# Wprowadzenie do eksploracji danych tekstowych w środowisku WWW

Dokumentacja końcowa

Maria Swianiewicz, Wojciech Gruszka, Jakub Gajownik

26 maja 2020

**Temat projektu:** *Stworzenie chatbota.*

## 1 Interpretacja tematu projektu

Ogólnym tematem projektu jest stworzenie chatbota, co jest zagadnieniem bardzo szerokim. W związku z tym musieliśmy podjąć decyzję co do konkretnej specjalizacji tworzonego chatbota. Zdecydowaliśmy się stworzyć program pozwalający na ludzką komunikację tekstową na błahe tematy (small-talk) w języku angielskim. Nasz program nie posiada wiedzy eksperckiej z konkretnej dziedziny, za to będzie pozwala na przeprowadzenie względnie satysfakcjonującej rozmowy.

Z technicznego punktu widzenia zastosowaliśmy rozwiązania Deep-Learning, w szczególności głębokie sieci neuronowe. Poniżej zostały przedstawione metody i modele stosowane w przetwarzaniu języka naturalnego oraz przy budowaniu chatbotów.

## 2 Przegląd dostępnych rozwiązań

### 2.1 Sieci neuronowe wykorzystywane w przetwarzaniu języka naturalnego (NLP)

Do przetwarzania języka naturalnego wykorzystuje się różne rodzaje sieci neuronowych. Poniżej zostały omówione te najczęściej stosowane.

#### 2.1.1 Multilayer Perceptron (MLP)

Perceptron wielowarstwowy zbudowany jest z jednej warstwy wejściowej, kilku warstw ukrytych i jednej wyjściowej. Każdy neuron składa się z wielu wejść, których wartości są sumowane z odpowiednimi wagami, a następnie poddawane działaniu funkcji aktywacji (poza warstwą wejściową). Każdy neuron z warstwy poprzedzającej jest połączony z każdym neuronem warstwy następnej.

#### 2.1.2 Convolutional Neural Network (CNN)

Konwolucyjne sieci neuronowe są wariacją perceptronów wielowarstwowych, które zawierają co najmniej jedną warstwę konwolucyjna. Warstwa taka redukuje złożoność danych poprzez funkcję na danych wejściowych, która przekazuje wynik do następnej warstwy. Funkcja taka przetwarza część danych (na przykład mniejszą część obrazu) jako całość,

redukując w ten sposób ich złożoność. Pozwala to na zastosowanie większej liczby warstw i przetwarzanie bardziej skomplikowanego zbioru danych. Ten typ sieci jest szczególnie użyteczny w procesie przetwarzania obrazów.

#### 2.1.3 Recurrent Neural Network (RNN)

Rekurencyjne sieci neuronowe pozwala na zachowanie poprzedniego stanu neuronu, poprzez podanie wartości wyjściowej jako jedno z jego wejść. Pozwala to sieci na przechowanie kontekstu oraz wygenerowanie danych wyjściowych na podstawie poprzedniego stanu. Ten typ sieci jest często stosowany do tłumaczenia, rozpoznawania mowy czy generowania tekstów.

#### 2.1.4 Long Short-Term Memory (LSTM)

LSTM jest rodzajem sieci RNN, która operuje na dwóch dodatkowych węzłach oprócz wejścia i wyjścia podstawowej sieci RNN. Ich struktura pozwala na dłuższe przechowywanie kontekstu co z kolei prowadzi do możliwości przetwarzania dłuższych sekwencji danych.

#### 2.1.5 Sequence to Sequence Models

Sequence to Sequence Models zbudowane są z dwóch sieci rekurencyjnych - kodera i dekodera. Zadaniem kodera jest przetwarzanie danych wejściowych do postaci pośredniej, a dekodera generacji danych wyjściowych.

