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 _____ Нечепуренко Н.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цели 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

Выполнение работы.

В предыдущих лабораторных работах с помощью искусственных нейронных сетей решалась задача классификации объектов, то есть отнесение объекта к какой-то конкретной категории из их конечного числа. Для цветов это был биологический вид, для отраженных сигналов – материал отражающей поверхности.

Задача регрессии формулируется иначе. Пусть имеется некоторый набор величин X, Y. Регрессией Y по X называется функция $g(x)$, такая что:

$$g(x) = E(Y|X = x)$$

В нашем случае мы будем вычислять среднее значение стоимости дома Y, при условии X, где X – множество признаков, таких как уровень преступности, ставка местного имущественного налога и т. д. Задача регрессии заключается в предсказании значения некоторой функции $f(x), x \in \mathbb{R}^n$, которая зачастую является непрерывной.

В данной лабораторной работе будет рассмотрена лишь одна архитектура ИНС (см. листинг 1).

```
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
```

Листинг 1 – Код создания модели

Модель состоит из двух слоев из 64 нейронов с функцией активации `relu` и

выходного линейного слоя. В качестве функции ошибки выбрана среднеквадратичная ошибка

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

где y_i целовое значение, $\tilde{y}_i$ – предсказанное. В качестве метрики используется средняя абсолютная ошибка

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \tilde{y}_i|$$

Чтобы получить более достоверное значение метрики, воспользуемся техникой кросс-валидации. Разделим тренировочное множество на «фолды», проще говоря, на равные части. Проведем столько итераций, на сколько частей поделили множество, вычисляя на каждой итерации нужные метрики, используя на i-той итерации i-тый фолд в качестве валидационного множества. Затем просто вычислим среднее по метрикам. Графичиски метод представлен на рисунке 1.

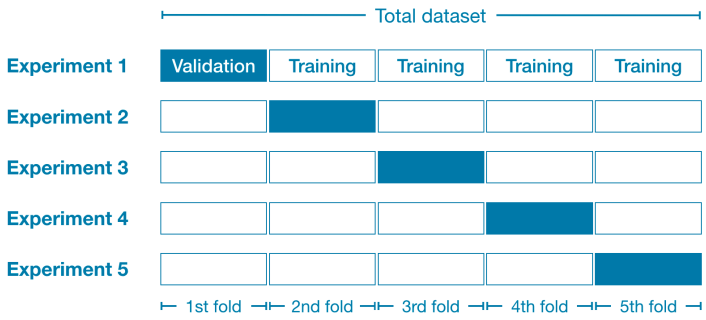


Рисунок 1 – Принцип использования кросс-валидации

Для небольших датасетов, как в данном случае, данная техника позволяет более точно оценить качество модели. На больших датасетах применение кросс-валидации затруднительно, так как связанно с большими вычислительными затратами.

