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. 7382 _____ Государкин Я.С.

Преподаватель _____ Жангиров Т.Р

Санкт-Петербург

2020

Задание

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.

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

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)

Теоретическая часть

С помощью анализа настроений можно определить отношение (например, настроение) человека к тексту, взаимодействию или событию. Поэтому

сентимент-анализ относится к области обработки естественного языка, в которой смысл текста должен быть расшифрован для извлечения из него тональности и настроений.

Датасет IMDb состоит из 50 000 обзоров фильмов от пользователей, помеченных как положительные (1) и отрицательные (0).

- Рецензии предварительно обрабатываются, и каждая из них кодируется последовательностью индексов слов в виде целых чисел.
- Слова в обзорах индексируются по их общей частоте появления в датасете. Например, целое число «2» кодирует второе наиболее частое используемое слово.
- 50 000 обзоров разделены на два набора: 25 000 для обучения и 25 000 для тестирования.

Ход работы

1,2. Найти набор оптимальных ИНС для классификации текста и провести ансамблирование моделей.

Воспользуемся рекуррентными нейросетями, а именно слоем GRU (упрощённый LSTM), они хорошо подходят для обработки текстов, потому что могут хранить свое состояние и принимают текущее решение с учётом предыдущих решений.

Также, помимо рекуррентных слоёв будем пользоваться одномерными свёрткой и пулингом. В итоге должна получиться составная нейросеть выдающая меньшее время обучения без особой потери точности, возможно даже с увеличением. Dropout сделаем около 0.1-0.3, и воспользуемся L2 регуляризацией, чтобы уменьшить оверфит модели, потому что реккурентные сети более склонны к нему, чем обычные.

Первая неплохая архитектура даёт точность на валидационных - 89,6%, на тестовых данных ~ 90,87%.

Результат немного лучше, чем в предыдущей лабораторной, однако не принципиально. Добавилось 2 блока одномерной свёртки и пулинга с L2 и дропаутом.

