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 _____ Гиззатов А.С.

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% Требования к выполнению задания.

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.

Датасет IMDb состоит из 50 000 обзоров фильмов от пользователей, помеченных как положительные (1) и отрицательные (0). Это пример бинарной или двуклассовой классификации, важный и широко применяющийся тип задач машинного обучения. 1. Рецензии предварительно обрабатываются, и каждая из них кодируется последовательностью индексов слов в виде целых чисел. 2. Слова в обзорах индексируются по их общей

частоте появления в датасете. Например, целое число «2» кодирует второе наиболее частое используемое слово. 3. 50 000 обзоров разделены на два набора: 25 000 для обучения и 25 000 для тестирования.

Ход работы.

1.Были построены и обучены нейронные сети для обработки. Код предоставлен в приложение А. С архитектурами:

1)

```
model = Sequential()
model.add(Embedding(10000, EMBEDING_VECOR_LENGTH, input_length=MAX_REVIEW_LENGTH))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(Dropout(0.3))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))
model.add(LSTM(100))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
H = model.fit(x_train,
          y_train,
          batch_size=64,
          epochs=2,
          verbose=1,
          validation_split=0.1
          )
```

2)

```
model1 = Sequential()
model1.add(Embedding(10000, EMBEDING_VECOR_LENGTH, input_length=MAX_REVIEW_LENGTH))
model1.add(LSTM(100))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model1.summary())
H1 = model1.fit(x_train,
          y_train,
          batch_size=64,
          epochs=2,
          verbose=1,
          validation_split=0.1
          )
```

Были взяты 2 модели и обучены отдельно, а потом результаты были усреднены и сравнены с результатами полученными по отдельности.