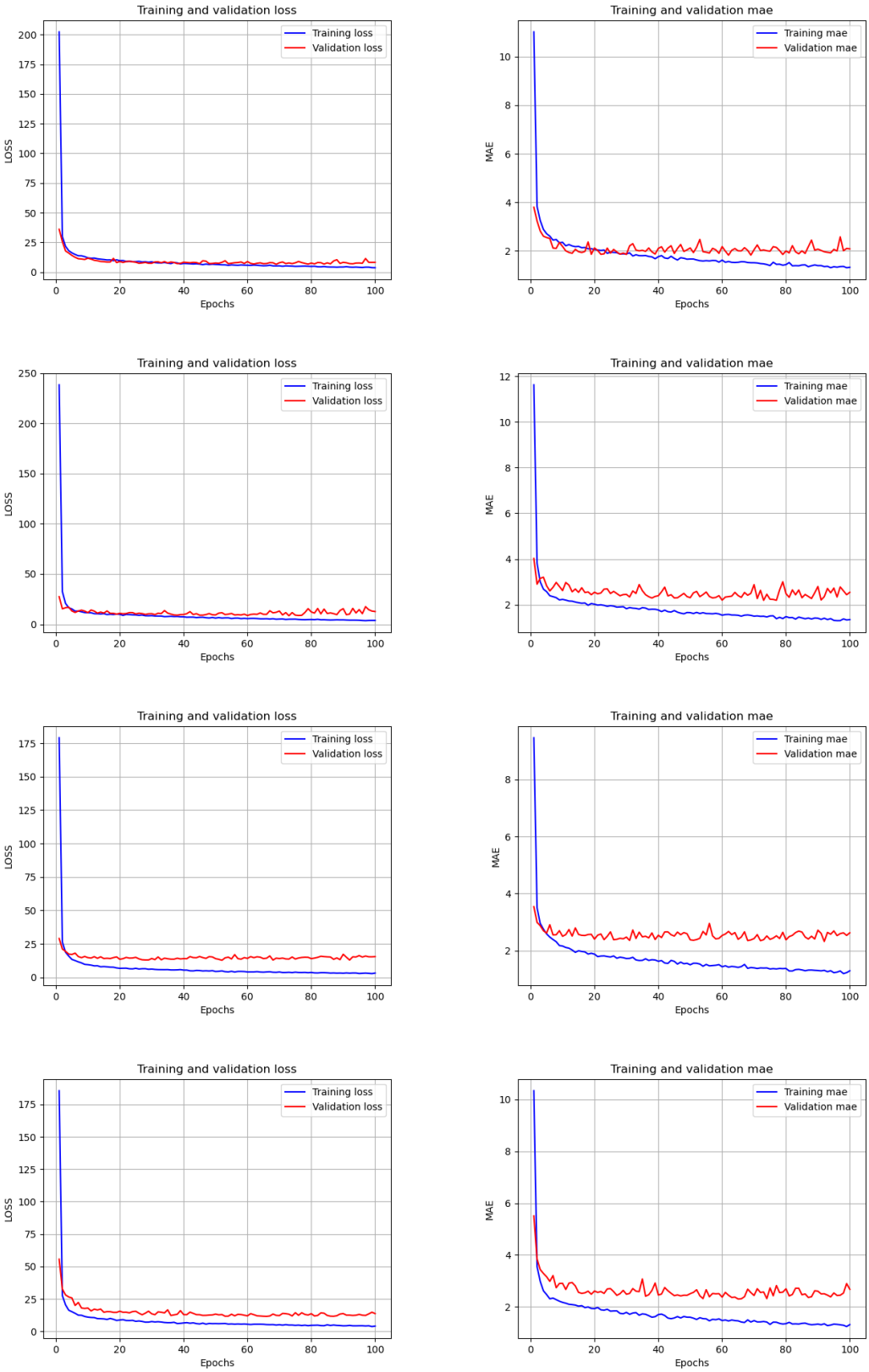


Рисунок 2 – Результаты обучения на 4 фолдах в течение 100 эпох

Обучим модель на 100 эпохах с разбиением на 4 фолда. Результаты обучения на фолдах приведены на рисунке 2, среднее значение метрик на рисунке 3.