Оптимизатор: Adam, lr=0.001

batch_size = 200

loss_func: binary_crossentropy

epochs = 2

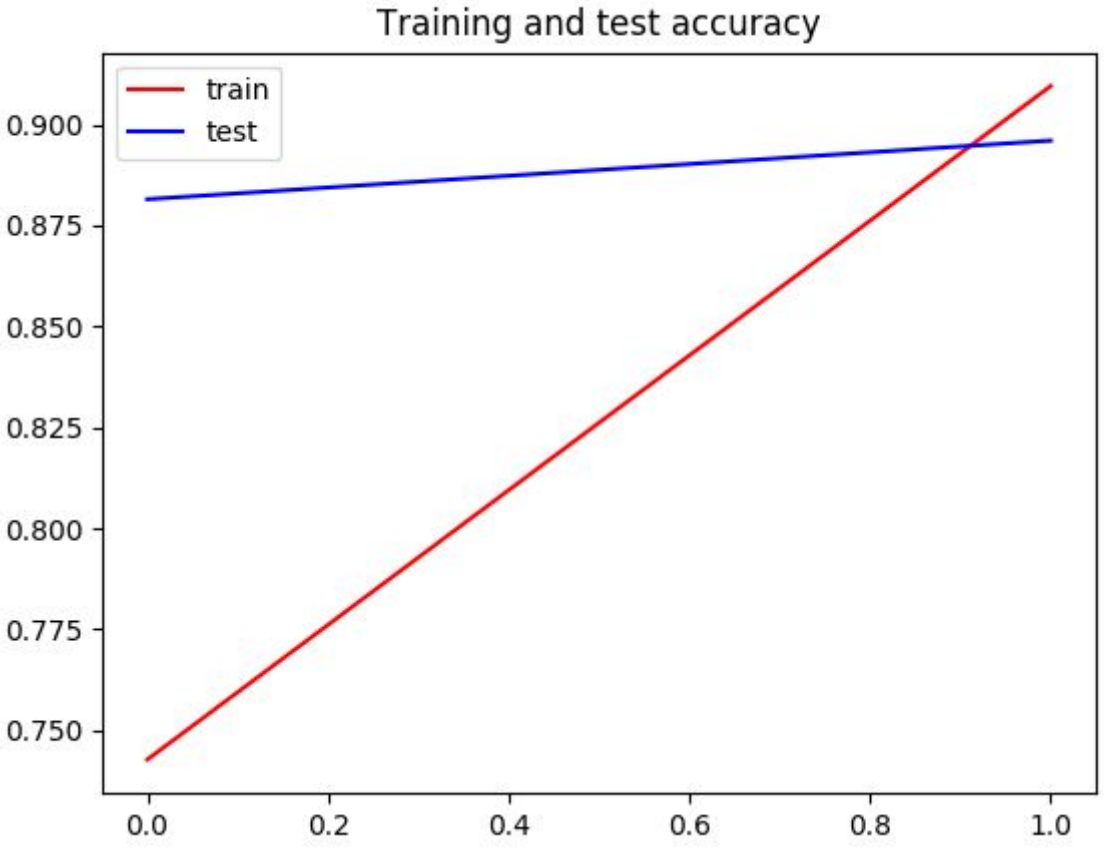
макс. кол-во слов в обзоре: 250

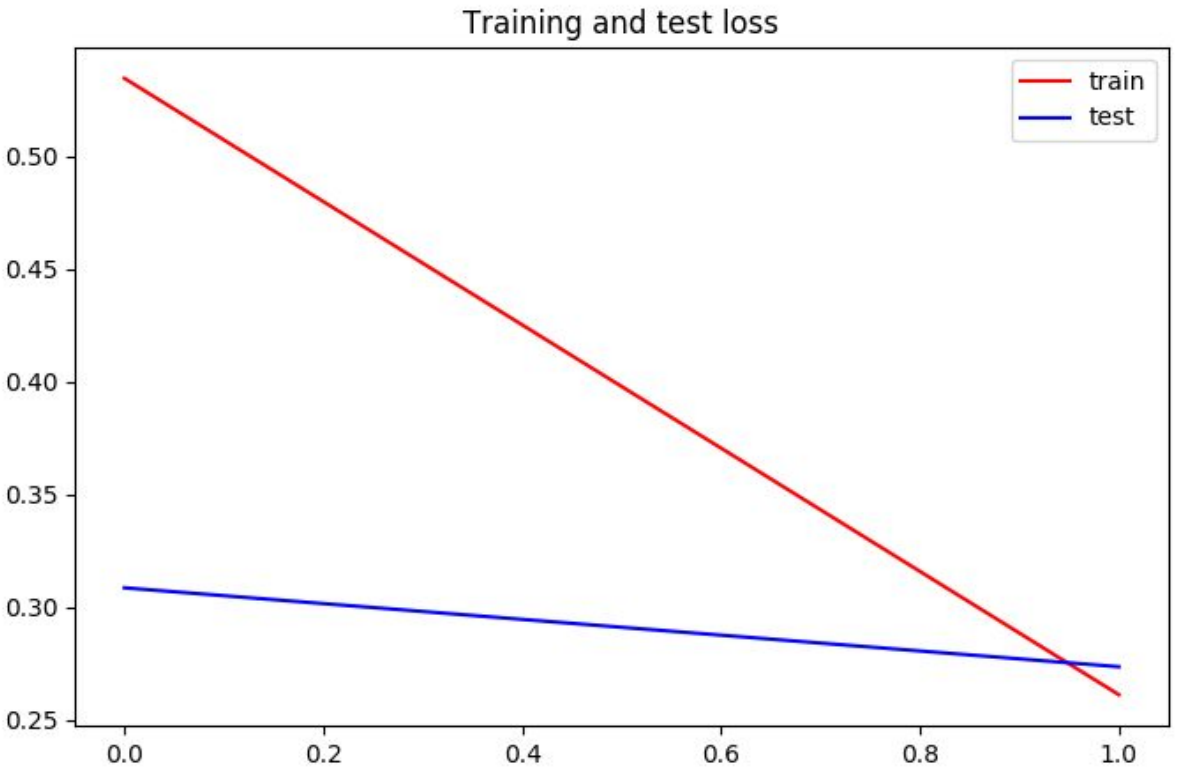
макс. размер словаря слов: 10000

Model:

```
self.features = Sequential([
    Embedding(input_dim=self.max_features, output_dim=1024, input_length=self.max_len),
    Dropout(0.3),

    Conv1D(filters=32, kernel_size=3, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    MaxPool1D(pool_size=2),
    Dropout(0.1),

    Conv1D(filters=64, kernel_size=3, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    MaxPool1D(pool_size=2),
    Dropout(0.1),

    GRU(units=64, use_bias=True, activation='relu', kernel_regularizer=regularizers.l2(0.001)),

    Dense(input_dim=64, units=32, activation='relu'),
    Dropout(0.2),

    Dense(units=1, activation='sigmoid')
])
```

Графики точности и ошибки см. ниже.





