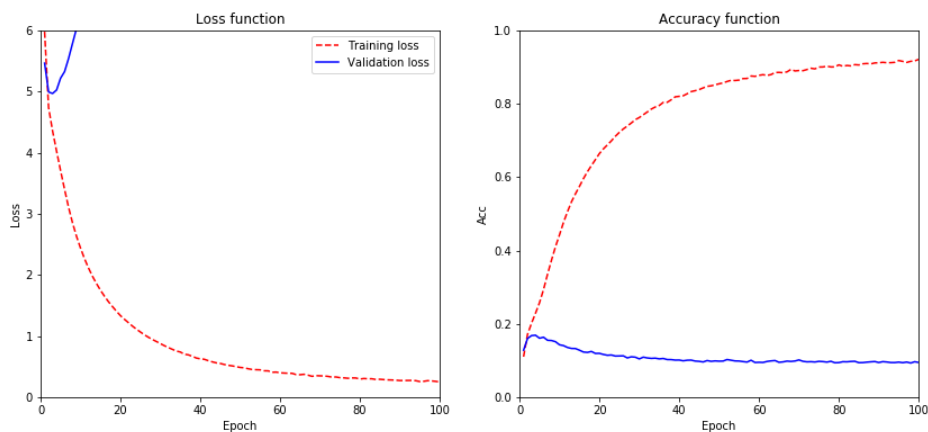
Tab. 3.4. Generowany tekst na początku uczenia modelu opartego na wyrazach

Epoch 1/100
Generated text: i had been very much as i had been to be much as i had been to be much as i had been to be much as i had been to be much as i had been to be much as i had been to be much as i had
Epoch 2/100
Generated text: i had been a little way to be a little, and that i had been a little way to be a little, and that i had been a little way to be a little, and that i had been a little way to be a little,
Epoch 3/100
Generated text: i had been a great deal of a great deal of. in the next day, i began to be very thankful for, and i had no notion of my mind, and to think of the ship, and i had made a great deal of it

Tab. 3.5. Generowany tekst w trakcie uczenia modelu opartego na wyrazach

Epoch 93/100 Generated text: place there was any nothing for it but my own wooden swords savages, in the wreck of the mainland, and immediately to carry only one way to another. in a word, i turned pale, and grew sick, and had in being seen at our
Epoch 94/100 Generated text: it was nothing but to spare or any part of every day. on the morning we came to the poor ship, and that he did not only go. he looked immediately within he knew the ship soever the bear. i cried out, he said he
Epoch 95/100 Generated text: deliver my place know that i would deliverance, who was now almost open, and particularly the best of six for the boy was too late. that day the next day the wind came to middle parts of the sea, and never known upon the heaven,

Funkcja strat i dokładności, użyta w poprzedniej aplikacji, nie jest odpowiednia dla modelu opartego na wyrazach (Rys. 3.4). Do oceny predykcji tekstu można użyć metryk innego typu. Przykładowo metryka BLEU (ang. Bilingual Evaluation Understudy) wskazuje procentowe podobieństwo dwóch ciągów wyrazów. W modelu zastosowanym w aplikacji nie można było wprowadzić metryk tego typu, ponieważ na wyjściu model przewiduje jedno słowo, więc podczas każdej iteracji porównywana by była sekwencja na przykład 10 wyrazów z jednym słowem.



Rys. 3.4. Funkcji strat i dokładności z nieodpowiednią metryką

Wygenerowane teksty, dla modeli o tej samej wielkości, ale z innym rozmiarem sekwencji wejściowej, przedstawiają: tabela 3.6 (sekwencja 20 wyrazów), tabela 3.7 (sekwencja 30 wyrazów), tabela 3.8 (sekwencja 40 wyrazów) oraz tabela 3.9 (sekwencja 50 wyrazów). Ciąg wyrazów, od którego model zaczął generować tekst został zaznaczony na zielono.

Tab. 3.6. Wygenerowany tekst dla sekwencji wejściowej 20 wyrazów

point of the rocks which occasioned this disaster stretching out , as is described before , to the southward , and casting off the current more southerly, had, was her to consider that were needful to cut them in my fright.

by the signs of signs and captain that received as i well die as much as the captain had sent one of the ship, and called eagerly away for help to the shore, and so he brought a great some of them, which i wanted nothing but was to get her a little just such to make it could hold it into also the two voyages of a large of barley, which i mentioned before and i had so much as on my thoughts.

i had had been secured thus to this began as to be secure days broad, inhabited corn by thankfulness, and what is in the way and what holding in my mind, and had more wet, a left and a little voyage so i gave my crow in the turtles, and grew all and boards like as these hinted said i left to make it again again with me that i had a cave right which, it would take me in its own and that i might expect a full account of the ship but i was not willing to banish all in my.

