**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»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7383 |  | Зуев Д.В. |
| 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.

**Задачи.**

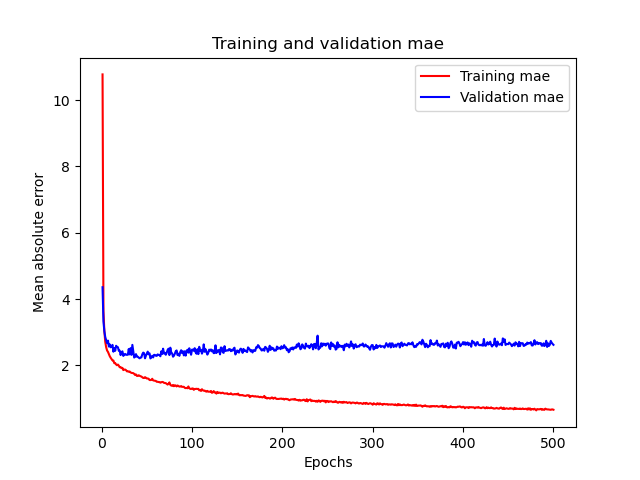
* Ознакомиться с задачей регрессии
* Изучить отличие задачи регрессии от задачи классификации
* Создать модель
* Настроить параметры обучения
* Обучить и оценить модели
* Ознакомиться с перекрестной проверкой

**Ход работы.**

Задача классификации определяет принадлежность объекта, описанного входными данными, к одному из заданных классов, а задача регрессии определяет значение какой-либо характеристики объекта, в зависимости от характеристик объекта, подаваемых на вход. В задаче классификации результатом будет значение из конечного множества значений, а результатом задачи регрессии может быть любое число.

1. Была создана и обучена модель искусственной нейронной сети для нахождения оптимального числа эпох (код представлен в приложении А). Количество блоков было выбрано равным 4, а количество эпох – равным 500.

Результат обучения нейронной сети представлен на графике на рис. 1.

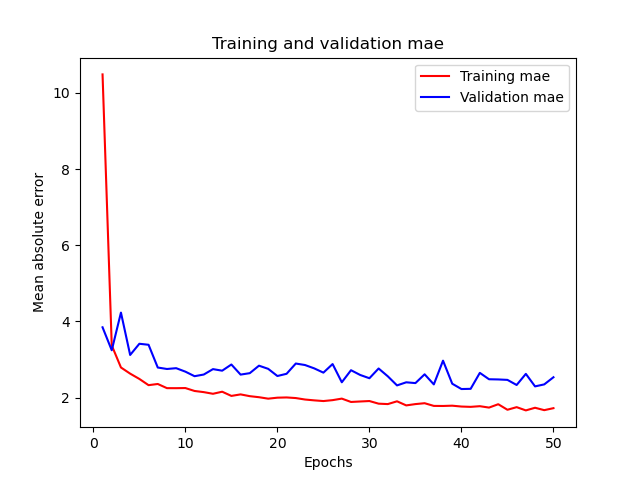
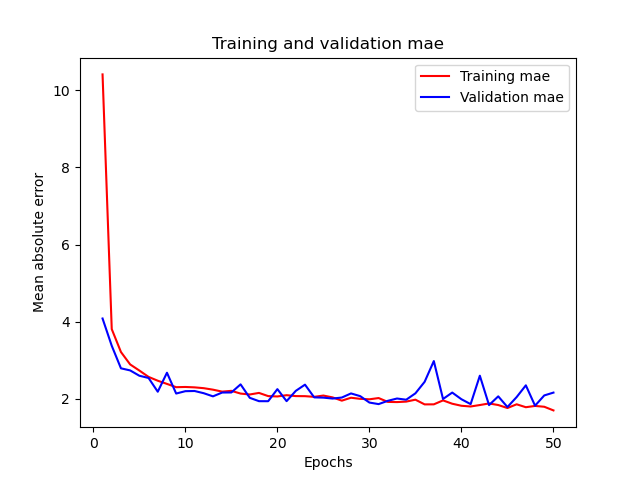


