МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

по дисциплине «Искусственные нейронные сети»

Тема: Sentiment140

Студентка гр. 8382 _____ Кузина А.М.

Студентка гр. 8382 _____ Кулачкова М.К.

Студент гр. 8382 _____ Янкин Д.О.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

СОДЕРЖАНИЕ

ЗАДАНИЕ

Дан датасет текста, который включает в себя набор твитов разной эмоциональной окраски, а также информацию о каждом твите. Задача заключается в определении того, насколько положительным или негативным является текст в твите.

Необходимо разработать модель нейронной сети, решающей поставленную задачу. Желательно использование ансамбля моделей. В отчете необходимо привести описание и краткий анализ датасета, а также отразить весь процесс разработки.

Исходный код программы представлен в приложении А.

Зоны ответственности между членами бригады не были распределены точно – каждый участник команды принимал участие в разработке и тестировании всей программы.

1. ОБРАБОТКА ДАННЫХ

1.1. Описание датасета

В работе используется датасет Sentiment140, состоящий из 1600000 тренировочных примеров и 498 тестовых примеров твитов различной эмоциональной окраски. Данные представлены в следующем формате:

0 – эмоциональная окраска твита (0 = негативный, 2 = нейтральный, 4 = позитивный)

1 – идентификатор твита

2 – дата написания твита

3 – запрос

4 – автор твита

5 – текст твита

В таблице 1 приведен фрагмент тренировочного датасета.

Таблица 1 – Фрагмент тренировочного датасета

0	1467811594	Mon Apr 06 22:20:03 PDT 2009	NO_QUERY	coZZ	@LOLTrish hey long time no see! Yes.. Rains a bit ,only a bit LOL , I'm fine thanks , how's you ?
0	1467811795	Mon Apr 06 22:20:05 PDT 2009	NO_QUERY	2Hood4Hollywood	@Tatiana_K nope they didn't have it
0	1467812025	Mon Apr 06 22:20:09 PDT 2009	NO_QUERY	mimismo	@twittera que me muera ?
0	1467812416	Mon Apr 06 22:20:16 PDT 2009	NO_QUERY	erinx3leannexo	spring break in plain city... it's snowing
0	1467812579	Mon Apr 06 22:20:17 PDT 2009	NO_QUERY	pardonlauren	I just re-pierced my ears

Мы будем осуществлять оценку эмоциональной окраски только на основании текста, не учитывая другие признаки, поэтому будем работать только с полями 0 и 5.

В процессе выполнения работы было замечено, что тренировочный датасет не содержит нейтральных твитов, а в тестовом их всего около 100. Из-за этого при классификации нейтральных твитов модель часто ошибалась, и было решено использовать и для обучения, и для тестирования модели

тренировочный датасет. Подробнее о возникшей проблеме написано в разделе 3.

1.2. Обработка текста

Тексты твитов необходимо привести к виду, пригодному для обработки нейросетью.

```
def process_data(series):
    pat = r"@.[^\s]+|https?:\/\/.[^\s]+"
    data = series.str.lower()
    data = data.str.replace(pat, "")
    data = data.str.replace(r":\)|: \)|:-\)|;-
\)|:d|:p|;\)|;d|;p|=\)", " yppahelims ")
    data = data.str.replace(r":\(|: \(|:-\(", " daselims ")
    data = data.str.replace(r":0|:o", " esirpruselims ")
    data = data.str.replace(r":@", " taelims ")
    return data
```

Сначала все буквы приводятся к нижнему регистру, чтобы сократить объем словаря. Затем из твитов удаляются упоминания других пользователей («@nickname») и гиперссылки («http://…» или «https://…»), так как они не несут эмоциональной нагрузки. В некоторых твитах содержатся эмотиконы, которые важны для определения эмоциональной окраски текста. Чтобы не потерять их при удалении знаков препинания, они заменяются наборами буквенных символов, которые получаются реверсом строки с описанием эмотикона. Так предполагается избежать совпадений с реальными словами. Поскольку эмотиконов много, они были разделены на группы по передаваемым эмоциям (см. таблицу 2).

Таблица 2 – Кодирование эмотиконов

Эмотиконы	**Замена**
:), :), :-), ;-), :d, :p, ;), ;d, ;p, =)	yppahelims
:(, : (, :-(	daselims
:0, :o	esirpruselims
:@	taelims

Представление слов в виде чисел осуществляется с помощью класса Tokenizer библиотеки Keras. Сначала составляется словарь, переводящий слова в числа, причем чем чаще встречается слово, тем меньше его индекс. При этом учитываются только `max_words = 30000` наиболее часто встречающихся в датасете слов. Затем осуществляется непосредственно кодирование слов в датасете. Полученные последовательности числовых индексов дополняются нулями или обрезаются так, чтобы все последовательности оказались одной длины – `max_length = 50` слов.

```
max_words = 30000
tokenizer = keras.preprocessing.text.Tokenizer(num_words=max_words)

tokenizer.fit_on_texts(train_data)
train_data = tokenizer.texts_to_sequences(train_data)

max_length = 50
train_data = sequence.pad_sequences(train_data, maxlen=max_length)
```

Так как изначально выходные данные представлены числами 0 и 4, они нормализуются – делятся на 4, – чтобы получились значения 0 и 1.

