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

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Искусственные нейронные сети»

Тема: Классификация обзоров фильмов

Студент гр. 7383 _____ Левкович Д. В.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы:

Реализовать прогнозирование успеха фильмов по обзорам

Задачи.

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.

1. Были созданы и обучены две модели искусственной нейронной сети, решающей задачу определения настроения обзора. Первая нейронная сеть Её архитектура представлена на рис. 1.

```
model = Sequential()
model.add(Embedding(top_words, embedding_vector_length, input_length=max_review_length))
model.add(LSTM(100))
model.add(Dropout(0.3))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Рисунок 1 – Модель первой сети

Вторая сеть сверточная. Её архитектура представлена на рис. 2.

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

Рисунок 2 - Модель третьей сети

Третья сеть рекуррентная с добавлением слоя свертки. Модель представлена на рис.3

```
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'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Рисунок 3 - Модель второй сети

Проведем тестирование каждой модели.

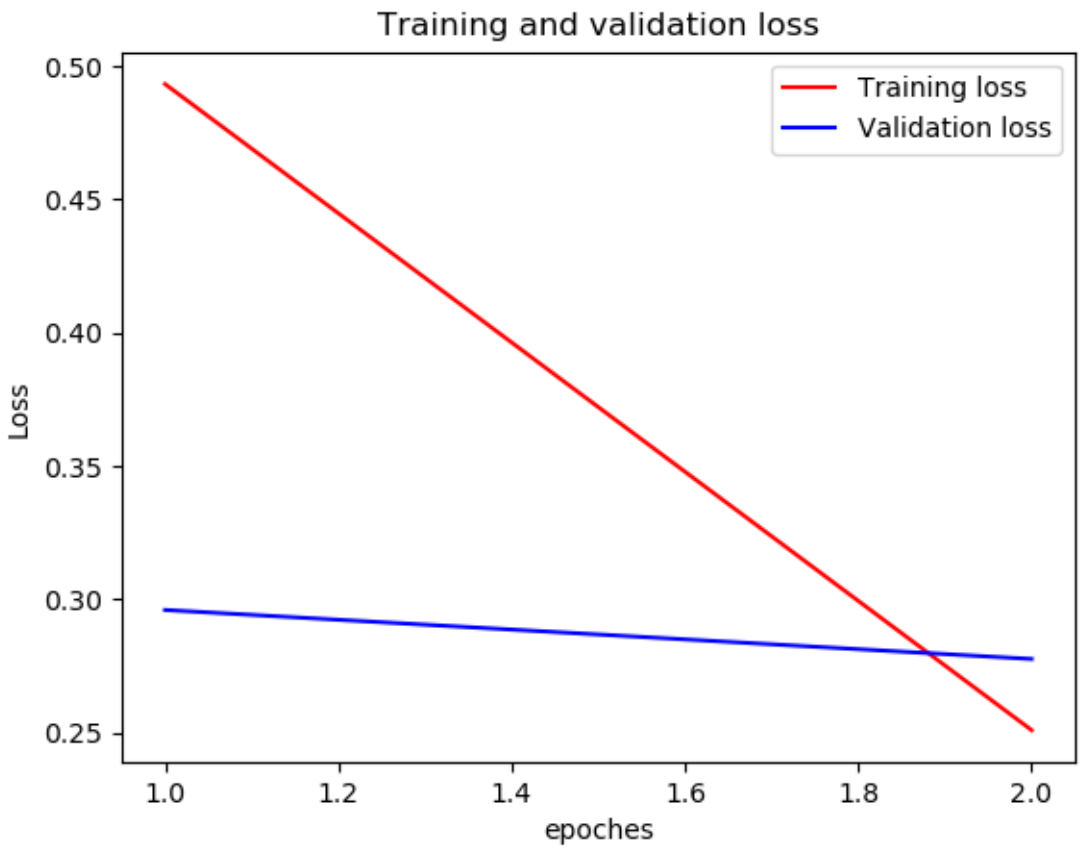

Рисунок 2 - Потери первой сети

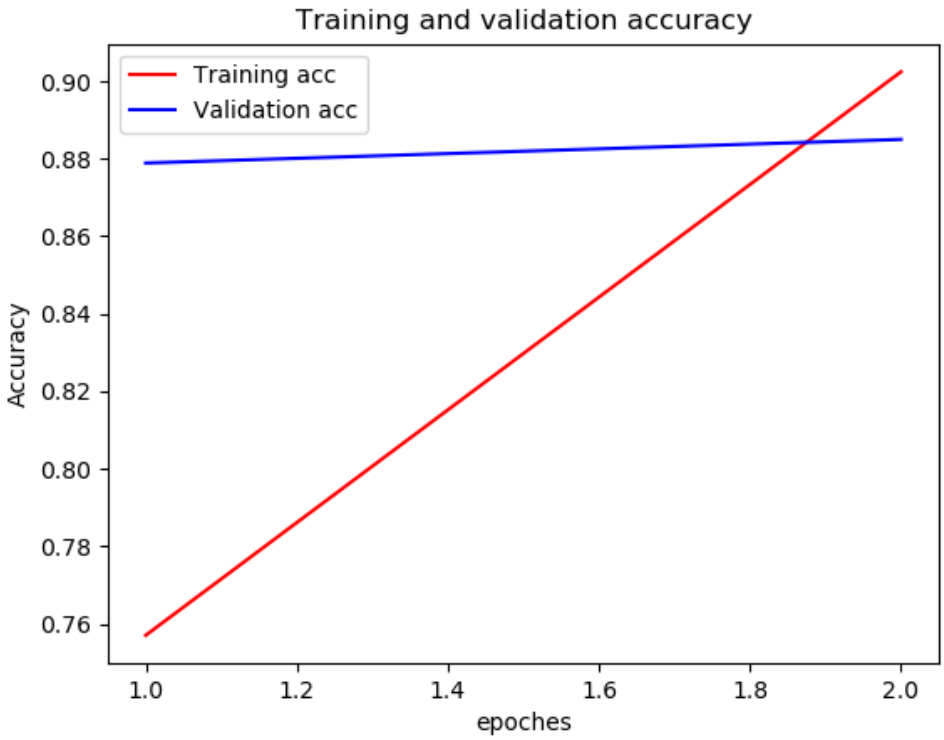

Рисунок 3 - Точность первой сети

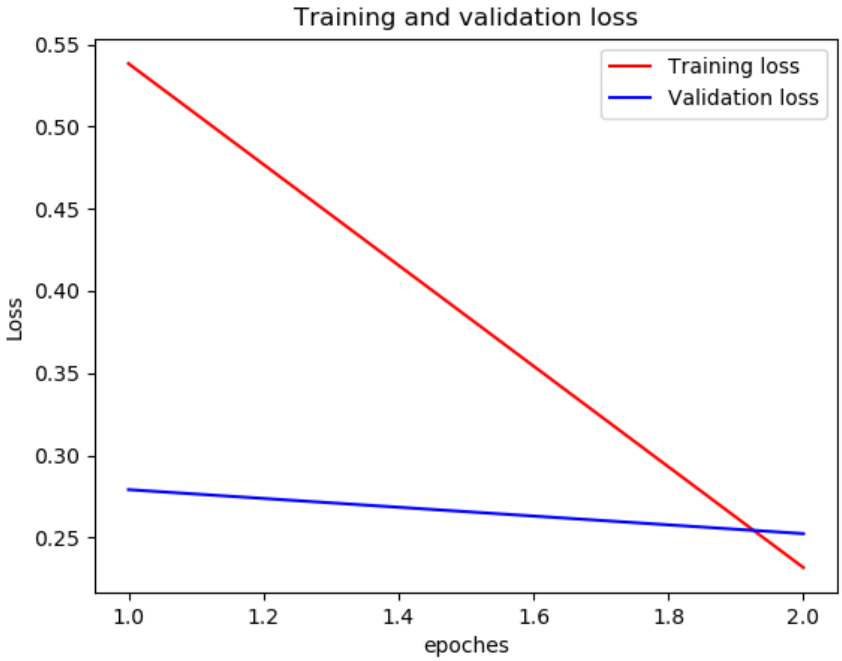

Рисунок 6 - Потери второй сети

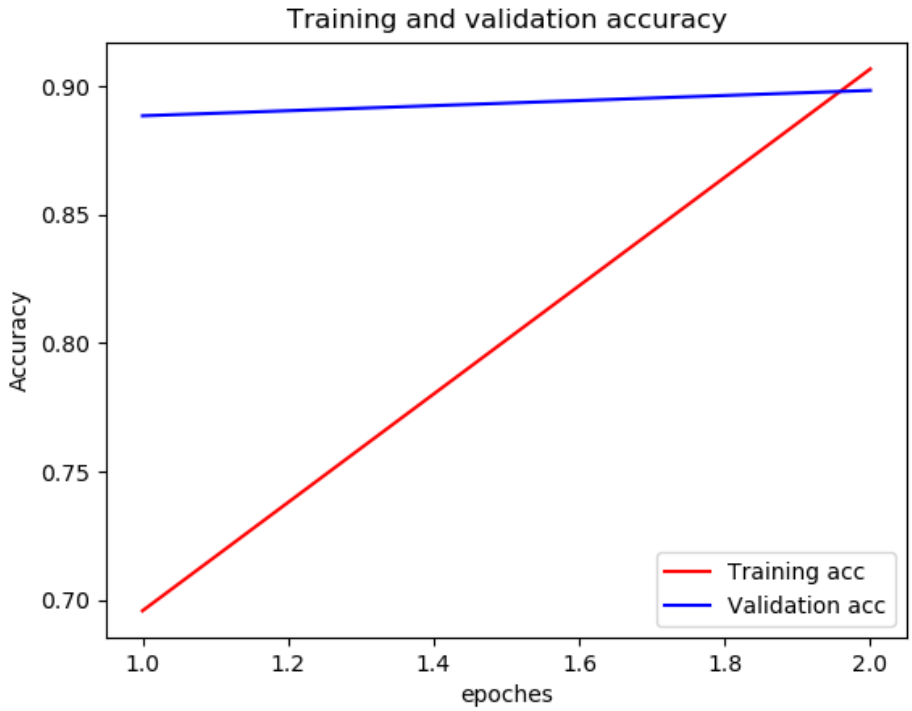

Рисунок 7 - Точность второй сети

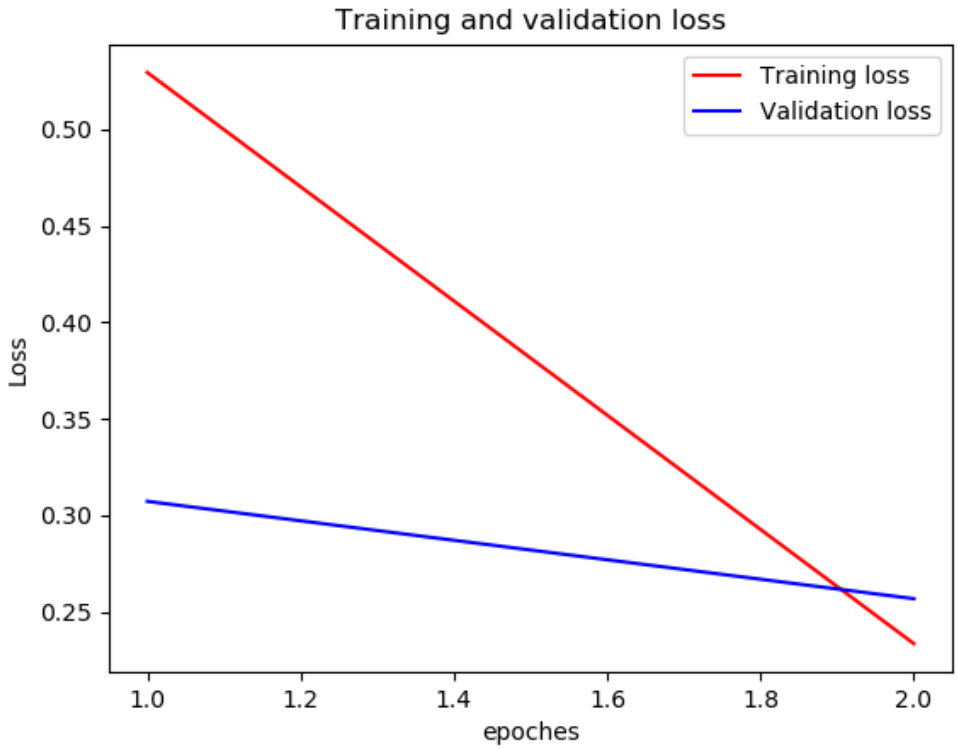

Рисунок 8 - Потери третьей сети

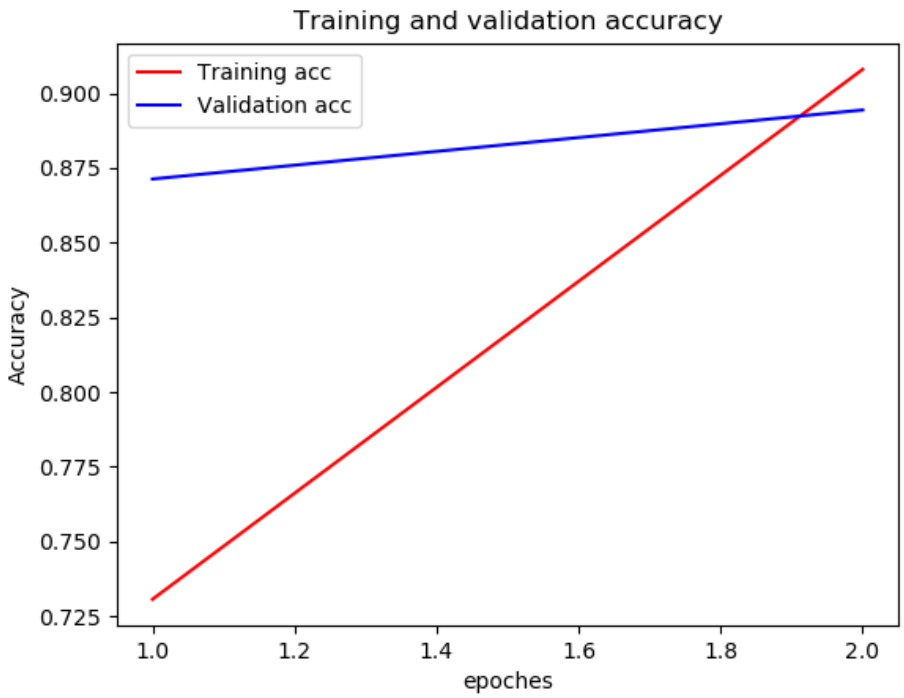

Рисунок 9 - Точность третьей сети

2. Для ансамблирования моделей была написана функция ensemble, объединение результатов работы сетей происходило по принципу среднего арифметического результатов обработки каждого обзора. Точность, достигнутая ансамблированием сетей, равна 89.95.

3. Была написана функция load_user_text для загрузки пользовательского текста и прогнозирования успеха фильма по этому тексту.

Пример текста: «this is very cool film with nice actors and scenario i like it so much» с результатом ансамбля 0.69.

