POLITECHNIKA KRAKOWSKA im. T. Kościuszki

Wydział Inżynierii Elektrycznej i Komputerowej

Katedra Automatyki i Informatyki E-1

Kierunek studiów: informatyka w inżynierii komputerowej

STUDIA STACJONARNE

PRACA DYPLOMOWA

INŻYNIERSKA

Michał Dubiel

ZASTOSOWANIE SIECI GPT-2 W WYBRANYM PROBLEMIE PRZETWARZANIA JĘZYKA NATURALNEGO

APPLICATION OF THE GPT-2 NETWORK IN A SELECTED NLP PROBLEM

Promotor pracy:

mgr inż. Kazimierz Kiełkowicz

Kraków, 2023

Składam podziękowania dla
mgr inż. Kazimierza Kiełkowicza
za wsparcie przy tworzeniu pracy
oraz wprowadzenie do Sztucznej Inteligencji

Spis treści

1 WSTĘP

Komunikacja człowieka z maszyną od zawsze opierała się na prostych instrukcjach, które nie są naturalną formą interakcji dla ludzi. Z tego powodu programiści i naukowcy od wielu lat szukają nowych sposobów przetwarzania języka naturalnego, aby maszyny mogły go zrozumieć i używać przy komunikacji. Najnowszym trendem w tej dziedzinie są modele transformerów, które aktualnie są najskuteczniejszym rozwiązaniem problemu przetwarzania języka naturalnego.

W ostatnich latach firma openAI rozwijała modele bazowane na architekturze transformera i stworzyła platformę ChatGPT, której podstawą jest agent konwersacyjny czyli algorytm pozwalający na interakcję użytkowników z samym sobą poprzez rozmowę tekstową. Odróżniającą cechą tego agenta od innych nie była tylko jego zdolność do generacji naturalnych odpowiedzi ale również analiza tekstu. ChatGPT ma dostęp do internetu co sprawiło że był wstanie rozmawiać na większość tematów. Zamysłem tego projektu jest zrozumienie technologii i dziedziny stojącej za tą platformą.

Rys. 1.1: Logo platformy ChatGPT
źródło: https://en.wikipedia.org/wiki/ChatGPT

Głównym celem projektu jest przedstawienie najważniejszych zagadnień związanych z dziedziną przetwarzania języka naturalnego oraz jej aktualnego trendu w postaci modeli transformerów. W pracy przedstawione zostaną różnice pomiędzy popularnymi modelami GPT-2 i BERT. Ponadto, wykorzystana zostanie sieć GPT-2 do zadania sumaryzacji tekstu w języku angielskim. Praca została podzielona na 5 etapów:

NLP

Dziedzina przetwarzania języka naturalnego (NLP, od natural language processing) składa się z wielu ważnych zagadnień. W pierwszej kolejności należy zrozumieć język oraz jego strukturę, przyglądając się jego najważniejszym częścią składowym. Zostanie wyjaśniona naturalna problematyka języka w kontekście jego komputerowego przetwarzania. Oprócz języka, równie ważne jest zrozumienie z jakimi zadaniami mierzą się specjaliści z tej dziedziny, które w pracy podzielono na podstawowe i branżowe. Pod koniec tego rozdziału omówione zostaną rozwiązania jakie występowały w tej dziedzinie, od heurystycznych opartych na regułach po sieci neuronowe NLP.

Głębokie uczenie i sieci neuronowe

Głębokie uczenie jest dziedziną sztucznej inteligencji skupiającą się na sieciach neuronowych. Na początku rozdziału zostanie omówiony koncept sztucznego neuronu. Praca przedstawi jego części składowe oraz przedstawi ich najpopularniejsze wariacje (w przypadku funkcji aktywacji). Następnie opisana zostanie architektura sieci feed forward która jest najpopularniejszą siecią neuronową w dziedzinie. Ostatnią częścią tego rozdziału będzie szczegółowe objaśnienie algorytmu uczenia się sieci neuronowej nazywane wsteczną propagacją.

Transformery

Model transformera jest najnowszym trendem w dziedzinie NLP. Ten rozdział dogłębnie opisuje jego architekturę, opisując algorytmy word2vec i Multi-head attention działające wewnątrz transformera oraz współpracę między nimi poprzez stosowane połączenia resztkowe. Opisane zostaną bloki enkoderów i dekoderów z których jest zbudowany transformer.

GPT vs BERT

W dziedzinie NLP wiele algorytmów inspirowało się modelem transformera. W tej części pracy zestawione zostaną dwa najpopularniejsze modele inspirowane jego budową: GPT i BART. Opisana zostanie ich architektura w celu podkreślenia najważniejszych różnic w ich budowie i działaniu. W kolejnym kroku praca skupi się na metodach treningowych na jakich te modele się uczyły oraz zamyśle twórców. Na końcu tego rozdziału zostaną wyciągnięte wnioski z zestawienia.

Zastosowanie GPT-2

Ostatni rozdział pracy będzie opisywać przebieg wytrenowania modelu GPT-2 do tworzenia podsumowań tekstów (sumaryzacji). W rozdziale przedstawione zostaną technologie wykorzystane w pracy. Opisane zostaną kolejne kroki w tworzeniu oraz trenowaniu modelu. Po treningu, wykonana zostanie jego ewaluacja, której wyniki zostaną opisane i wyjaśnione. W ostatniej fazie dokumentu zostanie opisana aplikacja okienkowa dołączona do pracy. Zawierać ona będzie interfejs graficzny pozwalający użytkownikowi wchodzić w interakcję z wytrenowanym modelem.

2 PRZETWARZANIE JĘZYKA NATURALNEGO

NLP (z ang. Natural language processing) jest dziedziną sztucznej inteligencji skupiającą się na przetwarzaniu języka naturalnego i interakcji między komputerem a człowiekiem. Aby pozwolić użytkownikowi współdziałać z komputerem w naturalnym języku, programiści pracują na stworzeniem algorytmów, które mogą rozumieć, analizować i generować ludzki język w formie mówionej i pisanej.

2.1 STRUKTURA I DEFINICJA JĘZYKA

W wielu przypadkach badania nad NLP wymagały znajomości lingwistyki, czyli nauki o systemie i strukturze języka naturalnego. Język możemy podzielić na 4 podstawowe elementy:

Fonemy

Bazowe jednostki dźwiękowe języka. Pojedyncze fonemy nie mają znaczenia, ale ich kombinacje już mają. Przykładowo język angielski, który będziemy obsługiwać w tym projekcie posiada 44 fonemy zawarte w alfabecie angielskim. Fonemy są wykorzystywane w obsłudze ludzkiej mowy.

Morfemy i leksemy

Są wynikiem połączenia fonemów i stanowią najmniejsze jednostki posiadające znaczenie w języku. Mogą występować jako słowa lub przedrostki i przyrostki. W słowie „multimedia” mamy 2 morfemy: „multi” oraz „media”. W zadaniach tokenizacji zdań, celem jest podzielenie zdania na morfemy.

Składnia

Regułu i struktury kombinacji słów(morfemów i leksemów) tworzących poprawne zdania oraz wyrażenia. Patrzymy na relacje pomiędzy słowami oraz ich układem. To ten element odpowiada za logikę podmiotu, orzeczenia czy dopełnienia. Składnia jest wykorzystywana przy ekstrakcki encji i relacji.

Kontekst

Znaczenie ułożonej kombinacji słów, zależne od okoliczności oraz otoczenia danego zdania. Określa jak elementy języka łączą się w nowe znaczenia. Stosowane w sumaryzacji tekstu czy analizy sentymentu.

Te elementy odpowiadają na pytanie czym jest język i dzięki ich zrozumieniu możemy tworzyć algorytmy rozwiązujące problemy NLP. Trzeba podkreślić jednak że wymieniono tylko kilka podstawowych pojęć w celu zobrazowania problemu. Lingwistyka jest bardzo rozległą dziedziną, której dogłębne zrozumienie może być

wymagane przy tworzeniu skuteczniejszych modeli i algorytmów wykonujących zadania obejmujące język naturalny.

2.2 Problemy natury języka

Kolejnym ważnym aspektem w tej dziedzinie są problemy językowe. Wywodzą się z natury języka, która dla maszyny jest obca. Poniżej znajduję się czołówka problemów z jakimi mierzą się programiści i naukowcy pracujący w dziedzinie NLP.

Wieloznaczność

Istnieją zdania w których kontekst oraz znaczenie mogą być różnie odbierane co utrudnia techniczne pra. W zdaniu „Monika spotkała idącą przyjaciółkę ze swoim mężem.” nie możemy jednogłośnie określić do kogo odwołuje się słowo „swoim”.

Powszechna wiedza

Przy komunikacji między ludzkiej jest używana powszechna wiedza, której maszyny nie posiadają. Jeśli maszyna dostanie zdanie „incydent pogryzienia z udziałem psa i człowieka” dla człowieka oczywistym będzie, że pies ugryzł człowieka, ale dla maszyny nie ma pewności kto ugryzł kogo z uwagi na brak powszechnej wiedzy o psach i ludziach.

Różnorodność języków

Prawdopodobnie największym problemem przy pracy z językiem jest jego różnorodność. W samej Europie jest 225 różnych języków których elementy mogą się diametralnie różnić. Algorytm przystosowany do języka angielskiego nie działa w języku polskim, a jego przeniesienie może być bardzo kosztowne i niepewne. Idąc w drugą stronę problemu, zbudowanie neutralnego językowo modelu jest bardzo trudne koncepcyjnie.

2.3 Zadania NLP

Zadania NLP określają cele do jakich algorytmy są trenowane. Możemy podzielić je na podstawowe oraz specyficzne dla branży. Podstawowymi zadaniami możemy nazwać:

- Klasyfikację tekstu
- Ekstrakcję informacji
- Agent konwersacyjny
- Generację tekstu
- Tłumaczenie maszynowe

Takie zadania są podstawą do tworzenia zaawansowanych algorytmów służących wielu branżom. Algorytmy przetwarzające język naturalny możemy spotkać codziennie w formie:

- Klasyfikacji spamu
- Wyszukiwarki
- Asystentów osobistych
- Wsparcia dla systemów medycznych

Modele językowe rewolucjonizują nasze podejście do wiedzy ludzkiej, umożliwiając przetwarzanie ogromnych ilości informacji i wydobywania nowych wniosków z języka naturalnego. Ta zdolność ma potencjał przekształcenia sposobu, w jaki postrzegamy oraz wykorzystujemy zgromadzoną ludzką wiedzę.

2.4 ZNANE ROZWIĄZANIA

W pierwszych etapach pracy nad NLP, jedynym możliwym podejściem było to oparte na heurystyce. Naukowcy i programiści musieli dogłębnie znać reguły języka aby skutecznie je wprowadzać prostymi instrukcjami do maszyny. Słowniki oraz tezaurusy[1] były wymagane przy działaniu algorytmów opartych na prostych zakodowanych regułach. Istniały bazy wiedzy, których zadaniem było przechowywanie relacji i powiązań między słowami. Dobrym przykładem algorytmów z początkowych lat dziedziny NLP są regeksy.

W kolejnych latach zaczęto stosować uczenie maszynowe do bardziej zaawansowanych algorytmów NLP. Wśród nich mogliśmy wyróżnić:

- Naiwny klasyfikator bayesowski
- Maszynę wektorów nośnych
- Model Markowa
- Warunkowe pola laserowe

Od kilku lat dziedzina uczenia maszynowego jest zdominowana przez głębokie uczenie, czyli naukę o sieciach neuronowych. Sieci te sprawdzały się w złożonych i nieustrukturyzowanych danych, do których nie mogły się dostosować zwykłe algorytmy uczenia maszynowego. Cechą języka jest jego złożoność oraz brak struktury co sprawiło że programiści zaczęli wykorzystywać sieci neuronowe również w tej dziedzinie.

Korzystano z modeli które stosowały rozbudowane sieci neuronowe do zapamiętywania powiązań oraz reguł w języku. Wśród takich modeli wyróżniały się następujące modele:

- Rekurencyjnej sieci neuronowe
- Długiej pamięci krótkoterminowej
- Konwolucyjnych sieci neuronowych

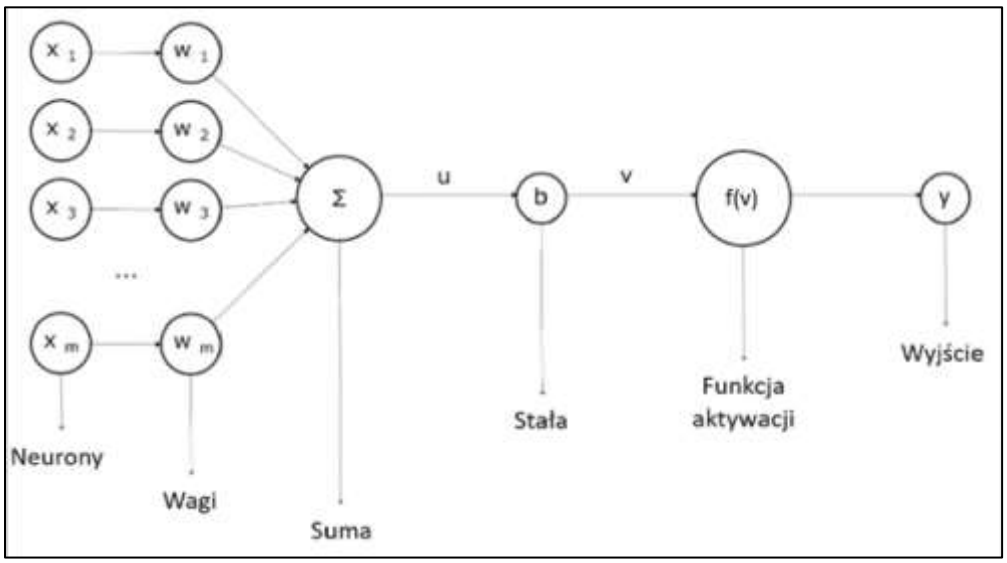
[1] Tezaurus – słownik odzwierciedlający strukturę pola semantycznego danego języka informacyjno-wyszukiwawczego, obejmujący deskryptory, ich relacje oraz reguły stosowania

3 UCZENIE GŁĘBOKIE I SIECI NEURONOWE

Uczenie głębokie jest dziedziną uczenia maszynowego skupiającą się wokół sieci neuronowych. Modele te są inspirowane budową ludzkiego mózgu i zrewolucjonizowały dziedzinę sztucznej inteligencji.

3.1 MODEL NEURONU

Neuron jest podstawową jednostką sieci neuronowej, która posiada moc generowania predykcji. Jego architektura wspiera łączenie się wielu neuronów w większe struktury które zapewniają lepsze wyniki



Rys. 3.1.1 Architektura sztucznego neuronu, źródło: Własne

Powyżej widzimy pojedynczy neuron w przybliżeniu. Na wejściu neuron otrzymuje wektor:

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \tag{3.1.1}$$

- $X \in \mathbb{R}^m$ – wektor wejściowy neuronu
- i = 1,…,m – indeks neuronu wejściowego
- $x_i \in \mathbb{R}$ – sygnał wejściowy (liczba) o indeksie i, może być wyjściem innego neuronu

W następnym kroku sumujemy elementy wektora, a do wyniku dodajemy wyraz wolny, dzięki któremu możemy dodatkowo kontrolować działanie neuronu.

Tak przygotowane wejście jest przesyłane do funkcji aktywacji, która zwróci wynik naszego neuronu. Postać neuronu możemy zatem zapisać jako:

$$y = f(\sum_i x_i W_i + b) \tag{3.1.2}$$

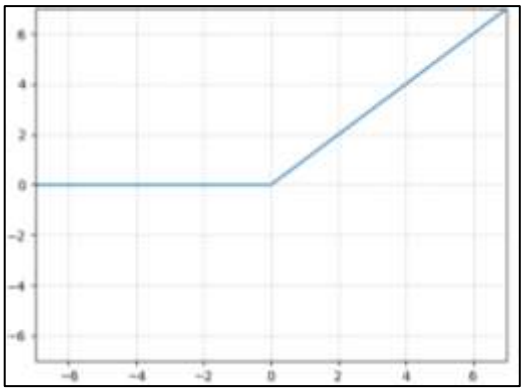
- y – wyjście neuronu
- $f: \mathbb{R} \to \mathbb{R}$ – funkcja aktywacji
- i = 1,…,m – indeks sygnału wejściowego neuronu
- x_i – sygnał na i-tym wejściu neuronu
- $W_i \in \mathbb{R}$ – waga wejścia na i-tym wejściu neuronu
- $b \in \mathbb{R}$ – stała dodawana na końcu sumowania

3.1.1 Funkcje aktywacji

Funkcja aktywacji definiuje zachowanie neuronów w sieci. Wybór odpowiedniej funkcji jest kluczową decyzją przy tworzeniu sieci neuronowych, dlatego w tym podrozdziale omówimy najpopularniejsze funkcje.

Relu

Relu jest jedną z najprostszych funkcji aktywacji używanych do tworzenia sieci neuronowych. W wielkim skrócie neuron jest aktywny (wyjście jest większe od zera) gdy wejście funkcji jest większe od zera. Oprócz prostoty, tego typu funkcja ma zachowanie przybliżone do funkcji liniowej co pomaga w optymalizacji sieci neuronowej.



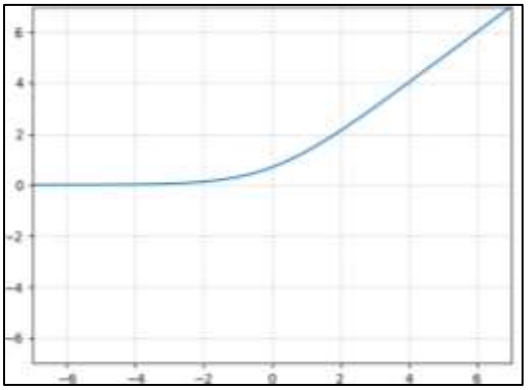
Rys. 3.1.1.1 Funkcja Relu, źródło: Własne

$$f(x) = \max(0\,,\, x) \tag{3.1.1.1}$$

- max(a,b) – funkcja wybierająca maksymalną wartość (a lub b)

Softplus

Jest to funkcja podobna do funkcji RELU z różnicą łagodnego przejścia od zera co może być pomocne przy zapobieganiu zanikania danych bliskich zeru. Prostota tej funkcji oraz brak ograniczeń górnych sprawiają, że ta funkcja sprawdzi się przy trenowaniu sieci, dla której zakładamy pozytywne aktywacje neuronów.

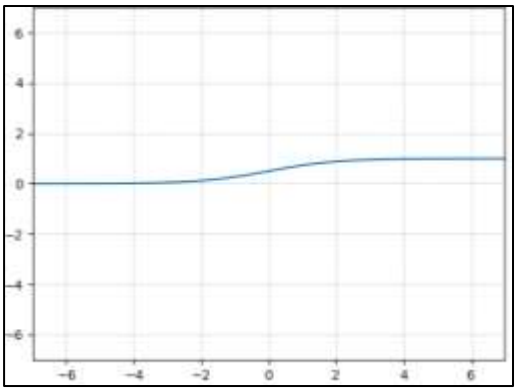


Rys. 3.1.1.2 Funkcja Softplus,
źródło: Własne

$$f(x) = \ln(1 + e^x) \tag{3.1.1.2}$$

Sigmoid

W odróżnieniu od wyżej opisanych ta funkcja może zwracać wartości w zbiorze od 0 do 1 co jest bardzo użyteczne w zadaniach klasyfikacyjnych, ponieważ wyjście można interpretować jako prawdopodobieństwa etykiet dwóch klas.



