Rozpoznawanie języka

z wykorzystaniem sztucznych sieci neuronowych

Projekt ze Sztucznej Inteligencji

Kamil Czepiel | Paweł Domaradzki | Jakub Grzybowski

Wprowadzanie

Aby poznać język obcy, człowiek musi nauczyć się zasad panujących w tym języku. Gramatyka to zasady łączenia słów, a słowa to zasady łączenia przedrostków z tematem, sylab ze sobą czy w końcu składania pojedynczych liter. Można stwierdzić, że po dłuższej nauce człowiek jest już w stanie jednoznacznie stwierdzić czy zdanie, które czyta jest w języku portugalskim czy hiszpańskim.

Podobnie wygląda proces nauki języka przez komputer. **Uczenie Maszynowe** to podzbiór Sztucznej Inteligencji, który polega na odkrywaniu zasad, którymi cechują się duże zbiory danych – program uczy się wzorców oraz korelacji między nimi. Tak samo jak człowiek, komputer może wyuczyć się i stwierdzić z pewnym prawdopodobieństwem, czy wpisywany ciąg znaków jest zdaniem w języku polskim czy angielskim. Potrzebuje do tego odpowiednio przygotowanych danych, zasobów sprzętowych i oczywiście czasu, aby wykonane operacje doskonalenia i poprawiania wyników były okraszone jak najmniejszym błędem.

$$Dane + Reguły = Wyniki$$

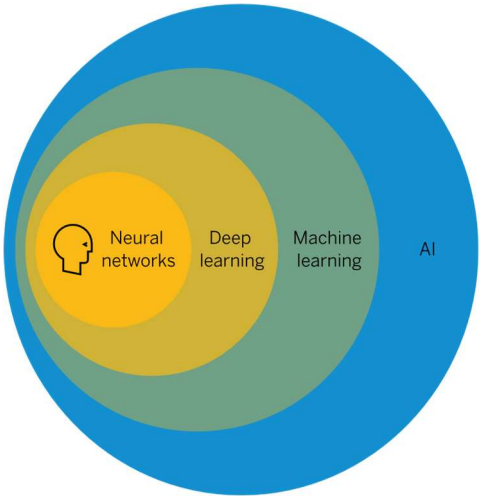
proces programowania

$$Dane + Wyniki = Reguły$$

proces uczenia maszynowego

Klasyfikacja uczenia maszynowego

Przytoczone uczenie maszynowe jest pojęciem bardzo ogólnym, które można podzielić na kolejne grupy. W naszym projekcie skupiliśmy się na użyciu modelu **Sztucznych Sieci Neuronowych**, które są z kolei częścią **Nauczania Głębokiego**. Wykorzystanie tego sposobu umożliwia nam efektywne przeprowadzenie procesu nauki.



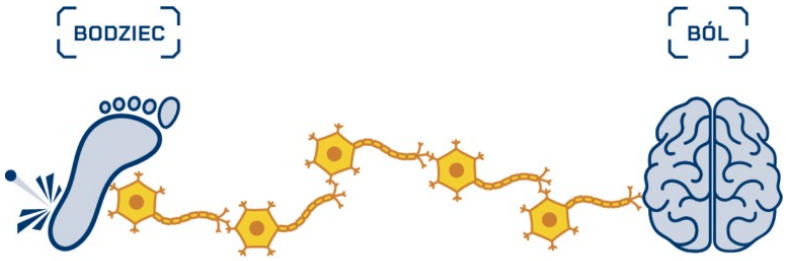
1. Sztuczne Sieci Neuronowe

1.1. Budowa neuronu

Naukowcy, zajmujący się neurobiologią, na podstawie opisu neuronów człowieka opracowali Sztuczne Sieci Neuronowe czyli model uczenia głębokiego. Działanie pojedynczego neuronu w tym systemie przypomina działanie tego w ludzkim organizmie, dlatego warto krótko przypomnieć sposób działania:

Neuron, czyli komórka nerwowa, to najmniejszy element układu nerwowego. Rejestruje bodziec, na przykład ukłucie szpilką, i przekazuje go następnemu neuronowi. Przekazywanie bodźca przez kolejne neurony trwa do chwili, gdy ostatni z nich prześle bodziec do mózgu. Wtedy czujemy ukłucie.

