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

ЛАБОРАТОРНАЯ РАБОТА №7

по дисциплине «Искусственные нейронные сети»

Тема: «Классификация обзоров фильмов»

Студент гр. 7381 _____ Минуллин М.А.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург
2020

Цели.

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.

Задачи.

- Ознакомиться с рекуррентными нейронными сетями
- Изучить способы классификации текста
- Ознакомиться с ансамблированием сетей
- Построить ансамбль сетей, который позволит получать точность не менее 97%

Ход работы.

Был выбран набор из следующих моделей:

Первая модель является ансамблем CNN и RNN:

```
model = Sequential()
model.add(Embedding(top_words, embedding_vector_length, input_length=max_review_length))
        model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(LSTM(100))
        model.add(Dense(1, activation='sigmoid'))
```

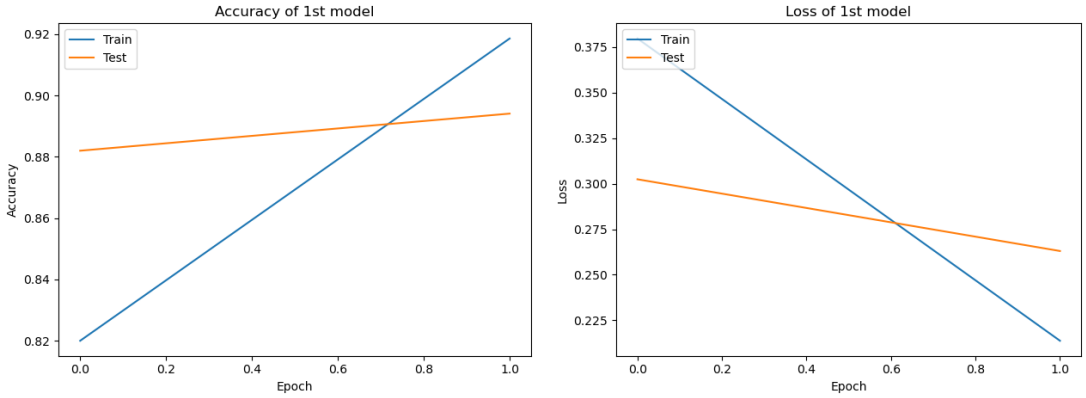

Рисунок 1 — Точность и потери модели 1.

Вторая модель — RNN:

```
model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
```

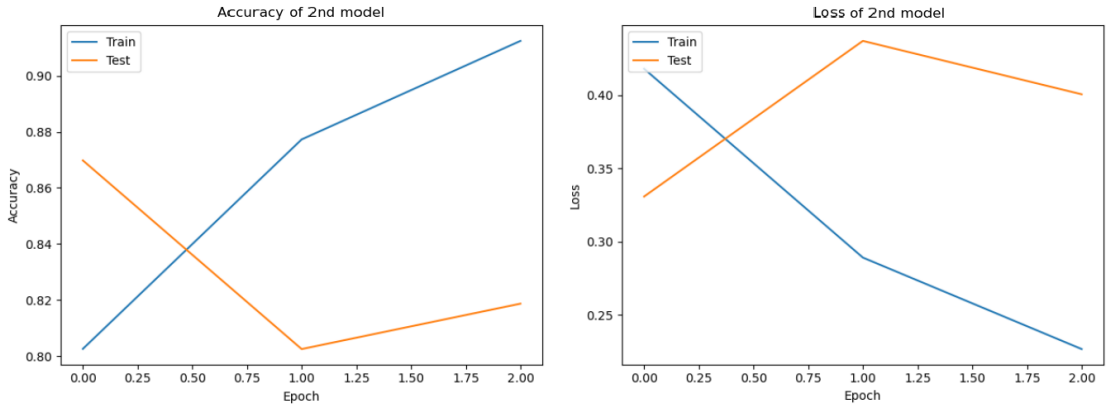

Рисунок 1 — Точность и потери модели 2.

Третья модель — CNN :

```
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
```

```
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

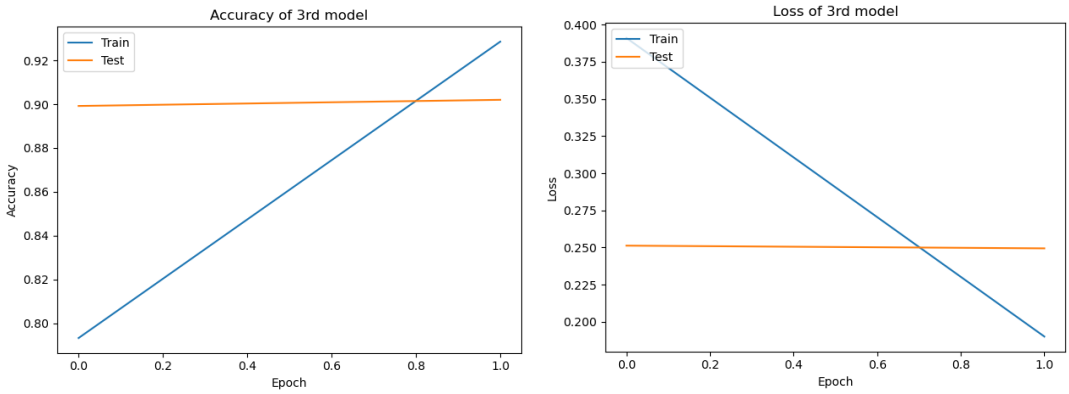

Рисунок 3 — Точность и потери модели 3.

Ансамблирование моделей проведено в функции `ensemble()`.

Проверка своего текста написана в функции `test_text()`.