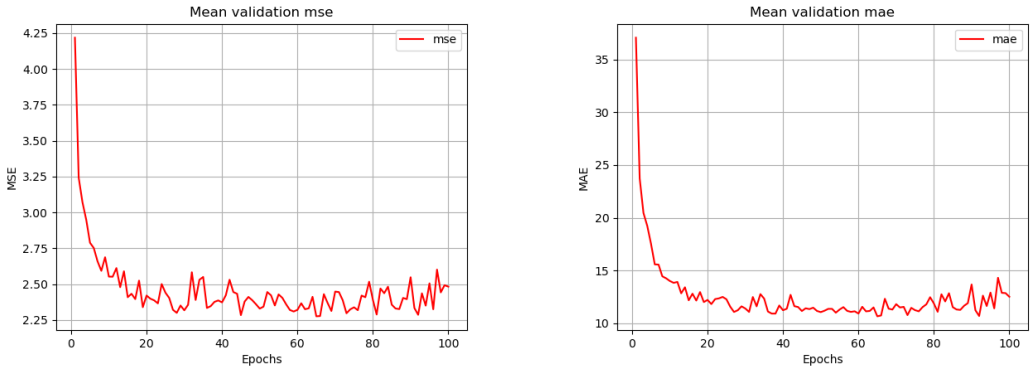
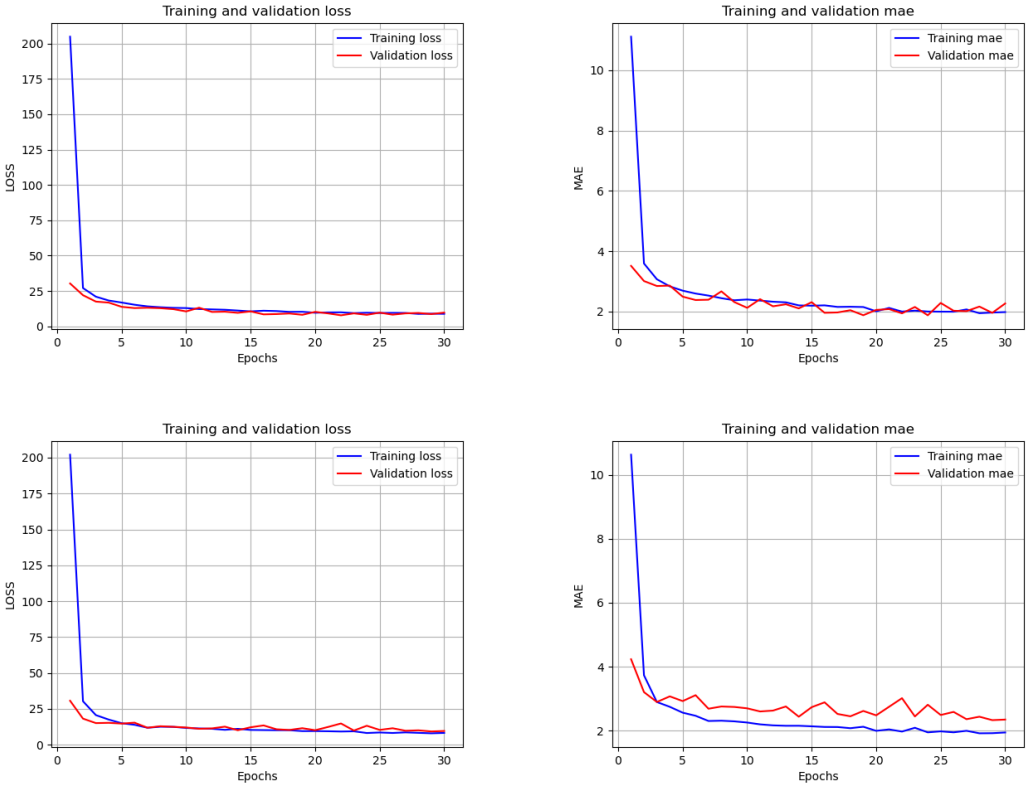


Рисунок 3 – Средние значения метрик на 4 фолдах в течение 100 эпох

Заметим, что через ~30 эпох модель начинает переобучаться. Обучим модель в течение 30 эпох.



