Generacja tekstów piosenek

Maciej Krzyżanowski, Sebastian Kutny, Tomasz Lewandowski

Kwiecień 2023

Spis treści

1 Wstęp

Celem projektu było stworzenie modelu generującego tekst piosenki na podstawie wybranych danych jako tekstów innych utworów. Wykorzystaliśmy 2 metody: łańcuchy Markova oraz rekurencyjne sieci neuronowe.

Projekt zawiera narzędzie ścraper"do pozyskiwania danych ze stron:

- `https://www.tekstowo.pl`
- `https://www.azlyrics.com`

Implementacja została wykonana w języku Python oraz wykorzystuje biblioteki:

- pandas
- BeautifulSoup
- nltk
- request
- queue
- re

Dostępna jest opcja łączenia zbiorów danych do jednego pliku w celu wykorzystania ich jednocześnie.

Przed rozpoczęciem przetwarzania danych są one oczyszczane poprzez ujednolcenie wielkości liter, usunięcie niepotrzebnych znaków interpunkcyjnych, słów zakazanych (np:. określających składowe tekstu utworu) oraz wyrażeń ze szczególnymi znakami interpunkcyjnymi jako określających zawartość tekstu.

Tekst jest generowany jako dowolna liczba wersów o dowolnej ilości słow.

2 Łańcuchy Markova

2.1 Wstęp

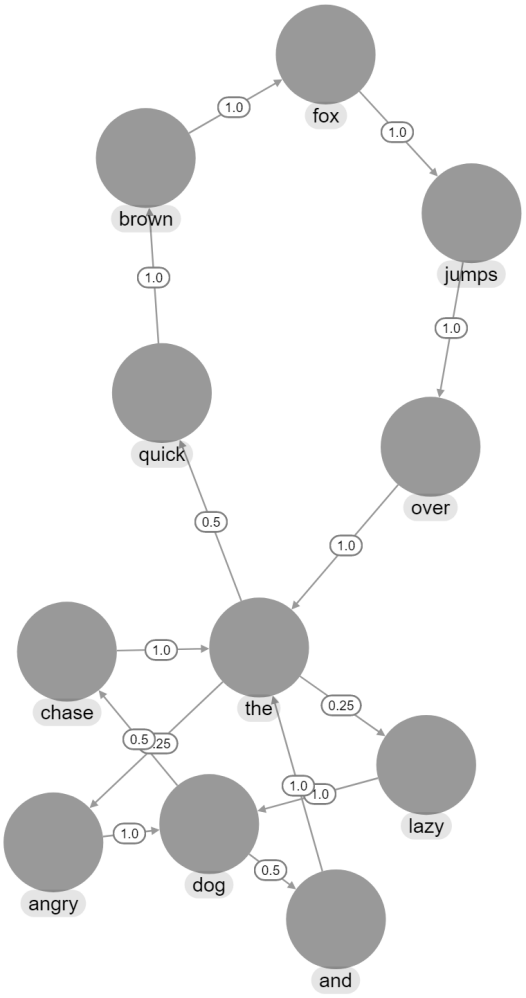
Łańcuchy Markova to matematyczny model służący do generowania tekstu lub sekwencji innych elementów, takich jak dźwięki lub obrazy. Ideą modelu jest analiza sekwencji istniejących elementów i wykorzystanie tych informacji do przewidywania, jakie elementy powinny pojawić się następnie.

W przypadku generowania tekstu, łańcuchy Markova są zwykle stosowane do analizy sekwencji słów w tekście źródłowym i generowania nowych sekwencji słów na podstawie tych informacji. Proces zaczyna się od wyboru deterministycznego lub losowego początkowego stanu łańcucha, a następnie generowania kolejnych stanów na podstawie informacji o pradwopodobieństwie wystąpienia po sobie stanów w analizowanym tekście w kontekście poprzednio wygenerowanych stanów. Przykładowo, jeśli w tekście źródłowym po słowie "generatorćzęsto pojawia się słowo "piosenek", to model łańcucha Markova przypisze wysokie prawdopodobieństwo wystąpienia słowa "piosenek"po słowie "generator".

Istnieją różne sposoby implementacji modelu łańcuchów Markova, ale zwykle opierają się one na analizie pewnej liczby poprzednich elementów, zwanej śtopniem"modelu. Na przykład, w przypadku modelu pierwszego stopnia, prawdopodobieństwo wystąpienia danego elementu zależy tylko od poprzedniego elementu, w modelu drugiego stopnia, prawdopodobieństwo zależy od dwóch poprzednich elementów, a w modelu trzeciego stopnia, prawdopodobieństwo zależy od trzech poprzednich elementów itd. Stopień łańcucha nazywamy N-gramem.

W naszym projektcie N-gram jest parametryzowany i bazuje na N uprzednio wygenerowanych słowach w wersie, losując następne słowo na podstawie prawdopodobieństwa jego wystąpienia po N sekwencji słów uprzednio wygenerowanych. Dodatkowo przy każdym wersie o nieparzystym numerze podejmowana jest próba stworzenia rymującego się wersu na podstawie ostatniej sylaby poprzedniego. Najpierw znajdowane są wszystkie rymujące się zakończenia wersu niebędące ostatnim słowem poprzedniego, a następnie - jeśli takie istnieją - losujemy jedno z nich zamiast z wszyskich pozycji. Przy nieznalezieniu rymujących się słów generacja odbywa się tak jak w zwykłym wypadku. W praktyce szansa na stworzenie rymu jest mała i powinna rosnąć z ilością danych przetwarzanych przez model.

Model łańcuchów Markova nie jest idealny i może generować sekwencje, które nie są sensowne lub poprawne gramatycznie. Dopiero przy wglądzie modeli w setki stanów wstecz oraz przy bardzo dużej ilości danych można wygenerować tekst podobny do pisanego przez człowieka.

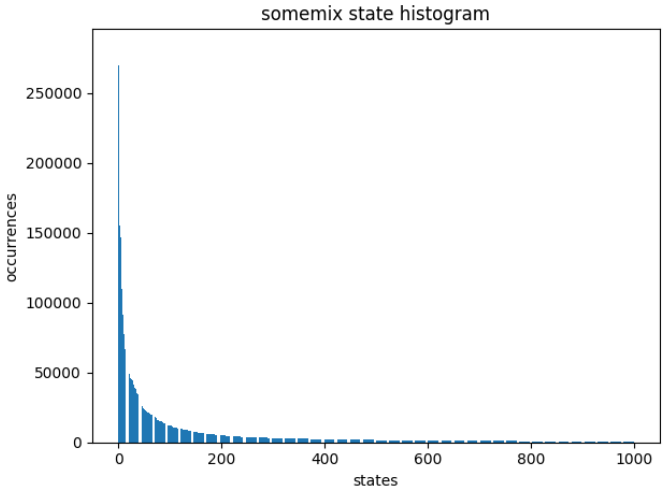


Rysunek 1: Przykład łańcucha Markova, dla zdania "The quick brown fox jumps over the lazy dog and the angry dog chase the quick brown fox.", dla wartości $ngram = 1$, oznaczającej stany jako pojedyncze słowama oraz wartościami prawdopodobieństw przejść pomiędzy stanami obliczonych na podstawie zdania wejściowego.

