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

ОТЧЕТ

по лабораторной работе №6
по дисциплине «Искусственные нейронные сети»
Тема: Прогноз успеха фильмов по обзорам

Студент гр. 8383 _____ Сосновский Д. Н.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2021

Цель работы.

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

Задачи.

1. Ознакомиться с задачей классификации
2. Изучить способы представления текста для передачи в ИНС
3. Достигнуть точности прогноза не менее 95%

Требования

1. Построить и обучить нейронную сеть для обработки текста
2. Исследовать результаты при различном размере вектора представления текста
3. Написать функцию, которая позволяет ввести пользовательский текст

Ход работы.

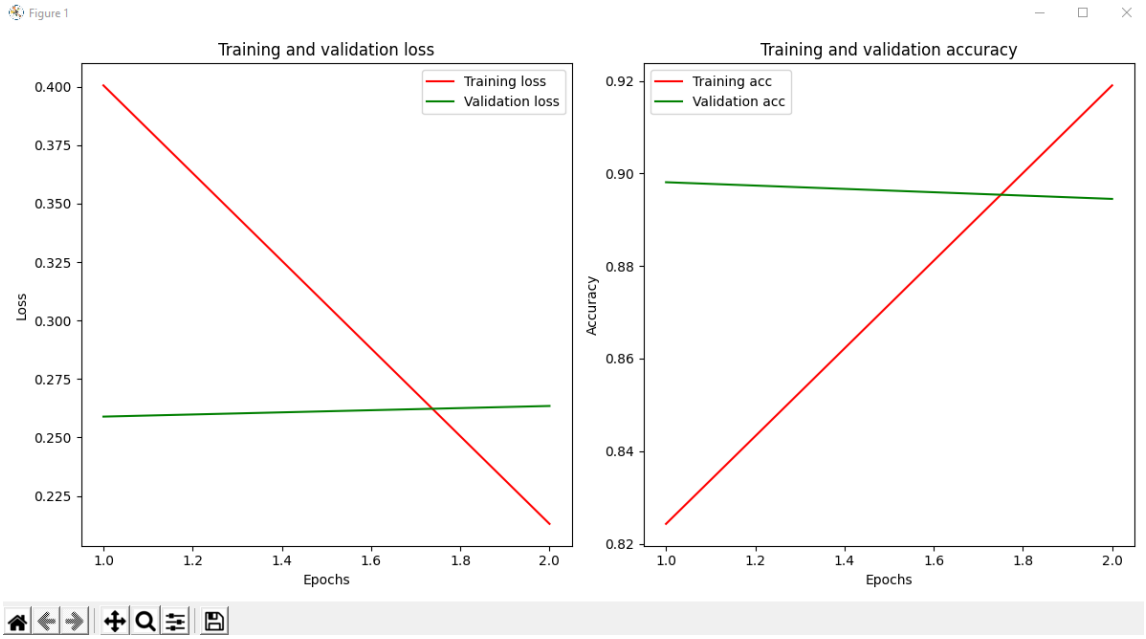
Согласно методическим указаниям была построена и обучена нейронная сеть. Исходный код программы приведён в приложении А. Архитектура модели имеет вид, изображенный на рисунке ниже.

```
# Input - Layer
model = Sequential()
model.add(layers.Dense(50, activation="relu", input_shape=(10000,)))

# Hidden - Layers

model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation="relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation="relu"))

# Output- Layer
model.add(layers.Dense(1, activation="sigmoid"))
```

Данная архитектура нейросети достигает точности 90% на исходных данных. Результат представлен на рисунке ниже.

```
Epoch 1/2
80/80 [==============================] - 3s 37ms/step - loss: 0.5218 - accuracy: 0.7310 - val_loss: 0.2575 - val_accuracy: 0.8971
Epoch 2/2
80/80 [==============================] - 1s 18ms/step - loss: 0.2140 - accuracy: 0.9186 - val_loss: 0.2656 - val_accuracy: 0.8941
0.896275000333786
```

Графики потерь и точности на тренировочных и валидационных наборах данных представлены на рисунке ниже.



Исследование результатов при различном размере вектора представления текста.

Проведем исследование изменения точности сети при уменьшении размера вектора с 10000 до 3000 и 1000. Результаты представлены на рисунках ниже.

```
Epoch 1/2
80/80 [==============================] - 3s 36ms/step - loss: 0.5352 - accuracy: 0.7307 - val_loss: 0.2607 - val_accuracy: 0.8943
Epoch 2/2
80/80 [==============================] - 2s 19ms/step - loss: 0.2124 - accuracy: 0.9214 - val_loss: 0.2703 - val_accuracy: 0.8919
0.8679875135421753
```

Рисунок 1 – точность при размере вектора 3000

```
2021-04-30 00:12:13.402498: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
Epoch 1/2
80/80 [==============================] - 3s 37ms/step - loss: 0.5302 - accuracy: 0.7337 - val_loss: 0.2595 - val_accuracy: 0.8999
Epoch 2/2
80/80 [==============================] - 2s 19ms/step - loss: 0.2119 - accuracy: 0.9213 - val_loss: 0.2623 - val_accuracy: 0.8936
0.8693499863147736
```

Рисунок 1 – точность при размере вектора 1000

Таким образом, при уменьшении размера вектора точность также уменьшается, исходя из сравнения запусков с запуском размера вектора 10000.