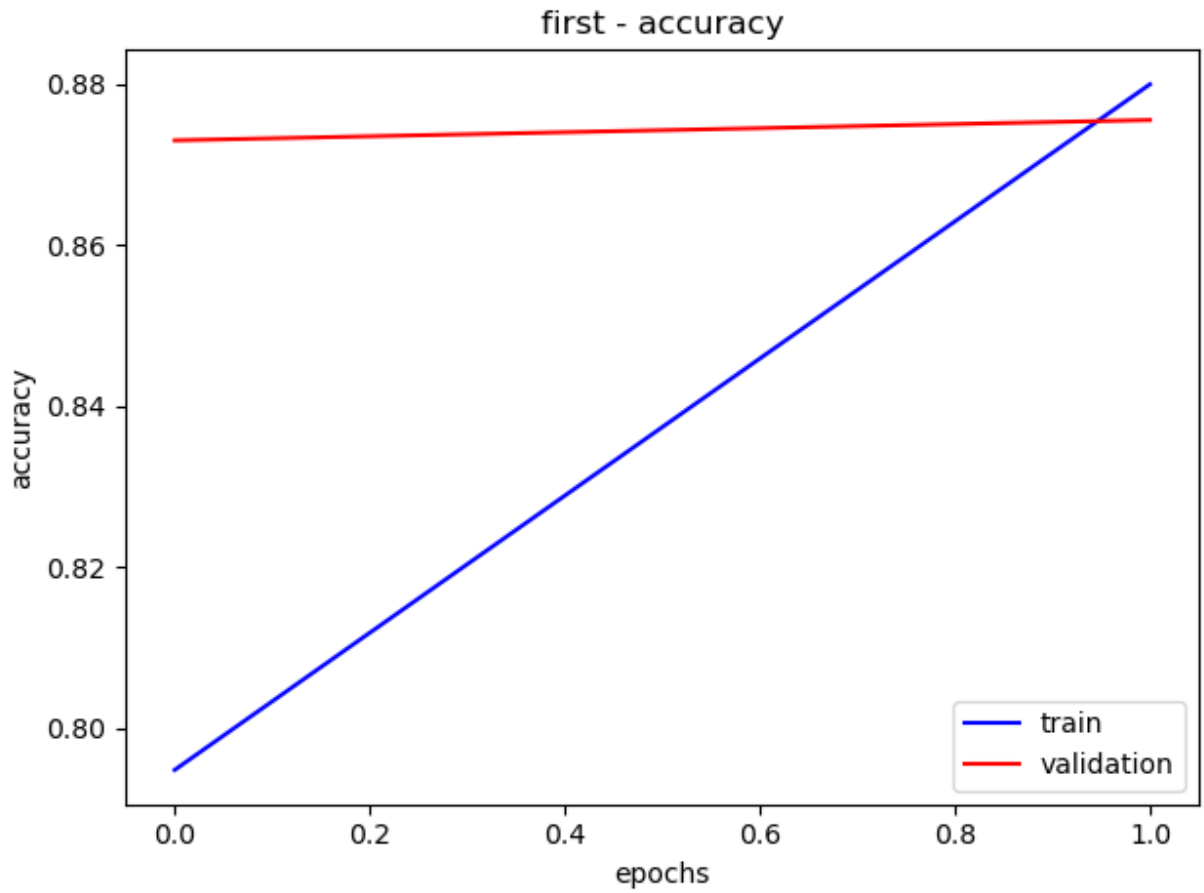

Рис.1 – точность полученная рекуррентной сетью.

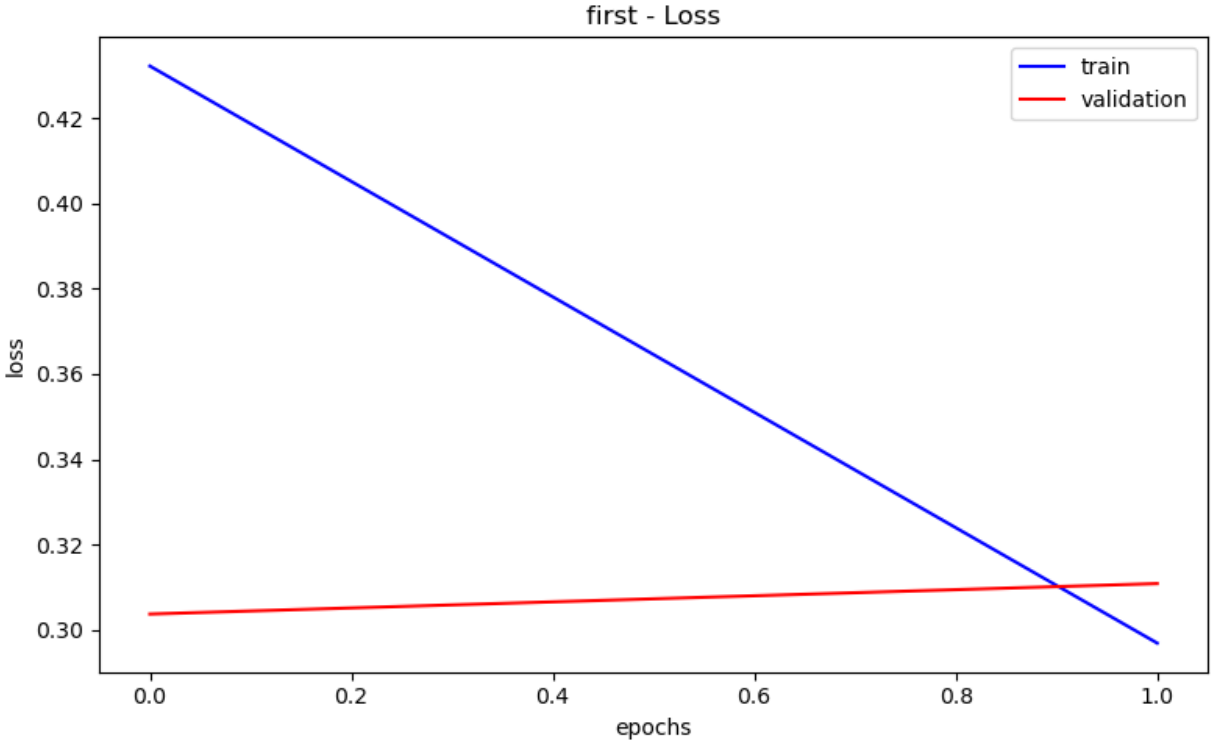

Рис.2 – потери полученные рекуррентной сетью.