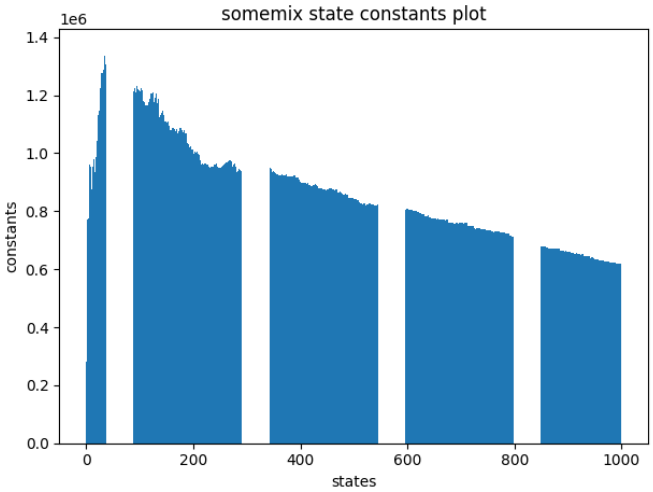
2.2 Prawo Zipfa

Prawo Zipfa to empiryczna obserwacja dotycząca częstotliwości występowania słów w korpusie tekstów. Mówi ono, że jeśli posortujemy słowa występujące w tekście według częstotliwości ich wystąpień i przyporządkujemy każdemu słowu rangę zgodną z jego pozycją w rankingu, to liczba wystąpień słowa o danej randze jest odwrotnie proporcjonalna do wartości tej rangi. W praktyce oznacza to, że najczęściej występujące słowo będzie występować dwa razy częściej niż drugie na liście, trzy razy częściej niż trzecie, i tak dalej.

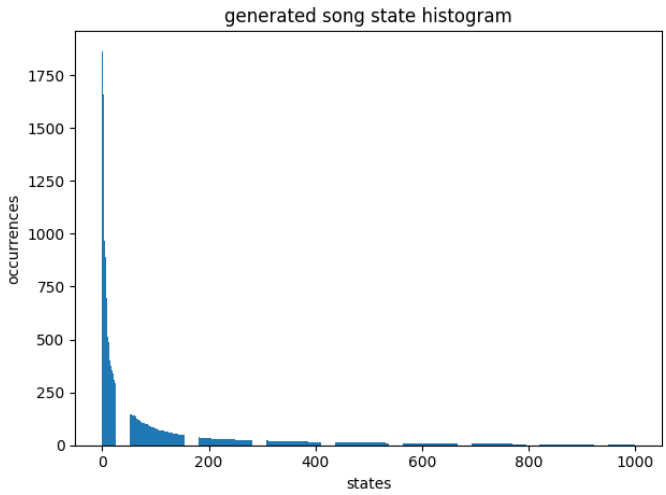
Wykorzystując Prawo Zipfa w generacji tekstów piosenek, można zapewnić, że wygenerowany tekst będzie przypominał rzeczywiste teksty pod względem częstotliwości występowania słów.



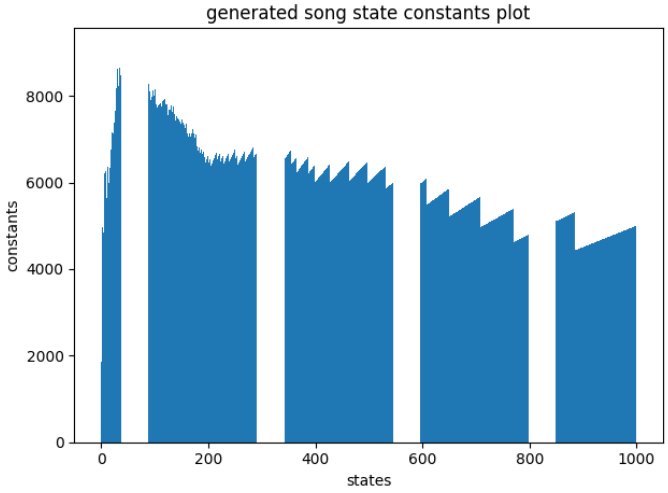
(a) Wykres 1000 najczęściej pojawiających się słów dla zbioru danych *somemix.csv*.



(b) Wykres stałej $constant = ranga * wystapienia$ dla zbioru danych *somemix.csv*



(a) Wykres 1000 najczęściej pojawiających się słów dla piosenki wygenerowanej na podstawie *somemix.csv* o 100 wersach po 500 słów.

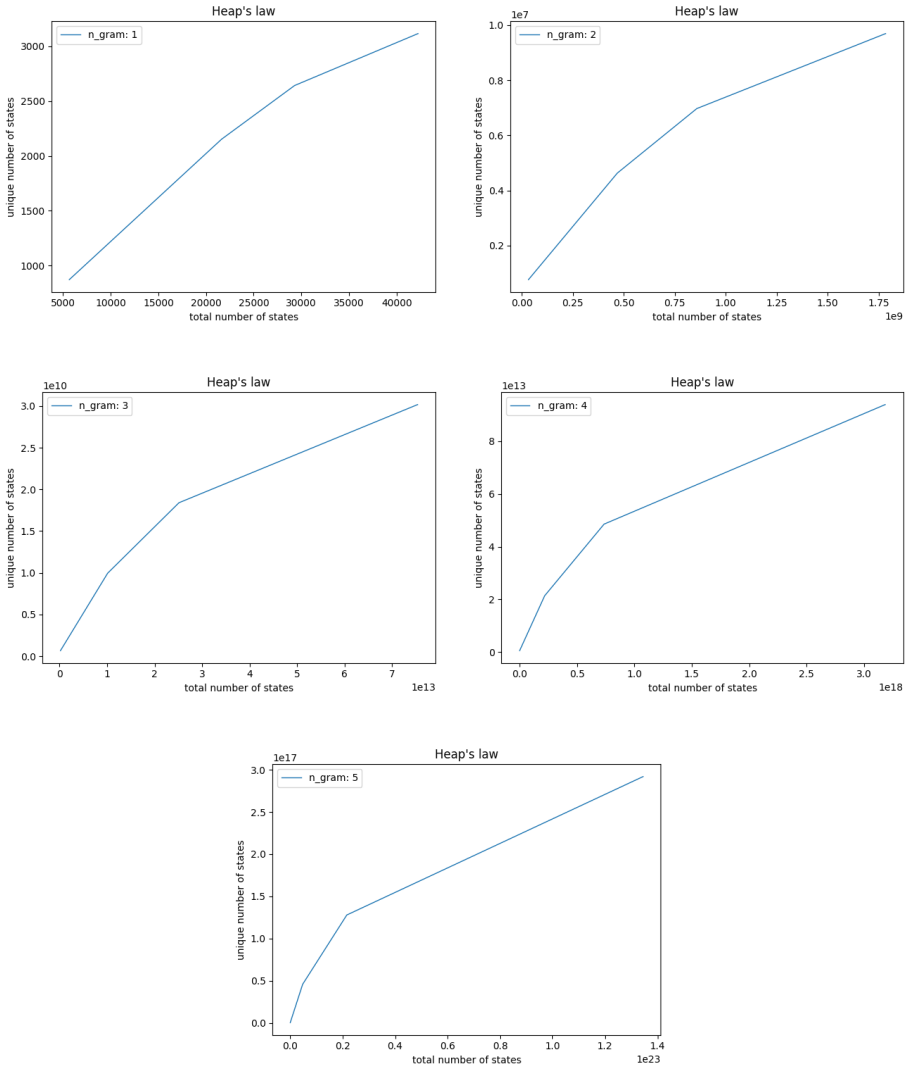


(b) Wykres stałej $constant = ranga * wystapienia$ dla piosenki wygenerowanej na podstawie *somemix.csv* o 100 wersach po 500 słów

Podobieństwo wykresów wskazuje na to, że tekst wyjściowy modelu będzie tej samej jakości, przypominając tekst dany na wejściu.

2.3 Prawo Heapsa

Prawo Heapsa opisuje zależność pomiędzy wielkością dokumentu w jednostce liter, słów, stanów złożonych ze słów w kontekście liczby unikalnych liter, słów, stanów złożonych ze słów pojawiających się w tekście. Na podstawie prawa, stwierdzamy, że liczba unikalnych stanów rośnie wolniej wraz ze zwiększającą się ilością stanów tekstu, co pozwala nam przewidzieć ilość unikalnych w zasobie danych, co pozwala na lepszą ocenę modelu Markova.