Вторая неплохая архитектура даёт точность на валидационных - 90,3%, на тестовых данных ~ 91,4%.

Результат немного лучше, чем у первой схемы, однако тоже не принципиально. Поскольку был запас по оверфиту я усложнил модель за счёт более масштабной свёртки и предсказания стали немного лучше.

Оптимизатор: Adam, lr=0.001

batch_size = 200

loss_func: binary_crossentropy

epochs = 2

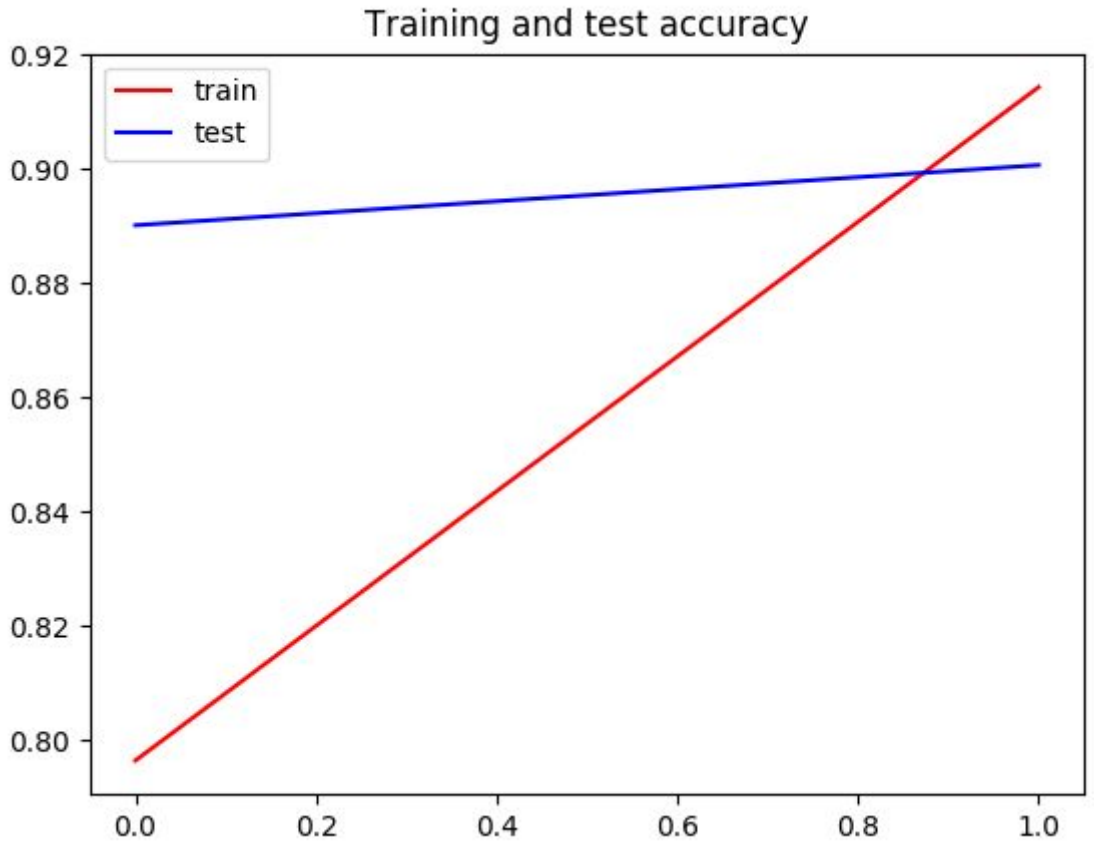
макс. кол-во слов в обзоре: 250

макс. размер словаря слов: 10000

Модель:

```
self.features = Sequential([
    Embedding(input_dim=self.max_features, output_dim=2048, input_length=self.max_len),
    Dropout(0.3),

    Conv1D(filters=64, kernel_size=3, activation='relu', use_bias=True, kernel_regularizer=regularizers.l2(0.0015)),
    MaxPool1D(pool_size=2),
    Dropout(0.1),

    Conv1D(filters=128, kernel_size=3, activation='relu', use_bias=True, kernel_regularizer=regularizers.l2(0.0015)),
    MaxPool1D(pool_size=2),
    Dropout(0.1),

    GRU(units=128, use_bias=True, activation='relu', kernel_regularizer=regularizers.l2(0.001)),

    Dense(input_dim=128, units=16, activation='relu'),
    Dropout(0.2),

    Dense(units=1, activation='sigmoid')
])
```

График точности см. ниже.



