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

Студент гр.7382 _____ Токарев А.П.
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 отчёте)

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. Была построена и обучена нейронная сеть для обработки текста. Код предоставлен в приложении А. Были использованы рекуррентные нейронные сети, они хорошо подходят для обработки текстов, т.к. могут хранить свое состояние и принимают текущее решение с учетом предыдущих. Также будем использовать одномерную свертку и пулинг. Данная архитектура дает точность: на тренировочных ~ 91,02%, на валидационных ~ 89,4%. Графики точности и ошибки предоставлены на рис. 1 и рис. 2 соответственно.

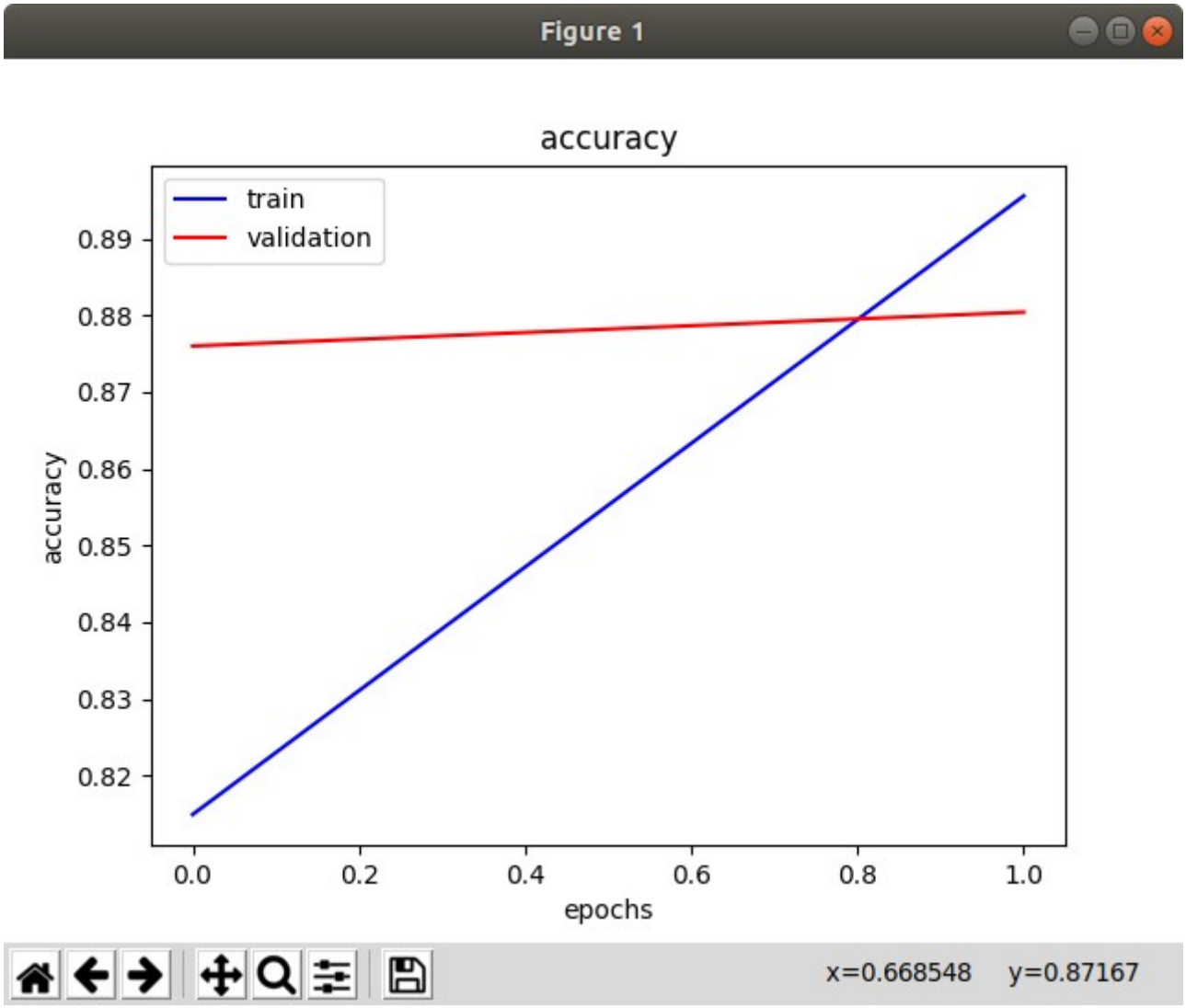

Рисунок 1 – График точности без dropout