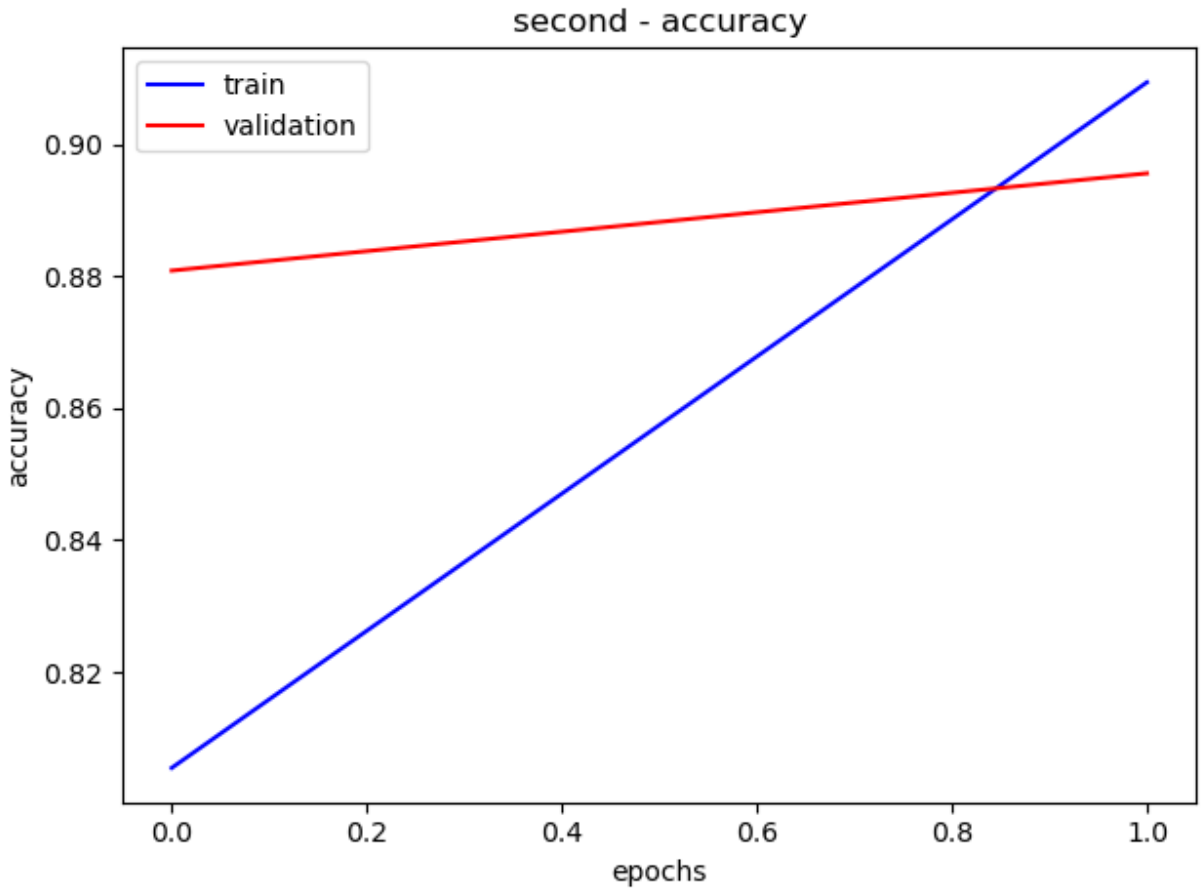

Рис.3 - точность полученная сверточной сетью.