Rys. 3.1.1.3 Funkcja Sigmoid ,
źródło: Własne

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.1.1.3}$$

Softmax

Funkcja sigmoid sprawdza się gdy do rozwiązania problemu wystarczą dwie klasy (np. prawda, fałsz), ale przy bardziej zaawansowanych problemach potrzebujemy większego zakresu klas. W takich sytuacjach korzystamy z rozszerzenia funkcji sigmoid, którą nazywamy softmax. Na wejściu funkcja otrzymuje wektor:

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{3.1.1.4}$$

- X – wektor wejściowy
- n = {1 … m} – liczba klas
- x_n – wejście dla klasy n

Dla każdego z tych wejść stosujemy funkcje:

$$f(x) = \frac{e^x}{\sum_{j=1}^{K} e^{x_j}} \tag{3.1.1.5}$$

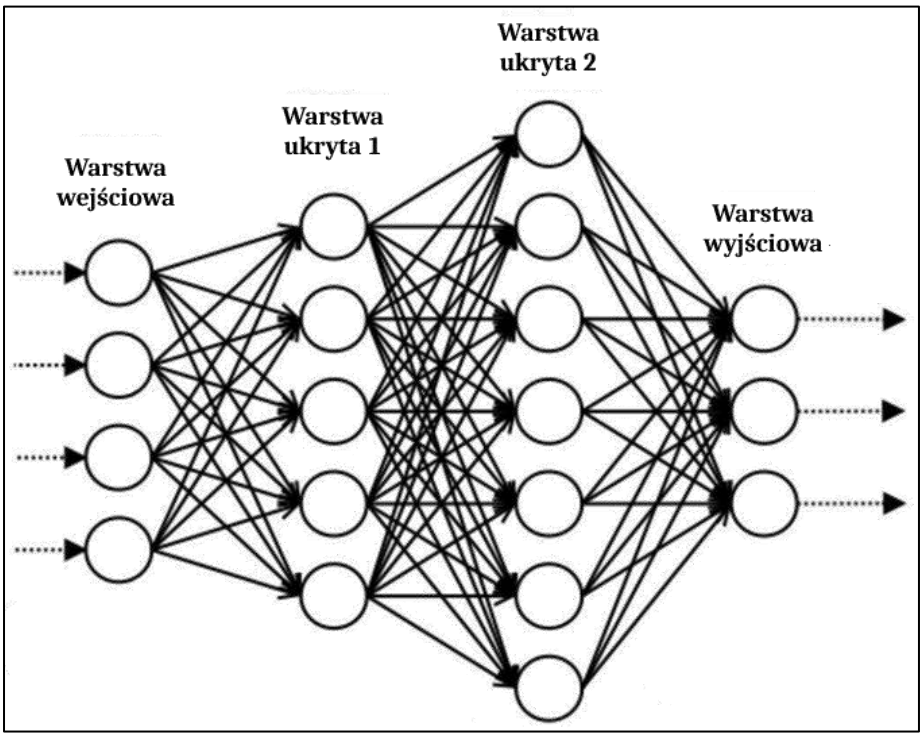
- j = {1 … K} – wymiar wektora wejściowego
- x_j – element wektora wejściowego

Powyższa funkcja zwraca prawdopodobieństwo danej klasy. Suma wyników działania powyższej funkcji dla całego wektora wejść wynosi 1.

Wszystkie wyżej wymienione funkcje mają jedną, bardzo ważną cechę, bez której nie dałoby się skutecznie zastosować algorytmów uczenia takich jak wsteczna propagacja jaką jest dobra różniczkowalność.

3.2 ARCHITEKTURA SIECI NEURONOWYCH

Możemy spojrzeć na sieć neuronową jako funkcje matematyczną która przyjmuje wektor z danymi, a zwraca wektor zawierający wynik. Poniżej znajduje się przykładowa architektura sieci neuronowej typu feed-forward:



Rys. 3.1.1 Architektura sieci neuronowej, źródło: Własne

Sieć neuronowa składa się z neuronów oraz połączeń między nimi ułożonych w co najmniej 3 warstwy.

W pierwszej warstwie, którą nazywamy wejściową dane są wprowadzane do sieci neuronowej. Ilość neuronów w warstwie wejściowej musi odpowiadać wielkości wektora z danymi jakie chcemy przesyłać do modelu. Ostatnia warstwa nazywana wyjściową posiada neurony które przechowują wektor wyjściowy (wynik naszej funkcji). Wszystkie warstwy pomiędzy wejściową i wyjściową nazywamy ukrytymi.

Połączenia służą do przesłania sygnału, czyli wyniku działania neuronu do kolejnych neuronów. Ważną cechą połączenia jest ich waga, czyli "siła" danego połączenia. Im wyższa wartość wagi tym neuron wysyłający sygnał ma większy wpływ na działanie neuronu odbierającego sygnał.

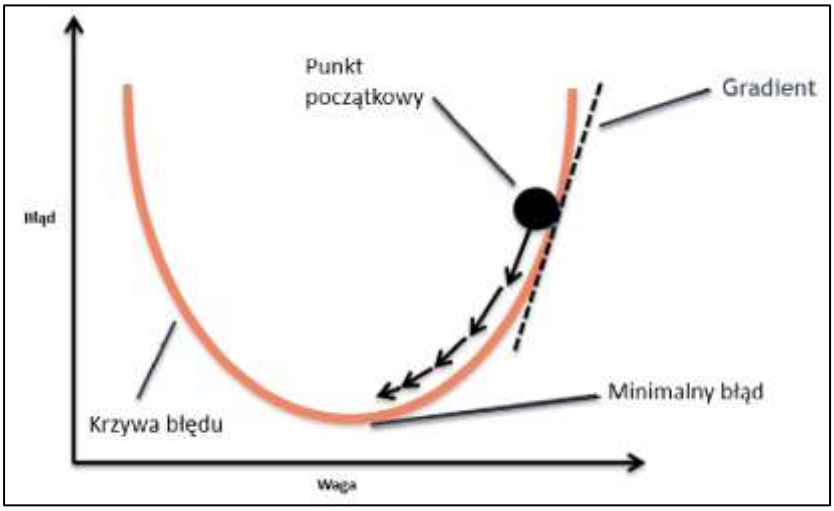
Sieci typu *feed-forward* są jednymi z najpopularniejszych, a wyróżniającą dla nich zasadą jest strategia połączeń. Każdy z neuronów jest połączony z każdym neuronem w warstwie następnej co zapewnia prostotę działania i zrozumienia działania modelu

3.3 WSTECZNA PROPAGACJA

W celu znalezienia odpowiednich wag i stałych dla sieci neuronowej korzystamy z algorytmu wstecznej propagacji, który jest implementacją algorytmu gradientu prostego dla sieci neuronowych.

Gradient jest wektorem pochodnych składowych funkcji i przy jego pomocy wyznaczamy kierunek (wektor) wzrostu funkcji. Algorytm gradientu prostego w kolejnych krokach działania wykorzystuje gradient funkcji do znalezienia jej minimum lokalnego.

Na samym początku ustawiamy losowy punkt. W każdym kroku model przewiduje na podstawie kolejnych danych w zbiorze treningowym poprawny wynik, który następnie jest wstawiany do gradientu funkcji dzięki czemu możemy obliczyć nowy punkt. Algorytm będzie się wykonywał do momentu spełnienia wyznaczonego przez programistę kryterium.



Rys. 3.3.1: Wizualizacja algorytmu Gradientu prostego dla funkcji kosztu
źródło: Własne

$$a_{n+1} = a_n - \gamma \nabla f(a_n) \tag{3.3.1}$$

- a_{n+1} - nowy punkt w kroku n + 1
- a_n - aktualny punkt w kroku n
- γ - współczynnik uczenia, wyznaczamy za jego pomocą wielkość skoków między punktami
- f – funkcja dla której szukamy minimum lokalnego
- $\nabla f(a_n)$ - gradient (pochodna lub wektor pochodnych) funkcji

W bardziej zaawansowanych algorytmach wartość współczynnika uczenia γ maleje w miarę zbliżania się gradientu do zera. To umożliwia bardziej precyzyjne zbliżanie się algorytmu do optymalnego rozwiązania, eliminując ewentualne "przeskakiwanie" ponad nim.

Aby model mógł skutecznie się uczyć, dane są dzielone na zbiór treningowy, który służy do trenowania modelu oraz zbiór walidacyjny służący do sprawdzania jakości modelu. Taki podział zapewnia skuteczność modelu przy wprowadzaniu nowych danych.

Struktura takich zbiorów wygląda następująco:

$$X \in \{(x^{(i)}, y^{(i)})\} \tag{3.3.2}$$

- $x^{(i)} \in \mathbb{R}^n$- wektor wejściowy zawierający dane, na podstawie których model wykonuję predykcję
- $y^{(i)} \in \mathbb{R}^n$- wektor wyjściowy
- $i = 1, \ldots, m$ – indeks obserwacji, gdzie m oznacza ich ilość danych z etykietami

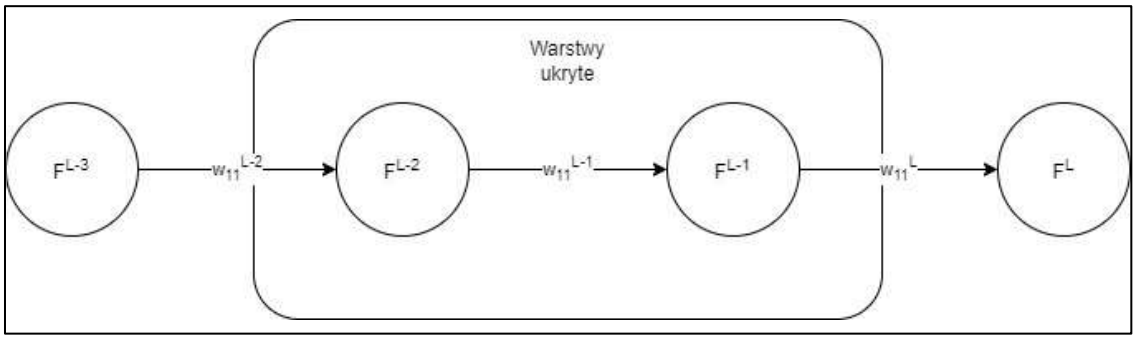
Algorytm gradientu prostego wykorzystywany we wstecznej propagacji korzysta z funkcji kosztu, która określa jak bardzo wynik sieci neuronowej jest błędny. Jest wiele różnych funkcji kosztu, a jedną z najpopularniejszych z nich jest suma błędów kwadratowych.

$$J = \sum_{i=1}^{n} (y_i - \overline{y_i})^2 \tag{3.3.3}$$

- i = 1…n – indeks danych dla którego model wyliczał predykcję
- $y_i = y^{(i)} \in X$ – rzeczywista wartość dla danych o indeksie i ze zbioru X
- $\overline{y}_i$ – wartość przewidziana przez model dla danych o indeksie i

Aby zastosować gradient prosty w kolejnych warstwach wchodzących w skład sieci neuronowej stosujemy regułę łańcuchową Leibniza, aby wyznaczyć pochodną funkcji kosztu (za zmienną zawsze przyjmujemy wagę lub stałą, której korektę wyznaczamy).

Poniżej znajduje się schemat prostej sieci neuronowej na której zastosujemy wsteczną propagację.



Rys. 3.3.2 Prosta sieć neuronowa,
źródło: Własne

Dla uproszczenia, pominiemy na ten moment stałe zawarte w neuronach. Możemy spojrzeć na sieć neuronową jako:

$$g(x) = f^L\left(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 x)\cdots)\right) \tag{3.3.4}$$

- f^i - wyjście neuronów z warstwy i,
- W^i - waga połączeń neuronów warstwy i-1 do i

Algorytm wstecznej propagacji korzysta z algorytmu gradientu prostego idąc w kierunku przeciwnym od kierunku działania sieci.

Niech funkcja aktywacji będzie liniowa ($f(x) = x$), a stała w neuronach wynosi zero. Wtedy wyjście neuronu możemy zapisać w postaci:

$$f^L(s^L) = f^L(w^L f^{L-1}) = w^L f^{L-1} \tag{3.3.5}$$

Za funkcje kosztu przyjmiemy sumę błędów kwadratowych:

$$J = \sum_{i=1}^{n} (y_i - g(x_i))^2 \tag{3.3.6}$$

- $i = 1, \cdots, n$ – indeks pary danych i ich obserwacji ze zbioru uczącego
- x_i – dane wejściowe o indeksie i ze zbioru uczącego
- y_i – obserwacja dla danych wejściowych o indeksie i ze zbioru uczącego
- $g(x_i)$ – predykcja wykonana przez sieć dla danych o indeksie i

Jeśli chcemy obliczyć nową wagę w_{11}^{L-1} musimy wyznaczyć pochodną funkcji kosztu od tej wagi.

$$\frac{\partial J}{\partial w^{L-1}} = \frac{\partial J}{\partial f^L}\frac{\partial f^L}{\partial s^L}\frac{\partial s^L}{\partial F^{L-1}}\frac{\partial f^{L-1}}{\partial s^{L-1}}\frac{\partial s^{L-1}}{\partial w^{L-1}} \tag{3.3.7}$$

Przy większej ilości neuronów w warstwie korzystamy z indexów dolnych do oznaczania poszczególnych neuronów.

$\frac{\partial J}{\partial f^L}$	$\sum_{i=1}^{n} -2(y^i - f^L)$ y^i – zaobserwowana wartość dla x^i
$\frac{\partial f^L}{\partial s^L}\frac{\partial f^{L-1}}{\partial s^{L-1}}$	1
$\frac{\partial s^L}{\partial f^{L-1}}$	w^{L-1}
$\frac{\partial s^{L-1}}{\partial w^{L-1}}$	f^{L-2}
$\frac{\partial J}{\partial w^{L-1}}$	$f^{L-2} * w^{L-1} * \sum_{i=1}^{n} -2(y^i - f^L)$

Tabela 3.3.1: Wartości pochodnej funkcji kosztu od wagi dla w^{L-1}

Gdybyśmy chcieli w kolejnym kroku poprawić wagę połączenia w warstwie L-2 to traktujemy w^{L-1} jako stałą:

$$\frac{\partial s^{L-1}}{\partial w^{L-1}} = \frac{\partial s^{L-1}}{\partial F^{L-2}}\frac{\partial F^{L-2}}{\partial s^{L-2}}\frac{\partial s^{L-2}}{\partial w^{L-2}} \tag{3.3.8}$$

Obliczenia w bardziej zaawansowanych sieciach neuronowych są wykonywane warstwami (poprzez obliczenia na wektorach).

Podsumowanie

Dzięki wykorzystywaniu metody łańcuchowej leibniza, gradientu prostego i pochodnych, algorytm wstecznej propagacji może propagować błąd wstecz sieci, poprawiając wagi i wyrazy wolne. W ten sposób sieci neuronowe przystosowują się do danych i są wstanie dokonywać skutecznych predykcji.

4 TRANSFORMERY

Transformery są nowym podejściem do problemu przetwarzania języka naturalnego zaproponowanym przez google w pracy naukowej „Attention is all you need” [2]. Zauważono, że ówczesne rozwiązania są zbyt skomplikowane i nie nadają się do skalowania na duże zbiory tekstów. Za jego sukcesem stoją 2 technologie:

Mechanizm Multi-head attention

Mechanizm samo-uwagi (lub jego implementacja Multi-head attention używana w transformerach) okazał się wystarczający do przedstawienia powiązań w języku, co pozwoliło znacznie uprościć model i za tym idzie wyróżnić go na tle pozostałych rozwiązań. Załóżmy, że w zdaniu występuje słowo „brzeg”. Zależnie od kontekstu może oznaczać wiele rzeczy (brzeg plaży, brzeg jakiegoś obiektu…), dlatego stosujemy mechanizm samo-uwagi aby szukał powiązań w kontekście co pozwala na lepsze zrozumienie i reprezentacje zdania.

Uczenie transferowe

Drugą rewolucyjną mechaniką jest uczenie transferowane. Możemy transferować modele pomiędzy zadaniami co znacznie skraca czas potrzebny na naukę. Bazowy model w pracy naukowej od google’a był trenowany na 40 GB zbiorze danych do najprostszych zadań, aby następnie zostać dotrenowanym do wielu innych zadań.

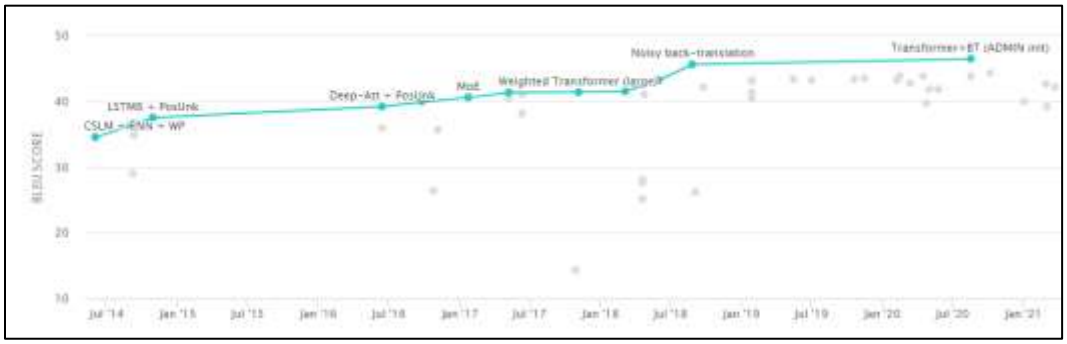
Wyniki

Zbiory WMT („Workshop on Machine Translation”) to zestawy danych zawierające tłumaczenia pomiędzy dwoma językami. Są używane w badaniach i konkursach w dziedzinie przetwarzania języka naturalnego.

Do oceny translacji maszynowej używamy punktacji BLEU. Zawiera się ona w skali od 0 do 100, gdzie im wyższe wartości tym skuteczniejszy model. Porównujemy tłumaczenie maszynowe z jednym lub więcej tłumaczeniami referencyjnymi, oceniając na ile tłumaczenie maszynowe jest podobne do tych referencyjnych pod względem trafności i precyzji.

Model został sprawdzony przy użyciu WMT 2014 English-to-German na których zdobył 28.4 punkty BLEU przebijając najlepszy aktualny wynik o 2 punkty. Na zadaniach WMT 2014 English-to-French model zyskał zadziwiające 41.8 punktów po zaledwie 3.5 dniach treningu na 8 kartach graficznych. Te wyniki pokazują jak ważne w dziedzinie NLP stały się transformery. Poniżej znajduje się aktualny wykres wyników uzyskanych w punktacji BLEU na zbiorze WMT 2014 English-French różnych modeli NLP.

