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

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Искусственные нейронные сети»

Тема: «Прогноз успеха фильмов по обзорам»

Студентка гр. 7381 _____ Алясова А.Н.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Прогноз успеха фильмов по обзорам (Predict Sentiment From Movie Reviews).

Задачи.

- Ознакомиться с задачей регрессии
- Изучить способы представления текста для передачи в ИНС
- Достигнуть точность прогноза не менее 95%

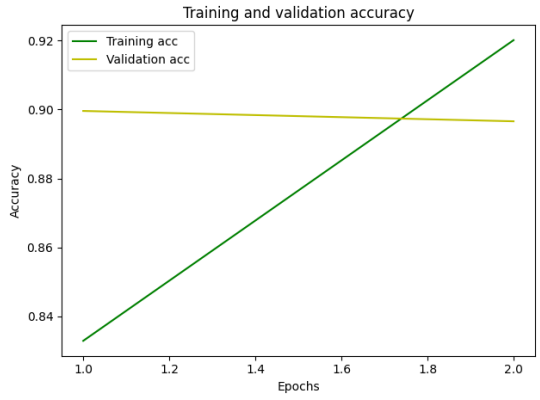
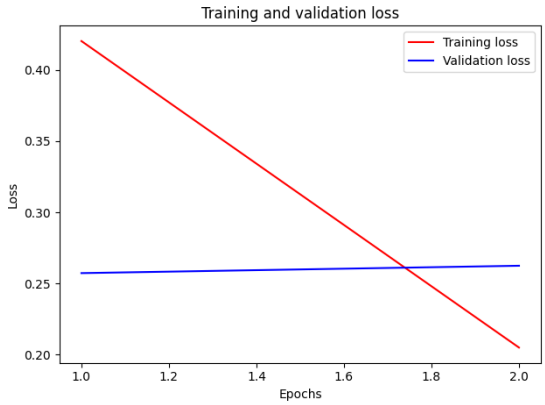
Требования.

- Построить и обучить нейронную сеть для обработки текста
- Исследовать результаты при различном размере вектора представления текста
- Написать функцию, которая позволяет ввести пользовательский текст (в отчете привести пример работы сети на пользовательском тексте)

Ход работы.

Была создана и обучена модель искусственной нейронной сети в соответствии с условиями (код представлен в приложении).

Далее в ходе различных тестирований мы выбрали строение модели, при тестировании которой получили следующие результаты.

Тестирование выбранной модели	
График точности	График потери
	

Для тестирования поведения сети в зависимости от размера вектора представления текста была написана функция `test_dimensions`.

Протестировано поведение при варьирующемся размере вектора представления текста. График точности показан на рис. 2.

