

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

**ОТЧЕТ**  
**по лабораторной работе №6**  
**по дисциплине «Искусственные нейронные сети»**  
**Тема: Прогноз успеха фильмов по обзорам**

Студент гр. 7382

\_\_\_\_\_

Глазунов С.А.

Преподаватель

\_\_\_\_\_

Жукова Н.А.

Санкт-Петербург

2020

### **Цель работы.**

Прогноз успеха фильмов по обзорам (Predict Sentiment From Movie Reviews).

### **Порядок выполнения работы.**

1. Ознакомиться с задачей регрессии.
2. Изучить способы представления текста для передачи в ИНС.
3. Достигнуть точность прогноза не менее 95%.

### **Требования к выполнению задания.**

1. Построить и обучить нейронную сеть для обработки текста.
2. Исследовать результаты при различном размере вектора представления текста.
3. Написать функцию, которая позволяет ввести пользовательский текст (в отчете привести пример работы сети на пользовательском тексте).

### **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.

- 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. Была построена и обучена нейронная сеть для обработки текста.

Код предоставлен в приложении А, Б, В, Г.

Архитектура:

- Оптимизатор adam, скорость обучения = 0.001.
- Epochs = 2, batch\_size = 500, loss = binary\_crossentropy
- Max кол. слов в обзоре 500, max. размер словаря слов 10000.
- Модель:

```
model = Sequential()
model.add(Dense(50, activation="relu", input_shape=(10000,)))
model.add(Dropout(0.2, noise_shape=None, seed=None))
model.add(Dense(50, activation="linear", kernel_regularizer=regularizers.l2()))
model.add(Dropout(0.5, noise_shape=None, seed=None))
model.add(Dense(100, activation="relu", kernel_regularizer=regularizers.l2()))
model.add(Dropout(0.5, noise_shape=None, seed=None))
model.add(Dense(50, activation="relu"))
model.add(Dense(1, activation="sigmoid"))
model.compile(Adam(), loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(
    train_x,
    train_y,
    batch_size=500,
    epochs=2,
    verbose=1,
    validation_data=(test_x, test_y)
)
```

Данная архитектура дает точность: на тренировочных ~ 89,1%, на валидационных ~ 89%. Графики точности и ошибки предоставлены на рис. 1 и рис. 2 соответственно.

