

Generacja tekstów piosenek

Maciej Krzyżanowski, Sebastian Kutny, Tomasz Lewandowski

Kwiecień 2023

Spis treści

| | | |
|----------|---|-----------|
| 1 | Wstęp | 2 |
| 2 | Łańcuchy Markova | 3 |
| 2.1 | Wstęp | 3 |
| 2.2 | Prawo Zipfa | 5 |
| 2.3 | Prawo Heapsa | 8 |
| 2.4 | Entropia Krzyżowa | 10 |
| 2.5 | Perpleksja | 11 |
| 2.6 | Self-BLEU | 12 |
| 2.7 | Przykładowe wyniki | 13 |
| 3 | Rekurencyjne Sieci Neuronowe | 15 |
| 3.1 | Wstęp o rekurencyjnych sieciach neuronowych | 15 |
| 3.2 | LSTM | 15 |
| 3.3 | GRU | 16 |
| 3.4 | Różnice między LSTM, a GRU | 16 |
| 3.5 | Problem znikającego gradientu | 16 |
| 3.6 | Embedding | 16 |

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 "scrapet" 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 oczyszczone poprzez ujednoczenie 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 słów.

2 Łańcuchy Markova

2.1 Wstęp

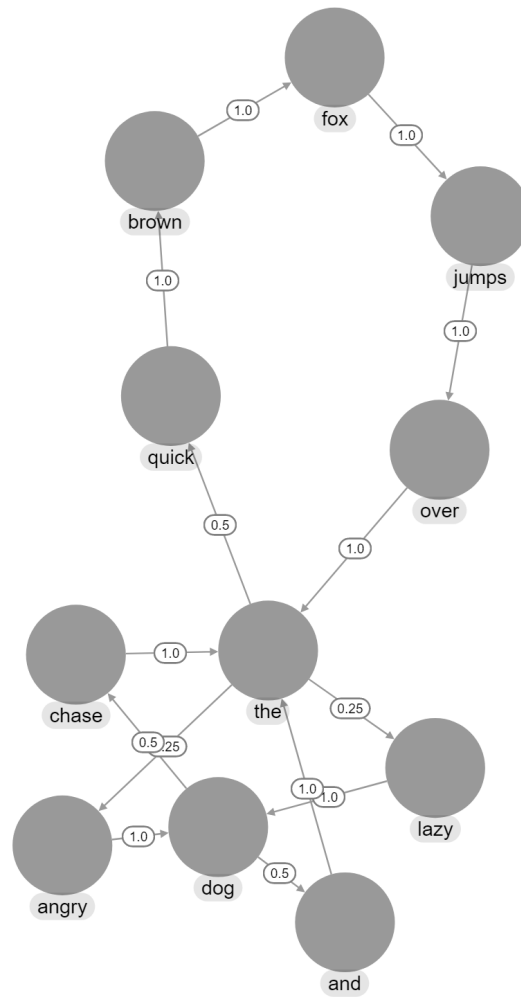
Łańcuchy Markova to matematyczny model służący do generowania tekstu lub sekwencji innych elementów, takich jak dźwięki lub obrazy. Ideą modelu jest analiza sekwencji istniejących elementów i wykorzystanie tych informacji do przewidywania, jakie elementy powinny pojawić się następnie.

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 prawdopodobieństwie wystąpienia po sobie stanów w analizowanym tekście w kontekście poprzednio wygenerowanych stanów. Przykładowo, jeśli w tekście źródłowym po słowie "generator" często 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 "stopniem" 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 projekcie 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 wszystkich pozycji. Przy nieznalezieniu rymujących się słów generacja odbywa się tak jak w zwykłym wypadku. W praktyce szansa na stworzenie rymu jest mała i powinna rosnać 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.