Исходный датасет делится на обучающую и тестовую выборки, на которые отводятся соответственно 90% и 10% всех данных. Полученные выборки готовы к обработке нейросетью.

2. ПОСТРОЕНИЕ МОДЕЛЕЙ

2.1. Модель первого типа

Рассмотрим модель первого типа:

```
def get_model_type_one(top_words, length):
    embedding_vector_length = 32
    model_one = Sequential()
    model_one.add(layers.Embedding(top_words, embedding_vector_length,
 input_length=length))
    model_one.add(layers.Flatten())
    model_one.add(layers.Dense(200, activation='relu'))
    model_one.add(layers.Dropout(0.3))
    model_one.add(layers.Dense(200, activation='relu'))
    model_one.add(layers.Dropout(0.4))
    model_one.add(layers.Dense(1, activation='sigmoid'))
    model_one.compile(loss='binary_crossentropy', optimizer='adam', me
trics=['accuracy'])
    return model_one
```

Данная модель состоит из слоя `Embedding` – необходим для составления языковой модели, представления слов в виде векторов, слоя `Flatten` – нужного для изменения формы данных в вектор, двух полносвязных слоев `Dense` с 200 нейронами на каждом и функцией активации relu и промежуточными слоями `Dropout`, для борьбы с переобучением сети, и заключающим слоем `Dense` с одним выходным нейроном и функцией активации sigmoid. Затем модель компилируется с оптимизатором Адам, метрикой точность и функцией потерь – бинарная кроссэнтропия.

2.2. Модель второго типа

Рассмотрим модель второго типа:

```
def get_model_type_two(num, length):
    embedding_vector_length = 32
    model_two = Sequential()
    model_two.add(layers.Embedding(num, embedding_vector_length, input
_length=length))
    model_two.add(layers.Conv1D(filters=32, kernel_size=3, padding='sa
me', activation='relu'))
    model_two.add(layers.MaxPooling1D(pool_size=2))
    model_two.add(layers.Dropout(0.2))
    model_two.add(layers.LSTM(200, return_sequences=True))
```

```
    model_two.add(layers.Dropout(0.2))
    model_two.add(layers.LSTM(200))
    model_two.add(layers.Dropout(0.2))
    model_two.add(layers.Dense(1, activation='sigmoid'))
    model_two.compile(loss='binary_crossentropy', optimizer='adam', me
trics=['accuracy'])
    return model_two
```

Данная модель состоит из слоя `Embedding` – необходим для составления языковой модели, представления слов в виде векторов, слоя свертки `Conv1D` – нужного для выделения главных признаков в векторе данных, слоя субдискретизации `MaxPooling1D` – необходимого для сокращения количества параметров модели, двух рекуррентные слоев `LSTM` с 200 умными нейронами на каждом и тремя промежуточными слоями `Dropout`, для борьбы с переобучением сети, и заключающим слоем `Dense` с одним выходным нейроном и функцией активации sigmoid. Затем модель компилируется с оптимизатором Адам, метрикой точность и функцией потерь – бинарная кроссэнтропия.

2.3. Модель третьего типа

Рассмотрим модель третьего типа:

```
def get_model_type_three(num, length):
    embedding_vector_length = 32
    model_three = Sequential()
    model_three.add(layers.Embedding(num, embedding_vector_length, inp
ut_length=length))
    model_three.add(layers.LSTM(300, return_sequences=True))
    model_three.add(layers.Dropout(0.2))
    model_three.add(layers.LSTM(300))
    model_three.add(layers.Dropout(0.3))
    model_three.add(layers.Dense(1, activation='sigmoid'))
    model_three.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model_three
```

Данная модель состоит из слоя `Embedding` – необходим для составления языковой модели, представления слов в виде векторов, двух рекуррентные слоев `LSTM` с 300 умными нейронами на каждом и двумя промежуточными

слоями `Dropout`, для борьбы с переобучением сети, и заключающим слоем `Dense` с одним выходным нейроном и функцией активации sigmoid. Затем модель компилируется с оптимизатором Адам, метрикой точность и функцией потерь – бинарная кроссэнтропия.

2.4. Ансамблирование

Для того, чтобы получить большую точность классификации данных, используется ансамбль из трех моделей, по одной каждого из перечисленных типов. Модели обучаются на разных подвыборках тренировочного датасета для внесения в ансамбль большей разнообразности.

Предсказание ансамбля вычисляется как среднее арифметическое предсказаний моделей. Ниже приведен код функции, вычисляющей предсказание ансамбля.

```
def get_ensemble_predictions(models, sequence, round = True):
  predictions = []
  for model in models:
    curr_prediction = model.predict(sequence)
    predictions.append(curr_prediction)
  predictions = np.asarray(predictions)
  predictions = np.mean(predictions, 0)
  if round:
    predictions = np.round(predictions)
  return predictions.flatten()
```

3. АНАЛИЗ РЕЗУЛЬТАТОВ

3.1. Callbacks

В работе использовано три Callback’а для контроля процесса обучения: EarlyStopping, TensorBoard, а также пользовательский callback, осуществляющий классификацию случайно выбранного из тестовой выборки твита раз в заданное число эпох и выводящий текст твита, предсказание и ожидаемый выход на экран.