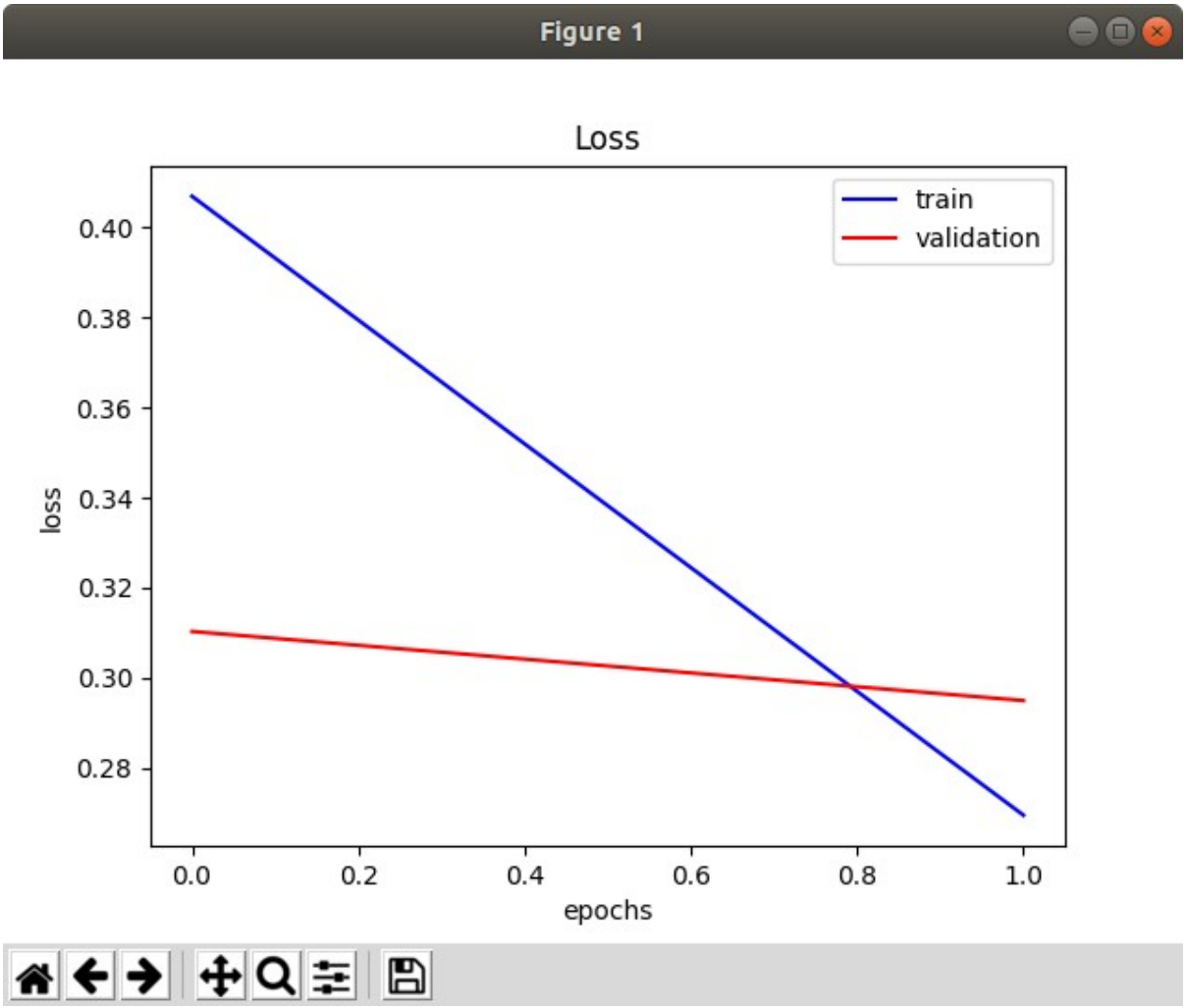

Рисунок 2 – График потерь без dropout

2. Рекуррентные нейронные сети, такие как LSTM, обычно имеют проблему переобучения. Решим эту проблему путем добавления в архитектуру сети слоев Dropout сделаем его около 0.3.

Сравнивая рис. 1 и рис. 2 и рис. 3 и рис. 4 мы видим, что без Dropout ошибка на валидации пошла вверх, что говорит о переобучении. Dropout используется для устранения переобучения, путем случайного отключения связей или нейронов, таким образом, что-либо связь выдает нулевой сигнал на выход, либо нейрон выдает на все свои выходы нулевой сигнал. Точность: на тренировочных ~ 91,8%, на валидационных ~ 89,91%. Графики точности и ошибки предоставлены на рис. 3 и рис. 4 соответственно.

