

**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. 8383

\_\_\_\_\_

Федоров И.А.

Преподаватель

\_\_\_\_\_

Жангиров Т.Р.

Санкт-Петербург

2021

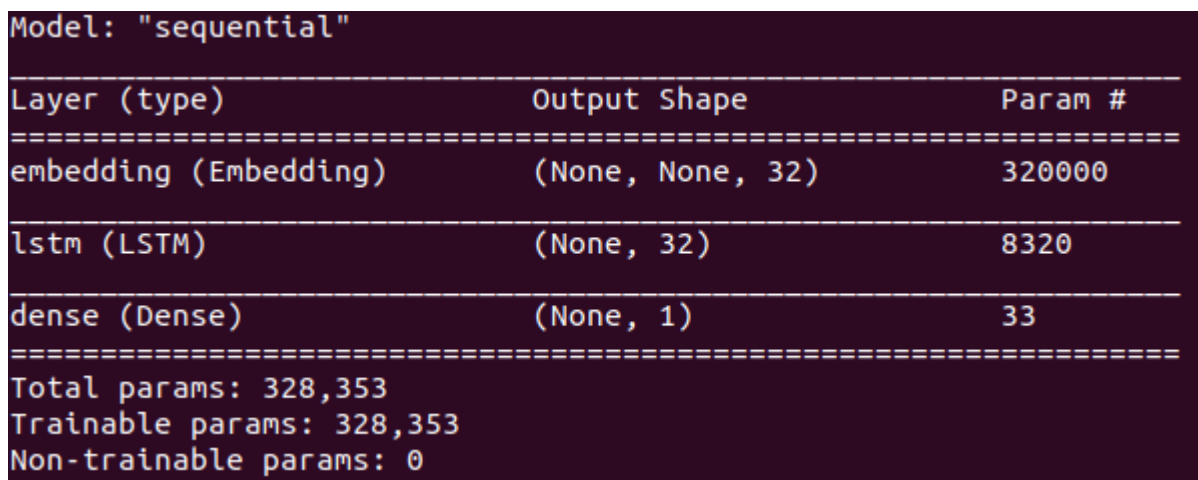
## Цель работы.

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 работе будет использоваться датасет IMDb, обучение будет проводиться с помощью рекуррентной нейронной сети.

## Выполнение работы.

Были импортированы необходимые для работы классы, модули, функции, а также данные *imdb*.

Для ознакомления с ансамблированием моделей, были реализованы четыре различные архитектуры нейронных сетей. Описание моделей показаны ниже на рис. 1-4.



```
Model: "sequential"
Layer (type)                Output Shape              Param #
=====
embedding (Embedding)       (None, None, 32)         320000
lstm (LSTM)                  (None, 32)                8320
dense (Dense)                (None, 1)                 33
=====
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
```

Рисунок 1 – Модель 1

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 8)	80000
dropout (Dropout)	(None, 500, 8)	0
flatten (Flatten)	(None, 4000)	0
dense_1 (Dense)	(None, 1)	4001

=====  
Total params: 84,001  
Trainable params: 84,001  
Non-trainable params: 0  
=====

Рисунок 2 – Модель 2

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 500, 32)	320000
conv1d (Conv1D)	(None, 500, 32)	4128
max_pooling1d (MaxPooling1D)	(None, 250, 32)	0
lstm_1 (LSTM)	(None, 100)	53200
dense_2 (Dense)	(None, 1)	101

=====  
Total params: 377,429  
Trainable params: 377,429  
Non-trainable params: 0  
=====

Рисунок 3 – Модель 3

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 500, 8)	80000
conv1d_1 (Conv1D)	(None, 496, 100)	4100
max_pooling1d_1 (MaxPooling1D)	(None, 248, 100)	0
conv1d_2 (Conv1D)	(None, 244, 100)	50100
max_pooling1d_2 (MaxPooling1D)	(None, 122, 100)	0
flatten_1 (Flatten)	(None, 12200)	0
dense_3 (Dense)	(None, 1)	12201

Total params: 146,401  
 Trainable params: 146,401  
 Non-trainable params: 0

Рисунок 4 – Модель 4

Точность 1-й модели: 86.61%, 2-й модели: 86.84%, 3-й модели: 84.40%, 4-й модели: 87.52%, точность ансамбля: 89.11%.

Были написаны функции, позволяющие ввести пользовательский текст. Код представлен ниже. Функция `input_user_comment()` считывает строку, которую вводит пользователь и удаляет из нее лишние символы. Функция `vectorize_user_comment()` приводит к виду, удобному для ИНС.

```
def clean_str(str):
    str = ''.join(c for c in str if c.isalpha() or
c.isspace()).expandtabs(tabsize=1).strip().lower()
    str = ' '.join(str.split())
    return str

def input_user_comment():
    print('Write your comment: ')
    result_str = ''
    while True:
        str = input()
        if str:
            result_str += str + " "
        else:
            break

    res_vect = clean_str(result_str).split(' ')
    return res_vect
```

```

def vectorize_comment(text, vector_len=500):
    words_index = imdb.get_word_index()
    vectorized = []
    for word in text:
        ind = words_index.get(word)
        if ind is not None and ind < vector_len:
            vectorized.append(ind + 3)

    res = []
    res.append(vectorized)
    return sequence.pad_sequences(res, maxlen=vector_len)

```

Ниже показан пример использования:

```

#Ensembling accuracy: 88.96%
Write your comment:
The 2014 movie Fury with Brad Pitt was dumb. Nothing that happened was believable, the characters were flat cartoon characters,
the story was predictable and nothing was historically accurate.

[[0.1610823]]
It's a negative comment.