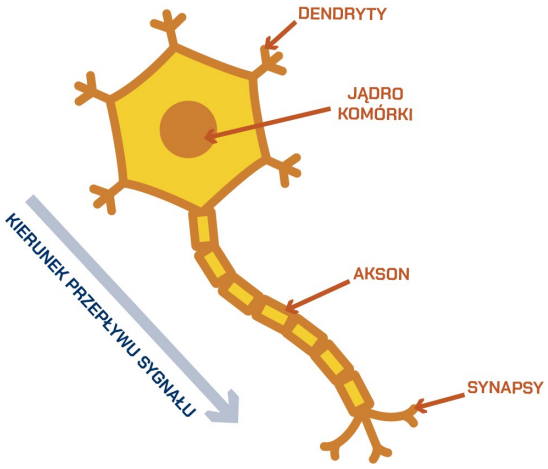
www.sztucznainteligencja.org.pl



PRZEKAZYWANIE I PRZETWARZANIE INFORMACJI O BODŹCU

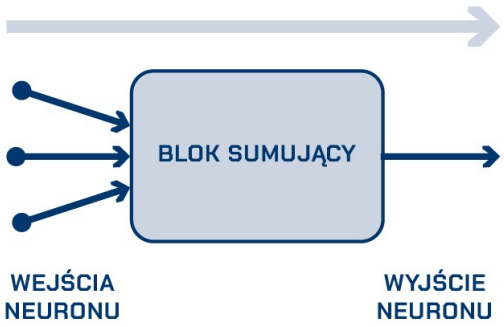
Neuron składa się z:

- **dendrytów**, które otrzymują od receptorów sygnały ze środowiska;
- **jądra komórki**, które przechowuje i powiela informację genetyczną;
- **aksonu**, który jest po prostu przewodem i przekazuje sygnał wejściowy dalej;
- **synaps**, które są połączeniem neuronu z innym neuronem, gruczołem albo mięśniem. Przekazują sygnał wyjściowy, np. uczucie bólu albo przypływ adrenaliny.



Budowa neuronu człowieka

Budowa sztucznego neuronu nieco różni się od tego ludzkiego, jest dostosowana do zastosowań uczenia głębokiego i jego implementacji. Na schemacie strzałka pokazuje kierunek, w którym płyną informacje. Można zauważyć podobieństwa sztucznego modelu z ludzkim neuronem.



Budowa sztucznego neuronu

W jego skład wchodzą:

- **Wejścia neuronu**, które imitują dentryty, czyli przesyłają informację, w tym przypadku będzie to jakaś wartość liczbowa odpowiadająca atrybutowi obiektu świata rzeczywistego.
- **Blok sumujący**, czyli miejsce, które sumuje informacje wyjść z wagami.
- **Wyjście neuronu** przesyła przeliczony sygnał wyjściowy, który może być wykorzystany do kolejnych operacji.

1.2. Zasada działania neuronów

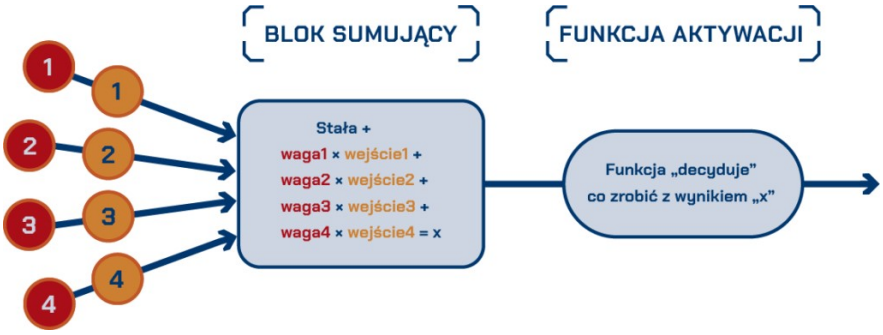
Chcąc zrozumieć dokładne działanie sieci trzeba wyjaśnić pojęcie **wag** oraz **funkcji aktywacyjnych**.