Rysunek 4: Wykres zależności liczby stanów unikalnych od rozmiaru tekstu oraz wartości *ngram* dla zbiorów danych: *kyuss.csv*, *led_zeppelin.csv*, *BlackSabbath.csv*, *ac_dc.csv*.

Prawo Heapsa sprawdza się dla danych. Dodatkowo widzimy, że zmiana jest zależna od *ngramu*. Rośnie ona zwykle wolniej dla większych *ngramow* co może oznaczać częste pojawianie się w tekstach określonych złożeń słów.

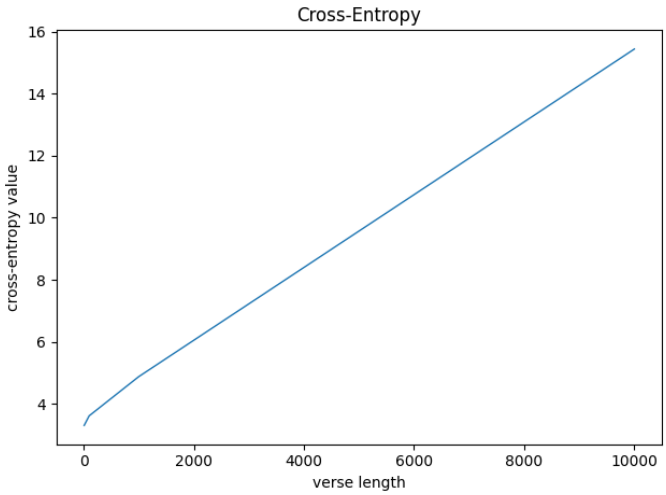
2.4 Entropia Krzyżowa

Entropia Krzyżowa to miara zgodności między dwoma rozkładami prawdopodobieństwa. Pozwala określić jak dobrze model generujący tekst przewiduje następny stan na podstawie poprzednich. W przypadku generowania tekstu, entropia krzyżowa może być wykorzystana do oceny jakości generacji. Im mniejsza wartość entropii krzyżowej, tym większa zgodność między rozkładem prawdopodobieństwa generowanego tekstu a rozkładem prawdopodobieństwa prawdziwych tekstów. Pozwala to ocenić czy model tworzy teksty podobne do tych ze zbioru danych, czy też tworzy nowe i oryginalne sentencje.

Obliczanie Entropii krzyżowej:

- Tworzymy rozkład prawdopodobieństwa wygenerowanego tekstu, zależnie od używanych n-gramów.
- Iterując po każdym n-gramie tekstu wygenerowanego obliczamy sumę iloczynów logarytmu prawdopodobieństwa wystąpienia następnego słowa w modelu oraz prawdopodobieństwa wystąpienia następnego słowa w rozkładzie wygenerowanego tekstu.

  $-\sum log(P(M)) * P(L)$
  , gdzie $P(M)$ oznacza prawdopodobieństwa wystąpienia następnego słowa w modelu, a $P(L)$ prawdopodobieństwa wystąpienia następnego słowa w rozkładzie wygenerowanego tekstu.

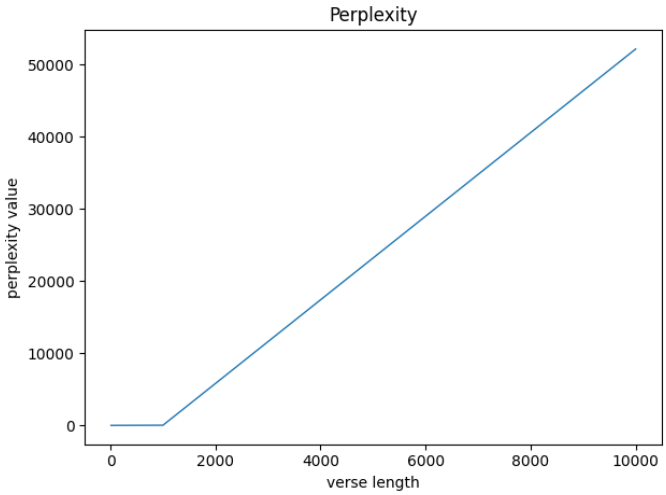


Rysunek 5: Wykres wartości entropii krzyżowej dla tekstu generowanego na podstawie zbioru danych *somemix.csv*, zależnie od rozmiaru wygenerowanego tekstu.

Jak widać, entropia krzyżowa rośnie niemal liniowo względem długości wygenerowanego tekstu. Oznacza to, że model coraz to bardziej generuje tekst niepodobny do oryginalnego zależnie od długości wygenerowanego tekstu.

2.5 Perpleksja

Perpleksja to stopień trudności zrozumienia tekstu, miara nieprzewidywalności modelu. Im niższa tym tekst bardziej przypomina oryginalny i jest bardziej kreatywny. Aby policzyć wartość perpleksji tekstu korzystamy ze wzoru: $Perplexity(M) = 2^{H(L,M)}$ gdzie M oznacza model, L oznacza wygenerowany tekst, a $H(L, M)$ wartość entropii krzyżowej.



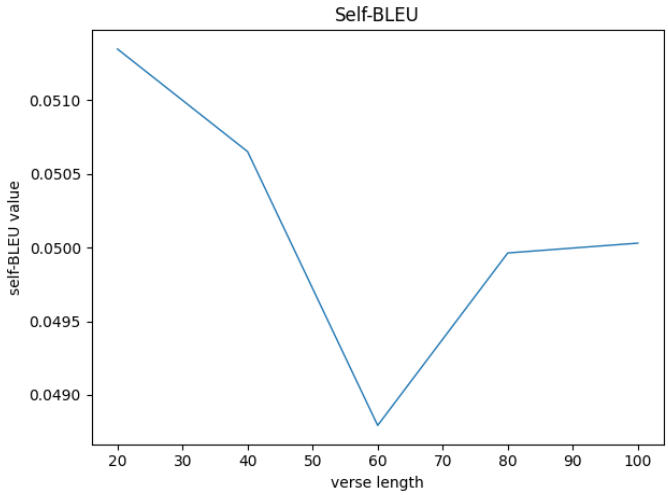
Rysunek 6: Wykres wartości perpleksji dla tekstu generowanego na podstawie zbioru danych *somemix.csv*, zależnie od rozmiaru wygenerowanego tekstu.

Jak widać, wykresy eksperymentu dla entropii krzyżowej i perpleksji się nie różnią, ponieważ w gruncie rzeczy znaczą tą samą miarę.

2.6 Self-BLEU

Self-BLEU określa różnorodność generowanego tekstu. Wykorzystuje wskaźnik BLEU (ang. BiLingual Evaluation Understudy), licząc jego wartość dla kombinacji par wszystkich unikalnych sentencji wygenerowanego tekstu, w tym przypadku wersów piosenki, otrzymując końcowo ich średnią. Im mniejsza wartość wskaźnika tym większa różnorodność w tekście. Metryka pozwala uniknąć monotonności tekstu.

Sam wskaźnik BLEU mierzy podobieństwo między tłumaczeniem maszynowym a jednym lub wieloma tłumaczeniami referencyjnymi poprzez porównanie stopnia pokrycia n-gramów (ciągów po n kolejnych słów) między nimi. Im wyższy wynik, tym większe podobieństwo między tłumaczeniem a referencją. Wartości wskaźnika BLEU mieszczą się w przedziale od 0 do 1, gdzie 1 oznacza idealne dopasowanie tłumaczenia maszynowego do referencji. W praktyce, oczekuje się wyników BLEU powyżej 0,4-0,5, aby uznać tłumaczenie maszynowe za akceptowalne.



