МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Искусственные нейронные сети»

Тема: Регрессионная модель изменения цен на дома в Бостоне

Студент гр. 7382 _____ Токарев А.П.

Преподаватель _____ Жукова Н.А.

Санкт-
Петербург
2020

Цель работы.

Реализовать предсказание медианной цены на дома в пригороде Бостона в середине 1970-х по таким данным, как уровень преступности, ставка местного имущественного налога и т. д.

Порядок выполнения работы.

1. Ознакомиться с задачей регрессии
2. Изучить отличие задачи регрессии от задачиклассификации
3. Создать модель
4. Настроить параметрыобучения
5. Обучить и оценитьмодели
6. Ознакомиться с перекрестнойпроверкой

Требования к выполнению задания.

1. Объяснить различия задач классификации и регрессии
2. Изучить влияние кол-ва эпох на результат обучения модели
3. Выявить точку переобучения
4. Применить перекрестную проверку по K блокам при различных K
5. Построить графики ошибки и точности во время обучения для моделей, а также усредненные графики по всем моделям

Основные теоретические положения.

1. *Классификационное моделирование*- это задача приближения функции отображения *f* от входных переменных (*X)* к дискретным выходным переменным (*Y*).

Задача классификации требует, разделения объектов в один или два класса.

2. *Регрессионное моделирование*- это задача приближения функции отображения *f* от входных переменных (*X)*) к непрерывной выходной переменной(*Y*).

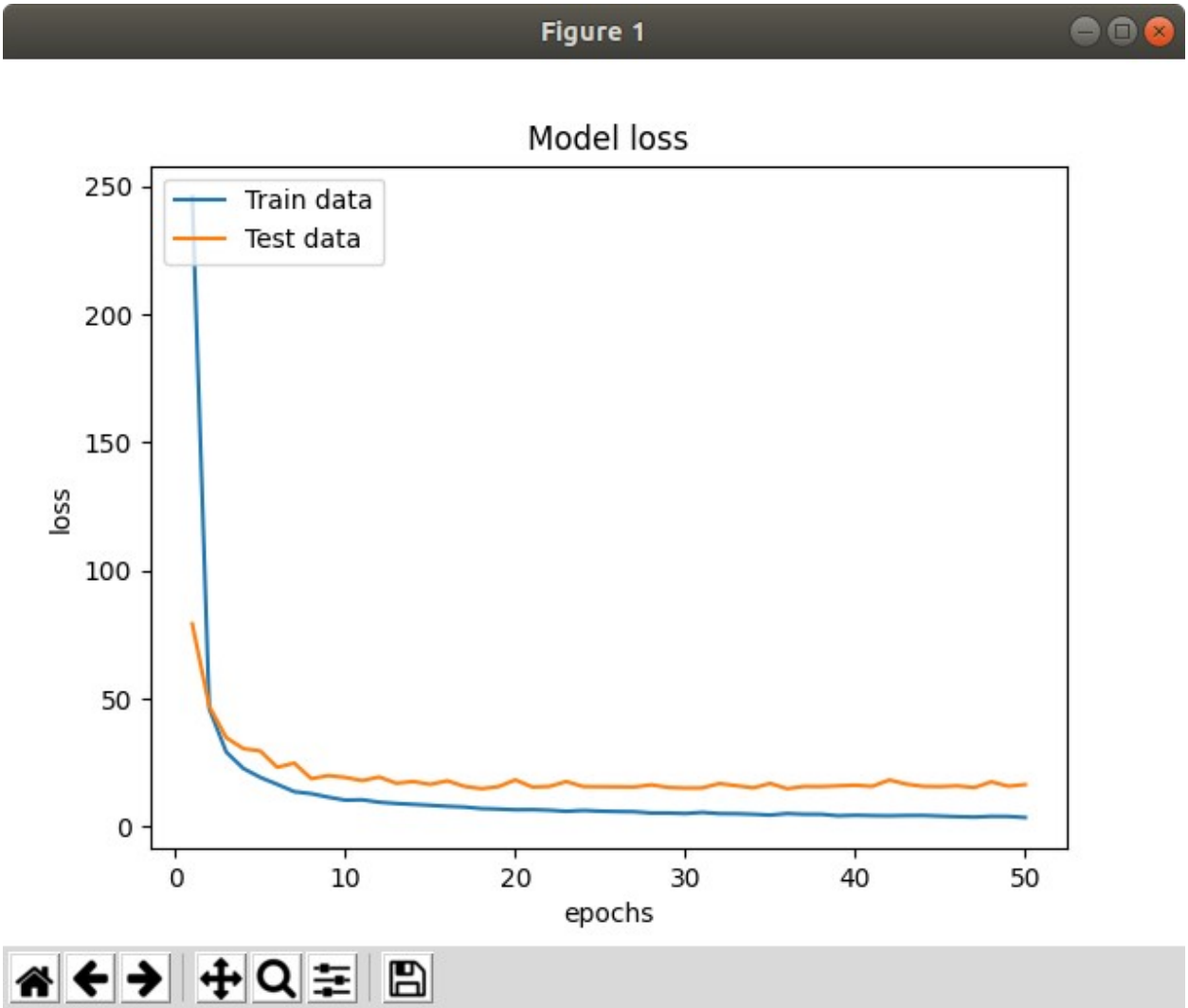
Задача регрессии требует предсказания количества.

