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. 8383 _____ Бессуднов Г. И.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2021

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.

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. Например, некоторые признаки являются пропорциями и имеют значения между 0 и 1, другие — между 1 и 12 и т. д.

Основные теоретические положения

Набор данных присутствует в составе Keras.

Листинг 1:

```
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

(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

print(train_data.shape)
print(test_data.shape)

print(test_targets)
```

404 обучающих и 102 контрольных образца, каждый с 13 числовыми признаками.

Цены в основном находятся в диапазоне от 10 000 до 50 000 долларов США.

Было бы проблематично передать в нейронную сеть значения, имеющие самые разные диапазоны. Сеть, конечно, сможет автоматически адаптироваться к таким разнородным данным, однако это усложнит обучение. На практике к таким данным принято применять нормализацию: для каждого признака во входных данных (столбца в матрице входных данных) из каждого значения

вычитается среднее по этому признаку, и разность делится на стандартное отклонение, в результате признак центрируется по нулевому значению и имеет стандартное отклонение, равное единице. Такую нормализацию легко выполнить с помощью Numpy.

Листинг 2:

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

Определим функцию build_model():

Листинг 3:

```
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
```

Сеть заканчивается одномерным слоем, не имеющим функции активации (это линейный слой). Это типичная конфигурация для скалярной регрессии (целью которой является предсказание одного значения на непрерывной числовой прямой). Применение функции активации могло бы ограничить диапазон выходных значений: например, если в последнем слое применить функцию активации sigmoid, сеть обучилась бы предсказывать только значения из диапазона между 0 и 1.

В данном случае, с линейным последним слоем, сеть способна предсказывать значения из любого диапазона.

Обратите внимание на то, что сеть компилируется с функцией потерь mse — mean squared error (среднеквадратичная ошибка), вычисляющей квадрат разности между предсказанными и целевыми значениями. Эта функция широко

используется в задачах регрессии. Также добавлен новый параметр на этапе обучения: mae — mean absolute error (средняя абсолютная ошибка). Это абсолютное значение разности между предсказанными и целевыми значениями. Например, значение MAE, равное 0,5, в этой задаче означает, что в среднем прогнозы отклоняются на 500 долларов США.

Чтобы оценить качество сети в ходе корректировки ее параметров (таких, как количество эпох обучения), можно разбить исходные данные на обучающий и проверочный наборы, как это делалось в предыдущих примерах. Однако так как у нас и без того небольшой набор данных, проверочный набор получился бы слишком маленьким (скажем, что-нибудь около 100 образцов). Как следствие, оценки при проверке могут сильно меняться в зависимости от того, какие данные попадут в проверочный и обучающий наборы: оценки при проверке могут иметь слишком большой разброс. Это не позволит надежно оценить качество модели.

Хорошей практикой в таких ситуациях является применение перекрестной проверки по K блокам (K-fold cross-validation). Суть ее заключается в разделении доступных данных на K блоков (обычно K = 4 или 5), создании K идентичных моделей и обучении каждой на K—1 блоках с оценкой по оставшимся блокам. По полученным K оценкам вычисляется среднее значение, которое принимается как оценка модели. В коде такая проверка реализуется достаточно просто.

Листинг 4:

```
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
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()
    model.fit(partial_train_data, partial_train_targets, epochs=num_epochs,
batch_size=1, verbose=0)
```

```
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)

print(np.mean(all_scores))
```

Выполнение работы

По теоретическим сведениям были построены k = 4 модели, на которых проводились эксперименты. Для начала было исследование влияние количества эпох на точность модели. Графики средней абсолютной ошибки и функции потерь всех моделей и усредненных значений для 100 эпох приведены на рис. 1, рис. 2. Графики для 75 эпох приведены на рис. 3, рис. 4. Графики для 200 эпох — на рис. 5, рис 6.