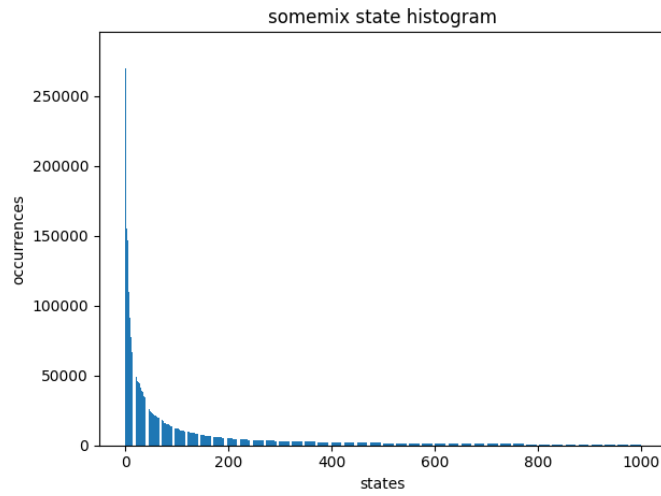


Rysunek 1: Przykład łańcucha Markowa, dla zdania "The quick brown fox jumps over the lazy dog and the angry dog chase the quick brown fox.", dla wartości $ngram = 1$, oznaczającej stany jako pojedyncze słowami oraz wartościami prawdopodobieństw przejść pomiędzy stanami obliczonych na podstawie zdania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

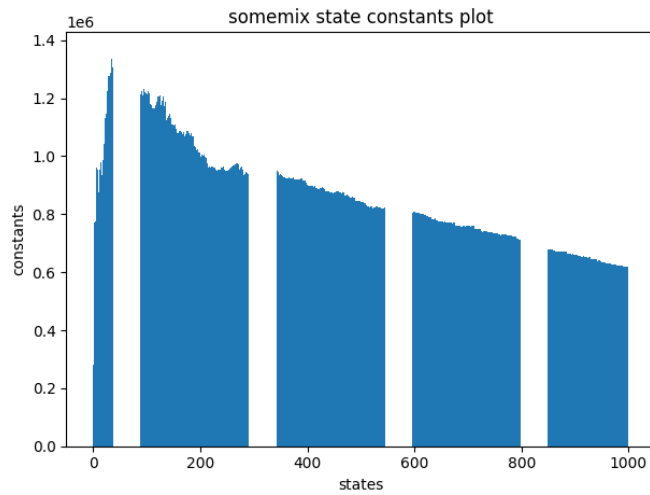
2.2 Prawo Zipfa

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.

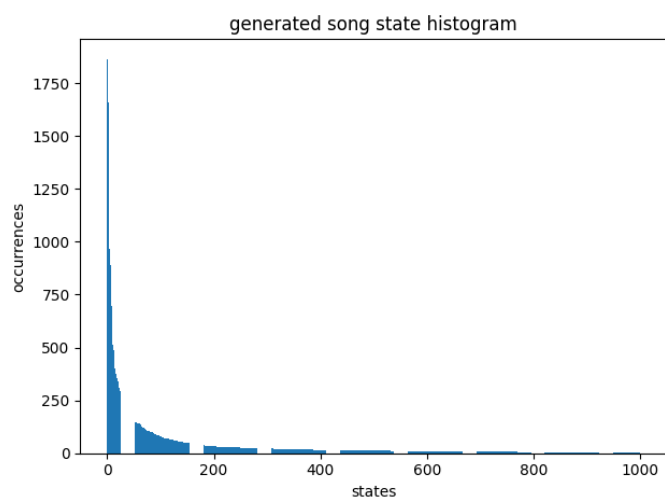
Wykorzystując Prawo Zipfa w generacji tekstów piosenek, można zapewnić, że wygenerowany tekst będzie przypominał rzeczywiste teksty pod względem częstotliwości występowania słów.



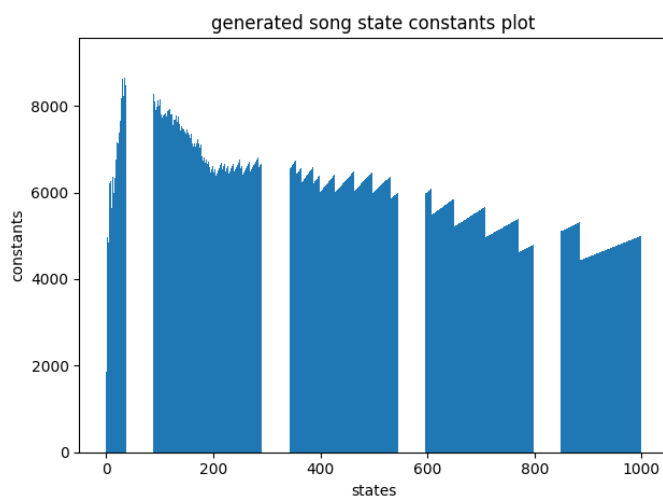
(a) Wykres 1000 najczęściej pojawiających się słów dla zbioru danych *sodemix.csv*.