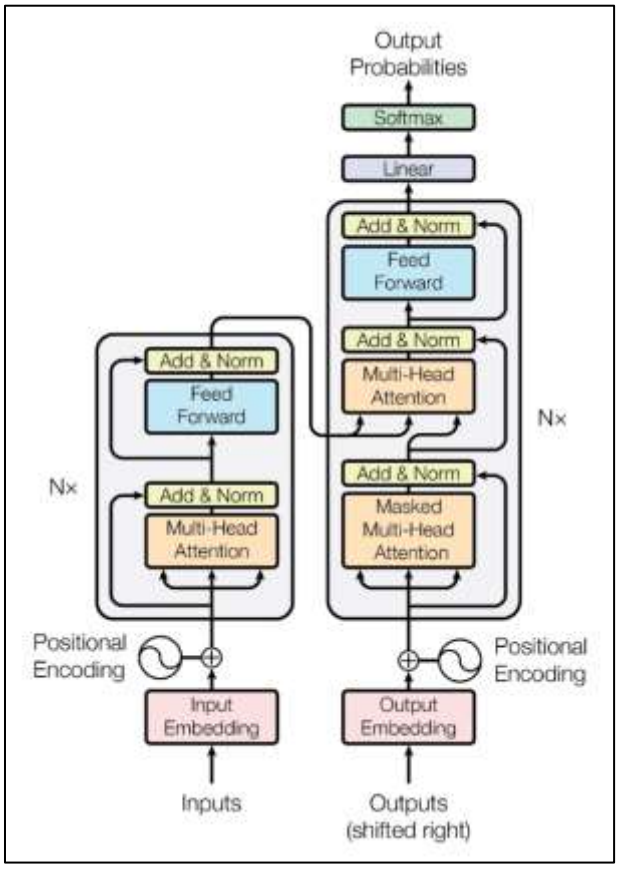
[2] Praca naukowa „Attention is all you need” - https://arxiv.org/pdf/1706.03762.pdf



Rys. 4.1: Wykres Najlepszych wyników BLEU dla WMT2014 English-French, źródło:https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-french

4.1 ARCHITEKTURA

Architekturę transformera możemy podzielić na enkodery (po lewej) i dekodery (po prawej). Taka budowa pozwala na rozdzielenie obowiązków w modelu i zapewnia lepsze wyniki.

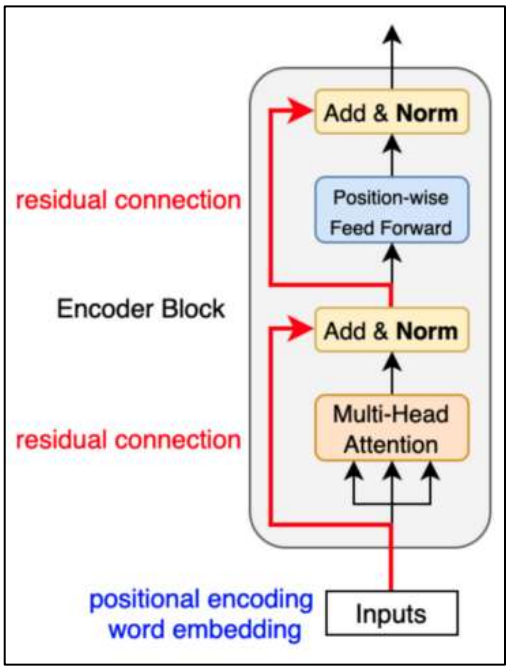


Rys. 4.1.1: Architektura transformera źródło: https://arxiv.org/pdf/1706.03762.pdf

Transformator za pomocą mechanizmu *multi-head attention* oraz wewnętrznych sieci neuronowych "uczą się" powiązań oraz reguł w języku naturalnym. Model zna pozycję tokenu w zdaniu oraz oblicza jego powiązania z innymi tokenami. Dzięki swojej architekturze są przygotowane do pracy wielowątkowej i są w stanie obsługiwać wielkie zasoby tekstu.

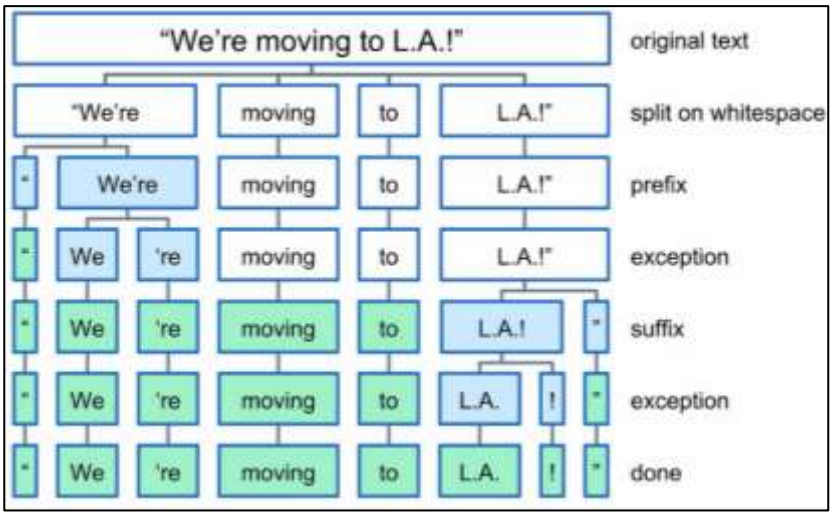
4.1.1 Enkoder

Bloki enkodera służą w transformerze do przyjęcia wejścia modelu. Znane modele takie jak bert korzystają z architektury enkodera do rozwiązywania wielu zadań NLP. W tranformerze odpowiadają za przygotowanie reprezentacji zdania i zawartych w nim tokenów do obsługi przez dekoder.



Rys. 4.1.1.1: Architektura Enkodera
źródło: https://kikaben.com/transformers-

Przed wprowadzeniem zdania do algorytmu musi ono zostać podzielone na tokeny. Są to krótkie zbiory znaków na których będzie pracować transformer. Istnieje wiele metod podziału zdania na tokeny. Poniżej znajduje się jeden z nich.



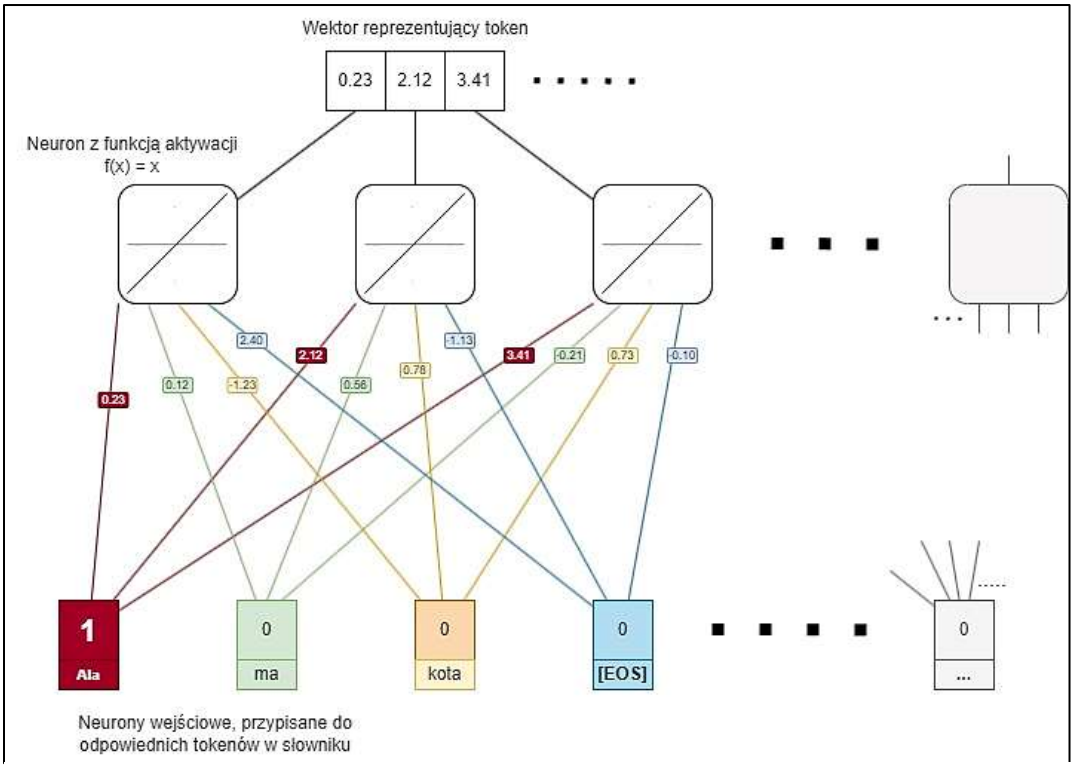
Rys. 4.1.1.2 Podział na tokeny
źródło: https://medium.com/@almassco2007/basics-of-nlp-part-1-tokenazation-d124c15a01a8

Po uzyskaniu zdania w formie tokenów musimy wyliczyć ich reprezentacje liczbową co pozwoli siecią neuronowym zawartym w enkoderze wykonywać z nimi działania.

Tego typu zamiany nazywamy word embedding, a najpopularniejszym algorytmem wykonującym je jest word2vec.

4.1.1.a *Word2Vec*

Algorytm word2vec architekturą przypomina najprostszą sieć neuronową. Dzięki swojej prostocie jesteśmy w stanie skalować tą sieć do wielkich słowników oraz możemy bez większych przeszkód zwiększać rozmiar wektora wyjściowego co pozytywnie wpływa na jego reprezentacje oraz skuteczność całego modelu.

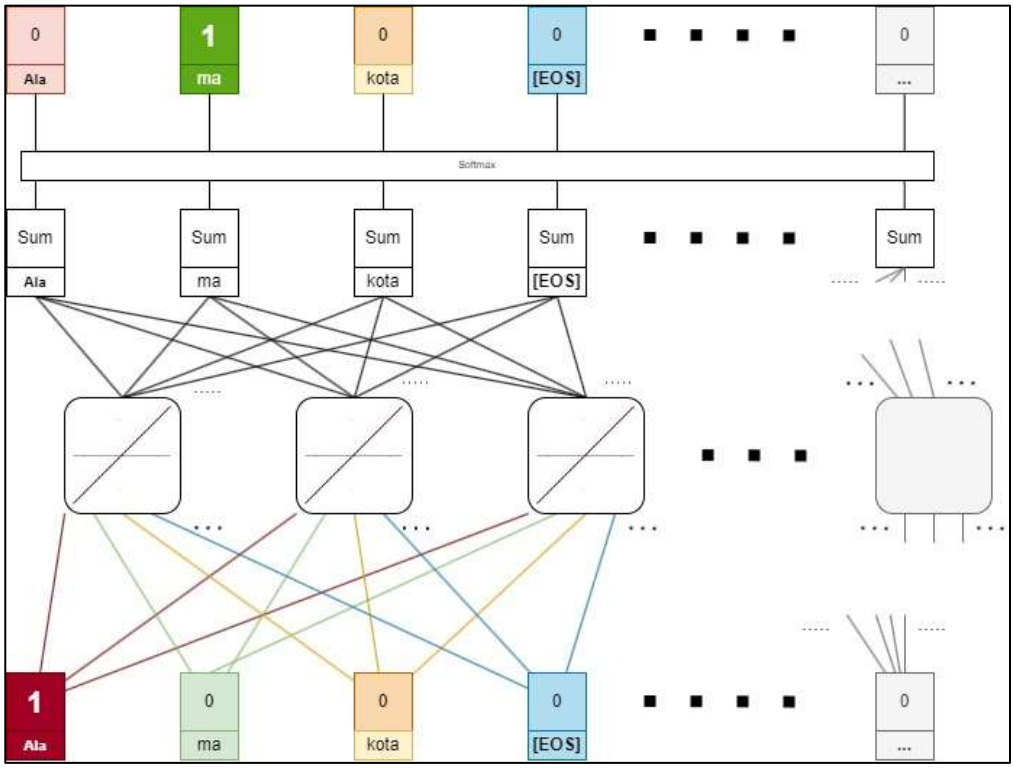


Rys. 4.1.1.a.1: Architektura sieci word2vec
źródło: Własne

Ilość neuronów w warstwie wyjścia odpowiada wielkości wektora, za pomocą którego chcemy reprezentować dany token. Na wejściu sieci podajemy sygnał 1.0 tylko dla neuronu odpowiadającego tokenowi dla którego liczymy reprezentujący wektor.

Warto podkreślić, że cała sieć algorytmu word2vec jest podłączona i trenowana na całości słownika na jakim był trenowany model. Na wejściu enkodera obliczamy tylko potrzebne tokeny (tokeny zdania wejściowego).

W celu wytrenowania takiej sieci neuronowej dodajemy kolejną warstwę i za pomocą funkcji softmax (lub innej funkcji wspierającej klasyfikację) przewidujemy kolejne słowo w zdaniu. Na podstawie tych prognoz trenujemy algorytm wsteczną propagacją.



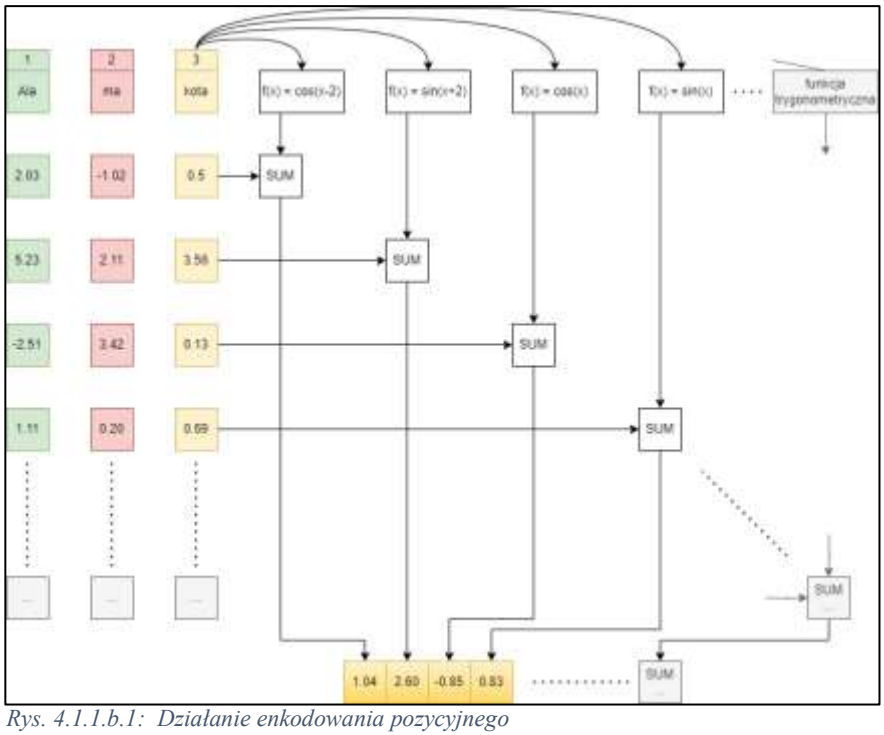
Rys. 4.1.1.a.2 Architektura trenowanej sieci word2vec
źródło: Własne

Większość zbiorów tokenów na którym się trenuje algorytm, stosuje specjalne tokeny rozpoczynające [BOS] lub kończące zdanie [EOS] (od *End of Sentence i Begining of sentence*). Po wykonaniu algorytmu wysyłamy wektory reprezentujące tokeny do algorytmu enkodowania pozycyjnego.

4.1.1.b *Enkodowanie pozycyjne*

W celu wprowadzenia informacji o pozycji tokenu w zdaniu korzystamy z enkodowania pozycyjnego. Istnieje wiele sposobów na jego wykonanie, ale najpopularniejszym z nich jest wykorzystanie funkcji trygonometrycznych.

Dla otrzymanego wektora o długości n wybieramy n funkcji (najczęściej trygonometrycznych). Następnie do każdej z tych funkcji przesyłamy pozycję tokenu w zdaniu i dodajemy wynik do odpowiedniego elementu w wektorze.



Rys. 4.1.1.b.1: Działanie enkodowania pozycyjnego
źródło: Własne

Załóżmy, że chcemy obliczyć wektor z uwzględnieniem pozycji dla tokenu “ma” w zdaniu “Michał ma kota.”. Pominiemy przy tokenizacji białe znaki, a za tokeny przyjmiemy kolejne słowa w zdaniu. Po algorytmie word2vec dla tokenu “ma“ otrzymujemy wektor:

$$[0.13, -1.24, 0.38, 1.20]$$

Widzimy, że ustalona długość wektorów reprezentujących tokeny wynosi 4. Dla każdego elementu wektora przypisujemy funkcje sigmoidalną.

- Dla pierwszego elementu (0.13): $f(x) = \sin(x)$
- Dla drugiego elementu (-1.24):$f(x) = \cos(x)$
- Dla trzeciego elementu (0.38):$f(x) = \sin(x - 3)$
- Dla czwartego elementu (1.20):$f(x) = \cos(x - 3)$

Przesyłamy pozycję “ma“ do funkcji oraz sumujemy je z odpowiadającymi im liczbami. W tym przypadku „ma” posiada 2 pozycję.

$$[0.13 + \mathbf{sin(2)}, -1.24 + \mathbf{cos(2)}, 0.38 + \mathbf{sin(2 - 3)}, 1.20 + \mathbf{cos(2 - 3)}]$$

Otrzymujemy:

$$[0.16, 0.24, 0.36, 2.19]$$

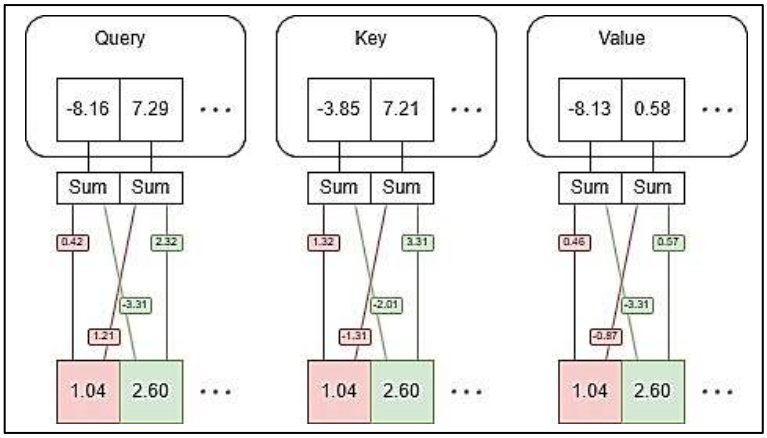
Powyższy wektor przechowuje informacje jaki token w sobie posiada oraz jaką pozycję miał w zdaniu wejściowym. Następnym krokiem jest wprowadzenie informacji o powiązaniach między tokenami stosując mechanizm multi-head attention.

4.1.1.c *Multi-head attetion*

Celem mechanizmów Multi-head attetion jest rejestracja powiązań między tokenami w zdaniu. Załóżmy, że mamy zdanie „Podczas spotkania z Janem okazało się, że jego wyliczenia były błędne.”. Słowo „jego” jest powiązane (odnosi się) do „Janka” zatem w nowej reprezentacji chcemy zarejestrować to powiązanie tak aby słowo „Janka” miało duży wpływ na słowo „jego”. W tym celu dla każdego tokenu wyliczamy 3 wektory:

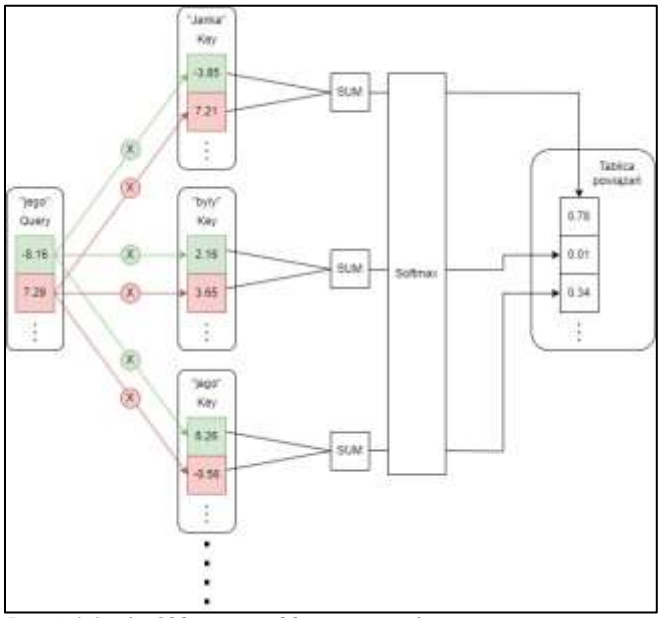
- **Query** oraz **key** – służą do wyliczania poziomu powiązania między tokenami
- **Value** – za pomocą tego wektora wpływamy na reprezentacje tokenów

Model wyliczający te wektory jest siecią neuronową.