### 2.2 Modele sieci neuronowych używanych w implementacji chatbotów

#### 2.2.1 Retrieval-based Neural Network

Ten rodzaj programów pozwalających na ludzką komunikację tekstową posiada stałe repozytorium, które pozwala na przyporządkowanie odpowiedzi do pytania. Bardziej zaawansowane modele przechowują kontekst rozmowy i generują wiele odpowiedzi na podstawie kontekstu, następnie wybierając najlepszą na podstawie wybranych kryteriów oceny.

#### 2.2.2 Generation-based Neural Network

Ten model sieci jest modelem przeciwstawnym do poprzedniego, nie bazuje on na bazie odpowiedzi. Są one generowane od zera i opierają się o uczenie maszynowe i dane treningowe. Dla takich modeli sieci najbardziej odpowiednie są sieci typu Sequence to Sequence.

## 3 Rozwiązanie i jego implementacja

Po dogłębnej analizie zdecydowaliśmy się na zastosowanie modelu sieci Generation-based Neural Network, który nie wymaga bazy odpowiedzi.

Do implementacji rozwiązania użyliśmy język Python wraz z biblioteką Keras, która w swoich niższych warstwach wykorzystuje bibliotekę TensorFlow. Taka kombinacja narzędzi udostępnia interfejs na wysokim poziomie, cechuje się wysoką wydajnością oraz pozwala na zastosowanie wysoce wydajnych jednostek GPU.

### 3.1 Przygotowanie korpusu

Jako korpus wybraliśmy zbiór danych pochodzący z konwersacji z forum internetowego *Reddit*. Dane zostały pozyskane z użyciem narzędzia *pushshift.io* która "scrapuje" portal Reddit w czasie rzeczywistym i pozwala na przegląd tych danych z użyciem narzędzia *ElasticSearch*. Dane pochodzą z podforum *r/CasualConversation* zawierającego posty o dość szerokim zakresie tematyki, skupiającym się na zwykłej rozmowie oraz z podforum *r/AskReddit* zawierającego różne pytania wraz z odpowiedziami. Ten konkretny zbiór danych został wybrany, ze względu na to, że w znacznym stopniu przypomina normalną rozmowę, co pozwala spełnić określony cel. Wypowiedzi z forum zostały przefiltrowane tak, aby nie przekroczyć limitu 160 znaków na wypowiedź (w celu zbliżenia danych do prawdziwej rozmowy w przeciwieństwie do długich monologów). Danych z *r/AskReddit* jest zdecydowanie mniej aby nie nauczyć bota tylko odpowiadania na pytania. Pozyskaliśmy dane o rozmiarze 65431 rekordów (z czego ok 8000 stanowią dane z podforum *r/AskReddit*). Korpus po eliminacji powtórzeń 62914 wypowiedzi.

### 3.2 Przetwarzenie danych i *Word Embedding*

Pierwszym etapem przetwarzania danych jest zamiana wszelkich wielkich liter na małe. Jest to dość mocna normalizacja, która zamienia również nazwy własne na małe litery, jednak należy uwzględnić, że nasze dane uczące pochodzą z forum internetowego na którym większość użytkowników nie przestrzega zasad pisowni. Następnie dokonywana jest tokenizacja wypowiedzi na pojedyncze słowa. Tokenizacja odbyła się za pomocą narzędzia *TwitterTokenizer* z pakietu *nltk*. Jest to tokenizer przystosowany do poprawnego rozdzielania wyrażeń charakterystycznych dla mediów społecznościowych. W szczególności pozwala na poprawne rozdzielanie znaków przestankowych w emotikonach. Kolejnym etapem jest przekonwertowanie każdego tokenu wykorzystując metodę *Word Embedding*. Zakłada ona odwzorowanie kolejnych słów na postać wektorów, a następnie przemnożenie wektora przez tablicę powiązań. Pozwala to na uchwycenie zależności między poszczególnymi słowami.