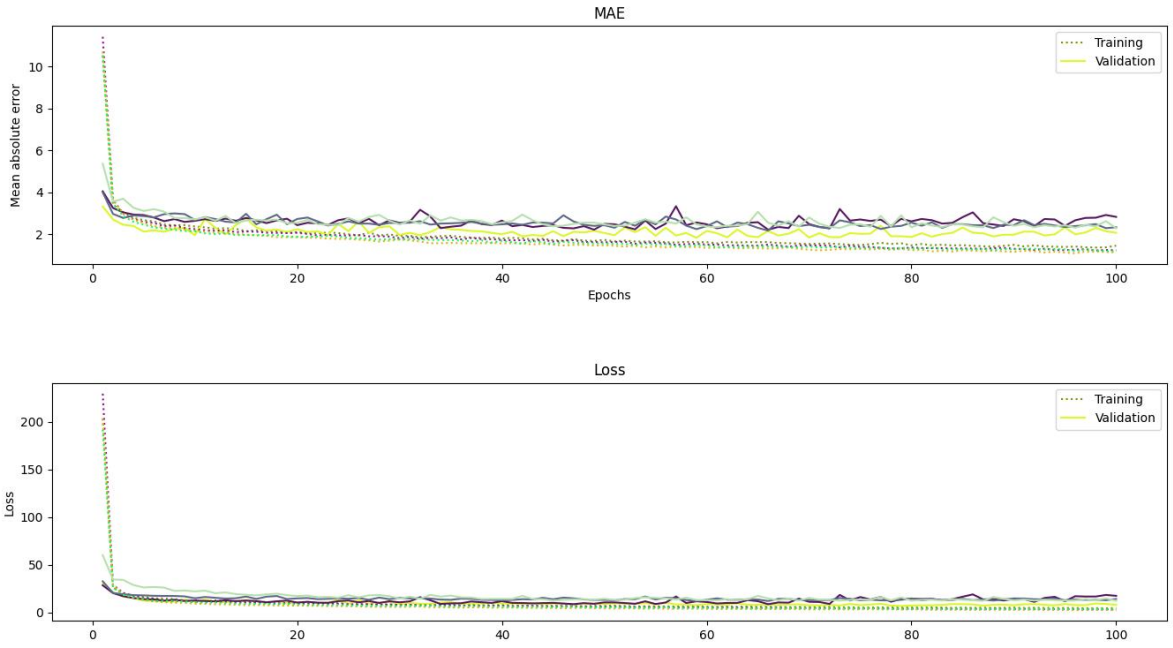

Рисунок 1: Результаты для всех моделей при 100 эпохах

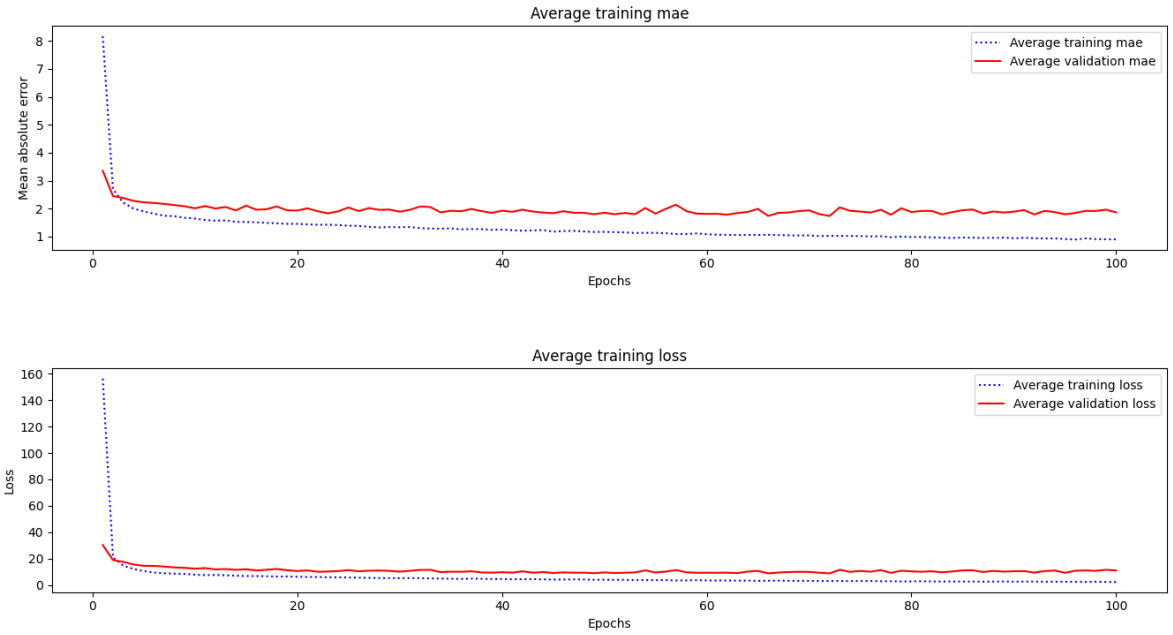

Рисунок 2: Результаты общие при 100 эпохах

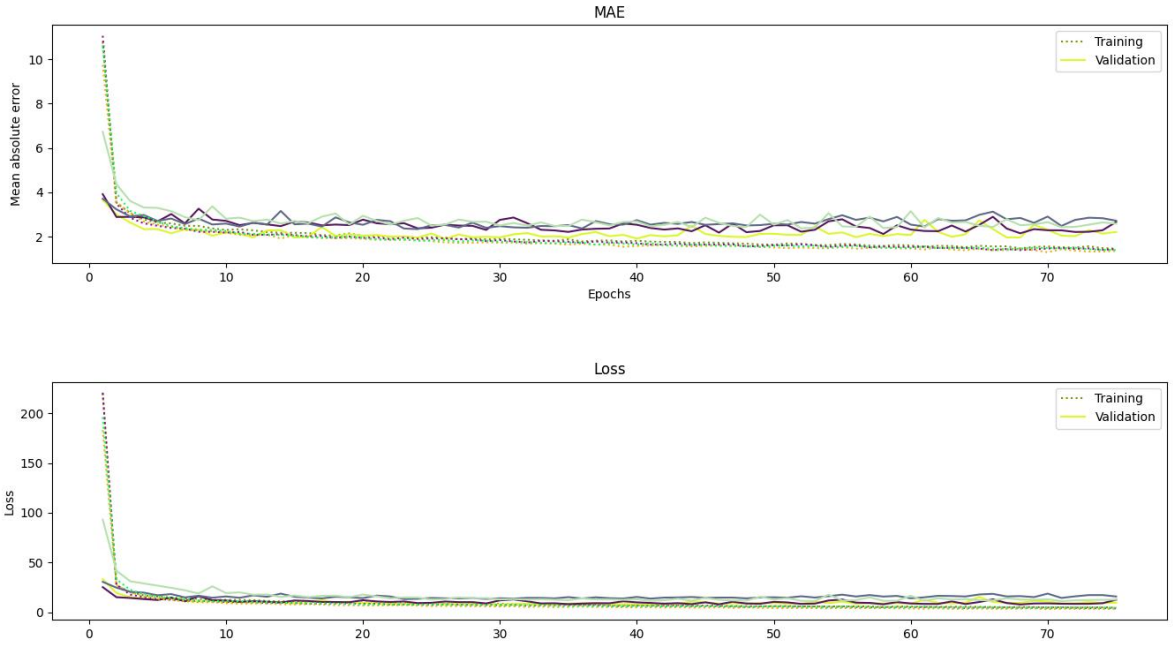

Рисунок 3: Результаты для всех моделей при 75 эпохах

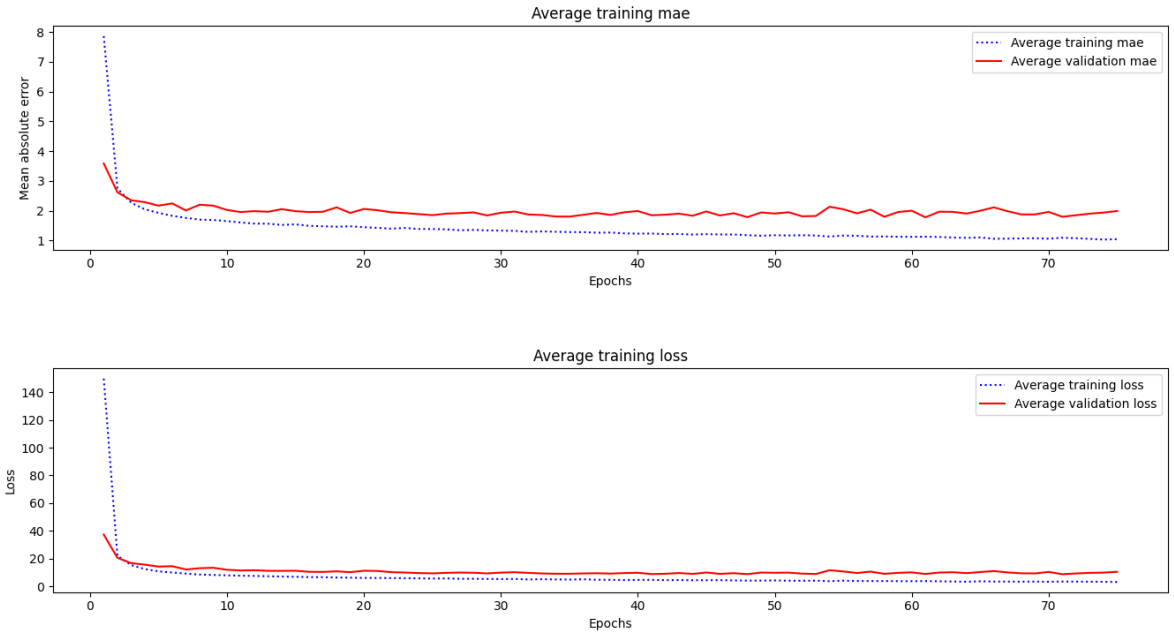

Рисунок 4: Результаты общие при 75 эпохах