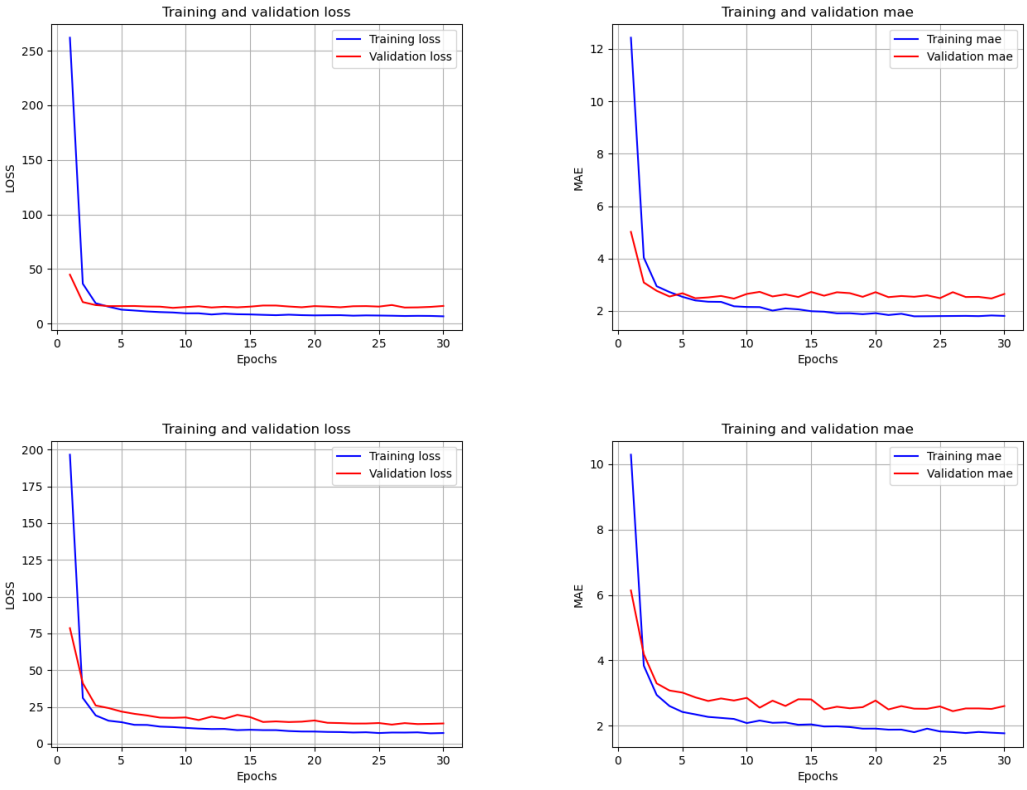


Рисунок 4 – Результаты обучения на 4 фолдах в течение 30 эпох

Средние значения метрик