Każda wejście sieci neuronowej posiada swoją **wagę**, czyli współczynnik określający jak bardzo istotna jest informacja na wejściu. Zmiana wagi wpływa również na obliczenia w bloku sumacyjnym, co kompletnie może zmodyfikować wynik do dalszej analizy. Odpowiednie dobranie wag jest kluczowe do skutecznego wykonania procesu nauczania i znacząco wpływa na rezultaty.

Po otrzymaniu wyniku z bloku sumacyjnego następuje interpretowanie otrzymanych danych. **Funkcja aktywacyjna** otrzymuje wynik jako argument i w zależności od dobranej funkcji zwracany jest rezultat, który ponownie można interpretować. Wyróżnia się kilka podstawowych funkcji aktywacyjnych, do których przejdziemy w kolejnej części tego opracowania.

Po powyższych objaśnieniach można wyciągnąć kilka wniosków:

- W sieciach neuronowych określamy stopień przydatności danych na wejściu. Waga bezpośrednio wpływa na analizę danych, może ona również jednoznacznie stwierdzić, że dana informacja nie jest potrzebna w procesie uczenia (waga równa zero)
- Blok sumacyjny przechowuje dane z wejść i oblicza wynik z iloczynem ich wag. Można do tego bloku dołączyć **stałą** (**bias**), która jest ustalana przy tworzeniu sieci
- Funkcja aktywacyjna aby została pobudzona musi otrzymać satysfakcjonujący argument na wejściu. Nazywa się to **progiem pobudzenia** (podobnie jak w neuronie człowieka niektóre informacje są pomijane lub bardzo mało znaczące)
- Czynności obliczania sumy i pobudzania neuronu można powtarzać w zależności od potrzeb.

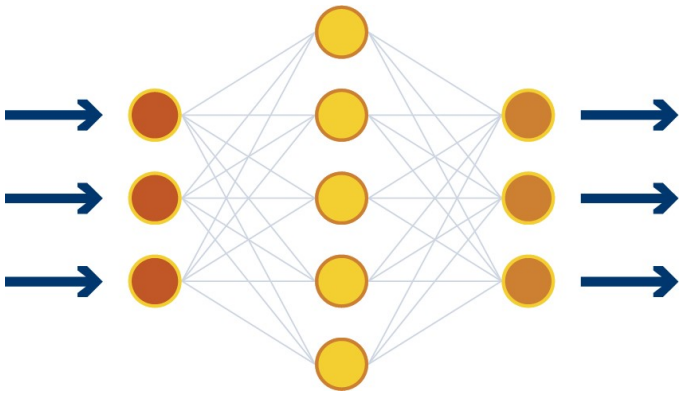


Praktyczny opis budowy
sztucznego neuronu

1.3. Opis sieci

Sieć neuronowa to wiele połączonych ze sobą sztucznych neuronów. Wyjście pierwszego neuronu przekazuje sygnał do wejścia następnego, ten przetwarza sygnał i wysyła go następnemu.

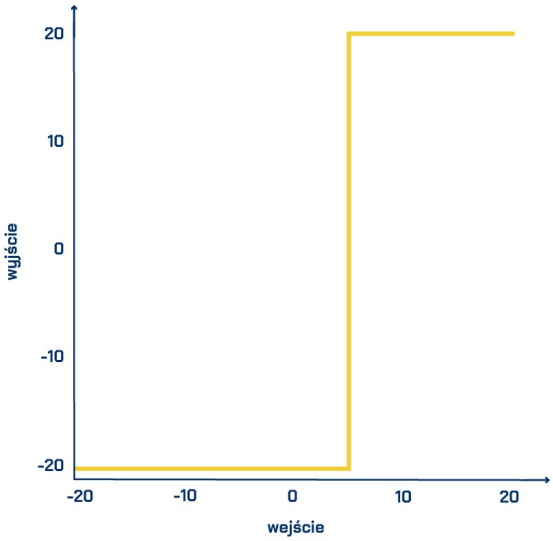
Neurony w sieciach neuronowych ułożone są w warstwy. Warstw może być od kilku do kilkunastu. **Warstwa zewnętrzna (wejściowa)** to ta, która rejestruje bodźce z zewnątrz. Zebrane informacje przetwarza w bloku sumującym i bloku funkcji aktywacji, a potem przekazuje (lub nie) informację następnej warstwie neuronów. Te następne warstwy nazywa się **warstwami ukrytymi**. Przeliczają one informacje uzyskane od poprzedniej warstwy i przekazują następnym warstwom ukrytym lub warstwie wyjściowej. **Warstwa wyjściowa** dokonuje ostatnich obliczeń tego, co otrzymały i przeliczyły wszystkie poprzednie warstwy, a następnie wysyła wynik w formie decyzji. Może być to rozpoznanie jakiegoś obiektu lub jego klasyfikacja do pewnej grupy, na przykład **zdanie do języka**.