```

## **Выводы.**

В ходе выполнения данной работы была реализована рекуррентная ИНС для прогнозирования успеха фильма по обзорам. Была реализована функция, для ввода пользовательского текста. Был изучен один из способов представления текста для передачи в ИНС. Был проведен эксперимент с ансамблированием разных моделей.

## Приложение

```
import pandas
import numpy as np
import plot_loss_acc
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Dropout,
LSTM, Flatten, SimpleRNN, GlobalMaxPooling1D
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import plot_model
#lr 7
from tensorflow.keras.datasets import imdb
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing import sequence

num_words = 10000
max_review_length = 500
test_len = 10000
embedding_vector_length = 32          #32
filters_ = 32                          # Conv1D
kernel_size = 4                       # Conv1D
pool_size = 2                         # MaxPooling1D
lstm_units = 100
validation_split=0.15

def clean_str(str):
    str = ''.join(c for c in str if c.isalpha() or
c.isspace()).expandtabs(tabsize=1).strip().lower()
    str = ' '.join(str.split())
    return str

def input_user_comment():
    print('Write your comment: ')
    result_str = ''
    while True:
        str = input()
        if str:
            result_str += str + " "
        else:
            break

    res_vect = clean_str(result_str).split(' ')
    return res_vect

def vectorize_comment(text, max_review_length=500):
    words_index = imdb.get_word_index()
    vectorized = []
    for word in text:
        ind = words_index.get(word)
        if ind is not None and ind < max_review_length:
            vectorized.append(ind + 3)

    res = []
    res.append(vectorized)
    return sequence.pad_sequences(res, maxlen=max_review_length)

def res_ensembling(models, x_test):
    predicts = []
    for i in range(len(models)):
```

```

        predicts.append(models[i].predict(x_test, verbose=0))
    return np.sum(predicts, axis = 0) * (1.0 / len(models))

def test_ensembling(y_test, final_preds):
    answer = 0
    for i in range(len(final_preds)):
        ans = 0
        if final_preds[i] > 0.5:
            ans = 1
        if ans == y_test[i]:
            answer = answer + 1
    print("#Ensembling accuracy: %.2f%%" % ((answer / len(y_test)*100)))

def create_model_1():
    model = Sequential()
    model.add(Embedding(num_words, 32))
    model.add(LSTM(32))
    model.add(Dense(1, activation='sigmoid'))
    return model

def create_model_2():
    model = Sequential()
    model.add(Embedding(num_words, 8, input_length=max_review_length)) # 2 ->
8
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model

def create_model_3():
    model = Sequential()
    model.add(Embedding(num_words, embedding_vector_length,
input_length=max_review_length))
    model.add(Conv1D(filters=filters_, kernel_size=kernel_size,
padding='same', activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(LSTM(lstm_units))
    model.add(Dense(1, activation='sigmoid'))
    return model

def create_model_4():
    model = Sequential()
    model.add(Embedding(num_words, 8, input_length=max_review_length))
    Dropout(0.2)
    model.add(Conv1D(100, 5, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(100, 5, activation='relu'))
    model.add(MaxPooling1D(pool_size=pool_size))
    Dropout(0.5)
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model

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

X_test = data[:test_len]
X_train = data[test_len:]

```



```

Y_test = targets[:test_len]
Y_train = targets[test_len:]

X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

model1 = create_model_1()
model2 = create_model_2()
model3 = create_model_3()
model4 = create_model_4()

models = [model1, model2, model3, model4]

train_len = len(X_train) // 4
test_len = len(X_test) // 4

epochs_num = [5, 7, 4, 3]
batch_size = 128

for i, model in enumerate(models):
    curr_x_train = X_train[i * train_len: (i+1)*train_len]
    curr_y_train = Y_train[i * train_len: (i+1)*train_len]
    curr_x_test = X_test[i * test_len: (i+1)*test_len]
    curr_y_test = Y_test[i * test_len: (i+1)*test_len]

    print("Model" + str(i) + " :")
    model.compile(
        optimizer="adam",
        loss="binary_crossentropy",
        metrics=["accuracy"]
    )

    history = model.fit(
        curr_x_train, curr_y_train,
        epochs=epochs_num[i],
        batch_size=128,
        verbose=1
    )

    acc = model.evaluate(curr_x_test, curr_y_test, verbose=0)
    print(model.summary())
    print("#Accuracy: %.2f%%" % (acc[1]*100))

#test ensembling
test_ensembling(Y_test, res_ensembling(models, X_test))

#test predict
comment = input_user_comment()
vec_comment = vectorize_comment(comment)
res = res_ensembling(models, vec_comment)
print(res)
if res[0] < 0.5:
    print('It`s a negative comment.')
else:
    print('It is positive comment')

```