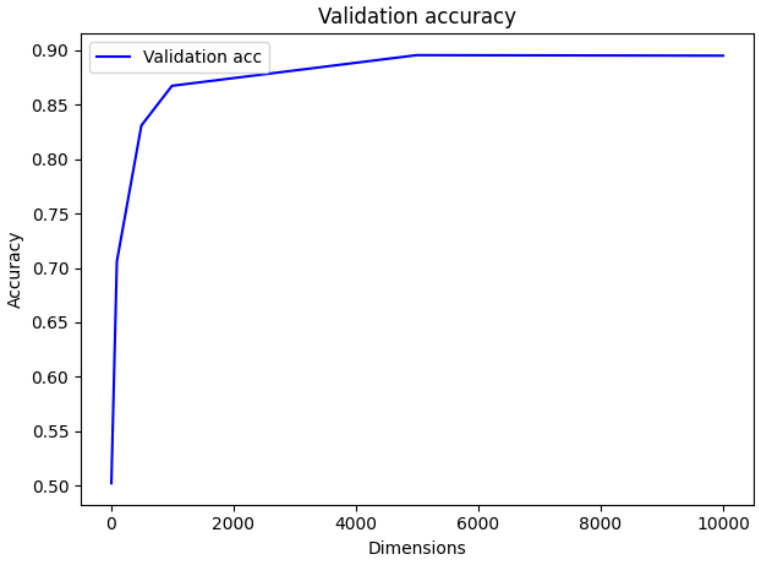
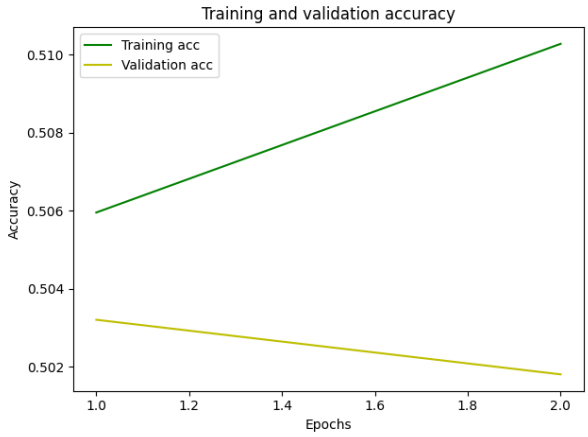
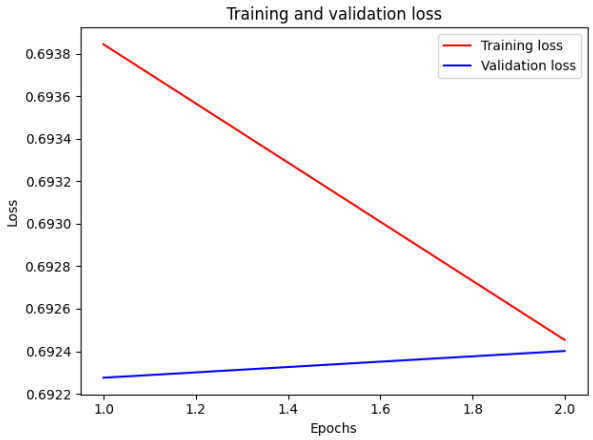
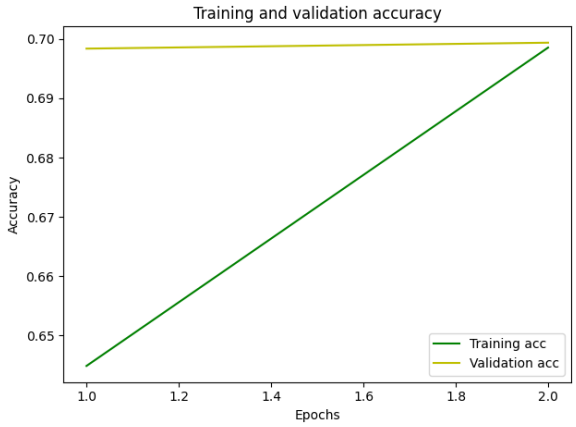
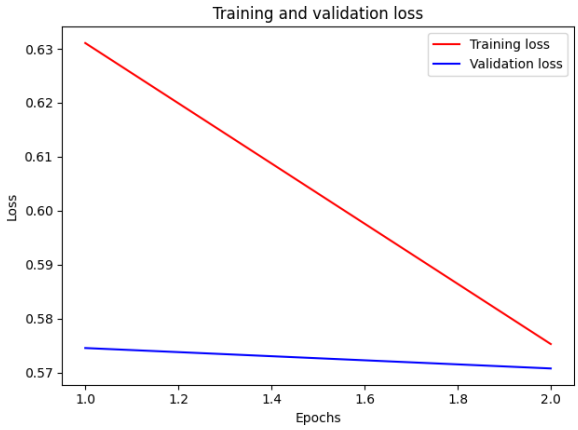