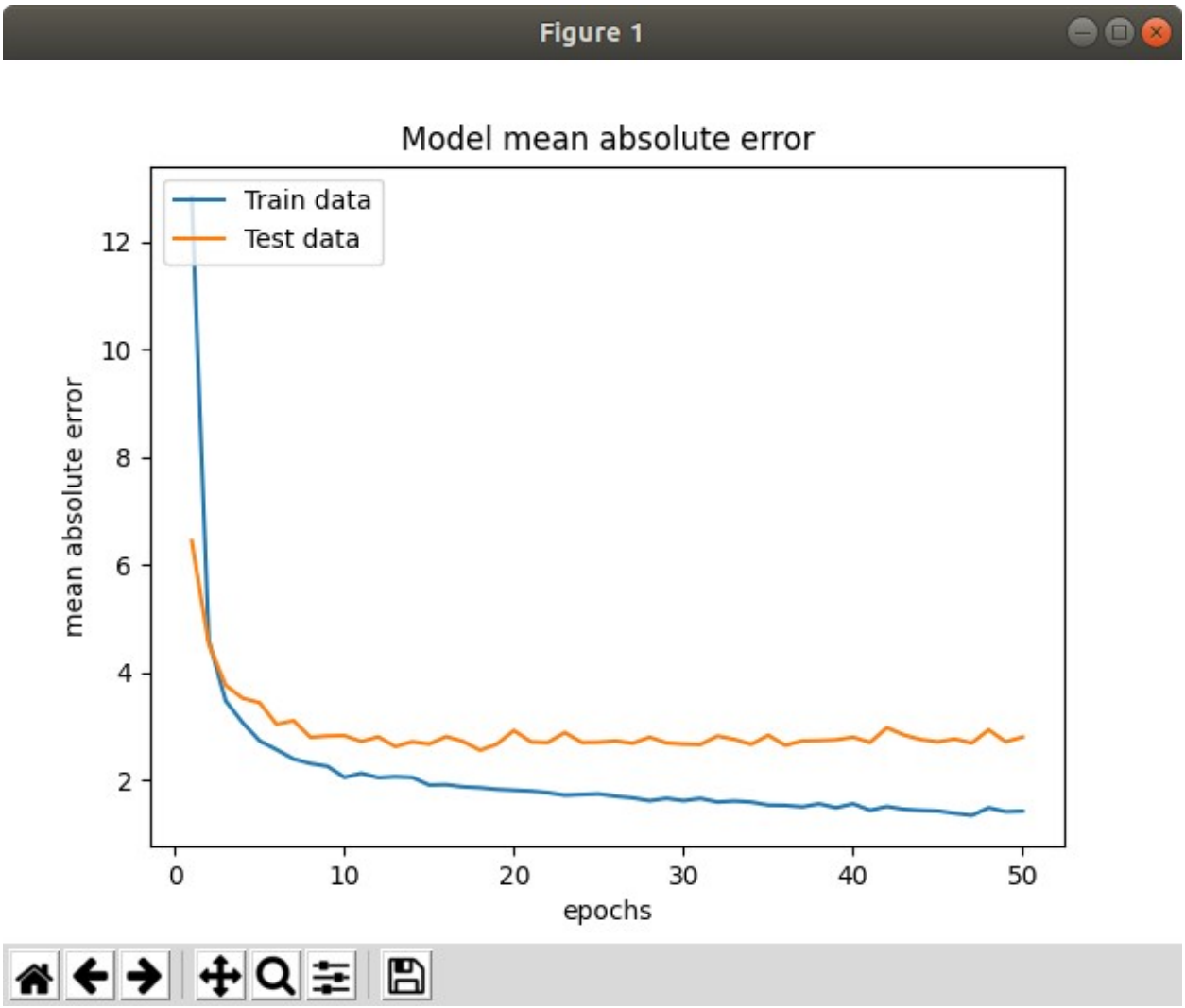
Ход работы.

