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

ОТЧЕТ

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

Студент гр. 7383 _____ Власов Р.А.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель работы.

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

Порядок выполнения работы.

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

Ход работы.

Для исследования была разработана и использована программа, код которой приведен в приложении А.

Для исследования зависимости результата от различного размера вектора представления текста, исследуем сеть при длинах: 10, 50, 100, 500, 1000, 5000 и 10000.

```
model = models.Sequential()
model.add(layers.Dense(50, activation="relu", input_shape=(dimensions,)))
model.add(layers.Dropout(0.4, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation="relu"))
model.add(layers.Dropout(0.35, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation="relu"))
model.add(layers.Dense(1, activation="sigmoid"))
```

Рисунок 1 — Модель сети

Результаты тестирования показаны на рис. 2. Как видно из графиков, при длине вектора 100 и меньше сеть показывает неутешительную точность, это может быть связано с тем, что большую часть из самых популярных слов составляют слова, необходимые для составления любого предложения. Однако при размере вектора 500 и выше, точность сети приемлемая и несильно возрастает с увеличением размера вектора представления текста. Стоит отметить, что макимальная точность достигается при максимальном размере вектора (0.897), однако она незначительно выше результата, полученного при размере вектора 5000 (0.892) и 1000 (0.861).

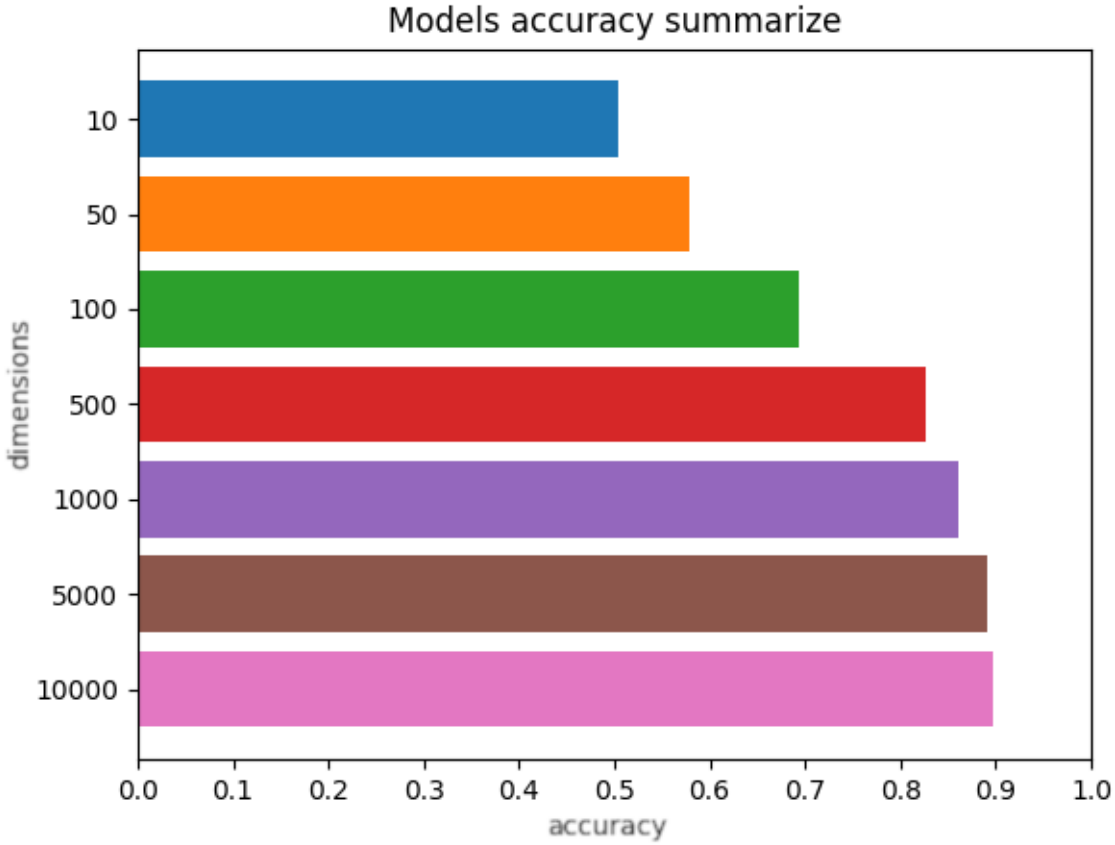

Рисунок 2 — Результаты тестирования

Была написана функция для получения оценки пользовательского текста с помощью прдоставленной модели: predict(review, model, dimensions=10000). Работа функции была протестирована на двух коротких отзывах, результат показан на рис. 3. Можно наблюдать, что полученная оценка вполне совпадает с действительным окрасом отзыва. Отзыв «Удивительно» действительно скорее положительный, а «Ничего особенного» достаточно нейтральный отзыв.

```
"amazing" for 65.74225425720215% is a good review
"nothing special" for 50.391435623168945% is a good review
```

Рисунок 3 — Пользовательские отзывы

Выводы.

