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. 8382 _____ Нечепуренко Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цели работы.

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

Задачи.

- Ознакомиться с задачей классификации
- Изучить способы представления текста для передачи в ИНС
- Достигнуть точность прогноза не менее 95

Требования.

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)

Выполнение работы.

Импортируем необходимые зависимости и загрузим датасет IMDB.

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np
from keras.datasets import imdb
from matplotlib import pyplot as plt
from tensorflow.python.keras.layers import Dense, Dropout
from tensorflow.python.keras.models import Sequential

dim = 5000
(training_data, training_targets), (testing_data,
   testing_targets) = imdb.load_data(num_words=dim)
```

```
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets),
    axis=0)
```

В приведенном датасете каждое слово кодируется целым числом от 0 до dim. Необходимо перевести его в «нормализованный» вид, чтобы наблюдение содержало нули и единицы. Данное преобразование производится в функции vectorize. Формируется матрица, количество строк которой соответствует числу наблюдений в датасете, а количество столбцов соответствует количеству уникальных слов. Для каждого наблюдения единицей в j-той ячейке помечается слово с частотой j. Например, $(1, 3, 4) \to (1, 0, 1, 1)$.

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

Разделим данные на тренировочное и валидационное множество в пропорции 4 к 1. Исследуем результаты модели при различном размере вектора представления текста.

Размер вектора представления 1000.

Выберем следующую архитектуру модели:

```
model = Sequential()
model.add(Dense(64, activation="relu", input_shape=(dim,)))
model.add(Dropout(0.5))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1, activation="sigmoid"))
```

И обучим ее со следующими параметрами:

```
model.compile(
    optimizer="nadam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

results = model.fit(
    train_x, train_y,
    epochs=3,
    batch_size=600,
    validation_data=(test_x, test_y)
)
```

В результате запуска были получены следующие результаты (см. рис. 1).

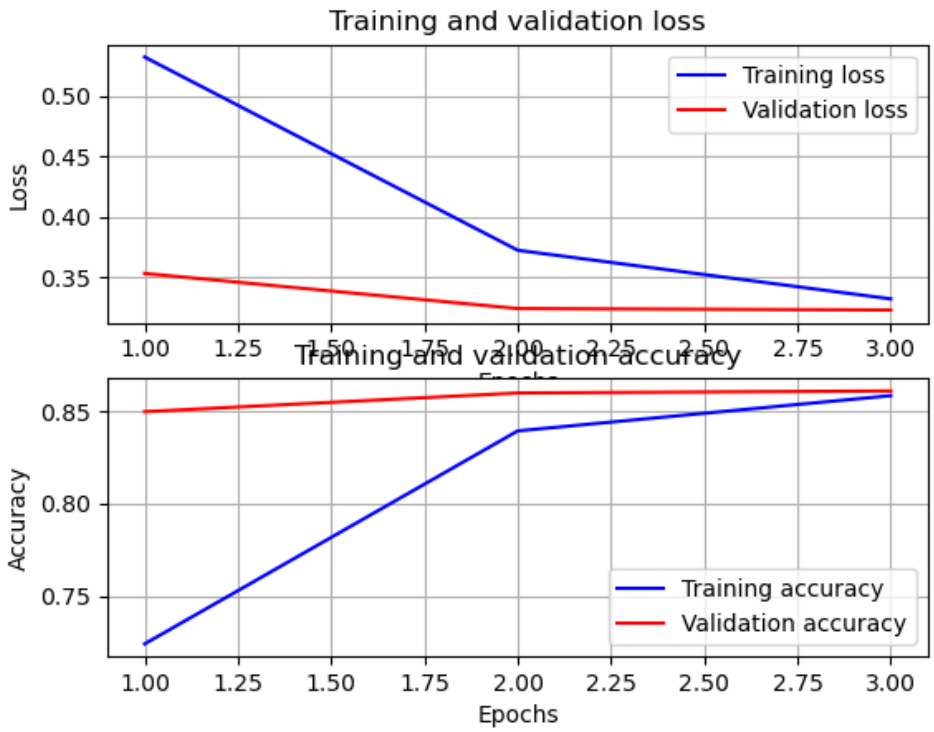

Рисунок 1 – Результат обучения модели (1000 слов)

Показатели точности на тренировочных и валидационных данных составили около 86% (0.8588 и 0.8610 соответственно).

Попробуем упростить модель, убрав из нее несколько слоев:

```
model = Sequential()
model.add(Dense(64, activation="relu", input_shape=(dim,)))
model.add(Dropout(0.5))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1, activation="sigmoid"))
```

Результаты модели не ухудшились, но архитектура стала проще, поэтому остановимся на этом варианте. В результате запуска были получены следующие результаты (см. рис. 1).

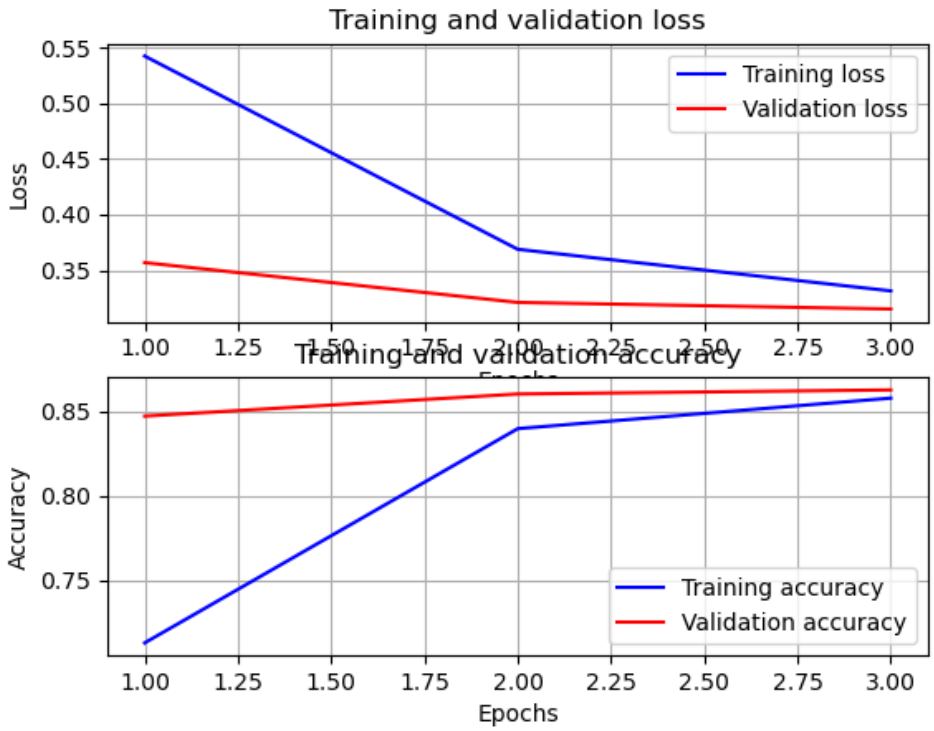

Рисунок 2 – Результат обучения упрощенной модели (1000 слов)

Точность на тренировочных данных – 0.8578, на валидационных – 0.8625.

Далее будем использовать последнюю приведенную архитектуру.

Размер вектора представления 5000.

Рассмотрим поведение модели при увеличении количества слов в векторе.

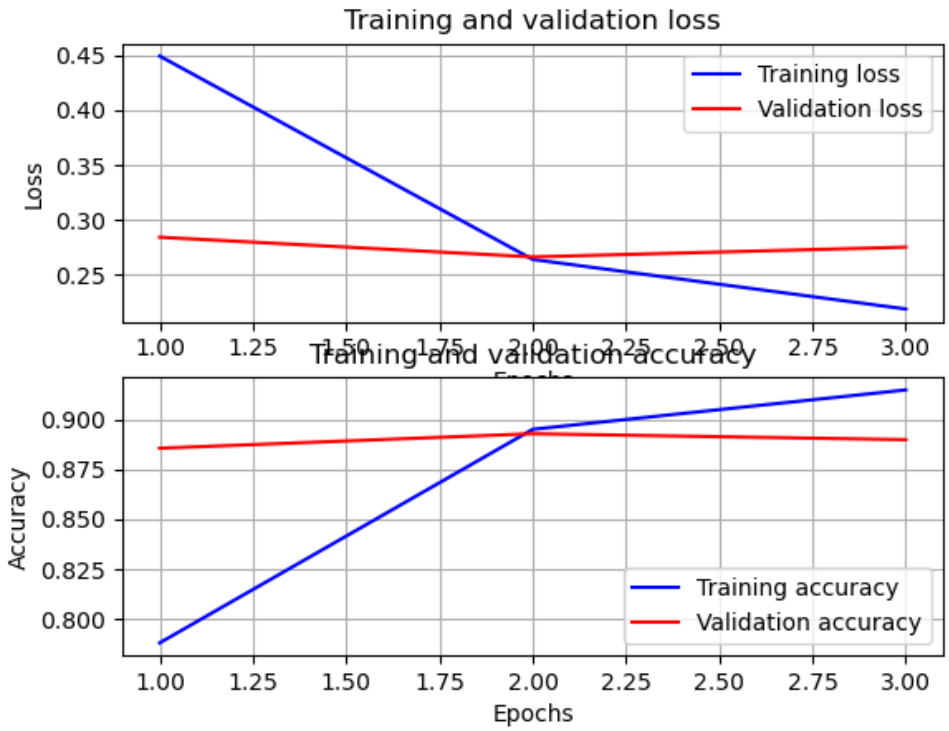