Была создана и обучена модель искусственной нейронной сети. Код предоставлен в приложении А.

Для выполнения поставленной задачи были опробованы разнообразные архитектуры сети, обучение проводилось при различных параметрах, было изменено количество блоков 2,4,8,12,24.

Сначала эмпирическим путём найдём точку переобучения при K=2.





Как видим, переобучение начинается на 15-25 эпохах, чтобы убедиться в этом, посмотрим на графики при 500 эпохах:

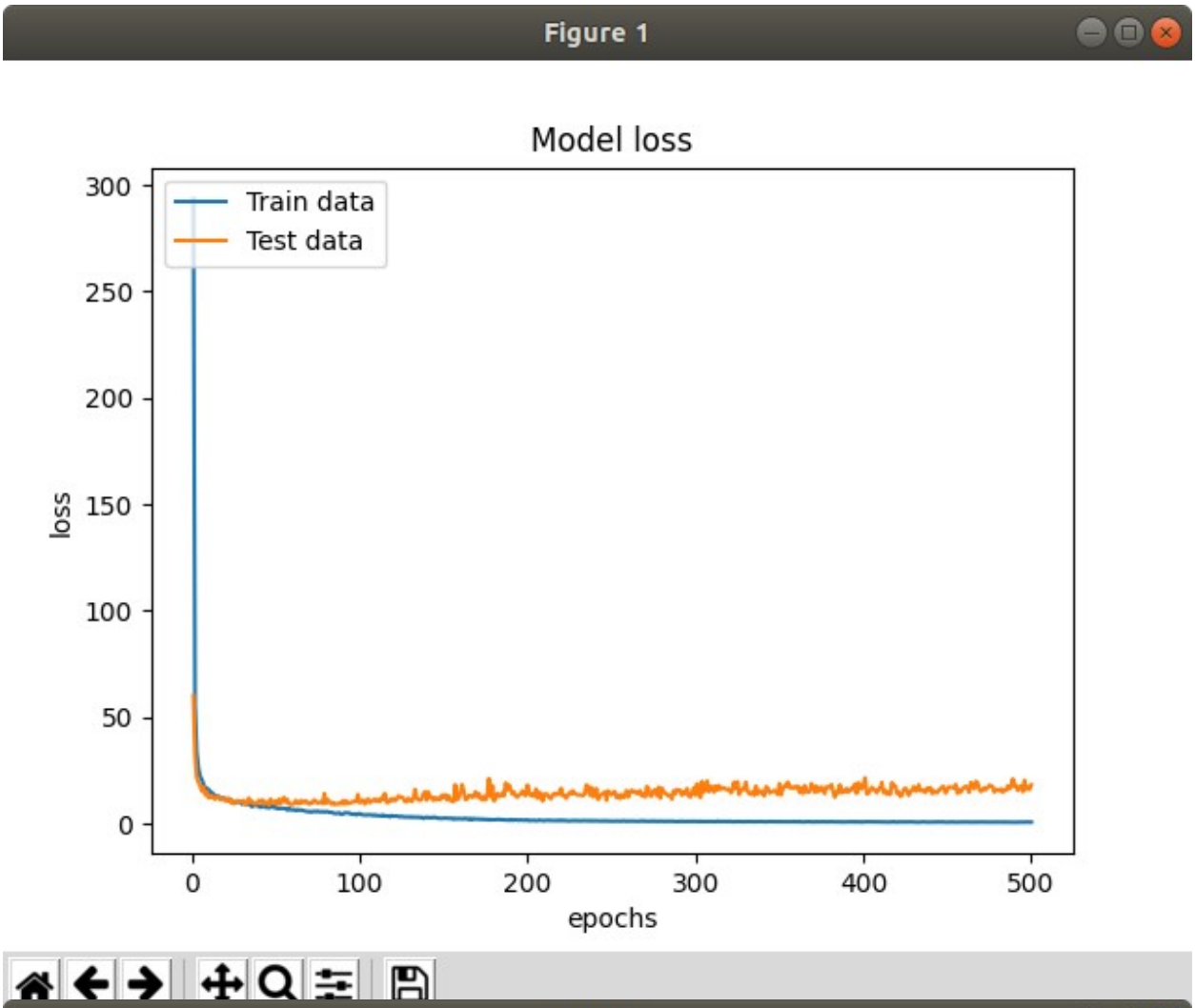


Figure 1
Model loss
Train data
Test data
loss
epochs
0
50
100
150
200
250
300
0
100
200
300
400
500

