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

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Искусственные нейронные сети»

Тема: Прогноз успеха фильмов по обзорам

Студент гр. 7383 Сычевский Р.А.

Преподаватель Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Классификация последовательностей - это проблема прогнозирующего моделирования, когда у вас есть некоторая последовательность входных данных в пространстве или времени, и задача состоит в том, чтобы предсказать категорию для последовательности.

Проблема усложняется тем, что последовательности могут различаться по длине, состоять из очень большого словарного запаса входных символов и могут потребовать от модели изучения долгосрочного контекста или зависимостей между символами во входной последовательности.

В данной лабораторной работе также будет использоваться датасет IMDb, однако обучение будет проводиться с помощью рекуррентной нейронной сети.

Порядок выполнения работы.

1. Ознакомиться с рекуррентными нейронными сетями
2. Изучить способы классификации текста
3. Ознакомиться с ансамблированием сетей
4. Построить ансамбль сетей, который позволит получать точность не менее 97%

Ход работы.

Для исследования была разработана и использована программа. Код программы приведен в приложении А.

В ходе работы были созданы две модели нейронной сети. Они представлены на рисунках 1-2. Средняя точность этих сетей в ансамбле составила 89%.

```python
def build_model_1():
    model = Sequential()
    model.add(Embedding(10000, 32, input_length=500))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
    model.add(Dropout(0.25))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.35))
    model.add(LSTM(50))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Рисунок 1 – Модель сети

```python
def build_model_2():
    model = Sequential()
    model.add(Embedding(10000, 32, input_length=500))
    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.35))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.35))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Рисунок 2 – Модель сети

На рисунках 3-4 приведены графики точности и потерь для каждой модели.

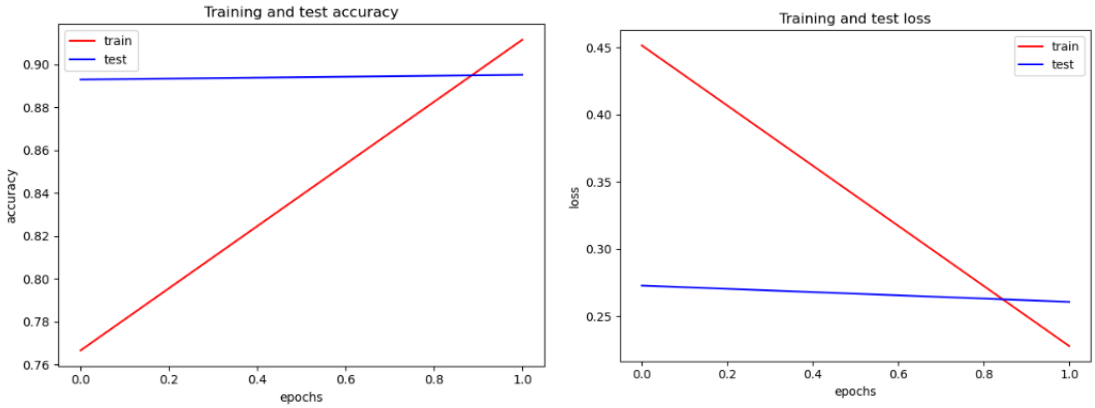

Рисунок 3 – Графики для первой модели

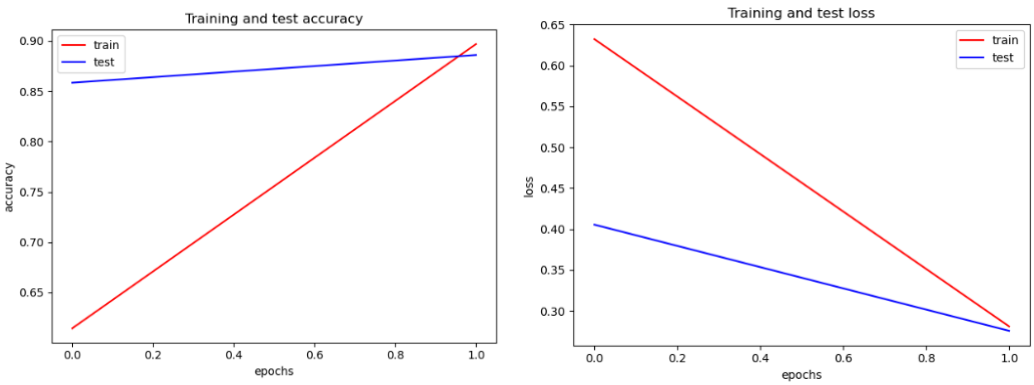

Рисунок 4 – Графики для второй модели

Так же была написана функция для распознавания пользовательского текста.