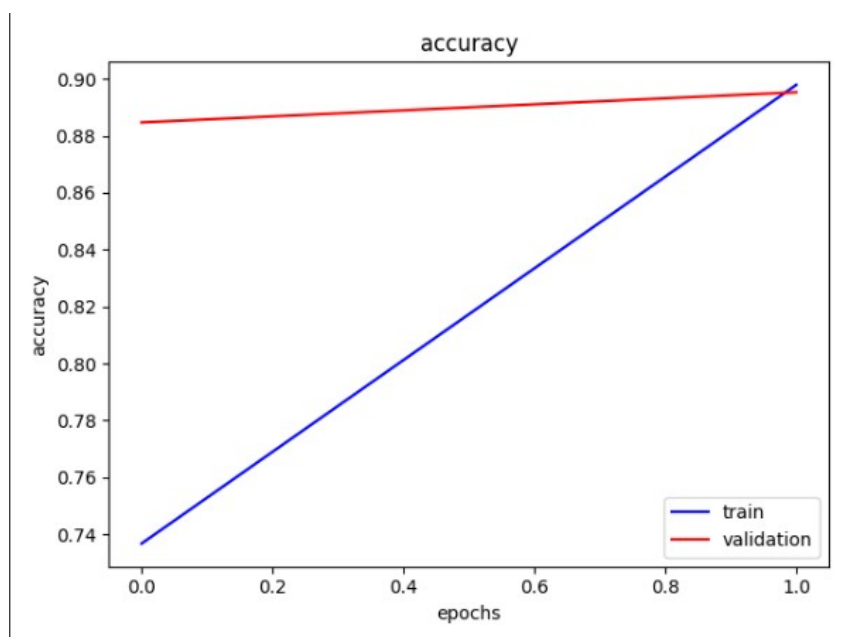


Рисунок 1 – График точности при размере словаря 10 тыс. обзоров

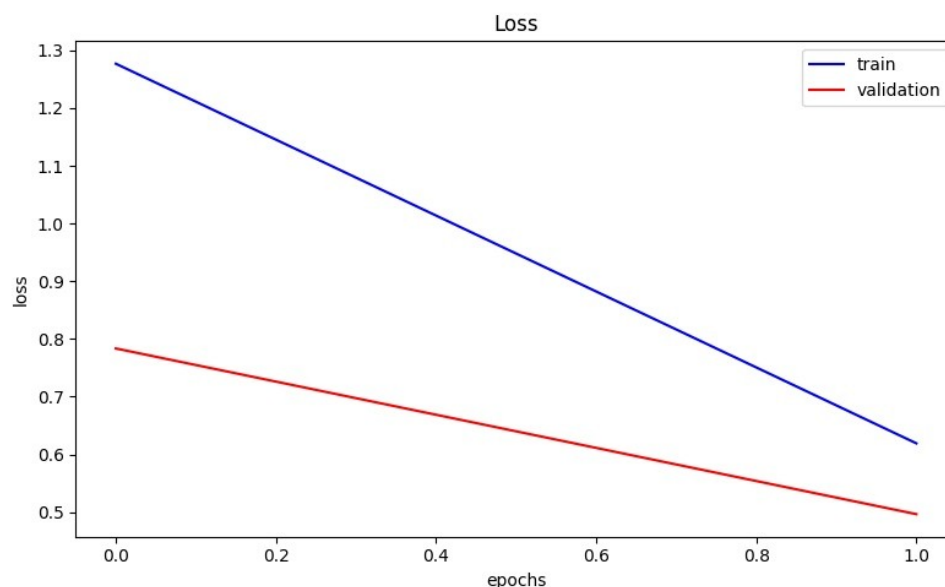


Рисунок 2 – График потерь при размере словаря 10 тыс. обзоров

2. Исследуем результаты при различном размере вектора представления текста.

При изменении максимального размера словаря с 10 тыс. до 1 тыс.. Точность: на тренировочных упала ~ 86,8%, на валидационных также уменьшилась ~ 86%. Графики точности и ошибки предоставлены на рис. 3 и рис. 4 соответственно.