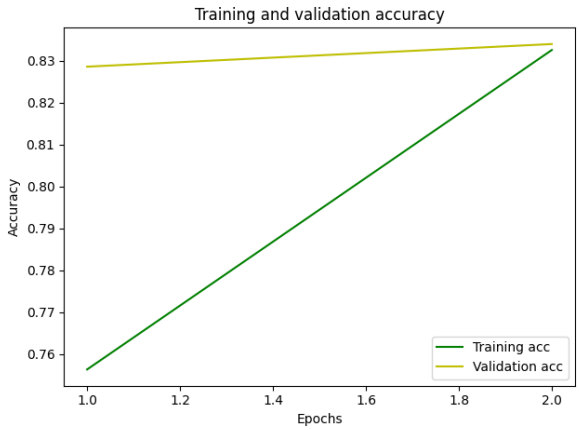
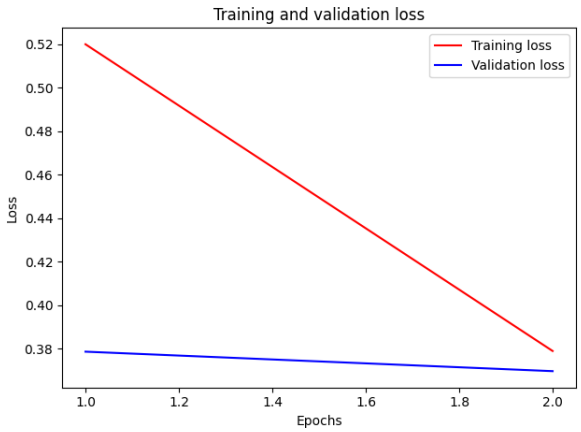
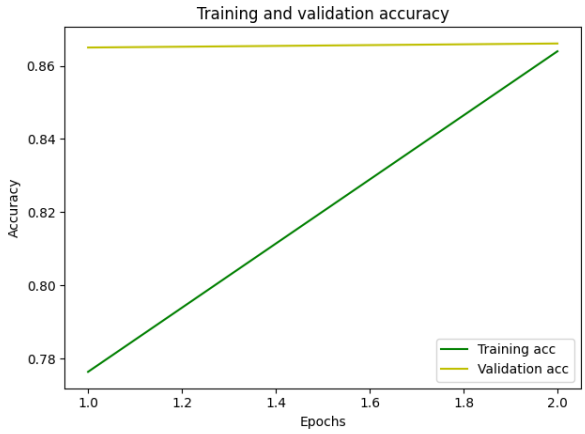
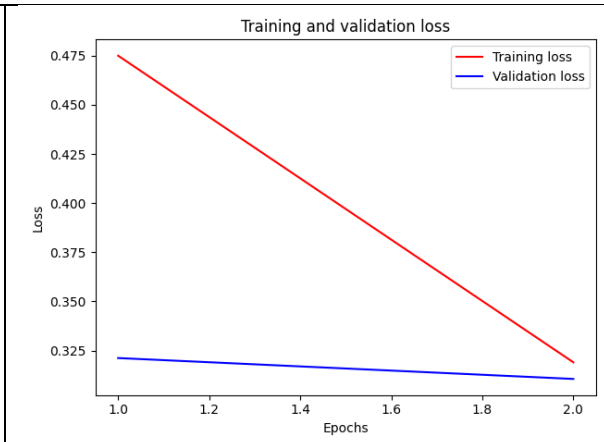