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



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

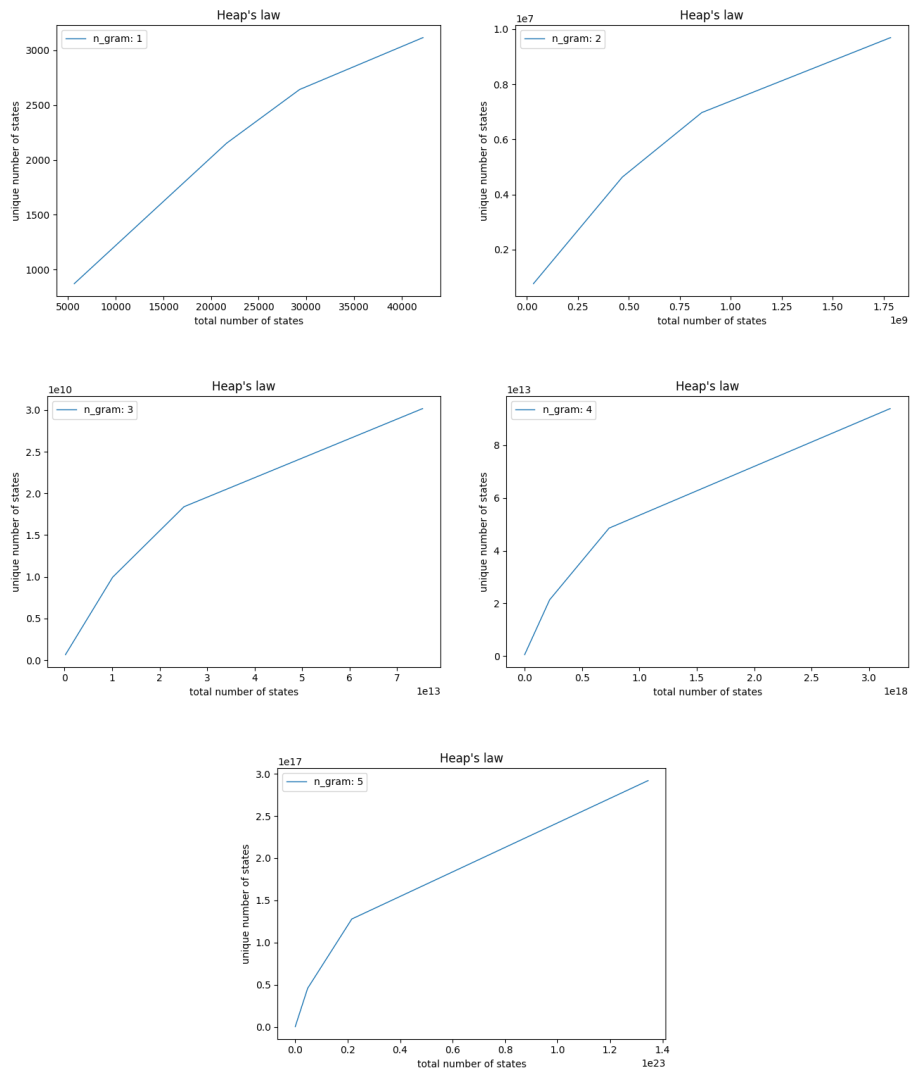


(b) Wykres stałej $constant = ranga * wystapienia$ dla piosenki wygenerowanej na podstawie *somemix.csv* o 100 wersach po 500 słów

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 złożań słów.

