**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**по дисциплине «Искусственные нейронные сети»**

**Тема: «Регрессионная модель изменения цен на дома в Бостоне»**

Студентка гр. 7382 Лящевская А.П.

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | Жукова Н. А. |

Санкт-Петербург

2020

**Цели.** 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.

Данный набор содержит относительно немного образцов данных: всего 506, разбитых на 404 обучающих и 102 контрольных образца. И каждый признак во входных данных (например, уровень преступности) имеет свой масштаб.

**Задачи.**

* Ознакомиться с задачей регрессии
* Изучить отличие задачи регрессии от задачи классификации
* Создать модель
* Изучить влияние кол-ва эпох на результат обучения модели
* Выявить точку переобучения
* Применить перекрестную проверку по K блокам при различных K
* Построить графики ошибки и точности во время обучения для моделей, а также усредненные графики по всем моделям

**Выполнение работы.**

1) Была создана и обучена модель искусственной нейронной сети в соответствии с условиями(весь код представлен в приложении А). 2) Для проверки влияния количества эпох на результат обучения модели был выбран диапазон от 10 до 23. Ниже на рис. 1 представлен график среднего абсолютного отклонения модели при всех значениях в зависимости от количества эпох.

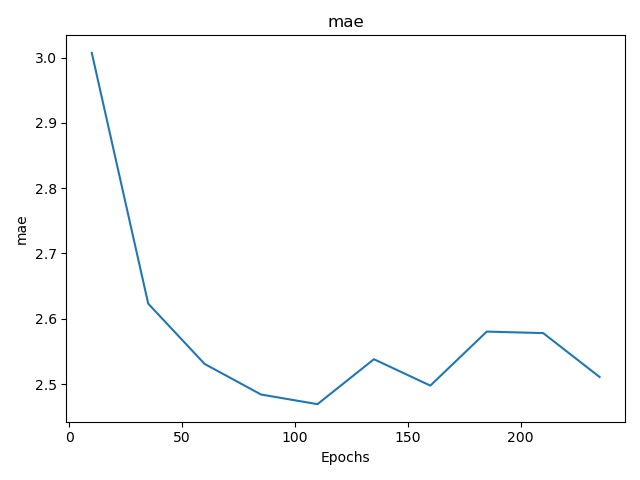


рисунок 1 – график зависимости средней абсолютной ошибки от кол-ва эпох обучения

1. Как видно из графика, оптимальным числом эпох является число 110. Далее нужно определить оптимальное число К. Для этого были перебраны значения от 3 до 8. Ниже представлен график среднего абсолютного отклонения модели при всех значениях в зависимости от значения К.

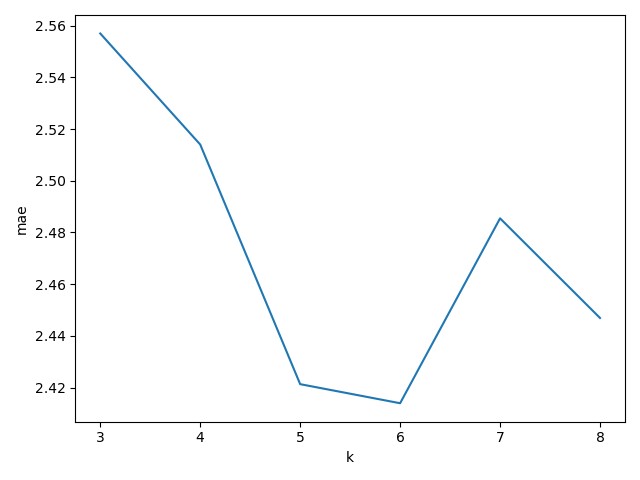


рисунок 2 – график зависимости средней абсолютной ошибки от числа k

По графику видно, что оптимальным К является 6.

1. Были построены графики точности и ошибок обучения модели с параметрами: количество эпох обучения - 110, количество блоков – 6.