W projekcie do *Word Embedding* zastosowaliśmy algorytm GloVe, w szczególności zastosowaliśmy modele nauczone uprzednio (pre-trained) na zbiorze danych użytym przy trenowaniu sieci. Planowaliśmy także zastosowanie gotowego modelu bazującego na wpisach z Wikipedii oraz z Twittera, jednak zrezygnowaliśmy z nich z powodów opisanych w dalszej części dokumentacji. Lokalne modele GloVe (uczone na zbiorze danych użytym do trenowania) zostały wygenerowane z użyciem następujących parametrów:

- Rozmiar kontekstu: 10 słów (5 z przodu, 5 z tyłu słowa)
- Rozmiar embeddingu (wymiar przestrzeni): 100
- Liczba powtórzeń słów: odpowiednio 10 i 1 dla różnych testów

Model GloVe był uczony w 500 epokach w paczkach po 512 wektorów, z parametrem uczenia 0.05. Fragment wygenerowanej mapy wektorów TSNE została przedstawiona na rysunku 1.

Z gotowych modeli GloVe rozważaliśmy:

- Wytrenowany model uczony na zbiorze sześciu milliardów zdań z Wikipedii (ref. `https://nlp.stanford.edu/projects/glove/`)
- Wytrenowany model uczony na zbiorze dwunastu milliardów wpisów na Twitterze (ref. `https://nlp.stanford.edu/projects/glove/`)

Rysunek 1: Fragment mapy TSNE wytrenowanego modelu GloVe

Z modelu uczonego na Wikipedii zrezygnowaliśmy ze względu na małą zgodność języka z językiem używanym na Reddicie. Pomimo, że w obu przypadkach jest to język angielski to użytkownicy Reddita często nie dbają o poprawność ortograficzną zapisywanych tekstów (np. *Im* zamiast *I'm*), używają emotikon (zarówno unicode jak i tekstowych) oraz pojęć i określeń slangowych. Słowa takie nie występują w modelu bazującym na Wikipedii.

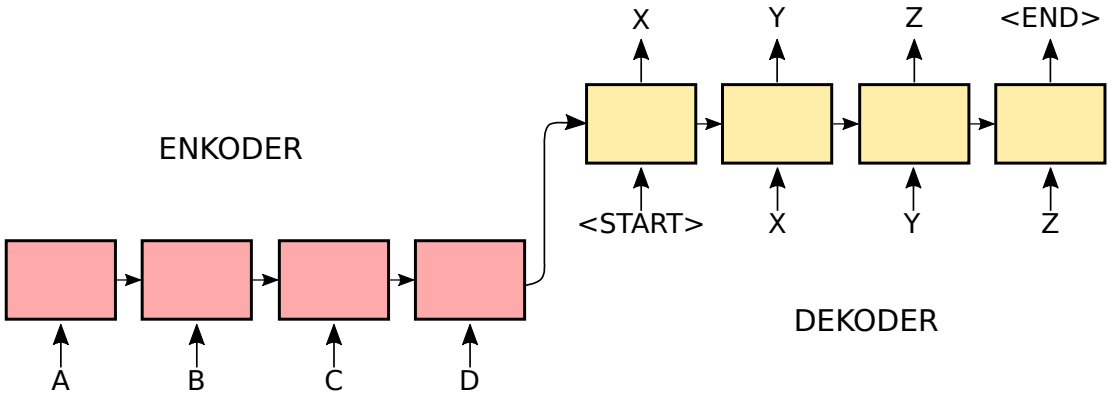
Drugą opcją było zastosowanie modelu bazującego na Twitterze - użytkownicy, podobnie jak na Reddicie, mają skłonności do niepoprawnych wypowiedzi oraz używania emotikon. Jednak model ten jest bardzo duży i obejmuje wiele języków. Patrząc na to jak mały, w porównaniu do liczby tokenów we wspomnianym modelu GloVe, jest nasz zbiór uczący można było się spodziewać niepoprawnej interpretacji tokenów i generację wpisów mieszających języki. Dodatkowo problemem okazała się granica wydajnościowa maszyny, na której wykonywane były obliczenia, w szczególności niemożliwa była alokacja pamięci na macierz embeddingów.

