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 _____ Дейнега В. Е.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2021

Цель работы

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

Задание.

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

Выполнение работы

1) Был загружен и обработан датасет 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)
    data = vectorize(data)
    targets = np.array(targets).astype("float32")
    test_x = data[:5000]
    test_y = targets[:5000]
    train_x = data[5000:]
    train_y = targets[5000:]
```

Была реализована модель ИНС.

```
        model = Sequential()
        model.add(layers.Dense(50, activation="sigmoid", input_shape=(20000,)))
        model.add(layers.Dense(50, activation="sigmoid"))
        model.add(layers.Dense(75, activation="sigmoid"))
        model.add(layers.Dense(1, activation="sigmoid"))
        opt= tf.keras.optimizers.Adam(lr=0.01, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0., amsgrad=True, clipnorm=1.,clipvalue=0.5)
        model.compile(optimizer=opt,loss="binary_crossentropy", metrics=["accuracy"])
```

Модель тестировалась множество раз с разичными праметрами. В частности добавлялись скрытые слои, изменялось количество нейронов на каждом слое, были протестрованы несколько оптимизаторов с различными

параметрами. Было выявлено, что модель начинает переобучаться после 2 эпохи, добавление дополнительных слоев dropout с различными вероятностями не помогают сети увеличить свою точность при увеличении количества эпох (тестровались модели с 7 эпохами обучения). Поэтому было решено отказаться от 1 слоя dropout. Функции активаци были заменены на sigmoid, хотя большого прироста точности это так же не дало.

2) Протестируем полученную нейросеть, изменяя длину вектора представления текста.

При длине 10 000:

Epoch 1/2

45/45 [==============================] - 2s 31ms/step - loss: 0.7045 - accuracy: 0.5622 - val_loss: 0.2792 - val_accuracy: 0.8944

Epoch 2/2

45/45 [==============================] - 1s 17ms/step - loss: 0.2400 - accuracy: 0.9115 - val_loss: 0.2611 - val_accuracy: 0.8976

При длине 20 000:

Epoch 1/2

45/45 [==============================] - 3s 49ms/step - loss: 0.6449 - accuracy: 0.6009 - val_loss: 0.2940 - val_accuracy: 0.8840

Epoch 2/2

45/45 [==============================] - 1s 27ms/step - loss: 0.2183 - accuracy: 0.9195 - val_loss: 0.2609 - val_accuracy: 0.8970

Затрачено ~ **13** гб оперативной памяти

При длине 30 000:

Epoch 1/2

45/45 [==============================] - 4s 69ms/step - loss: 0.5574 - accuracy: 0.6795 - val_loss: 0.2588 - val_accuracy: 0.8934

Epoch 2/2

45/45 [==============================] - 2s 41ms/step - loss: 0.1981 - accuracy: 0.9252 - val_loss: 0.2680 - val_accuracy: 0.8972

Затрачено ~ **20** гб оперативной памяти

При длине 40 000:

Epoch 1/2

45/45 [==============================] - 5s 82ms/step - loss: 0.6203 - accuracy: 0.6274 - val_loss: 0.2629 - val_accuracy: 0.8914

Epoch 2/2

45/45 [==============================] - 2s 53ms/step - loss: 0.2106 - accuracy: 0.9218 - val_loss: 0.2658 - val_accuracy: 0.8940

Затрачено ~ **27** гб оперативной памяти

При длине 70 000:

Epoch 1/2

45/45 [==============================] - 7s 121ms/step - loss: 0.6144 - accuracy: 0.6247 - val_loss: 0.2631 - val_accuracy: 0.8968

Epoch 2/2

45/45 [==============================] - 4s 88ms/step - loss: 0.2190 - accuracy: 0.9179 - val_loss: 0.2615 - val_accuracy: 0.8952

Затрачено ~ **47** гб оперативной памяти

Как видно из результатов, при увеличении длины вктора представления текста, точность на валидационных данных не увеличивается, однако сильно растет количество используемой оперативной памяти. Поэтому оптимально будет использовать длину вектора от 10 000 до 20 000.

3) Была написана функция пользовательского ввода текста из файла. Пользователь вводит название файла с текстом рецензии.

```
def user_input():
    print("Type file name:\n")
    filename = input()
    f = open(filename+".txt", "r")
    text = f.read()
```

```
    prediction(text_prepare(text), load_model())
    return 0
```

Вспомогательная функция, подготавливающая и кодирующая текст.