1

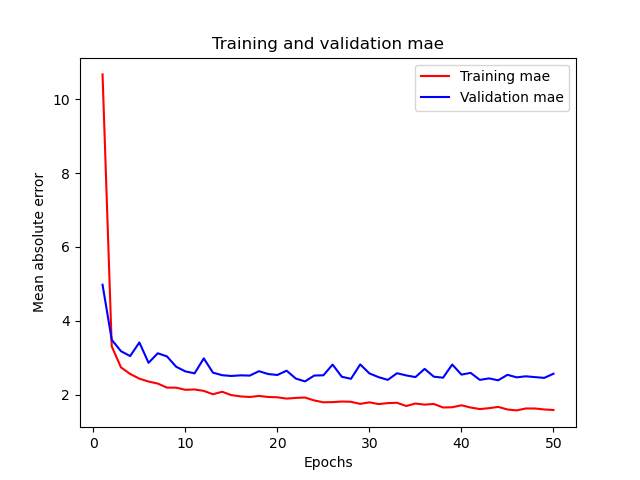
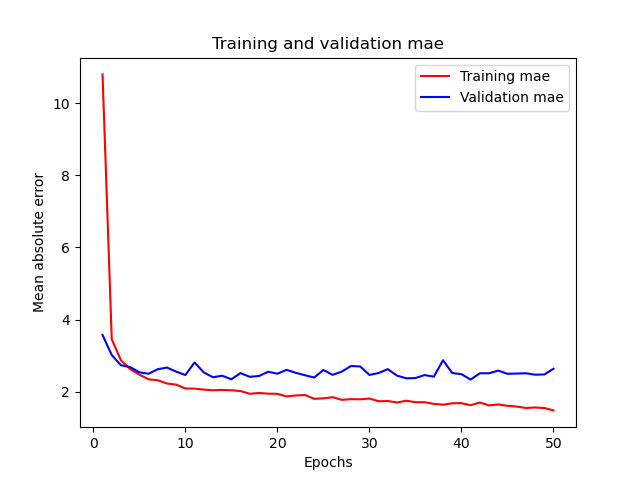
Рисунок 1 - Нахождение оптимального числа эпох

По рис. 1 видно, что ошибка на проверочных данных уменьшается до 40-50 эпох обучения, после она либо не меняется, либо становится больше при уменьшении ошибки на тестовых данных. Это говорит о переобучении модели, поэтому оптимальным значением числа эпох будет 50.

1. Было проведено тестирование обучения модели на изменяющемся числе блоков, на которые делятся данные. Значения числа блоков были взятыми 4, 6 и 8. Промежуточные результаты для 4 блоков представлены на рис. 2.



а) б)



в) г)

Рисунок 2 - Ошибка для блока а) 1, б) 2, в) 3, г) 4

График среднего значения ошибки для 4 блоков представлен на рис. 3.