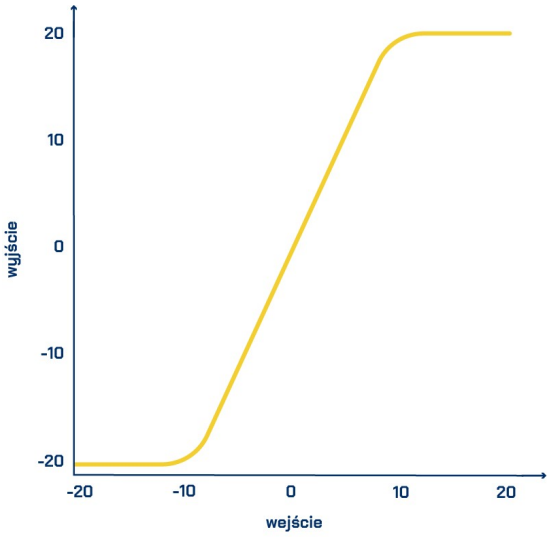
Budowa sieci neuronowych

1.4. Funkcje aktywacyjne

W sieciach neuronowych wyróżnia się kilka funkcji aktywacyjnych, których wybór jest zależny od badanego problemu. Najbardziej popularne z nich to funkcja liniowa, funkcja progowa czy funkcja sigmoidalna. Dwie ostatnie wyróżniają się przypisaniem wartości dyskretnych, to znaczy że funkcja zakwalifikuje podany argument do jednej bądź drugiej wartości. Przy funkcji sigmoid ten skok jest zauważalny i rezultaty nie będą „binarne", ponieważ narastanie funkcji następuje w jakimś przedziale, a nie jednym punkcie. Na potrzeby projektu korzystamy z funkcji aktywacyjnych **ReLu** oraz **SoftMax**, które zostaną opisane w dalszej części opracowania.



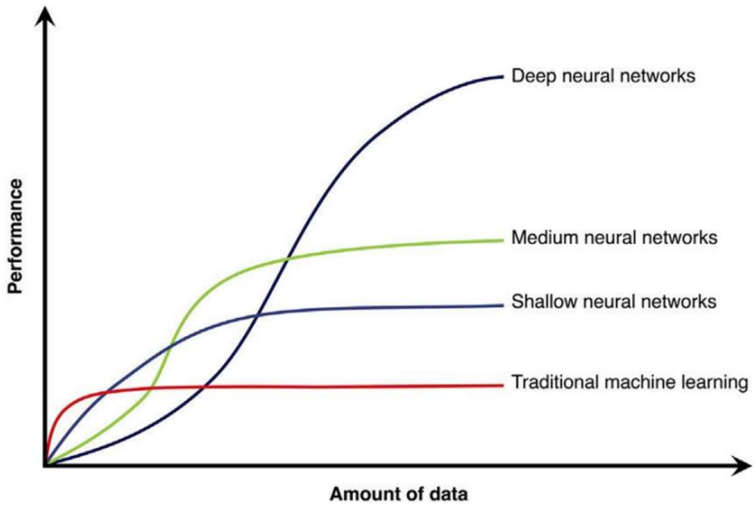
Funkcja skokowa



Funkcja sigmoid

1.5. Zalety głębokiego uczenia i sztucznych sieci neuronowych

Model sztucznych sieci cechuje się wysoką wydajnością przy podaniu dużego zestawu danych testowych na wejściu. Nasz projekt zakładał wprowadzenie długiego łańcucha znaków, aby jak najdokładniej nauczyć model konkretnego języka. Sieci mają możliwość wyszukiwania ukrytych wzorców, z czym zwykłe uczenie maszynowe sobie nie radzi.