Рисунок 3 – Результат обучения модели (5000 слов)

Точность на тренировочных данных – 0.9168, на валидационных – 0.8900.

При увеличении количества слов в векторе представления с 1000 до 5000, были получены значимые отличия в качестве предсказаний модели при той же архитектуре.

Размер вектора представления 10000.

Проверим тенденцию, увеличив число слов с 5000 до 10000.

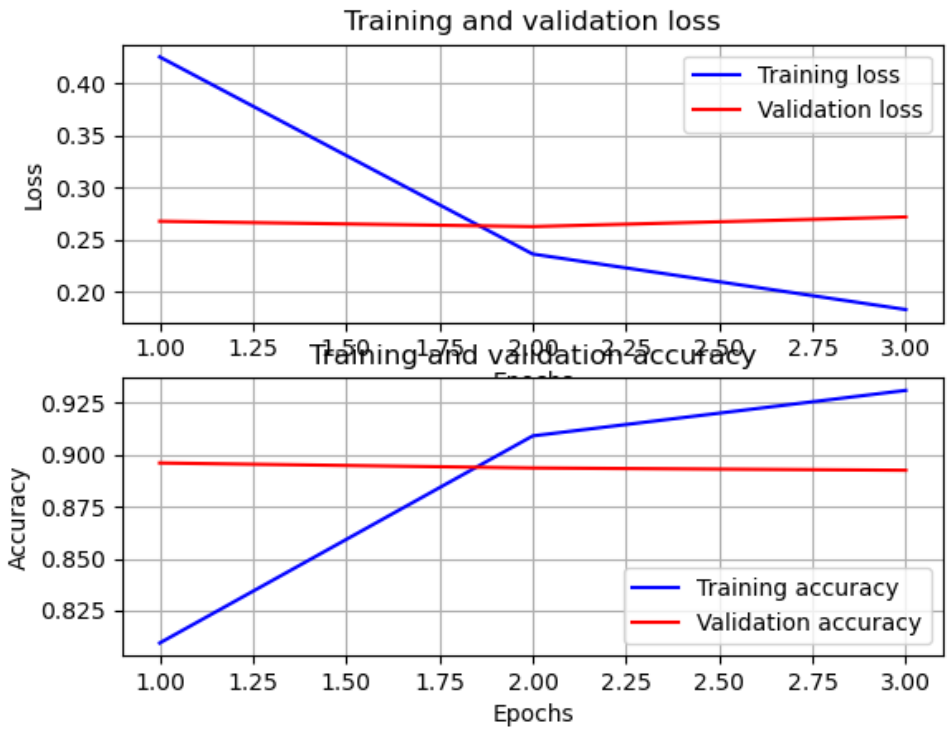

Рисунок 4 – Результат обучения модели (10000 слов)

Точность на тренировочных данных – 0.9348, на валидационных – 0.8926.

Увеличим количество эпох до 5 (см. рис. 5).

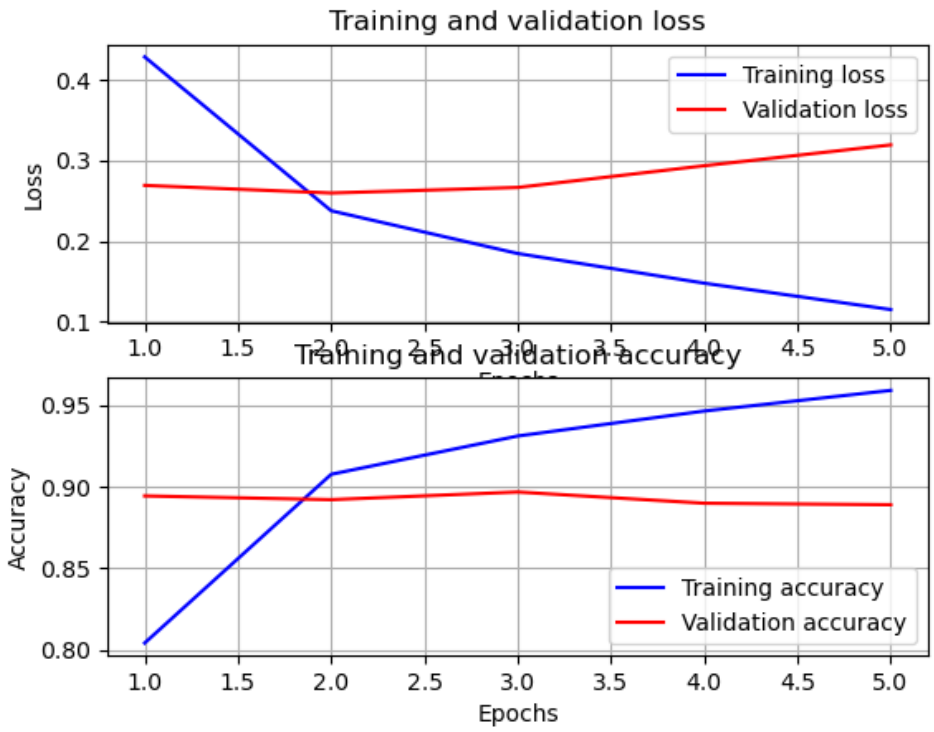

Рисунок 5 – Результат обучения модели (10000 слов, 5 эпох)

По логам можно увидеть деградацию точности на валидационных данных.