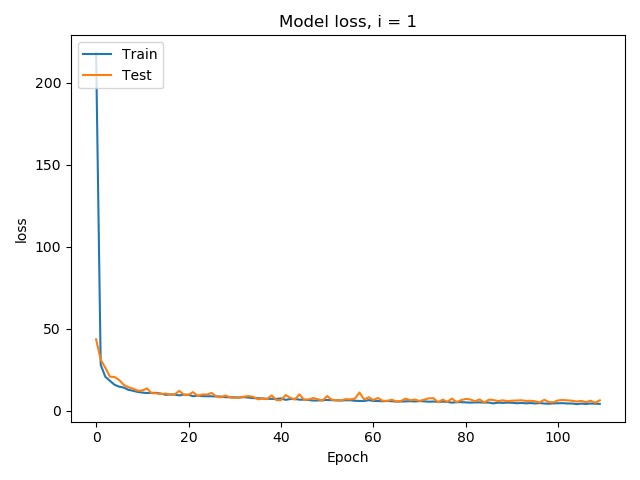
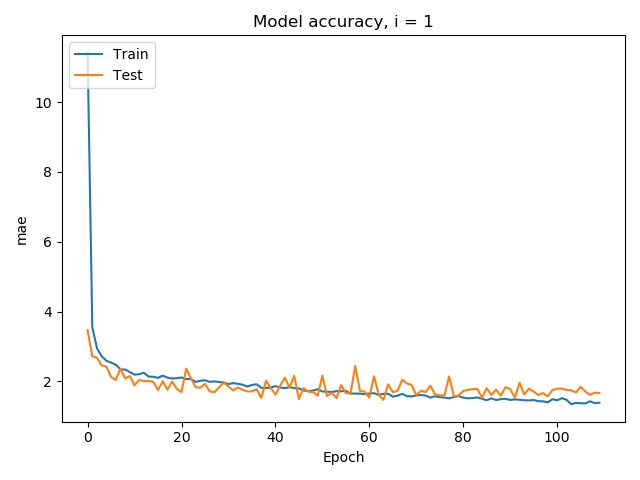


рисунок 3 – график точности и обучения модели на 1-ом блоке

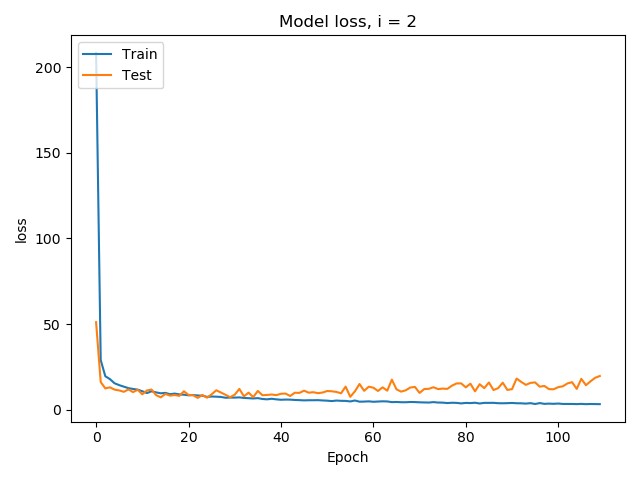
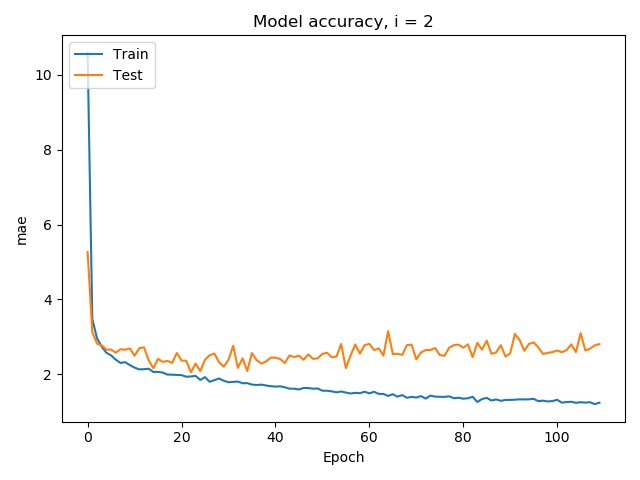