2.4 Entropia Krzyżowa

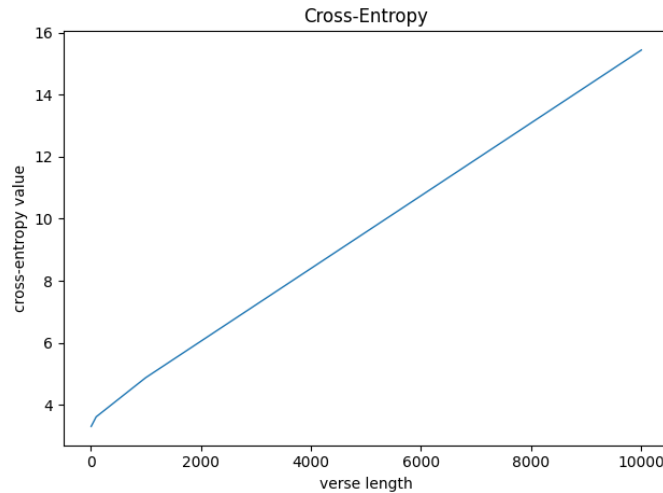
Entropia Krzyżowa to miara zgodności między dwoma rozkładami prawdopodobieństwa. Pozwala określić jak dobrze model generujący tekst przewiduje następny stan na podstawie poprzednich. W przypadku generowania tekstu, entropia krzyżowa może być wykorzystana do oceny jakości generacji. Im mniejsza wartość 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.

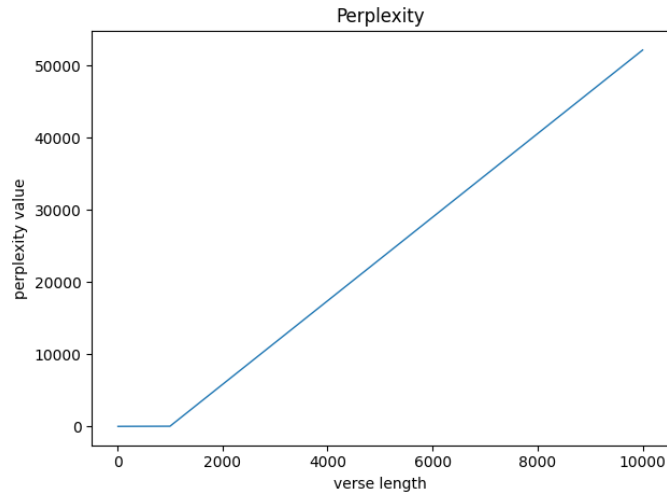


Rysunek 5: Wykres wartości entropii krzyżowej dla tekstu generowanego na podstawie zbioru danych *somemix.csv*, zależnie od rozmiaru wygenerowanego tekstu.

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.



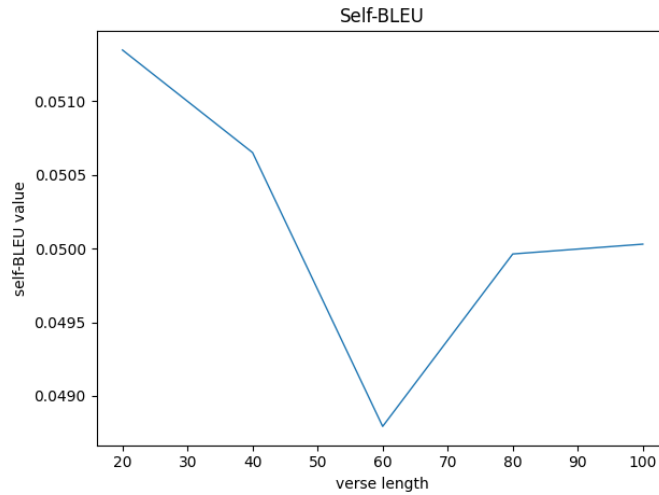
Rysunek 6: Wykres wartości perplexji dla tekstu generowanego na podstawie zbioru danych *somemix.csv*, zależnie od rozmiaru wygenerowanego tekstu.

Jak widać, wykresy eksperymentu dla entropii krzyżowej i perplexji 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 błędów jest stosowana w celu minimalizacji błędów wyjścia. Jednak w odróżnieniu od tradycyjnych sieci jednokierunkowych, RNN używają również propagacji wstecznej w czasie, aby rozpropagować 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.

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 przechowania

wywania informacji przez wiele kroków czasowych dzięki mechanizmowi "ścieżki pamięci"

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.

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 embeddingu 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 embeddingu 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.