Rys. 4.1.1.c.1: Sieci do obliczania Query, Key oraz Value
źródło: Własne

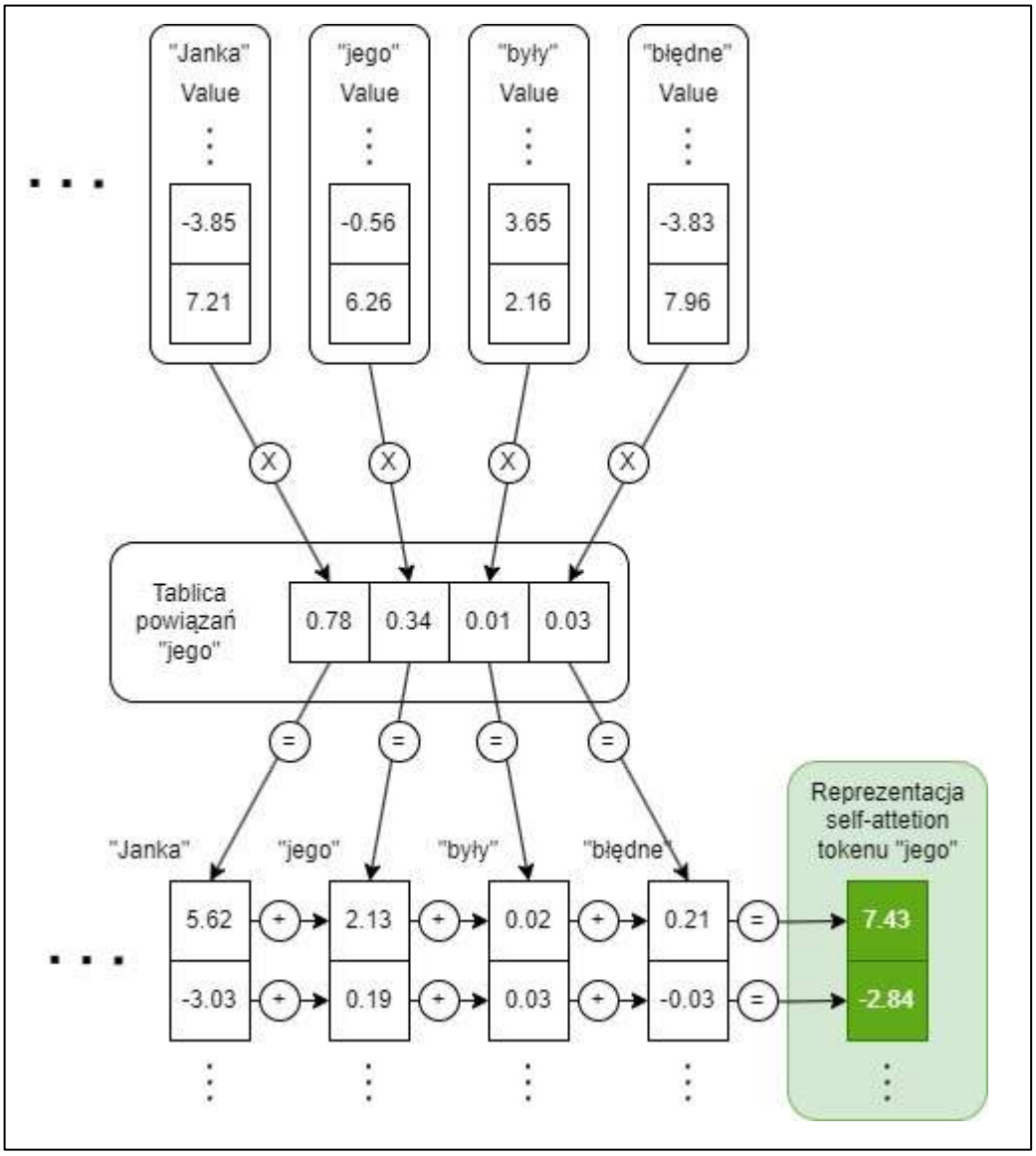
Dla całego słownika używamy tych samych sieci i wag. Następnym krokiem jest policzenie dla każdego tokenu tablicy w której będziemy przechowywać siłę powiązań ze wszystkimi tokenami na wejściu.



Rys. 4.1.1.c.2: Obliczanie tablicy powiązań
źródło: Własne

W tym celu bierzemy wektor Query tokenu dla którego liczymy tablicę i mnożymy jego elementy przez elementy wektora Key tokenu z którym powiązanie chcemy wyliczyć. Obliczamy w ten sposób kolejne poziomy powiązań z kolejnymi tokenami zdania.

Ostatnim krokiem jest przemnożenie wszystkich wektorów **Value** przez odpowiednie elementy w tablicy powiązań oraz zsumowanie ich do jednego wektora.



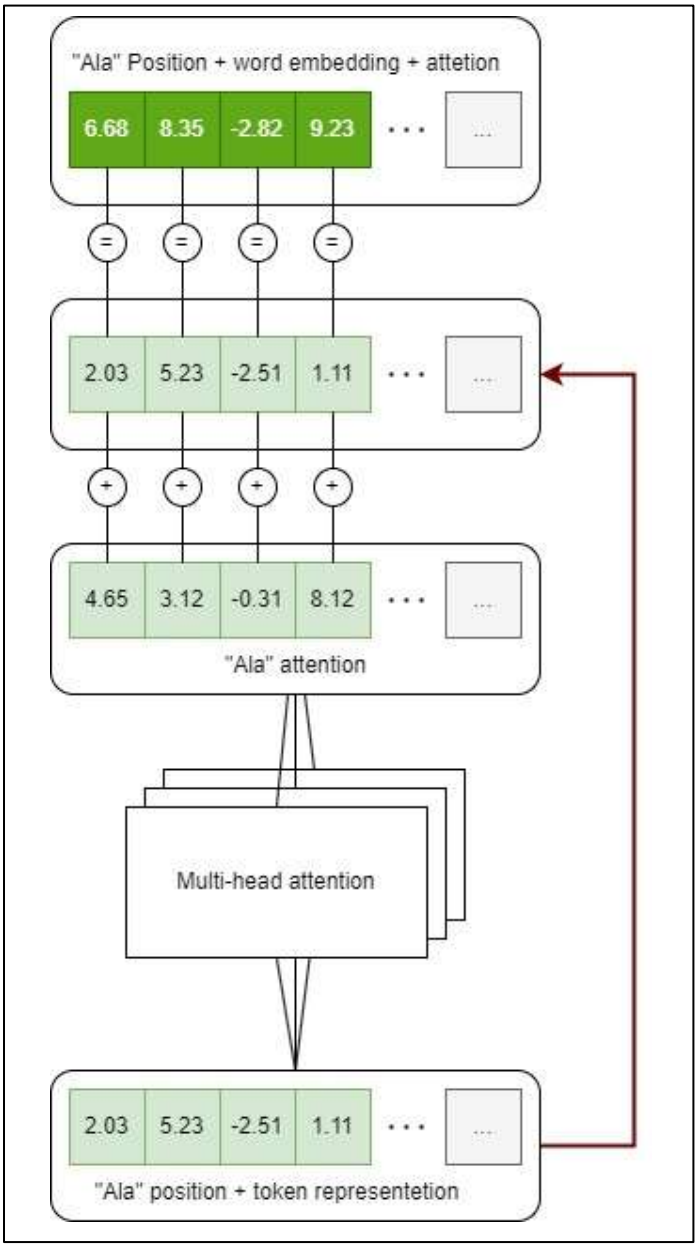
*Rys. 4.1.1.c.3: Wykorzystanie tablicy powiązań oraz wektorów **value***
źródło: Własne

Dzięki wykorzystaniu sieci neuronowej przy tworzeniu wektorów oraz zastosowania powyższego schematu, transformery są wstanie rejestrować powiązania pomiędzy tokenami. W zaawansowanych modelach występuje wiele bloków mechanizmu self-attention (stąd nazwa multi-head), a każdy z nich ma różne sieci i wagi dzięki czemu możemy wykrywać powiązania w wielkich słownikach.

W tym momencie enkoder posiada tylko wektor pochodzący z mechanizu multi-head attention, zatem informacje o tokenie i jego pozycji zostały zagubione. Żeby temu zaradzić stosujemy połączenie „resztkowe”.

4.1.1.d *Połączenia Resztkowe*

Dzięki połączeniom resztkowym nie tracimy informacji obrabianych w mechanizmach transformera. Połączenia resztkowe są stosowane w całej strukturze, co poprawia działanie modelu.



Rys. 4.1.1.d.1: Połączenie resztkowe przy mechaniźmie attetion
źródło: Własne

Przykładowo przy mechanizmie multi-head attention sumujemy zbiór wektorów pochodzących z algorytmu word2vec oraz enkodowania pozycyjnego z wektorem wynikowym działania mechanizmu. W ten sposób transformer jest wstanie zapamiętywać

wyniki działania wszystkich algorytmów wchodzących w jego skład. Na wyniku sumowania stosujemy normalizację liniową.

4.1.1.e *Sieć neuronowa w enkoderze*

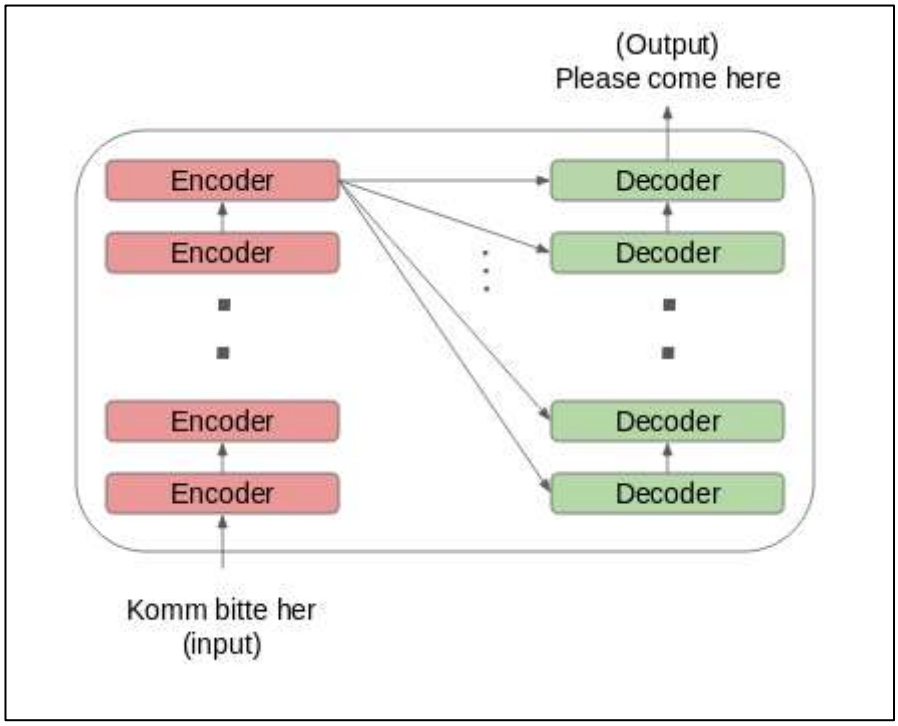
Na samym końcu działania bloku enkodera znajduje się prosta 3 warstwowa sieć neuronowa z funkcją aktywacji Relu.

$$Y = ReLU(XW_1 + b_1)W_2 + b_2 \tag{4.1.1.e.1}$$

- Y – wektor wyjściowy enkodera
- X - wektor wejściowy, jest on wynikiem działania bloku enkodera. Zawiera word embedding, enkodowanie pozycyjne oraz reprezetnacje mechanizmu attention
- W_i- Wektor wag połączeń w warstwie i
- b_i- wektor stałych w neuranach wartwy i

Tak jak przy mechanizmie Multi-head attetion, korzystamy z połączenia „resztkowego", aby zapobiec utracie wcześniejszych reprezentacji (dodajemy wynik sprzed działania sieci do wyniku jej działania). Finalny wynik jest przesyłany do dekodera.

Warto spojrzeć na architekturę transformera w przypadku większej ilości bloków enkodera i dekodera.

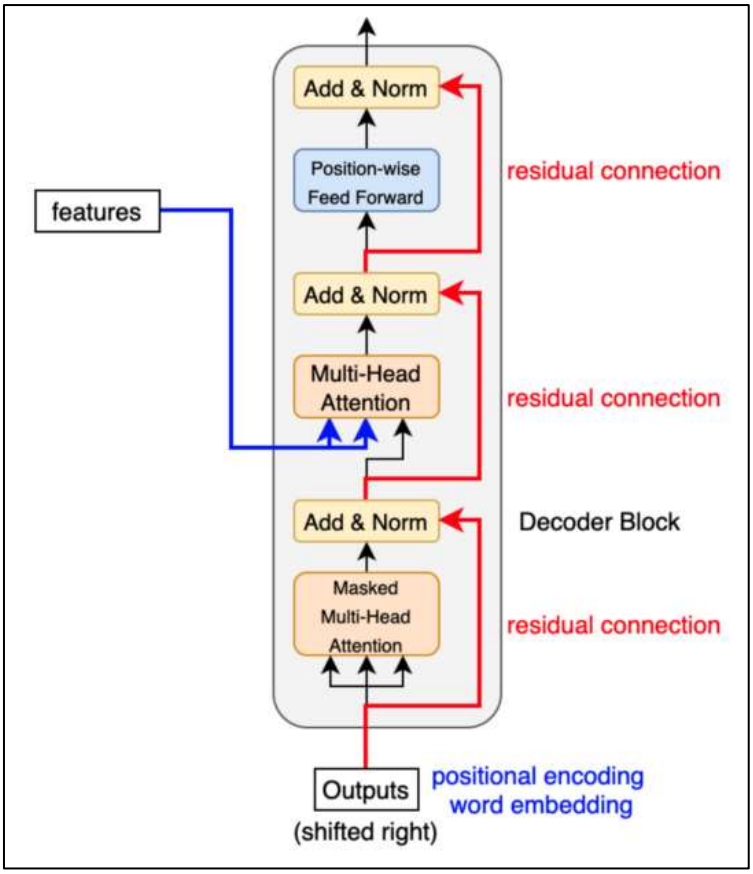


Rys. 4.1.1.e.1: Architektura transformera
źródło: Własne

Wejście przechodzi przez kolejne bloki enkodera w celu uzyskania jak najlepszej reprezentacji która jest wykorzystywana przez dekodery w ich wewnętrznych specjalnych mechanizmach Multi-head attention.

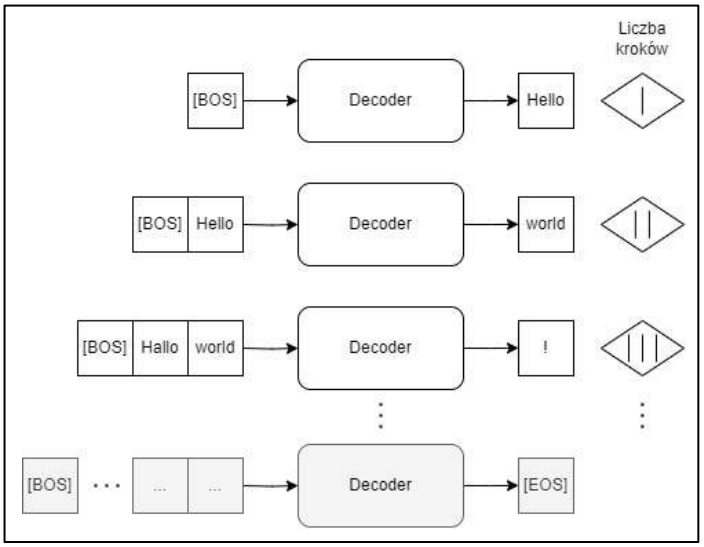
4.1.2 Dekodery

Dekodery korzystają z mechanizmów multi-head attetion z dodatkami, kodowania pozycyjnego oraz algorytmów typu word2vec, aby na podstawie danych z enkoderów oraz swoich własnych generacji wyliczać jak najlepsze wyniki.



Rys. 4.1.2.1: Architektura dekodera
źródło: https://kikaben.com/transformers-encoder-decoder/

Na wejściu dekoder otrzymuje wektor z wyjściem wygenerowanym w poprzednim kroku przesuniętym w prawo.



Rys. 4.1.2.2: Wejście dekodera
źródło: Własne

4.1.2.a *Masked Multi-head attention*

Wejście jest następnie przesyłane do mechanizmu Masked Multi-head attention. Różnicą od zwykłego mechanizmu (Multi-head attention) jest maskowanie kolejnych tokenów od tokenu dla którego liczymy powiązania. Zamaskowane tokeny nie są brane pod uwagę w wyliczeniach.

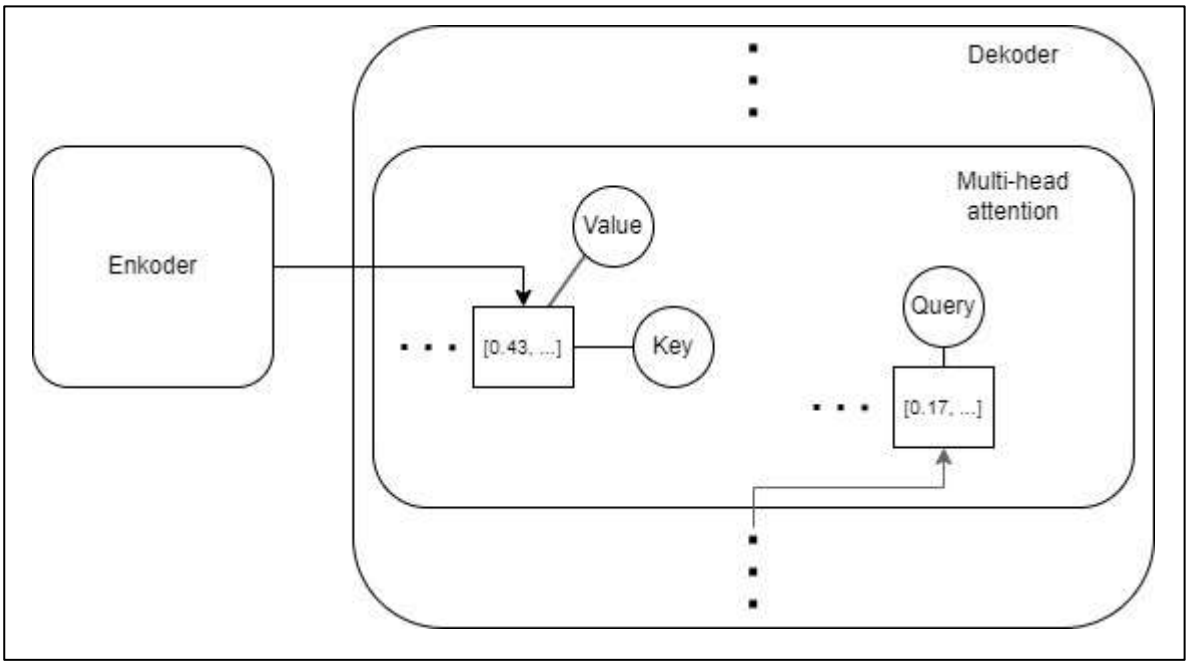
Kroki	**Wejście**				
I	[BOS]	Witaj	świecie	!	[EOS]
II	[BOS]	Witaj	świecie	!	[EOS]
III	[BOS]	Witaj	świecie	!	[EOS]
IV	[BOS]	Witaj	świecie	!	[EOS]
V	[BOS]	Witaj	świecie	!	[EOS]

Tabela 4.1.2.a.1: Działanie maskowania

Na szaro zostały oznaczone tokeny które nie były brane pod uwagę przy obliczaniu tablicy powiązań, a na niebiesko token którego powiązania wyliczamy. Na samym końcu stosujemy połączenie resztkowe, aby zachować informację sprzed działania mechanizmu (word embedding i enkodowanie pozycyjne).