Рисунок 2

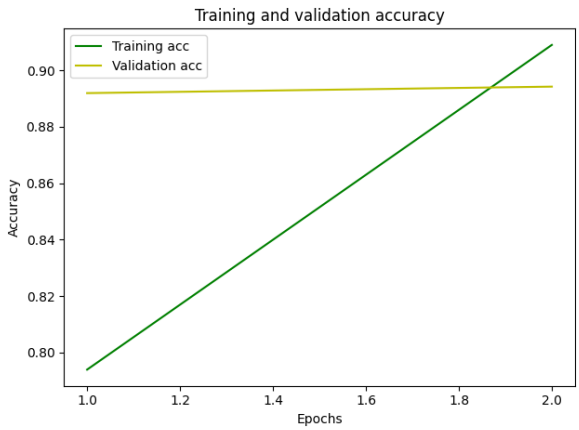
10	
График точности	График потерь
	
50	
	
500	
	

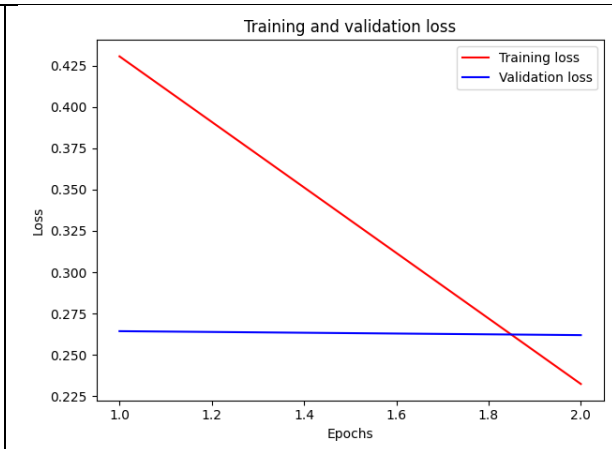
1 000



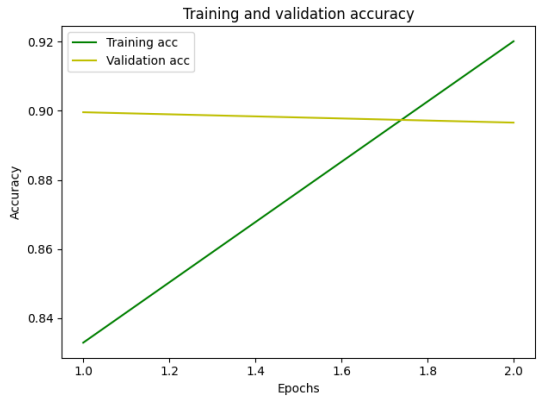


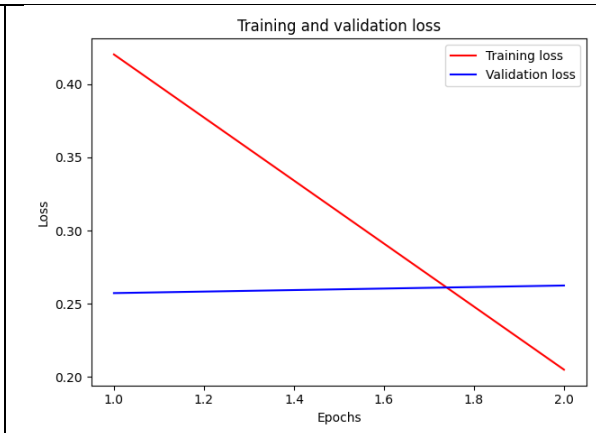
5 000





10 000





Была написана функция `user_load` для загрузки пользовательского текста и прогнозирования успеха фильма по этому тексту.

Точность прогнозирования оценки по тексту обзора сети равна 0.6.