рисунок 4 – график точности и обучения модели на 2-ом блоке

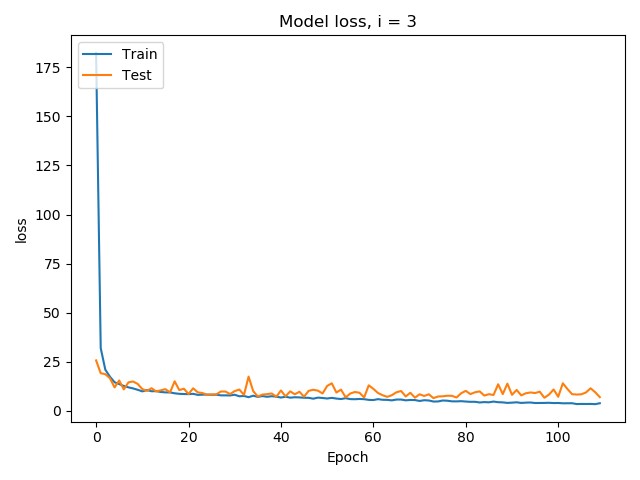
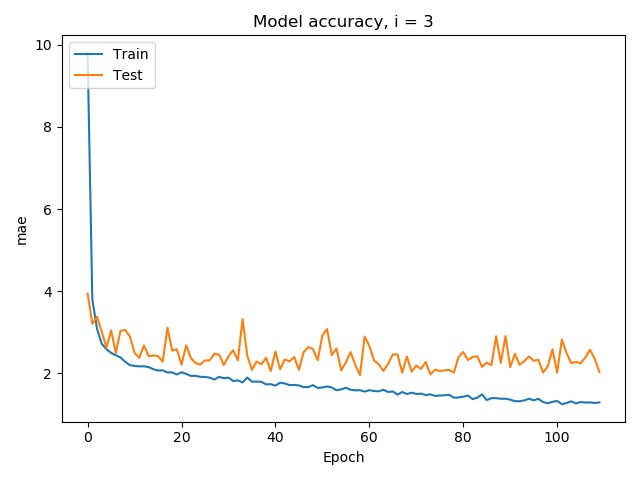


рисунок 5 – график точности и обучения модели на 3-ом блоке

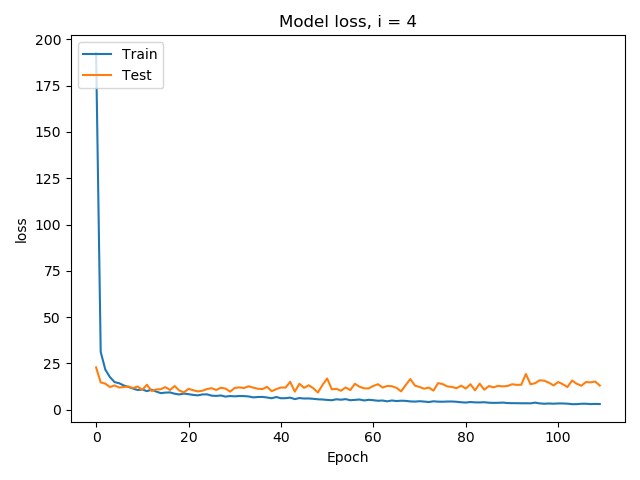
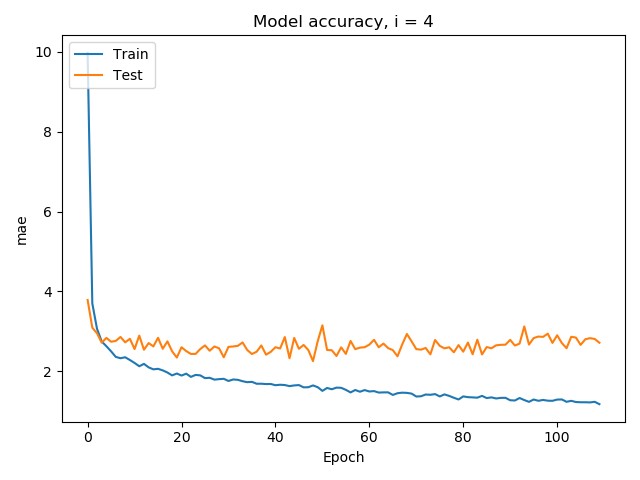