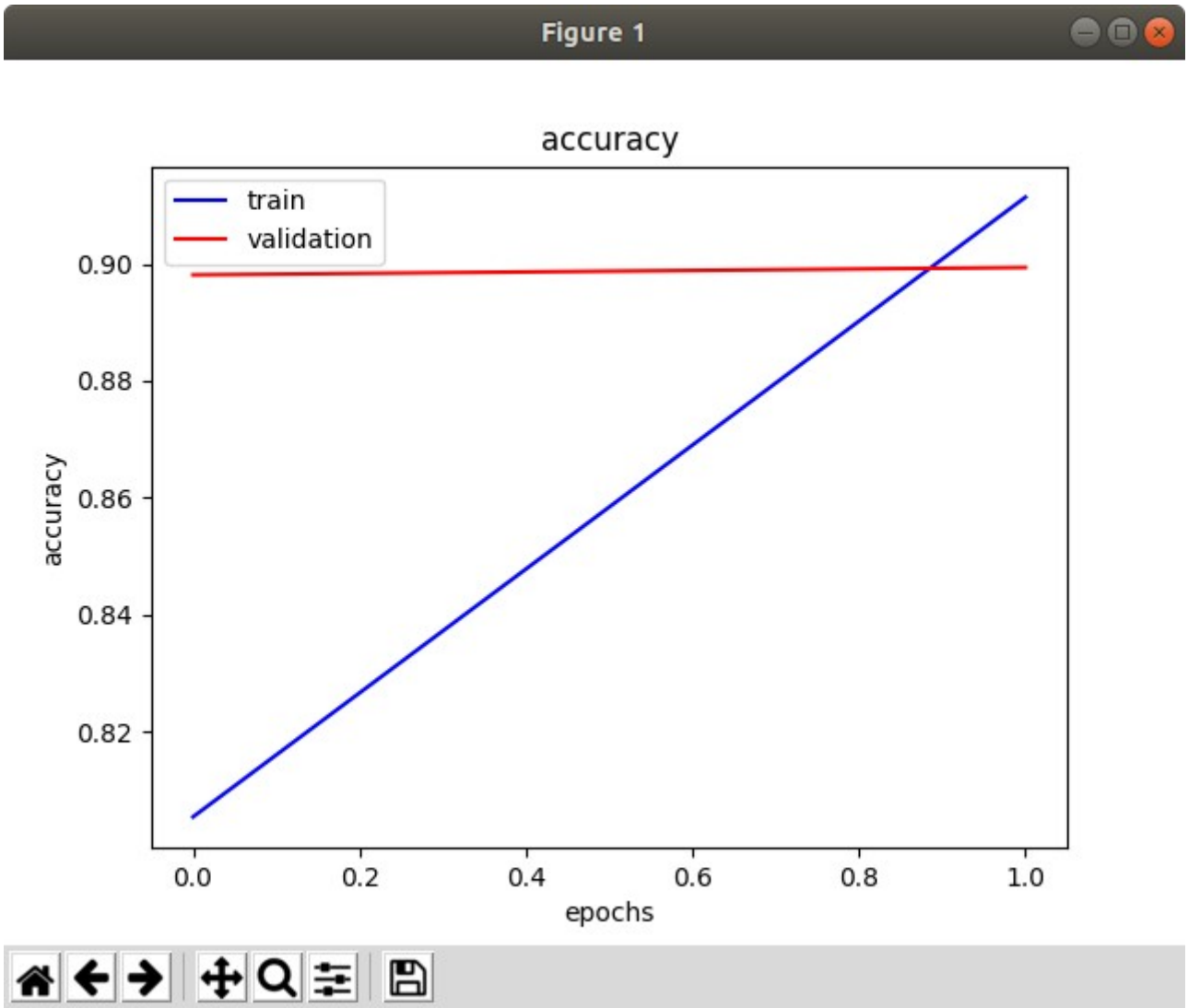

Рисунок 3 – График точности с dropout

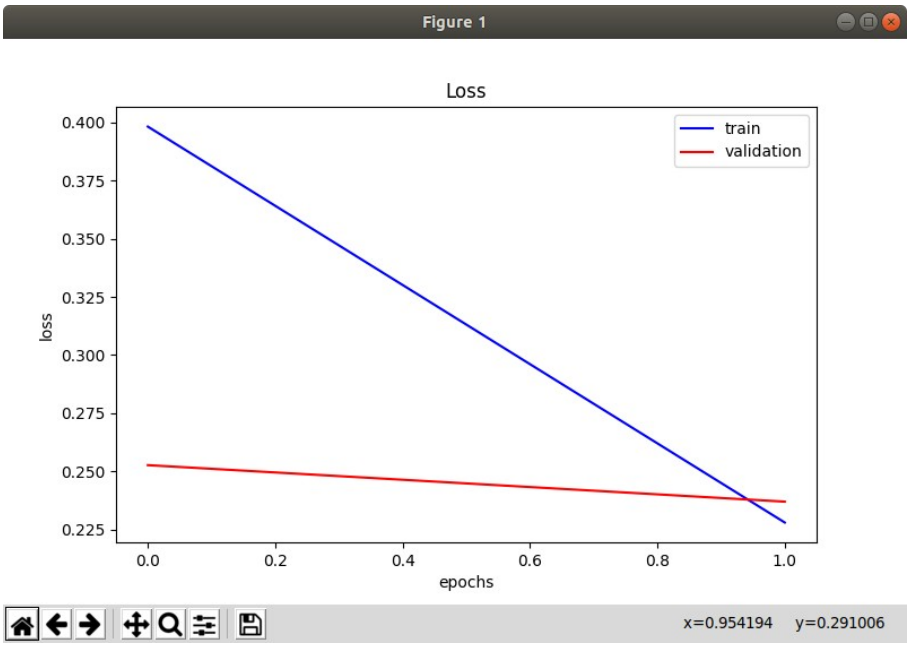

Рисунок 4 – График потерь с dropout

Функция, выполняющая ансамблирование:

```
def smart_predict(model, model_2, input):
    pred1 = model.predict(input)
    pred2 = model_2.predict(input)
    pred = [1 if (pred1[i] + pred2[i]) / 2 > 0.5 else 0 for i in range(len(pred1))]
    return np.array(pred)
```

Напишем функцию, которая позволяет ввести пользовательский текст. 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. На выходе нейронная сеть получила обзоры «0 0 1 1 1», 0 – отрицательные, 1 – положительные.

```
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
```

Выводы.

В ходе работы была изучена задача классификация обзоров из датасета IMDB. Подобрана архитектура, дающая точность 91,8%. Проведя исследование, было выяснено, что при добавлении нескольких слоев свертки и пулинга увеличивается точность предсказания.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
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):
	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()
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's very boring film",
	"it's very good",
	"it's boring",
	"fantastic film wonderful",
	"beautiful, good"
]
custom_y = [0., 1., 0, 1., 1.]
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))
(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, random_state=123)
y_test = np.asarray(y_test).astype("float32")
y_train = np.asarray(y_train).astype("float32")
custom_y = np.asarray(custom_y).astype("float32")
def encode_review(rev):
	res = []
	for i, el in enumerate(rev):
		el = el.lower()