при обучении за 30 эпох приведены на рисунке 5.

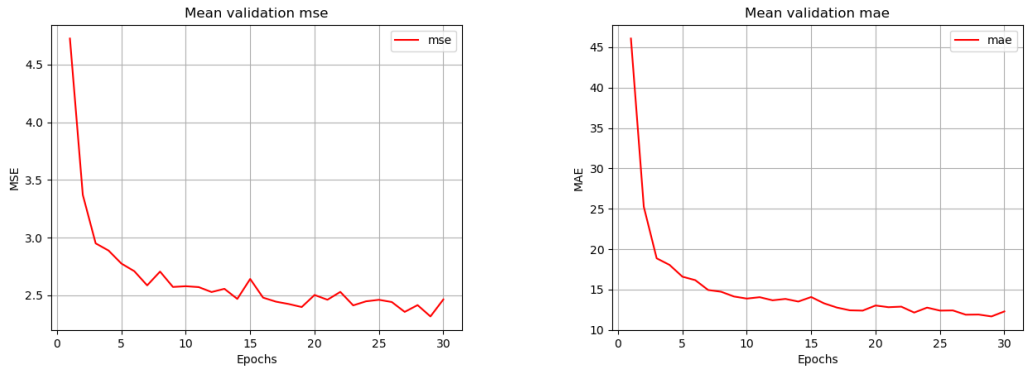
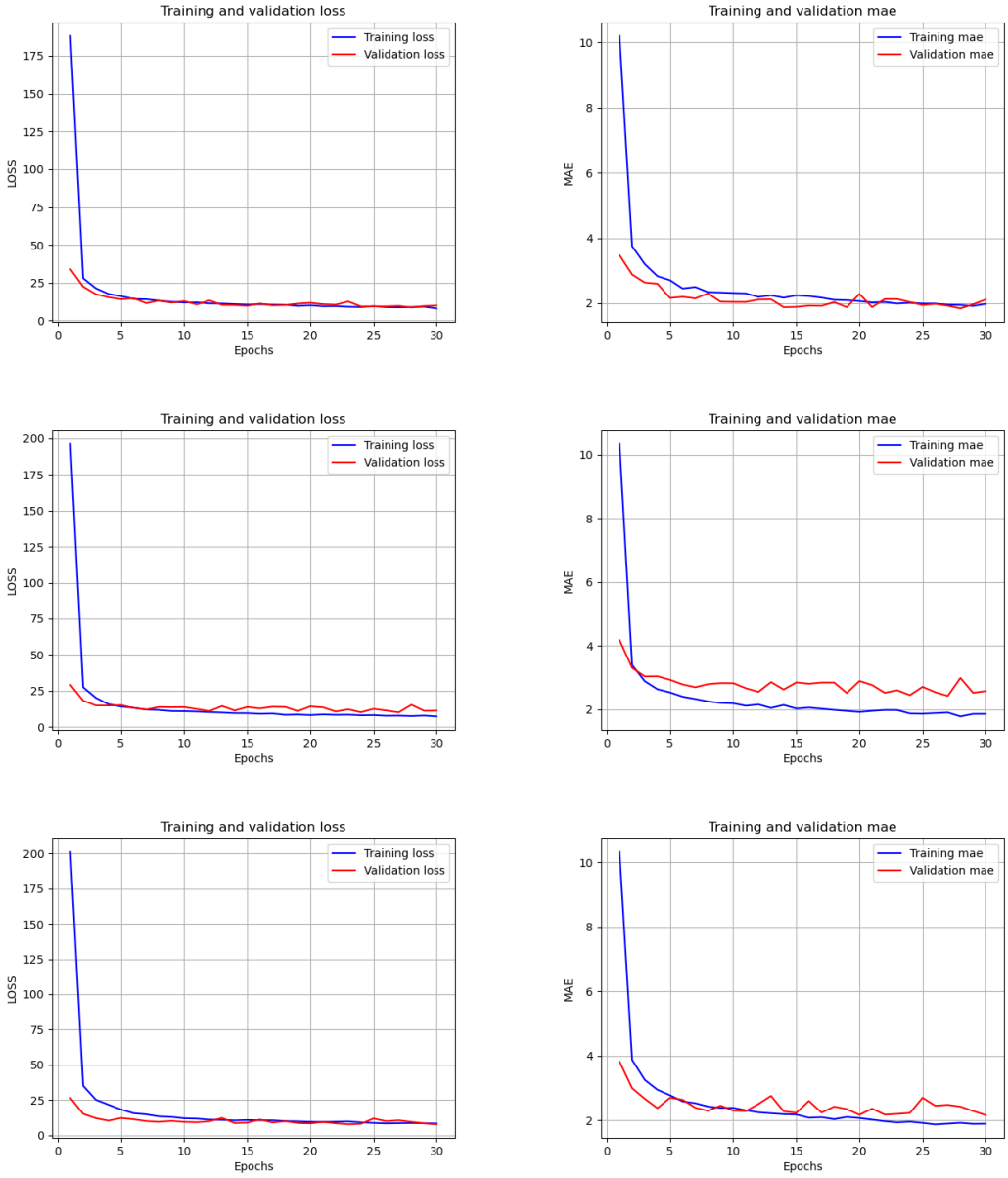