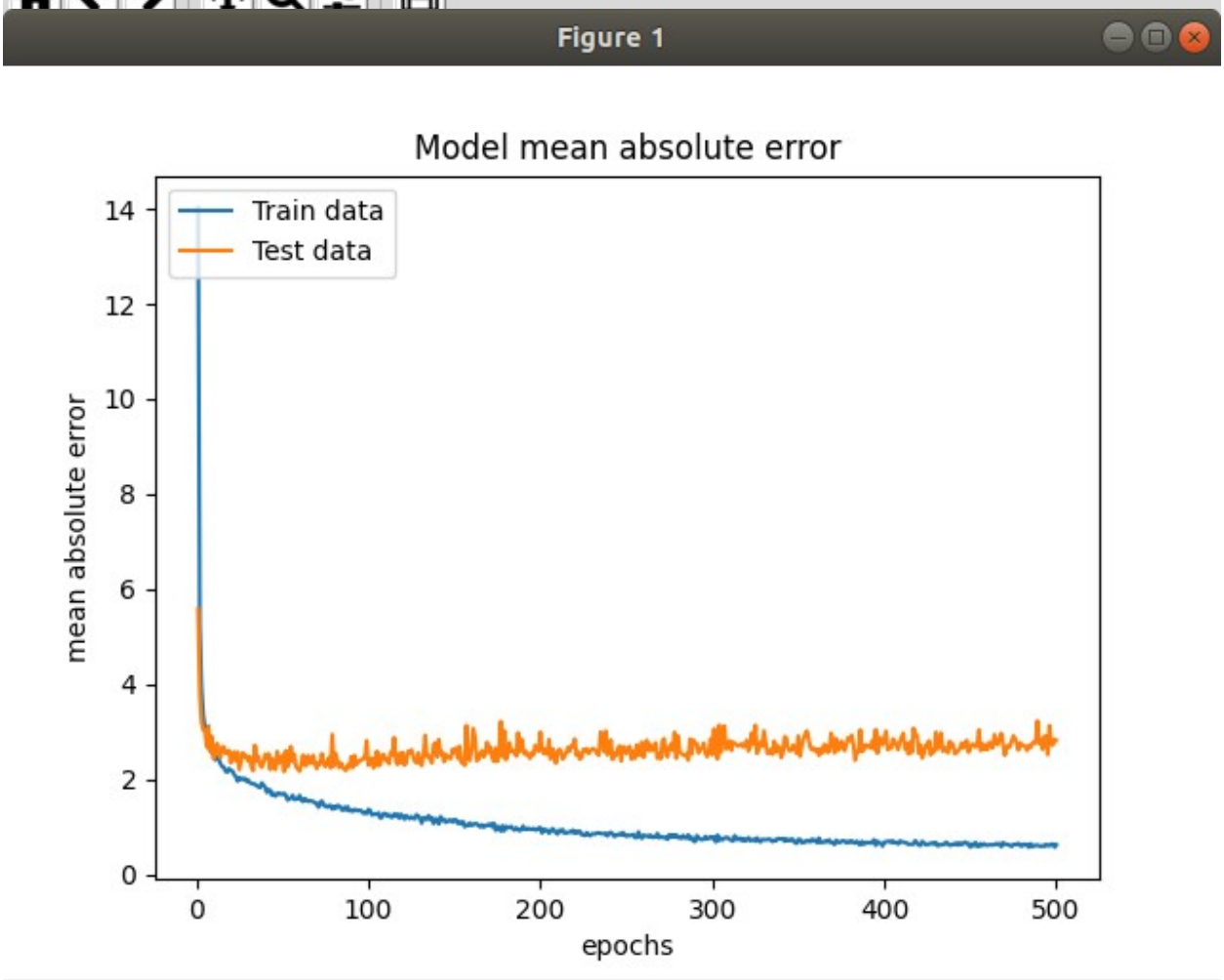


Figure 1
Model mean absolute error
Train data
Test data
mean absolute error
epochs
0
2
4
6
8
10
12
14
0
100
200
300
400
500
x=31.3827
y=2.26988

Здесь чётко видно переобучение в районе 30 эпох.

Что же касается количества блоков, то рассмотрим данные по средней ошибке для различного K:

K=2 MAE = 2.684626340866089

K=4 MAE = 2.4150189459323883

K=8 MAE = 2.2920690923929214

K=12 MAE = 2.268309007088343

K=24 MAE = 2.281951288382212

Как видим, сильное уменьшение размера блоков (увеличение их количества) ведёт к ухудшению точности сети после K = 12.

Из графиков видно, что точка переобучения находится где-то между 15 и 25 эпохами, так как средняя ошибка на тестовых данных начинала возрастать (на графике с 50 эпохами не очень заметно, но на 500 видно возрастание ошибки от ~30 эпох). Поэтому оптимальным вариантом является k=12 и кол-во эпох = 20.

Выводы.

В ходе работы было изучено влияние числа эпох на результат обучения в задаче регрессии, найдена точка переобучения, которое происходит на 15-25 эпохах. Оптимальным вариантом будет модель с 12-ю блоками и 20 эпохами.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import tensorflow
import numpy as np
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import boston_housing
import matplotlib.pyplot as plt

(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
k = 24
num_val_samples = len(train_data) // k
num_epochs = 30
all_scores = []



def build_model():
    model = Sequential()
    model.add(Dense(64, activation='relu',
input_shape=(train_data.shape[1],)))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='rmsprop', loss='mse',