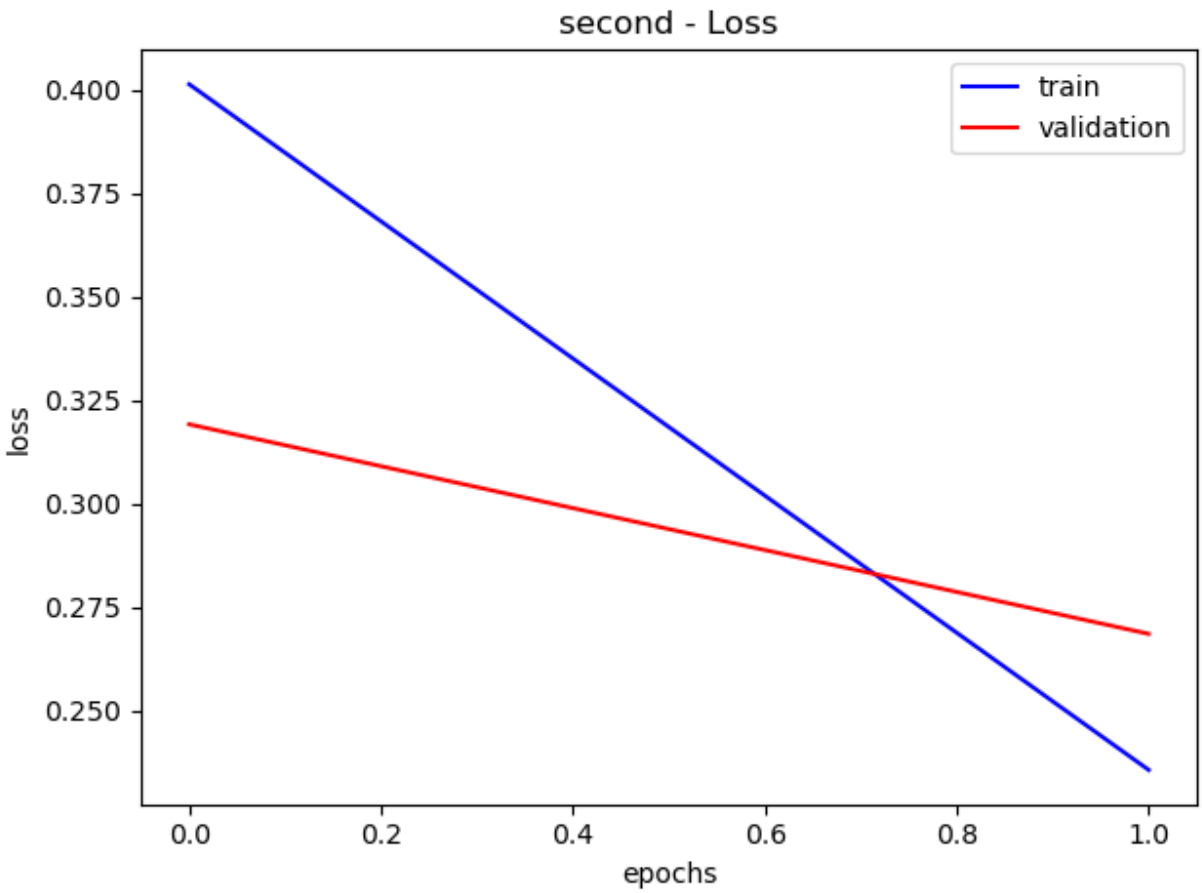

Рис.4 - потери полученная сверточной сетью.

