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: Классификация обзоров фильмов

Студент гр. 7383 _____ Власов 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 выполнения задания была разработана и использована программа, код которой приведен в приложении А.

Для ансамблирования моделей было решено разработать 2 разные архитектуры сети, которые приведены на рис. 1 и 2, и использовать по 2 модели на каждой архитектуре. Таким образом, в ансамбль вошли 4 модели со схожей точностью: каждая модель, согласно model.evaluate, показывает точность от 0.88 до 0.9. Средняя точность сетей, включенных в ансамбль составила 0.8903.

```
model = Sequential()
model.add(Embedding(NUM_WORDS, embedding_vector_length, input_length=REVIEW_LENGTH))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(Dropout(0.2))
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 — Первая архитектура сети

```
model = Sequential()
model.add(Embedding(NUM_WORDS, embedding_vector_length, input_length=REVIEW_LENGTH))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Рисунок 2 — Вторая архитектура сети

На рисунках 3-6 приведены графики точности для каждой сети.

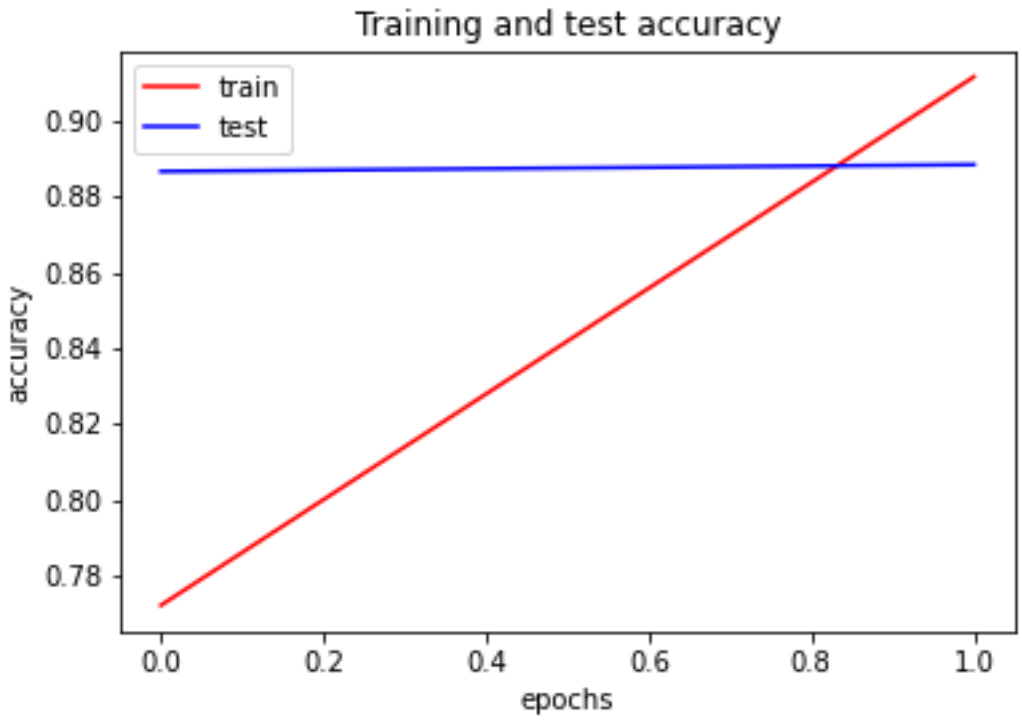

Рисунок 3 — Точность модели 1 с архитектурой 1

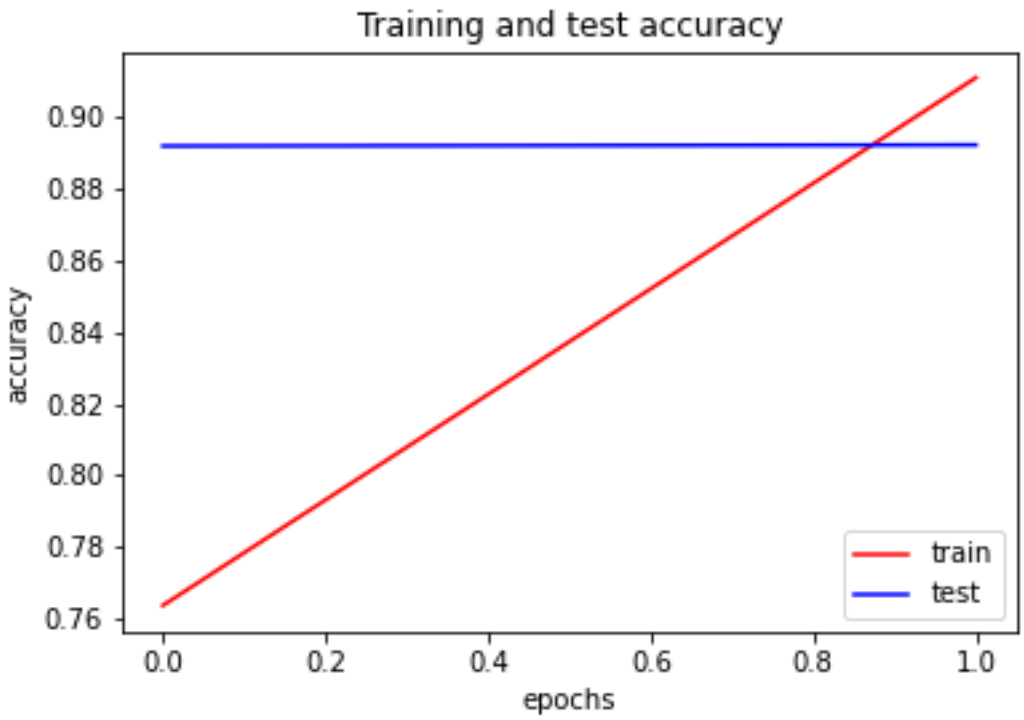

Рисунок 4 — Точность модели 2 с архитектурой 1

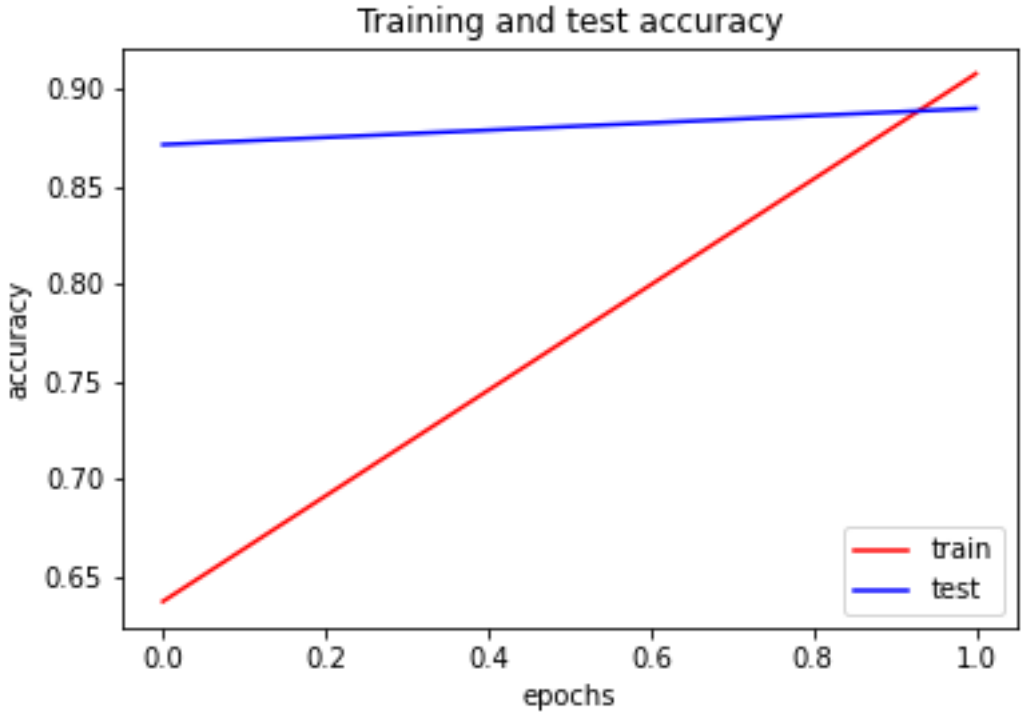

Рисунок 5 — Точность модели 1 с архитектурой 2

Рисунок 6 — Точность модели 2 с архитектурой 2

Была написана функция для получения оценки пользовательского текста с помощью предоставленного ансамбля моделей: predict(review, models). Работа функции была протестирована на нескольких коротких отзывах, результат показан на рис. 7. Можно наблюдать, что полученные оценки вполне совпадает с действительным окрасом отзывов.

```
"That's a really really cool movie!! I like it very much" for 85.83% is a good review
"There is nothing special in this film" for 52.62% is a good review
"I do hate cast and soundtracks" for 20.92% is a good review
```

Рисунок 7 — Пользовательские отзывы

Выводы.