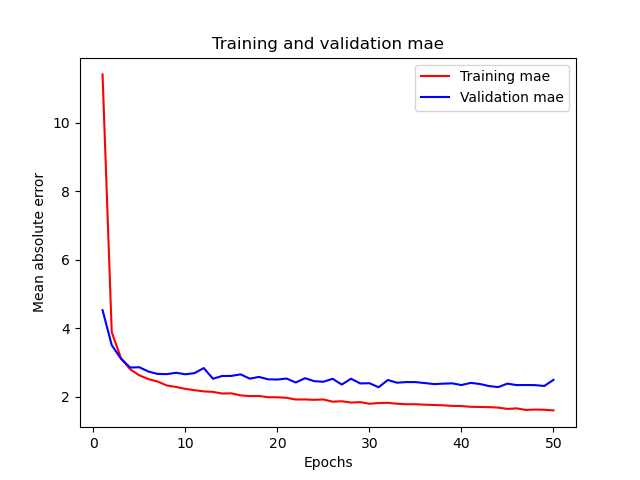
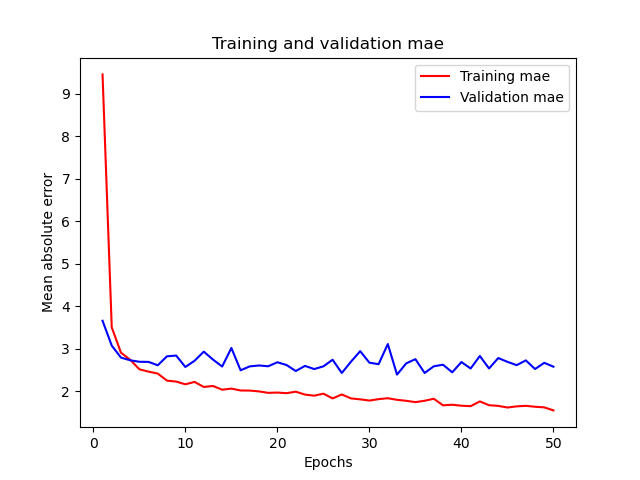
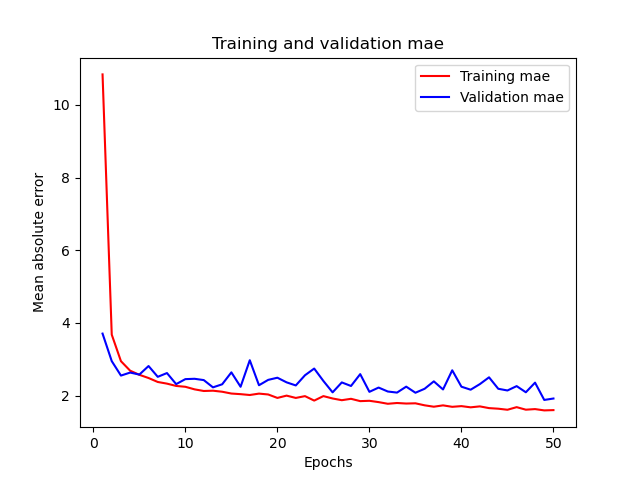
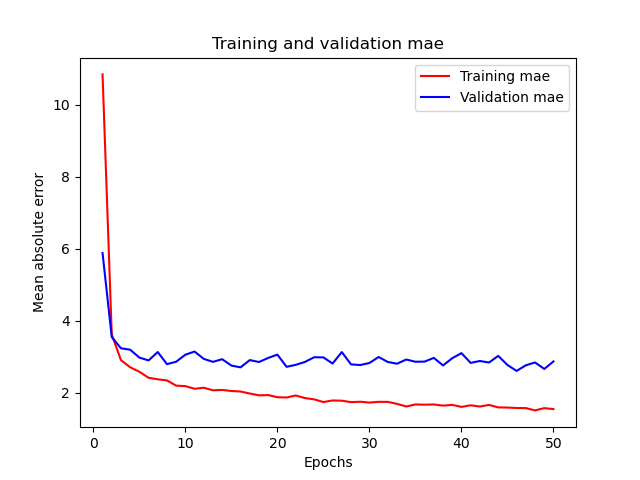
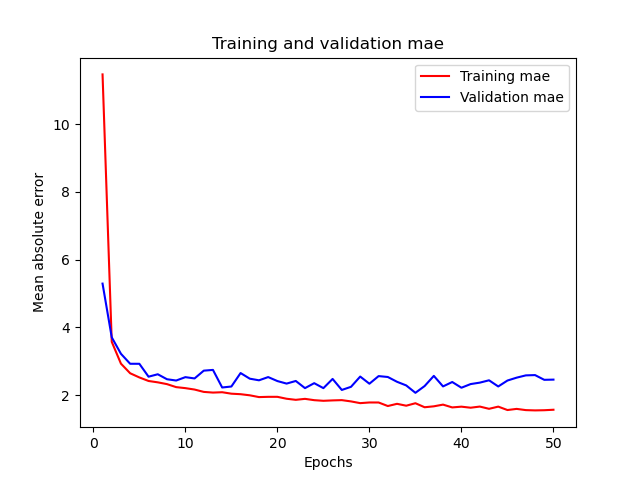