### 3.3 Model sieci

Do przetwarzania wypowiedzi i generacji odpowiedzi zastosowaliśmy model Sequence to Sequence. Złożony jest z dwóch sieci: pierwsza pełni rolę enkodera, który przetwarza dane wejściowe, druga zaś dekodera, który generuje wynik. Zastosowane zostały rekurencyjne sieci LSTM, ponieważ ich struktura pozwala na dłuższe przechowywanie kontekstu, co jest niezbędne do przeprowadzenia konwersacji posiadającej jakikolwiek sens. Dodatkowo LSTM, w przeciwieństwie do innych sieci rekurencyjnych, pozwala wyeliminować problem zanikającego gradientu.

Na rysunku 3.3 przedstawiono uproszczoną architekturę modelu Sequence to Sequence. Kolejne prostokąty przedstawiają kolejne iteracje sieci LSTM. Na wejście każdej iteracji enkodera podawany jest wektor reprezentujący kolejne słowo, oznaczony na rysunku literami A, B, C i D. Na wejście pierwszej iteracji dekodera podawany jest token startowy (<start>), a w kolejnych wynik poprzedniej iteracjach, aż do momentu uzyskania tokenu końcowego (<end>). Stan ukryty dekodera inicjowany jest końcowym stan ukrytym

enkodera.




Rysunek 2: Architektura modelu Sequence to Sequence

W czasie trenowania sieci, na wejście dekodera podawane są tokeny oczekiwane, a nie uzyskane w poprzedniej iteracji. Dzięki temu wynik iteracji nie jest zależny od poprzednich. Dla każdej iteracji obliczany jest rozkład prawdopodobieństwa przewidywanego tokenu, a na jego podstawie funkcja straty poprzez zastosowanie entropii krzyżowej. Następnie wszystkie wartości są uśredniane i propagowane wstecznie.

Niezadowalające efekty spowodowały, że do modelu Sequence to Sequence dodaliśmy mechanizm atencji. W modelu Sequence to Sequence końcowy stan ukryty enkodera musi przechowywać zebrane wszystkie informacje z całej wypowiedzi, ponieważ tylko on jest przekazywany do dekodera. Szczególnie problematyczne staje się to przy przetwarzaniu dłuższych wypowiedzi np. z powodu wybuchającego gradientu. Rozwiązaniem problemu jest mechanizm atencji, który polega na dodaniu wektora kontekstowego (*ang. context vector*) obliczanego osobno dla każdej iteracji dekodera jako średniej ważonej ukrytych stanów ze wszystkich iteracji enkodera. Wagi zależą od tokenu wejściowego w dekoderze i oznaczają atencję poświęconą konkretnemu tokenowi do wygenerowania następnego tokenu dekodera. Wektor kontekstowy ma wpływ na to, jaki token zostanie wygenerowany. W ten sposób model może wyłonić najważniejsze informacje i lepiej odwzorować złożoną relację między wypowiedzią wejściową, a oczekiwany wyjściem.

## 4 Testy

W ramach testów podjęliśmy próbę wytrenowania kilku sieci, które różniły się liczbą neuronów, a także zastosowany model *Word Embedding*. Przy testowaniu sieci stosowaliśmy test Turinga, oceniając sensowność odpowiedzi i w jakim stopniu można było odnieść wrażenie, że rozmawia się z normalnym człowiekiem.