Выводы:

В ходе выполнения данной лабораторной работы были построены модели сетей, прогнозирующих оценку фильма по обзорам, и проведено ансамблирование этих моделей. Также была написана функция прогнозирования оценки по пользовательскому тексту с помощью ансамблированных моделей.

ПРИЛОЖЕНИЕ А

```
import matplotlib.pyplot as plt

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv1D, MaxPooling1D, Flatten
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from numpy import loadtxt
from keras.models import load_model
from keras.datasets import imdb
from sklearn.metrics import accuracy_score
top_words = 10000
embedding_vector_length = 32
max_review_length = 500



def load_dataset():
    (X_train,         y_train),         (X_test,         y_test)         =
imdb.load_data(num_words=10000)
    data = np.concatenate((X_train, X_test), axis=0)
    targets = np.concatenate((y_train, y_test), axis=0)
    data = sequence.pad_sequences(data, maxlen=max_review_length)
    targets = np.array(targets).astype("float32")
    test_x = data[:10000]
    test_y = targets[:10000]
    train_x = data[10000:]
    train_y = targets[10000:]
    return (train_x, train_y, test_x, test_y)



def build_model_1():
    model = Sequential()
    model.add(Embedding(top_words,               embedding_vector_length,
input_length=max_review_length))
```

```
    model.add(LSTM(100))
    model.add(Dropout(0.3))
    model.add(Dense(50, activation='relu'))
    model.add(Dropout(0.3))
    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(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'))
    model.compile(loss='binary_crossentropy',           optimizer='adam',
metrics=['accuracy'])
    return model


def build_model_3():
    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(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',           optimizer='adam',
metrics=['accuracy'])
    return model
```

```
def ensemble(models, X_test, y_test):
    predict = np.array([0])
    for i in range(0, len(models)):
        predict = np.add(predict, np.array(models[i].predict(X_test)))
    print(predict / 3)
    print(accuracy_score(y_test, (predict/3).round(), normalize=False) /
100)



def test_model(model):
    X_train, y_train, X_test, y_test = load_dataset()
    results = model.fit(X_train, y_train, epochs=2, batch_size=256,
                        validation_data=(X_test, y_test))
    return (results.history['accuracy'], results.history['val_accuracy'],
results.history['loss'], results.history['val_loss'])



def plot(res_train_acc, res_test_acc, res_traing_loss, res_test_loss):
    epo = [1, 2]
    plt.plot(epo, res_traing_loss, 'r', label='Training loss')
    plt.plot(epo, res_test_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('epoches')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()
    plt.plot(epo, res_train_acc, 'r', label='Training acc')
    plt.plot(epo, res_test_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('epoches')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

mod = build_model_1()
```

```
X_train, y_train, X_test, y_test = load_dataset()
#res_train_acc,    res_test_acc,    res_traing_loss,    res_test_loss    = test_model(mod)
#plot(res_train_acc, res_test_acc, res_traing_loss, res_test_loss)


# for model in models:
#        res_train_acc,  res_test_acc,  res_traing_loss,  res_test_loss  = test_model(model)
#     model.save('model'+str(i)+'.h5')
#     i+=1
#      plot(res_train_acc, res_test_acc, res_traing_loss, res_test_loss, model)


models      =      [load_model('model0.h5'),      load_model('model1.h5'), load_model('model2.h5')]
ensemble(models, X_test, y_test)


#text separated only with spaces
def load_user_text(review, models):
    text = review.split()
    print(text)
    dict = imdb.get_word_index()
    words_num = []
    for word in text:
        word = dict.get(word)
        if word is not None and word < 10000:
            words_num.append(word)
    text = sequence.pad_sequences([words_num], maxlen=max_review_length)
    res = []
    for model in models:
        tmp = model.predict(text)
        print(tmp)
        res.append(tmp)
    print((res[0]+res[1]+res[2])/3)


string = "this is very cool film with nice actors and scenario i like it so much"
```

```
load_user_text(string, models)
```