Рисунок 3 - Средняя ошибка для 4 блоков

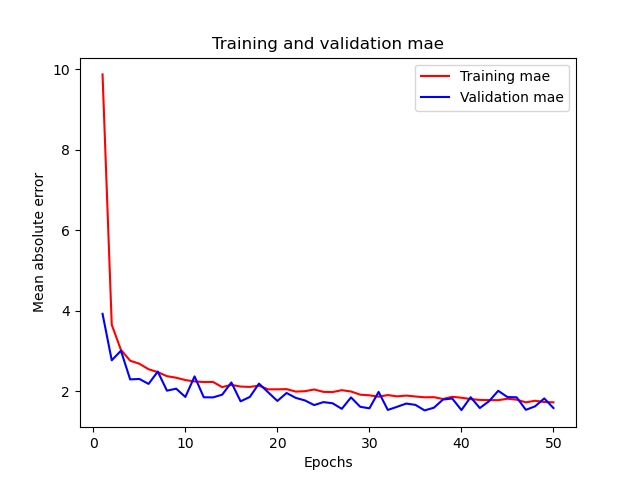
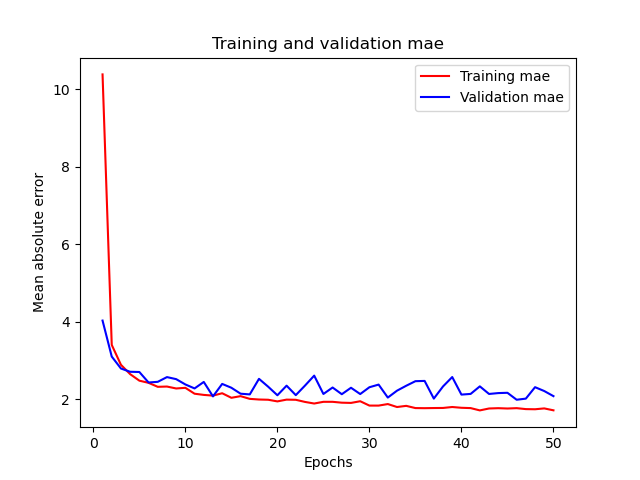
Промежуточные результаты для 6 блоков представлены на рис. 4.



а) б)



в) г)



д) е)

Рисунок 4 - Ошибка для блока а) 1, б) 2, в) 3, г) 4, д) 5, е) 6

График среднего значения ошибки для 4 блоков представлен на рис. 5.

