МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

Институт ИТКН

Кафедра инженерной кибернетики

Направление подготовки: «01.03.04 Прикладная математика»

Квалификация: бакалавр

Группа: БПМ-17-1

**ОТЧЕТ**

**ПО КУРСОВОЙ РАБОТЕ**

**«ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ»**

на тему: «Оценка семантической схожести англоязычных текстов с использованием сиамских искусственных нейронных сетей»

**Студент** \_\_\_\_\_\_\_\_\_\_/ Ундалов Н.С.

**Руководитель** \_\_\_\_\_\_\_\_\_\_/ старший преподаватель, Кондыбаева А.Б.

**Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва 2020**

**Содержание**

[1. Введение 2](#_Toc59388532)

[2. Описание модели 3](#_Toc59388533)

[3. Реализация модели 6](#_Toc59388534)

[3.1. Используемые программные средства 7](#_Toc59388535)

[3.2. Реализация модели 7](#_Toc59388536)

[3.3. Описание датасета 8](#_Toc59388537)

[3.4. Тестирование модели 8](#_Toc59388538)

[4. Заключение 9](#_Toc59388539)

[5. Список использованных источников 10](#_Toc59388540)

[6. Приложение 11](#_Toc59388541)

# Введение

Глубокие нейронные сети в настоящее время становятся одним из самых популярных методов машинного обучения. Они показывают лучшие результаты по сравнению с альтернативными методами в таких областях, как распознавание речи, обработка естественного языка, компьютерное зрение, медицинская информатика и др. Одна из причин успешного применения глубоких нейронных сетей заключается в том, что сеть автоматически выделяет из данных важные признаки, необходимые для решения задачи.

Для обработки текстов используются рекуррентные нейронные сети, потому что они учитывают последовательную структуру текста. Данный тип нейронных сетей способен решать множество задач по обработке текста таких как классификация текстов, выделение ключевых слов, тональности и схожести текстов. В настоящей работе предложен вариант нейронной сети для оценки семантической схожести двух текстов на английском языке. Проведено обучение нейронной сети и оценка ее качества.

# Описание модели

Сеть LSTM была создана, чтобы решить проблему долгосрочных зависимостей. Разновидности данной сети хорошо показали себя во многих задачах. Одна ячейка сети состоит из нескольких компонентов.

Во-первых, в каждой ячейки есть переменная состояния. Обозначим ее как , где t – момент времени. Пусть – выход сети, – вход сети в момент времени t, тогда

где так называемый гейт забывания, [,] – операция конкатенации. Выход этого слоя будет использован для записи изменения состояния ячейки. Идея заложенная в этот механизм подразумевает необходимость забывания части данных по мере обработки новых. После идет добавление информации в ячейку состояния. Этот этап состоит из двух частей. Первая часть, ее также называют входным гейтом, отвечает за выделение необходимых элементов состояния для обновления. Вторая генерирует сами значения для обновления.

С помощью данных операций можно определить правило изменения состояния ячейки LSTM. Сначала состояние ячейки модифицируется с помощью гейта забывания, а потом с помощью входного гейта. В виде формулы это представляется так:

Для определения выхода ячейки, в данный момент, необходимо проделать еще несколько операций. Выход ячейки состоит комбинации операций над прошлым состоянием ячейки, текущим состоянием и входом. Для начала, вычисляется выход слоя, основанный на входе ячейки и прошлом состоянии. Данные вектора конкатенируются и пропускаются через сигмоиду.

Полученные значения используются как “фильтр” значений, полученных из текущего состояния. Над вектором текущего состояния производится операция гиперболического тангенса. На основании данных значений рассчитывается выход ячейки в данный момент времени.

Выше была представлена оригинальная архитектура сети LSTM. За последнее время появилось несколько вариаций архитектур, которые комбинируют гейты и немного меняют общую архитектуру сети. Рассмотрим несколько вариаций архитектур.

Например, в статье [19] авторы предлагают учитывать прошлое состояние ячейки при работе всех гейтов. Т.е. формулы вычисления и меняются на следующие:

Также есть архитектура, авторы которой руководствовались идеей, что забывать нужно только ту информацию, которая должна быть перезаписаны. Т.е. в архитектуре объединены гейты входа и забывания. Относительно первой представленной архитектуры изменяется правило обновления состояния. В данной архитектуре оно выглядит следующим образом.