Выводы.

В ходе выполнения данной работы были разработаны две архитектуры сети. Несколько сетей были объединены в ансамбль сетей. Так же была написана функция, позволяющая определить оценку фильма по пользовательскому обзору с помощью ансамбля сетей.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import numpy
from keras.models import Sequential, load_model
from keras.layers import Dense, LSTM, Conv1D, Dropout, MaxPooling1D,
Flatten
from keras.layers.embeddings import Embedding
from tensorflow.keras.preprocessing import sequence
from keras.datasets import imdb
import numpy as np
import matplotlib.pyplot as plt
import string

EPOCHS = 2
BATCH_SIZE = 250

def load_data():
    (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 = sequence.pad_sequences(data, maxlen=500)
    targets = np.array(targets).astype("float32")
    return data, targets

def build_model_1():
    model = Sequential()
    model.add(Embedding(10000, 32, input_length=500))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(Dropout(0.25))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.35))
    model.add(LSTM(50))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

def build_model_2():
    model = Sequential()
    model.add(Embedding(10000, 32, input_length=500))
    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.35))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.35))
    model.add(Dense(32, activation='relu'))
```

```
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model

def load_text(filename):
    punctuation = ['.', ',', ':', ';', '!', '?', '(', ')']
    text = []
    with open(filename, 'r') as f:
        for line in f.readlines():
            text += [s.strip(''.join(punctuation)).lower() for s in
line.strip().split()]
    indexes = imdb.get_word_index()
    encoded = []
    for w in text:
        if w in indexes and indexes[w] < 10000:
            encoded.append(indexes[w])

def test_text():
    results = []
    model1 = load_model("m1.h5")
    model2 = load_model("m2.h5")
    results.append(model1.predict(text))
    results.append(model2.predict(text))
    result = np.array(results).mean(axis=0)
    result = np.reshape(result, result.shape[0])
    print(result)

def trainModels():
    data, targets = load_data()
    model_1 = build_model_1()
    model_2 = build_model_2()
    history_1 = model_1.fit(data[10000:], targets[10000:],
epochs=EPOCHS, batch_size=BATCH_SIZE,
                            validation_data=(data[:10000],
targets[:10000]))
    history_2 = model_2.fit(data[10000:], targets[10000:],
epochs=EPOCHS, batch_size=BATCH_SIZE,
                            validation_data=(data[:10000],
targets[:10000]))
    plt.title('Training and test accuracy')
    plt.plot(history_1.history['accuracy'], 'r', label='train')
    plt.plot(history_1.history['val_accuracy'], 'b', label='test')
    plt.xlabel("epochs")
    plt.ylabel("accuracy")
    plt.legend()
    plt.show()
    plt.clf()

    plt.title('Training and test loss')
    plt.plot(history_1.history['loss'], 'r', label='train')
```

```
    plt.plot(history_1.history['val_loss'], 'b', label='test')
    plt.xlabel("epochs")
    plt.ylabel("loss")
    plt.legend()
    plt.show()
    plt.clf()

    plt.title('Training and test accuracy')
    plt.plot(history_2.history['accuracy'], 'r', label='train')
    plt.plot(history_2.history['val_accuracy'], 'b', label='test')
    plt.xlabel("epochs")
    plt.ylabel("accuracy")
    plt.legend()
    plt.show()
    plt.clf()

    plt.title('Training and test loss')
    plt.plot(history_2.history['loss'], 'r', label='train')
    plt.plot(history_2.history['val_loss'], 'b', label='test')
    plt.xlabel("epochs")
    plt.ylabel("loss")
    plt.legend()
    plt.show()
    plt.clf()

    loss_1, acc_1 = model_1.evaluate(data[:10000], targets[:10000])
    loss_2, acc_2 = model_2.evaluate(data[:10000], targets[:10000])
    model_1.save("m1.h5")
    model_2.save("m2.h5")
    print("Ensemble accuracy: %s" % ((acc_1 + acc_2) / 2))


trainModels()
filename = 'text.txt'
text=load_text(filename)
text = sequence.pad_sequences([text], maxlen=500)
test_text(text)
```