```
def text_prepare(text):
    text = text.lower().strip()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\n', ' ', text)
    text = text.split(' ')
    index = imdb.get_word_index()
    coded = []

    for word in text:
        ind = index.get(word)
        if ind is not None and ind < 20000:
            coded.append(ind + 3)
    return vectorize(np.asarray([coded]))
```

Протестируем нейросеть, на обзоре фильма Форрест Гамп и Кошки.

Текст рецензии Форреста Гампа:

“*I've seen most of the top-voted 30 or 40 movies here at IMDb, and Forrest Gump probably wins my vote for the single best movie ever made.*

This is not a conclusion I came to the first time I saw this film, or the second. The more you understand about this movie, the more you like it. And in terms of understanding it, my wife and I both were still gaining new understanding the 5th or 6th time we watched it. (For example: Why does Jenny behave exactly as she does throughout the movie? It'll take a lot of thinking to =fully= understand her.)

Just the fact that we've seen it at least half a dozen times says a lot. There are few movies that I ever even give a second viewing.

Like many of the truly good movies, there are people out there who see it once and consider it to be a waste of time. Such people are usually just looking for the most "action," gore and sex they can find, and haven't a clue about films that actually involve thought, ideas, and life.

Did I mention the script is absolutely brilliant? There are a lot of =truly= funny moments. And the music rocks. But those are only specific aspects of a movie that has just about everything else right about it as well.

See it, enjoy it, think about it, see it again. Think about it some more. Forrest Gump is one of the greatest movies -- if not THE greatest -- ever made.”

Оценка нейросети:

0.93372136

Good

Рецензия Кошек:

” *In 25 years this film will have a weird cult following, and an aged James Franco will make a "meta" film about it. Until then, it will remain universally regarded as an utter shambles.*”

Оценка нейросети:

0.15140894

bad

Выводы.

В ходе лабораторной работы был реализован прогноз успеха фильмов по обзорам.

Приложение А

```
import numpy as np
from tensorflow.keras.models import Sequential
from keras import layers
import tensorflow as tf
from keras.datasets import imdb
import re
from keras.models import model_from_json


def vectorize(sequences, dimension=20000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results


def load_model():
    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    loaded_model.load_weights("model.h5")
    optim=tf.keras.optimizers.Adam(lr=0.01,       beta_1=0.9,       beta_2=0.999,
epsilon=None,decay=0.,amsgrad=True,clipnorm=1.,                              clipvalue=0.5)
    loaded_model.compile(optimizer=optim,             loss="binary_crossentropy",
metrics=["accuracy"])
    return loaded_model


def user_input():
    print("Type file name:\n")
    filename = input()
    f = open(filename+".txt", "r")
    text = f.read()
    prediction(text_prepare(text), load_model())
    return 0


def prediction(coded, model):
    res = model.predict(coded)
    print(res)
    if res >= 0.5:
        print('good')
    else:
        print('bad')


def text_prepare(text):
    text = text.lower().strip()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\n', ' ', text)
    text = text.split(' ')
    index = imdb.get_word_index()
    coded = []

    for word in text:
        ind = index.get(word)
        if ind is not None and ind < 20000:
            coded.append(ind + 3)
```

```
            return vectorize(np.asarray([coded]))

        def create_model():
            (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)

            data = vectorize(data)
            targets = np.array(targets).astype("float32")

            test_x = data[:5000]
            test_y = targets[:5000]
            train_x = data[5000:]
            train_y = targets[5000:]

            model = Sequential()
            model.add(layers.Dense(50, activation="sigmoid", input_shape=(20000,)))
            model.add(layers.Dense(50, activation="sigmoid"))
            model.add(layers.Dropout(0.4, noise_shape=None, seed=None))
            model.add(layers.Dense(75, activation="sigmoid"))
            model.add(layers.Dense(1, activation="sigmoid"))

            opt    =    tf.keras.optimizers.Adam(lr=0.01,    beta_1=0.9,    beta_2=0.999,
epsilon=None, decay=0., amsgrad=True, clipnorm=1.,
                                          clipvalue=0.5)
            model.compile(optimizer=opt,                       loss="binary_crossentropy",
metrics=["accuracy"])

            hist    =    model.fit(train_x,    train_y,    epochs=2,    batch_size=1000,
validation_data=(test_x, test_y))

            model_json = model.to_json()
            with open("model.json", "w") as json_file:
                json_file.write(model_json)
            model.save_weights("model.h5")


        user_input()
```