4.1.2.b *Multi-head attention*

W kolejnym kroku stosujemy zwykły mechanizm Multi-head attention. W przypadku dekodera dla wejściowych danych, obliczamy powiązania wyłącznie z danym wyjściowymi enkodera.



Rys. 4.1.2.b.1: Połączenie enkodera z dekoderem
źródło: Własne

Dla wejścia dekodera obliczamy wektor **Query**, a dla wyjścia enkodera **Key** oraz **Value**. Następnymi krokami tak jak w zwyczajnym mechanizmie Mulit-head attention obliczamy tablicę powiązać wejścia dekodera z wyjściem enkodera. Po działaniu tego

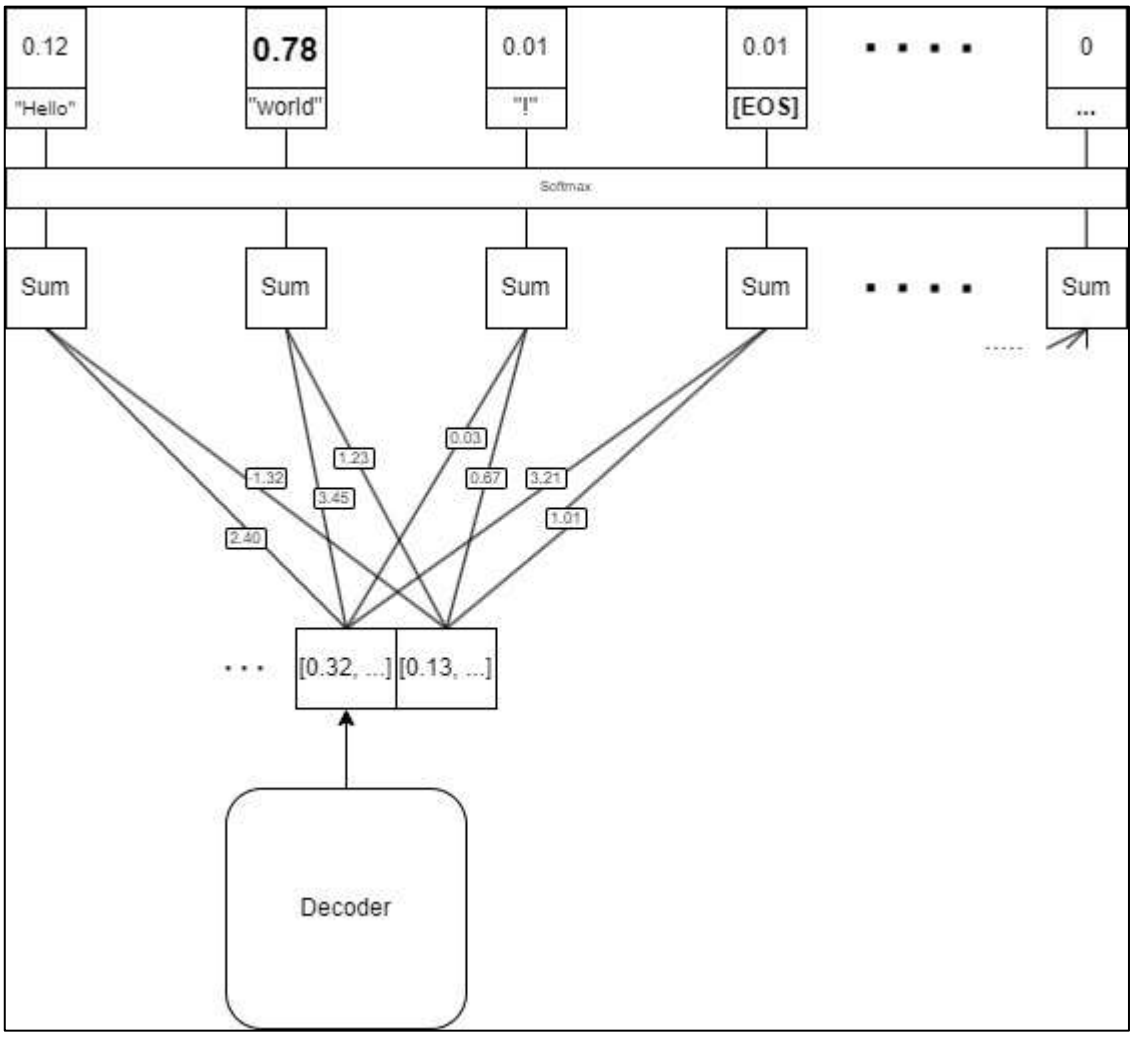
mechanizmu również stosujemy połączenie resztkowe, we celu zachowania informacji wejściowych.

4.1.2.c *Sieć neuronowa w dekoderze*

Na samym końcu działania bloku dekodera znajduje się prosta 3 warstwowa sieć neuronowa z funkcją aktywacji Relu, o takiej samej architekturze jak w enkoderze. Tutaj również stosujemy połączenie resztkowe.

4.1.3 Wyjście z transformera

Po wyjściu z dekodera dane są wysyłane do prostej sieci neuronowej, której zadaniem jest obliczyć prawdopodobieństwa dla tokenów w słowniku.



Rys. 4.1.3.1: Wyjście z transformatora
źródło: Własne

Dekoder będzie wykonywał kolejne iteracje do momentu gdy wygeneruje token kończący zdanie [EOS] lub nie zostanie spełniony warunek ustalany przez użytkownika.

5 PORÓWNANIE MODELI OPARTYCH NA TRANSFORMERZE

Po zaproponowaniu przez google'a modelu transformera, wiele programistów i naukowców tworzyło jego alteracje dzięki czemu powstało wiele nowych algorytmów i modeli NLP. Ogólna jakość wyników w dziedzinie NLP się podniosła, a branże spoza NLP zaczęły wprowadzać rozwiązania i modele NLP do swoich struktur. Nie ograniczano się tylko do korzystania z samej bazowej struktury transformera, ale rozbierano go na pomniejsze części w celu tworzenia prostszych i skuteczniejszych modeli. W tym rozdziale spojrzymy na 2 najpopularniejsze modele, które powstały na bazie architektury transformera.

BERT

Pierwszym z nich jest Bert (Bidirectional Encoder Representations from Transformers) stworzony przez google, którego architektura skupia się na blokach enkodera. Bert może korzystać z informacji kontekstowej całego tekstu dzięki czemu jest skuteczny przy zadaniach w których potrzebna jest znajomość całego tekstu np. sumaryzacja. Model bazowy jest wytrenowany na zadaniach znajdowania brakujących słów w zdaniu oraz sprawdzania czy dwa podane zdania następują po sobie (czy zdanie b jest kolene po zdaniu a). Bert otworzył drogę programistom i naukowcom do tworzenia innych modeli opartych na transformerze. Jednym z nich jest GPT.

GPT

GPT (Generative Pre-trained Transformer) zaproponowany przez firmę openAI zajmującą się sztuczną inteligencją w odróżnieniu od bert'a nie posiada informacji o kontekście całego tekstu. Jego architektura polega na blokach dekodera które maskują część tekstu, co zwiększa skuteczność przy generatywnych zadaniach. Model bazowy był trenowany na ogromnych zbiorach tekstu pochodzących z Internetu, gdzie jego zadaniem była prosta genracja tekstu. Model ten stoi za popularnym agentem konwersacyjnym Chat-gpt, który jest w stanie prowadzić konwersacje na wiele tematów.

Porównanie

Bert został stworzony z myślą tworzenia modelu skupionego na kontekście zdania. Rozumie tekst oraz jego tokeny lepiej od GPT-2, ale za cenę zdolności do generacji dużych tekstuów. Gpt-2 został natomiast tworzony w celu jak najlepszej generacji tekstu. Skupia się na powiązaniach wejścia z tym co do tej pory wygenerował dzięki czemu tworzony kontekst jest budowany, tak jak w języku naturalnym. Patrząc na wyniki obu modeli możemy zauważyć że gpt2 wykonuje lepsze generacje długich tekstów. Jest to spowodowaną różnicą w architekturze oraz podejściu do obu modeli.

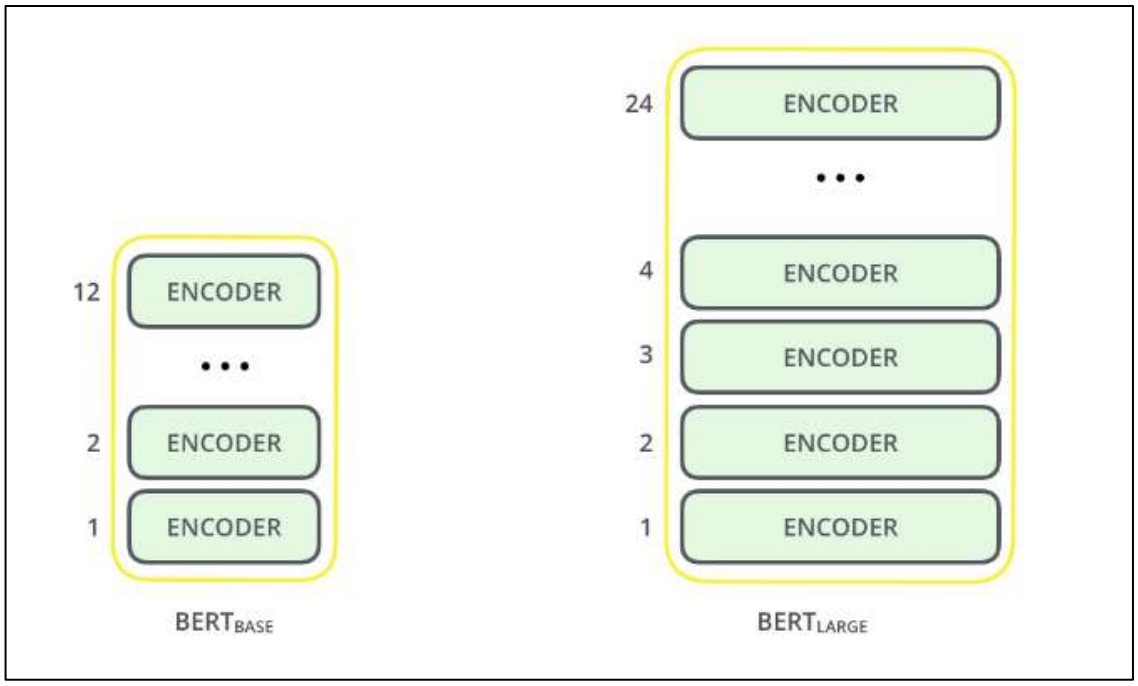
5.1 Architektura Bert'a

Architektura bert'a skupia się na blokach enkodera. Korzystają one z całego tekstu przy obliczaniu powiązań co pozwala na zrozumienie całego kontekstu zdania.

Tak jak przy transormerze, na wejściu bert otrzymuje zbiór wektorów reprezentujący zdanie wejściowe. Samą reprezentacje tworzymy za pomocą algorytmów word2vec oraz enkdowania pozycyjnego.

Enkodery w modelu bert korzystają z wielu bloków mechanizmu multi-head attention. Aby zsumować wektory wielu bloków działania tego mechanizmu, używamy prostej sieci neuronowej, której zadaniem jest jak najlepsze dopasowanie wyników do wejścia kolejnej części enkodera (sieci neuronowej z podrozdziału 4.1.1.e).

Na wyjściu berta znajduje się identyczna sieć neuronowa jaką widzieliśmy na końcu działania transformera (4.1.3). Na podstawie reprezentacji wygenerowanej przez enkodery, wyliczamy prawdopodobieństwo dla każdego tokena w słowniku.



Rys. 5.1.1: Modele bert base (po lewej) i bert large (po prawej)
źródło: http://jalammar.github.io/illustrated-bert/

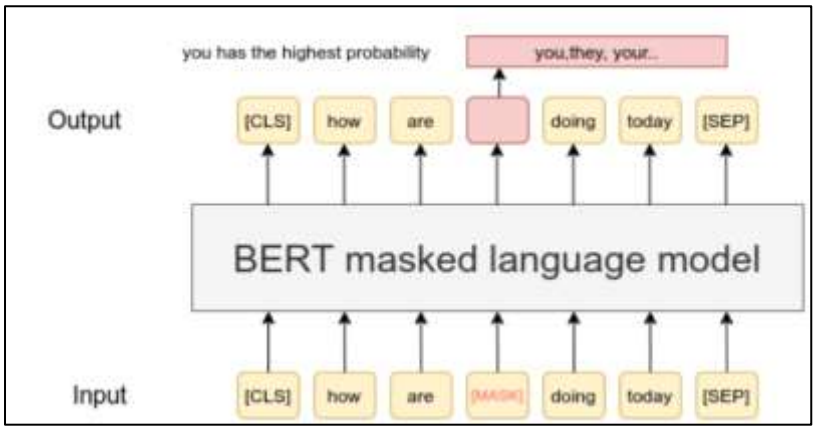
5.1.1 Modele

Zaproponowano 2 bazowe modele :

- Bert base (po lewej) posiada 12 bloków encodera nałożonych na siebie, a każdy z nich posiadał 12 bloków mechanizmu multi-head attention co dawało 110 milionów parametrów (połączeń oraz wag)
- Bert Large (po prawej) posiada 24 bloków encodera nałożonych na siebie, a każdy z nich posiadał 16 bloków mechanizmu multi-head attention co dawało 340 milionów parametrów

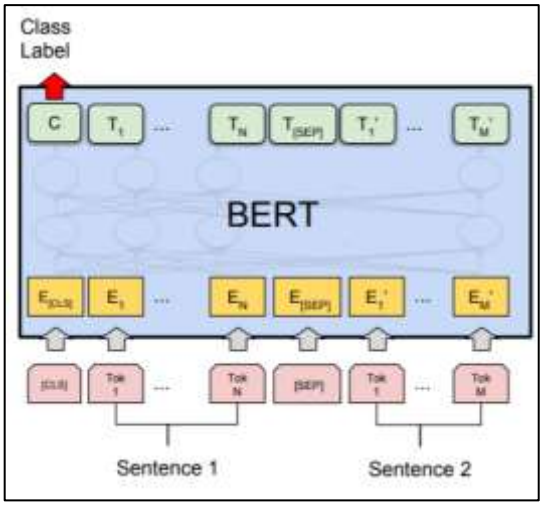
5.1.2 Trening

Trening bert'a skupiał się na dwóch zadaniach wykorzystujących kontekst całego zdania. Pierwszym z nich jest zadania MLM (Masked Language Modeling) polegające na wypełnieniu zamaskowanych słów w zdaniu.



Rys. 5.1.2.1: Zadanie Masked language modeling
źródło: https://quantpedia.com/bert-model-bidirectional-encoder-representations-from-transformers/

Dzięki temu zadaniu bert nauczył się powiązań między tokenami. Aby ochronić model przed nadmiernym skupianiem się nad pozycją zamaskowanego tokenu, naukowcy losowo maskowali ok. 15 % tokenów. Drugim zadaniem jest NSP (Next Sentence Prediction). Zadaniem modelu jest stwierdzenie czy dwa zdania podane sąsiadują ze sobą (są powiązane).



Rys. 5.1.2.2: Zadanie Next Sentence Prediction
źródło: https://arxiv.org/pdf/1810.04805v2.pdf

Na wejściu model otrzymuje 2 zdania oddzielone tokenem separacyjnym. Zadaniem modelu jest stwierdzenie czy zdanie 2 następuje po zdaniu 1. Zakładając zbiór tekstu posiadający 1000 zdań, model posiadałby 500 przykładów uczących. W tym zadaniu model zwraca nie tylko nowe reprezentacje zdań, ale też token klasy w którym przechowujemy wynik dla zadania NSP. Warto zauważyć, że na rysunku 5.1.2.1 również występuje token klasy [CLS] ponieważ oba te zadania były trenowane na tym samym modelu.

5.2 ARCHITEKTURA GPT-2

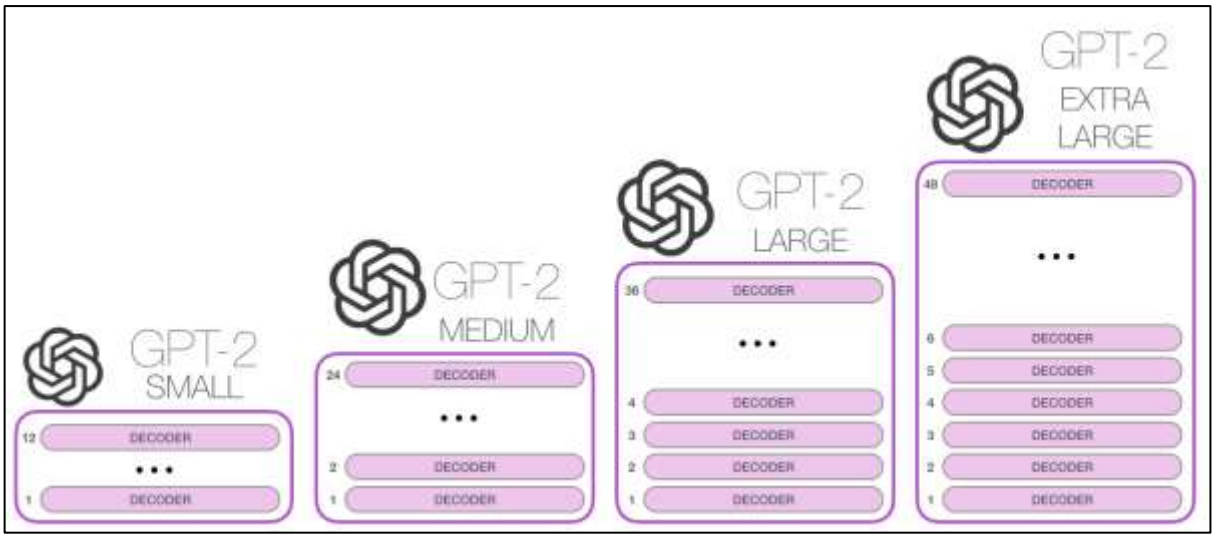
Model gpt-2 skupia się na dekoderze który skutecznie wykonuje zadania generacji tekstu.

Na wejściu gpt-2 otrzymuje zbiór wektorów. Jest on wynikiem działania algorytmów word2vec oraz enkdowania pozycyjnego na zdaniu wejściowym.

Z uwagi na brak bloków enkodera pozbyto się mechanizmu mutli-head attention odpowiadającego za powiązania z wynikiem działania enkoderów (4.1.2.b). Wykorzystujemy jedynie mechanizm Masked Multi-head attention oraz prostą sieć neuronową, aby wyliczać powiązania w zbiorze tokenów wejściowych oraz nowo wygenerowanych.