Callback EarlyStopping, определенный в библиотеке Keras, отслеживает точность предсказания модели на валидационном множестве, и останавливает обучение, если она не увеличивается в течение одной эпохи. При остановке обучения весам модели присваиваются значения, использованные при получении наилучшего результата.

С помощью callback’а TensorBoard строятся графики точности и ошибок моделей в процессе обучения (приведены в разделе 3.2.).

Код пользовательского callback’а приведен ниже.

```
class RandomPredictionCb(keras.callbacks.Callback):
    def __init__(self, interval, data, labels, tokenizer):
        super(RandomPredictionCb, self).__init__()
        self.interval = interval
        self.data = data
        self.labels = labels
        self.tokenizer = tokenizer

    def on_epoch_end(self, epoch, logs=None):
        if epoch % self.interval == 0 or epoch == self.params["epochs"
] - 1:
            rn = np.random.randint(0, len(self.data) - 1)
            prediction = self.model.predict(np.asarray([self.data[rn]]
))
            txt = self.tokenizer.sequences_to_texts([self.data[rn]])
            print("Text: ", txt)
            print("Prediction: ", prediction)
            print("Real value: ", self.labels[rn].flatten())
```

Callback использует определенный при обработке данных токенизатор для восстановления текста твита по последовательности чисел, которая

обрабатывается нейросетью. При этом некоторые слова могут быть потеряны, так как не попали в словарь. В таблицах 3-5 приведены выводы этого callback’а в процессе обучения моделей.

Таблица 3 – Вывод пользовательского callback’а для модели 1

Текст	Предсказание	Реальный результат
internet is severly sucky today not sure but i think it's on end	0.2443317	0
i no im soooo cut	0.04913864	0
shame its sydney based	0.08958885	0

Таблица 4 – Вывод пользовательского callback’а для модели 2

Текст	Предсказание	Реальный результат
thank you for saving me from crappy dns server my internet is back up to par finally	0.5905723	1
lol the magic gonna take it next game they lost tonight so they can win it at home	0.4629443	1
watching the curious case of button really good movie	0.9938682	1

Таблица 5 – Вывод пользовательского callback’а для модели 3

Текст	Предсказание	Реальный результат
i guess i'll leave it hooked up to the computer overnight i'm so sad right now i lost all of my stuff	0.05188182	0
well that's good at least i hope you do well then you'll have to talk to me in july to tell me your score	0.93187505	1
good morning	0.9923798	1
still recovering from the accident headed for arkansas on friday for service so sad	0.00208088	0

Как видно из таблиц, большинство твитов в процессе обучения классифицируются правильно.

3.2. Оценка точности

В процессе выполнения работы были рассмотрены разные варианты работы с данными. При текущей конфигурации были получены наибольшие значения точности на тестовой выборке.

В первых версиях программы обучающие данные не перемешивались, из-за чего первая модель ансамбля обучалась только на негативных твитах, а последняя – только на позитивных. Точность каждой из моделей при этом составляла около 35%.

Также изначально в качестве тестового множества использовался тестовый датасет, содержащий не только позитивные и негативные, но и нейтральные твиты. Точность моделей на этом множестве, вычисляемая обычным способом (методом `model.evaluate()`), составляла около 50%, так как модель просто не предсказывала нейтральные отзывы. Была написана собственная функция для оценки точности, которая классифицировала как нейтральные те отзывы, для которых модель предсказывала значение в диапазоне от 0.4 до 0.6. Код функции приведен ниже.

```
def evaluate_model(model, x_data, y_data):
    predictions = model.predict(x_data)
    predictions_rounded = []
    for p in predictions:
        if p <= 0.4:
            predictions_rounded.append(0.0)
        elif p >= 0.6:
            predictions_rounded.append(1.0)
        else:
            predictions_rounded.append(0.5)
    accuracy = predictions_rounded == y_data
    return np.count_nonzero(accuracy)/y_data.shape[0]
```

При этом считалось, что модели решают не задачу классификации, а задачу регрессии между значениями 0 и 1, и в качестве функции потерь модели использовали MSE. При использовании приведенной выше функции для оценки качества предсказаний модели точность все равно не превышала 60%. В связи с этим было принято решение работать только с тренировочным исходным датасетом, содержащим только позитивные и негативные твиты.

Для оценки точности ансамбля используется следующая функция:

```
def evaluate_ensemble(models, x_data, y_data):
    predictions = get_ensemble_predictions(models, x_data)
    accuracy = predictions == y_data
    return np.count_nonzero(accuracy)/y_data.shape[0]
```

Предсказания ансамбля, вычисляемые как округленное до целого среднее арифметическое предсказаний всех моделей, тут сравниваются с реальными значениями выходных данных и подсчитывается доля совпадений.

