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

ОТЧЕТ

по лабораторной работе №7
по дисциплине «Искусственные нейронные сети»
Тема: «Классификация обзоров фильмов»

Студент гр. 7382 _____ Находько А.Ю.

Преподаватель _____ Жукова Н.В.

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.

Задачи работы.

•Ознакомиться с рекуррентными нейронными сетями

•Изучить способы классификации текста
•Ознакомиться с ансамблированием сетей
•Построить ансамбль сетей, который позволит получать точность не менее 97

Требования работы.

1. Найти набор оптимальных ИНС для классификации текста

2. Провести ансамблирование моделей
3. Написать функцию/функции, которые позволят загружать текст и получать результат ансамбля сетей
4. Провести тестирование сетей на своих текстах (привести в отчете)

Основные теоретические положения.

Искусственные нейронные сети - совокупность моделей, которые представляют собой сеть элементов - искусственных нейронов, связанных между собой синаптическими соединениями.

Нейронные сети используются как среда, в которой осуществляется адаптивная настройка параметров дискриминантных функций. Настройка

происходит при последовательном предъявлении обучающих выборок образов из разных классов. Обучение - такой выбор параметров нейронной сети, при котором сеть лучше всего справляется с поставленной проблемой.

Нейрон — элемент, преобразующий входной сигнал по функции.

Сумматор — элемент, осуществляющий суммирование сигналов поступающих на его вход.

Синапс — элемент, осуществляющий линейную передачу сигнала.

Экспериментальные результаты.

В ходе выполнения лабораторной работы была разработана ИНС, которая классифицирует обзоры фильмов. Код данной ИНС представлен в Приложении А.

Были разработаны 2 архитектуры сети для разрешения задачи ансамблирования. На каждой из архитектур — по 2 модели. Получается что в ходе ансамблирования создаются 4 модели.

1 модель:

```
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'])
```

2 модель:

```
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'])
```

Была написана функция, которые позволят загружать текст и получать результат ансамбля сетей. Также полученная функция была протестирована на своих текстах. Полученные результаты:

"Very good movie, amazing play of actors, great scenery and soundtracks deserve separate praise" for 71.70% is a good review

"Bad movie, wasted time watching this movie, the worst protagonist I've ever seen" for 38.02% is a good review

Ниже на рисунках 1, 2 приведены результаты точности модели для 1, 2 архитеркуры, 1, 2, 3, 4 модели соответственно.

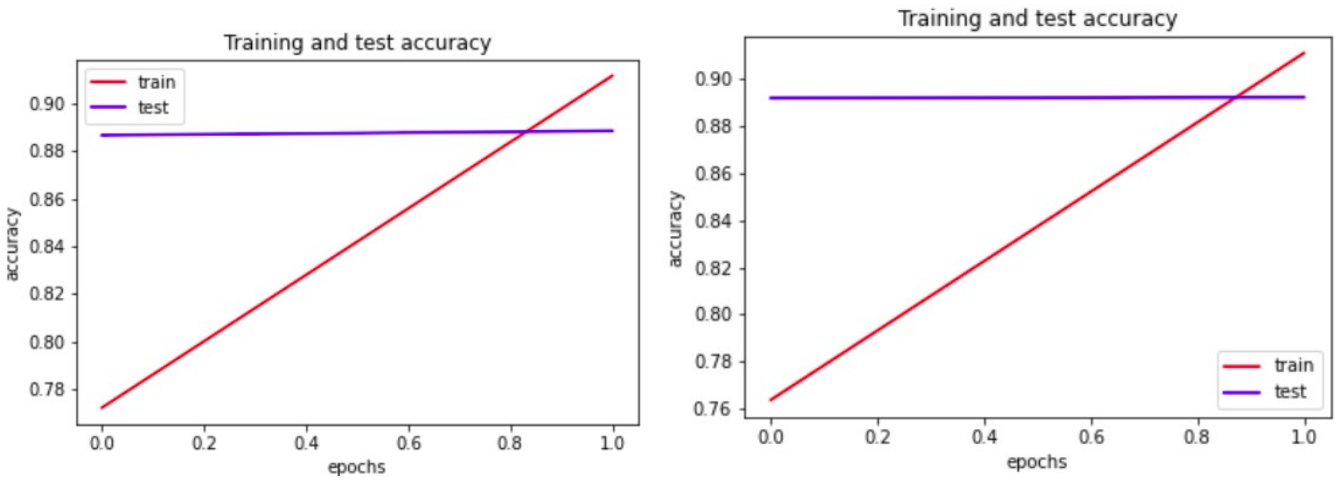

Рисунок 1 — Результат точности модели для 1 архитектуры

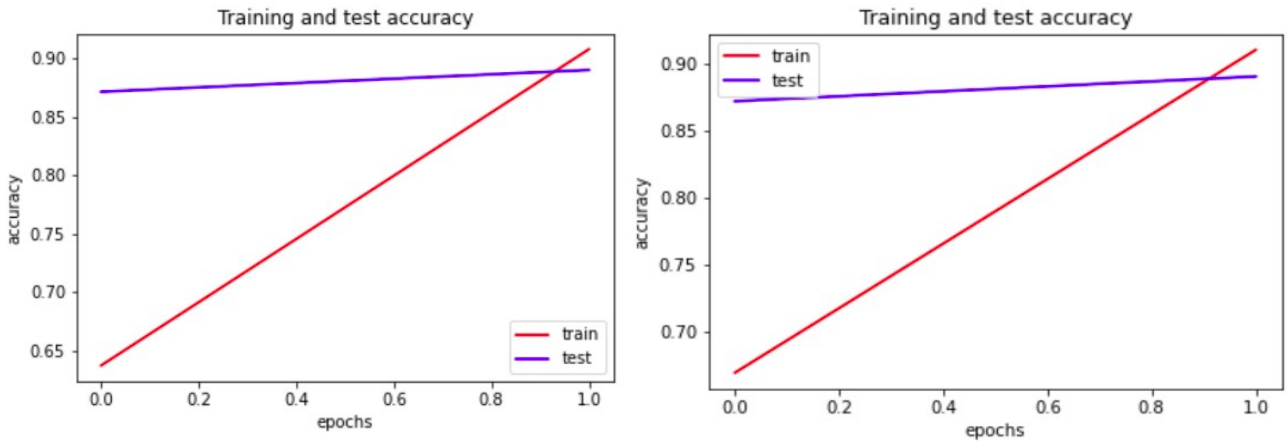

Рисунок 2 — Результат точности модели для 2 архитектуры

Согласно полученным графикам, каждая из моделей показывает точность от 0.88 до 0.9. Средняя точность моделей составила 0.8905.

Выводы.

В ходе выполнения лабораторной работы была реализована программа классифицирующая прогнозы фильмов, изучены способы классификации текста. Также ознакомился с ансамблированием сетей, построил ансамбль сетей, который позволил получить высокую точность.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
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
import numpy as np
import string

REVIEW_LENGTH = 500
NUM_WORDS = 10000
EPOCHS = 2
BATCH_SIZE = 250
embedding_vector_length = 32
CUSTOM_REVIEWS = [
"Very good movie, amazing play of actors, great scenery and soundtracks
deserve separate praise",
"Bad movie, wasted time watching this movie, the worst protagonist I've
ever seen"
]


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()
```