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»

Студентка гр. 8383 Сырцова Е.А.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2021

Цель

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

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)

Ход работы

В данной лабораторной работе использовался датасет IMDb, встроенный в библиотеку Keras.

После загрузки данных были разработаны две модели сети:

рекуррентная

```
model_1 = Sequential()
model_1.add(Embedding(top_words, embedding_vector_length, input_length=max_review_length))
model_1.add(LSTM(100))
model_1.add(Dropout(0.4))
model_1.add(Dense(64, activation='relu'))
model_1.add(Dropout(0.3))
model_1.add(Dense(1, activation='sigmoid'))
```

рекуррентная сверточная

```
model_2 = Sequential()
model_2.add(Embedding(top_words, embedding_vector_length, input_length=max_review_length))
model_2.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model_2.add(MaxPooling1D(pool_size=2))
model_2.add(Dropout(0.3))
model_2.add(LSTM(100))
model_2.add(Dropout(0.3))
model_2.add(Dense(1, activation='sigmoid'))
```

Графики точности и ошибки после обучения сетей представлены на рис.1-4.

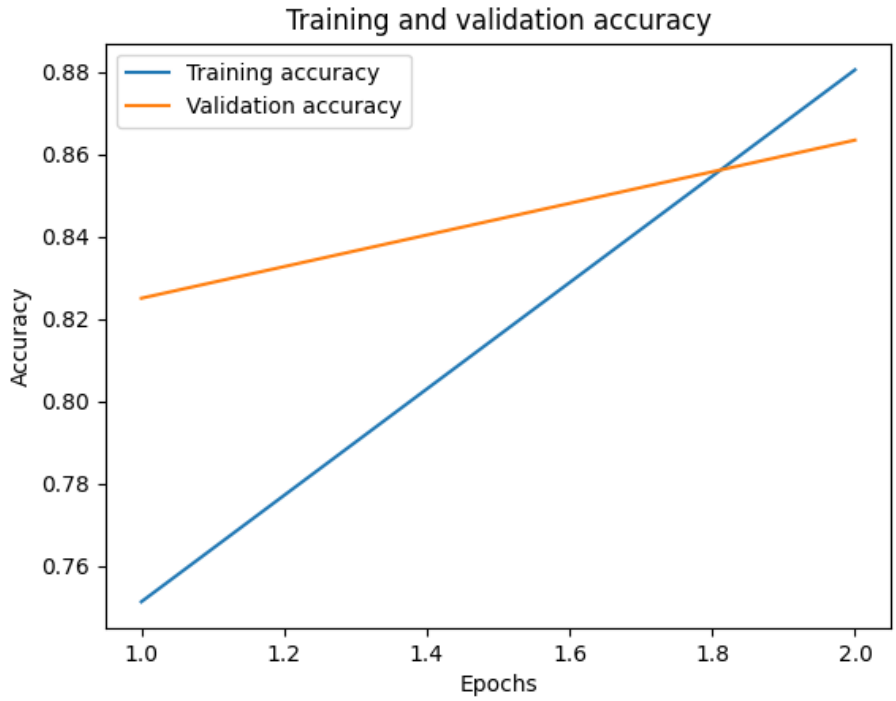

Рисунок 1 – График точности рекуррентной сети

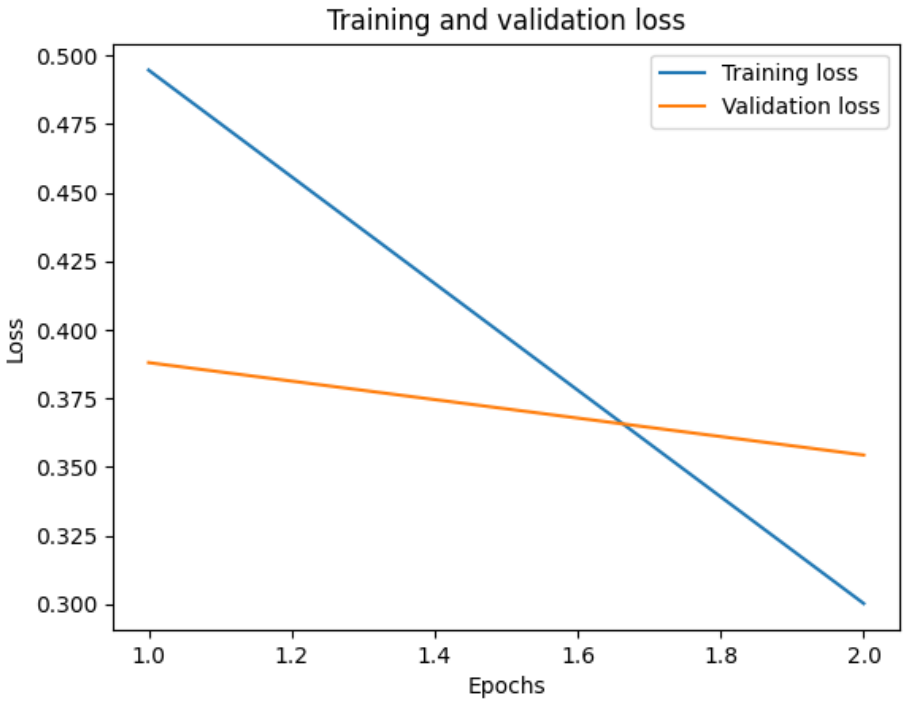

Рисунок 2 – График ошибки рекуррентной сети

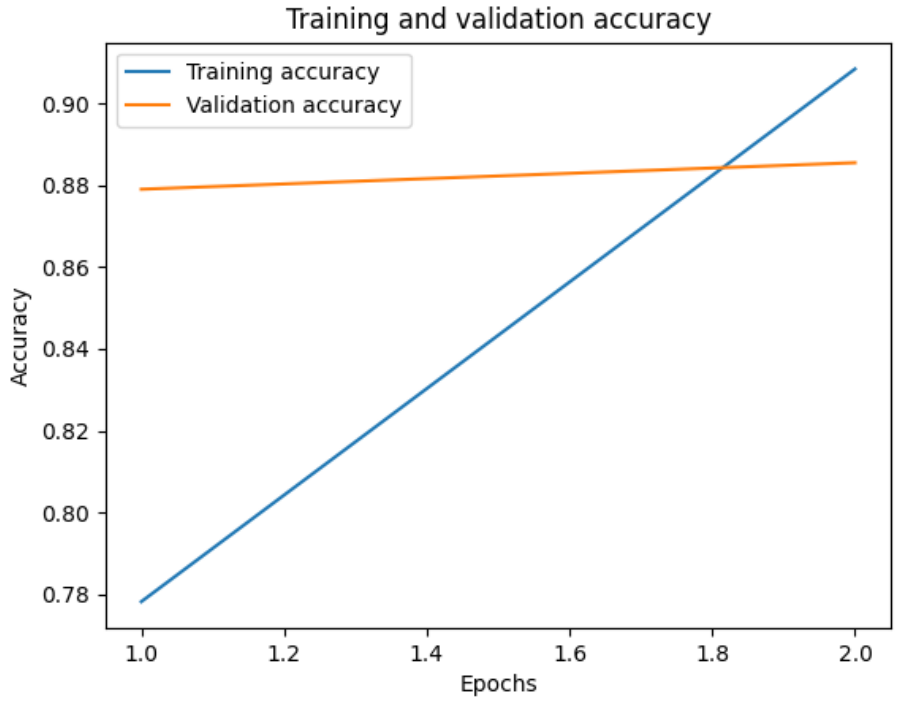

Рисунок 3 – График точности рекуррентной сверточной сети

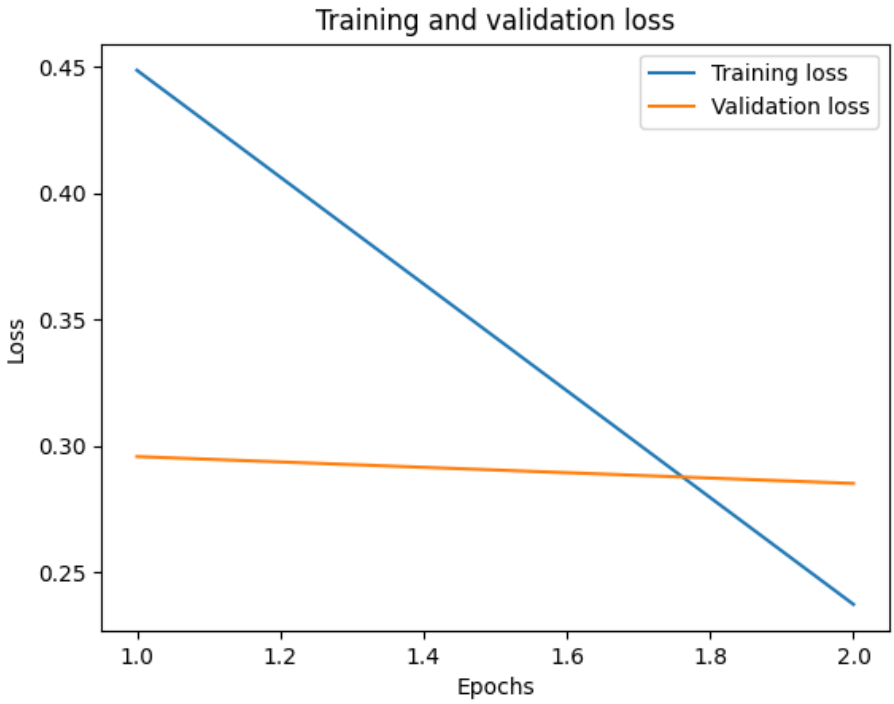

Рисунок 4 – График ошибки рекуррентной сверточной сети

В результате обучения были достигнуты точности 86.34% и 88.55% соответственно.