Рисунок 5 – Средние значения метрик на 4 фолдах в течение 30 эпох

Полученная оценка модели равна 2.464, это значит, что средняя разница между реальной стоимостью дома и предсказанной составляет $2464.

Оставим число эпох равным 30, попробуем провести кросс-валидацию на 5 фолдах (т.е. 20% тренировочного множества будут валидационными). Результаты модели приведены на рисунках 6 и 7.



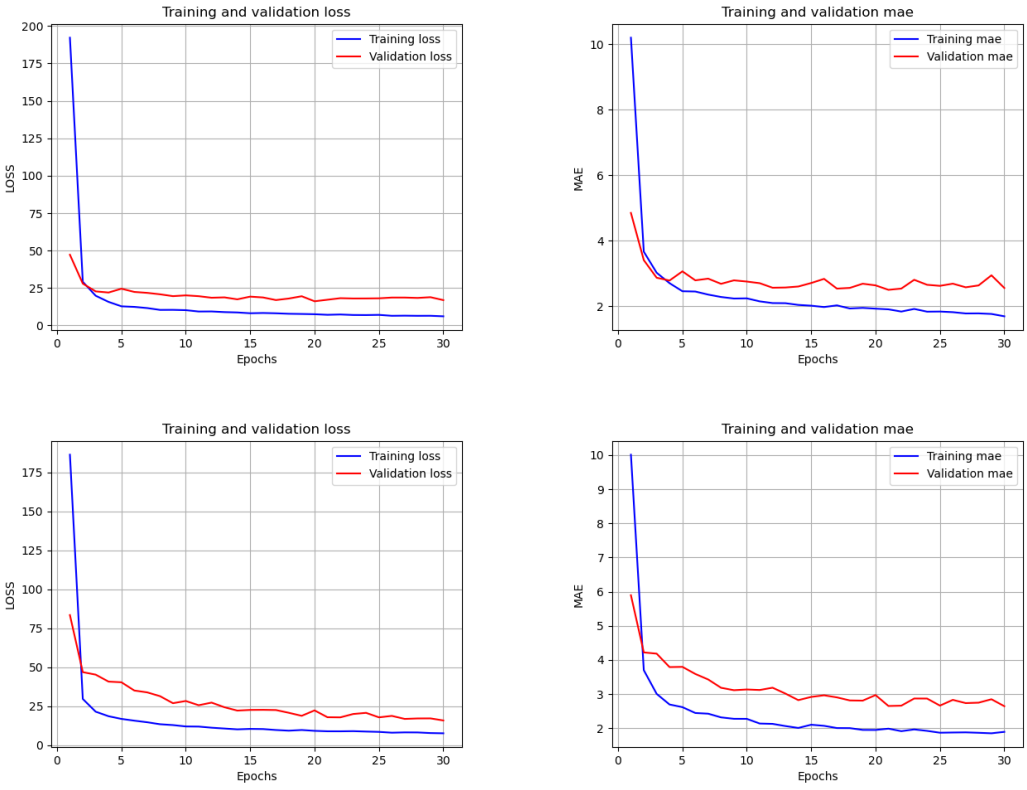


Рисунок 6 – Результаты обучения на 5 фолдах в течение 30 эпох

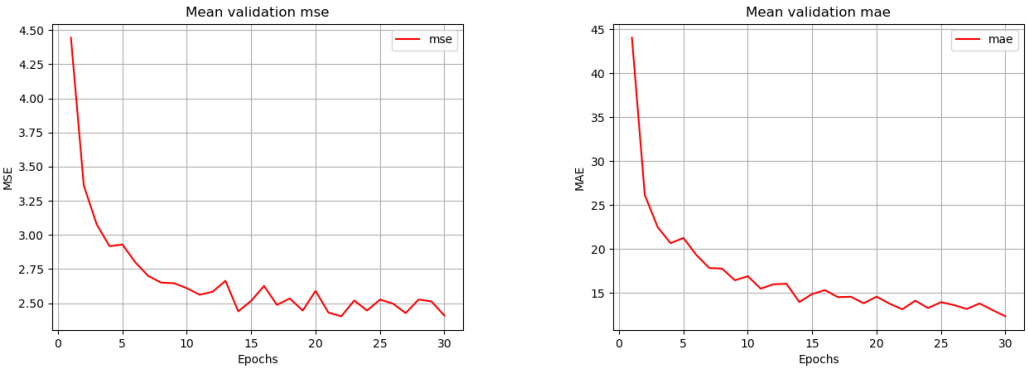


Рисунок 7 – Средние значения метрик на 5 фолдах в течение 30 эпох

Полученная оценка модели чуть лучше: 2.409.

Выводы.

В результате выполнения лабораторной работы были изучены отличия задачи регрессии от задачи классификации. Была построена искусственная нейронная сеть, предсказывающая цену домов в Бостоне. Была проведена кросс-валидация модели. По графикам средних значений метрик на валидационных множествах была обнаружена точка переобучения.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

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


def next_prefix():
    while True:
        yield "b"


def next_postfix():
    i = 0
    while True:
        i += 1
        yield i


prefix_generator = next_prefix()
postfix_generator = next_postfix()


def save_fig():
    plt.savefig(f"images/{next(prefix_generator)}{next(
       postfix_generator)}.png")


def plot(epochs, train, validation, metrics):
    plt.plot(epochs, train, 'b', label=f'Training {metrics}')
    plt.plot(epochs, validation, 'r', label=f'Validation {metrics
```

```
        }')
    plt.title(f'Training and validation {metrics}')
    plt.xlabel('Epochs')
    plt.ylabel(metrics.upper())
    plt.grid(True)
    plt.legend()


def plot_history(history):
    loss = history['loss']
    val_loss = history['val_loss']
    acc = history['mae']
    val_acc = history['val_mae']
    epochs = range(1, len(loss) + 1)

    plt.figure()
    plot(epochs, loss, val_loss, "loss")
    save_fig()
    plt.figure()
    plot(epochs, acc, val_acc, "mae")
    save_fig()


def plot_mean(epochs, mean_list, name):
    plt.plot(epochs, mean_list, 'r', label=f'{name}')
    plt.title(f'Mean validation {name}')
    plt.xlabel('Epochs')
    plt.ylabel(name.upper())
    plt.grid(True)
    plt.legend()
```

```
def plot_means(loss_list, metrics_list):
    mean_loss_list = np.mean(loss_list, axis=0)
    mean_metrics_list = np.mean(metrics_list, axis=0)
    epochs = range(1, len(mean_loss_list) + 1)
    plt.figure()
    plot_mean(epochs, mean_loss_list, "mse")
    save_fig()
    plt.figure()
    plot_mean(epochs, mean_metrics_list, "mae")
    save_fig()


def read_data():
    (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
    return (train_data, train_targets), (test_data, test_targets)


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 cross_validation(k=4, epochs=100):
    num_val_samples = len(train_data) // k
    all_scores = []
    all_val_mae = []
    all_val_mse = []
    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=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_val_mae.append(history.history["val_mae"])
        all_val_mse.append(history.history["val_loss"])
        plot_history(history.history)
```

```
    print(np.mean(all_scores))
    plot_means(all_val_mae, all_val_mse)


(train_data, train_targets), (test_data, test_targets) =
   read_data()
cross_validation(5, 30)
```