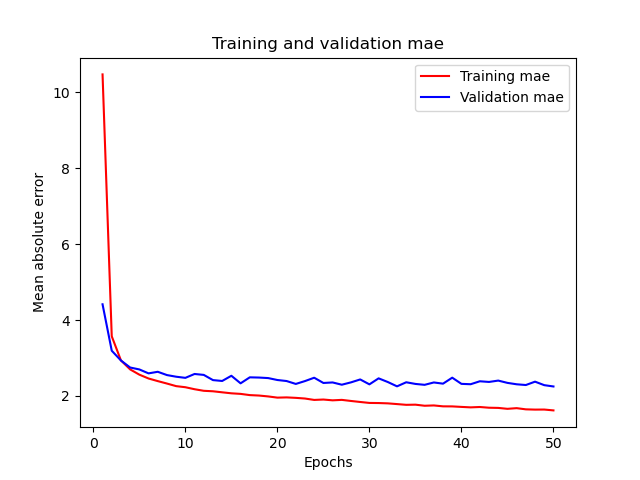
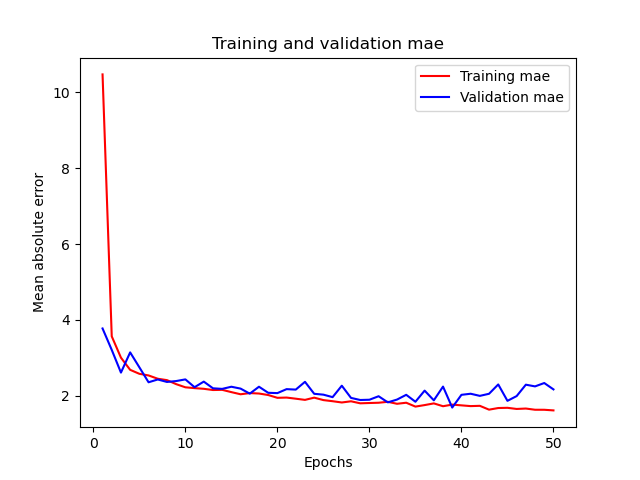
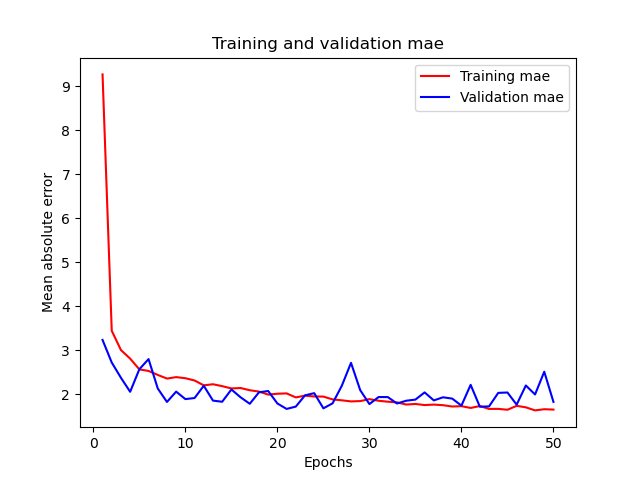
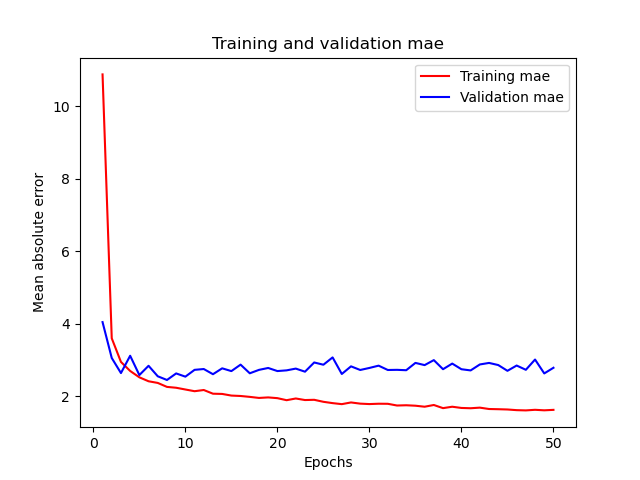
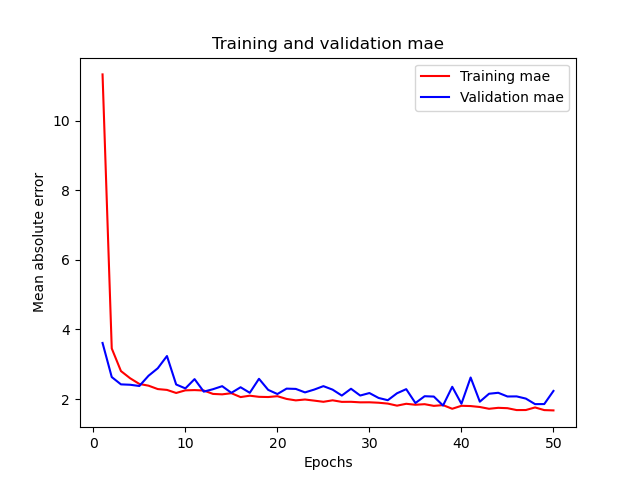