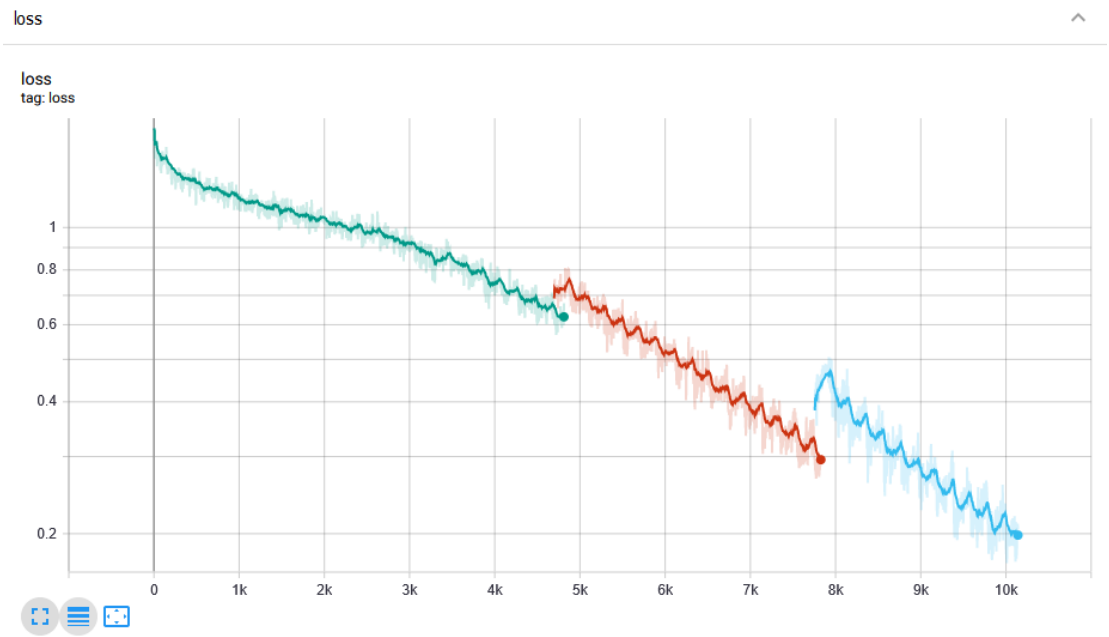
Pierwszą próbę podjęliśmy stosując same sieci LSTM posiadające 256 neuronów. Uzyskane wyniki były bardzo słabe. Przypuszczaliśmy, że wynika to z trudności z utrzymaniem kontekstu przez dłuższy czas w obrębie wypowiedzi. Problem ten rozwiązaliśmy poprzez użycie mechanizmu atencji. Kolejne testy potwierdziły nasze przypuszczenia i znacząco polepszyły uzyskiwane wyniki.

Następna faza testów dotyczyła dobrania optymalnej liczby neutronów. Zastosowaliśmy kolejno 256, 512, 768 i 1024 komórki LSTM. Najlepsze wyniki uzyskaliśmy dla sieci o 768 neuronach. Dla niższych rozmiarów sieci wartość funkcji straty spadała wolno i była niestabilna. Także przy manualnych testach odpowiedzi chatbota były często pozbawione sensu. Przy próbach zwiększenia liczby komórek LSTM do 1024, mimo dłuższego ucze-

nia nie uzyskaliśmy lepszych wyników, dlatego w dalszych testach pozostaliśmy przy 768 neuronach.