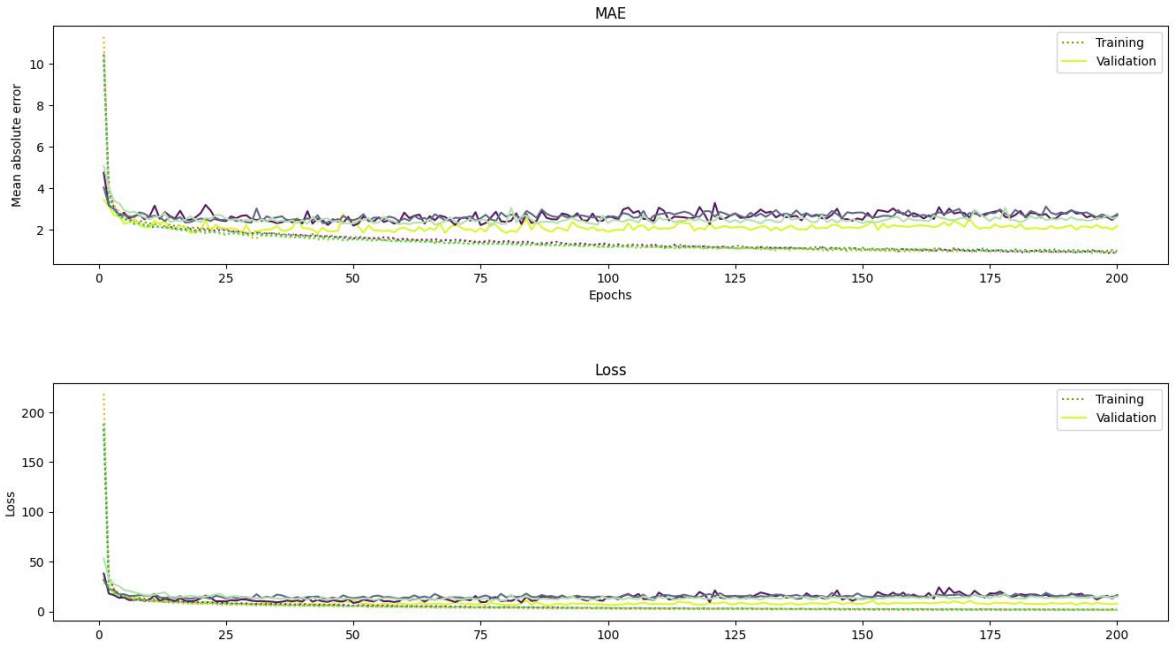

Рисунок 5: Результаты для всех моделей при 200 эпохах

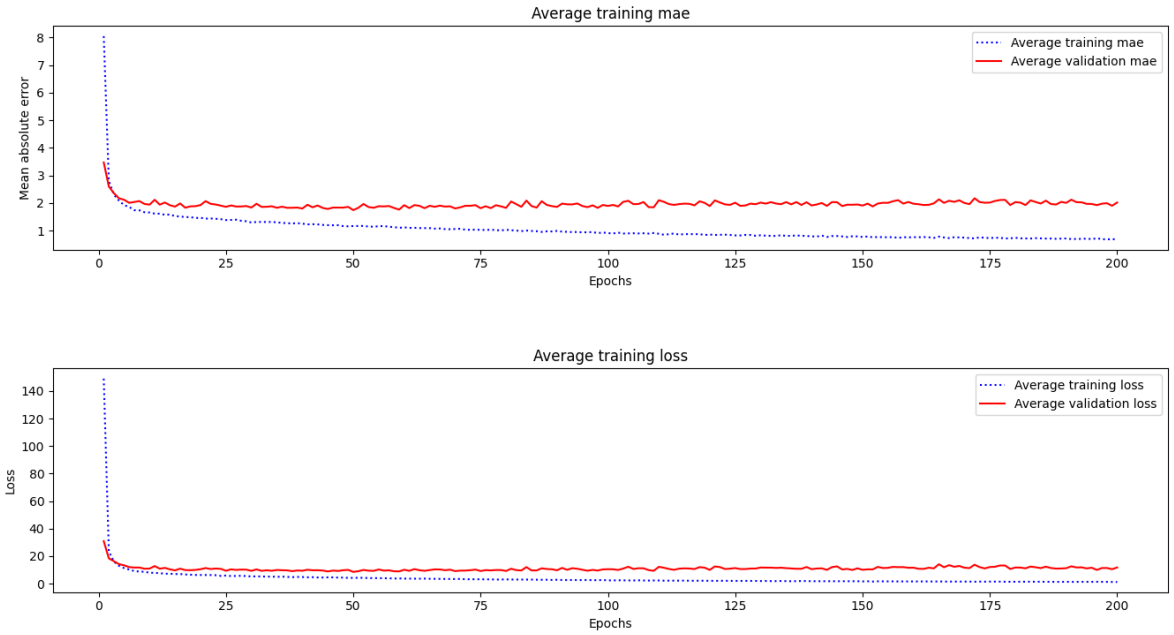

Рисунок 6: Результаты общие при 200 эпохах

Результаты были следующими: для количества эпох 75 в среднем mae = 2.336, для количества эпох 100 в среднем mae = 2.383, для количества эпох 200 в среднем mae = 2.563. Таким образом можно сделать вывод о том, что