Полученная точность:

Модель 1 – 80.23%

Модель 2 – 81.53%

Модель 3 – 81.64%

Ансамбль – 82.35%

Точность ансамбля выше точностей отдельных моделей, что говорит о целесообразности его использования.

Графики точности и ошибок в процессе обучения, полученные с помощью TensorBoard, изображены на рисунках 1-6. Красная линия – обучающие данные, синяя – проверочные.

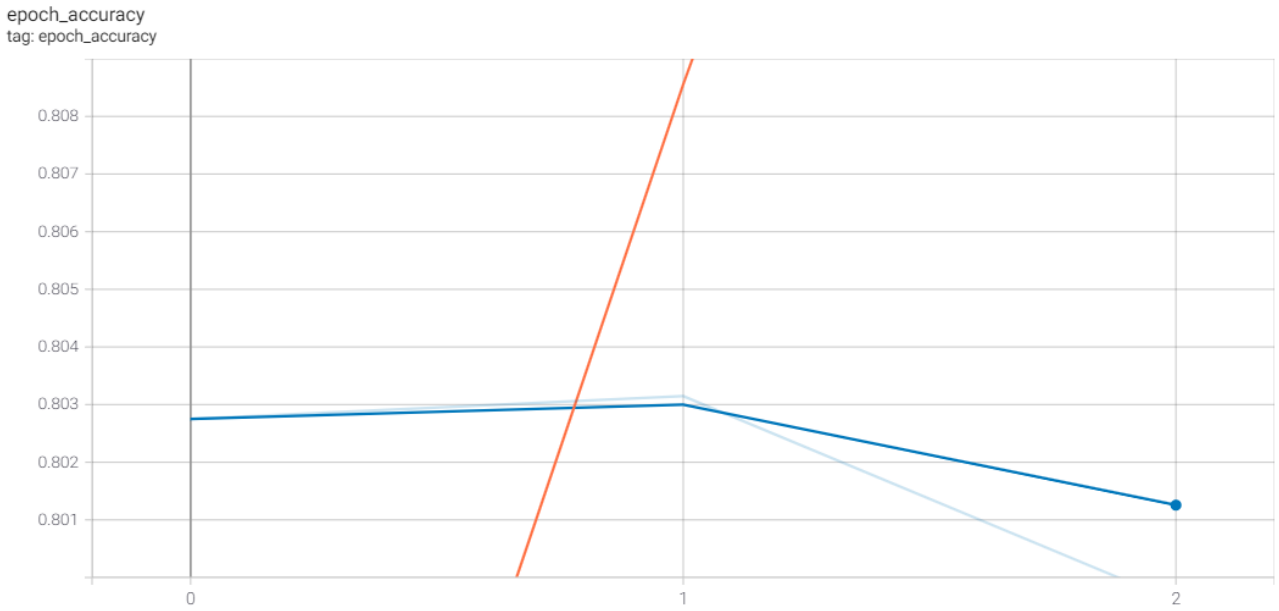

Рисунок 1. График точности первой модели

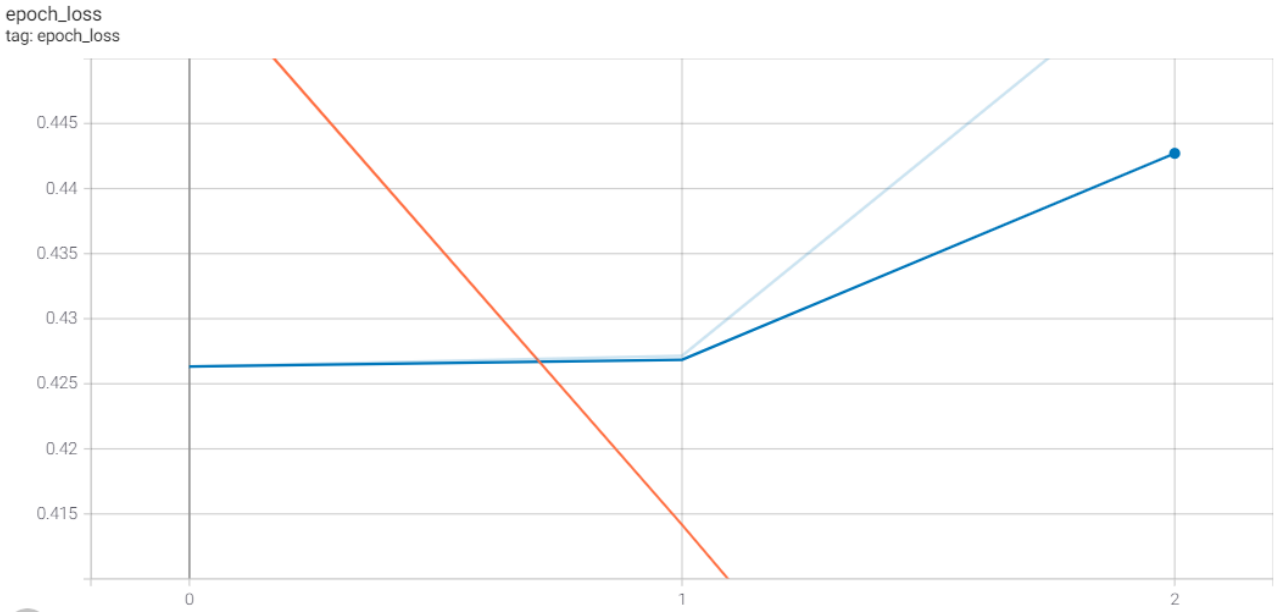

Рисунок 2. График ошибок первой модели

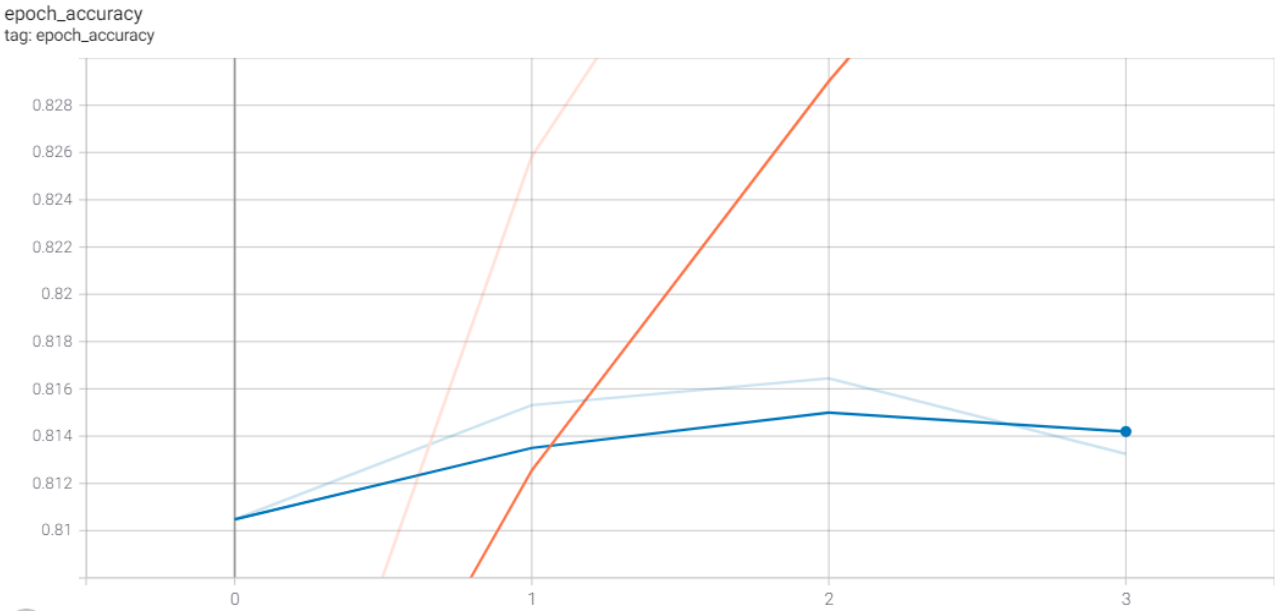

Рисунок 3. График точности второй модели

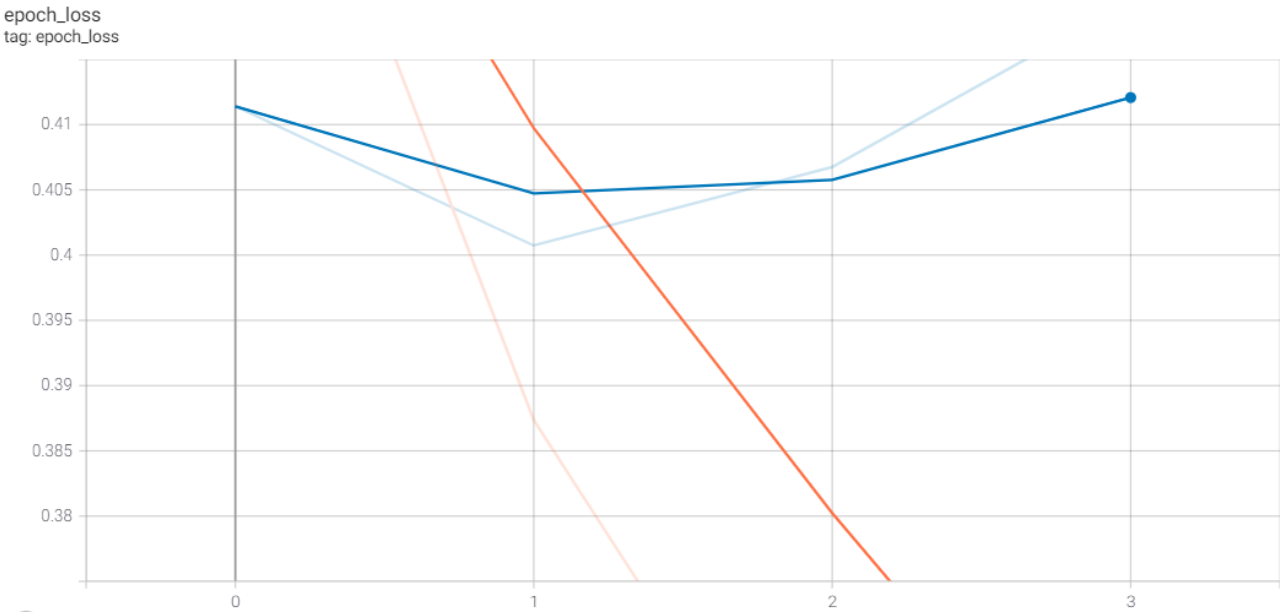

Рисунок 4. График ошибок второй модели

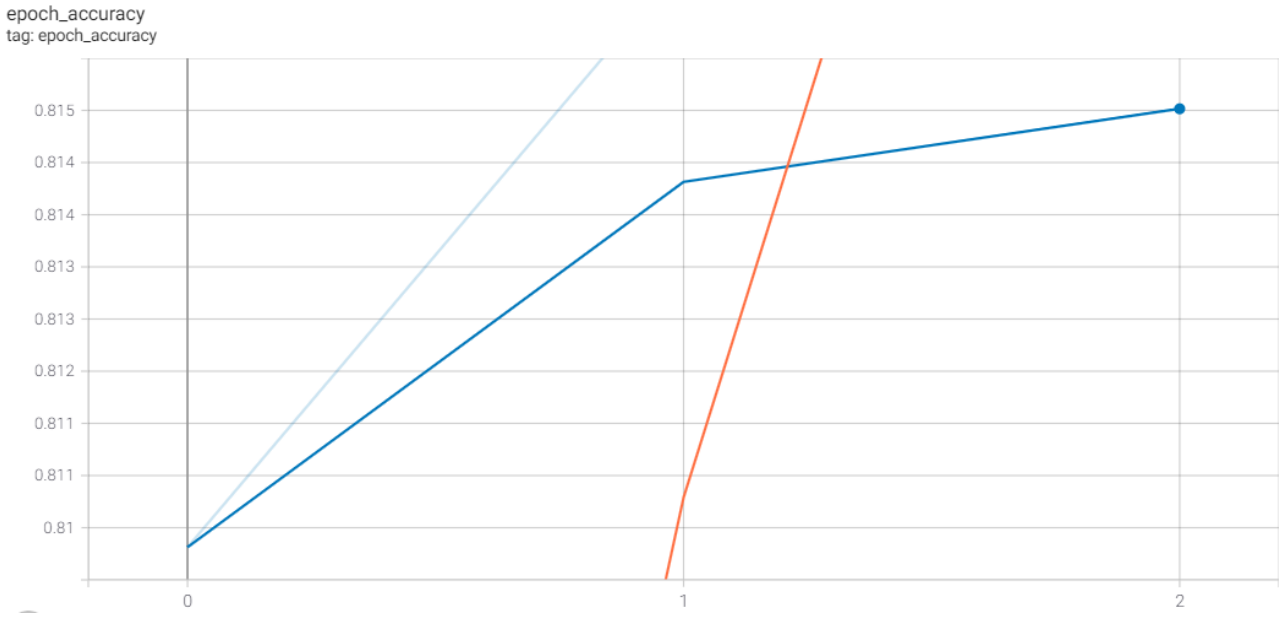

Рисунок 5. График точности третьей модели

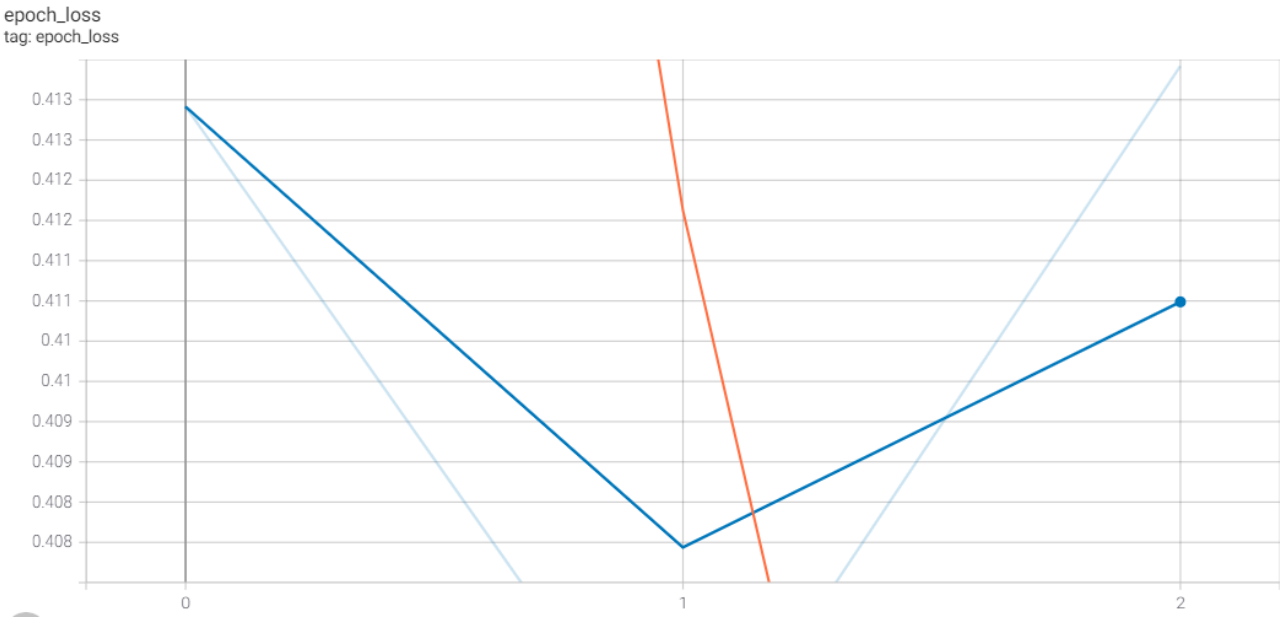

Рисунок 6. График ошибок третьей модели

3.3. Проблемы модели

Основной проблемой решаемой задачи, как уже упоминалось, являются нейтральные твиты. Из 1.6 млн образцов таких твитов всего около 100, что недостаточно для того, чтобы модель научилась их распознавать.

Другой проблемой является специфика языка, используемого в твитах. Во-первых, многие слова пишутся с ошибками (иногда специально), а некоторые слова вообще представляют собой случайный набор букв, набранный в порыве эмоций. Скорее всего, такие слова не попали в словарь в силу того, что они редко