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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Искусственные нейронные сети»

Тема: Регрессионная модель изменения цен на дома в Бостоне

Студент гр.8382 _____ Терехов А.Е.

Преподаватель _____ Жангриров 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.

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

Требования

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

Набор данных присутствует в составе Keras.

Листинг 1 – Подключение модулей

```
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 10000 до 50000 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 – Обработка данных

```
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 – Обработка данных

```
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 – Обработка данных

```
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))
```

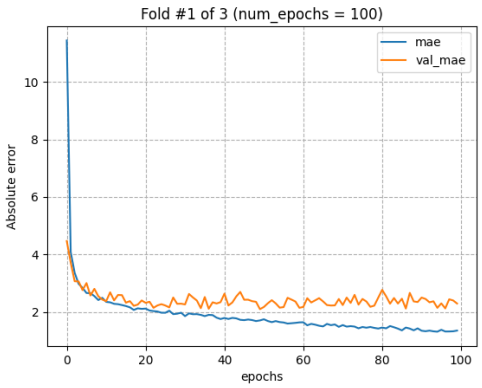
Разные прогоны действительно показывают разные оценки, от 2,6 до 3,2. Средняя (3,0) выглядит более достоверно, чем любая из оценок отдельных прогонов, — в этом главная ценность перекрестной проверки по K блокам. В данном случае средняя ошибка составила 3000 долларов, что довольно много, если вспомнить, что цены колеблются в диапазоне от 10000 до 50000 долларов.

Необходимо уменьшить или увеличить количество эпох обучения и проанализировать полученные результаты.

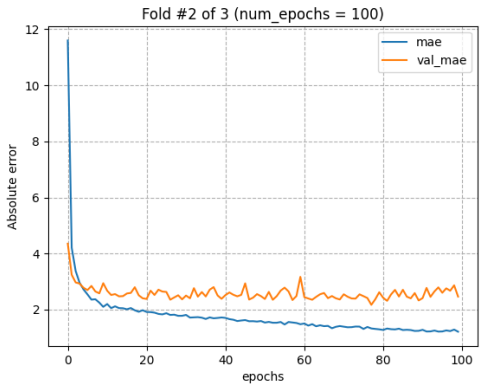
Ход работы

В работе производилась серия запусков с разным количеством блоков для кросс-валидации.

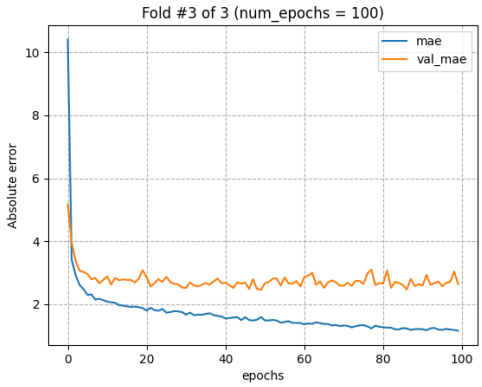
Графики оценки средних абсолютных ошибок при количестве блоков – 3 представлены на рисунке 1.



(a) Первый блок



(b) Второй блок



(c) Третий блок

Рис. 1 – Графики абсолютных ошибок при количестве блоков равном 3

График средних значений ошибок по трем блокам представлен на рисунке 2.