с увеличением количества эпох растет и ошибка. Это может говорить о том, что точка переобучения находиться где-то до 75 эпохи.

Далее была проведена попытка найти точку переобучения. Для этого необходимо было найти точку, где средняя абсолютная ошибка минимальна. Результаты представлены на рис. 7, минимальная точка выделена зеленым.

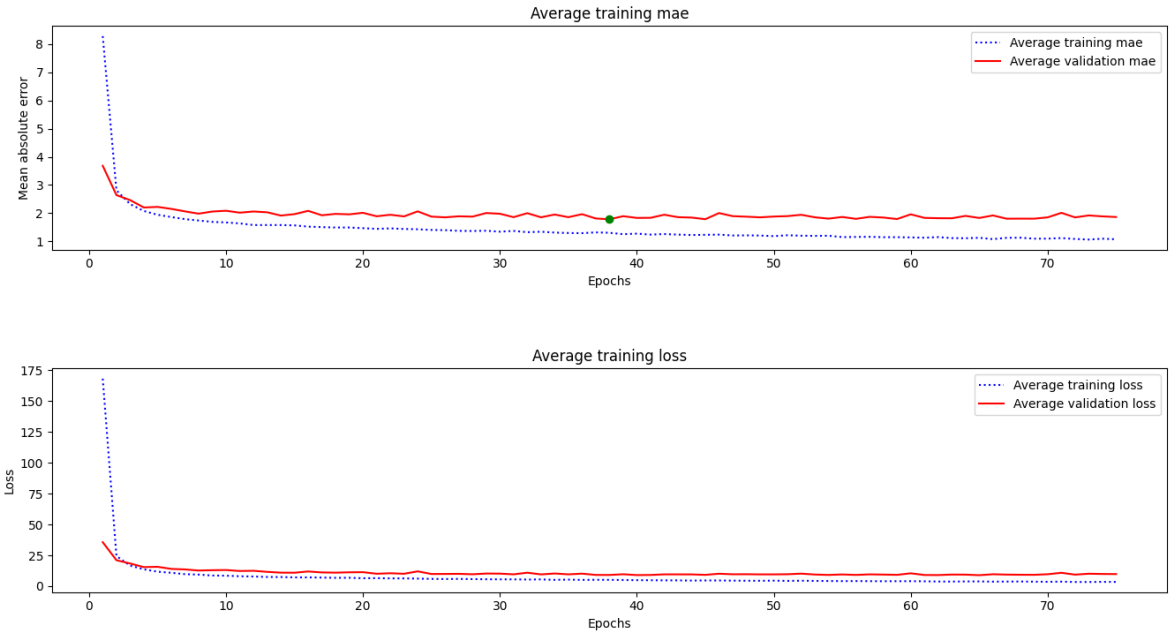

Рисунок 7: Точка с минимальным значением mae

Точка находится на 38 эпохе. Значение mae = 1.79. Следовательно достаточно делать 35-45 эпох для получения оптимального результат.

Далее было проведено исследование зависимости поведения модели от параметра k на 75 эпохах. Графики для k = 2 представлены на рис. 8, рис. 9. Графики для k = 12 представлены на рис. 10, рис. 11.