Rysunek 7: Wykres wartości Self-BLEU dla tekstu generowanego na podstawie zbioru danych *somemix.csv*, zależnie od rozmiaru wygenerowanego tekstu.

Jak widać, tekst zachowuje wysoką różnorodność, poprzez losowy wybór początku wersu rozkładem równomiernym, a z coraz to większym rozmiarem tekstu napotykamy na nowe frazy, co zwiększa jego różnorodność, jednak wciąż wynik jest zależny od wygenerowanych tekstów.

2.7 Przykładowe wyniki

Przykładowe wyniki generacji 10 wersów po 10 słów dla zbioru danych *english_mixtape.csv*:

Shy yeah repeat everything i want you hard dont get
Slow with plenty of desperation in the night end of
Until that day lost my way you bat your eyes
Back baby cause your man is back wonder where you
Under water forever was their faith i will let you
Windows feel like giving up cause you know theres only
Lets shout lets make it baby now worry like lying
Some room for you and me can you heal what
Shot cmon terminator uzi makers regulators gon na blow my
A poto over the road youre on your move what

Przykładowe wyniki generacji 10 wersów po 10 słów dla zbioru danych *somemix.csv*:

Hell forget about me making a movie turn on a
Bas en haut jaimais manger sa peau je sais que
Main banu tera ehsaas main yaar banavanga akhiyaan milavanga akhiyaan
So stroke me and no reason to believe that parted
Now sexy dance sexy dancer hot as hades early eighties
How sophisticated you know what they do they laugh and
Of brotherly love the feel of silk and your talents
Goddamn alotta brilliant bitch have it you be not much
Line trill tell me youre always gon na need your
Crawling on them haters sick itd be worth more dan

3 Rekurencyjne Sieci Neuronowe

3.1 Wstęp o rekurencyjnych sieciach neuronowych

Rekurencyjne sieci neuronowe (RNN) są specjalnym rodzajem sieci neuronowych, które mają zdolność do uwzględniania kontekstu sekwencji danych. Oznacza to, że RNN są w stanie analizować dane wejściowe w sposób sekwencyjny, zachowując informacje o poprzednich krokach. Ta cecha czyni je szczególnie skutecznymi w modelowaniu danych sekwencyjnych, takich jak język naturalny, dźwięk czy szereg czasowy.

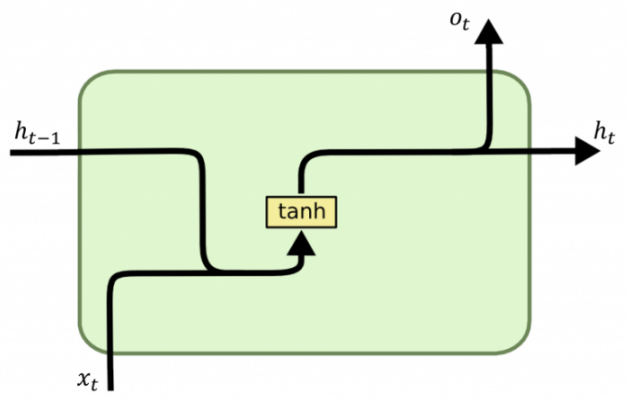
Podstawowym elementem rekurencyjnej sieci neuronowej jest rekurencyjna jednostka, która wykonuje operację na aktualnym kroku czasowym oraz przechowuje stan, który jest przekazywany do następnego kroku. Najpopularniejszym typem jednostki rekurencyjnej jest jednostka LSTM (Long Short-Term Memory) oraz GRU (Gated Recurrent Unit). Obie te jednostki są zaprojektowane w taki sposób, aby rozwiązywać problem znikającego i eksplodującego gradientu, który często występuje podczas uczenia rekurencyjnych sieci neuronowych.

W trakcie uczenia rekurencyjnej sieci neuronowej wsteczna propagacja błędu jest stosowana w celu minimalizacji błędu wyjścia. Jednak w odróżnieniu od tradycyjnych sieci jednokierunkowych, RNN używają również propagacji wstecznej w czasie, aby rozprowadzić gradienty przez wszystkie kroki czasowe. Dzięki temu sieć jest w stanie uczyć się na podstawie kontekstu historycznego i uwzględniać informacje z poprzednich kroków.

Rekurencyjne sieci neuronowe jednak nie są pozbawione wad. Jednym z problemów jest trudność w uczeniu długotrwałych zależności, ponieważ gradienty mogą zanikać lub eksplodować w czasie. W praktyce często stosuje się różne techniki, takie jak LSTM czy GRU, aby radzić sobie z tym problemem. Ponadto, obliczenia w rekurencyjnych sieciach neuronowych są bardziej czasochłonne niż w przypadku sieci jednokierunkowych, co może stanowić wyzwanie w przypadku dużych zbiorów danych.

Podsumowując, rekurencyjne sieci neuronowe są potężnym narzędziem do analizy danych sekwencyjnych. Dzięki swojej zdolności do uwzględniania kontekstu historycznego, są one szczególnie skuteczne w modelowaniu danych sekwencyjnych. Jednak ich skomplikowana natura i trudności w uczeniu długotrwałych zależności wymagają starannego projektowania i optymalizacji.

x_t : input vector $(m \times 1)$.
h_t : hidden layer vector $(n \times 1)$.
o_t : output vector $(n \times 1)$.
b_h : bias vector $(n \times 1)$.
U, W : parameter matrices $(n \times m)$.
V : parameter matrix $(n \times n)$.
σ_h, σ_y : activation functions.

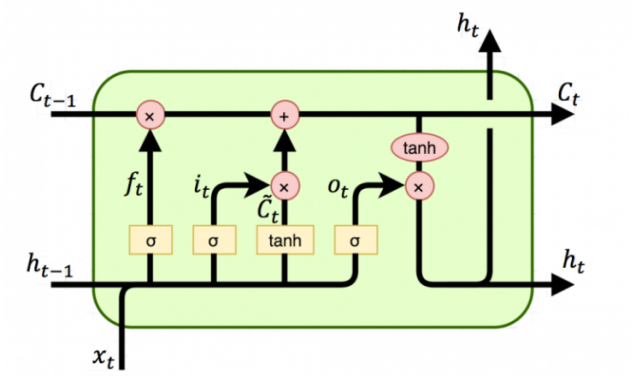


Rysunek 8: Obrazek przedstawia schemat działania Recurrent Neural Network.

3.2 LSTM

LSTM (Long Short-Term Memory) to rodzaj rekurencyjnej jednostki używanej w rekurencyjnych sieciach neuronowych (RNN), która rozwiązuje problem znikającego gradientu. Składa się z bramek wejściowej, zapominającej i wyjściowej, które kontrolują przepływ informacji. Jednostka LSTM ma zdolność do przechowywania informacji przez wiele kroków czasowych dzięki mechanizmowi "ścieżki pamięci"

h_t , C_t : hidden layer vectors.
x_t : input vector.
b_f , b_i , b_c , b_o : bias vector.
W_f , W_i , W_c , W_o : parameter matrices.
σ , tanh : activation functions.

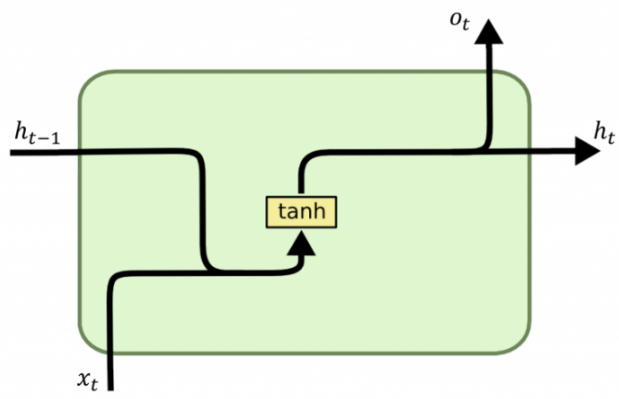


Rysunek 9: Obrazek przedstawia schemat działania Long-Short Term Memory.