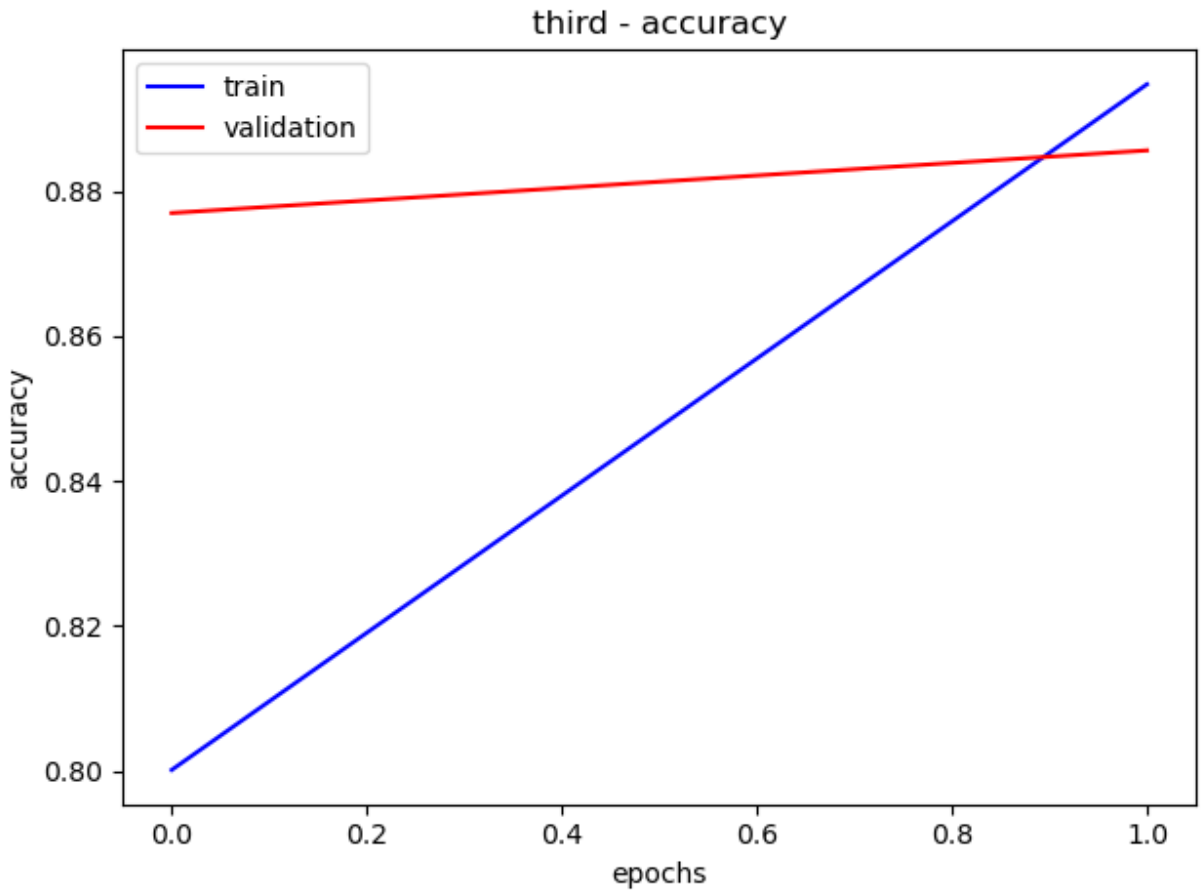

Рис.5 – усредненные значения точности значения 2 нейронных сетей

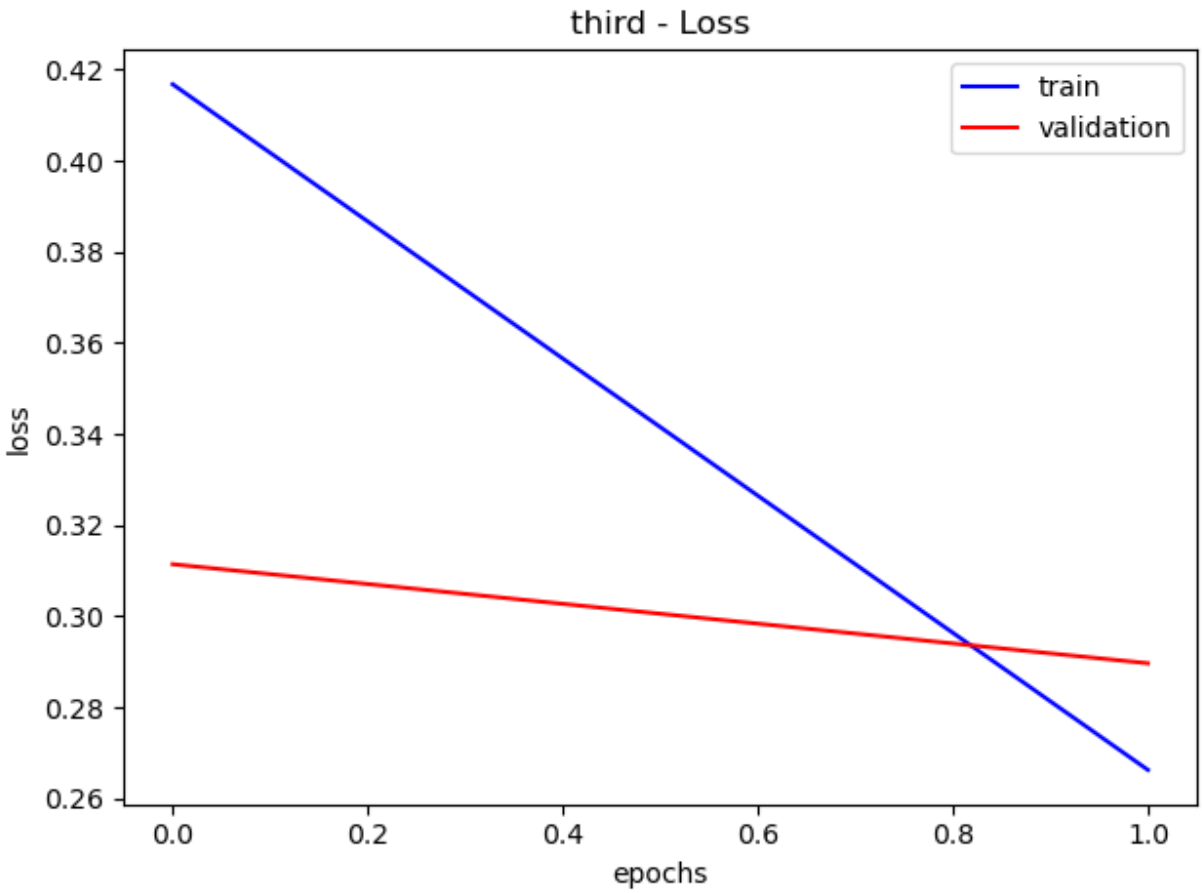

Рис.6 – усредненные значения потерь значения 2 нейронных сетей

Сравнив все рисунки можно сделать вывод, что при результатов точности мы получим более устойчивую к изменениям модель.