В ходе выполнения лабораторной работы была создана сеть для оценки успеха фильма по отзыву. Было исследовано влияние размера вектора представления текста на результат работы сети. Приемлемая точность

достигается сетью при размере вектора 500 и больше. При увеличении размера вектора возрастает и точность сети. Также была написана функция, позволяющая получить оценку фильма по введенному обзору с помощью предоставленной модели.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import matplotlib.pyplot as plt
import numpy as np
from keras import models, layers
from keras.datasets import imdb
import string

EPOCHS = 2
BATCH_SIZE = 500
TEST_DIMENSIONS = [10, 50, 100, 500, 1000, 5000, 10000]
CUSTOM_REVIEWS = [
    "amazing",
    "nothing special"]


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


def buildModel(dimensions=10000):
    model = models.Sequential()
    model.add(layers.Dense(50, activation="relu",
input_shape=(dimensions,)))
    model.add(layers.Dropout(0.4, noise_shape=None, seed=None))
    model.add(layers.Dense(50, activation="relu"))
    model.add(layers.Dropout(0.35, noise_shape=None, seed=None))
    model.add(layers.Dense(50, activation="relu"))
    model.add(layers.Dense(1, activation="sigmoid"))
    model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
    return model


def loadData(dimension=10000):
    (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=dimension)
    targets = np.array(targets).astype("float32")
    return data[10000:], targets[10000:], data[:10000],
targets[:10000]
```

```
def testDimensions():
    ev_accuracy = dict()
    ev_loss = dict()
    for dim in TEST_DIMENSIONS:
        print("testing %d dimensions" % dim)
        train_x, train_y, test_x, test_y = loadData(dim)
        model = buildModel(dim)
        results = model.fit(train_x, train_y, epochs=EPOCHS,
batch_size=BATCH_SIZE, validation_data=(test_x, test_y))
        ev_loss["%s" % dim], ev_accuracy["%s" % dim] =
model.evaluate(test_x, test_y)
        plt.title('Training and test accuracy')
        plt.plot(results.history['accuracy'], 'r',
label='train')
        plt.plot(results.history['val_accuracy'], 'b',
label='test')
        plt.xlabel("epochs")
        plt.ylabel("accuracy")
        plt.legend()
        plt.savefig("Graphics/%s_acc.png" % dim, format='png')
        plt.clf()

        plt.title('Training and test loss')
        plt.plot(results.history['loss'], 'r', label='train')
        plt.plot(results.history['val_loss'], 'b', label='test')
        plt.xlabel("epochs")
        plt.ylabel("loss")
        plt.legend()
        plt.savefig("Graphics/%s_loss.png" % dim, format='png')
        plt.clf()

    for entry in ev_accuracy:
        print("%s: %s" % (entry, ev_accuracy[entry]))
        plt.barh(entry, ev_accuracy[entry])
    plt.title('Models accuracy summarize')
    plt.ylabel('accuracy')
    plt.xlabel('dimensions')
    plt.xticks(np.arange(0, 1.05, 0.1))
    plt.gca().invert_yaxis()
    plt.savefig('Graphics/summarize.png', format='png')
    plt.clf()


def predict(review, model, dimensions=10000):
    punctuation =
str.maketrans(dict.fromkeys(string.punctuation))
    review = review.lower().translate(punctuation).split(" ")
    indexes = imdb.get_word_index()
    encoded = []
```

```
    for w in review:
        if w in indexes and indexes[w] < dimensions:
            encoded.append(indexes[w])
    review = vectorize([np.array(encoded)], dimensions)
    return model.predict(review)[0][0]


def testCustomReview():
    dims = 6000
    train_x, train_y, test_x, test_y = loadData(dims)
    model = buildModel(dims)
    model.fit(train_x, train_y, epochs=EPOCHS,
batch_size=BATCH_SIZE, validation_data=(test_x, test_y))
    for review in CUSTOM_REVIEWS:
        print('"%s" for %s%% is a good review' % (str(review),
predict(review, model, dims) * 100))


testDimensions()
testCustomReview()
```