3.3 GRU

GRU (Gated Recurrent Unit) to rodzaj jednostki w rekurencyjnych sieciach neuronowych (RNN), której struktura obejmuje bramki resetowania i aktualizacji. Działa podobnie do LSTM, umożliwiając skuteczne modelowanie długotrwałych zależności w danych sekwencyjnych. GRU ma mniejszą liczbę parametrów niż LSTM, a mimo to osiąga podobne efektywności. Jest popularnym rozwiązaniem w przetwarzaniu języka naturalnego, rozpoznawaniu mowy i generowaniu tekstu.

x_t : input vector $(m \times 1)$.
h_t : hidden layer vector $(n \times 1)$.
o_t : output vector $(n \times 1)$.
b_h : bias vector $(n \times 1)$.
U, W : parameter matrices $(n \times m)$.
V : parameter matrix $(n \times n)$.
σ_h, σ_y : activation functions.



Rysunek 10: Obrazek przedstawia schemat działania Gated Recurrent Unit.

3.4 Różnice między LSTM, a GRU

GRU i LSTM są dwoma popularnymi typami jednostek w rekurencyjnych sieciach neuronowych (RNN). GRU ma prostszą strukturę, mniejszą liczbę parametrów i bramki resetowania. LSTM ma bardziej złożoną strukturę, oddzielną komórkę pamięci i trzy bramki. Wybór między nimi zależy od kontekstu i danych sekwencyjnych.

3.5 Problem znikającego gradientu

Problem znikającego gradientu występuje w rekurencyjnych sieciach neuronowych (RNN), gdy gradienty maleją wraz z propagacją wsteczną przez kolejne kroki czasowe. To utrudnia naukę długotrwałych zależności. Jednostki LSTM i GRU zostały opracowane w celu rozwiązania tego problemu, umożliwiając skuteczniejsze modelowanie długoterminowych zależności w danych sekwencyjnych.

3.6 Embedding

Embedding w rekurencyjnych sieciach neuronowych (RNN) to proces przekształcania dyskretnych elementów, takich jak słowa lub symbole, na gęste wektory o niskiej wymiarowości. W przypadku analizy języka naturalnego, embeddingi są

używane do reprezentowania słów lub sekwencji słów w sposób, który zachowuje ich semantykę i relacje między nimi.

Podstawowym celem embedingu w RNN jest przechwycenie znaczenia słów lub sekwencji słów w sposób, który umożliwia modelowi RNN efektywne przetwarzanie i wnioskowanie na podstawie tych danych. Embeddingi są trenowane wraz z resztą sieci RNN i są aktualizowane podczas procesu uczenia.

Proces embedingu zaczyna poprzez przypisania do unikalnego wektora liczba rzeczywistych dla każdego słowa, losowych wartości. Będą one aktualizowane podczas uczenia, aby zwiększyć strukturę symetralną języka. Istotną kwestią jest aby słowa o podobnym znaczeniu miały bliskie sobie wektory, dzięki czemu model RNN może wykrywać podobieństwa i zależności między słowami w trakcie analizy tekstu.

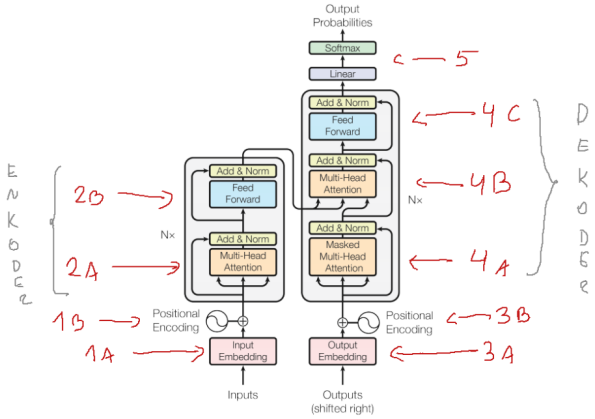
4 Sieci neuronowe na transformatorach

4.1 Przykładowe wykorzystanie sieci neuronowych na transformatoarach:

- przewidywania kolejnego słowa
- tłumaczenia z innego języka
- odpowiadania na pytania
- podsumowywania dokumentu
- parafrazowania
- tworzenia nowego tekstu
- gry w szachy i wielu innych

Modele językowe takie jak BERT, GPT-2 czy GPT-3 wykorzystują właśnie transformatory. Ale przyjżyjmy się jak działa transformator?

4.2 Analiza działania transformera:



Rysunek 11: Obrazek przedstawia schemat działania tranformera z opisami potrzebnymi do dalszego opisu.

1A

Embedding (osadzenie) to proces zamiany słów na liczby lub wektory, które można przetwarzać za pomocą modelu. Słowa są mapowane na przestrzeń wielowymiarową, gdzie podobne słowa są blisko siebie, a różne słowa są oddalone od siebie. Embedding jest podstawowym krokiem w procesie przetwarzania tekstu w modelach językowych.

1B

Positional encoding (kodowanie pozycyjne) to technika, która dodaje informację o pozycji słowa do jego osadzenia (embedding). Pozwala to modelowi uwzględnić kontekst i kolejność słów w zdaniu. Wektory kodowania pozycji są dodawane do wektorów embeddingowych, aby utworzyć reprezentację wektorową słowa z uwzględnieniem kontekstu.

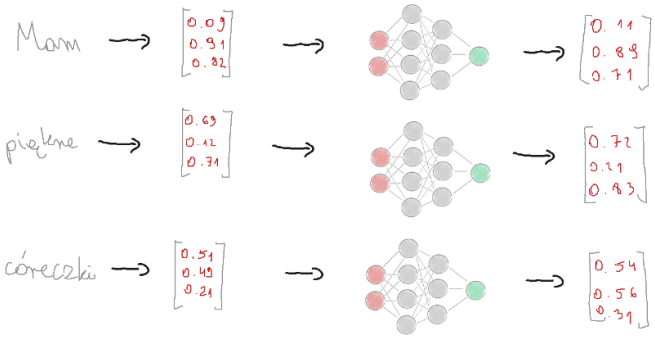
2A

Multi-head attention to mechanizm, który pozwala modelowi na skupienie się na różnych częściach zdania jednocześnie. Jest obliczany dla każdego słowa i bierze pod uwagę zarówno samo słowo, jak i kontekst między słowami. W efekcie otrzymujemy wektor uwagi dla każdego słowa.

Rysunek 12: Obrazek przedstawia schemat działania wekotra uwagi.

2B

Feed-forward network (sieć jednokierunkowa) to warstwa w transformerze, która przekształca wektory uwagi dla każdego słowa. Każdy wektor jest przekazywany przez sieć wielowarstwową, która działa niezależnie dla każdego słowa. Ten proces można zrównoleglić, co przyspiesza obliczenia.



Rysunek 13: Obrazek przedstawia schemat działania wekotra uwagi.

3A i 3B

W tych krokach powtarza się proces opisany w 1A i 1B. Dane wejściowe

to wynikowe zdanie, a wynikiem jest zaktualizowana reprezentacja wektorowa zdania z uwzględnieniem kontekstu i pozycji słów.

4A

Masked multi-head attention (uwaga wielogłowicowa z maskowaniem) jest wykorzystywana w dekoderze modelu transformerowego. Maskowanie pozwala na przetwarzanie sekwencji w sposób sekwencyjny, bez łamania kierunku czasowego. Modele transformers z maskowaniem mogą generować predykcje sekwencyjne, nie mając informacji o przyszłych elementach sekwencji.

Rysunek 14: Obrazek przedstawia schemat działania maskowania na przykładzie zdania po języku hiszpańskim, które ma być przetłumaczone na inny język.