		delete_el = [',', '!', '.', '?']
		for d_el in delete_el:
			el = el.replace(d_el, '')
		el = el.split()
		for j, word in enumerate(el):
			code = imdb.get_word_index().get(word)
			if code is None:
				code = 0
			el[j] = code
		res.append(el)
	for i, r in enumerate(res):
		res[i] = sequence.pad_sequences([r], maxlen=MAX_REVIEW_LENGTH)
	res = np.array(res)
	return res.reshape((res.shape[0], res.shape[2]))
def smart_predict(model, model_2, input):
	pred1 = model.predict(input)
	pred2 = model_2.predict(input)
	pred = [1 if (pred1[i] + pred2[i]) / 2 > 0.5 else 0 for i in range(len(pred1))]
	return np.array(pred)
def review(model, model_2, review):
	data = encode_review(review)
	print(smart_predict(model, model_2, data))
model = Sequential()
model.add(Embedding(10000, 32, input_length=500))
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'])
model_2 = Sequential()
model_2.add(Embedding(10000, 32, input_length=500))
model_2.add(LSTM(100))
model_2.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_2.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)
```

```
plot_loss(H.history['loss'], H.history['val_loss'])
plot_acc(H.history['accuracy'], H.history['val_accuracy'])
custom_loss, custom_acc = model.evaluate(custom_x, custom_y)
H_2 = model_2.fit(x_train, y_train, batch_size=64, epochs=2, verbose=1, validation_split=0.1)
plot_loss(H_2.history['loss'], H_2.history['val_loss'])
plot_acc(H_2.history['accuracy'], H_2.history['val_accuracy'])

review(model, model_2, [
        "It is bad, i hate it",
        "It's too boring and awful",
        "It's amazing, fantastic and exiting",
        "Fine film, i love it too much",
        "Really good, fantastic actors, i like"
])
```