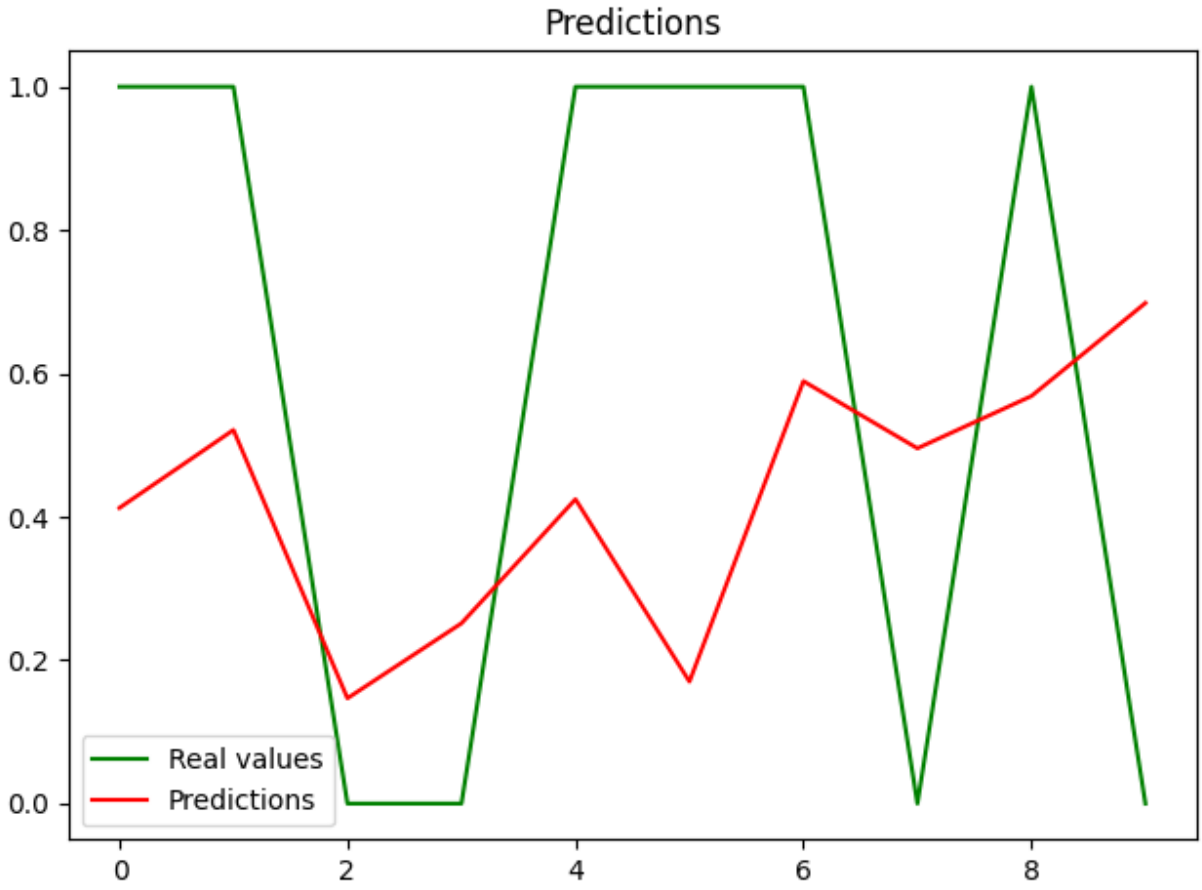


Рисунок 12 - Соответствие реальной оценки и предсказанной сетью

Можно сделать вывод, что данных не достаточно для высокой точности, потому что точность сильно меньше.

Вывод.

Была построена сеть, прогнозирующая оценку фильма по обзорам. Было рассмотрено преобразование текста в формат, с которым может работать нейронная сеть. Было исследовано влияние размера вектора представления текста и выявлено, что наибольшей точностью обладает сеть с максимальным размером вектора, равным 10000. Была написана функция прогнозирования оценки по пользовательскому тексту.