встречаются, однако они хорошо отражают эмоциональную окраску. Во-вторых, зачастую для передачи эмоций используются заглавные буквы, которые также не учитываются, так как весь текст приводится к нижнему регистру. В-третьих, многие люди в твиттере используют сарказм, который нейронной сетью уловить трудно.

Еще одной проблемой является изначальная разметка данных. При прочтении некоторых твитов нам показалось, что приведенная в датасете эмоциональная оценка является не совсем корректной. Например, твиты «@LettyA ahh ive always wanted to see rent love the soundtrack!!», «almost

bedtime» и «dierks bentley is comin' to columbus, OH!! i wanna go so bad» оценены как негативные, хотя в самом тексте этих твитов нет ничего, что бы очевидно указывало на негативные эмоции. Также твит, состоящий из одного слова «death», оценивается как менее негативный, чем твит, состоящий из слов «Farrah Fawcett».

ЗАКЛЮЧЕНИЕ

Была написана программа, осуществляющая сентимент-анализ твитов из датасета Sentiment140. Тексты твитов проходили предварительную обработку: из них удалялись гиперссылки, упоминания других пользователей, эмотиконы заменялись набором буквенных символов, удалялись знаки препинания.

Для классификации твитов использовался ансамбль из трех моделей: модели прямого распространения, модели со слоем одномерной свертки для предварительного выделения признаков и с двумя LSTM-слоями по 200 нейронов