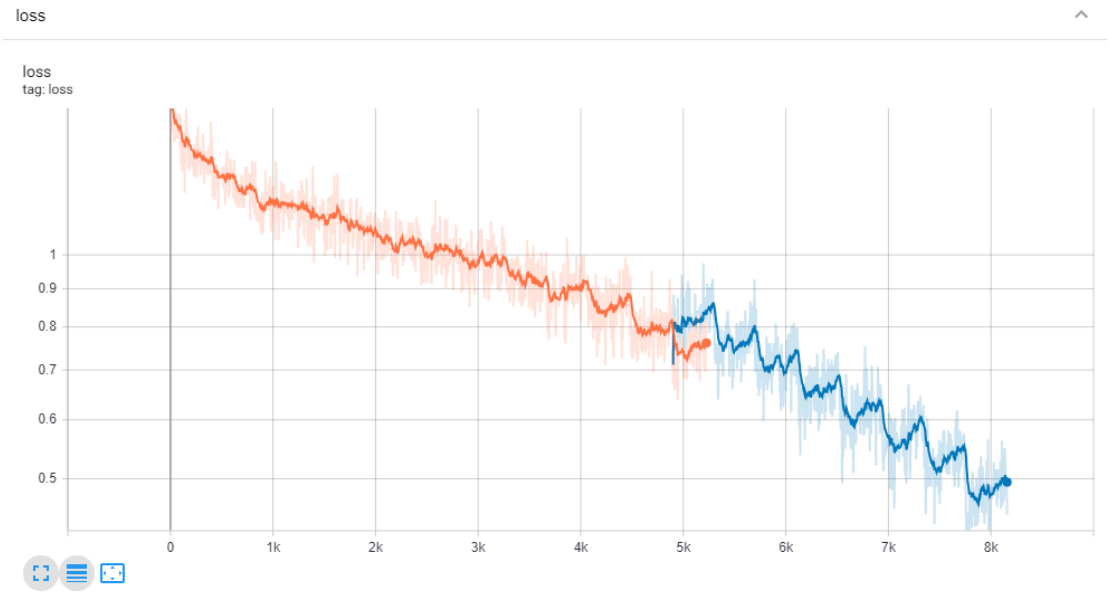
Testowaliśmy także różne modele *Word Embedding*. Podjęliśmy próbę wykorzystania gotowego modelu bazującego na wpisach z Wikipedii oraz z Twittera. Niestety z powodu wielkości modelu i niewystarczającej ilości pamięci RAM, nie udało się uruchomić procesu uczenia. Zastosowaliśmy zatem wygenerowany przez nas model przy użyciu algorytmu *Glove*. Przy uwzględnianiu w modelu *Word Embedding* wszystkich słów, sieć uczyła się bardzo wolno i nie dawała sensownych odpowiedzi. Kiedy w modelu *Word Embedding* zostały uwzględnione tylko słowa występujące minimum 10 razy, uzyskaliśmy najbardziej satysfakcjonujące odpowiedzi podczas konwersacji.

Ze względu na to, że nie wszystkie słowa występują w modelu *GloVe*, musieliśmy podjąć decyzję o tym co zrobić w sytuacji wystąpienia takich słów w danych treningowych. Początkowo zastępowane były wektorami zerowymi. Okazało się, że ich całkowite pominięcie dawało lepsze efekty. W naszym zbiorze uczącym jest znacząca przewaga słów, które nie zostały zawarte w modelu *Word Embedding* w wyniku zbyt małej liczby powtórzeń. Z tego powodu wyniki po usunięciu słów nieposiadających reprezentacji były lepsze, chociaż literatura nie zaleca takich kroków, ponieważ powoduje to mocną utratę kontekstu.




Rysunek 3: Wykres funkcji straty dla modelu z 768 neuronami oraz paczkami po 256 rekordów na epokę. Wykres podzielony jest na kilka sekcji ze względu na przerwy w trenowaniu powstałe przez błędy pamięci GPU. Nieciągłości wynikają z wygładzania funkcji.

Początkowe eksperymenty pozwoliły również estymować wartość funkcji straty poniżej której chatbot zaczyna prowadzić konwersację na zadowalającym poziomie. Po wykonaniu kilku iteracji, wartość ta została przez nas ustalona na 0,5. Jako warunek końcowy procesu uczenia ustaliliśmy zejście poniżej opisanej powyżej wartości progowej oraz rozpoczęcie stabilizacji wartości funkcji straty. Trenowanie było przerywane zatem w momencie, kiedy kolejna epoka nie wprowadzała znaczącej zmiany dla funkcji straty.




Rysunek 4: Wykres funkcji straty dla modelu z 768 neuronami oraz paczkami o rozmiarze 128. Wykres podzielony jest na kilka sekcji ze względu na przerwy w trenowaniu powstałe przez błędy pamięci GPU. Nieciągłości wynikają z wygładzania funkcji.