```
    Epoch 3/5
67/67 [==============================] - 3s 50ms/step - loss:
   0.1785 - accuracy: 0.9341 - val_loss: 0.2665 - val_accuracy:
   0.8969
    Epoch 4/5
67/67 [==============================] - 3s 41ms/step - loss:
   0.1419 - accuracy: 0.9501 - val_loss: 0.2934 - val_accuracy:
   0.8900
    Epoch 5/5
67/67 [==============================] - 3s 38ms/step - loss:
   0.1126 - accuracy: 0.9612 - val_loss: 0.3193 - val_accuracy:
   0.8891
```

После примерно 3 эпохи модель начинает переобучаться, добавление

слоев, изменение числа нейронов на слоях и вероятность dropout недостаточно сглаживают этот эффект. Требуемая точность больше 95% была достигнута, хоть и с небольшим переобучением. Сохраним эту модель для дальнейших предсказаний.

Предсказание произвольного комментария.

Для приведения произвольного комментария к необходимому виду воспользуемся функцией prepare.

```
def prepare(content, dim):
    words = [word.strip(":;?!.,'\"-_").lower() for word in
       content.strip().split()]
    index = imdb.get_word_index()
    data = []
    for word in words:
        if word in index and index[word] < dim:
            data.append(index[word])
    data = vectorize([np.array(data)], dim)
    return data
```

а затем предскажем класс обзора.