каждый, а также модели из двух LSTM-слоев по 300 нейронов. Для контроля процесса обучения моделей используются колбэки EarlyStopping, TensorBoard и пользовательский колбэк, выводящий предсказание модели для случайного элемента тестовой выборки после каждой эпохи обучения.

Полученный ансамбль не достигает очень высокой точности из-за проблем, связанных со спецификой языка, используемого пользователями твиттера. Также вызывает сомнения корректность классификации твитов в исходном датасете.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import numpy as np
import re
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.preprocessing import sequence
import pandas
from sklearn.utils import shuffle


# Callback, делающий предсказания для случайного твита в после каждой эпохи
class RandomPredictionCb(keras.callbacks.Callback):
    def __init__(self, interval, data, labels, tokenizer):
        super(RandomPredictionCb, self).__init__()
        self.interval = interval
        self.data = data
        self.labels = labels
        self.tokenizer = tokenizer

    def on_epoch_end(self, epoch, logs=None):
        if epoch % self.interval == 0 or epoch == self.params["epochs"] - 1:
            rn = np.random.randint(0, len(self.data) - 1)
            prediction = self.model.predict(np.asarray([self.data[rn]]))
            txt = self.tokenizer.sequences_to_texts([self.data[rn]])
            print("Text: ", txt)
            print("Prediction: ", prediction)
            print("Real value: ", self.labels[rn].flatten())


# Вычисление предсказаний ансамбля
def get_ensemble_predictions(models, sequence, round=True):
    predictions = []
    for model in models:
        curr_prediction = model.predict(sequence)
        predictions.append(curr_prediction)
    predictions = np.asarray(predictions)
    predictions = np.mean(predictions, 0)
    if round:
        predictions = np.round(predictions)
    return predictions.flatten()


# Оценка точности предсказаний ансамбля
def evaluate_ensemble(models, x_data, y_data):
    predictions = get_ensemble_predictions(models, x_data)
    accuracy = predictions == y_data
```

```
    return np.count_nonzero(accuracy)/y_data.shape[0]


# Генератор модели 1-го типа
def get_model_type_one(top_words, length):
    embedding_vector_length = 32
    model_one = Sequential()
    model_one.add(layers.Embedding(top_words, embedding_vector_length, input
_length=length))
    model_one.add(layers.Flatten())
    model_one.add(layers.Dense(200, activation='relu'))
    model_one.add(layers.Dropout(0.3))
    model_one.add(layers.Dense(200, activation='relu'))
    model_one.add(layers.Dropout(0.4))
    model_one.add(layers.Dense(1, activation='sigmoid'))
    model_one.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
    return model_one


