Politechnika Krakowska

Projekt z przedmiotu Przetwarzanie języka naturalnego

Quora Question Pairs

Hubert Wątorczyk

Maciej Zawisz

Szymon Żylski

2022

Spis treści

Wstęp

Celem projektu jest zbudowanie modelu sztucznej inteligencji, który będzie w stanie zidentyfikować, czy dwa pytania zamieszczone na platformie internetowej Quora są takie same. Quora to serwis internetowy, w którym użytkownicy mogą dzielić się swoją wiedzą i doświadczeniem. Stworzenie takiego modelu może być bardzo pomocne w wykrywaniu pytań, które są powtórzeniami lub duplikatami innych pytań zadawanych na tej platformie. Dzięki temu można zapobiec sytuacjom, w których użytkownicy zadają te same pytania wielokrotnie, co zwiększa ilość zawartości na platformie i utrudnia jej przeglądanie.

Model sztucznej inteligencji, który zostanie stworzony w ramach tego projektu, będzie wykorzystywał dane pochodzące z turnieju Quora Question Pairs, zorganizowanego przez Kaggle. Zbiór danych zawiera 404 287 par pytań podzielonych na zbiór trenujący i testujący. Zbiór trenujący zawiera informację, czy dane pytania są duplikatami, natomiast zbiór testujący tej informacji nie zawiera. W projekcie wykorzystany zostanie wyłącznie zbiór trenujący. Pytania zawarte w zbiorze są napisane w języku angielskim.

Struktura danych:

id	qid1	qid2	question1	question2	is_duplicate
0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

Wstępne przetworzenie danych

```
punctuation_pattern = f'([{string.punctuation}“”¨«»®´·º½¾¿¡§£€‘’])'
re_tok = re.compile(punctuation_pattern)

def tokenize(s):
    return re_tok.sub(r' \1 ', s).split()

def clean_text(s):
    try:
        return re.sub(r'[^A-Za-z0-9,?"\'. ]+', '', s).lower()
    except:
        return ""

stopwords = set(nltk.corpus.stopwords.words("english"))

df_train['question1'] = df_train['question1'].apply(lambda x: clean_text(x))
df_train['question2'] = df_train['question2'].apply(lambda x: clean_text(x))
```

Funkcje przetwarzania danych

Zmienna punctuation_pattern zawiera wyrażenie regularne służące do wyszukiwania znaków interpunkcyjnych w tekście. Następnie zmienna re_tok jest deklarowana jako obiekt typu re.Pattern, który jest utworzony za pomocą wyrażenia regularnego zapisanego w punctuation_pattern.

Funkcja tokenize przyjmuje jeden argument s, który jest ciągiem znaków i zwraca listę tokenów (czyli poszczególnych słów lub fragmentów tekstu) wyodrębnionych z tego ciągu. Funkcja wykorzystuje wyrażenie regularne zapisane w re_tok do wyszukiwania znaków interpunkcyjnych w tekście i dodaje do nich spację, aby oddzielić je od pozostałych słów. Następnie funkcja split dzieli ciąg znaków na poszczególne tokeny, używając spacji jako separatora.

Funkcja clean_text przyjmuje jeden argument s, który jest ciągiem znaków i zwraca ten ciąg po usunięciu wszystkich znaków, które nie są literami, cyframi lub niektórymi znakami interpunkcyjnymi. Funkcja również zamienia wszystkie litery na małe.

Zmienna stopwords jest deklarowana jako zbiór słów, które są uważane za zbędne w analizie tekstu i są usuwane z tekstu przed jego dalszą obróbką.

Następnie wykonuje się metoda apply na kolumnach 'question1' i 'question2' w zmiennej df_train. Metoda ta pozwala na zastosowanie funkcji clean_text do każdego elementu w kolumnach. W ten sposób teksty zawarte w kolumnach są oczyszczane z niechcianych znaków i zamieniane na małe litery.