Последняя разновидность архитектуры LSTM, рассматриваемая в данной работе называется GRU (gated recurrent units). Рассмотрим ее детальнее. Гейты забывания и входа представляются одним гейтом обновления. Кроме того, состояние ячейки объединяется с выходом ячейки. Такая архитектура проще, чем стандартная LSTM, и пользуется большой популярностью. Рассмотрим операции, описывающие данную модель.

Данные модели показали прекрасные результаты в свое время, и даже сейчас, есть спектр задач, где они могут соревноваться с новейшими архитектурами. Например, суммаризация текста является одной из таких задач.

# Реализация **модели**

## **3.1. Используемые программные средства**

Реализация модели в рамках данной работы производилась в облачном сервисе Google Colaboratory, на языке Python v3.8. Список используемых библиотек:

* Tensorflow – библиотека для создания нейронных сетей;
* NumPy – библиотека, необходимая для работы с векторами и массивами;
* Word2vec – веторные представления слов.

## 3.2. Реализация модели

Для достижения заданной цели работы была реализована сиамская нейронная сеть с двумя сетями LSTM. Выход с LSTM сетей конкатенируется и передается в полносвязный с функцией активации ReLu. Финайльный слой представляет из себя один нейрон с сигмоидой.

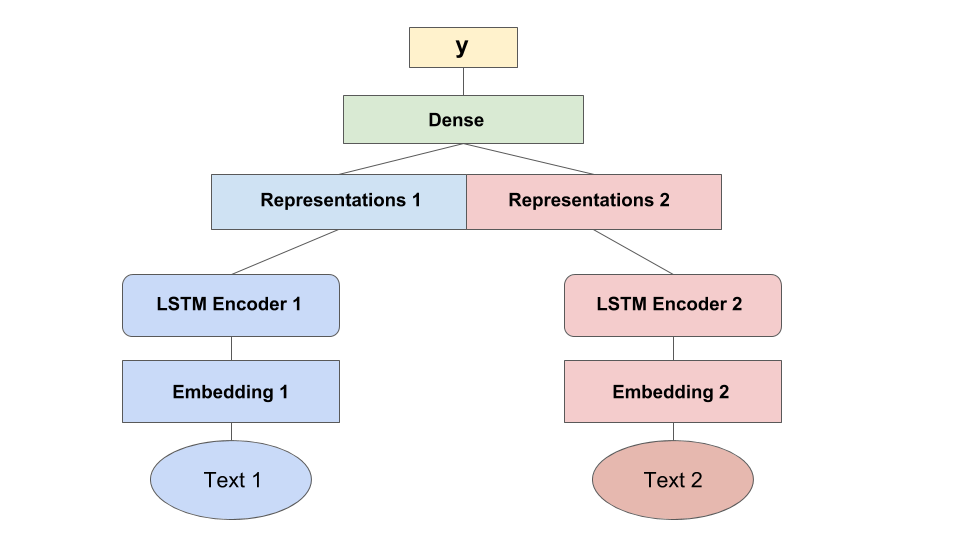


Рисунок 1. Сиамская нейронная сеть с LSTM блоками

## 3.3. Описание датасета

В качестве датасета для обучения и валидации модели используется датасет Quora Question Pairs. В датасете содержаться 404290 пары текстов и метка являются ли они дубликатами (в семантическом смысле). Распределение классов в датасете имеет сбалансированный характер.

## 3.4. Тестирование модели

Для тестирования данные датасета были предварительно разбиты на тестовую и обучающую выборки. Тестовая выборка составляет 10% от размера всего датасета. Так как датасет сбалансирован, то в качестве метрики используется метрика accurancy. По результатам тестов модель показала качество 82.7% на обучающей выборке и 81.92% на тестовой выборке в соответствие с раннее определенной метрикой. Модель немного переобучилась, но, несмотря на это, все еще достаточно хороша.

# **Заключение**

В процессе курсовой работы была изучена сиамская нейронная сеть с блоками LSTM, а также подходы работы с рекуррентными нейронными сетями. Подводя итоги, можно сделать вывод, что сиамская нейронная сеть с блоками LSTM способна решать задачу оценки семантической схожести текстов. По результатам тестов модель показала качество 82.7% на обучающей выборке и 81.92% на тестовой выборке в соответствие с раннее определенной метрикой. Модель немного переобучилась, но, несмотря на это, все еще достаточно хороша.

# **Список использованных источников**

1. Quora Question Pairs, <https://www.kaggle.com/c/quora-question-pairs>
2. LONG SHORT-TERM MEMORY, Sepp Hochreiter et al.