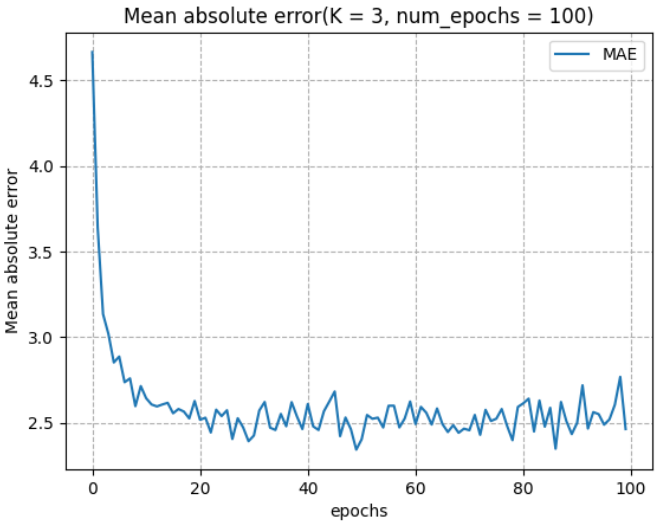
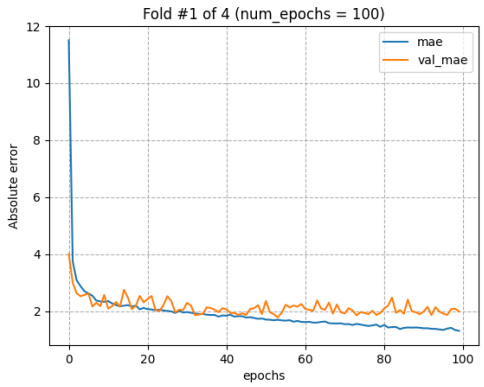


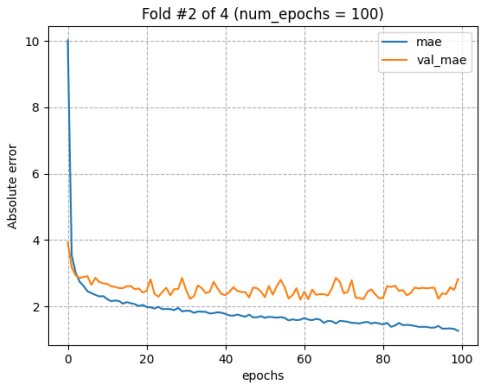
Рис. 2 – Средние значения ошибок при разбиении на 3 блока

Проанализировав графики можно сделать вывод, что средняя ошибка составила $2500.

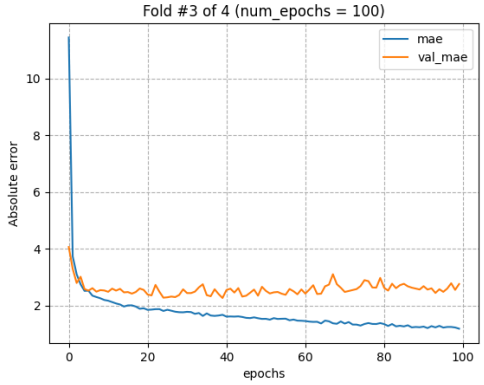
Увеличим количество блоков до 4. Полученные графики представлены на рисунке 3.



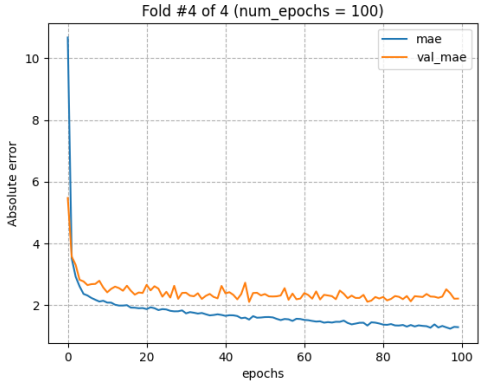
(a) Первый блок



(b) Второй блок



(c) Третий блок



(d) Четвертый блок

Рис. 3 – Графики абсолютных ошибок при количестве блоков равном 4

График средних значений ошибок по трем блокам представлен на рисунке 4.