рисунок 6 – график точности и обучения модели на 4-ом блоке

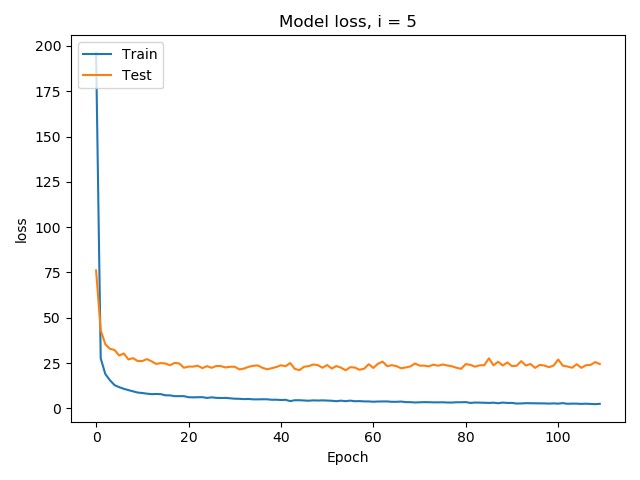
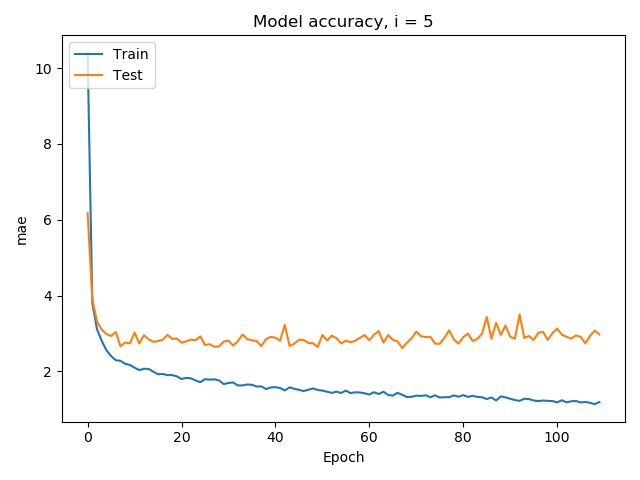


рисунок 7 – график точности и обучения модели на 5-ом блоке

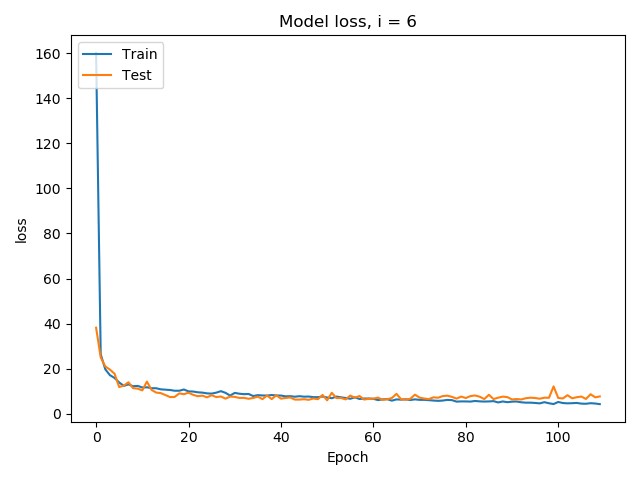
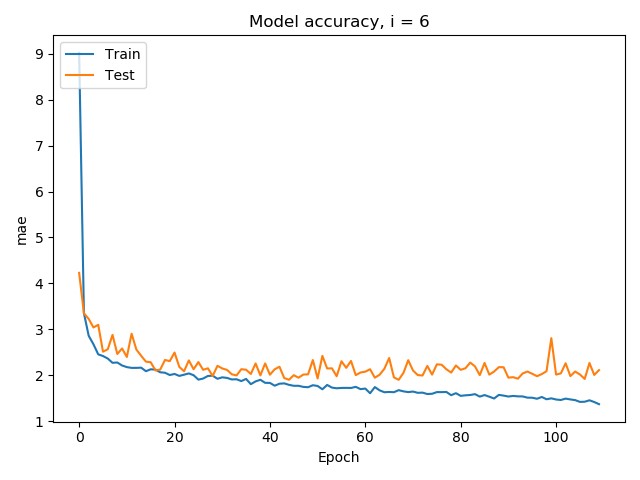