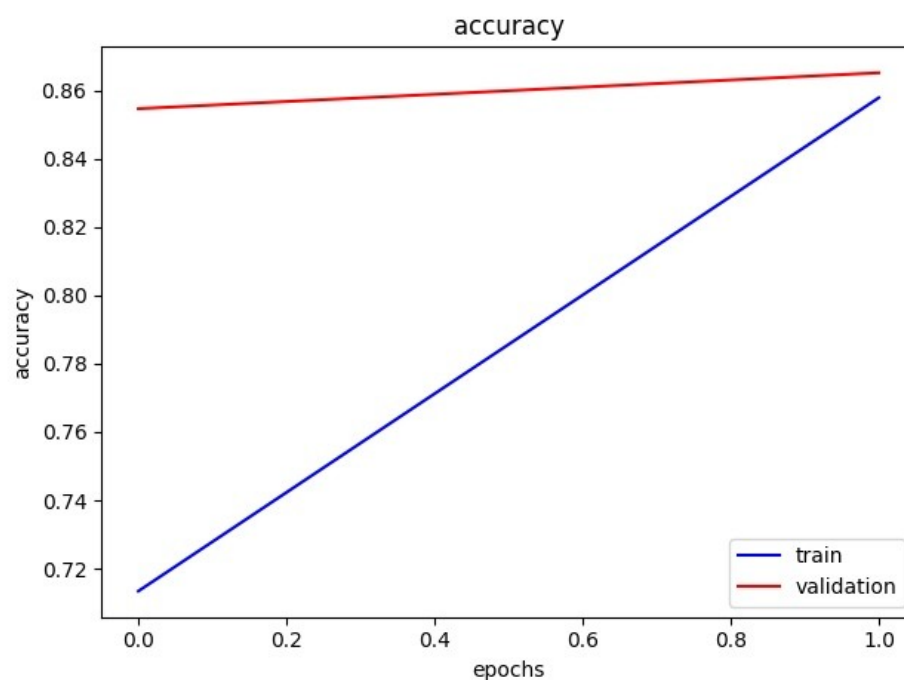


Рисунок 3 – График точности при размере словаря 1 тыс. обзоров

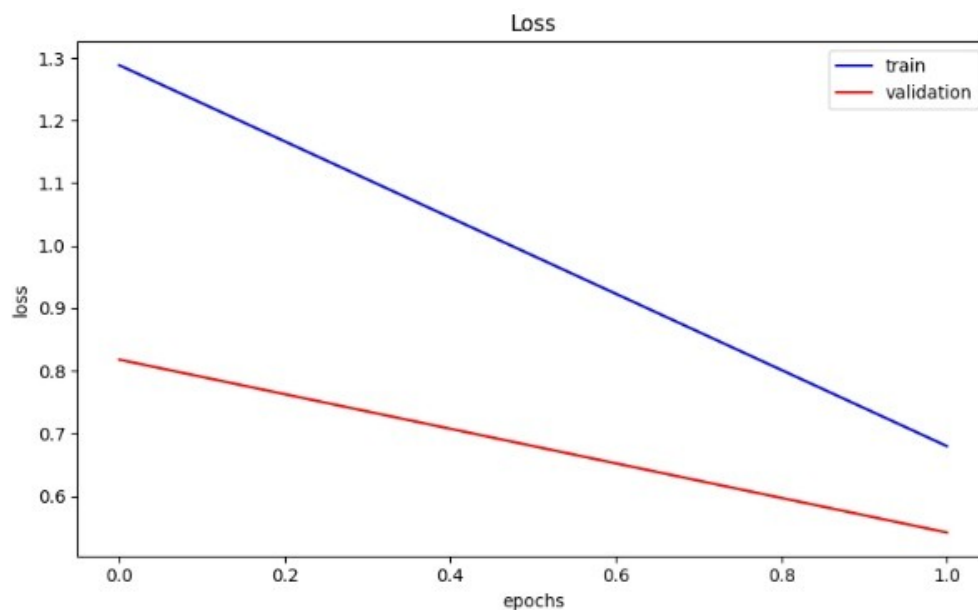


Рисунок 4 – График потерь при размере словаря 1 тыс. обзоров

При изменении максимального размера словаря до 500 тыс.. Точность: на тренировочных упала  $\sim 83,3\%$ , на валидационных также уменьшилась до  $\sim 82,2\%$ . Графики точности и ошибки предоставлены на рис. 5 и рис. 6 соответственно.