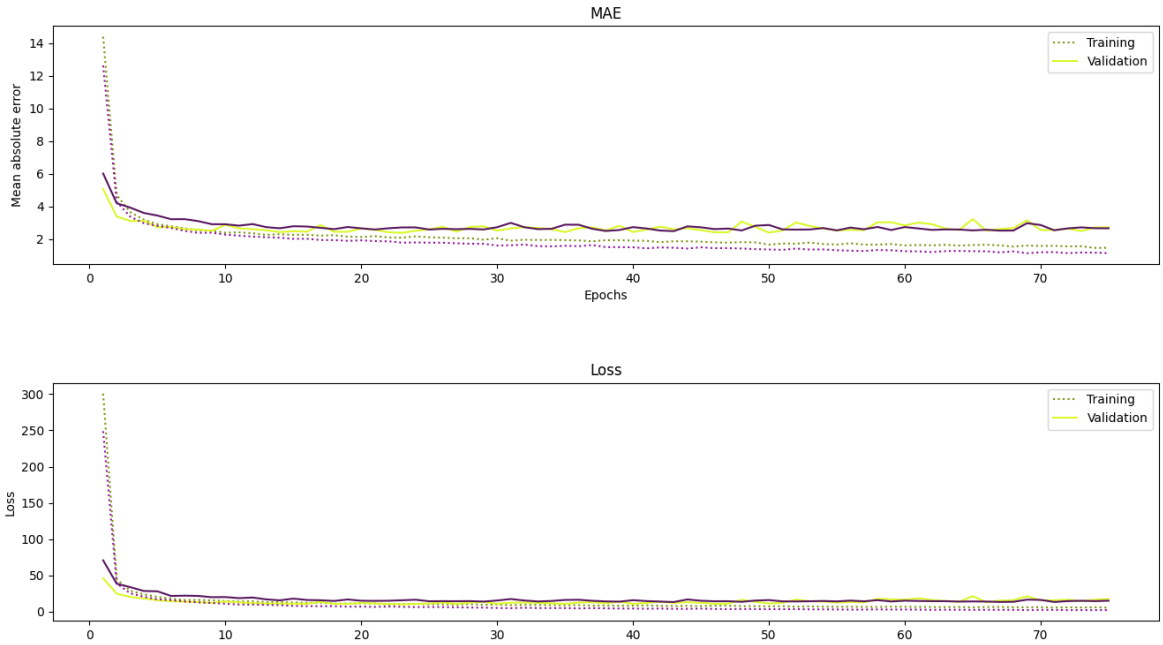

Рисунок 8: Результаты для всех моделей при K=2

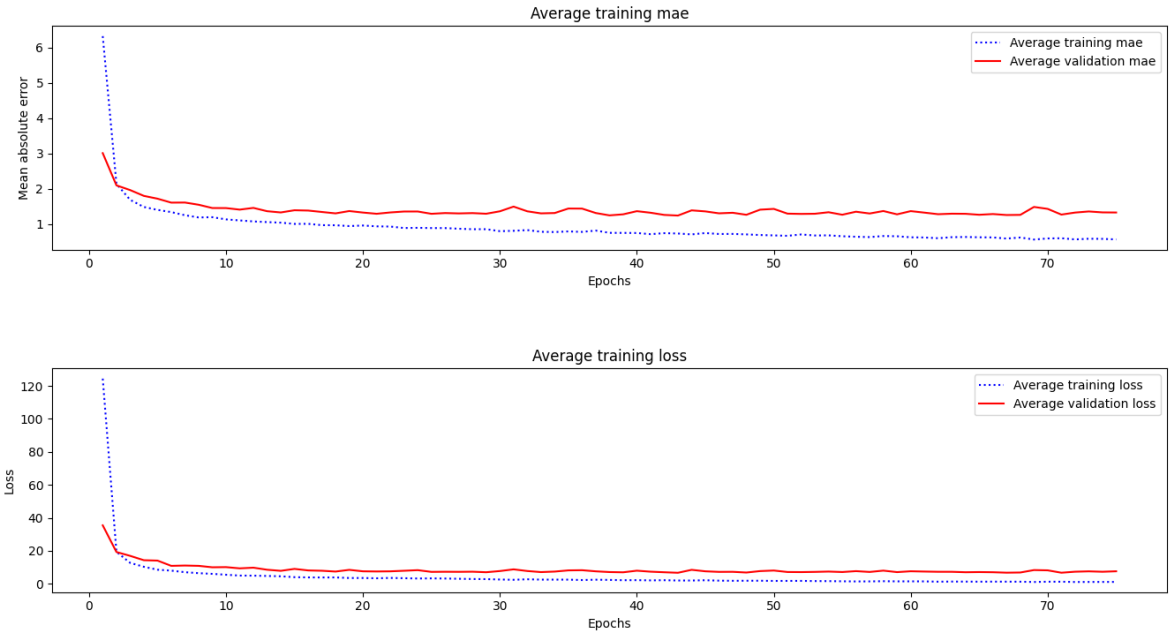

Рисунок 9: Результаты общие при K=2

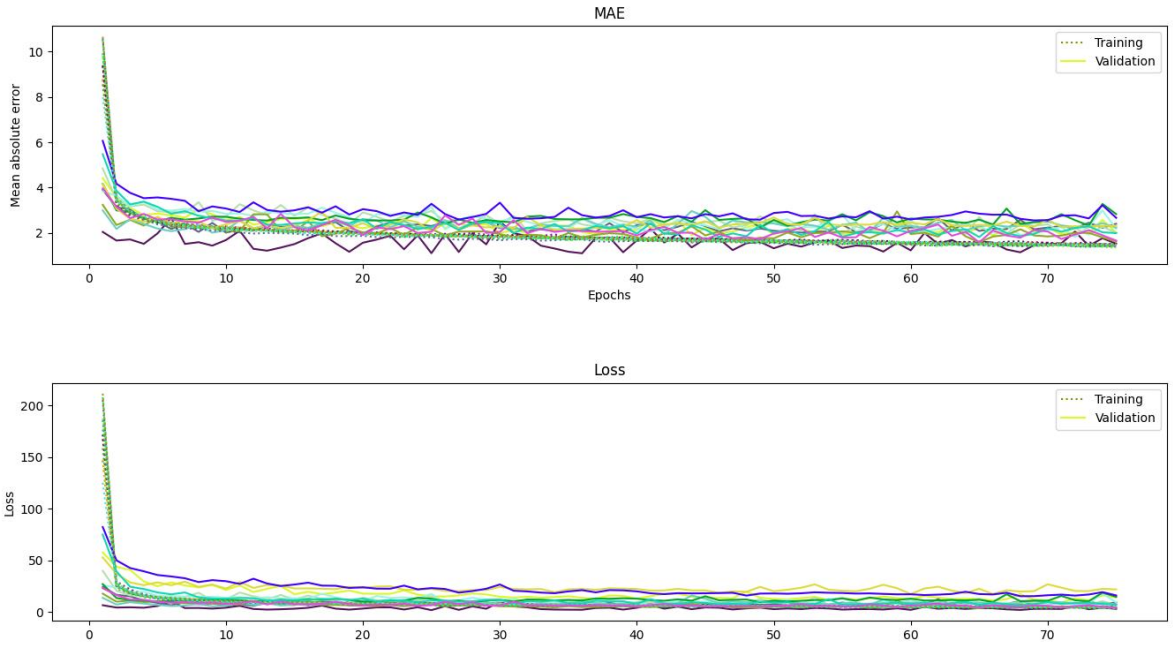

Рисунок 10: Результаты для всех моделей при K=12