рисунок 8– график точности и обучения модели на 6-ом блоке

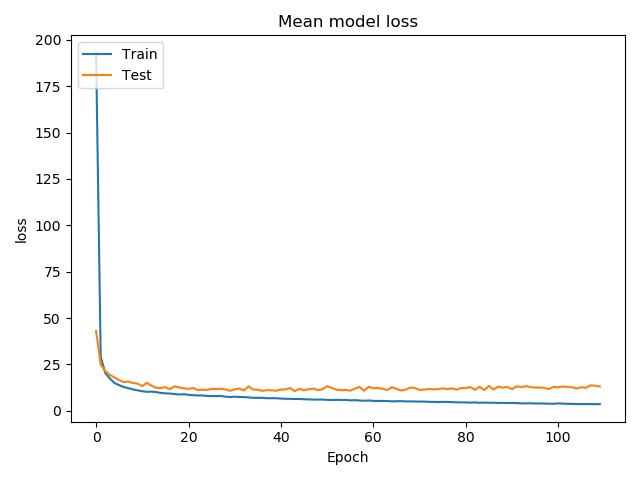
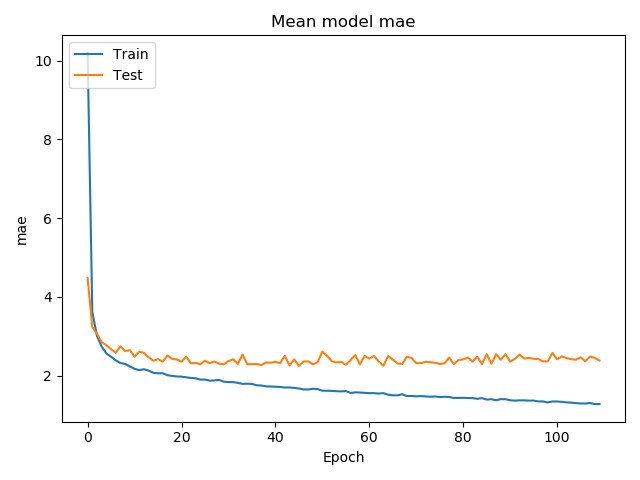


рисунок 9 – график точности и обучения усредненной модели

**Вывод.**

В ходе выполнения данной работы была изучена задача регрессии с помощью библиотеки keras и ее отличие от задачи классификации.

**ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential

from tensorflow.keras.datasets import boston\_housing

import numpy as np

import matplotlib.pyplot as plt

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=['mae'])

return model

(train\_data, train\_targets), (test\_data, test\_targets) = boston\_housing.load\_data()

mean = train\_data.mean(axis=0)

std = train\_data.std(axis=0)

train\_data -= mean

train\_data /= std

test\_data -= mean

test\_data /= std

k = 7

num\_val\_samples = len(train\_data) // k

num\_epochs = 150

all\_mae\_histories = []

mean\_val\_mae = []

for i in range(k):

print(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\_target = np.concatenate([train\_targets[: i \* num\_val\_samples],

train\_targets[(i + 1) \* num\_val\_samples:]], axis=0)

model = build\_model()

history = model.fit(partial\_train\_data, partial\_train\_target, epochs=num\_epochs, batch\_size=1,

validation\_data=(val\_data, val\_targets), verbose=0)

mean\_val\_mae.append(history.history['val\_mean\_absolute\_error'])

plt.plot(history.history['mean\_absolute\_error'])

plt.plot(history.history['val\_mean\_absolute\_error'])

plt.title('Block #' + str(i+1))

plt.ylabel('mae')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'])

plt.show()

def smooth\_curve(points, factor=0.9):

smoothed\_points = []

for point in points:

if smoothed\_points:

prev = smoothed\_points[-1]

smoothed\_points.append(prev\*factor+point\*(1-factor))

else:

smoothed\_points.append(point)

return smoothed\_points

average\_mae\_history = [np.mean([x[i] for x in mean\_val\_mae]) for i in range(num\_epochs)]

smooth\_mae\_history = smooth\_curve(average\_mae\_history)

plt.plot(range(1, len(smooth\_mae\_history)+1), smooth\_mae\_history)

plt.xlabel('EPOCHS')

plt.ylabel("Validation MAE")

plt.show()