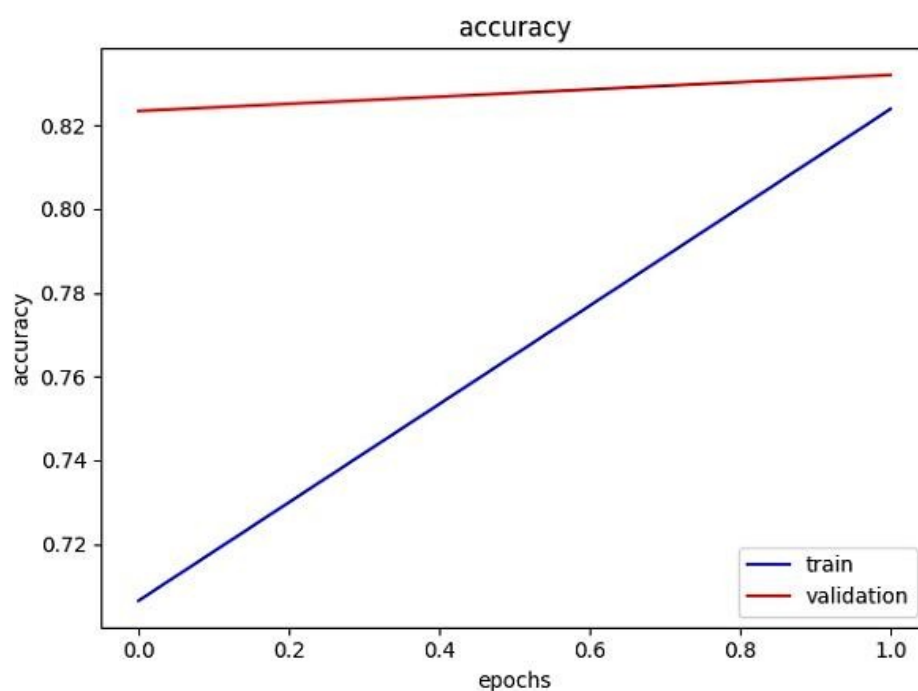


Рисунок 5 – График точности при размере словаря 500 тыс. обзоров

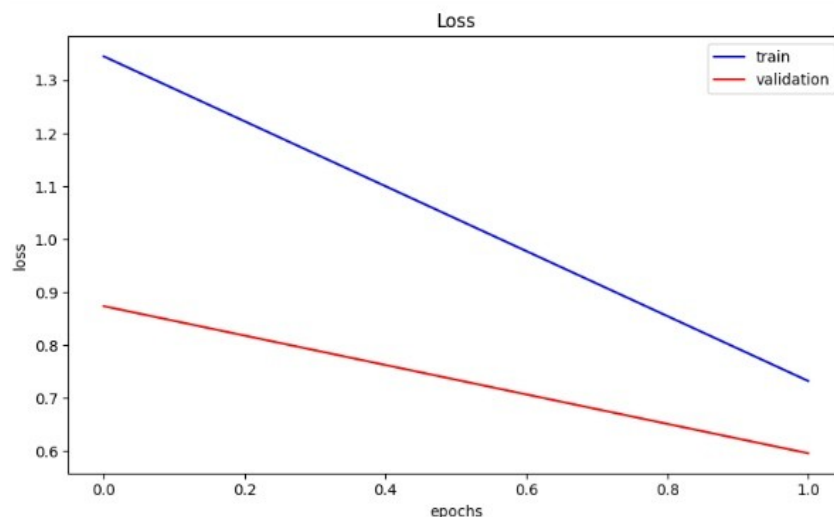


Рисунок 6 – График потерь при размере словаря 500 тыс. обзоров

Из графиков на рис. 1 - 5, можно сделать вывод, что точность падает с уменьшением размера словаря, так как мы убираем часть слов из обзоров. Из-за этого оценка некоторых обзоров меняется, поэтому сеть не может их точно классифицировать.

3. Напишем функцию, которая позволяет ввести пользовательский текст.

```
def gen_custom_x(custom_x, word_index):
    def get_index(a, index):
        new_list = a.split()
        for i, v in enumerate(new_list):
            new_list[i] = index.get(v)
        return new_list
    for i in range(len(custom_x)):
        custom_x[i] = get_index(custom_x[i], word_index)
    return custom_x
```

При помощи данной функции можно получить из массива строк (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 5 обзоров (см. рис. 7), предоставлена на рис. 8.

```

custom_x = [
    "It is very interesting film",
    "it's fantastic",
    "it's good",
    "very very bad",
    "it was boring"
]
custom_y = [1., 1., 1., 0., 0.]

```

Рисунок 7 – Пользовательский текст

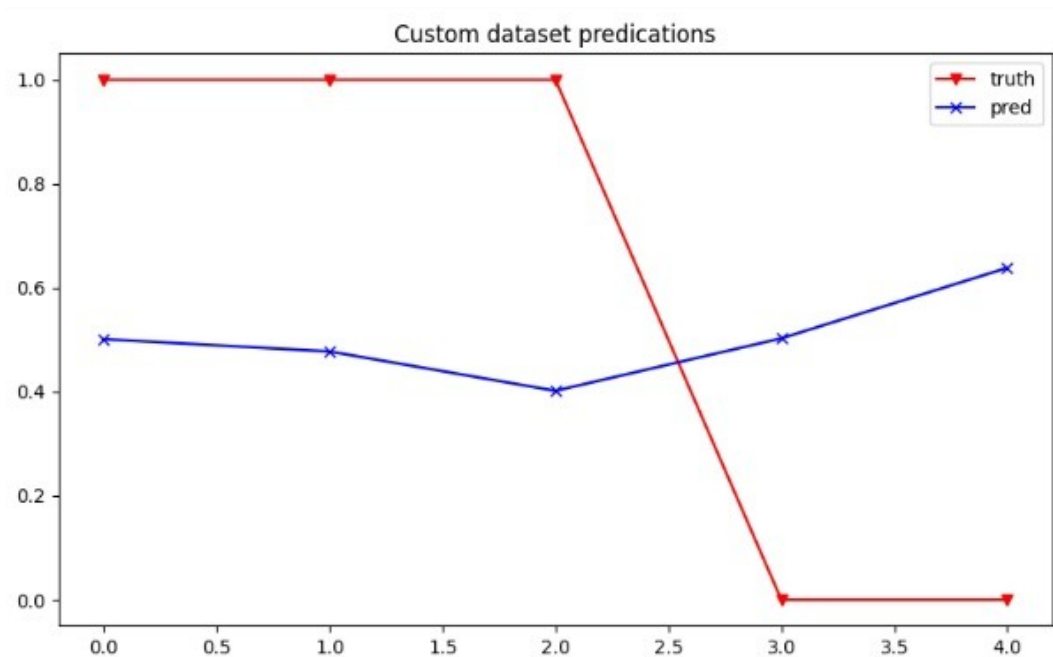


Рисунок 8 – График точности оценки фильма

Из графика на рис. 8 видно, что точность оценки фильма  $\sim 62\%$ , т.е. 3/5.