Po wyjściu z modelu, wektor reprezentujący trafia do sieć neuronowej, której zadaniem jest wyliczyć prawdopodobieństwa następnego tokenu ze słownika (4.1.2c).

5.2.1 Modele



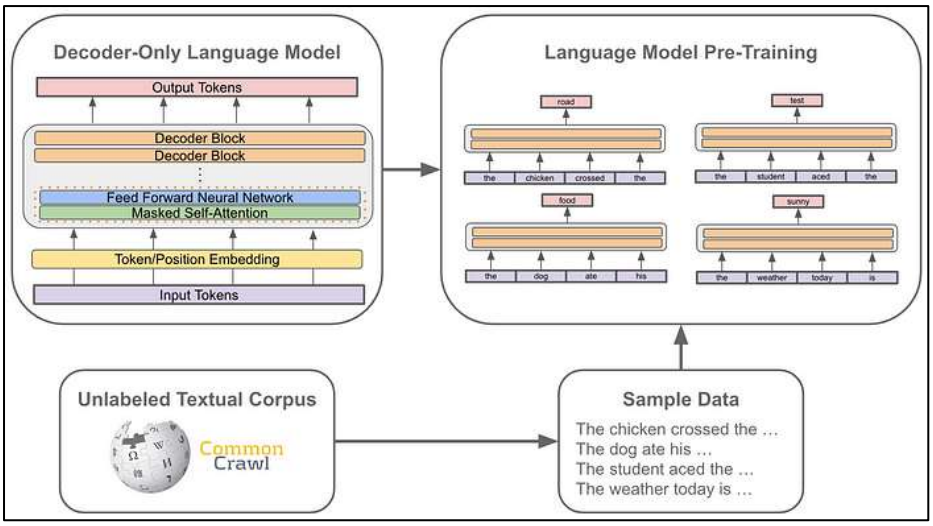
Rys. 5.1.2.1: Modele gpt2
źródło: http://jalammar.github.io/illustrated-gpt2/

Zaproponowano 4 modele:

- Gpt-2 Small posiada 12 bloków dekodera oraz 12 bloków mechanizmu Multi-head attention, co daje w sumie 117 milionów parametrów
- Gpt-2 Medium posiada 24 bloki dekodera oraz 16 bloków mechanizmu Multi-head attention co sumuje się na 345 milionów parametrów
- Gpt-2 Large posiada 36 bloków dekodera oraz 20 bloków mechanizmu Multi-head attention co sumuje się na 762 miliony parametrów
- Gpt-2 Extra large posiada 48 bloków dekodera oraz 24 bloki mechanizmu Multi-head attention co sumuje się na 1.542 miliarda parametrów

5.2.2 Trening

Model gpt2 był trenowany w sposób nienadzorowany metodą CLM (casual language modeling). Jego zadaniem było generowanie kolejnych słów w ogromnych zbiorach danych.



Rys. 5.2.2.1: Trenowanie modeli gpt2
źródło: https://towardsdatascience.com/language-model-training-and-inference-from-concept-to-code-483cf9b305ef

Dzięki takiemu podejściu, gpt2 (oraz inne modele z rodziny gpt) są wstanie bardzo skutecznie generować tekst przypominający język naturalny. Zbiory danych pochodziły między innymi z wikipedi, reddita czy twittera.

5.3 PODSUMOWANIE

Modele Gpt-2 oraz Bert zostały stworzone do odmiennych celów. Gpt-2 został stworzony do generacji dużych tekstów, a Bert do pracy nad tekstem. Oba korzystają z rozwiązań zawartych w transformerze, dzięki czemu są wstanie dobrze rozumieć powiązania pomiędzy tokenami. Przy zadaniach generacyjnych im Bert sprawdzi się lepiej przy małych generacjach a Gpt-2 na dużych.

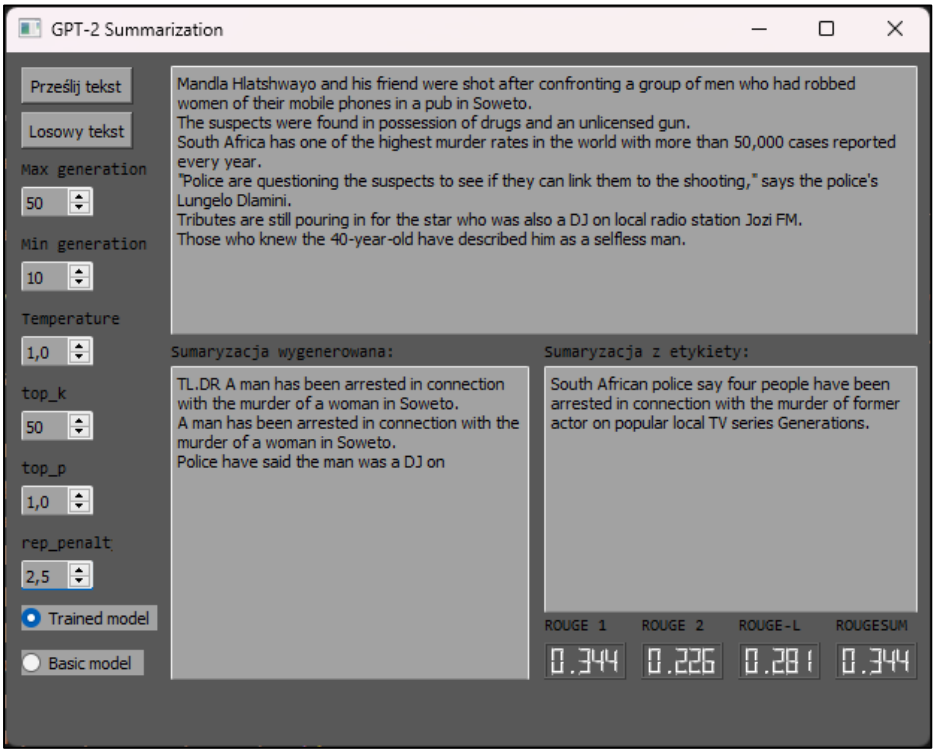
6 ZASTOSOWANIE SIECI GPT

Przechodząc do praktycznej części pracy za cel założono zaimplementowanie sieci gpt-2 do wybranego zadania NLP. Do wykonania tego zadania wykorzystamy właściwość tej sieci jaką jest uczenie transferowe i „dotrenujemy” model do naszych potrzeb.

Wszystko będzie się odbywać na platformie Jupyter. Będziemy wykorzystywać frameworki pytorch i tensorflow oraz technologie obsługi Karty graficznej CUDA. Modele oraz narzędzia do ich obsługi zapewni nam Hugging face.

Zadaniami na których chcemy się skupić jest sumaryzacja oraz generacja tekstu. W przypadku sumaryzacji model będzie otrzymywał tekst którego podsumowanie będzie nam zwracał. Możemy tworzyć sumaryzację np. z artykułów lub rozmów. Przez generację tekstu możemy rozumieć w tym kontetkście generowanie tekstu sumaryzacji.

Po stworzeniu modelu wytrenowanym do danego zadania wystawimy go do prostej aplikacji okienkowej która będzie odpowiadała za obsługę tego modelu. Do jej implementacji wykorzystamy framework PyQT.



Rys. 6.1: Aplikacja GUI dołączona do pracy
źródło: Własne

6.1 WYKORZYSTANE TECHNOLOGIE

Do wytrenowania modelu gpt-2 wykorzystano wiele powiązanych ze sobą technologii w celu uzyskania jak najlepszych efektów.

6.1.1 Python

Język programowania stworzony przez Guido van Rossuma w 1991 roku. Jest to wysokopoziomowy język interpretowany do ogólnego zastosowania. Składnia pythona jest czytelna i intuicyjna dzięki czemu w prosty sposób możemy implementować w niej zaawansowane struktury takie jak transformery lub sieci neuronowe. Python jest bogaty w biblioteki, moduły i frameworki.

Rys. 6.1.1.1: Logo języka Python
źródło: https://en.m.wikipedia.org/wiki/File:Python-logo-notext.svg

Największe modele z dziedziny sztucznej inteligencji są implementowane lub przenoszone do pythona z uwagi na jego prostotę możliwości wystawiania bibliotek oraz modułów. Oficjalna wersja modelu Gpt-2 została zaimplementowana w pythonie przy użyciu tensorflow.

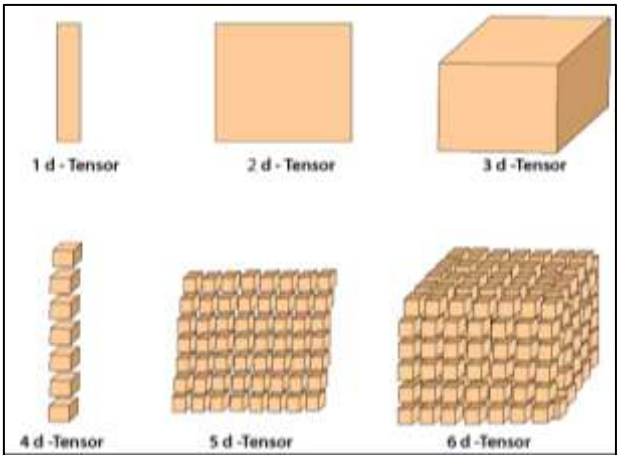
6.1.1.a *TensorFlow*

Biblioteka pythona rozwijana przez google'a. Służy do budowy i treningu modeli uczenia maszynowego przy pomocy głębokich sieci neuronowych.

Rys. 6.1.1.a.1: Logo tensorflow
źródło: https://en.wikipedia.org/wiki/TensorFlow

Na tensorflow możemy znaleźć modele do wielu zadań sztucznej inteligencji takich jak, klasyfikacja obrazów, predykcja czasów rzeczywistych, analiza danych czy w naszym przypadku przetwarzanie języka naturalnego.

Podstawową jednostką biblioteki tensorflow są tensory czyli wielowymiarowe macierze, które reprezentują dane w formie tablic numerycznych.



Rys. 6.1.1.a.2: Proste przykłady tensorów
źródło: https://medium.com/mlearning-ai/what-are-tensors-495cf37c18e6

Są one fundamentem w algorytmach uczenia maszynowego, a tensorflow jest przystosowany do wykonywania potężnych obliczeń na nich. Kolejną biblioteką wykorzystaną w projekcie jest Pytorch.

6.1.1.b Pytorch

Jest to framework do uczenia maszynowego skupiony na uczeniu głębokim. Jest bardziej elastyczny od tensorflow dzięki czemu idealnie nadaje się do badań.

Rys. 6.1.1.b.1: Logo pytorch
źródło: https://pl.m.wikipedia.org/wiki/Plik:PyTorch_logo_black.svg

Tak jak tensorflow ten framework pracuje na tensorach. Moduły torchtext oraz torchvision zawierają zestawy narzędzi i zbiorów danych dla zadań NLP i wizji komputerowej. Pytorch pozwala na dynamiczne modyfikowanie modeli oraz korzysta z autograd, które jest to mechanizmem do automatycznego obliczania pochodnych w trakcje wstecznej propagacji.

6.1.2 Hugging Face

Hugging Face jest to firma zajmująca się sztuczną inteligencją i NLP. Stworzyli platformę do zarządzania modelami sztucznej inteligencji aktualnie skupiając się na transformerach.

Rys. 6.1.2.1: Logo Hugging Face
źródło: https://huggingface.co/brand

Na platformie możemy znaleźć wiele przydatnych bibliotek dostarczających narzędzia oraz funkcje ułatwiające prace z zaawansowanymi modelami językowymi. Firma jest zaangażowana w społeczność open-source i zachęca do współpracy przy rozwoju i ulepszaniu modeli oraz narzędzi.

6.1.2.a *Modele*

Platforma zapewnia wiele modeli językowych implementowanych przez użytkowników z których każdy może skorzystać dzięki prostocie i przejrzystości samej platformy i jej elementów.

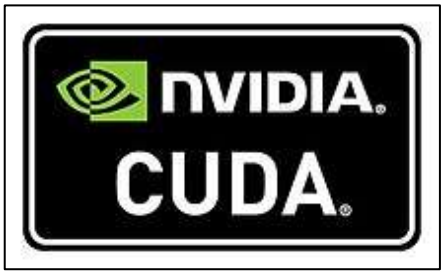
Możemy znaleźć modele bert, gpt czy bart. Oprócz podstawowych modeli, użytkownicy mogą wrzucać również ich zmodyfikowane wersje które są przystosowane do specyficznych zadań czy aplikacji.

Oprócz modeli możemy platforma zapewnia również:

- Datasety – zbiory danych do trenowania modeli zapewnione przez samą firmę oraz użytkowników
- ‘Spaces’ – są to aplikacje tworzone przez użytkowników implementujące niektóre modele NLP działające na platformie

6.1.3 CUDA

Pytorch oraz tensorflow dają możliwość trenowania modeli za pomocą GPU. Do wykonania tego potrzebujemy CUDA (Compute Unified Device Architecture). Jest to architektura obliczeniowa stworzona przez firmę Nvidia.



Rys. 6.1.3.1: Logo firmy Nvidia i narzędzia CUDA
źródło: https://en.wikipedia.org/wiki/CUDA

Ogólnie jest używana do przyśpieszania obliczeń równoległych i graficznych, ale wyżej wymienione biblioteki pythona używają tego narzędzia do przenoszenia obliczeń z CPU na GPU co znacząco poprawia ich szybkość.

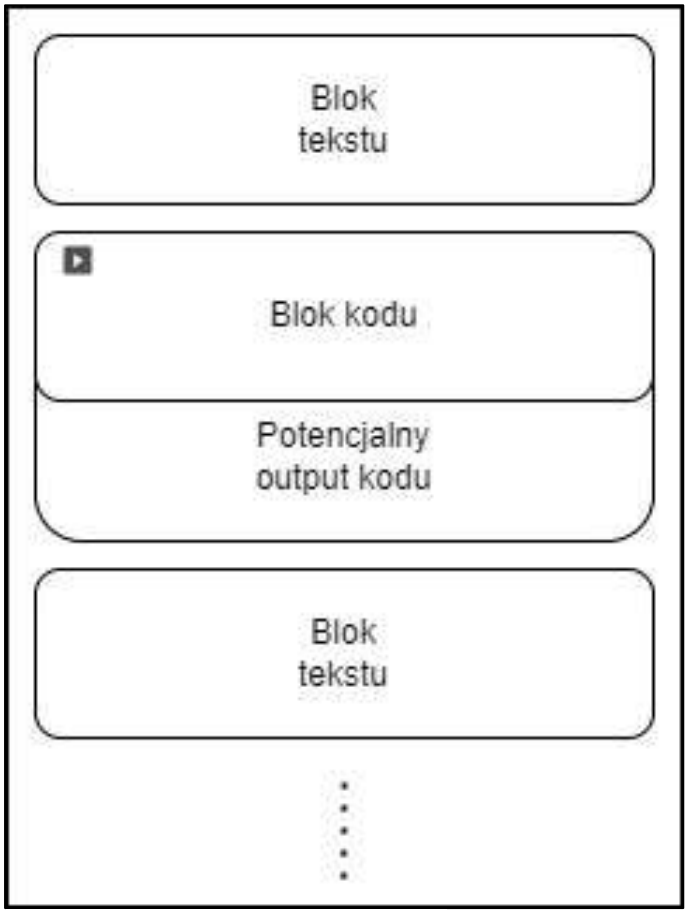
6.1.4 Jupyter

Jupyter jest projektem open-source pozwalającym na obliczenia w wielu językach programowania. Obliczenia mogą być wykonywane interaktywnie krok po kroku. Dzięki swojej przejrzystości i prostocie zyskał uznanie w dziedzinie uczenia maszynowego.

Rys. 6.1.4.1: logo jupitera
źródło:https://en.wikipedia.org/wiki/Project_Jupyter

Aktualnie projekt posiada dwa flagowe produkty: jupyter lab i jupyter notebook. Lab jest rozwiniętą wersją notebooka, ale z uwagi na prostote do projektu został wykorzystany notebook. Na notebook możemy patrzeć jak na edytor kodu taki jak vsc lub eclipse z tą różnicą, że notebook składa się z bloków tekstowych i kodu.

- Bloki tekstowe – są to bloki w których zamieszczamy tekst, obsługują formatowanie mardown.
- Bloki kodu – posiadają kod, który możemy odpalić w każdej chwili. Zostanie on wykonany na zewnętrznym kernelu. Możemy dowolnie odpalać wszystkie bloki w całym dokumencie (edytorze). Wyjście działania kodu pokazuje się tuż pod jego blokiem
- Kernel (w notebook'u nazywane jądrem) – środowisko w którym wykonują się odpalone bloki kodu. Znajduje się poza notebookiem działając cały czas w tle i przechowując wszystkie utworzone obiekty wszystkich bloków kodu. Możemy użyć własnego kernela języka (np. kernel python'a) albo skorzystać z serwera jupytera.



Rys. 6.1.4.2: Struktura notebooka jupitera
źródło: Własne

Dzięki blokom kodu i tekstu, możemy bez problemowo opisywać kod oraz wykonowywać odpowiednie bloki kodu wedle naszego uznania. Taki system zapewnia uzytkownikowi ogromne możliwości prezentacji oraz planowania skryptów. Bloki są ułożone od góry do dołu. Możemy przestawiać je między sobą oraz dowolnie usuwać i tworzyć nowe w dowolnym miejscu.

6.2 TWORZENIE MODELU

Na samym początku trzeba było zdecydować z jakiego modelu z dostępnych na Hugging face chcemy skorzystać. Oprócz wyboru wielkości należało wybrać odpowiednie parametry oraz strukturę.

6.2.1 Model GPT-2

Na stronie Hugging Face możemy znaleźć 4 oficjalne implementacje modelu GPT-2:

- GPT-2 small (default)
- GPT-2 medium
- GPT-2 large
- GPT-2 Extra large

Z uwagi na aktualne zasoby oraz potrzeby pracy został wybrany domyślny model GPT-2 Small. Do pobrania modelu wykorzystaliśmy bibliotekę transformers, w której hugging face przechowuje swoje modele. Mamy 2 dostępne modele różniące się strukturą.

- GPT2Model – podstawowy model GPT2, zwracający ukryte stany modelu czyli wewnętrzną reprezentację wejścia
- GPT2LMHeadModel–GPT2Model rozszerzony o dodatkową warstwę wyjściową zamieniającą ukryte stany na wyjście modelu w formie wektorów liczbowych, które możemy przetłumaczyć na tekst za pomocą tokenizerów.

Ponieważ naszym ogólnym celem jest generacja tekstu, skorzystamy z modelu posiadającym warstwę wyjściową. Omówmy teraz najważniejsze parametry jakich używamy dla modelu:

- "activation_function": "gelu_new" – funkcja aktywacji neuronów w enkoderze
- "n_head": 12 - ilość bloków mechanizmu Multi-head attention w enkoderze
- "n_positions": 10000 – maksymalne wejście modelu (podane w tokenach)
- "vocab_size": 50257 – wielkość słownika

Przy przesyłaniu danych do modelu, możemy przesłać wiele dodatkowych ustawień które wpłyną na wyniki. Najważniejsze z nich to:

- max_new_tokens – maksymalna ilość generowanych tokenów
- min_new_tokens – analogicznie minimalna ilość generowanych tokenów
- pad_token_id – ustawiamy indentyfikator tokenu paddingu.
- bos_token_id – nadpisujemy indentyfikator tokenów startu zdania [BOS]
- eos_token_id – nadpisujemy indentyfikator tokenów końca zdania [EOS]

Padding uzupełnia najmniejsze teksty w zbiorze specjalymi tokenami tak aby wielkościowo były równe, albo przyrównuje je do maksymalnej przyjmowanej długości wejścia

6.2.2 Model Tokenizer

Oprócz modelu GPT-2 potrzebujemy również model tokenizera. Jest to model odpowiedzialny za przygotowywanie wejścia tekstowego dla modelów transformerów. Korzysta z algorytmu word embedding i kodowania pozycyjnego.