В ходе выполнения лабораторной работы были разработаны две архитектуры сети. Несколько сетей на разработанных архитектурах были объеденены в ансамбль сетей. Также была написана функция, позволяющая получить оценку фильма по введенному обзору с помощью предоставленного ансамбля моделей.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

REVIEW_LENGTH = 500
NUM_WORDS = 10000
EPOCHS = 2
BATCH_SIZE = 250
embedding_vector_length = 32
CUSTOM_REVIEWS = [
    "That's a really really cool movie!! I like it very much",
    "There is nothing special in this film",
    "I do hate cast and soundtracks"
]



def buildModel_1():
    model = Sequential()
    model.add(Embedding(NUM_WORDS, embedding_vector_length,
input_length=REVIEW_LENGTH))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(Dropout(0.2))
    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 buildModel_2():
    model = Sequential()
    model.add(Embedding(NUM_WORDS, embedding_vector_length,
input_length=REVIEW_LENGTH))
    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.25))
```

```
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    return model


def loadData():
    (training_data, training_targets), (testing_data,
testing_targets) = imdb.load_data(num_words=NUM_WORDS)
    data = np.concatenate((training_data, testing_data), axis=0)
    targets = np.concatenate((training_targets,
testing_targets), axis=0)
    data = sequence.pad_sequences(data, maxlen=REVIEW_LENGTH)
    targets = np.array(targets).astype("float32")
    return data, targets


def createPlots(history, num):
    plt.title('Training and test accuracy')
    plt.plot(history.history['accuracy'], 'r', label='train')
    plt.plot(history.history['val_accuracy'], 'b', label='test')
    plt.xlabel("epochs")
    plt.ylabel("accuracy")
    plt.legend()
    plt.savefig("%s_acc.png" % num, format='png')
    plt.clf()

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


def trainModels():
    data, targets = loadData()
    model_1 = buildModel_1()
    model_2 = buildModel_1()
    model_3 = buildModel_2()
    model_4 = buildModel_2()
    hist_1 = model_1.fit(data[10000:], targets[10000:],
epochs=EPOCHS, batch_size=BATCH_SIZE,
                         validation_data=(data[:10000],
targets[:10000]))
    hist_2 = model_2.fit(data[10000:], targets[10000:],
epochs=EPOCHS, batch_size=BATCH_SIZE,
```

```
                          validation_data=(data[:10000],
targets[:10000]))
    hist_3 = model_3.fit(data[10000:], targets[10000:],
epochs=EPOCHS, batch_size=BATCH_SIZE,
                          validation_data=(data[:10000],
targets[:10000]))
    hist_4 = model_4.fit(data[10000:], targets[10000:],
epochs=EPOCHS, batch_size=BATCH_SIZE,
                          validation_data=(data[:10000],
targets[:10000]))
    createPlots(hist_1, 1)
    createPlots(hist_2, 2)
    createPlots(hist_3, 3)
    createPlots(hist_4, 4)
    loss_1, acc_1 = model_1.evaluate(data[:10000],
targets[:10000])
    loss_2, acc_2 = model_2.evaluate(data[:10000],
targets[:10000])
    loss_3, acc_3 = model_3.evaluate(data[:10000],
targets[:10000])
    loss_4, acc_4 = model_4.evaluate(data[:10000],
targets[:10000])
    model_1.save("m1.h5")
    model_2.save("m2.h5")
    model_3.save("m3.h5")
    model_4.save("m4.h5")
    print("Ensemble accuracy: %s" % ((acc_1 + acc_2 + acc_3 +
acc_4) / 4))


def predict(review, models):
    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] < NUM_WORDS:
            encoded.append(indexes[w])
    review = sequence.pad_sequences([encoded],
maxlen=REVIEW_LENGTH)
    pred = 0
    for model in models:
        pred += model.predict(review)[0][0]
    return pred / len(models)


def testCustomReview():
    model_1 = load_model("m1.h5")
    model_2 = load_model("m2.h5")
```

```
    model_3 = load_model("m3.h5")
    model_4 = load_model("m4.h5")
    for review in CUSTOM_REVIEWS:
        print('"%s" for %.2f%% is a good review' % (review,
predict(review, [model_1, model_2, model_3, model_4]) * 100))


trainModels()
testCustomReview()
```