4B

Multi-head attention w dekoderze to proces porównywania wektorów uwagi między dekoderem a enkoderem. Sprawdza on jakość powiązania między wektorami słów w obu częściach modelu, aby model mógł zrozumieć, jak różne słowa są ze sobą powiązane.

4C

Feed-forward network jest ponownie stosowane w dekoderze w celu uproszczenia wektorów uwagi i ułatwienia dalszego przetwarzania przez model.

5

Na końcu wynik z transformera przekazywany jest przez warstwę liniową (linear layer), która przekształca wyniki w wymiarze odpowiadającym liczbie słów w zdaniu. Następnie funkcja softmax zamienia wyniki w prawdopodobieństwa, które można interpretować jako pewność modelu co do przynależności słów do różnych kategorii lub etykiet.

4.3 Wady i zalety:

Zalety

Główną zaletą transformerów jest szybkość działania w porównaniu z siecią RNN lub LSTM. Ponadto elementem, który wpływa na to, że coraz chętniej się po nie sięga, są dużo lepsze wyniki w porównaniu z innymi algorytmami.

Wady

- potrzeba dużej mocy obliczeniowej, aby zostały wytrenowane
- potrzeba naprawdę dużej ilości danych do wytrenowania
- dostępne są przykłady, że gorzej sobie radzą z hierarchicznymi danymi
- w związku z tym, że są stosunkowo nowe, to w przypadku błędów możliwe, że informacja o jego naprawieniu będzie ciężka do znalezienia w internecie.

5 GPT-2

GPT-2 (Generative Pre-trained Transformer 2) to zaawansowany model językowy oparty na architekturze transformer. Został opracowany przez OpenAI i jest jednym z najbardziej znanych i potężnych modeli generatywnych opartych na uczeniu maszynowym.

GPT-2 ma zdolność do generowania wysokiej jakości tekstu, co sprawia, że jest bardzo przydatny w różnych zastosowaniach, takich jak generowanie treści, redagowanie, tłumaczenie i wiele innych. Model ten został wytrenowany na ogromnych zbiorach danych tekstowych z internetu, co pozwoliło mu na zdobycie ogromnej wiedzy na temat języka naturalnego.

Jednym z kluczowych elementów GPT-2 jest jego zdolność do kontekstowego rozumienia tekstu. Dzięki warstwom transformer i mechanizmowi uwagi, model jest w stanie analizować kontekst i zależności między słowami, co prowadzi do generowania bardziej spójnych i sensownych tekstów.

GPT-2 wykorzystuje uczenie nienadzorowane, co oznacza, że jest trenowany na dużych zbiorach danych bez konkretnych etykiet czy celów zadania. Wytrenowany model jest w stanie generować teksty na podstawie danego kontekstu, a jakość generacji zależy od jakości danych treningowych i rozmiaru modelu.

Warto zaznaczyć, że GPT-2 ma również pewne ograniczenia. Ze względu na to, że jest oparty na uczeniu maszynowym nienadzorowanym, może generować nieodpowiednie, niepoprawne lub niezgodne z rzeczywistością treści, co dla niektórych niestety może być zaletą. Ponadto, model może być podatny na wprowadzanie błędnych informacji, które znalazły się w danych treningowych.

Jako generator obrazów

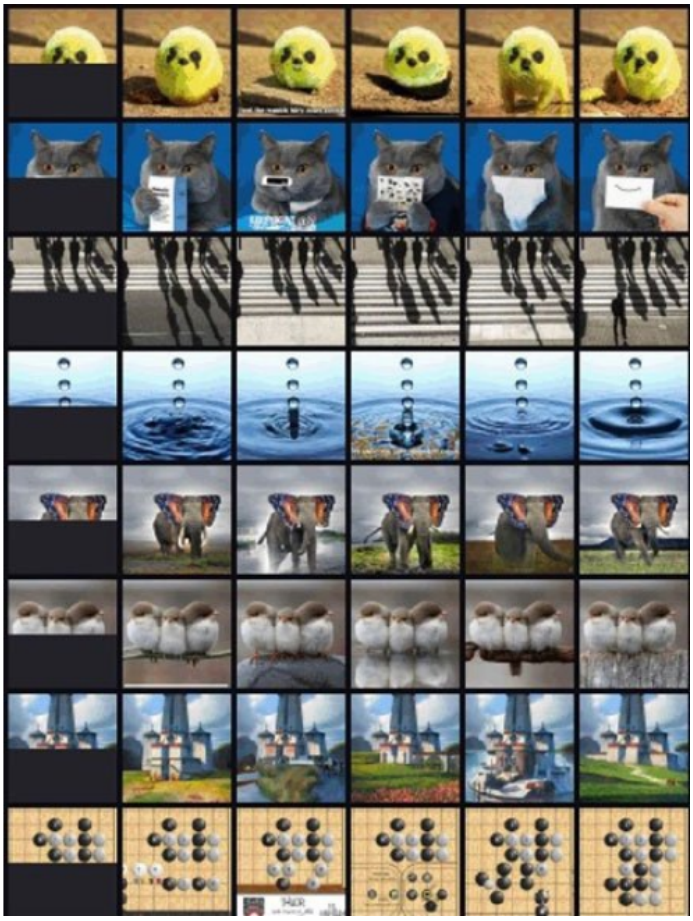
W tym momencie GPT jest wykorzystywany również jako generator obrazów. Nieco ponad miesiąc po premierze GPT-3, superpotężnego algorytmu generującego tekst, jego poprzednik – GPT-2 – znalazł nowe wcielenie. Jego twórcy postanowili sprawdzić, jak ich model zachowałby się, gdyby wyszkolić go nie na terabajtach tekstu, ale na milionach fotografii.

Otóż modele typu Transformer, do których należy GPT-2, określa się jako domain-agnostic, czyli niezależne od przetwarzanych danych. Oznacza to, że model teoretycznie zadziała dla każdych danych, które można zamienić na jednowymiarową sekwencję.

Tekst jest taką sekwencją z natury, ale co z dwuwymiarowym obrazem? Żeby to zadziałało, trzeba go było rozłożyć na czynniki pierwsze – obraz został „linijka po linijce" zapisany jako ciąg pikseli. Każdy piksel z osobna został zakodowany w specjalnie zaprojektowanym dziewięciobitowym systemie określania koloru.

Okazało się, że algorytm działający na jednowymiarowych sekwencjach danych, którego działanie w wielkim uproszczeniu sprowadza się do przewidywania kolejnego elementu układanki z zachowaniem kontekstu całości, świetnie sprawdza się także w generowaniu dwuwymiarowych obrazów.

Jeśli dać mu fragmet obrazu to wytworzy spójny stylistycznie ciąg dalszy i to w różnych wariantach. To nie ma na celu odtworzenie orginału, ale coś nowego o różnych wariantach. Efekt jest analogiczny jak w przypadku generowania tekstu – dalszy ciąg jest całkowicie zmyślony, ale stylistycznie spójny z początkowym fragmentem. Próbki działania algorytmu można zobaczyć na poniższej ilustracji.



Rysunek 15: Obrazek przedstawia schemat działania gpt-2 na obrazach.