Na platformie Hugging face mamy przygotowane modele tokenizera specjalnie dopasowane pod odpowiednich modeli. Dla naszego modelu posiadamy tokenizer GPT2Tokenizer. Tak jak dla modelu GPT-2 tutaj też możemy wybrać analogiczne wielkości modelu więc decydujemy się na najmniejszy model. Dla tego modelu również mamy wiele parametrów które można ustalić przy jego definicji. Najważniejszymi są:

- vocab_file – ścieżka do pliku słownika którego ma użyć tokenizer
- unk_token – ustawiamy jaki token ma się pojawić w przypadku nie znanego tokenu, czyli takiego który nie występuje w słowniku ani nie jest specjalny
- bos_token - ustawiamy tokeny rozpoczynające zdania
- eos_token- ustawiamy tokeny kończące zdania
- pad_token – ustwiamy token paddingu

Do wytrenowania modelu GPT-2 potrzebujemy pętli treningowej. W tej pracy skorzystamy z pętli trainer od frameworku pytorch.

6.2.3 Trainer

Trainer jest pętlą treningową od frameworku pytorch. Za jej pomocą możemy dotrenować nasz model do wybranego celu. Sam hugging face zapewnia API do korzystania z tej pętli co ułatwia korzystanie z niej. Dla trainera możemy ustawić ogromną ilość parametrów (105) dlatego wymienimy najważniejsze które były zmieniane w pracy:

- per_device_train_batch_size – ustawiamy batch size dla trainera. Dzięki temu kolejne iteracje nie będą używać pojedynczych wartości z datasetu tylko z paczek wartości co przyśpiesza trening.
- data_collator – ustawiamy funkcję która będzie odpowiedzialna za podział danych na paczki (batch). W projekcie został wykorzystany DataCollatorForSeq2SeqTraining.
- num_train_epochs – ilość epok jakie ma wykonać pętla. Przez epokę rozumiemy pojedyncze przejście pętli po całym zbiorze danych.
- model – model który chcemy trenować
- train_dataset – dataset zbioru danych na których będziemy uczyć model, musi posiadać kolumny

- input_ids - wejście modelu
- attention_mask – lista 0 i 1 stwierdzająca które tokeny w input_ids należy brać pod uwagę przy wyliczeniach. Dla 0 nie liczymy i analogicznie dla 1 liczymy
- labels – etykiety danych treningowych. Na ich podstawie stwierdzamy błąd predykcji modelu i go poprawiamy

Po zdefiniowaniu trainera, możemy wykonać funkcje .train() która rozpocznie pętle. Podczas treningu jest zapisywany co jakiś czas checkpoint, czyli aktualny stan modelu i treningu. Dzieki niemu możemy wrócić do danego stanu w przypadku przerwania pętli.

6.3 IMPLEMENTACJA ZADANIA NLP

Po zapoznaniu się z wykorzystanymi modelami oraz technologiami, możemy przejść do zastosowania ich. Celem założonym w projekcie jest stworzenie modelu, który skutecznie będzie sumaryzował tekst. Zadanie sumaryzacji polega na wygenerowaniu podsumawania podanego tekstu. Branże i dziedziny które korzystają z dużych zasobów tekstów (raporty, książki itp.) mogą wykorzystywać modele NLP do tworzenia zwięzłych podsumowań oraz raportów.

Wybór GPT-2 do tego zadania patrząc na takie algorytmy jak BERT nie jest oczywisty z uwagi na fakt, że nie jest on przystosowany do obsługi dużych tekstów oraz przechwytywania ogólnego kontekstu, a do skutecznej generacji tekstu. Dekodery nie patrzą na cały tekst tylko na tą część która aktualnie została wygenerowana razem z wejściem. Jednakże dzięki wyborowi GPT-2 jesteśmy wstanie pokazać problemy przy przetwarzaniu języka naturalnego oraz ich rozwiązania. Zrozumiemy procesy działania jakich należy się podejmować przy pracy nad tego typu modelami.

6.3.1 Zbiór danych

W pierwszym kroku należało zdecydować jaką baze tekstów wybrać do trenowania sumaryzacji. Baza powinna posiadać tekst, który model będzie sumaryzowany oraz jego sumaryzację jako etykiety na których będzie bazował model przy poprawkach.

Jednym z największych zbiorów danych stworzonych do trenowania sumaryzacji jest Xsum dataset. Powstał on aby wytrenować konwulencyjną sieć neuronową Xsum (od Extreme sumarization). Poniżej znajduje się tablica pokazująca budowę zbioru.

Document	summary	id
23 October 2015 Last updated at 17:44 BST It's the highest rating a tropical storm can get and is the first one of this magnitude to hit …	Hurricane Patricia has been rated as a category 5 storm.	34615665

Tabela 6.3.1.1: Przykładowy wiersz zbioru Xsum

Dane w zbiorze pochodzą ze strony wayback machine, która posiadała zarchiwowane artykuły nadawcy BBC. Twórcom zbioru udało się zebrać 226,711 artykułów z wachlarzem tematów takich jak sport, polityka itd.. Zbiór następnie został podzielony w sposób losowy na podzbiory treningu, testu i walidacji.

- Train set - zestaw treningu posiada 90% danych i służy do trenowania modelu.
- Test set – zestaw testowy posiada 5% zebranych danych i służy do testowania modelu
- Validation set - zestaw walidacji również posiada 5% zebranych danych i służy do walidacji modelu. Jest używany razem z zestawem treningowym w pętli treningowej.

Z uwagi na sprzęt na którym został wykonany projekt będziemy korzystać z 10-30% dataset'u.

Ponieważ korzystamy przy projekcie z najmniejszego modelu GPT-2, wejście takiego modelu musi być odpowiedniej wielkości. Model jest wstanie maksymalnie przyjąć wejścia do wielkości ok 2000 dlatego przed dalszą obróbką filtrujemy te wiersze które zawierają dokumenty artykułów mniejsze od 1000.

Przy trenowaniu modelu korzystamy z wariacji Trainera nazwaną seq2seqTraining które wspierają zadania które na podstawie zdania wejściowego A generują zdanie wejściowe B. W przypadku sumaryzacji chcemy tekst wejściowy zsumaryzować czyli wygenerować jego krótszą reprezentację.

Problemem jest fakt, że architektura dekodera nie pozwala aby jego wyjście i wejście były różnej wielkości, ponieważ model biorąc zdanie wejściowe i generując dla niego nowe słowo, w kolejnym kroku przesyła całe wyjściowe zdanie z nowo wygenerowanym tokenem na wejście w celach dalszej generacji.

Przy treningu zbiór danych jest dzielony na wejście oraz jego etykiety. Pierwszą myślą było użycie kolumn „document" jako wejścia oraz „summary" jako etykiety, ale z uwagi na problem opisany wyżej, jedynym dobrym rozwiązaniem było powiązanie tych obu kolumn i przesłanie ich jako wejścia i etykiety. Dlatego przy filtrowaniu zbioru w aspekcie wielkości zostawiono ok. 1000 znaków dla sumaryzacji oraz znaków specjalnych. Poniżej znajdziemy przykładowy tekst do generacji dla modelu.

> ***summarize*** *: 23 October 2015 Last updated at 17:44 BST It's the highest rating a tropical storm can get and is the first one of this magnitude to hit ...* **. TL:DR** *Hurricane Patricia has been rated as a category 5 storm.*

Podkreślona została docelowa sumaryzacja. Pomimo faktu, że model będzie w treningu miał za zadanie wygenerować cały powyższy tekst nauczy się rozumieć nie tylko artykuły ale również sumaryzacje.

6.3.2 Trenowanie sumaryzacji

Jak już wspomniano będziemy korzystać z pętli treningowej seq2seq która jest proponowana przez samą platformę hugging face przy zadaniach sumaryzacji. Do pętli treningowej przesyłamy następujące parametry:

- learning rate = 0.00002 – współczynnik uczenia γ . Dzięki niemu ustawiamy wielkość zmian jakie wykonuje model przy korekcie wag.
- weight_decay = 0.01 – wpływa na wielkość i zmiany wag. Dzięki niemu chronimy model przed problemem przetrenowania (problem w którym model za bardzo dostosowuje się do danych treningowych)
- batch_size = 10 – wielkość paczek danych z których ma korzystać pętla treningowa. Dzięki zbieraniu danych w paczki znacznie przyśpieszamy trening
- num_train_epochs = 30 – ilość epok procesu, czyli przejść pętli treningowej po całym zbiorze

Sam trening wykorzystywał ponad 15gb pamięci GPU dlatego w pracy nie zostały zastosowane większe ilości. Trening trwał ok. 5 godzin.

Do ewaluacji podczas treningu użyto metryki rouge, służącej do mierzenia jakość sumaryzacji. Jest wiele wariacji tej metryki ale w naszym przypadku skorzystaliśmy z największej i najlepszej jaką rest ROUGE-L. Aby obliczyć jego wartość spójrzmy na przykłady dwóch tekstów: „I really liked this movie" i „I liked this movie" , gdzie drugi będzie etykietą.

Na wynik ROUGE-L składają się dwa elementy, precision i recall. Recall przedstawia czy dana wygenerowana sumaryzacja posiada najważniejsze słowa z sumaryzacji etykiety.

$$ROUGE - L_{recall} = \frac{liczba\ pasujących\ do\ siebie\ słów}{liczba\ słów\ w\ etykiecie} \qquad (6.3.2.1)$$

Dla naszego przykładu recall będzie wynosił jeden. Precision z kolei patrzy na ilość wygenerowanych słów, więc reprezentuje ilość niepotrzebnych słów.

$$ROUGE - L_{precision} = \frac{liczba\ pasujących\ do\ siebie\ słów}{liczba\ słów\ w\ wygenerowane\ sumaryzacj} \qquad (6.3.2.2)$$

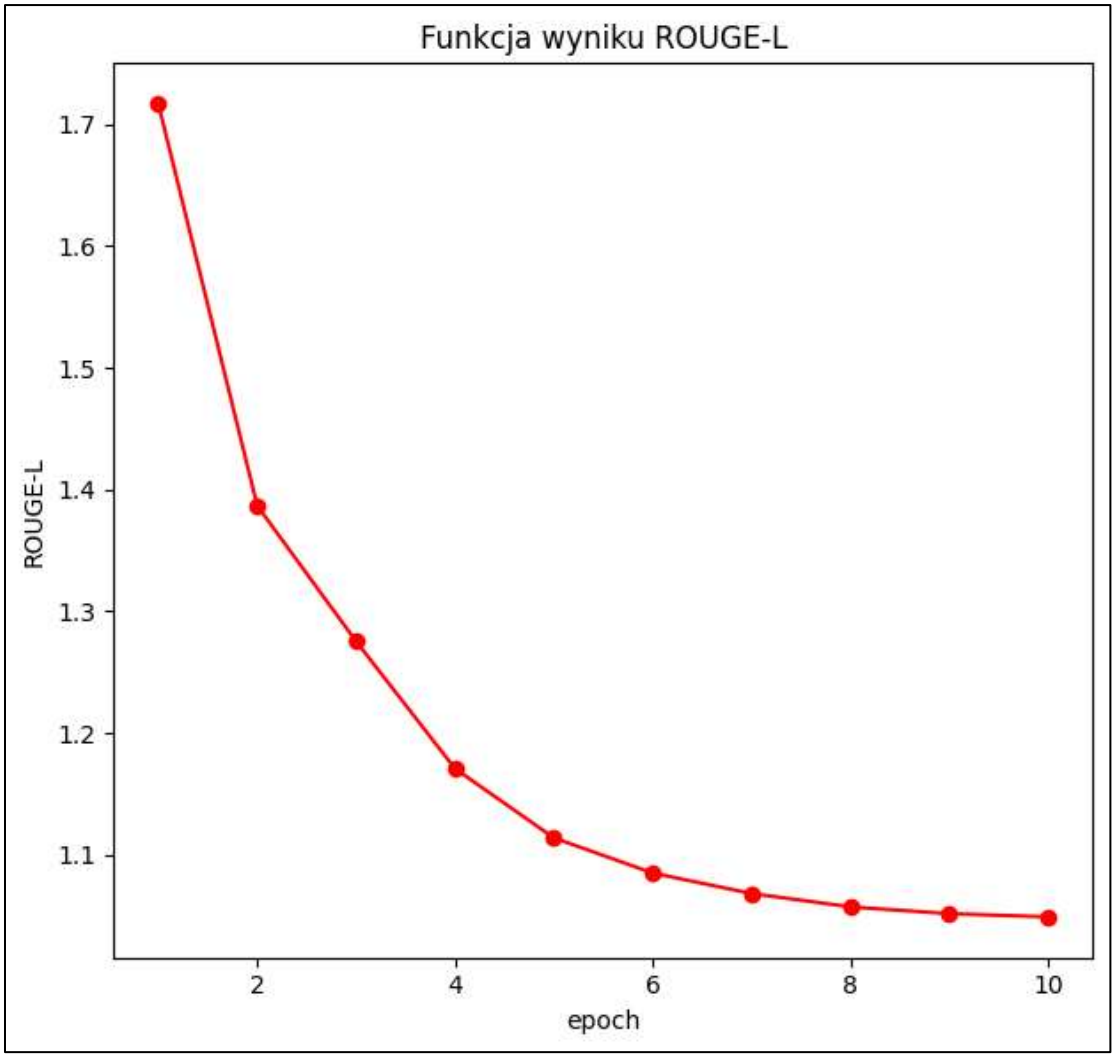
Precision dla powyższych przykładów będzie wynosił 6/7. Na koniec obliczamy jakość sumaryzacji czyli wynik ROUGE-L.

$$ROUGE - L = 2 * \left(\frac{precision * recall}{precision + recall}\right) \qquad (6.3.2.3)$$

Taki wynik pokazuje nam w jasny i wymowny sposób jakość generacji naszego modelu. Zasadniczo wyniki ROUGE-L są tym lepsze im bliższe są wartości 1.0.

6.3.3 Ewaluacja modelu i wyniki

Po przejściu przez pętle treningową model poprawił się w generowaniu sumaryzacji. Nadal są popełniane dość duże błędy, ale przy większym treningu i modelu jest możliwe osiągnięcie dobrych wyników.



Rys. 6.3.3.1: Wyniki treningu
źródło: Własne

Powyższy wykres przedstawia mniejszy trening oparty na takich samych zasadach na których był trenowany model główny, ale z mniejszą liczbą epok. Widzimy lekką poprawę w wynikach, co pokazuje nam że treningi wpływają pozytywnie na działanie modelu. Dobry wynik nie koniecznie musi oznaczać dobre sumaryzacje, ponieważ pamiętajmy że zadaniem modelu jest generacja całych artykułów razem z ich sumaryzacjami. Z tego powodu model może mieć problemy w momencie gdy otrzyma cały artykuł którę będzie trzeba zsumaryzować.

W poniższej tabeli przedstawimy przykładowe wyniki modelu z etykietami danych oraz nie wytrenowanym modelem

		Przed treningiem	Po treningu
Dokument	summary	Wynik	Wynik
The crash happened about 07:20 GMT at the junction of the A127 and Progress Road in …	A jogger has been hit by an unmarked police car responding to an emergency call, leaving him with "serious life-changing injuries".	The crash happened at about 07:20 GMT at the junction of the A127 and Progress Road in Leigh-on-Sea, Essex. The man, who police said is aged in his 20s, was treated at the scene for	The crash happened on the A127 at about 07:30 GMT. The driver of the car was taken to hospital with minor injuries. The crash happened on the A127 at about 07:30 GMT.TL.DR A man has been
23 October 2015 Last updated at 17:44 BST It's the highest rating a tropical storm can get and is the first one of this magnitude to hit mainland Mexico since 1959.…	Hurricane Patricia has been rated as a category 5 storm.	The Category 5 storm is the most powerful storm to hit Mexico since the 1950s. It is the second-most powerful storm to hit Mexico since the 1950s. It is the second-most powerful storm to hit Mexico since	The Category 5 storm is the strongest in the world, with winds of up to 100mph (130km/h) and winds of up to 100mph (150km/h). It's also the second highest in the world to hit
Administrators confirmed the redundancies affecting 38 staff at Galashiels-based Murray and Burrell. The business, established in 1928, went into administration last week citing …	Dozens of jobs have been lost after efforts to save an historic building firm in the Scottish Borders failed.	"We are looking at all options to ensure that we have the best possible staff," she said. "We are looking at all options to ensure that we have the best possible staff." The Scottish Government has said it will no	A new Scottish government has announced plans to cut jobs at Murray and Burrell. The company has been hit by a £1.5bn windfall from the sale of its shares.TL.DR A new Scottish government has announced
The Bath-born player, 28, has made 36 appearances for the Dragons since joining from Wasps in 2015…	Newport Gwent Dragons number eight Ed Jackson has undergone shoulder surgery and faces a spell on the sidelines.	Dragons v Munster: Matchday 1, Saturday, 7.30pm<|endoftext|>	Dragons boss Steve Hansen said: "We are delighted to have signed Eddie Jackson on a new contract. "He has been a fantastic player for us and we are looking forward to working with him to improve our squad and our squad."TL
The announcement ends months of uncertainty for Cornish Language Partnership staff whose contracts had been …	The government is spending nearly £400,000 to help save the Cornish language.	The Cornish Language Partnership is a non-profit organisation that works to improve the quality of life for people in the Cornish language. The Cornish Language Partnership is a non-profit organisation that works to improve the quality of life for	TL.DR The Cornish Language Partnership has been awarded a $1.5m grant from the United Nations to help it develop its skills. The Cornish Language Partnership has been awarded a $1.5m grant from the United Nations to

Tabela 6.3.3.1: Tabela z wynikami modelu.

Pomimo dobrych nadziei związanych z wynikami ROUGE-L widzimy pewne problemy patrząc się na surowe wyniki. Pojawiają się częste powtórzenia i tokeny „TL.DR" co nie jest pożądane w skutecznej sumaryzacji. .Może to być spowodowane:

- Modelem – model gpt-2 jest najmniejszym reprezentantem wśród swoich wariacji i nie nadaje się do sumaryzacji z uwagi na zamysł z jakim był tworzony
- Dość nie wielkim treningiem – trening był wykonywany tylko w 30 epokach na dość małej liczbie danych. Problemem może również być ich jakość
- Błędny trening – model był trenowany do generacji całych tekstów i ich sumaryzacji, co może powodować błędy
- Ewaluacja – ROUGE sprawdza się w porównywaniu technicznych aspektów etykiety i wyniku modelu. Lepszą ewaluacje moglibyśmy uzyskać trenując inny model zewnętrzny którego zadaniem będzie porównywanie tekstów.

Podsumowując udało nam się poprawić (w bardzo niewielkim stopniu) jakość modelu GPT-2 w zadaniu sumaryzacji tekstu, ale w przypadku potrzeby uzyskania lepszych wyników należałoby sięgnąć bo większe modele i dłuższe treningi.