2. Напишем функцию для ввода пользовательского обзора.

```
def index(str, indx):
    lst = str.split()
    for i, w in enumerate(lst):
        lst[i] = indx.get(w)
    return lst


def user_text(str, indx):
    for i in range(len(str)):
        str[i] = index(str[i], indx)
    return str
```

и создадим массив обзоров

```
user_txt = [
    "The movie affects you in a way that makes it physically painful to
experience, but in a good way",
    "Seriously, unbelievable",
    "One of the best movies in these few years",
    "It is very boring film",
    "it's very good",
    "it's very boring",
    "fantastic film, wonderful casting, good job, creators",
```

```
    "beautiful picture, good scenario, it's amazing",
    "Absolute disappointment. I believed the hype like a fool"
]
user_y = [1., 1., 1., 0., 1., 0., 1., 1., 0.]
```

При помощи данной функции можно получить из массива обзоров массив представлений в виде индексов слов в imdb датасете и подготовленные для прогона через модель. График точности оценки фильма, при прогоне через написанный датасет из 9 обзоров предоставлена на рис. 7,8.

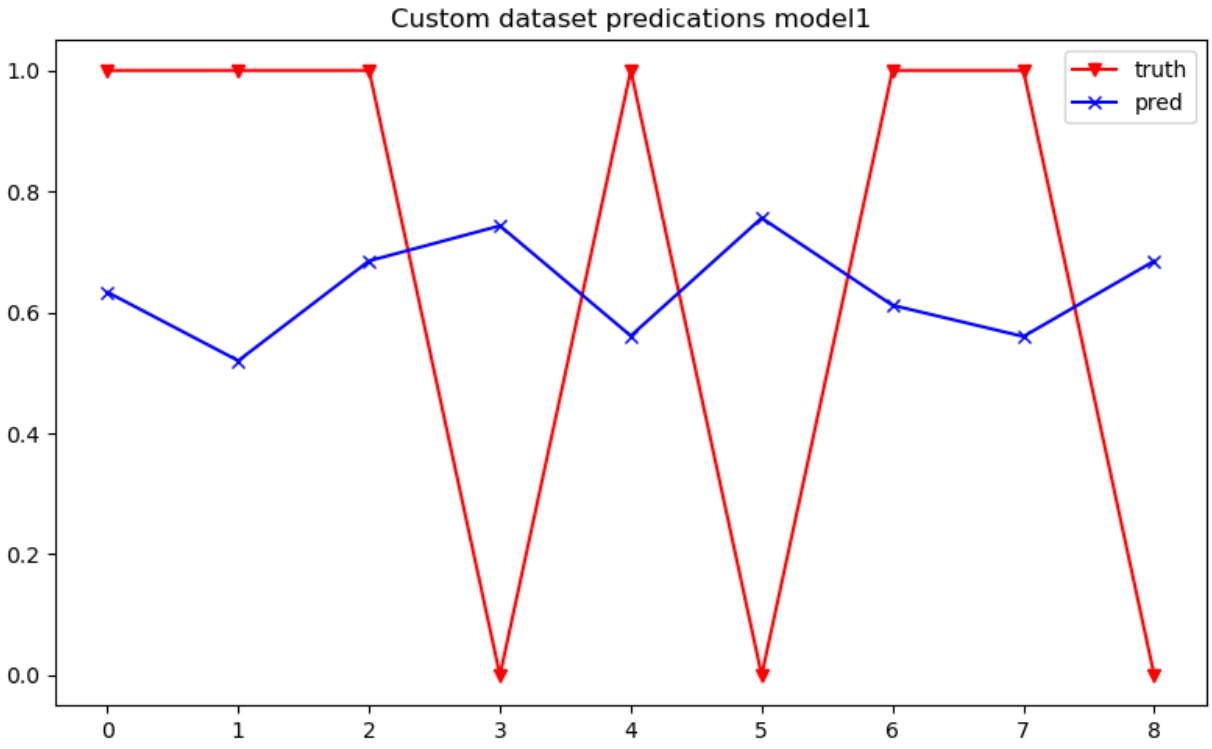

Рис.7- предсказание сверточной нейронной сети

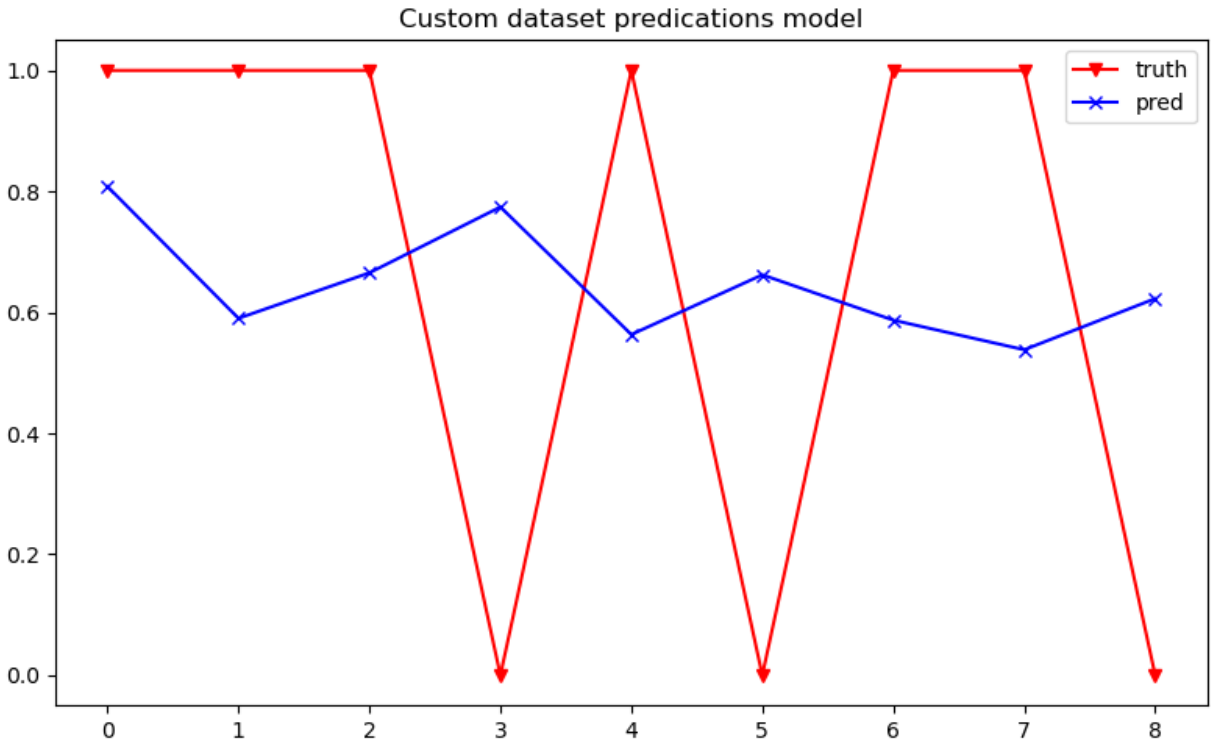

Рис.8 – предсказание рекуррентной нейрнной сети

Выводы.

В ходе работы была изучена задача классификация обзоров из датасета IMDB. Подобрана архитектура, дающая точность 91,1%. Проведя исследование, было выяснено, что при добавлении нескольких слоев свертки и пулинга увеличивается точность предсказания. Функция для подготовки вручную введенных обзоров, продемонстрировала точность в ~60%

