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»

Студентка гр. 7381 _____ Алясова А.Н.

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.

Задачи.

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

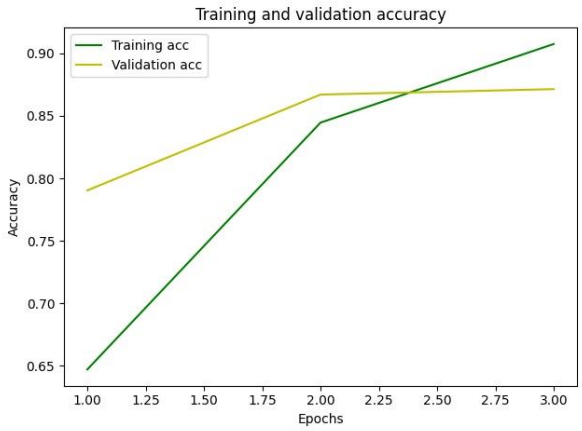
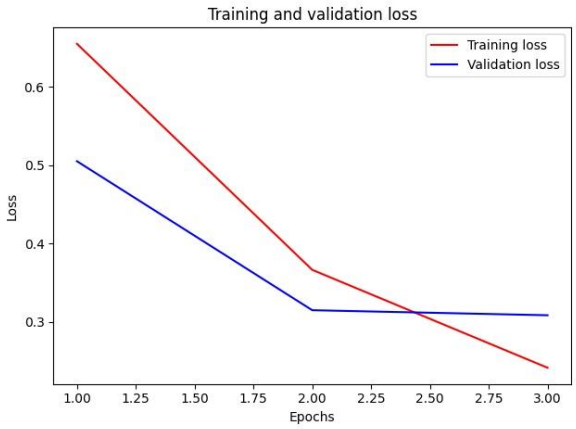
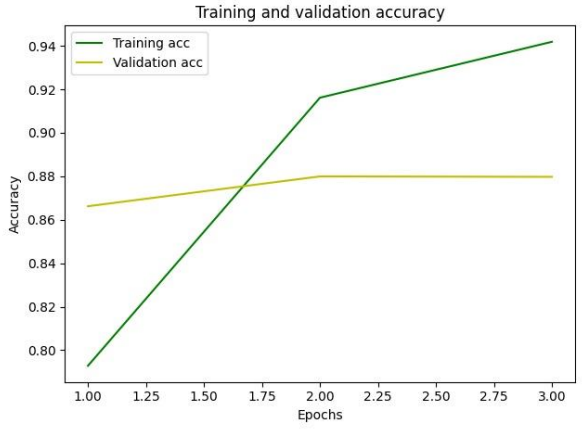
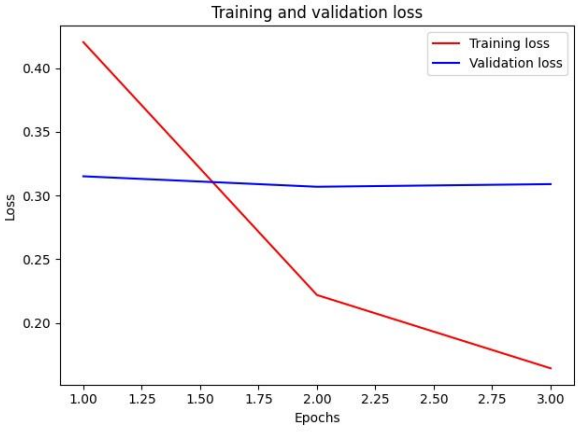
Требования.

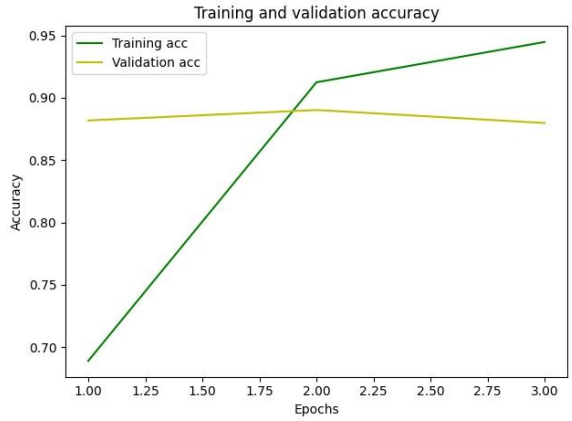
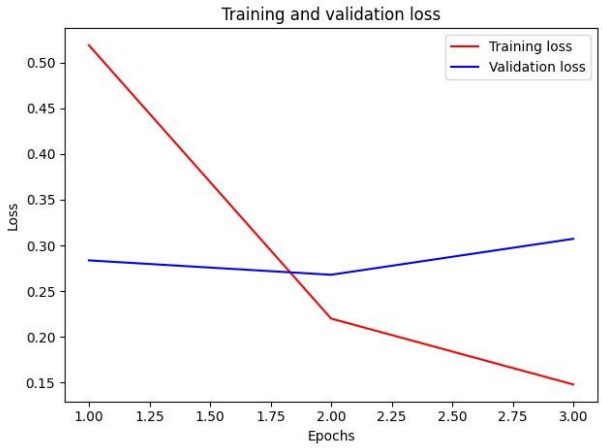
- Найти набор оптимальных ИНС для классификации текста
- Провести ансамблирование моделей
- Написать функцию/функции, которые позволят загружать текст и получать результат ансамбля сетей
- Провести тестирование сетей на своих текстах (привести в отчете)