### Выводы.

В ходе работы была изучена задача классификация обзоров из датасета IMDB. Подобрана архитектура, дающая точность 89,1%. Проведя исследование, было выяснено, что при уменьшении максимального размера словаря было точность уменьшается, что логично, так как мы убираем часть «словарного запаса». Функция для подготовки вручную введенных обзоров, продемонстрировала точность в  $\sim 62\%$ .

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД LAB6.PY

```
import numpy as np
from keras.datasets import imdb
from setuputils.dist import sequence
import matplotlib.pyplot as plt
from model import Net

(X_train, y_train), (X_test, y_test) = imdb.load_data()

(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)

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 = [
    "It is very interesting film",
    "it's fantastic",
    "it's good",
    "very very bad",
    "it was boring"
]
custom_y = [1., 1., 1., 0., 0.]
```



```

def gen_custom_x(custom_x, word_index):
    def get_index(a, index):
        new_list = a.split()
        for i, v in enumerate(new_list):
            new_list[i] = index.get(v)
        return new_list
    for i in range(len(custom_x)):
        custom_x[i] = get_index(custom_x[i], word_index)
    return custom_x

print('Before: {}'.format(custom_x))
custom_x = gen_custom_x(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))

data = vectorize(data)
targets = np.array(targets).astype("float32")
custom_y = np.asarray(custom_y).astype("float32")

test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]

net = Net()
net.build_net()
net.compile()
net.fit(train_x, train_y, test_x, test_y)

_, acc = net.evaluate(test_x, test_y)
print('Test', acc)
net.demonstration()

```

```
custom_x = vectorize(custom_x)
print(custom_x, custom_y)

custom_loss, custom_acc = net.evaluate(custom_x, custom_y)
print('custom_acc:', custom_acc)
preds = net.model.predict(custom_x)
print(preds)
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()
```

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД MODEL.PY

```
from keras import Sequential, regularizers
from keras.layers import Dense, Dropout, Embedding, GRU
from keras.losses import BinaryCrossentropy
from keras.optimizers import Adam

from config import *
from plot import plot_loss, plot_acc

class Net:
    def __init__(self):
        self.model = None
        self.history = None

    def build_net(self):
        self.model = Sequential()
        self.model.add(Dense(50, activation="relu",
input_shape=(10000,)))
        self.model.add(Dropout(0.2, noise_shape=None,
seed=None))
        self.model.add(Dense(50, activation="linear",
kernel_regularizer=regularizers.l2()))
        self.model.add(Dropout(0.5, noise_shape=None,
seed=None))
        self.model.add(Dense(100, activation="relu",
kernel_regularizer=regularizers.l2()))
        self.model.add(Dropout(0.5, noise_shape=None,
seed=None))
        self.model.add(Dense(50, activation="relu"))
        self.model.add(Dense(1, activation="sigmoid"))
        self.model.summary()

    def compile(self):
        self.model.compile(Adam(lr=ADAM_LR),
loss='binary_crossentropy', metrics=['accuracy'])

    def fit(self, x_train, y_train, x_test, y_test):
```

```

self.history = self.model.fit(
    x_train,
    y_train,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    verbose=1,
    #validation_split=0.1
    validation_data=(x_test, y_test)
)

def evaluate(self, x, y):
    return self.model.evaluate(x, y)

def demonstration(self):
    H = self.history
    plot_loss(H.history['loss'], H.history['val_loss'])
    plot_acc(H.history['accuracy'],
H.history['val_accuracy'])

```

**ПРИЛОЖЕНИЕ В**  
**ИСХОДНЫЙ КОД PLOT.PY**

```
import matplotlib.pyplot as plt

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

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

**ПРИЛОЖЕНИЕ Г**  
**ИСХОДНЫЙ КОД CONFIG.PY**

```
BATCH_SIZE = 50  
EPOCHS = 2  
ADAM_LR = 0.0001
```