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»

Студент гр. 8383 _____ Киреев К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

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

Ход работы

Построение и обучение нейронной сети

Была реализована архитектура нейронной сети. Также модель представлена в файле `model.png`.

```
model = Sequential([
    Dense(64, activation = "relu", input_shape=(10000, )),
    Dropout(.4),
    Dense(32, activation = "relu"),
    Dropout(.2),
    Dense(32, activation = "relu"),
    Dropout(.2),
    Dense(1, activation = "sigmoid")])
```

Была достигнута точность 89% на валидационных данных.

```
Epoch 1/2
80/80 [==============================] - 5s 21ms/step - loss: 0.5369 - accuracy: 0.7190 - val_loss: 0.2656 - val_accuracy: 0.8936
Epoch 2/2
80/80 [==============================] - 1s 14ms/step - loss: 0.2319 - accuracy: 0.9117 - val_loss: 0.2642 - val_accuracy: 0.8942
0.8939000070095062
```

Модель обучалась на протяжении 2 эпох, пакетами по 500 образцов. На

рисунке 1 можно увидеть, что при увеличении количества эпох обучения действительно наступает переобучение после 2 эпохи.

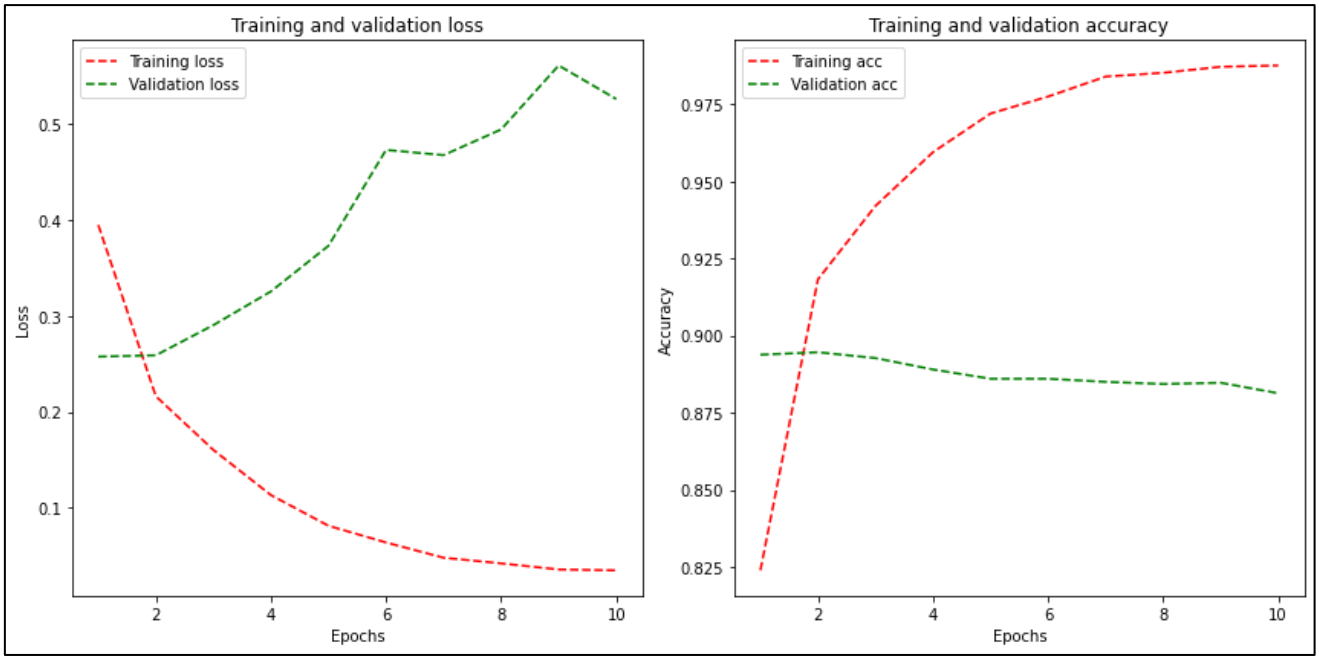

Рис. 1 – Переобучение нейронной сети

Исследование результатов при различном размере вектора представления текста

Исследуем результаты обучения нейросети при измененном размере вектора представления. При уменьшении размера вектора представления текста с 10000 до 5000 результаты ухудшились совсем незначительно до 88.9%.

```
Epoch 1/2
80/80 [==============================] - 1s 11ms/step - loss: 0.5504 - accuracy: 0.7057
                                                      - val_loss: 0.2750 - val_accuracy: 0.8871
Epoch 2/2
80/80 [==============================] - 1s 8ms/step - loss: 0.2479 - accuracy: 0.9043
                                                     - val_loss: 0.2656 - val_accuracy: 0.8910
0.8890499770641327
```