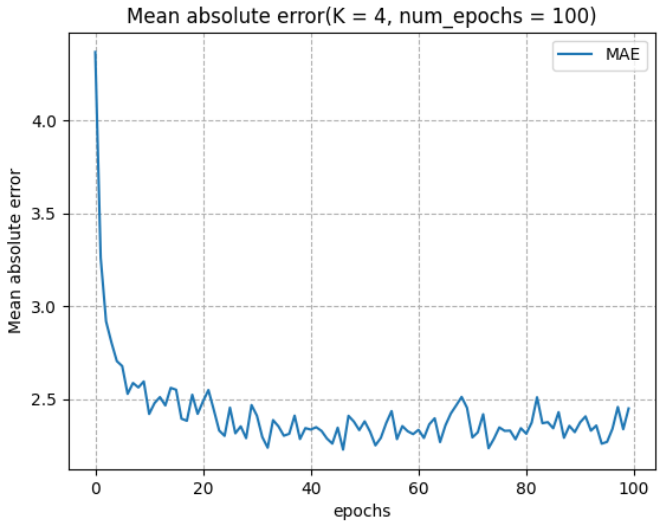
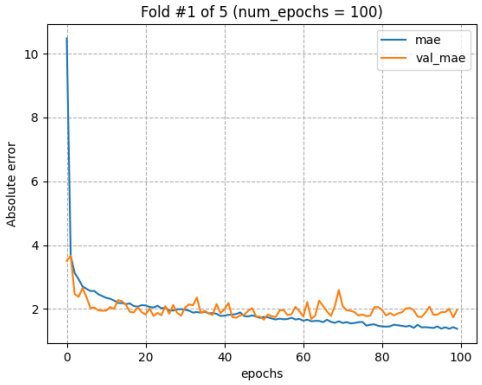


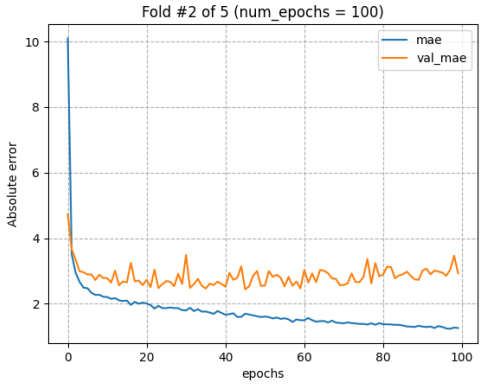
Рис. 4 – Средние значения ошибок при разбиении на 4 блока

Как видно на графике сеть смогла снизить оценку ошибки до значения меньше чем 2.5. На эпохах 20-60 было получено наименьшее значение ошибки. Но затем график начал понемногу расти, что говорит о переобучении модели.

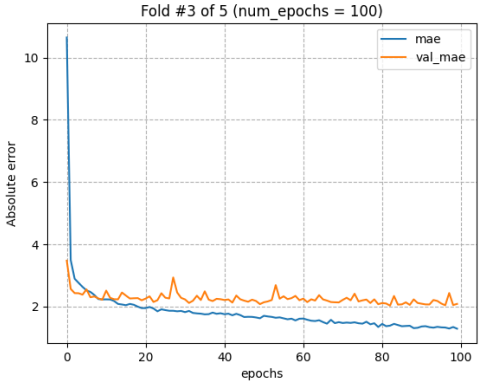
Протестируем сеть разбив данные на 5 фолдов. Графики при таком разбиении изображены на рисунке 5.



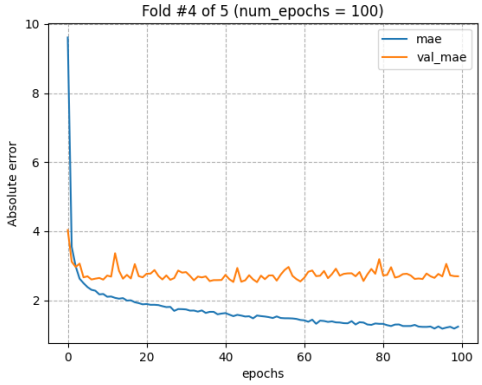
(a) Первый блок



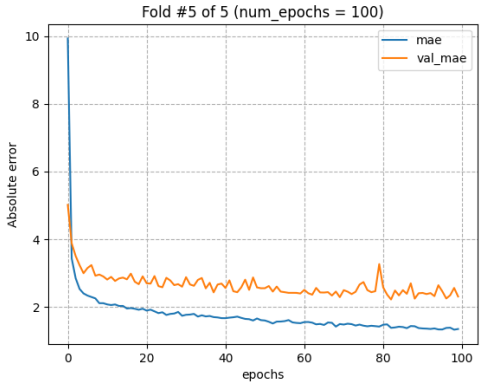
(b) Второй блок



(c) Третий блок



(d) Четвертый блок



(e) Пятый блок

Рис. 5 – Графики абсолютных ошибок при количестве блоков равном 5

График средних значений ошибок по трем блокам представлен на рисунке 6.