Wykres wydajności metod uczenia maszynowego względem liczby danych wejściowych

Kolejną korzyścią płynącą z użycia sieci neuronowych jest duża dostępność narzędzi ułatwiających tworzenie modelu i uczenie. Dodatkowo szukając informacji na temat rozpoznawania języków, w większości źródeł korzystano z modelów głębokiego uczenia.

2. Implementacja rozpoznawania języka

2.1. Środowisko programistyczne i narzędzia

Postanowiliśmy skorzystać z języka **python**, który dostarcza najszerszą grupę narzędzi i bibliotek do zastosowań naukowych. Do stworzenia programu wykorzystaliśmy:

- Bibliotekę **sklearn**, która dostarcza funkcji do tworzenia modeli, trenowania modelu oraz wektoryzacji
- Bibliotekę **Tensorflow**, która również umożliwia wykonywania czynności związanych z uczeniem głębokim i sieciami neuronowymi
- **DataFrame** do uporządkowania danych i elastycznego korzystania z nich
- **pandas** do operacji i manipulacji na danych
- **keras** do przetwarzania sekwencyjnego
- **matplotlib** do tworzenia wykresów wydajnościowych
- **jupyter notebook** do przejrzystego tworzenia dokumentu z programem i przedstawiania wykonywanych czynności po kolei
- **tkinter** oraz **pillow** do utworzenia prostego GUI
- **GitHub** do utworzenia repozytorium i możliwości rozproszonej pracy między trzema osobami.

2.2. Dane wejściowe

Do projektu wykorzystaliśmy duży zestaw danych w pięciu językach obcych. Na początku uwzględniliśmy zasady, którymi trzeba się kierować przy modelowaniu wejść do systemu sieci neuronowych. Dane te muszą być:

- **zbalansowane**, czyli zachować proporcje między liczbą słów danych języków
- **wyczyszczone**, należy pominąć nieznaczące dane oraz kolumny, w których dane się powtarzają
- **sformatowane**, czyli ujednolicone – jeden wyraz ma jedno znaczenie, np. nie używać skrótów
- **znormalizowane** – częstość występowania słowa w danym języku musi być równomiernie wartościowana, bo sieć może błędnie intepretować dane i dobierać złe wagi. W przypadku naszego projektu normalizowaliśmy występowanie słowa w zdaniach do zbioru $\{0,1\}$. Czyli wystąpienie słowa więcej razy w zdaniu było traktowane jako wartość 1, a nie liczba jego wystąpień. Tak samo najmniejsza liczba wystąpień w zdaniu jest normalizowana do 0, a nie do konkretnej wartości. Metoda ta nazywa się **skalowanie min-max**. Poprawia to znacząco wydajność programu i zmniejsza ilość kompleksowych obliczeń, a dodatkowo niweluje ryzyko pominięcia danych.

W procesie uczenia, dane wejściowe dzielą się na dwie grupy: pierwsza z nich to **dane treningowe**, czyli dane wykorzystywane w procesie uczenia, a drugie to **dane testowe**, czyli dane wykorzystywane w testowaniu wyuczonych zależności. Ważne jest to, żeby były to zbiory rozłączne, ponieważ pokazuje to prawdziwą skuteczność modelu, a nie tylko wyuczenie ograniczonego zestawu „na pamięć” i problemy przy wprowadzeniu wcześniej nie poznanych danych.

W naszym projekcie zachowaliśmy proporcje $\mathbf{80:20}$ danych treningowych do danych testowych, a samych zdań na pięć języków przypadło $\mathbf{200000}$. Podział nastąpił dopiero po wyborze metody rozpoznawania języka, o których mowa w dalszej części opracowania.