Далее была написана функция ensemble() для осуществления ансамблирование моделей по среднему арифметическому. Благодаря этому была достигнута точность 95%.

Для загрузки текстов была написана функция load_text(filename). Для классификации были протестированы следующие отзывы:

1) Very good movie, I got unforgettable emotions!
2) A normal movie, nothing special, but not bad either.
3) I want to forget this movie, it's terrible. I won’t watch it again.

Были получены следующие результаты: 1 – 0.7243; 2 – 0.3319; 3 – 0.2211, это может означать положительный, нейтральный и отрицательный отзывы, что соответствует содержанию текстов.

Вывод

В процессе выполнения лабораторной работы был разработан ансамбль рекуррентной и рекуррентной сверточной сетей. Были изучены методы классификации текстов.

Код программы представлен в приложении А.

ПРИЛОЖЕНИЕ А

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

(X_train, Y_train), (X_test, Y_test) = imdb.load_data(num_words=10000)
data = np.concatenate((X_train, Y_test), axis=0)
targets = np.concatenate((Y_train, Y_test), axis=0)
max_review_length = 500
top_words = 10000
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
embedding_vector_length = 32


def build_models():
    models = []
    model_1 = Sequential()
    model_1.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model_1.add(LSTM(100))
    model_1.add(Dropout(0.4))
    model_1.add(Dense(64, activation='relu'))
    model_1.add(Dropout(0.3))
    model_1.add(Dense(1, activation='sigmoid'))
    models.append(model_1)

    model_2 = Sequential()
    model_2.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model_2.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
    model_2.add(MaxPooling1D(pool_size=2))
    model_2.add(Dropout(0.3))
    model_2.add(LSTM(100))
    model_2.add(Dropout(0.3))
    model_2.add(Dense(1, activation='sigmoid'))
    models.append(model_2)
    return models


def fit_models(models):
    i = 1
    for model in models:
```

```
        model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
        history = model.fit(X_train, Y_train, validation_data=(X_test,
Y_test), epochs=2, batch_size=64)
        scores = model.evaluate(X_test, Y_test, verbose=0)
        model.save('model' + str(i) + '.h5')
        print("Accuracy: %.2f%%" % (scores[1] * 100))
        epochs = range(1, len(history.history['loss']) + 1)
        plt.plot(epochs, history.history['loss'], label='Training loss')
        plt.plot(epochs, history.history['val_loss'], label='Validation
loss')
        plt.title('Training and validation loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()
        plt.clf()

        plt.plot(epochs, history.history['accuracy'], label='Training
accuracy')
        plt.plot(epochs, history.history['val_accuracy'], label='Validation
accuracy')
        plt.title('Training and validation accuracy')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.show()
        i += 1


def ensemble():
    model1 = load_model("model1.h5")
    model2 = load_model("model2.h5")
    predictions1 = model1.predict(X_train)
    predictions2 = model2.predict(X_train)
    predictions = np.divide(np.add(predictions1, predictions2), 2)
    targets = np.reshape(Y_train, (25000, 1))
    predictions = np.greater_equal(predictions, np.array([0.5]))
    predictions = np.logical_not(np.logical_xor(predictions, targets))
    acc = predictions.mean()
    print("Accuracy of ensemble  %s" % acc)


def load_text(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    words = text.split()
    import string
```

```
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in words]
    stripped_low = []
    for w in stripped:
        stripped_low.append(w.lower())
    print(stripped_low)
    indexes = imdb.get_word_index()
    encoded = []
    for w in stripped_low:
        if w in indexes and indexes[w] < 10000:
            encoded.append(indexes[w])
    data = np.array(encoded)
    test = sequence.pad_sequences([data], maxlen=max_review_length)
    model1 = load_model("model1.h5")
    model2 = load_model("model2.h5")
    results = []
    results.append(model1.predict(test))
    results.append(model2.predict(test))
    print(results)
    result = np.array(results).mean(axis=0)
    print(result)


models = build_models()
fit_models(models)
ensemble()
load_text("good.txt")
#load_text("normal.txt")
#load_text("bad.txt")
```