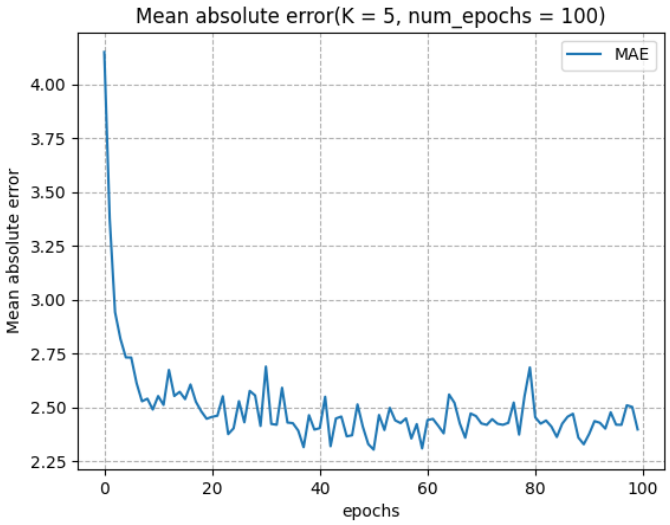
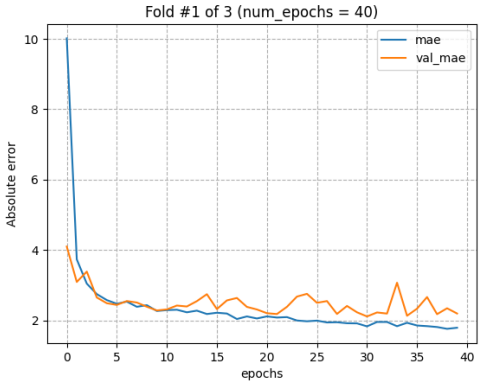


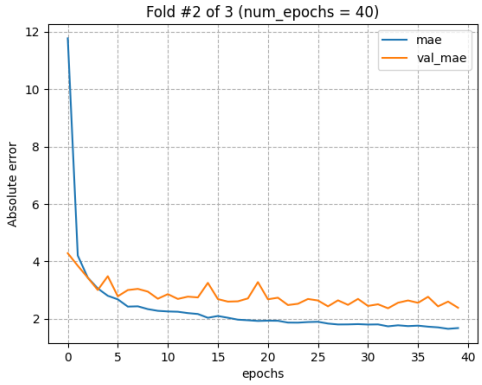
Рис. 6 – Средние значения ошибок при разбиении на 5 блоков

В этом случае, аналогично предыдущему, модель перестает уменьшать оценку ошибки после 40 эпохи. Поэтому проведем еще одну серию запусков, уменьшив количество эпох до 40.

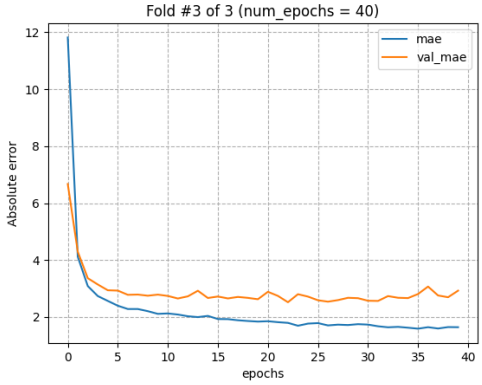
Графики оценки средних абсолютных ошибок представлены на рисунках 7-12.