# Генератор модели 2-го типа
def get_model_type_two(num, length):
    embedding_vector_length = 32
    model_two = Sequential()
    model_two.add(layers.Embedding(num, embedding_vector_length, input_lengt
h=length))
    model_two.add(layers.Conv1D(filters=32, kernel_size=3, padding='same', a
ctivation='relu'))
    model_two.add(layers.MaxPooling1D(pool_size=2))
    model_two.add(layers.Dropout(0.2))
    model_two.add(layers.LSTM(200, return_sequences=True))
    model_two.add(layers.Dropout(0.2))
    model_two.add(layers.LSTM(200))
    model_two.add(layers.Dropout(0.2))
    model_two.add(layers.Dense(1, activation='sigmoid'))
    model_two.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
    return model_two


# Генератор модели 3-го типа
def get_model_type_three(num, length):
    embedding_vector_length = 32
    model_three = Sequential()
    model_three.add(layers.Embedding(num, embedding_vector_length, input_len
gth=length))
    model_three.add(layers.LSTM(300, return_sequences=True))
    model_three.add(layers.Dropout(0.2))
    model_three.add(layers.LSTM(300))
    model_three.add(layers.Dropout(0.3))
    model_three.add(layers.Dense(1, activation='sigmoid'))
```

```
    model_three.compile(loss='binary_crossentropy', optimizer='adam', metric
s=['accuracy'])
    return model_three


# Обработка текста твитов
def process_data(series):
    pat = r"@.[^\s]+|https?:\/\/.[^\s]+"
    data = series.str.lower()
    data = data.str.replace(pat, "")
    data = data.str.replace(r":\)|: \)|:-\)|;-
\)|:d|:p|;\)|;d|;p|=\)", " yppahelims ")
    data = data.str.replace(r":\(|: \(|:-\(", " daselims ")
    data = data.str.replace(r":0|:o", " esirpruselims ")
    data = data.str.replace(r":@", " taelims ")
    return data


# Функция для классификации пользовательского текста
def dialog(models, tokenizer, max_length):
    while True:
        data = input('Your text: ')
        if data == 'exit':
            break
        data = process_data(pandas.Series([data]))
        data = tokenizer.texts_to_sequences(data)
        data = np.asarray(data)
        data = sequence.pad_sequences(data, maxlen=max_length)
        result = get_ensemble_predictions(models, np.asarray(data), round=Fa
lse)
        print('Prediction: ', result)
        print('\n')


# Загрузка датасета
train_path = "training.csv"
train_df = pandas.read_csv(train_path, header=None, encoding='latin-1')

# Обработка датасета
train_df = shuffle(train_df)
train_data = process_data(train_df[5])
train_data = np.asarray(train_data)
train_labels = np.asarray(train_df[0])

# Кодирование твитов
max_words = 30000
tokenizer = keras.preprocessing.text.Tokenizer(num_words=max_words)

tokenizer.fit_on_texts(train_data)
train_data = tokenizer.texts_to_sequences(train_data)
```

```
max_length = 50
train_data = sequence.pad_sequences(train_data, maxlen=max_length)

# Нормализация выходных данных
train_labels = train_labels/4

# Разбиение выборки на тренировочную и тестовую
test_num = train_data.shape[0] // 10
test_data = train_data[:test_num]
test_labels = train_labels[:test_num]
train_data = train_data[test_num:]
train_labels = train_labels[test_num:]


print("Here we go again")

# Построение моделей
all_models = [
    get_model_type_one(max_words, max_length),
    get_model_type_two(max_words, max_length),
    get_model_type_three(max_words, max_length)
]
k = len(all_models)
train_batch_len = len(train_labels) // k

# Обучение моделей и оценка их точности
for i in range(k):
    train_data_k = train_data[i*train_batch_len:(i+1)*train_batch_len]
    train_labels_k = train_labels[i*train_batch_len:(i+1)*train_batch_len]

    earlystopping_cb = keras.callbacks.EarlyStopping(monitor="val_accuracy",
 mode="max",
                                                     patience=1, restore_bes
t_weights=True)
    tensorboard_cb = keras.callbacks.TensorBoard(log_dir=f'logs/{i}', histog
ram_freq=1)
    randomprediction_cb = RandomPredictionCb(1, test_data, test_labels, toke
nizer)
    callback_list = [earlystopping_cb, tensorboard_cb, randomprediction_cb]

    all_models[i].fit(train_data_k, train_labels_k, validation_split=0.1,
                      epochs=10, batch_size=128, callbacks=callback_list)
    scores = all_models[i].evaluate(test_data, test_labels, verbose=1)
    print("Accuracy: %.2f%%" % (scores[1]*100))
    print("\n")

# Вывод точности ансамбля
print("Ensemble accuracy: %.2f%%" % (evaluate_ensemble(all_models, test_data
, test_labels) * 100))
```

```
# Тестирование модели на пользовательском тексте
dialog(all_models, tokenizer, max_length)
```