6 Realizacja praktyczna z wykorzystaniem RNN

6.1 Użyte narzędzia

- Python3
- Tensorflow
- Keras i Keras_NLP
- Embedding GloVe (6B.100d)
- Gotowy model GPT-2 (za pomocą pakietu gpt_2_simple)
- Teksty piosenek AC/DC, Metalliki i Jimiego Hendrixa

6.2 Struktura kodu

- main.py - plik obsługujący pobieranie danych od użytkownika i wykorzystujący inne moduły do ładowania i trenowania lub generacji tekstów z modeli.
- models.py - zawiera definicje poniżej opisanych modeli.
- data_processor.py - impementacja klasy służącej do obróbki danych m.in. przygotowania danych treningowych i testowych
- song_generator.py - generator piosenek, który wykorzystuje przekazany model i instancję klasy DataProcessor
- edit_distances.py - kod obliczający metryki edit distance
- plots.py - kod rysujący wykresy metryk
- transformer.py - kod trenujący i generujący piosenki za pomocą GPT-2

6.3 Dane uczące

Zbiory danych wykorzystane do treningu i oceny rekurencyjnych sieci neuronowych to łącznie kilkaset tekstów utworów:

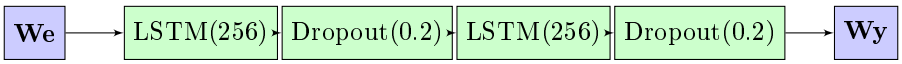
- Tekty piosenek AC/DC do treningu sieci opartych o LSTM lub GRU (ac_dc.csv)
- Połączenie tekstów Metalliki i Hendrixa do dotrenowania modelu GPT-2 (metallica_hendrix.txt)

6.4 Modele RNN

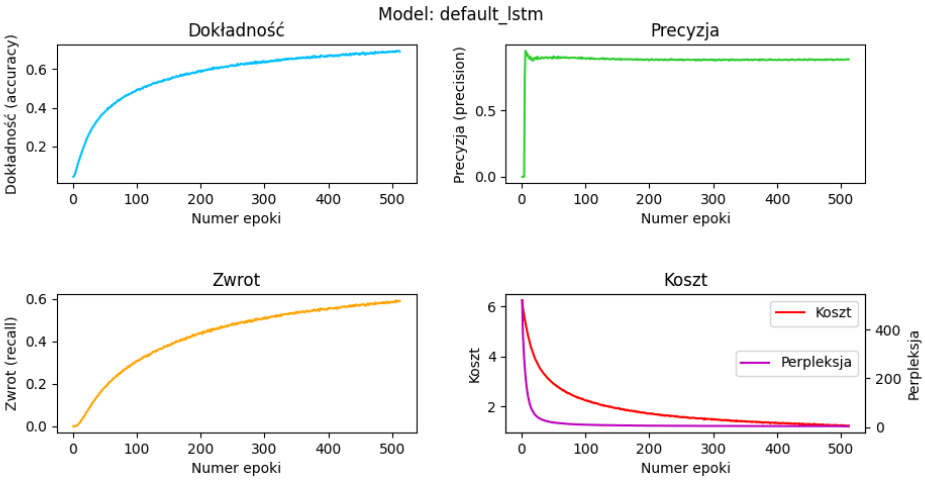
Opisując metrykę Edit Distance wartości rozdzielone średnikami oznaczają generacje odpowiednio dla 4 wersów po 6 słów, 2 wersów po 6 słów, 1 wersu składajacego się z 6 słów i 1 wersu składającego się z 2 słów.

6.4.1 default_lstm

Warstwy modelu



Ocena modelu



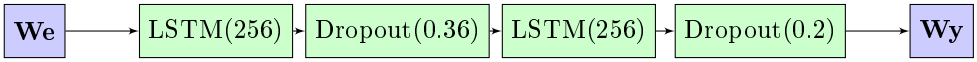
Rysunek 16: Metryki modelu default_lstm

Wynik edit distance (po normalizacji) dla w pełni wytrenowanego modelu:
Średni ED: 0.98 ; 0.97 ; 0.95 ; 0.8125
Minimalny ED: 0.88 ; 0.83 ; 0.67 ; 0.0
Maksymalny ED: 1.0 ; 1.0 ; 1.0 ; 1.0

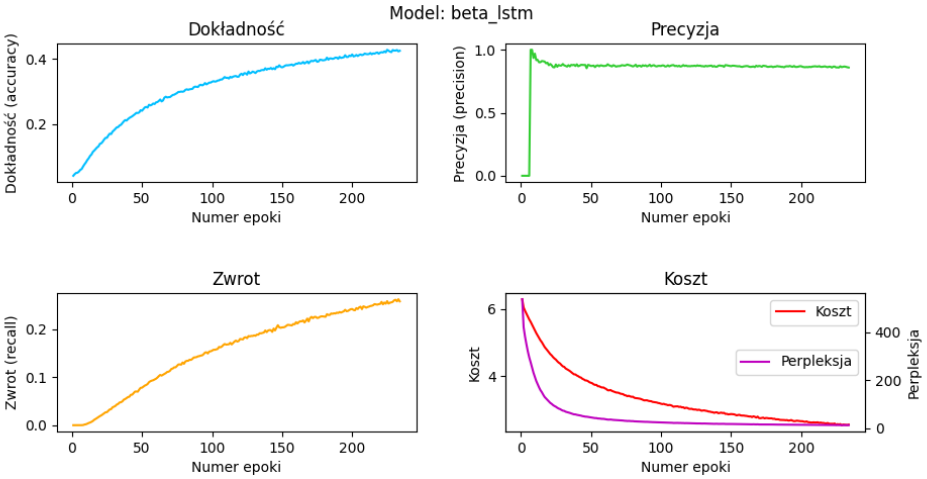
Przykład generacji
Of broad for the way she's
Love out the gotta rock all
Reputation up all yeah on down
Finally oh a man you gotta

6.4.2 beta_lstm

Warstwy modelu



Ocena modelu



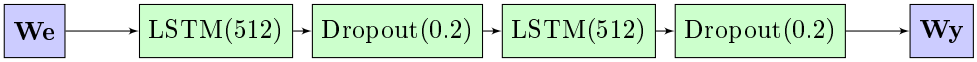
Rysunek 17: Metryki modelu beta_lstm

Wynik edit distance (po normalizacji) dla w pełni wytrenowanego modelu:
Średni ED: 0.98 ; 0.98 ; 0.99 ; 0.97
Minimalny ED: 0.91 ; 0.92 ; 0.83 ; 0.5
Maksymalny ED: 1.0 ; 1.0 ; 1.0 ; 1.0

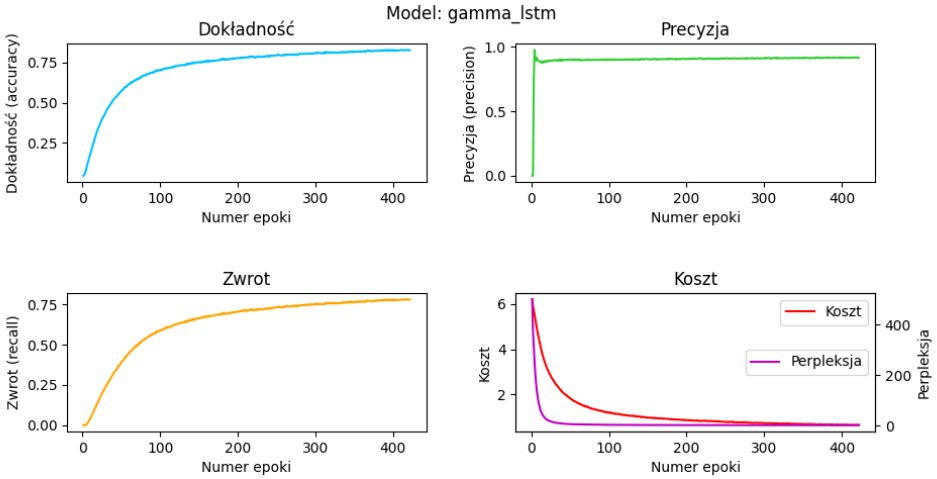
Przykład generacji
Blood sideway up a right I'm
Up love and fight got the
Dark oh rock a with feel
What go come your a can