# 5 Instrukcja obsługi

Do poprawnego działania projektu wymagane jest posiadanie zainstalowanego środowiska `python` w wersji minimum: 3.8.3, managera pakietów python `pip3` oraz pobranie odpowiednich zależności projektu z użyciem komendy `pip3 install -r requirements.txt` w folderze projektu. W przypadku chęci uczenia modelu zaleca się także posiadanie zainstalowanego i skonfigurowanego pakietu `NVIDIA CUDA` oraz pakietu `tensorflow-gpu`.

W ramach projektu, stworzony został główny plik o nazwie *chatbot.py*, przez który odbywa się cała interakcja. Udostępnione zostały następujące opcje:

- `-h` lub `--help` - wyświetla ekran z dostępnymi opcjami programu
- `-t` lub `--train` - ustawia wykonanie w tryb uczenia sieci
- `-d <katalog>` lub `--dir=<katalog>` - pozwala specyfikować katalog który będzie wykorzystywany do zapisu/odczytu konfiguracji modelu (domyślnie *training_checkpoints*)
- `-b <rozmiar>` lub `--batch-size=<rozmiar>` - ustawia rozmiar jednej paczki danych uczących
- `-e <liczba>` lub `--epochs=<liczba>` - pozwala dobrać liczbę epok do wykonania w ramach procesu uczenia
- `-u <liczba>` lub `--units=<liczba>` - umożliwia zmianę liczby neuronów na warstwach LSTM
- `-g <ścieżka>` lub `--glove=<ścieżka>` - pozwala na zmianę ścieżki do wytrenowanego modelu GloVe (domyślnie *data/glove.local.txt*)
- `-s <ścieżka>` lub `--dataset=<ścieżka>` - ustawia ścieżkę do zbioru danych uczących (domyślnie *data/reddit_merged.csv*)

W przypadku gdy program wykryje katalog (domyślny lub podany wraz z opcją `--dir` w którym znajduje się plik konfiguracyjny (pozostały po poprzednim procesie uczenia), wczytane zostaną znajdujące się tam parametry sieci i dopiero po tym rozpocznie się faktyczne uruchomienie programu. Po uruchomieniu w trybie interaktywnym (bez opcji `--train`), załadowany zostanie cały model programu, a następnie wyświetlone zostanie pole do wprowadzenia tekstu. Wypowiedź jest zatwierdzana poprzez wciśnięcie klawisza *Enter*. Po wprowadzeniu tekstu przez użytkownika, chatbot generuje odpowiedź, wyświetla ją w kolejnej linii po czym program rozpocznie oczekiwanie na kolejną porcję tekstu użytkownika. Przykładowy interfejs wraz z odpowiedzią zaprezentowano poniżej.

```
User: Hi :)
Bot: hello ! <end>
User:
```

Kiedy program zostanie uruchomiony w trybie uczenia (opcja `--train`), utworzony zostanie plik konfiguracyjny zawierający rozmiar partii uczącej, liczbę epok oraz ścieżki do danych dla *Word Embeddings*). Po zakończeniu poprzedniego kroku, nastąpi faktyczne trenowanie sieci. Po każdej epoce, odkładane są artefakty (parametry modelu) do zdefiniowanego katalogu. Po zakończeniu procesu uczenia, program automatycznie przechodzi do trybu interaktywnego, opisanego powyżej.