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

Были написаны функции, с помощью которых можно получить из массива строк (обзоров) массив представлений в виде индексов слов в imdb датасете и подготовлены для прогона через модель. Код функций

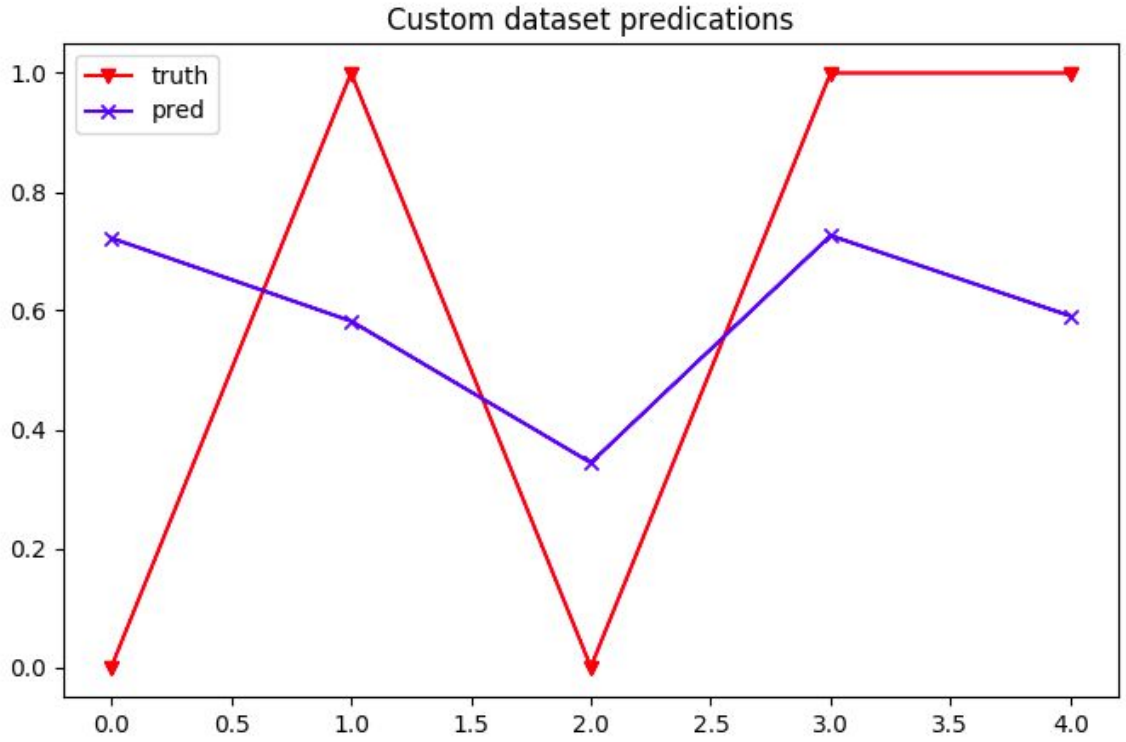
```
# funcs for generating data for net from reviews
def encodeUserInput(string, dic):
    str_list = re.split('; |, |\*|\n| ', string)
    for i in range(len(str_list)):
        idx = dic.get(str_list[i])
        str_list[i] = idx
    return str_list
```

```
def generateReviews(reviews,dic):
    for i in range(len(reviews)):
        reviews[i] = encodeUserInput(reviews[i], dic)
    return reviews
```

После обучения модели на основной выборке, прогоним вручную написанный датасет из 5 обзоров через модель и посмотрим на точность.

```
custom_x = ["it is very boring, bad film",
            "fantastic, the best film ever",
            "i think this film is the worst film i've seen, nothing interesting, junk",
            "fantastic film, wonderful casting, good job, creators",
            "beautiful picture, good scenario, it's amazing"]
custom_y = [0., 1., 0., 1., 1.]
```

Результат: точность предсказания второй нейроннойсетью настроения обзора: 80%, т.е 4 из 5



Выводы.