# Приложение

import os

import re

import csv

import codecs

import numpy as np

import pandas as pd

from nltk.corpus import stopwords

from nltk.stem import SnowballStemmer

from string import punctuation

from gensim.models import KeyedVectors

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation

from keras.layers.merge import concatenate

from keras.models import Model

from keras.layers.normalization import BatchNormalization

from keras.callbacks import EarlyStopping, ModelCheckpoint

import sys

def load\_data(df):

    question1 = df['question1'].astype(str).values

    question2 = df['question2'].astype(str).values

    # combined: to get the tokens

    df['combined'] = df['question1'] + df['question2']

    labels = df['is\_duplicate'].values

    return question1, question2, labels

BASE\_DIR = './'

EMBEDDING\_FILE = BASE\_DIR + 'GoogleNews-vectors-negative300.bin.gz'

TRAIN\_DATA\_FILE = 'drive/MyDrive/NN\_CW/quora-question-pairs/train.csv'

MAX\_SEQUENCE\_LENGTH = 30

MAX\_NB\_WORDS = 200000

EMBEDDING\_DIM = 300

VALIDATION\_SPLIT = 0.1

num\_lstm = np.random.randint(175, 275)

num\_dense = np.random.randint(100, 150)

rate\_drop\_lstm = 0.15 + np.random.rand() \* 0.25

rate\_drop\_dense = 0.15 + np.random.rand() \* 0.25

act = 'relu'

re\_weight = True # whether to re-weight classes to fit the 17.5% share in test set

STAMP = 'lstm\_%d\_%d\_%.2f\_%.2f'%(num\_lstm, num\_dense, rate\_drop\_lstm, \

        rate\_drop\_dense)

word2vec = KeyedVectors.load\_word2vec\_format(EMBEDDING\_FILE, \

        binary=True)

# The function "text\_to\_wordlist" is from

# https://www.kaggle.com/currie32/quora-question-pairs/the-importance-of-cleaning-text

def text\_to\_wordlist(text, remove\_stopwords=False, stem\_words=False):

    # Clean the text, with the option to remove stopwords and to stem words.

    # Convert words to lower case and split them

    text = text.lower().split()

    # Optionally, remove stop words

    if remove\_stopwords:

        stops = set(stopwords.words("english"))

        text = [w for w in text if not w in stops]

    text = " ".join(text)

    # Clean the text

    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)

    text = re.sub(r"what's", "what is ", text)

    text = re.sub(r"\'s", " ", text)

    text = re.sub(r"\'ve", " have ", text)

    text = re.sub(r"can't", "cannot ", text)

    text = re.sub(r"n't", " not ", text)

    text = re.sub(r"i'm", "i am ", text)

    text = re.sub(r"\'re", " are ", text)

    text = re.sub(r"\'d", " would ", text)

    text = re.sub(r"\'ll", " will ", text)

    text = re.sub(r",", " ", text)

    text = re.sub(r"\.", " ", text)

    text = re.sub(r"!", " ! ", text)

    text = re.sub(r"\/", " ", text)

    text = re.sub(r"\^", " ^ ", text)

    text = re.sub(r"\+", " + ", text)

    text = re.sub(r"\-", " - ", text)

    text = re.sub(r"\=", " = ", text)

    text = re.sub(r"'", " ", text)

    text = re.sub(r"(\d+)(k)", r"\g<1>000", text)

    text = re.sub(r":", " : ", text)

    text = re.sub(r" e g ", " eg ", text)

    text = re.sub(r" b g ", " bg ", text)

    text = re.sub(r" u s ", " american ", text)

    text = re.sub(r"\0s", "0", text)

    text = re.sub(r" 9 11 ", "911", text)

    text = re.sub(r"e - mail", "email", text)

    text = re.sub(r"j k", "jk", text)

    text = re.sub(r"\s{2,}", " ", text)

    # Optionally, shorten words to their stems

    if stem\_words:

        text = text.split()

        stemmer = SnowballStemmer('english')

        stemmed\_words = [stemmer.stem(word) for word in text]

        text = " ".join(stemmed\_words)