При дальнейшем уменьшении размера вектора до 2000 результаты ухудшились уже до 87.4%.

```
Epoch 1/2
80/80 [==============================] - 1s 7ms/step - loss: 0.5899 - accuracy: 0.6627
                                                    - val_loss: 0.3063 - val_accuracy: 0.8710
Epoch 2/2
80/80 [==============================] - 0s 5ms/step - loss: 0.3009 - accuracy: 0.8758
                                                    - val_loss: 0.2890 - val_accuracy: 0.8773
0.8741500079631805
```

Результаты при уменьшении до 1000 представлены ниже.

```
Epoch 1/2
80/80 [==============================] - 1s 6ms/step - loss: 0.6098 - accuracy: 0.6503
                                                    - val_loss: 0.3457 - val_accuracy: 0.8503
Epoch 2/2
80/80 [==============================] - 0s 4ms/step - loss: 0.3625 - accuracy: 0.8452
                                                    - val_loss: 0.3208 - val_accuracy: 0.8592
0.854750007390976
```

Можно сделать вывод, что при недостаточном размере вектора представления текста результаты работы нейросети ухудшаются.

Функция ввода пользовательского текста

Была написана функция для ввода пользовательского текста `load_text()`

```
def load_text():
    dictionary = imdb.get_word_index()
    load_x = []

    words = input()
    words = re.sub(r"[^a-zA-Z0-9']", " ", words)
    words = words.split(' ')

    valid = []
    for word in words:
        word = dictionary.get(word) # в число
        if word in range(1, 10000):
            valid.append(word+3)
    load_x.append(valid)

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

Получаем словарь со словами и их индексами. Далее обрабатываем введенный текст, удаляя лишние символы. Заменяем числа на их индексы,

оставляя только 10000 самых частых слов. Далее векторизуем текст, написанной ранее функцией и выводим результат предсказания нейронной сети. Пример работы представлен ниже.

```
With so many characters, the movie spends too much time
on discovery and not enough on showing those powers in action.
[[38, 111, 105, 4, 20, 2617, 99, 76, 58, 23, 3808, 5, 24, 195, 23, 800, 148, 1722, 11, 206]]
[[0.22339423]]

 Is Warner Bros.' and Simon McQuoid's reboot perfect? No. But hot damn it is good!
[[758, 5, 404, 893, 1543, 12, 9, 52]]
[[0.88753694]]
```

Выводы

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

Приложение А. Исходный код программы

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
import re
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import gridspec
import numpy as np
from keras.utils import to_categorical
from keras.layers import Dense, Dropout
from keras.models import Sequential


from keras.datasets import imdb
(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()
print(index['good'])
reverse_index = dict([(value, key) for (key, value) in index.items()])
print(reverse_index[49])
print(reverse_index[14-3])
print([i for i in data[0]])
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

        print(type(data))
        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:]

        model = Sequential([
            Dense(64, activation = "relu", input_shape=(10000, )),
            Dropout(.4),
            Dense(64, activation = "relu"),
            Dropout(.2),
            Dense(32, activation = "relu"),
            Dropout(.1),
            Dense(1, activation = "sigmoid")])
        # model.save('model.h5')

        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 load_text():
    dictionary = imdb.get_word_index()
    load_x = []

    words = input()
    words = re.sub(r"[^a-zA-Z0-9']", " ", words)
    words = words.split(' ')

    valid = []
    for word in words:
        word = dictionary.get(word) # в число
        if word in range(1, 10000):
            valid.append(word+3)
```

```
    load_x.append(valid)

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

load_text()
```