Рисунок 5 - Средняя ошибка для 6 блоков

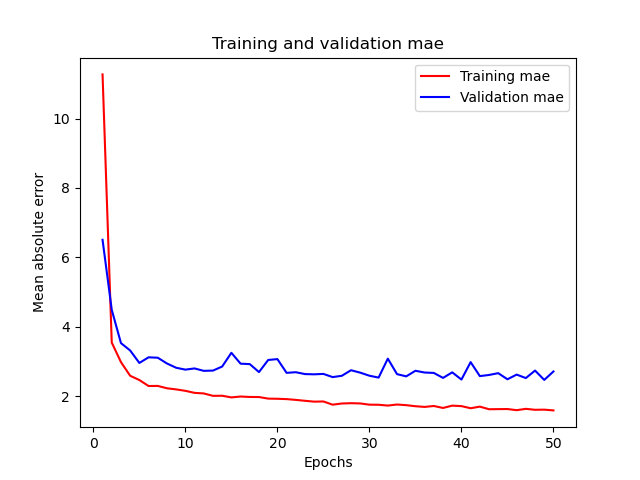
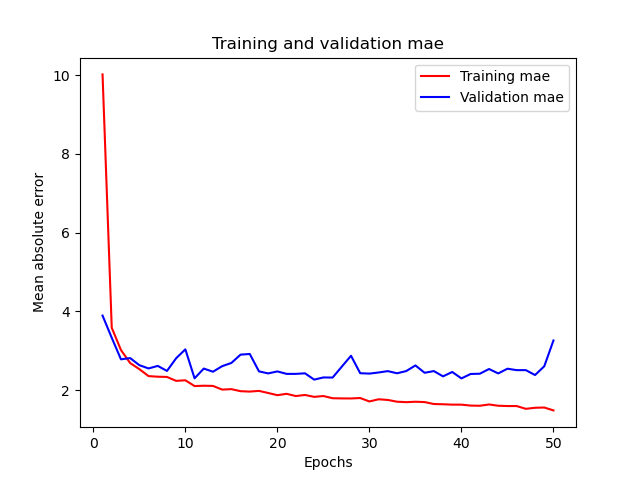
Промежуточные результаты для 8 блоков представлены на рис. 6.



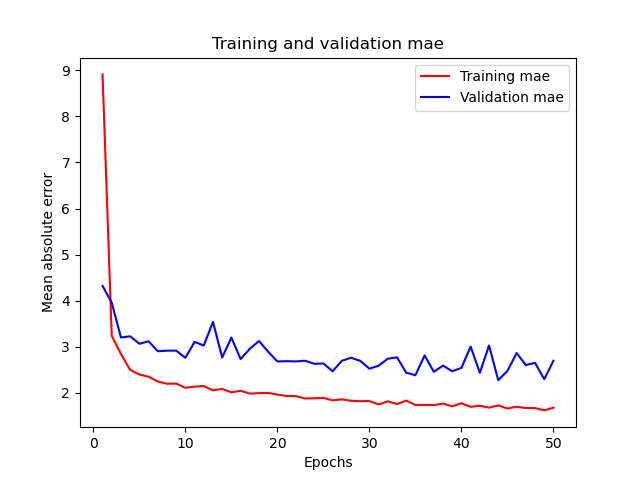
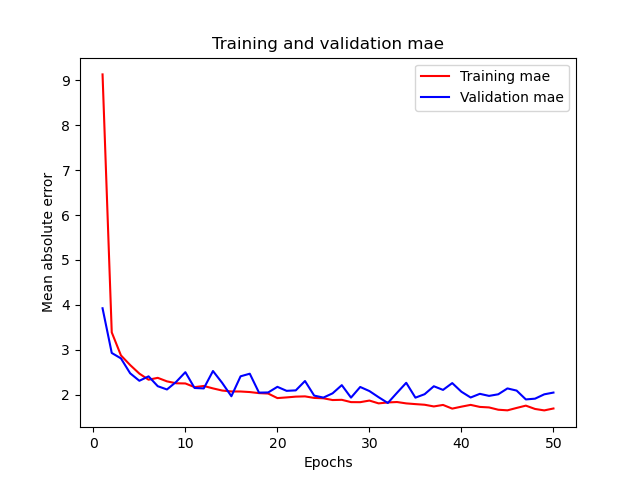
а) б)



в) г)



д) е)



ж) з)

Рисунок 6 - Ошибка для блока а) 1, б) 2, в) 3, г) 4, д) 5, е) 6, ж) 7, з) 8

График среднего значения ошибки для 4 блоков представлен на рис. 7.