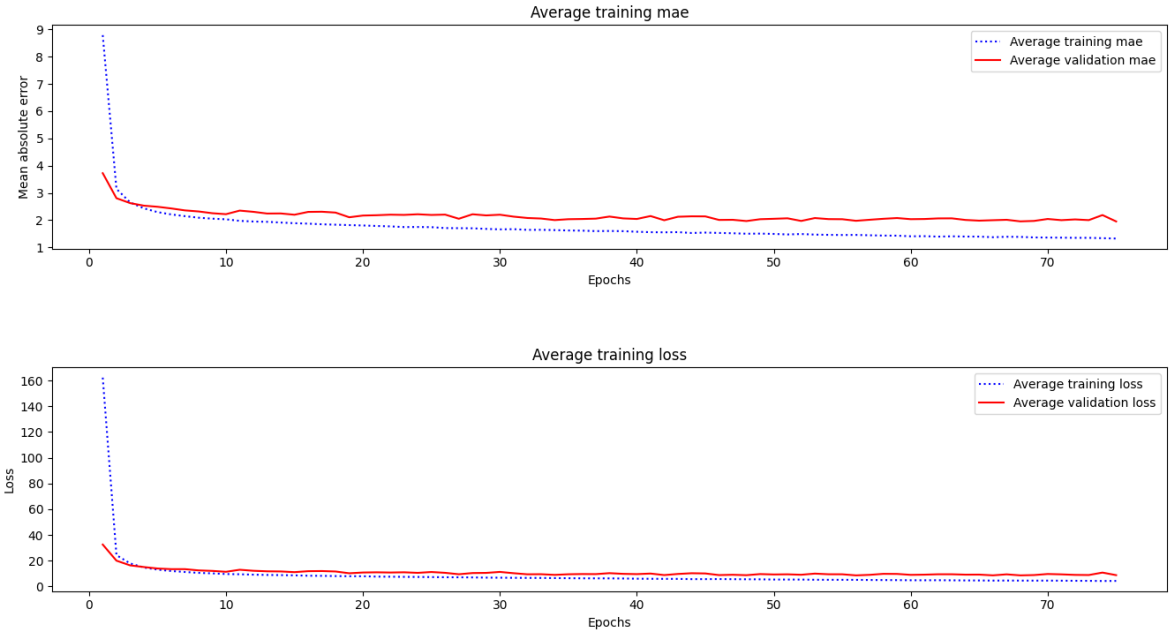

Рисунок 11: Результаты общие при K=12

Результаты были следующими: для k = 2 в среднем mae = 2.686, для k = 12 в среднем mae = 2.122. Результат из предыдущего эксперимента, где k = 4 в среднем показал что mae = 2.336.