Tab. 3.7. Wygenerowany tekst dla sekwencji wejściowej 30 wyrazów

. i observed that the three other men had liberty to go also where they pleased but they sat down all three upon the ground , very pensive , and looked like howlings and break directly up to the place so close to the creek very taking by the horror of the negroes but i could not stir, by any more of the savages, who was in the middle of the longboat of his justice of god had delivered me that many things.

the man come, i was the most dexterous fellow at managing, at a word, and then catch them.

upon this affair by my last means to this piece of my story, and three fowlingpieces of the fowlingpiece, with friday, tying one out with the stock for a row of long two men might ironwork enough, and gave their orders and that men hanged them.

this appeared not.

but with these thoughts i had been comforted with a mark, that i was about three leagues from the place, and i believe no more nor friends, yet so i never could really worked

undertaken my boat but at that i made a great hollow, and hardly stood on the shore she was so easy to see that i could see if i could be all divided to so many things as i should have been willing to sell he bought of me, and asked his revenge, and tell me with his enemies several ways, and particularly the day of the our old man, who was very hard an white man here, he was tied with flags, and show head, if you will not stroke any power to do.

Tab. 3.8. Wygenerowany tekst dla sekwencji wejściowej 40 wyrazów

of life very true . and what is the prime of life may it not be defined as a period of about twenty years in a womans life , and thirty in a mans which years do you mean to include the end of gyges there were some wise for men who are many who have their work in life and act just as they are, and therefore they may not have to consider whether they ever is or any other state of any one who loves him and misery, who are not only the love of knowledge, whether inherent beauty, or, or desire may be allowed also true that he will have no difference to distinguish them.

then, as far as we can him at pleasure is the knowledge of good thing.

and this, i said, that we cannot say what a good deal.

then the cowardly and nature gives more about the essence of truth and less, or of essence over the art of beauties.

yes.

but in what way he mean in the former case of his proposals he is introduced even because he makes a true philosopher, he said.

he must be a soldier should have a power.

then i know that wellbred dogs are perfectly gentle to their familiars and acquaintances, and the reverse to strangers.

yes, i know.

Tab. 3.9. Wygenerowany tekst dla sekwencji wejściowej 50 wyrazów

pray the muses to tell us how discord first arose shall we imagine them in solemn mockery , to play and jest with us as if we were children , and to address us in a lofty tragic vein , making believe to be in earnest how would they address us after this the judge of private nature is supposed to be real companions and freemen in a great class of tyrants and these things which he would not be given to them the actual, they will not.

and did you not say that they attempt to make any sense of them in what point of view can be terrible if he answered them so, if they will go and give the one who came from his soul to say, to utter.

shall be his right good yes.

then you would not have to have right to contemplate us both if i knew what he was, and that the man draws the best of itself of the relation is, but that the truth is to be just and then if the just is unjust and answered to be just it would have been perfectly ordered, is a mere aspiration and right honours.

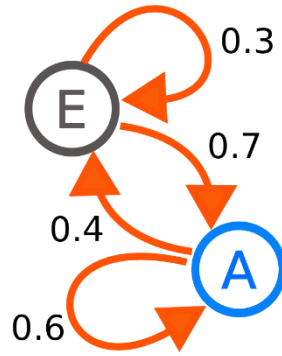
these are serious giving up the name of folly and afterwards the unjust, if he is so ill, if you take eyes with the same relation to which is or very good.

certainly not.

Przy zwiększaniu liczby wyrazów w sekwencji generowane zdania mają większy sens.

3.3. Porównanie z Łańcuchem Markowa

Łańcuch Markowa to model stochastyczny opisujący sekwencję możliwych zdarzeń. Prawdopodobieństwo wystąpienia każdego ze zdarzeń zależy tylko od poprzednich zdarzeń [29]. Model składa się z określonej liczby stanów docelowych i połączeń, czyli prawdopodobieństw wystąpienia następnego elementu między nimi (Rys. 3.5). Przy przewidywaniu kolejnego zdarzenia brane jest pod uwagę tylko najbardziej prawdopodobne zdarzenie, które jest wybierane. Istnieje możliwość wyboru kolejnych zdarzeń na podstawie nie tylko ostatniego zdarzenia, ale także wielu poprzednich (znacznie wtedy rośnie złożoność obliczeniowa).