2.3. Wykorzystane funkcje aktywacyjne

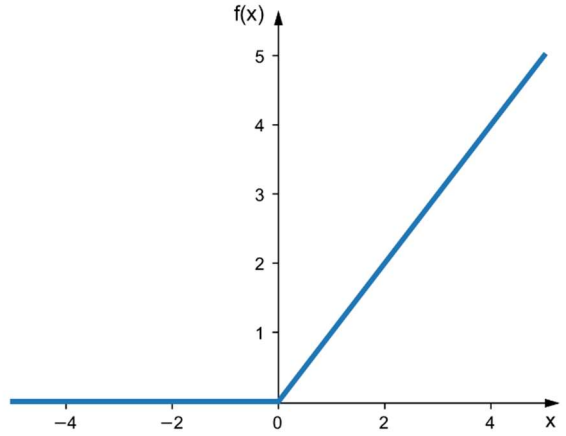
W naszym projekcie zastosowaliśmy dwie funkcje aktywacyjne: **ReLu** oraz **SoftMax.**

2.3.1

Funkcja **ReLu** czyli funkcja aktywacyjna **Rectified Linear Unit** zwraca wartość wejściową, jeżeli jest dodatnia, bądź zero w pozostałych przypadkach.

Wzór: $f(x) = x^+ = \max(0, x)$

Wykres:

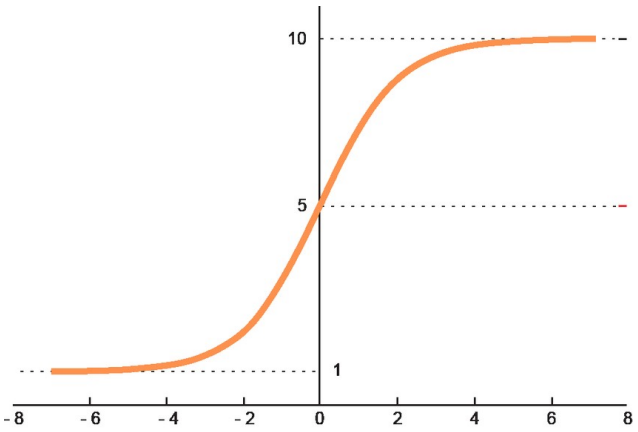


2.3.2

Funkcja **Softmax** zwraca wartości (0,1), zaletą jest bazowanie na wielu danych (wektor) oraz zwrot prawdopodobieństwa każdej z wartości, używana w sieci jako ostatnia funkcja.

Wzór: $\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$

Wykres:



2.4. Metody rozpoznawania języka

Do przygotowaniu zestawu danych potrzebowaliśmy wiedzieć, jak będziemy je prezentować w programie. Postawiliśmy na dwa rozwiązania: metodę trigramów oraz metodą całych słów.

2.4.1. Metoda trigramów

Do rozpoznania z jakiego języka pochodzi słowo wykorzystujemy trigramy, czyli złączone ze sobą trzy litery. Początkowym etapem jest pobranie wszystkich takich trójek korzystając z naszej bazy, a następnie wybranie (200 najbardziej popularnych). Dodatkowo dodajemy typowe trigramy dla danego języka. Z całości tworzymy listę unikalnych trigramów. W rezultacie tworzymy macierz zawierającą informację o liczbie trigramów w każdym zdaniu znajdującym się w bazie.

indeks/trigram	the	and	for	...
0	0	1	2	...
1	0	0	1	...
2	0	0	0	...
3	1	1	1	...

Tabela wystąpień trigramów

```
trigrams = []
for i, c in enumerate(sentence):
    trigram = sentence[i:(i + 3)]
    if len(trigram) == 3:
        trigrams.append(trigram)
```

Pobranie trigramów z bazy zdań

```
all_trigrams: set[str] = set()
lang_trigrams: dict[str, list[tuple[str, int]]] = dict()
for lang in languages:
    trigrams = Counter()
    series = df[df["lang"] == lang]["sentence"]
    for sentence in series:
        tri = get_trigrams(sentence)
        trigrams.update(tri)
    trigrams += Counter()
    mc = trigrams.most_common(trigrams_by_lang)
    all_trigrams.update([v[0] for v in mc])
    lang_trigrams[lang] = mc
```

Wybranie najpopularniejszych trigramów dla poszczególnych języków

```
vectorizer = CountVectorizer(vocabulary=dic,
                             ngram_range=(3,3),
                             analyzer="char")
```

Stworzenie macierzy poszczególnych trigramów

```
X = vectorizer.fit_transform(sentences)
# Tworzymy macierz wystąpień poszczególnych trigramów
features = pd.DataFrame(data=X.toarray(), columns=all_trigrams)
```