    # Return a list of words

    return(text)

data = pd.read\_csv(TRAIN\_DATA\_FILE)

texts\_1, texts\_2, labels = load\_data(data)

print('Found %s texts in train.csv' % len(texts\_1))

tokenizer = Tokenizer(num\_words=MAX\_NB\_WORDS)

tokenizer.fit\_on\_texts(texts\_1 + texts\_2)

sequences\_1 = tokenizer.texts\_to\_sequences(texts\_1)

sequences\_2 = tokenizer.texts\_to\_sequences(texts\_2)

word\_index = tokenizer.word\_index

print('Found %s unique tokens' % len(word\_index))

data\_1 = pad\_sequences(sequences\_1, maxlen=MAX\_SEQUENCE\_LENGTH)

data\_2 = pad\_sequences(sequences\_2, maxlen=MAX\_SEQUENCE\_LENGTH)

labels = np.array(labels)

print('Shape of data tensor:', data\_1.shape)

print('Shape of label tensor:', labels.shape)

# prepare embeddings

print('Preparing embedding matrix')

nb\_words = min(MAX\_NB\_WORDS, len(word\_index))+1

embedding\_matrix = np.zeros((nb\_words, EMBEDDING\_DIM))

for word, i in word\_index.items():

    if word in word2vec.vocab:

        embedding\_matrix[i] = word2vec.word\_vec(word)

print('Null word embeddings: %d' % np.sum(np.sum(embedding\_matrix, axis=1) == 0))

#sample train/validation data

#np.random.seed(1234)

perm = np.random.permutation(len(data\_1))

idx\_train = perm[:int(len(data\_1)\*(1-VALIDATION\_SPLIT))]

idx\_val = perm[int(len(data\_1)\*(1-VALIDATION\_SPLIT)):]

data\_1\_train = np.vstack((data\_1[idx\_train], data\_2[idx\_train]))

data\_2\_train = np.vstack((data\_2[idx\_train], data\_1[idx\_train]))

labels\_train = np.concatenate((labels[idx\_train], labels[idx\_train]))

data\_1\_val = np.vstack((data\_1[idx\_val], data\_2[idx\_val]))

data\_2\_val = np.vstack((data\_2[idx\_val], data\_1[idx\_val]))

labels\_val = np.concatenate((labels[idx\_val], labels[idx\_val]))

weight\_val = np.ones(len(labels\_val))

if re\_weight:

    weight\_val \*= 0.472001959

    weight\_val[labels\_val==0] = 1.309028344

# define the model structure

embedding\_layer = Embedding(nb\_words,

        EMBEDDING\_DIM,

        weights=[embedding\_matrix],

        input\_length=MAX\_SEQUENCE\_LENGTH,

        trainable=False)

lstm\_layer = LSTM(num\_lstm, dropout=rate\_drop\_lstm, recurrent\_dropout=rate\_drop\_lstm)

sequence\_1\_input = Input(shape=(MAX\_SEQUENCE\_LENGTH,), dtype='int32')

embedded\_sequences\_1 = embedding\_layer(sequence\_1\_input)

x1 = lstm\_layer(embedded\_sequences\_1)

sequence\_2\_input = Input(shape=(MAX\_SEQUENCE\_LENGTH,), dtype='int32')

embedded\_sequences\_2 = embedding\_layer(sequence\_2\_input)

y1 = lstm\_layer(embedded\_sequences\_2)

merged = concatenate([x1, y1])

merged = Dropout(rate\_drop\_dense)(merged)

merged = BatchNormalization()(merged)

merged = Dense(num\_dense, activation=act)(merged)

merged = Dropout(rate\_drop\_dense)(merged)

merged = BatchNormalization()(merged)

preds = Dense(1, activation='sigmoid')(merged)

if re\_weight:

    class\_weight = {0: 1.309028344, 1: 0.472001959}

else:

    class\_weight = None

# train the model

model = Model(inputs=[sequence\_1\_input, sequence\_2\_input], \

        outputs=preds)

model.compile(loss='binary\_crossentropy',

        optimizer='nadam',

        metrics=['acc'])

#model.summary()

print(STAMP)

early\_stopping =EarlyStopping(monitor='val\_loss', patience=3)

bst\_model\_path = STAMP + '.h5'

model\_checkpoint = ModelCheckpoint(bst\_model\_path, save\_best\_only=True, save\_weights\_only=True)

hist = model.fit([data\_1\_train, data\_2\_train], labels\_train, \

        validation\_data=([data\_1\_val, data\_2\_val], labels\_val, weight\_val), \

        epochs=200, batch\_size=2048, shuffle=True, \

        class\_weight=class\_weight, callbacks=[early\_stopping, model\_checkpoint])

model.load\_weights(bst\_model\_path)

bst\_val\_score = min(hist.history['val\_loss'])