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД LAB7.PY

```
import numpy as np
from keras import Sequential
from keras.datasets import imdb
import matplotlib.pyplot as plt
from keras.layers import Embedding, Conv1D, Dropout, MaxPooling1D, LSTM, Dense
from keras_preprocessing import sequence
from sklearn.model_selection import train_test_split

(training_data, training_targets), (testing_data, testing_targets) =
imdb.load_data(num_words=500)
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets), axis=0)
index = imdb.get_word_index()
reverse_index = dict([(value, key) for (key, value) in index.items()])
decoded = " ".join([reverse_index.get(i - 3, "#") for i in data[0]])
print(decoded)
MAX_REVIEW_LENGTH = 500
EMBEDING_VECOR_LENGTH = 32


def plot_loss(loss, v_loss,model):
    plt.figure(1, figsize=(8, 5))
    plt.plot(loss, 'b', label='train')
    plt.plot(v_loss, 'r', label='validation')
    plt.title(model + " - " + 'Loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.legend()
    plt.show()
    plt.clf()


def plot_acc(acc, val_acc,model):
    plt.plot(acc, 'b', label='train')
    plt.plot(val_acc, 'r', label='validation')
    plt.title(model + " - " + 'accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epochs')
    plt.legend()
    plt.show()
    plt.clf()


def vectorize(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results

custom_x = [
    "The movie affects you in a way that makes it physically painful to experience,
but in a good way",
    "Seriously, unbelievable",
    "One of the best movies in these few years",
    "It is very boring film",
    "it's very good",
    "it's very boring",
    "fantastic film, wonderful casting, good job, creators",
```

```
    "beautiful picture, good scenario, it's amazing",
    "Absolute disappointment. I believed the hype like a fool"
]
custom_y = [1., 1., 1., 0., 1., 0., 1., 1., 0.]


def index(str, indx):
    lst = str.split()
    for i, w in enumerate(lst):
        lst[i] = indx.get(w)
    return lst


def user_text(str,indx):
    for i in range(len(str)):
        str[i] = index(str[i],indx)
    return str


print('Before: {}'.format(custom_x))
custom_x = user_text(custom_x, imdb.get_word_index())
print('After: {}'.format(custom_x))
for index_j, i in enumerate(custom_x):
    for index, value in enumerate(i):
        if value is None:
            custom_x[index_j][index] = 0
print('After after: {}'.format(custom_x))

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

x_train = sequence.pad_sequences(x_train, maxlen=MAX_REVIEW_LENGTH)
x_test = sequence.pad_sequences(x_test, maxlen=MAX_REVIEW_LENGTH)
custom_x = sequence.pad_sequences(custom_x, maxlen=MAX_REVIEW_LENGTH)

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)

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


model1 = Sequential()
model1.add(Embedding(10000, EMBEDING_VECOR_LENGTH, input_length=MAX_REVIEW_LENGTH))
model1.add(Dropout(0.3))
model1.add(LSTM(100))
model1.add(Dropout(0.5))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model1.summary())
H1 = model1.fit(x_train,
          y_train,
          batch_size=64,
          epochs=2,
          verbose=1,
          validation_split=0.1
          )
scores = model1.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
plot_loss(H1.history['loss'], H1.history['val_loss'],'first')
```

```
plot_acc(H1.history['accuracy'], H1.history['val_accuracy'],'first')


model = Sequential()
model.add(Embedding(10000, EMBEDING_VECOR_LENGTH, input_length=MAX_REVIEW_LENGTH))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(Dropout(0.3))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))
model.add(LSTM(100))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
H = model.fit(x_train,
          y_train,
          batch_size=64,
          epochs=2,
          verbose=1,
          validation_split=0.1
          )

_, acc = model.evaluate(x_test, y_test)
print('Test', acc)

res_acc = (np.array(H.history['accuracy'])+np.array(H1.history['accuracy']))/2
res_los = (np.array(H.history['loss'])+np.array(H1.history['loss']))/2
res_val_acc =
(np.array(H.history['val_accuracy'])+np.array(H1.history['val_accuracy']))/2
res_val_los = (np.array(H.history['val_loss'])+np.array(H1.history['val_loss']))/2

plot_acc(res_acc, res_val_acc, 'third')
plot_loss(res_los, res_val_los, 'third')

print(custom_x, custom_y)
plot_loss(H.history['loss'], H.history['val_loss'],'second')
plot_acc(H.history['accuracy'], H.history['val_accuracy'],'second')
custom_loss, custom_acc = model.evaluate(custom_x, custom_y)
print('custom_acc:', custom_acc)
preds = model.predict(custom_x)
print(preds)
plt.figure(3, figsize=(8, 5))
plt.title("Custom dataset predications model")
plt.plot(custom_y, 'r', marker='v', label='truth')
plt.plot(preds, 'b', marker='x', label='pred')
plt.legend()
plt.show()
plt.clf()

preds = model1.predict(custom_x)
print(preds)
plt.figure(3, figsize=(8, 5))
plt.title("Custom dataset predications model1")
plt.plot(custom_y, 'r', marker='v', label='truth')
plt.plot(preds, 'b', marker='x', label='pred')
plt.legend()
plt.show()
plt.clf()
```