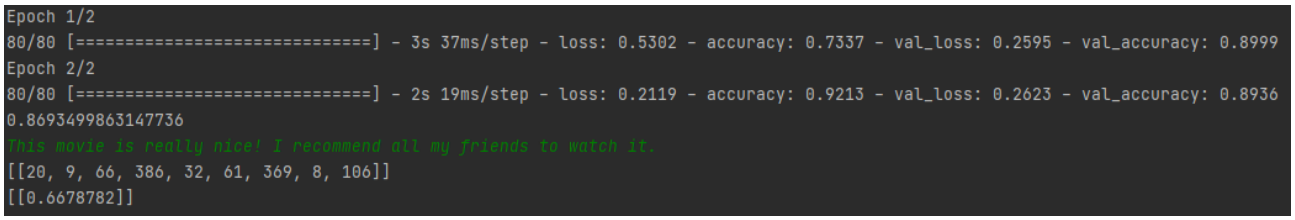
Реализация функции ввода текста.

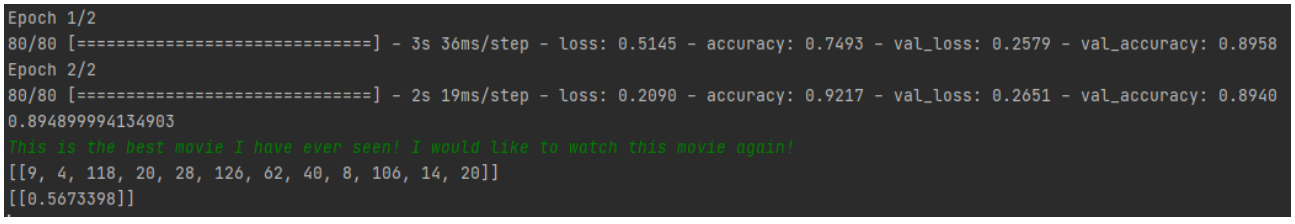
Была реализована функция textInput() для того, чтобы считывать ввод текста пользователем с консоли. Исходный код функции приведён в листинге ниже.

Листинг 1 – исходный код функции textInput()

```
def textInput():
    words = input().split(' ')

    imdb_dict = imdb.get_word_index()
    words_rate = []
    tmp = []

    words_range = 1000
    for word in words:
        if imdb_dict.get(word) in range(1, words_range):
            tmp.append(imdb_dict.get(word) + 3)
    words_rate.append(tmp)

    print(words_rate)
    words_rate = vectorize(words_rate)
    result = model.predict(words_rate)
    print(result)
```

Преобразуем исходный текст в массив слов, и ищем для каждого слова его рейтинг в словаре imdb. Далее передаём векторизированный список слов нейронной сети и выводим результат. Пример работы функции представлен на рисунках ниже.





Вывод.

В ходе выполнения лабораторной работы была создана нейронная сеть для прогнозирования успеха фильма по обзору. Была реализована функция для ввода пользовательского текста. Был изучен способ представления текста для передачи в нейронную сеть.

ПРИЛОЖЕНИЕ А

В этом приложении приведён исходный код программы.

```
import matplotlib.pyplot as plt
from matplotlib import gridspec
import numpy as np
from keras.models import Sequential
from keras.datasets import imdb
from tensorflow.python.keras.applications.densenet import layers

(training_data, training_targets), (testing_data, testing_targets) =
imdb.load_data(num_words=10000)
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets), axis=0)
print("Categories:", np.unique(targets))
print("Number of unique words:", len(np.unique(np.hstack(data))))
length = [len(i) for i in data]
print("Average Review length:", np.mean(length))
print("Standard Deviation:", round(np.std(length)))
print("Label:", targets[0])
print(data[0])
index = imdb.get_word_index()
reverse_index = dict([(value, key) for (key, value) in index.items()])
decoded = " ".join([reverse_index.get(i - 3, "#") for i in data[0]])
print(decoded)


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


data = vectorize(data)
targets = np.array(targets).astype("float16")
test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]

# Input - Layer
model = Sequential()
model.add(layers.Dense(50, activation="relu", input_shape=(10000,)))

# Hidden - Layers

model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation="relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
```

```
model.add(layers.Dense(50, activation="relu"))

# Output- Layer
model.add(layers.Dense(1, activation="sigmoid"))

model.summary()

model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])

results = model.fit(train_x, train_y, epochs=2, batch_size=500,
validation_data=(test_x, test_y))

print(np.mean(results.history["val_accuracy"]))


def draws(H):
    loss = H.history['loss']
    val_loss = H.history['val_loss']
    acc = H.history['accuracy']
    val_acc = H.history['val_accuracy']
    epochs = range(1, len(loss) + 1)

    fig = plt.figure(figsize=(12, 6))
    gs = gridspec.GridSpec(1, 2, width_ratios=[3, 3])
    plt.subplot(gs[0])
    plt.plot(epochs, loss, 'r', label='Training loss')
    plt.plot(epochs, val_loss, 'g', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(gs[1])
    plt.plot(epochs, acc, 'r', label='Training acc')
    plt.plot(epochs, val_acc, 'g', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()


draws(results)


def textInput():
    words = input().split(' ')
```

```
    imdb_dict = imdb.get_word_index()
    words_rate = []
    tmp = []

    words_range = 1000
    for word in words:
        if imdb_dict.get(word) in range(1, words_range):
            tmp.append(imdb_dict.get(word) + 3)
    words_rate.append(tmp)

    print(words_rate)
    words_rate = vectorize(words_rate)
    result = model.predict(words_rate)
    print(result)


if __name__ == '__main__':
    textInput()
```