Tworzenie sieci neuronowej

```
embed = hub.KerasLayer("https://tfhub.dev/google/nnlm-en-dim128-with-normalization/2", trainable=False)
```

```
input1 = Input(shape=(), dtype=tf.string)
input2 = Input(shape=(), dtype=tf.string)

embed1 = embed(input1)
embed2 = embed(input2)

dist = Lambda(lambda x: K.sqrt(K.sum(K.square(x[0] - x[1]), axis=-1, keepdims=True)))([embed1,embed2])

concat = Concatenate(axis=1)([dist])

hidden = Dense(9, activation="relu", kernel_regularizer=l2(1e-4))(concat)

hidden1 = Dense(9, activation="relu", kernel_regularizer=l2(1e-4))(hidden)

hidden2 = Dense(9, activation="relu", kernel_regularizer=l2(1e-4))(hidden1)

hidden3 = Reshape((1, 9))(hidden2)

gru = Bidirectional(GRU(units=32, return_sequences=False))(hidden3)

dropout = Dropout(rate=0.5)(gru)

out = Dense(1, activation="sigmoid", kernel_regularizer=l2(1e-4))(dropout)
model = Model(inputs=[input1, input2], outputs=out)
```

Zmienna embed jest deklarowana jako obiekt hub.KerasLayer za pomocą adresu URL do modelu naturalnego języka opublikowanego na TensorFlow Hub. Model ten jest używany do zamiany słów z pytań na wektory liczbowe (ang. word embeddings). Opcja trainable jest ustawiona na False, co oznacza, że model nie będzie uczył się nowych wag podczas treningu sieci neuronowej.

Następnie są tworzone dwa obiekty Input, które reprezentują wejście do modelu. Każdy z nich ma kształt pustego n-wymiarowego tensora i typ danych tf.string.

Zmienne embed1 i embed2 są tworzone za pomocą modelu embed i odpowiednio przekształcają teksty z pytań w wektory liczbowe.

Zmienna dist jest deklarowana jako obiekt Lambda. Funkcja ta oblicza odległość pomiędzy wektorami liczbowymi odpowiadającymi pytaniom. Odległość ta jest obliczana jako pierwiastek kwadratowy sumy kwadratów różnic pomiędzy poszczególnymi elementami wektorów.

Zmienna concat łączy odległość pomiędzy pytaniami w jeden tensor.

Następnie tworzone są kolejne warstwy sieci neuronowej: hidden, hidden1, hidden2, hidden3, gru i dropout. Warstwa hidden jest warstwą gęsto połączoną z 9 ukrytymi neuronami i funkcją aktywacji relu. Warstwy hidden1, hidden2 i hidden3 są takie same jak warstwa hidden. Warstwa gru jest warstwą typu GRU (gated recurrent unit), która jest rodzajem warstwy ukrytej sieci neuronowej

rekurencyjnej. Warstwa ta jest używana do przetwarzania sekwencji danych, takich jak tekst. Warstwa gru jest używana w modelu w dwóch kierunkach (ang. bidirectional), co oznacza, że przetwarza dane zarówno od początku do końca, jak i od końca do początku. Warstwa ta zwraca tensor o kształcie (batch_size, units), gdzie units to liczba jednostek (czyli neuronów) w warstwie. W przypadku tego kodu units wynosi 32.

Warstwa dropout jest warstwą, która losowo usuwa pewien procent neuronów podczas treningu, co ma zapobiegać nadmiernemu dopasowaniu (ang. overfitting). W przypadku tego kodu procent neuronów usuwanych podczas treningu wynosi 0.5.

Na końcu tworzona jest warstwa wyjściowa sieci neuronowej, która ma jeden neuron i funkcję aktywacji sigmoid. Funkcja ta jest używana do zwrócenia prawdopodobieństwa, że dwa pytania są podobne.