Przypisanie cech

2.4.2. Metoda całych słów

Metoda ta różni się jedynie wykorzystaniem słów jako dane wejściowe. Dalej postępowanie jest takie samo.

```
all_words: set[str] = set()
for lang in ["eng", "deu", "spa", "ita"]:
    words = Counter()
    series = df[df["lang"] == lang]["sentence"]
    for sentence in series:
        words.update([w for w in sentence.split(" ") if len(w) > 1])

    mc = words.most_common(trigrams_by_lang)
    all_words.update([v[0] for v in mc])
return all_words
```

Pobieranie całych słów

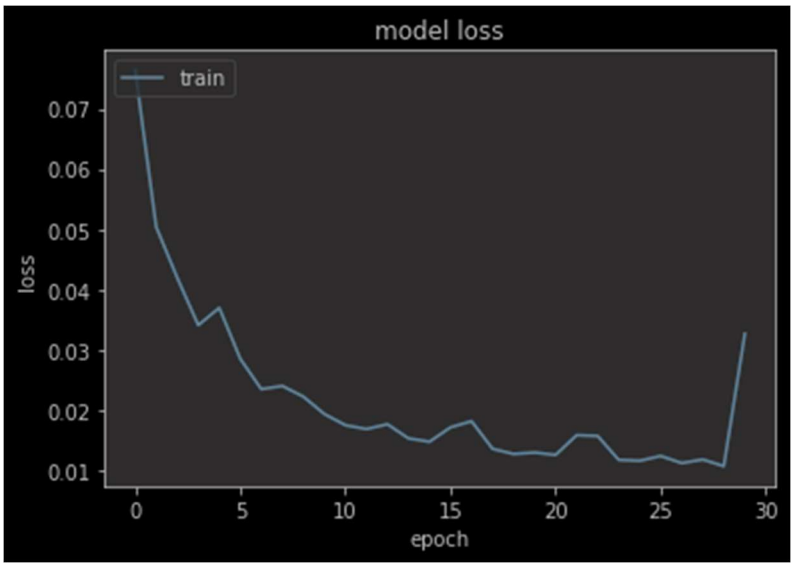
2.5. Terminologia procesu nauczania

Uczenie modelu charakteryzuje się konkretnymi elementami, które poniżej zdefiniowaliśmy:

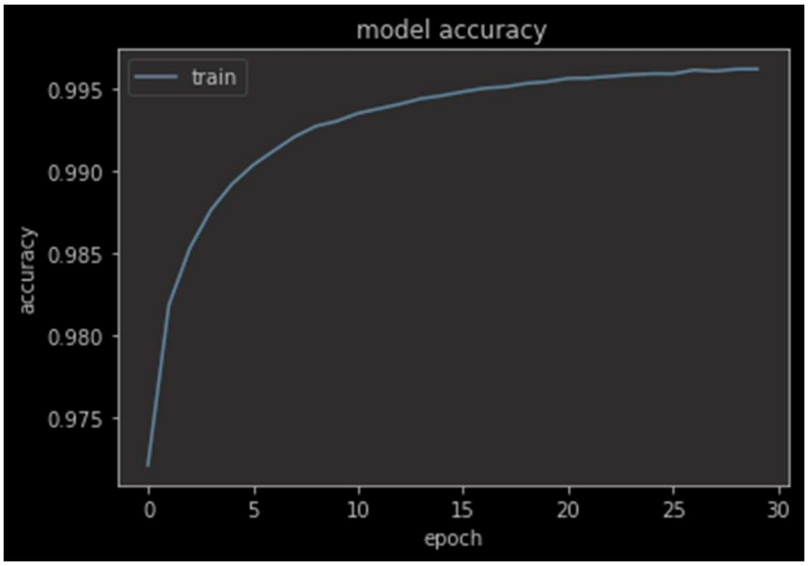
- **Sample** – jeden wiersz danych wejściowych (jeden przykład)
- **Batch** – liczba próbek (w tym przypadku zdań), które należy przejść w sieci, aby wagi zostały zaktualizowane
- **Epoch** – liczba przejść danych przez sieć
- **Batch Size** – wielkość zestawu treningowego
- **Learning rate** – jak szybko funkcja kosztu schodzi do minimum