metrics=['mean_absolute_error'])
    return model
```

```
def fit_model():
    res = []
    for i in range(k):
        print('processing fold #', i)
        val_data = train_data[i * num_val_samples: (i + 1) *
num_val_samples]
        val_targets = train_targets[i * num_val_samples: (i +
1) * num_val_samples]
        partial_train_data = np.concatenate([train_data[:i *
num_val_samples], train_data[(i + 1) *
num_val_samples:]],axis=0)
        partial_train_targets = np.concatenate(
            [train_targets[:i * num_val_samples],
train_targets[(i + 1) * num_val_samples:]], axis=0)
        model = build_model()
        H = model.fit(partial_train_data,
partial_train_targets, epochs=num_epochs, batch_size=1,
verbose=0, validation_data=(val_data, val_targets))
        val_mse, val_mae = model.evaluate(val_data,
val_targets, verbose=0)
        all_scores.append(val_mae)
        loss = H.history['loss']
        mae = H.history['mean_absolute_error']
        v_loss = H.history['val_loss']
        v_mae = H.history['val_mean_absolute_error']
        x = range(1, num_epochs + 1)
        # plt.plot(x, loss)
        # plt.plot(x, v_loss)
        # plt.title('Model loss')
        # plt.ylabel('loss')
        # plt.xlabel('epochs')
        # plt.legend(['Train data', 'Test data'], loc='upper
left')
        # plt.show()
        # plt.plot(x, mae)
```

```
        # plt.plot(x, v_mae)
        # plt.title('Model mean absolute error')
        # plt.ylabel('mean absolute error')
        # plt.xlabel('epochs')
        # plt.legend(['Train data', 'Test data'], loc='upper
left')
        # plt.show()
    print(np.mean(all_scores))


fit_model()
```