(a) Первый блок



(b) Второй блок



(c) Третий блок

Рис. 7 – Графики абсолютных ошибок при количестве блоков равном 3

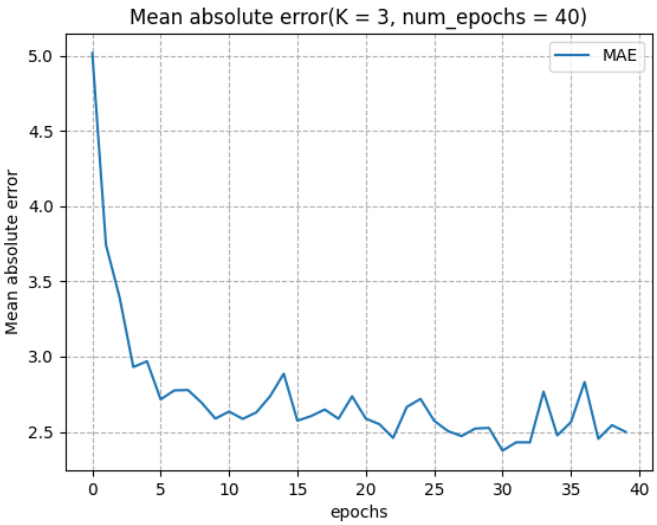
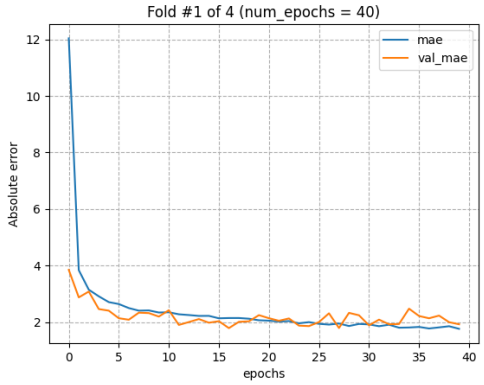
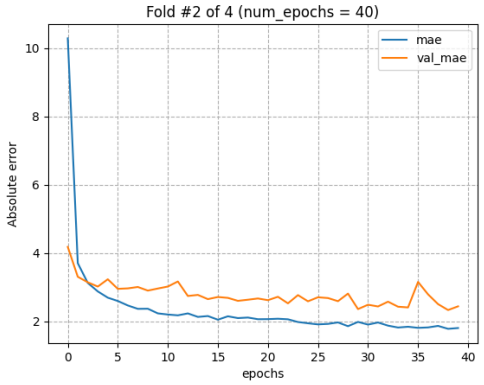


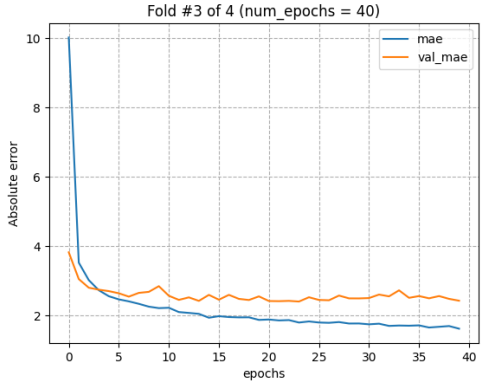
Рис. 8 – Средние значения ошибок при разбиении на 3 блока



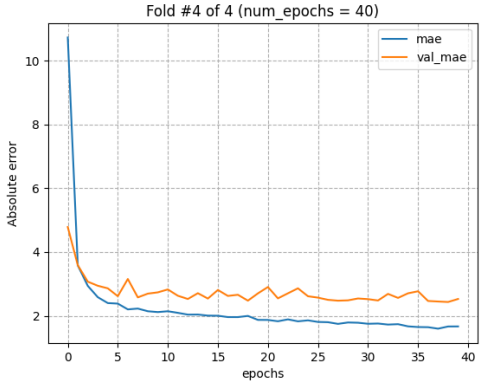
(a) Первый блок



(b) Второй блок



(c) Третий блок



(d) Четвертый блок

Рис. 9 – Графики абсолютных ошибок при количестве блоков равном 4

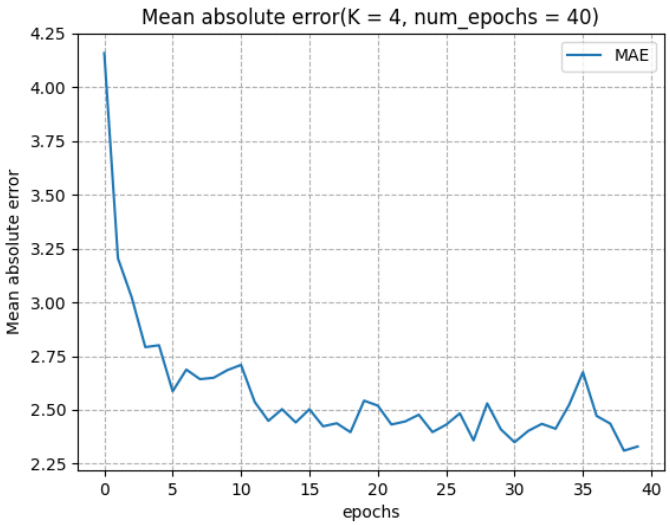
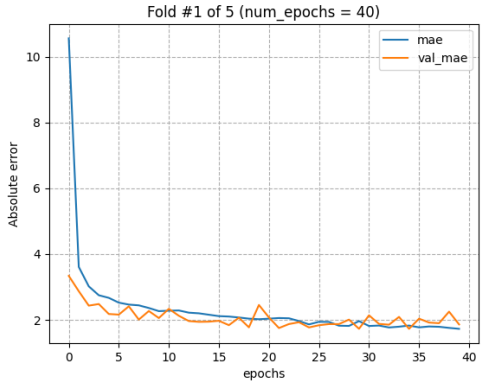
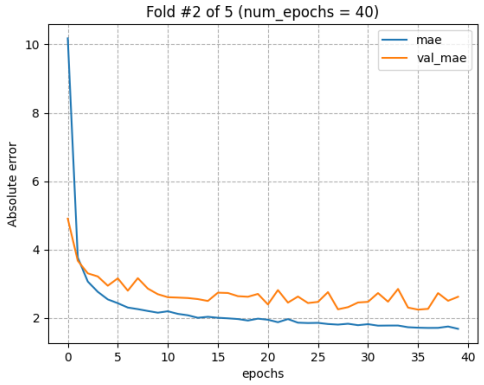


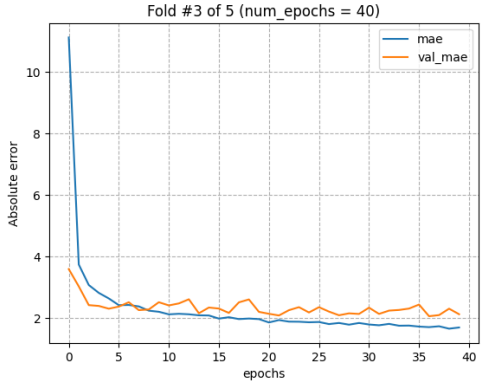
Рис. 10 – Средние значения ошибок при разбиении на 4 блока



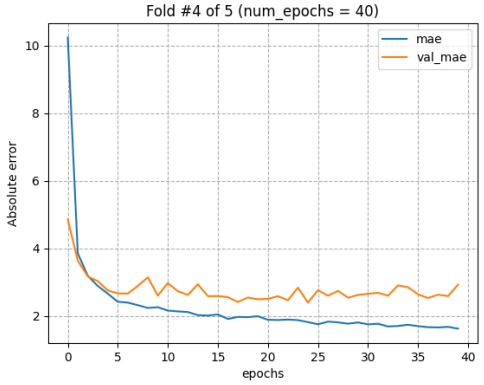
(a) Первый блок



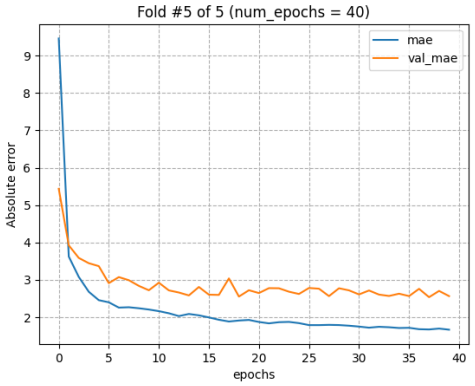
(b) Второй блок



(c) Третий блок



(d) Четвертый блок



(e) Пятый блок

Рис. 11 – Графики абсолютных ошибок при количестве блоков равном 5

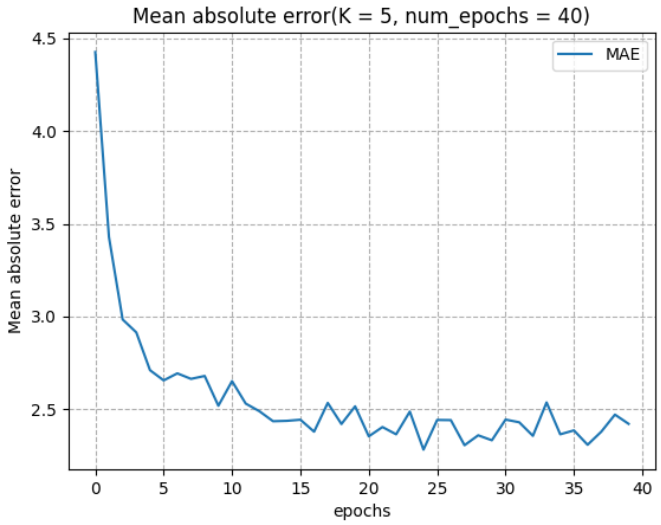


Рис. 12 – Средние значения ошибок при разбиении на 5 блоков