6.4 APLIKACJA GUI

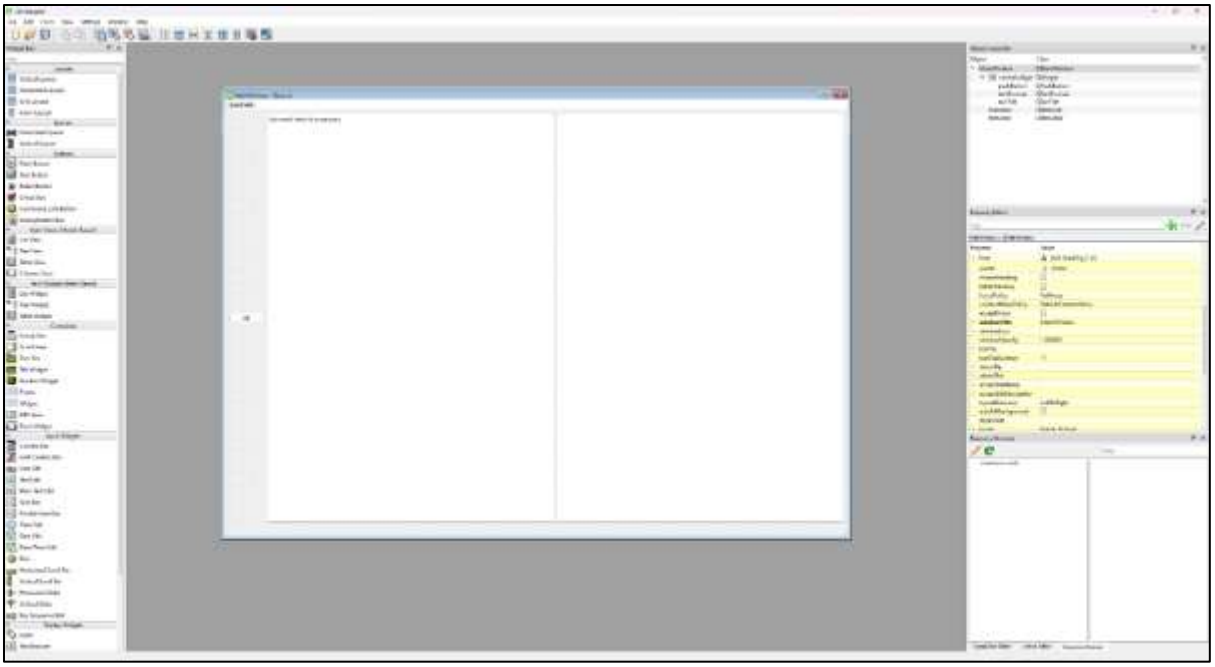
W celu prezentacji działania modelu, do pracy jest dołączona prosta aplikacja GUI stworzona w programie QT designer z wykorzystaniem biblioteki pythona PyQT. Aplikacja pozwala użytkownikowi wchodzenie w interakcję z modelem poprzez ustawianie parametrów i przesyłanie do niego tekstu na wejście w celu uzyskania sumaryzacji.

6.4.1 PyQT i QT Designer

Qt designer jest aplikacją do tworzenia prostych aplikacji GUI i eksportowania ich do wybranych języków programowania.

Rys. 6.4.1.1: Logo QT Designer
źródło: https://pl.wikipedia.org/wiki/Qt_Creator

Posiada intuicyjny interfejs graficzny oraz daje możliwość generacji kodu na podstawie stworzonych przez nas szablonów graficznych dla aplikacji GUI.

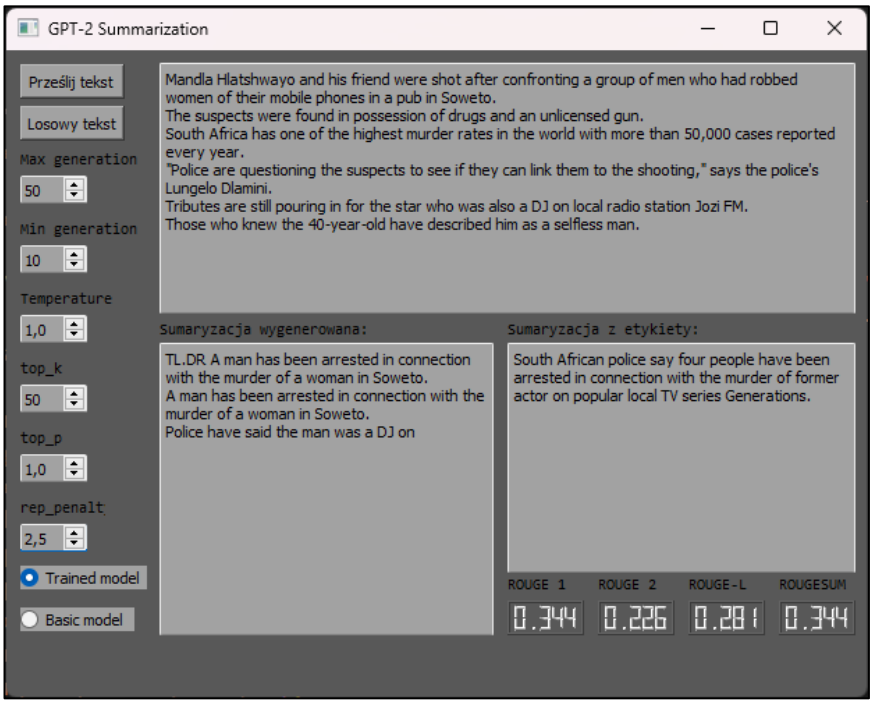


Rys. 6.4.1.2: Interfejs QT Designer
źródło: Własne

PyQT jest nakładką na biblioteki QT pozwalającą przenosić projekty stworzone w programie QT Designer do skryptów python'a.

6.4.2 Wygląd interfejsu oraz opcje

Celem aplikacji było pozwolenie użytkownikowi na wejście w interakcję z modelem poprzez przesyłane do niego parametry oraz teksty wejściowe.



Rys. 6.4.2.1: Interfejs aplikacj gui
źródło: Własne

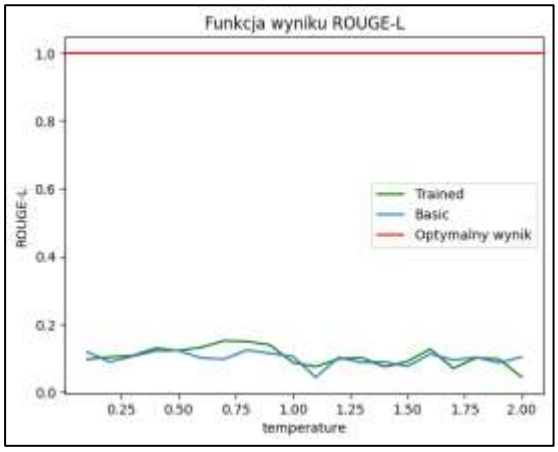
Skupmy się na razie na blokach z tekstem. Na samym górze znajduje się pole do którego wstawiamy tekst który chcemy zsumaryzować. Tak jak podpisy pod wskazują, na dole znajdują się pola, w których będą się pojawiać wyniki działania modelu i (jeśli to możliwe) etykiety dla tego tekstu. Za pomocą przycisku „Losowy tekst" przesyłamy losowy wiersz ze zbioru testowego danych. Jeśli wybieramy opcje losowego tekstu zostanie również przesłana etykieta tych danych, a po wyliczeniu przez model sumaryzacji w prawym dolnym rogu pokażą się metryki ROUGE jakie udało się modelowi osiągnąć dla tych danych.

Pod przyciskami mamy listę kolejnych parametrów które możemy przesłać do modelu:

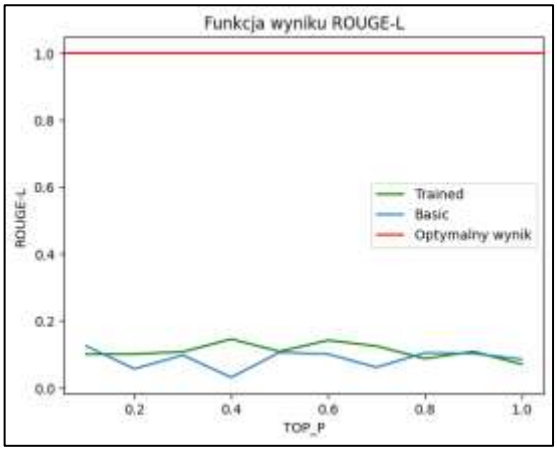
- Max generation = 200 – maksymalna dozwolona ilość wygenerowanych tokenów
- Min generation = 10 – minimalna ilość wygenerowanych tokenów
- Temperatura = 1.0 – openAi za pomocą temperatury zwiększa losowość wyników poprzez zwiększanie prawdobodobieństwa tokenom które mają go słabsze. Wartości są pomiędzy 0 a 2, gdzie większe wartości zwiększają losowość.
- top_k = 50 – na wyjściu modelu jest lista z prawdopodobieństwami kolejnego tokenu,a za pomocą filtriowania tzw. Top K wybieramy kolejny. Tylko te tokeny które mają prawdopodobieństwo wyższe od top_k są wybierane, aby następnie wylosować spośród nich kolejny token.

- top_p = 1.0 – na wyjściu modelu jest lista z prawdopodobieństwami kolejnego tokenu, a za pomocą filtriowania tzw. Top P wybieramy kolejny. Szukamy najmniejszej sumy tokenów których suma prawdopodobieństw przekroczy top_p, a następnie losujemy z nich kolejny token.
- repetition_penalty = 1.0 – za pomocą tego parametru możemy „kazać" model za powtarzanie tokenów. Kara zwiększa się analogicznie do tego parametru.

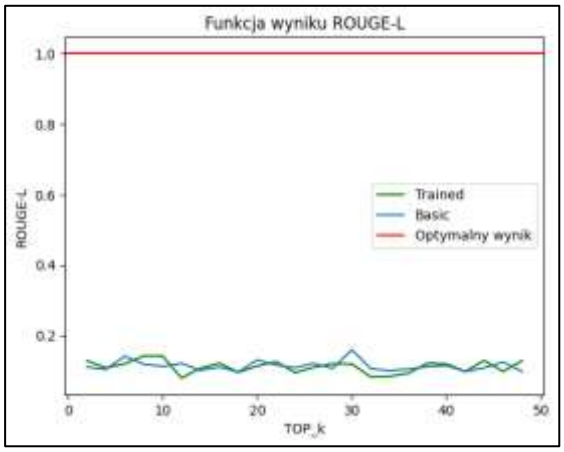
Aby zdecydować jakie parametry są najlepsze, poniżej znajdziemy wykresy ich zależności od metryki ROUGE-L.



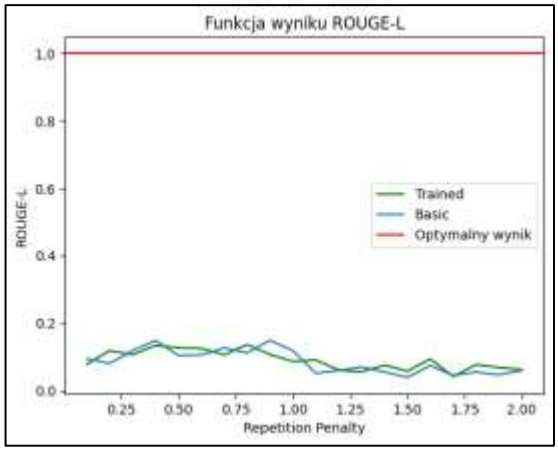
Rys. 6.4.2.1: Wykres Rouge-L od temperatury
źródło: Własne



Rys. 6.4.2.2:: Wykres Rouge-L od top_p
źródło: Własne



Rys. 6.4.2.3: Wykres Rouge-L od top_k
źródło: Własne



Rys. 6.4.2.4: Wykres Rouge-L od Repetition Penalty
źródło: Własne

Nie widzimy znaczącej poprawy przy zmianie parametrów. Może to być spowodowane takimi czynnikami jak przetrenowanie modelu lub nie odpowiedni dobór innych parametrów. Każdy wynik w tych wykresach jest średnim wynikiem wyciąganym na podstawie 10 różnych artykułów oraz ich sumaryzacji.

7 PODSUMOWANIE

Na początku pracy omówiono dziedzinę NLP. Można było zrozumieć czym jest język oraz z czego się składa. Udało się podkreślić największe problemy związane z naturalnym aspektem języka ludzkiego. Przedstawiono najważniejsze zadania z dziedziny, dzieląc je na podstawowe oraz branżowe. Praca wymieniła rozwiązania występujące w dziedzinie na przełomie jej istnienia, zaczynając od podstawowych algorytmów heurystycznych takich jak regeksy, a kończąc na rozbudowanych modelach sieci neuronowych wykonujących zaawansowane zadania tej dziedziny.

Aby zrozumieć najnowsze trendy w NLP, praca wyjaśnia czym jest głębokie uczenie oraz sieci neuronowe. Został omówiony koncept sztucznego neuronu poprzez głębsze spojrzenie na jego architekturę oraz części składowe z ich wariacjami w przypadku funkcji aktywacji. Dokument opisuje architekturę sieci neuronowej feed forward, tłumacząc czym są połączenia między neuronami oraz strategie połączeń między neuronami i warstwami. Po przedstawieniu sieci neuronowych praca wyjaśniła koncepty nauczania sieci poprzez algorytm wstecznej propagacji.

W kolejnym etapie praca przeszła do opisania modelu transformera który aktualnie jest najnowszym trendem w dziedzinie NLP. W celu zrozumienia budowy transformera rozebrano go na bloki enkodera i dekodera dzięki czemu łatwiej było zrozumieć architekturę modelu. Wyjaśniono budowę każdego z bloków, tłumacząc szczegółowo algorytm word2vec i mechanizmy Multi-head attention działające w ich wnętrzu. Pokazano na przykładach w jaki sposób proste zdania są zamieniane na wektory liczbowe przechowujące informacje o powiązaniach i pozycjach słów w tekście.

Jednym z celów pracy było porównanie dwóch najpopularniejszych modeli pochodzących od transformerów. W dokumencie zostały zestawione ze sobą modele GPT i BART. Scharakteryzowano każdy z tych modeli w celu ukazania różnic strukturalnych między nimi. Modele były trenowane różnymi metodami a ich twórcy mieli odmienne zamysły przy ich tworzeniu co sprawiło że pomimo korzystania z podobnych rozwiązań pochodzących z transformerów, posiadają ważne różnice przez co wybór odpowiedniego modelu do odpowiedniego zadania jest kluczowy do osiągnięcia skutecznych wyników.

W części praktycznej pracy podjęto próbę wytrenowania modelu GPT-2 do tworzenia sumaryzacji tekstów. W osiągnięciu tego celu wykorzystano modele i narzędzia pochodzące z platformy hugging face oraz bibliotek TensorFlow oraz PyTorch. Trening opierał się na generacji całych artykułów pochodzących od nadawcy BBC z dołączonymi do niego sumaryzacjiami wykonanymi przez człowieka. Dzięki temu model miał nauczyć się generować dobre podsumowania dla podanych artykułów. Było to nie typowe podejście do problemu, z uwagi na naturę modelu co skutkowało wieloma niepożądanymi efektami ubocznymi takimi jak pojawianie się nie pożądanych tokenów, częste

powtórzenia lub ogólny brak sensu w wygenerowanej sumaryzacji. Po wynikach modelu można stwierdzić że trening przyniósł pewne rezultaty, ale aby osiągnąć pożądane sumaryzacje należałoby podjąć następujące kroki:

- Zamienić model na jego bardziej rozbudowaną wersję: w pracy wykorzystano najmniejszą wersję modelu GPT-2, która ma najgorszą zdolność rozumienia i generowania tekstu. Większe modele posiadające więcej parametrów i dające większe możliwości wejściowe pozwoliłyby na bardziej zaawansowany trening co mogłoby skutkować lepszymi wynikami
- Zwiększyć podstawowe parametry treningu oraz danych treningowych: przy pętli treningowej zastosowano tylko 30 epok z rozmiarem danymi na poziomie mniejszym od 1000 słów. Przy lepszym sprzęcie, większej ilości epok i danych z większym rozmiarem model miałby większą szanse na przystosowanie się do nowego zadania co w rezultacie dałoby lepsze sumaryzacje
- Metoda treningowa: na ten moment, zastosowano klasyczną metodę CLM do wytrenowania GPT-2 poprzez generowanie całych arytkułów z sumaryzacjami. Taka metoda nie jest poprawna w kontekście generacji podsumowań tekstu dlatego należałoby poszukać lepszych metod, które model GPT-2 mógłby użyć przy treningu.
- Ewaluacja: Metodyka Rouge pomimo faktu że jest oficjalnym rozwiązaniem do sumaryzacji tekstu możę się nie spełniać w bardziej zaawansowanych przypadkach lub w kontekście modeli GPT. W celu poprawy wyników należyałoby poszukać lepszych metod ewaluacyjnych.

Powyższe rozwiązania mogą poprawić jakość wyników modelu i zwiększyć efektywność jego nauki.

Na koniec praca opisuję aplikację okienkową dołączoną do pracy. Aplikacja za pomocą interfejsu graficznego ma za zadanie wystawić model do interakcji z użytkownikiem. Użytkownik ma opcję wprowadzić swój własny tekst do sumaryzacji albo skorzystać z losowego tekstu pochodzącego z części testowej zbioru na którym był trenowany model. Interfejs zapewnia również możliwość zmiany najważniejszych ustawień generacji wpływających na wybierane tokeny w wynikiu.

Dzięki pracy możliwe było zdobycie wiedzy na tematy NLP i uczenia głębokiego. Poznać dogłębnie architekturę transformera oraz zrozumieć różnicę pomiędzy modelami GPT-2 a BERT. Doświadczenie zdobyte w trakcie próby wytrenowania GPT-2 do konkretnego zadania pozwala zetknąć się z problemami, z jakimi borykają się programiści i naukowcy pracujący nad rozwiązaniami NLP.

BIBLIOGRAFIA

Publikacje książkowe

[1] Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, Harshit Surana
Przetwarzanie języka naturalnego w praktyce. Przewodnik po budowie rzeczywistych systemów NLP

[2] Hobson Lane, Cole Howard, Hannes Hapke
Przetwarzanie języka naturalnego w akcji

Artykuły naukowe

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin
Attention Is All You Need, https://arxiv.org/pdf/1706.03762.pdf

[4] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever
Language Models are Unsupervised Multitask Learners, https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, https://arxiv.org/pdf/1810.04805v2.pdf

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei
Language Models are Few-Shot Learners, https://arxiv.org/pdf/2005.14165.pdf

Źródła internetowe

[7] How Does the Gradient Descent Algorithm Work in Machine Learning?
https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/

[8] Machine Translation on WMT2014 English-French
https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-french

[9] Machine Translation on WMT2014 English-German
https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-german

[10] Transformer’s Encoder-Decoder

https://kikaben.com/transformers-encoder-decoder/

[11] Basics of NLP Part-1 TOKENAZATION, Abdur Rahman Shabab Khan
https://medium.com/@almassco2007/basics-of-nlp-part-1-tokenazation-d124c15a01a8

[12] The Illustrated GPT-2 (Visualizing Transformer Language Models), Jay Alammar
http://jalammar.github.io/illustrated-gpt2/

[13] The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), Jay Alammar
http://jalammar.github.io/illustrated-bert/

[14] BERT Model – Bidirectional Encoder Representations from Transformers
https://quantpedia.com/bert-model-bidirectional-encoder-representations-from-transformers/

[15] Language Model Training and Inference: From Concept to Code
https://towardsdatascience.com/language-model-training-and-inference-from-concept-to-code-483cf9b305ef

[16] Josh Starmer, statQuest
https://statquest.org

[17] Grant Sanderson, 3blue1brown
https://www.3blue1brown.com