Ход работы.

В ходе работы была создана и обучена модель искусственной нейронной сети в соответствии с условиями (код представлен в приложении).

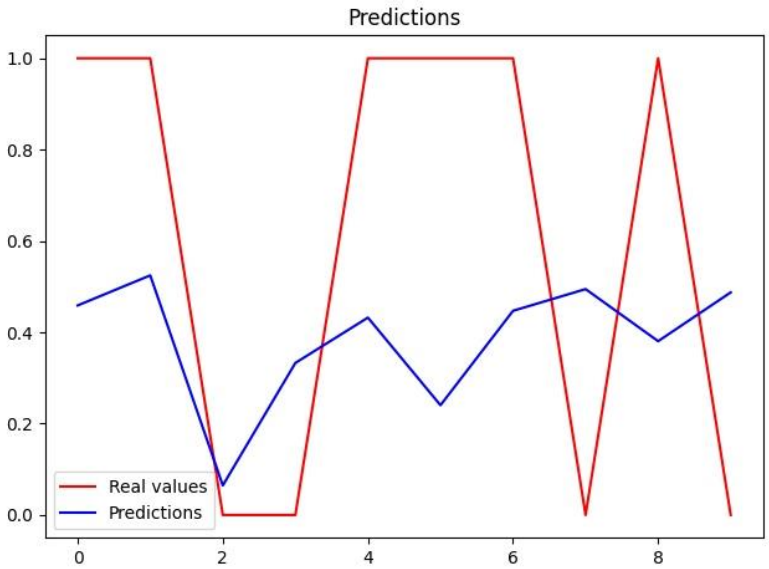
Были построены разные модели. Обучим их и ассемблируем. Результаты представлены в следующей таблице.

<table>
<tr><td colspan="2">Простая рекуррентная сеть</td></tr>
<tr><td>График точности</td><td>График потерь</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Рекуррентная сверточная сеть</td></tr>
<tr><td>График точности</td><td>График потерь</td></tr>
<tr><td> 
</td><td> 
</td></tr>
</table>

Сверточная сеть	
График точности	График потерь
	

В итоге, наиболее удачным ансамблем является всех трех моделей. Точности ансамблей: первая сеть — 86.6%, вторая сеть — 87.2%, третья сеть — 89.1%, ансамбль первой и второй сетей — 88.7%, ансамбль второй и третьей сетей — 89,2%, ансамбль первой и третьей сетей — 88,6%, ансамбль всех трех сетей — 89,3%.

Также была написана функция загрузки собственного текста user_load. Результаты работы ансамбля представлены на рис. 8:



Выводы.

Были найдены оптимальные сети, построен ансамбль сетей. Была продемонстрирована работа ансамбля на собственном тексте. Результат оказался лучше, чем в предыдущей работе.

ПРИЛОЖЕНИЕ

Исходный код

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

EPOCHS = 3
BATCH_SIZE = 64
max_review_length = 500
top_words = 10000
embedding_vector_length = 32

strings = ["A gorgeous movie, I'll review it again. I cried with
emotion. I really liked it!",
           "Very interesting film. Great story and talented actors.",
           "Who is the producer of this shit. I had enough for 10
minutes and I turned it off.",
           "A very flashy plot. Scenes are typical and predictable.
Actors replay.",
           "Great movie with a great story",
           "This film was just brilliant casting location scenery
story direction everyone's really suited the part they played and you
could just imagine being there robert",
           "Is an amazing actor and now the same being director!",
           "It is so bad.",
           "This film shows humanity in all its glory. We can try to
claim the rich are evil and the poor deserve better but these are
delusions we create to make us feel better. People are terrible. They
are selfish. They are greedy. They are unkind. It's part of human
nature. No matter how much you have, or how little, humanity still
exists, and humanity is the problem.",
           "I'm crying that I spent my time on it."]

val = [1, 1, 0, 0, 1, 1, 1, 0, 1, 0]