```
10000/10000 [==============================] - 1s 97us/step
Ensemble accuracy:0.8821666638056437 Ensemble loss:0.2921859386205673
Ensemble predict: 0.8511303
```

Рисунок 4 — Точность и потери ансамбля, а также предсказание ансамбля для положительного ревью из test1.txt, которое вполне соответствует ожидаемому результату (положительной оценке).

Вывод.

В ходе выполнения данной работы было произведено ознакомление с рекуррентными нейронными сетями и ансамблированием сетей, а также классификация обзоров фильмов с помощью рекуррентной сети.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
import re
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, GRU, Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
import matplotlib.pyplot as plt
from keras.layers import *
import numpy as np
import string
from keras.models import Model
from keras.layers import concatenate
from statistics import mean

def Plots(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
    return

def build_1st_model():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
```

```
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    return model

def build_2nd_model():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    print(model.summary())

    return model

def build_3rd_model():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model

def ensemble(train_x,train_y, test_x, test_y, to_predict = None):
    model1 = build_1st_model()
    model2 = build_2nd_model()
    model3 = build_3rd_model()
    model1.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    Plots(model1.fit(train_x, train_y, validation_data=(test_x,
test_y), epochs=epochs, batch_size=batch_size))
    print(model1.evaluate(test_x, test_y))
    model2.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    Plots(model2.fit(train_x, train_y, validation_data=(test_x,
test_y), epochs=epochs, batch_size=batch_size))
```

```
    model3.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    Plots(model3.fit(train_x, train_y, validation_data=(test_x,
test_y), epochs=epochs, batch_size=batch_size))
    ensemble_info = "Ensemble
accuracy:"+str(mean((model1.evaluate(test_x,
test_y)[1],model2.evaluate(test_x, test_y)[1],
model3.evaluate(test_x, test_y)[1])))+" Ensemble
loss:"+str(mean((model1.evaluate(test_x, test_y)[0],
model2.evaluate(test_x, test_y)[0], model3.evaluate(test_x,
test_y)[0])))
    if to_predict.all()!=None:
        print(ensemble_info)
        return "Ensemble predict:
"+str(mean((model1.predict(to_predict)[0][0],model2.predict(to_predi
ct)[0][0],model3.predict(to_predict)[0][0])))
    return ensemble_info

def test_text(text):
    (training_data, training_targets), (testing_data,
testing_targets) = imdb.load_data(num_words=top_words)
    # truncate and pad input sequences

    training_data = sequence.pad_sequences(training_data,
maxlen=max_review_length)
    testing_data = sequence.pad_sequences(testing_data,
maxlen=max_review_length)

    data = np.concatenate((training_data, testing_data), axis=0)
    targets = np.concatenate((training_targets, testing_targets),
axis=0)

    test_x = data[:10000]
    test_y = targets[:10000]
    train_x = data[10000:]
    train_y = targets[10000:]

    output_str = ''
    with open(text, 'r') as f:
        for input_str in f.readlines():
            output_str += re.sub('[^A-Za-z0-9 ]+', '',
input_str).lower()
    indexes = imdb.get_word_index()
    encode = []
```

```
    text = output_str.split()
    #print(text)
    for index in text:
        if index in indexes and indexes[index] < 10000:
            encode.append(indexes[index])
    #print(encode)
    encode = sequence.pad_sequences([np.array(encode)],
maxlen=max_review_length)
    print(ensemble(train_x, train_y, test_x, test_y, encode))


embedding_vector_length = 32
top_words = 10000
max_review_length = 500
batch_size = 100
epochs = 2

if __name__ == '__main__':
    (training_data, training_targets), (testing_data,
testing_targets) = imdb.load_data(num_words=top_words)
    # truncate and pad input sequences

    training_data = sequence.pad_sequences(training_data,
maxlen=max_review_length)
    testing_data = sequence.pad_sequences(testing_data,
maxlen=max_review_length)

    data = np.concatenate((training_data, testing_data), axis=0)
    targets = np.concatenate((training_targets, testing_targets),
axis=0)

    test_x = data[:10000]
    test_y = targets[:10000]
    train_x = data[10000:]
    train_y = targets[10000:]
    # create the model
    embedding_vector_length = 32


   # print(ensemble(test_x, test_y, train_x, train_y))

    test_text('test.txt')
```