Podsumowanie architektury sieci:

```
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_65 (InputLayer)          [(None, 3000)]       0           []

 input_66 (InputLayer)          [(None, 3000)]       0           []

 reshape_72 (Reshape)           (None, 1, 3000)      0           ['input_65[0][0]']

 reshape_73 (Reshape)           (None, 1, 3000)      0           ['input_66[0][0]']

 lstm_86 (LSTM)                 (None, 1, 128)       1602048     ['reshape_72[0][0]']

 lstm_87 (LSTM)                 (None, 1, 128)       1602048     ['reshape_73[0][0]']

 concatenate_31 (Concatenate)   (None, 1, 256)       0           ['lstm_86[0][0]',
                                                                  'lstm_87[0][0]']

 dropout_54 (Dropout)           (None, 1, 256)       0           ['concatenate_31[0][0]']

 batch_normalization_56 (BatchN  (None, 1, 256)      1024        ['dropout_54[0][0]']
 ormalization)

 lstm_88 (LSTM)                 (None, 128)          197120      ['batch_normalization_56[0][0]']
```

```
 dropout_55 (Dropout)           (None, 128)          0           ['lstm
_88[0][0]']

 batch_normalization_57 (BatchN  (None, 128)         512         ['drop
out_55[0][0]']
 ormalization)

 dense_35 (Dense)               (None, 128)          16512       ['batc
h_normalization_57[0][0]']

 dropout_56 (Dropout)           (None, 128)          0           ['dens
e_35[0][0]']

 batch_normalization_58 (BatchN  (None, 128)         512         ['drop
out_56[0][0]']
 ormalization)

 dense_36 (Dense)               (None, 64)           8256        ['batc
h_normalization_58[0][0]']

 dropout_57 (Dropout)           (None, 64)           0           ['dens
e_36[0][0]']

 batch_normalization_59 (BatchN  (None, 64)          256         ['drop
out_57[0][0]']
 ormalization)

 dense_37 (Dense)               (None, 1)            65          ['batc
h_normalization_59[0][0]']

==================================================================
==========================
Total params: 3,428,353
Trainable params: 3,427,201
Non-trainable params: 1,152
```

Uczenie sieci neuronowej

```
model.compile(optimizer=Adam(1e-3), loss="binary_crossentropy", metrics=["accuracy"])

def step_decay(epoch):
    initial_lrate = 0.003
    drop = 0.5
    epochs_drop = 3
    lrate = initial_lrate * (drop**((1 + epoch)/epochs_drop))
    return lrate

lrate_scheduler = LearningRateScheduler(step_decay)
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
model_chkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', verbose=1, save_best_only=True)

callbacks = [lrate_scheduler, early_stop, model_chkpoint]

history = model.fit(x=[np.array(X_train_q1), np.array(X_train_q2)],
                    y=np.array(y_train),
                    batch_size=128,
                    epochs=5,
                    validation_data=([np.array(X_test_q1), np.array(X_test_q2)], np.array(y_test)),
                    callbacks=callbacks)
```

Pierwsza linia kodu używa metody compile obiektu model do kompilacji modelu. Metoda ta przyjmuje trzy argumenty: optimizer, loss i metrics. optimizer to obiekt opisujący algorytm optymalizacji, który jest używany do aktualizacji wag podczas treningu. Używana jest klasa Adam z modułu keras.optimizers, która implementuje algorytm optymalizacji Adam. Klasa ta przyjmuje jeden argument - wartość początkową stałej uczenia (ang. learning rate).

loss to funkcja straty, która jest używana do określenia, jak dobrze model przewiduje wyniki. W tym kodzie używana jest funkcja binary_crossentropy, która jest odpowiednia do problemu klasyfikacji binarnej.

metrics to lista miar, które są używane do oceny jakości modelu podczas treningu. W tym kodzie używana jest tylko jedna miarą - accuracy, czyli dokładność.

Następnie tworzona jest funkcja step_decay, która zwraca wartość stałej uczenia w zależności od numeru epoki (ang. epoch). Wartość ta maleje liniowo w czasie.

Obiekt lrate_scheduler jest deklarowany jako obiekt LearningRateScheduler z modułu keras.callbacks. Ten obiekt jest używany do zmiany wartości stałej uczenia w trakcie treningu zgodnie z zdefiniowaną funkcją step_decay.

Obiekt early_stop jest deklarowany jako obiekt EarlyStopping z modułu keras.callbacks. Ten obiekt jest używany do przerwania treningu, jeśli wynik na zbiorze walidacyjnym (ang. validation set) nie poprawia się przez określoną liczbę epok (w tym przypadku 5).