```
def predict(text):
    model = load_model("models/2.h5")
    x = prepare(text, 10000)
    prediction = model.predict(x)
    print(f"Prediction for this review is {prediction}")
    prediction_message(prediction)
```

Возьмем отзывы с сайта Rotten Tomatoes для фильма American History X. Примеры хороших отзывов:

1. A true masterpiece of storytelling.

2. An outstanding and well-thought-out story of a perspective of one's suffering by its acknowledged ideology. The acting and directing are finest bringing the

film an absolute classic. The emotions are well intact throughout the whole movie generating a realistic impression. I swear this film will strike you with feels and other senses depending on your notions.

Получаем предсказания:

```
Input filename of review: >? reviews/good1.txt
Prediction for this review is [[0.7674546]]
This review is a good one

Input filename of review: >? reviews/good2.txt
Prediction for this review is [[0.8983946]]
This review is a good one
```

Примеры плохих отзывов:

1. Not an exploration of the racial divide for the modern era; there is far too much prominently displayed nazi paraphernalia, and far too many white people yelling the n-word for an explicitly stated thesis of ”hate is bad.”It’s hard to take Edward Norton’s Derek Vinyard seriously as we move back and forth between his portrayals of a bright-eyed high-schooler, a murderous skin-head, and a hardened ex-con. The core narrative is simple, but the film is over-ambitious in its execution. The characters mostly fall flat, which makes it difficult to take their grappling with extremism seriously. The use of black and white is far too on the nose, and embodies the simplemindedness with which this complex subject matter is handled.

2. Tries to tackle racism head-on and in a raw, brutal way. But ends up treating the audience like juvenile idiots. Cheesy operatic music (screams: look at me, I’m a cool art house movie! This is meant to be a dramatic scene!) and pointless usage of slow-mo shots. Characters are caricatures. Derek Vinyard’s sweeping shift from hardcore skinhead to reformed man is unrealistic and almost comical.

Получаем предсказания:

```
Input filename of review: >? reviews/bad1.txt
```

```
Prediction for this review is [[0.07343122]]
This review is a bad one

Input filename of review: >? reviews/bad2.txt
Prediction for this review is [[0.1774292]]
This review is a bad one
```

Выводы.

В результате выполнения работы была реализована модель, позволяющая анализировать настроение человека по его комментарию к фильму. Была достигнута требуемая точность модели. Была реализована возможность ввода произвольного комментария с получением предсказания настроения.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np
from keras.datasets import imdb
from matplotlib import pyplot as plt
from tensorflow.python.keras.layers import Dense, Dropout
from tensorflow.python.keras.models import Sequential


def plot(epochs, train, validation, metrics):
    plt.plot(epochs, train, 'b', label=f'Training {metrics}')
    plt.plot(epochs, validation, 'r', label=f'Validation {metrics
       }')
    plt.title(f'Training and validation {metrics}')
    plt.xlabel('Epochs')
    plt.ylabel(metrics.capitalize())
    plt.grid(True)
    plt.legend()


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

    plt.figure()
    plt.subplot(211)
    plot(epochs, loss, val_loss, "loss")
```

```
    plt.subplot(212)
    plot(epochs, acc, val_acc, "accuracy")
    plt.show()


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


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


def decode_review(review):
    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
       review])
    print(decoded)


data = vectorize(data, dimension=dim)
targets = np.array(targets).astype("float32")
test_x = data[:10000]
```

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

model = Sequential()
model.add(Dense(64, activation="relu", input_shape=(dim,)))
model.add(Dropout(0.5))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(1, activation="sigmoid"))

model.compile(
    optimizer="nadam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

results = model.fit(
    train_x, train_y,
    epochs=3,
    batch_size=600,
    validation_data=(test_x, test_y)
)

plot_history(results.history)
```

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРЕДИКТОРА.

```
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np
from keras.datasets import imdb
from tensorflow.python.keras.models import load_model

from lb6.main import vectorize

model = load_model("models/2.h5")


def input_():
    filename = input("Input filename of review: ")
    with open(filename, "r") as f:
        predict(f.read())


def prepare(content, dim):
    words = [word.strip(":;?!.,'\"-_").lower() for word in
       content.strip().split()]
    index = imdb.get_word_index()
    data = []
    for word in words:
        if word in index and index[word] < dim:
            data.append(index[word])
    data = vectorize([np.array(data)], dim)
    return data


def prediction_message(prediction):
```

```
    if prediction > 0.7:
        print("This review is a good one")
    elif prediction < 0.3:
        print("This review is a bad one")
    else:
        print("Cannot determine a class of the review")


def predict(text):
    x = prepare(text, 10000)
    prediction = model.predict(x)
    print(f"Prediction for this review is {prediction}")
    prediction_message(prediction)
```