(training_data, training_targets), (testing_data, testing_targets) =
imdb.load_data(num_words=10000)
#data = np.concatenate((training_data, testing_data), axis=0)
```

```
#targets = np.concatenate((training_targets, testing_targets), axis=0)

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

def create_graphics(history):
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    epochs = range(1, len(loss) + 1)
    print(len(loss))

    # График потерь
    plt.plot(epochs, loss, 'r', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()

    #График точности
    plt.plot(epochs, acc, 'g', label='Training acc')
    plt.plot(epochs, val_acc, 'y', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

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

```
    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(Dropout(0.3))
    model.add(LSTM(50))
    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_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(Dropout(0.3))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.3))
    model.add(Flatten())

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

def model_fit(model):
    model.fit(training_data, training_targets,
validation_data=(testing_data, testing_targets), epochs=EPOCHS,
batch_size=BATCH_SIZE)
    scores = model.evaluate(training_data, training_targets,
verbose=0)
    print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
def train_models():
    model_1 = build_model_1()
    model_2 = build_model_2()
    model_3 = build_model_3()

    history1 = model_1.fit(training_data, training_targets,
validation_data=(testing_data, testing_targets), epochs=EPOCHS,
                  batch_size=BATCH_SIZE)
    scores = model_1.evaluate(testing_data, testing_targets,
verbose=0)
    print("Accuracy: %.2f%%" % (scores[1] * 100))
    model_1.save('model1.h5')
    create_graphics(history1)

    history2 = model_2.fit(training_data, training_targets,
validation_data=(testing_data, testing_targets), epochs=EPOCHS,
                  batch_size=BATCH_SIZE)
    scores = model_2.evaluate(testing_data, testing_targets,
verbose=0)
    print("Accuracy: %.2f%%" % (scores[1] * 100))
    model_2.save('model2.h5')
    create_graphics(history2)

    history3 = model_3.fit(training_data, training_targets,
validation_data=(testing_data, testing_targets), epochs=EPOCHS,
                  batch_size=BATCH_SIZE)
    scores = model_3.evaluate(testing_data, testing_targets,
verbose=0)
    print("Accuracy: %.2f%%" % (scores[1] * 100))
    model_3.save('model3.h5')
    create_graphics(history3)

def ensembling_models(testing_targets):
    model_1 = load_model("model1.h5")
    model_2 = load_model("model2.h5")
    model_3 = load_model("model3.h5")

    predictions1 = model_1.predict(testing_data)
    predictions2 = model_2.predict(testing_data)
    predictions3 = model_3.predict(testing_data)

    predictions = np.divide(np.add(predictions1, predictions2,
predictions3), 3)
```

```
    testing_targets = np.reshape(testing_targets, (25000, 1))
    predictions = np.greater_equal(predictions, np.array([0.5]))
    predictions = np.logical_not(np.logical_xor(predictions,
testing_targets))
    acc = predictions.mean()
    print("Accuracy of ensembling models is %s" % acc)
##########################################################################
########################################

def user_load():
    dictionary = dict(imdb.get_word_index())
    test_x = []
    test_y = np.array(val).astype("float32")
    for string in strings:
        string = string.lower()
        words = string.replace(',', ' ').replace('.', '
').replace('?', ' ').replace('\n', ' ').split()
        num_words = []
        for word in words:
            word = dictionary.get(word)
            if word is not None and word < 10000:
                num_words.append(word)
        test_x.append(num_words)
    test_x = sequence.pad_sequences(test_x, maxlen=max_review_length)
    model1 = load_model("model1.h5")
    model2 = load_model("model2.h5")
    model3 = load_model("model3.h5")

    predictions1 = model1.predict(test_x)
    predictions2 = model2.predict(test_x)
    predictions3 = model3.predict(test_x)
    predictions = np.divide(np.add(predictions1, predictions2,
predictions3), 3)

    plt.title("Predictions")
    plt.plot(test_y, 'r', label='Real values')
    plt.plot(predictions, 'b', label='Predictions')
    plt.legend()
    plt.show()
    plt.clf()

    predictions = np.greater_equal(predictions, np.array([0.5]))
    test_y = np.reshape(test_y, (10, 1))
    predictions = np.logical_not(np.logical_xor(predictions, test_y))
```

```
    _, acc1 = model1.evaluate(test_x, test_y)
    _, acc2 = model2.evaluate(test_x, test_y)
    print("Validation accuracy of 1st model is %s" % acc1)
    print("Validation accuracy of 2nd model is %s" % acc2)
    acc = predictions.mean()
    print("Validation accuracy of ensembling models is %s" % acc)

#########################################################################
########################################
train_models()
ensembling_models(testing_targets)
user_load()
```