Obiekt model_chkpoint jest deklarowany jako obiekt ModelCheckpoint z modułu keras.callbacks. Ten obiekt jest używany do zapisywania najlepszego modelu (czyli modelu o najmniejszej wartości funkcji straty na zbiorze walidacyjnym) podczas treningu.

Na końcu tworzona jest lista callbacks, która zawiera obiekty lrate_scheduler, early_stop i model_chkpoint.

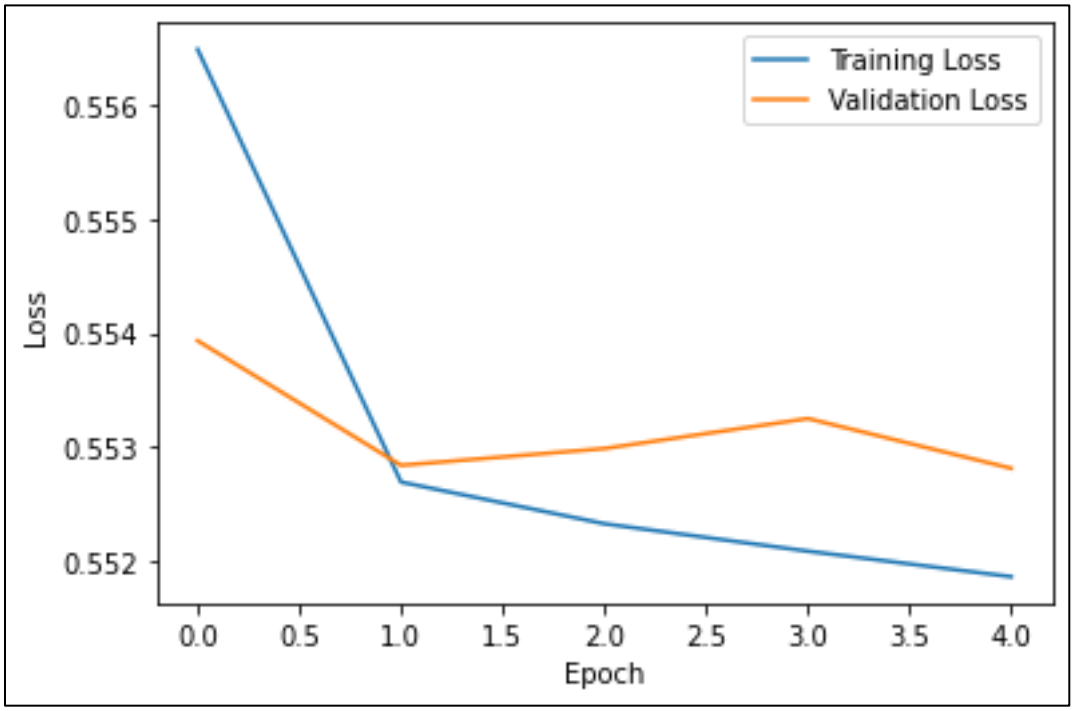
Ostatnia linia kodu używa metody fit obiektu model do rozpoczęcia treningu modelu.