ПРИЛОЖЕНИЕ

Исходный код программы

```
import numpy as np
from keras import models
from keras import layers
import matplotlib.pyplot as plt

from keras.datasets import imdb

EPOCHS = 2
BATCH_SIZE = 500

strings = ["A gorgeous movie, I'll review it again. I cried with emotion. I really liked it!",
           "Very interesting film. Great story and talented actors.",
           "Who is the producer of this shit. I had enough for 10 minutes and I turned it off.",
           "A very flashy plot. Scenes are typical and predictable. Actors replay.",
           "Great movie with a great story",
           "This film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert",
           "Is an amazing actor and now the same being director!",
           "It is so bad.",
           "This film shows humanity in all its glory. We can try to claim the rich are evil and the poor deserve better but these are delusions we create to make us feel better. People are terrible. They are selfish. They are greedy. They are unkind. It's part of human nature. No matter how much you have, or how little, humanity still exists, and humanity is the problem.",
           "I'm crying that I spent my time on it."]

val = [1, 1, 0, 0, 1, 1, 1, 0, 1, 0]
```

```
#########################################################################
########################################
def create_graphics(history):
    # График потерь
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    epochs = range(1, len(loss) + 1)
    print(len(loss))
    plt.plot(epochs, loss, 'r', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()

    #График точности
    plt.plot(epochs, acc, 'g', label='Training acc')
    plt.plot(epochs, val_acc, 'y', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
#########################################################################
########################################

#########################################################################
########################################
#векторизовать каждый обзор
```

```
def vectorize(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1

    return results
##########################################################################
########################################

##########################################################################
########################################
def load_data(dimension):
    (training_data, training_targets), (testing_data, testing_targets)
= imdb.load_data(num_words=dimension)

    data = np.concatenate((training_data, testing_data), axis=0)
    targets = np.concatenate((training_targets, testing_targets),
axis=0)
    data = vectorize(data, dimension)

    #преобразование переменных в тип float
    targets = np.array(targets).astype("float32")

    test_x = data[:10000]
    test_y = targets[:10000]
    train_x = data[10000:]
    train_y = targets[10000:]

    return (train_x, train_y), (test_x, test_y)
##########################################################################
########################################

##########################################################################
########################################
```

```
def build_model(dimension):
    model = models.Sequential()
    # Input - Layer
    model.add(layers.Dense(500, activation="relu",
input_shape=(dimension,)))

    # Hidden - Layers
    model.add(layers.Dropout(0.4, noise_shape=None, seed=None))
    model.add(layers.Dense(50, activation="relu"))
    model.add(layers.Dropout(0.25, noise_shape=None, seed=None))
    model.add(layers.Dense(50, activation="relu"))

    # Output- Layer
    model.add(layers.Dense(1, activation="sigmoid"))
    model.compile(optimizer="adagrad", loss="binary_crossentropy",
metrics=["accuracy"])

    model.summary()
    return model
##########################################################################
########################################

##########################################################################
########################################
def model_fit(train_x, train_y, test_x, test_y, dimension):
    model = build_model(dimension)
    history = model.fit(train_x, train_y, epochs=EPOCHS,
batch_size=BATCH_SIZE, validation_data=(test_x, test_y))
    return history
##########################################################################
########################################

##########################################################################
########################################
```

```
def user_load():
    dictionary = dict(imdb.get_word_index())
    test_x = []
    test_y = np.array(val).astype("float32")

    for string in strings:
        words = string.replace(',', ' ').replace('.', '
').replace('?', ' ').replace('\n', ' ').split()
        numb = []
        for word in words:
            word = dictionary.get(word)
            if word is not None and word < 10000:
                numb.append(word)
        test_x.append(numb)
    print(test_x)
    test_x = vectorize(test_x)
    model = build_model(10000)

    (train_x, train_y), (s1, s2) = load_data(10000)

    model.fit(train_x, train_y, epochs = EPOCHS, batch_size =
BATCH_SIZE)
    val_loss, val_acc = model.evaluate(test_x, test_y)

    print("Validation accuracy is %f" % val_acc)
    predictions = model.predict(test_x)

    plt.title("Predictions")
    plt.plot(test_y, 'g', label='Real values')
    plt.plot(predictions, 'r', label='Predictions')
    plt.legend()
    plt.show()
    plt.clf()
```

```
##########################################################################
########################################

##########################################################################
########################################
def test_dim(dimension):
    (train_x, train_y), (test_x, test_y) = load_data(dimension)
    history = model_fit(train_x, train_y, test_x, test_y, dimension)
    create_graphics(history)

    return history.history['val_accuracy'][-1]
##########################################################################
########################################

##########################################################################
########################################
#lдля тестирования размера
def test_dimensions():
    dimensions = [10, 100, 500, 1000, 5000, 10000]
    val_accuracies = []
    for dim in dimensions:
        val_accuracies.append(test_dim(dim))

    plt.plot(dimensions, val_accuracies, 'b', label='Validation acc')
    plt.title('Validation accuracy')
    plt.xlabel('Dimensions')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
##########################################################################
########################################

##########################################################################
########################################
```

```
(train_x, train_y), (test_x, test_y) = load_data(10000)
history = model_fit(train_x, train_y, test_x, test_y, 10000)
create_graphics(history)

#user_load()
#test_dimensions()
```