6.4.3 gamma_lstm

Warstwy modelu



Ocena modelu



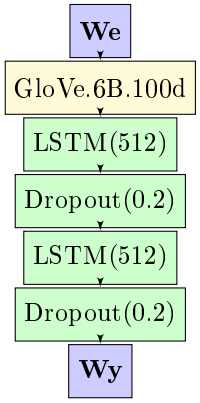
Rysunek 18: Metryki modelu gamma_lstm

Wynik edit distance (po normalizacji) dla w pełni wytrenowanego modelu:
Średni ED: 0.98 ; 0.97 ; 0.99 ; 1.0
Minimalny ED: 0.83 ; 0.83 ; 0.83 ; 1.0
Maksymalny ED: 1.0 ; 1.0 ; 1.0 ; 1.0

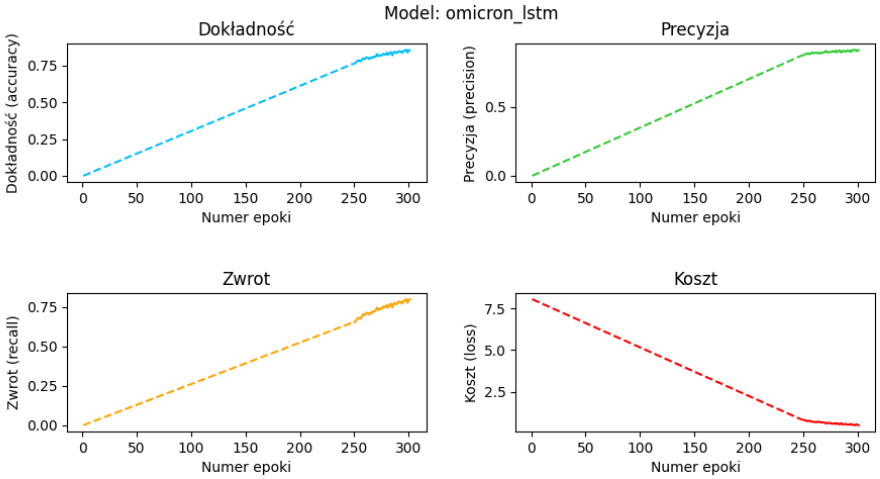
Przykład generacji
You got her this it the
Fire what we're like chorus me
All come she yeah the way
She's commin' all the way prowler

6.4.4 omicron_lstm

Warstwy modelu



Ocena modelu



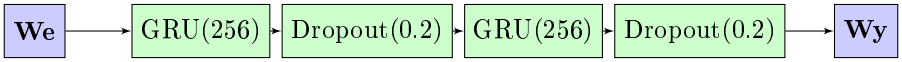
Rysunek 19: Metryki modelu omicron_lstm

Przykład generacji

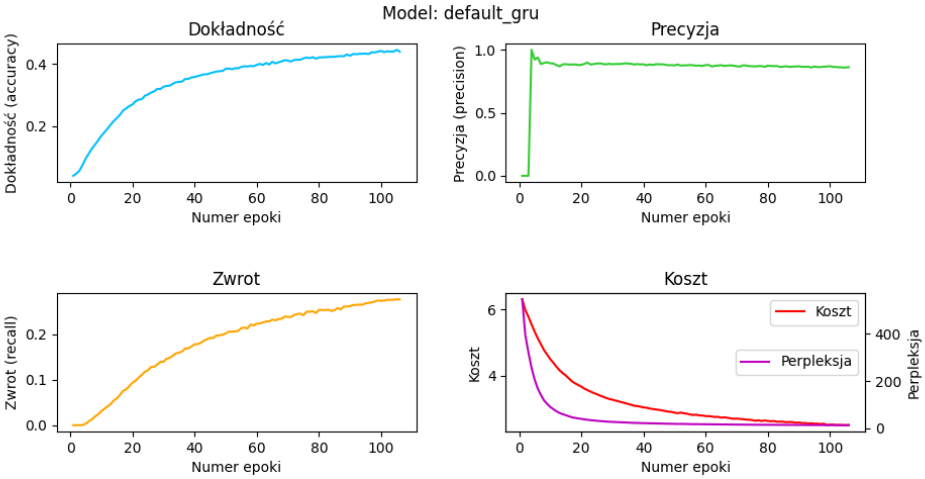
Twilight sting time who crown pain
In a legless need no tears
Yours a take yours lovin' million
Woods lovin' trip a hearts that

6.4.5 default_gru

Warstwy modelu



Ocena modelu



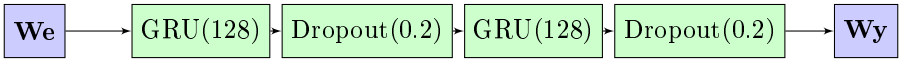
Rysunek 20: Metryki modelu default_gru

Wynik edit distance (po normalizacji) dla w pełni wytrenowanego modelu:
Średni ED: 0.99 ; 0.99 ; 0.99 ; 1.0
Minimalny ED: 0.96 ; 0.92 ; 0.83 ; 1.0
Maksymalny ED: 1.0 ; 1.0 ; 1.0 ; 1.0

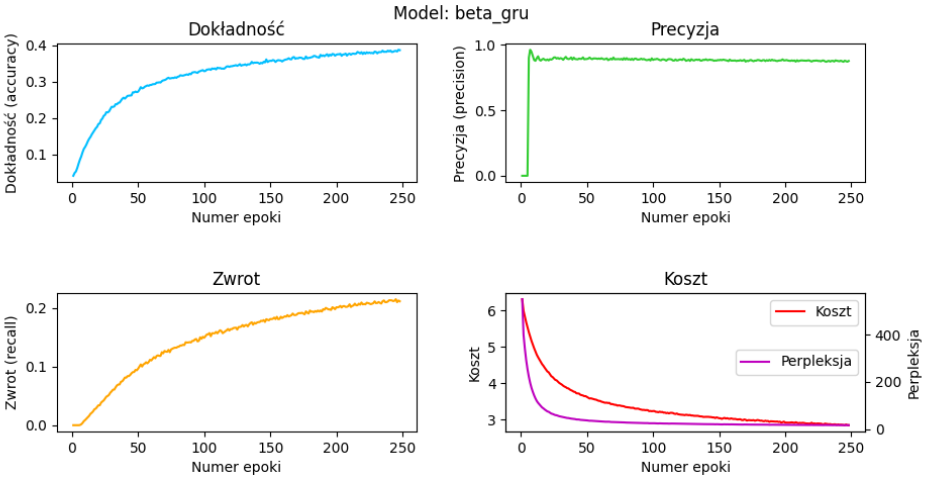
Przykład generacji
Sunset but got on have call
I pull shook oh a can
Of conduct known back satisfy out
You bustin' himself the handful run

6.4.6 beta_gru

Warstwy modelu



Ocena modelu



Rysunek 21: Metryki modelu beta_gru

Wynik edit distance (po normalizacji) dla w pełni wytrenowanego modelu:
Średni ED: 0.98 ; 0.96 ; 0.99 ; 1.0
Minimalny ED: 0.92 ; 0.83 ; 0.83 ; 1.0
Maksymalny ED: 1.0 ; 1.0 ; 1.0 ; 1.0

Przykład generacji
Into from rock way down the
Way the go jack which train
Play when down just I'm the
Way restless makin' and like damnation

6.4.7 Dotrenowany model GPT-2

Do generacji określonego rodzaju tekstów można też wykorzystać już gotowe modele. Jednym z nich jest wcześniej opisany GPT-2. Wykorzystując pakiet gpt-2-simple dotrenowaliśmy model wykorzystując połączenie zbioru piosenek Metalliki i Jimiego Hendrixa. Poniżej znajduje się przykład generacji tekstu przez takie rozwiązanie:

Got to stay up through tonight
Nobody's fool around tonight
Yeah, yeah, ya better get over yourself
Yeah, ya better get over yourself
Everybody's gotta live together
Under the same bed
Every night
Night falls over o'er the Sun