Przebieg uczenia

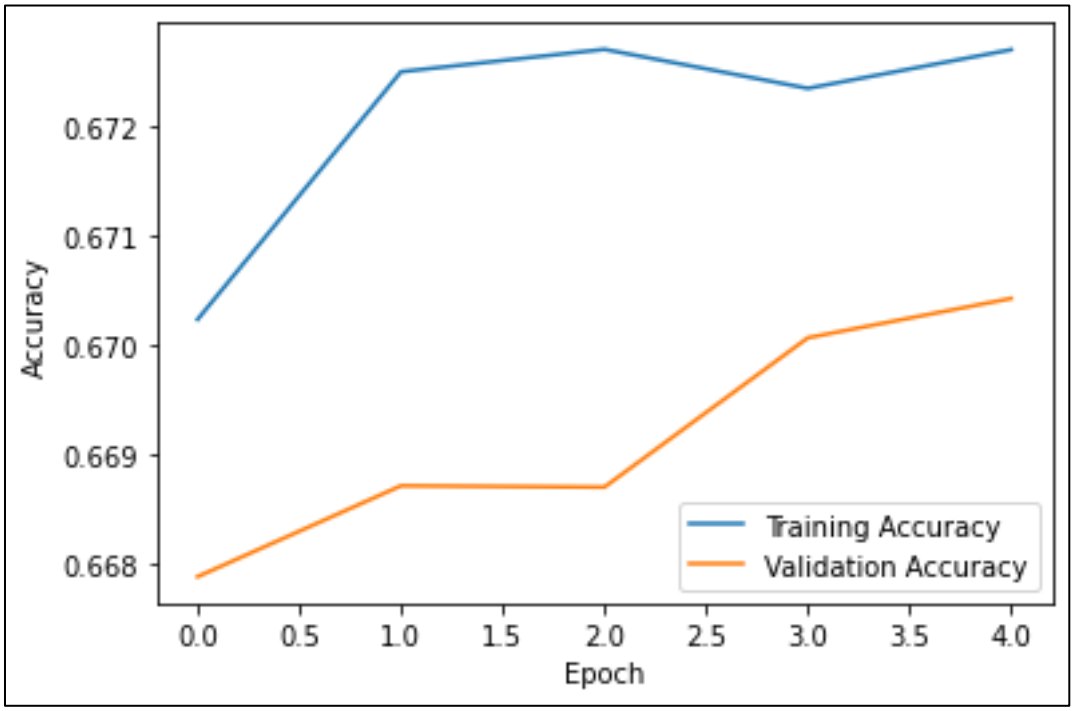
```
Epoch 1/5
2522/2527 [============================>.] - ETA: 0s - loss: 0.5565 - a
ccuracy: 0.6702
Epoch 1: val_loss improved from inf to 0.55393, saving model to best_mo
del.h5
2527/2527 [==============================] - 19s 6ms/step - loss: 0.556
5 - accuracy: 0.6702 - val_loss: 0.5539 - val_accuracy: 0.6679 - lr: 0.
0024
Epoch 2/5
2524/2527 [============================>.] - ETA: 0s - loss: 0.5527 - a
ccuracy: 0.6725
Epoch 2: val_loss improved from 0.55393 to 0.55284, saving model to bes
t_model.h5
2527/2527 [==============================] - 14s 6ms/step - loss: 0.552
7 - accuracy: 0.6725 - val_loss: 0.5528 - val_accuracy: 0.6687 - lr: 0.
0019
Epoch 3/5
2521/2527 [============================>.] - ETA: 0s - loss: 0.5524 - a
ccuracy: 0.6727
Epoch 3: val_loss did not improve from 0.55284
2527/2527 [==============================] - 14s 5ms/step - loss: 0.552
3 - accuracy: 0.6727 - val_loss: 0.5530 - val_accuracy: 0.6687 - lr: 0.
0015
Epoch 4/5
2526/2527 [============================>.] - ETA: 0s - loss: 0.5521 - a
ccuracy: 0.6723
Epoch 4: val_loss did not improve from 0.55284
2527/2527 [==============================] - 14s 6ms/step - loss: 0.552
1 - accuracy: 0.6723 - val_loss: 0.5532 - val_accuracy: 0.6701 - lr: 0.
0012
Epoch 5/5
2523/2527 [============================>.] - ETA: 0s - loss: 0.5519 - a
ccuracy: 0.6726
Epoch 5: val_loss improved from 0.55284 to 0.55281, saving model to bes
t_model.h5
2527/2527 [==============================] - 15s 6ms/step - loss: 0.551
9 - accuracy: 0.6727 - val_loss: 0.5528 - val_accuracy: 0.6704 - lr: 9.
4494e-04
```

Wyniki uczenia

Wartość funkcji straty: 0.5519
Dokładność: 0.6726



Wykres przedstawiający wartość funkcji straty w poszczególnych epokach



Wykres przedstawiający dokładność w poszczególnych epokach