Видно, что при слишком малом значении k модель ведет себя хуже, тогда как при большем значение результат модели лучше. При слишком большом значении k размер тестовых данных значительно уменьшается, из-за чего модель будет давать менее объективный результат, так как объем тестовый выборки мал. При слишком малом значении k данный метод не будет решать проблему, для которой он создавался, а именно усреднение оценки нескольких моделей и получение более качественного результата на малых объемах данных.

В данной работе решалась задача регрессии. Задача регрессии подразумевает под собой количественное определение одной переменной от других факторов. Задача классификации заключается в качественном определении одной переменной от другой.

Выводы

Была реализована искусственная нейронная сеть для предсказание медианной цены на дома в пригороде Бостона. В ходе работы был изучен метод разделения по k блокам, была выявлена точка переобучения модели, исследовано влияние количества эпох и блоков на поведение модели. Были построены необходимые графики и проведены их исселдования.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
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


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


def plot_mae(history):
    for i in range(len(history)):
        hist_mae = history[i].history["mae"]
        hist_mae_val = history[i].history["val_mae"]
        hist_loss = history[i].history["loss"]
        hist_loss_val = history[i].history["val_loss"]
        epochs_vals = range(1, len(hist_mae) + 1)
        color1 = np.random.uniform(0, 1, 3)
        color2 = np.random.uniform(0, 1, 3)
        plt.subplot(2, 1, 1)
        plt.plot(epochs_vals, hist_mae, color=color1, linestyle="dotted")
        plt.plot(epochs_vals, hist_mae_val, color=color2)
        plt.subplot(2, 1, 2)
        plt.plot(epochs_vals, hist_loss, color=color1, linestyle="dotted")
        plt.plot(epochs_vals, hist_loss_val, color=color2)
    plt.subplot(2, 1, 1)
    plt.title("MAE")
    plt.xlabel("Epochs")
```

```
    plt.ylabel("Mean absolute error")
    plt.legend(("Training", "Validation"))
    plt.subplot(2, 1, 2)
    plt.legend(("Training", "Validation"))
    plt.title("Loss")
    plt.ylabel("Loss")

    plt.tight_layout()
    plt.show()


def plot_avg_mae(history):
    avg_mae = np.zeros(num_epochs)
    avg_validation_mae = np.zeros(num_epochs)
    avg_loss = np.zeros(num_epochs)
    avg_validation_loss = np.zeros(num_epochs)
    for i in range(1, len(history)):
        avg_mae += history[i].history["mae"]
        avg_validation_mae += history[i].history["val_mae"]
        avg_loss += history[i].history["loss"]
        avg_validation_loss += history[i].history["val_loss"]
    avg_mae /= len(history)
    avg_validation_mae /= len(history)
    avg_loss /= len(history)
    avg_validation_loss /= len(history)
    epochs = range(1, num_epochs + 1)

    # min_mae_index = np.argmin(avg_validation_mae)

    plt.subplot(2, 1, 1)
    plt.plot(epochs, avg_mae, color="b", label="Average training mae",
linestyle="dotted")
    plt.plot(epochs, avg_validation_mae, color="r", label="Average validation
mae")
    # plt.plot(epochs[min_mae_index], avg_validation_mae[min_mae_index], "go")
    plt.title("Average training mae")
    plt.xlabel("Epochs")
```

```
    plt.ylabel("Mean absolute error")
    plt.legend()
    plt.subplot(2, 1, 2)
    plt.plot(epochs, avg_loss, color="b", label="Average training loss",
linestyle="dotted")
    plt.plot(epochs, avg_validation_loss, color="r", label="Average validation
loss")
    plt.title("Average training loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()

    plt.tight_layout()
    plt.show()

(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 = 12
num_val_samples = len(train_data) // k
num_epochs = 75
all_scores = []
all_history = []

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()
    history = model.fit(partial_train_data, partial_train_targets,
epochs=num_epochs, batch_size=1,
                        validation_data=(val_data, val_targets), verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
    all_history.append(history)

print(np.mean(all_scores))
plot_mae(all_history)
plot_avg_mae(all_history)
```