3. Wyniki i wnioski

3.1. Wyniki metodą trigramów



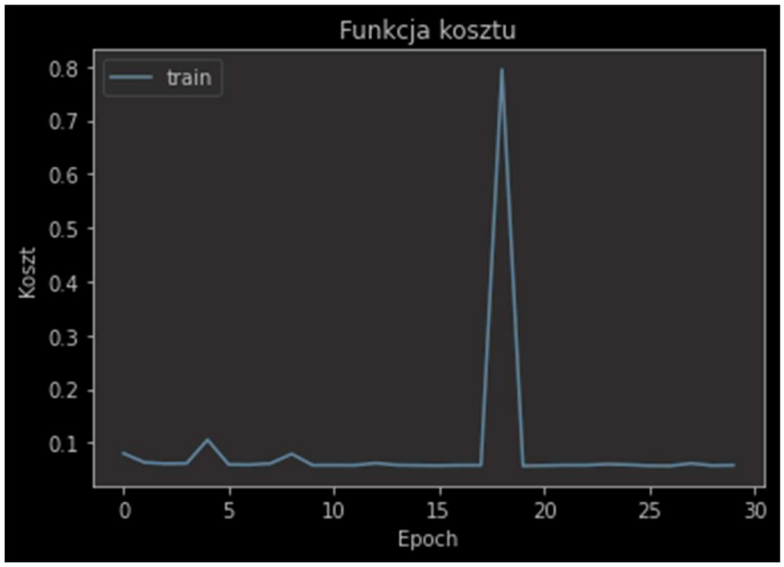
Utrata danych w czasie



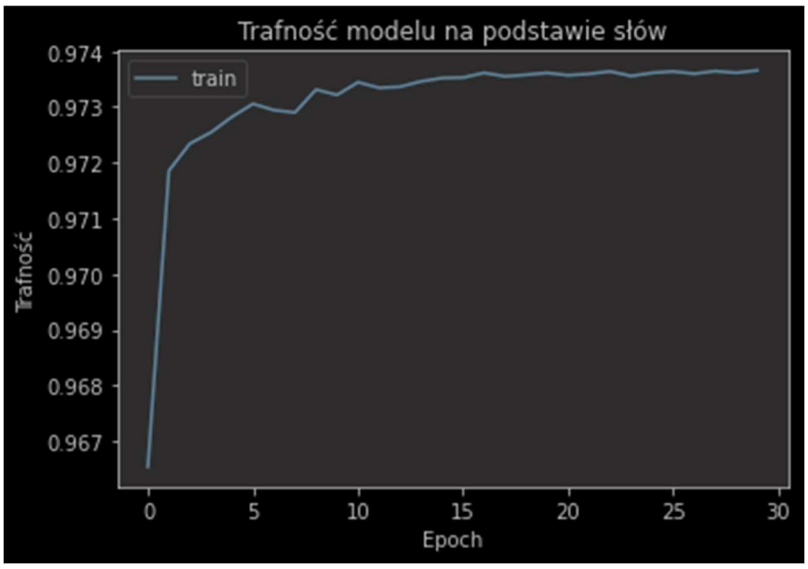
Dokładność modelu w czasie

Wnioski: z wykresów można odczytać, że model całkiem szybko się uczy, bo do uzyskania optymalnej dokładności potrzebuje około 5 epoch, po 15 zbliża się do absolutnej pewności. Natomiast widać efekt Rungego dla wykonywanych epoch – utrata danych występuje na krańcach przedziału.

3.2. Wyniki metodą całych słów



Funkcja kosztu



Dokładność modelu w czasie

Wnioski: Można zauważyć delikatną różnicę w czasie otrzymania optymalnej dokładności – metoda całych słów dochodzi do wyniku nieco dłużej, a także nie osiąga tak wysokiej wartości jak metoda trigramów. Za to funkcja kosztu przy epoch=18 nie jest optymalna. Funkcja kosztu to różnica wyników uzyskanych od oczekiwanych i trzeba zachować jak najmniejsze wartości aby uzyskać optymalne wyniki.