Rys. 3.5. Diagram przejść w łańcuchu Markowa na przykładzie dwóch stanów [29]

W przypadku przewidywania wyrazów, liczbą stanów przejściowych jest liczba unikalnych słów (około 6 tysięcy). Połączenia między słowami są tworzone w taki sposób, że dla słowa/kilku słów (jeden stan) jest wyszukiwane słowo, które najczęściej po nich występuje w tekście, i zostaje ono przyporządkowane do tego stanu.

Łańcuch Markowa wykorzystano, aby porównać przewidywanie tekstu używając sieci neuronowej do podejścia statystycznego. Do eksperymentu wykorzystano tylko sekwencje słów wejściowych składających się z 1 i 2 wyrazów, ponieważ większa ilość obciąża pamięć w znaczący sposób. Tablica przejść, czyli prawdopodobieństwa pomiędzy połączeniami, składa się z połączeń typu każdy z każdym. Mając 6 tysięcy zdarzeń i jedno poprzednie wydarzenie brane pod uwagę, taka tablica ma wielkość 6000x6000, czyli 36 milionów komórek w pamięci przechowujące informacje o prawdopodobieństwie (liczby od 0 do 1). Biorąc pod uwagę dwa poprzednie wydarzenia, liczba komórek w tablicy przejść rośnie do 216 miliardów. Wynika z tego, że łańcuch Markowa zupełnie nie nadaje się do rozważanego przypadku, gdzie liczba elementów wejściowych mieściła się w zakresie od 10 do 50. Tabela 3.10 prezentuje działanie modelu Markowa dla jednego zdarzenia, a tabela 3.11 pokazuje działanie modelu sieci neuronowej dla jednego zdarzenia. Generowany tekst bardzo szybko się zapętlą w obu przypadkach. Biorąc pod uwagę dwa poprzednie zdarzenia wygenerowany tekst jest znacznie lepszej jakości (Tab. 3.12, Tab. 3.13).

Tab. 3.10. Wygenerowany tekst używając łańcuchu Markowa dla 1 wyrazu

years i dont know what he had been able to the door and the door and the door and the door and the door and the door and the door and the door and the door and

Tab. 3.11. Wygenerowany tekst używając sieci neuronowej dla 1 wyrazu

to these things, and the ship, and the ship, and the ship, and the ship, and the ship, and the ship, and the ship, and the ship, and the ship, and the ship, and the ship, and the

Tab. 3.12. Wygenerowany tekst używając łańcuchu Markowa dla 2 wyrazów

years that he had been very unsatisfactory of late i grant you that its not the slightest doubt that the change in gregors voice probably could not be able to get up my train leaves at five and he was not a pleasant experience and gregor would have been more

Tab. 3.13. Wygenerowany tekst używając sieci neuronowej dla 2 wyrazów

and i thrust her off at sea, the captain, who was very good one and both loose, they were, by the help of their own hands, to see if he had been in the boat, and the other, o master o master o master o master o master o master o master o master o master o

Za pomocą sieci neuronowej można wprowadzać długie sekwencje wejściowe, natomiast nie można tego wykonać przy użyciu łańcucha Markowa.

4. Podsumowanie

Sztuczne sieci neuronowe to dziedzina nauki, która jest bardzo intensywnie rozwijana. Sieci neuronowe mają ogromne możliwości i systemy opierające się na nich stoją u podstaw współczesnej sztucznej inteligencji.

Celem pracy było wykorzystanie nowoczesnych sieci neuronowych w modelowaniu języka naturalnego. Praca ta powstawała przez ponad pół roku. Większość czasu poświęcona tej tematyce zaowocowała poznaniem ogromu możliwości uczenia maszynowego, a w szczególności sztucznych sieci neuronowych. Najwięcej czasu zostało poświęcone aplikacji do przewidywania znaków. Dużą trudnością było osiągnięcie coraz to lepszych wyników na zbiorze walidacyjnym bez przeuczenia modelu. Wyniki są zadawalające i mogą stanowić podstawę do dalszego rozwoju. Aplikacja do przewidywania wyrazów uzyskuje lepsze rezultaty. Pomimo mniejszej ilości czasu spędzonego nad rozwojem aplikacji niż w przypadku aplikacji przewidującej znaki, efekty również są zadawalające. Aplikacja do przewidywania wyrazów działa bardzo dobrze, jednak jest kilka pomysłów, które mogłyby polepszyć jakość modelu, a wraz z tym jakość generowanego tekstu:

- Zastosowanie większej ilości danych, również pomieszanych z wielu źródeł i różnych autorów,
- Uczenie sieci neuronowej trwa dosyć długo, dlatego dobranie odpowiednich parametrów sieci nie jest prostym zadaniem. Istnieją metody automatycznego wyboru parametrów, które pomogłyby w odpowiednim wyborze m.in: funkcji aktywacji, współczynnika uczenia, funkcji kosztu czy nawet struktury sieci,
- Szkolenie modelu przez większą liczbę epok,
- Istnieją podobne rozwiązania do LSTM, które można by było wypróbować, takie jak na przykład zaprezentowany w 2014 roku GRU (ang. Gated Recurrent Unit), charakteryzujący się mniejszą ilością parametrów sieci,
- Zastosowanie bardziej adekwatnej metryki do sprawdzania procesu uczenia modelu, na przykład BLEU (ang. Bilingual Evaluation Understudy).

Bibliografia

- [1] https://www.sas.com/en_us/insights/analytics/neural-networks.html
(dostęp 26.11.2019)
- [2] <https://www.globema.pl/ai-sztuczne-sieci-neuronowe-zastosowania/>
(dostęp 26.11.2019)
- [3] Simon O. Haykin, Neural Networks and Learning Machines (3rd Edition), 2008
- [4] J.Żurada, M.Barski, W. Jędruch, Sztuczne sieci neuronowe, PWN, 1996
- [5] <https://www.digitalvidya.com/blog/types-of-neural-networks/> (dostęp 30.11.2019)
- [6] <https://mindmajix.com/artificial-neural-network-and-how-it-works>
(dostęp 30.11.2019)
- [7] <https://towardsdatascience.com/machine-learning-recurrent-neural-networks-and-long-short-term-memory-lstm-python-keras-example-86001ceaaebc>
(dostęp 1.12.2019)
- [8] <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a>
(dostęp 1.12.2019)
- [9] <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (dostęp 1.12.2019)
- [10] <https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e>
(dostęp 1.12.2019)
- [11] https://en.wikipedia.org/wiki/Long_short-term_memory (dostęp 1.12.2019)
- [12] <https://pl.wikipedia.org/wiki/Python> (dostęp 2.12.2019)
- [13] <https://pl.wikipedia.org/wiki/TensorFlow> (dostęp 2.12.2019)
- [14] <https://opensource.com/article/17/11/intro-tensorflow> (dostęp 2.12.2019)
- [15] <https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346> (dostęp 2.12.2019)
- [16] <https://en.wikipedia.org/wiki/Keras> (dostęp 2.12.2019)
- [17] <https://bulldogjob.pl/news/738-google-colab-pythonowy-obszar-roboczy-w-chmurze> (dostęp 2.12.2019)

- [18] <https://www.kdnuggets.com/2015/12/how-do-neural-networks-learn.html>
(dostęp 2.12.2019)
- [19] https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets
(dostęp 2.12.2019)
- [20] <https://intellipaat.com/community/2624/the-correctness-of-neural-networks>
(dostęp 2.12.2019)
- [21] <https://keras.io/getting-started/sequential-model-guide/> (dostęp 5.12.2019)
- [22] <https://keras.io/models/sequential/> (dostęp 5.12.2019)
- [23] <https://machinelearningmastery.com/statistical-language-modeling-and-neural-language-models/> (dostęp 5.12.2019)
- [24] <https://medium.com/@david.campion/text-generation-using-bidirectional-lstm-and-doc2vec-models-1-3-8979eb65cb3a> (dostęp 6.12.2019)
- [25] <https://darektidwell.com/character-level-text-generation-with-lstm-rnn-in-tensorflow-alice-in-wonder-land/> (dostęp 8.12.2019)
- [26] https://keras.io/examples/lstm_text_generation/ (dostęp 8.12.2019)
- [27] <https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/> (dostęp 8.12.2019)
- [28] <https://stackabuse.com/text-generation-with-python-and-tensorflow-keras/>
(dostęp 8.12.2019)
- [29] https://en.wikipedia.org/wiki/Markov_chain (dostęp 14.12.2019)
- [30] https://www.gutenberg.org/wiki/Gutenberg:The_Project_Gutenberg_License
(dostęp 31.12.2019)
- [31] <https://www.cs.toronto.edu/~hinton/science.pdf> (dostęp 3.01.2020)