Исходя из полученных графиков, можно говорить о том, что уменьшение числа эпох было оправданно. Если при 100 эпохах график ошибок колебался около значения 2.5, то при 40 эпохах с увеличением количества блоков все реже достигает этого значения. Но все равно сеть, обучающаяся на 3 блоках кросс-валидации, показала достаточно плохие результаты. Модели оказалось недостаточно данных для обучения. Сеть же, обучающаяся на 5 блоках, смогла уменьшить ошибки до значения ~2.3.

Вывод

В ходе лабораторной работы была обучена одна модель, решающая задачу регрессии. Для упрощения задачи входные данные были нормализованы. Для оценки качества модели была применена кросс-валидация с разным количеством блоков. Так же была выявлена склонность модели к переобучению при 100 эпохах, поэтому их количество было сокращено до 40. И при 40 эпохах и разбиении на 5 блоков сеть была обучена наилучшим образом, снизив ошибку до $2300.

ПРИЛОЖЕНИЕ А

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from tensorflow.keras.datasets import boston_housing
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential


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 crossval_cycle(k, num_epochs):
    num_val_samples = len(train_data) // k
    all_scores = []
    for i in range(k):
        print(f'processing fold #{i + 1}')
        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()
            fitted = model.fit(partial_train_data,
                partial_train_targets, epochs=num_epochs, batch_size
                =1,
                                validation_data=(val_data, val_targets
                                    ), verbose=0)
            mae = pd.DataFrame(fitted.history)[['mae', 'val_mae']]
            all_scores.append(mae['val_mae'])
            plot(mae, "Absolute error", f"Fold #{i + 1} of {k} (
                num_epochs = {num_epochs})")
        mean_mae = [np.mean([x[j] for x in all_scores]) for j in
            range(num_epochs)]
        plot(pd.DataFrame(mean_mae, columns=['MAE']), "Mean absolute
            error",
                f"Mean absolute error(K = {k}, num_epochs = {num_epochs
                    })")


def plot(data: pd.DataFrame, label: str, title: str):
    axis = sns.lineplot(data=data, dashes=False)
    axis.set(ylabel=label, xlabel='epochs', title=title.split('.'
        )[0])
    axis.grid(True, linestyle="--")
    # plt.show()
    plt.savefig(f"img/{title.replace(' ', '_').replace('#', '').
        replace('.', '-')}_{label.replace(' ', '_')}")
    plt.clf()
```

```
if __name__ == '__main__':
    (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
    for i in [3, 4, 5]:
        crossval_cycle(i, 100)
    for i in [3, 4, 5]:
        crossval_cycle(i, 40)
```