В ходе лабораторной работы мы ознакомились с задачей классификации обзоров из встроенного в keras датасета IMDB, подобрали архитектуру, дающую точность на тестовой выборке ~ 91,4%. Было показано, что добавление нескольких слоёв свёртки и пулинга немного увеличивает точность предсказания. Также, была написана функция для подготовки вручную введённых обзоров к проверке моделью и продемонстрирован результат работы сети на выборке из 5 вручную написанных обзоров. Код в приложении А.

ПРИЛОЖЕНИЕ А

```
import tensorflow.keras as keras
import re

from tensorflow.keras import losses
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dropout, Dense, Embedding, GRU, Conv1D, MaxPool1D
from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing import sequence
from tensorflow.keras import initializers
from tensorflow.keras import regularizers

from tensorflow.keras import datasets

from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np



custom_x = ["it is very boring, bad film",
        "fantastic, the best film ever",
        "i think this film is the worst film i've seen, nothing interesting, junk",
        "fantastic film, wonderful casting, good job, creators",
        "beautiful picture, good scenario, it's amazing"]
custom_y = [0., 1., 0., 1., 1.]



class ReviewsClassifier(keras.Model):
   def __init__(self, max_features=5000, max_len=1000):
      super(ReviewsClassifier, self).__init__()

      self.weight_init = initializers.normal
      self.max_features = max_features
      self.max_len = max_len

      self.features = Sequential([
         Embedding(input_dim=self.max_features, output_dim=1024, input_length=self.max_len),
         Dropout(0.3),
```

```
            Conv1D(filters=32, kernel_size=3, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
            MaxPool1D(pool_size=2),
            Dropout(0.1),

            Conv1D(filters=64, kernel_size=3, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
            MaxPool1D(pool_size=2),
            Dropout(0.1),

            GRU(units=64, use_bias=True, activation='relu', kernel_regularizer=regularizers.l2(0.001)),

            Dense(input_dim=64, units=32, activation='relu'),
            Dropout(0.2),

            Dense(units=1, activation='sigmoid')
        ])

    def call(self, inputs):
        x = self.features(inputs)
        return x


# funcs for generating data for net from reviews
def encodeUserInput(string, dic):
    str_list = re.split('; |, |\*|\n| ', string)
    for i in range(len(str_list)):
        idx = dic.get(str_list[i])
        str_list[i] = idx
    return str_list


def generateReviews(reviews,dic):
    for i in range(len(reviews)):
        reviews[i] = encodeUserInput(reviews[i], dic)
    return reviews


# constants
batch_size = 200
epochs = 2
```

```
max_len = 250
max_features = 10000

# load data
(x_train, y_train), (x_test, y_test) = datasets.imdb.load_data(seed=911, num_words=max_features)

# custom data testing
index = datasets.imdb.get_word_index()
dic = dict(index)

custom_x = generateReviews(custom_x, dic)

# prepare data
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
custom_x = sequence.pad_sequences(custom_x, maxlen=max_len)

X = np.concatenate((x_train, x_test))
Y = np.concatenate((y_train, y_test))

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.05, random_state=123)

print(x_train.shape)

y_test = np.asarray(y_test).astype("float32")
y_train = np.asarray(y_train).astype("float32")
custom_y = np.asarray(custom_y).astype("float32")

# init model
classifier = ReviewsClassifier(max_features=max_features,max_len=max_len)
optimizer = optimizers.Adam(lr=0.001)
loss = losses.BinaryCrossentropy()

classifier.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
H = classifier.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)

# testing
test_loss, test_acc = classifier.evaluate(x_test, y_test)
print('test_acc:', test_acc)
```

```
# custom testing
custom_loss, custom_acc = classifier.evaluate(custom_x, custom_y)
print('custom_acc:', custom_acc)
preds = classifier.predict(custom_x)

# plot
plt.figure(3,figsize=(8,5))
plt.title("Custom dataset predications")
plt.plot(custom_y, 'r', marker='v', label='truth')
plt.plot(preds, 'b', marker='x', label='pred')
plt.legend()
plt.show()
plt.clf()

plt.figure(1,figsize=(8,5))
plt.title("Training and test accuracy")
plt.plot(H.history['acc'], 'r', label='train')
plt.plot(H.history['val_acc'], 'b', label='test')
plt.legend()
plt.show()
plt.clf()

plt.figure(2,figsize=(8,5))
plt.title("Training and test loss")
plt.plot(H.history['loss'], 'r', label='train')
plt.plot(H.history['val_loss'], 'b', label='test')
plt.legend()
plt.show()
plt.clf()
```