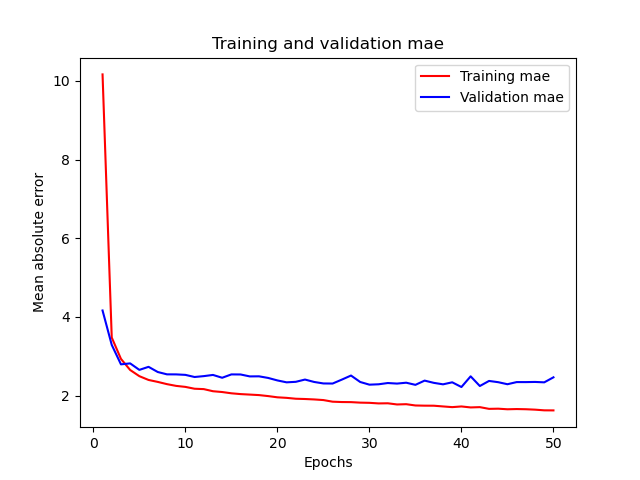


Рисунок 7 - Средняя ошибка для 8 блоков

По вышеприведенным графикам видно, что наименьшая ошибка наблюдается в модели, использующей 6 блоков.

**Вывод.**

В ходе выполнения данной лабораторной работы было изучено влияние количества эпох и количества блоков в перекрестной проверке на результат обучения модели искусственной нейронной сети, решающей задачу регрессии.

**ПРИЛОЖЕНИЕ А**

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()

print(train\_data.shape)

print(test\_data.shape)

print(test\_targets)

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

train\_data -= mean

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

train\_data /= std

test\_data -= mean

test\_data /= std

def draw\_test(arg, label, loss, val\_loss, acc, val\_acc):

# if loss != 0:

plt.plot(arg, loss, 'r', label='Training loss')

plt.plot(arg, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel(label)

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

# if acc != 0:

plt.plot(arg, acc, 'r', label='Training mae')

plt.plot(arg, val\_acc, 'b', label='Validation mae')

plt.title('Training and validation mae')

plt.xlabel(label)

plt.ylabel('Mean absolute error')

plt.legend()

plt.show()

def test\_epochs():

num\_epochs = 500

k = 4

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

loss\_array = []

val\_loss\_array = []

mae\_array = []

val\_mae\_array = []

for i in range(k):

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,

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

mae\_array.append(H.history['mae'])

val\_mae\_array.append(H.history['val\_mae'])

loss\_array.append(H.history['loss'])

val\_loss\_array.append(H.history['val\_loss'])

draw\_test(range(1, num\_epochs + 1), 'Epochs', np.mean(loss\_array, axis=1), np.mean(val\_loss\_array, axis=1),

np.mean(mae\_array, axis=0), np.mean(val\_mae\_array, axis=0))

print(np.mean(val\_mae\_array))

def test\_k():

num\_epochs = 50

for k in range(6, 9, 2):

build\_k\_block\_model(k, num\_epochs)

def build\_k\_block\_model(k, num\_epochs):

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

loss\_array = []

val\_loss\_array = []

mae\_array = []

val\_mae\_array = []

for i in range(k):

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,

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

mae\_array.append(H.history['mae'])

val\_mae\_array.append(H.history['val\_mae'])

loss\_array.append(H.history['loss'])

val\_loss\_array.append(H.history['val\_loss'])

draw\_test(range(1, num\_epochs + 1), 'Epochs', H.history['loss'], H.history['val\_loss'], H.history['mae'],

H.history['val\_mae'])

draw\_test(range(1, num\_epochs + 1), 'Epochs', np.mean(loss\_array, axis=0), np.mean(val\_loss\_array, axis=0),

np.mean(mae\_array, axis=0), np.mean(val\_mae\_array, axis=0))

print(np.mean(val\_mae\_array))

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

test\_epochs()

test\_k()