## 6 Przykładowa konwersacja

Konwersacja przprawadzona przy użyciu modelu (768 neuronów, 64 rekordy na paczkę).

```
User: hi
Bot: hey there ! <end>
User: hey!
Bot: hey ! how are you doing today ? <end>
User: hello
Bot: hello :) how are you ? <end>
User: im fine
Bot: i ' m not that good <end>
User: that is sad
Bot: it just makes me feel ugly and now i like pineapple . <end>
User: i love you
Bot: i love you too . <end>
User: :red_heart:
Bot: :two_hearts: :two_hearts: <end>
User: what have you had for dinner
Bot: and a glass of coffee <end>
User: what did you have for dinner ?
Bot: coffee and protein shake a quick bite <end>
User: i like reading books
Bot: i like soccer , i like soccer too . <end>
User: kill me
Bot: i don't plan on it . <end>
User: nice hair dude
Bot: not your age i guess <end>
User: what's up dude ?
```

```
Bot: nothing much , i'm just watching black friday . how about you ? <end>
User: how old are you ?
Bot: i'm 16 , a junior in high school . <end>
User: are you my friend ?
Bot: not me . <end>
User: where are you from
Bot: netherlands <end>
User: do you like me ?
Bot: i do <end>
User: you are nice friend
Bot: i hope you have a good day . <end>
User: i'm a girl
Bot: i am a dude <end>
User: my cat just died
Bot: i ' m sorry for your loss . i want to do today ! <end>
User: are you human being
Bot: i am <end>
User: what is the most effective way to save species ?
Bot: don't kill the animals ! ! at all ! ! <end>
User: if seasons never changed , which would you most like to live in ?
Bot: autumn <end>
```

# 7 Wnioski

W czasie testów, zaobserwowaliśmy pewne problemy, które wynikają z zastosowanego modelu:

- Brak pewnych słów w modelu *Word Embedding*. Jeśli słowo nie występuje w słowniku, jest zastępowane wektorem zerowym lub pomijane, przez co, duża część sensu wypowiedzi może zostać pominięta. Wiadomość trudna do z interpretowania przez sieć, daje nieprzewidziane rezultaty. Okazało się to, największą trudnością w stworzonym przez nas modelu.
- Zastosowany *Word Embedding* nie rozróżnia różnych znaczeń tego samego wyrazu, a także nie wychwytuje związków frazeologicznych. Do rozwiązania tego problemu może być zastosowany *Word Embedding* kontekstowy. Jego zastosowanie nie ma sensu dla modeli Sequence to Sequence, szczególnie dla dekodera, gdzie istotne jest uzyskanie reprezentacji dla pojedynczych tokenów.
- Trudność z utrzymaniem kontekstu w obrębie dłuższej wypowiedzi. Zostało to częściowo rozwiązane przez dodanie warstwy atencji.
- Brak kontekstu całej rozmowy. Podczas testów manualnych zaobserwowaliśmy, że uciążliwe jest, że chatbot bierze pod uwagę tylko ostatnią wypowiedź, a nie posiada informacji o kontekście całej rozmowy. Wydaje się, że dla lepszego efektu, warto byłoby uwzględnić szerszy kontekst.

Najlepsze rezultaty otrzymaliśmy dla modelu Sequence to Sequence, zbudowanego z 768 komórek LSTM i przy zastosowaniu własnego modelu *Word Embedding*. W celu uzyskania lepszych wyników należałoby powiększyć korpus używany do tworzenia modelu *Word Embedding*, a także korpus do uczenia całej sieci. Nie udało nam zwiększyć wystarczająco

ilości danych, aby mogły reprezentować całą dziedzinę języka z powodu ograniczeń sprzętowych (wielkość pamięci RAM). Pozwoliłoby to rozwiązać problem dużej liczby tokenów nieposiadających reprezentacji, a także nauczyć chatbota sensownej rozmowy na większą liczbę tematów.

## Bibliografia


[1] Vyas Ajay Bhagwat. "Deep Learning for Chatbots". In: (2018).

[2] Jianfeng Gao, Michel Galley, Lihong Li, et al. "Neural approaches to conversational ai". In: *Foundations and Trends® in Information Retrieval* 13.2-3 (2019), pp. 127–298.

[3] *Reddit Conversations - Dataset*. Dostęp zdalny (01.05.2020): `https://www.kaggle.com/jerryqu/reddit-conversations`.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[5] Oriol Vinyals and Quoc Le. "A neural conversational model". In: *arXiv preprint arXiv:1506.05869* (2015).