МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Искусственные нейронные сети»

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

Студент гр. 7381 _____ Кортев Ю. В.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

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.

Порядок выполнения работы.

- 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

. Объяснить различия задач классификации и регрессии
. 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

Ход работы.

В двух предыдущих работах решались задачи классификации, цель которых состояла в предсказании одной дискретной метки для образца входных данных. Другим распространенным типом задач машинного обучения является регрессия, которая заключается в предсказании не дискретной метки, а значения на непрерывной числовой прямой: например, предсказание температуры воздуха по имеющимся метеорологическим данным или предсказание времени завершения программного проекта по его спецификациям.

Из-за небольшого объема выборки будем использовать маленькую сеть с двумя скрытыми слоями по 64 нейрона в каждом. Чем меньше обучающих данных, тем скорее наступит переобучение, а использование маленькой сети - один из способов в борьбе с ним.

В данной работе для оценки качества сети и корректировки количества эпох используется метод перекрестной проверки, в процессе которого создается несколько моделей, поэтому процесс инициализации модели вынесен в отдельную функцию. Функция инициализации модели показана в листинге 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 - Инициализация модели

Также к входным данным необходимо применить нормализацию. Нормализация необхадима затем, чтобы все признаки определялись в близких диапозонах. Если передавать в сеть данные, которые могут принимать очень большие значения, это может привести к значительным изменениям градиента, что будет препятствовать сходимости сети. Нормализация данных представлена в листинге 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
```

Листинг 2 - Нормализация данных

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.

Корректировку количества эпох обучения буду проводить с помощью метода перекрестной проверки по К блокам. В данном методе тренировачные данные разбиваются на К блоков, создается К моделей, каждая модель обучается на К-1 блоках, а оценка производится по оставшимся блокам. В качестве оценки архитектуры сети берется среднее значения оценки каждой модели. Реализация метода показана в листинге 3.

```
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, validation_data=(val_data,val_targets),epochs=num_epochs, batch_size=1, verbose=0).history
```

Листинг 3 - Реализация метода перекрестной проверки

В конце каждого прогона строится график точности, которая характеризуется средней абсолютной ошибкой. В конце всех прогонов строится график усредненный по всем моделям. Установим количество эпох равное 300 и проанализируем полученные графики. Усредненные графики средней абсолютной ошибки и потери показаны на рисунках 1-2.

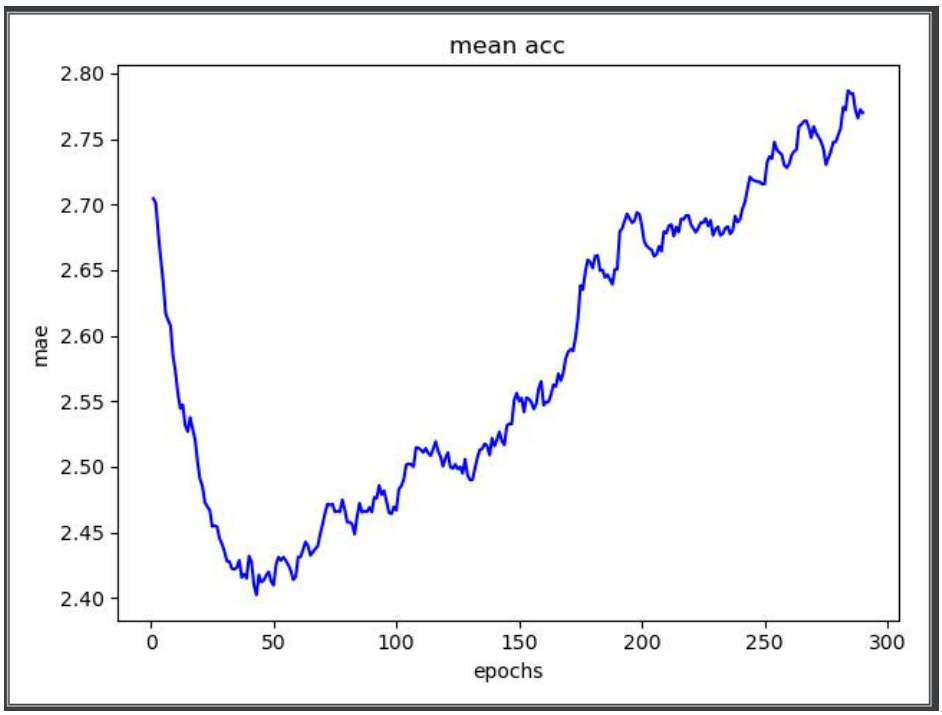

Рисунок 1 - График усредненной средней абсолютной ошибки

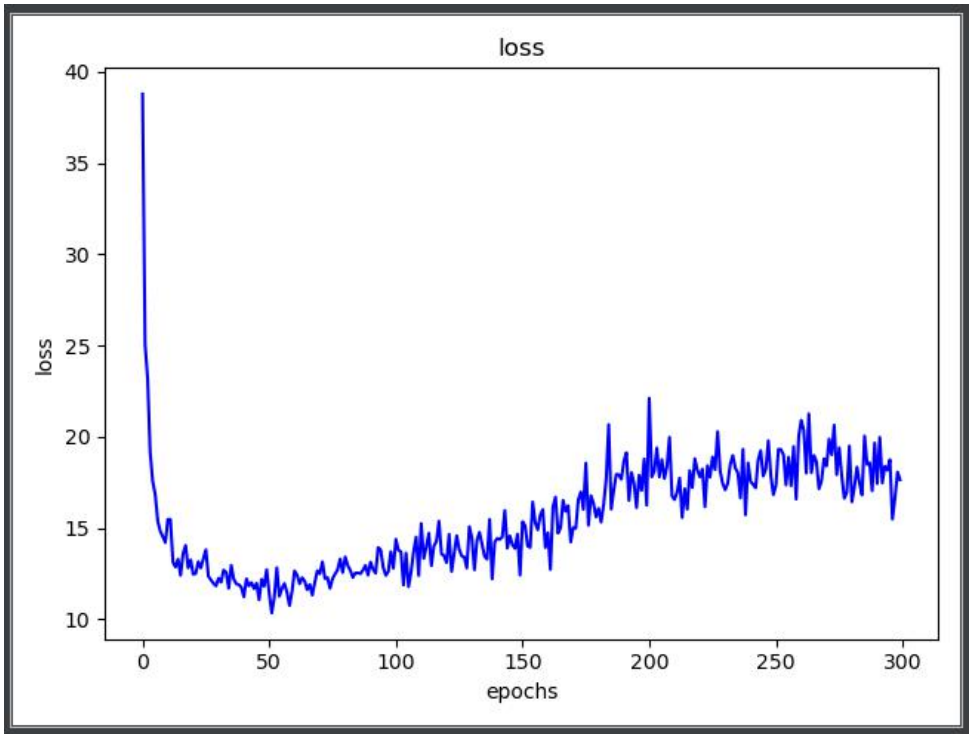

Рисунок 2 - График усредненной потери модели

График на рисунке один начинается с 10й эпохи, а также сглажен для более простого анализа. Как видно потери и средняя абсолюнтая ошибка достигают минимума примерно на 50й эпохе. Дальше сеть начинает переобучаться. Построим итоговую модель, обученную на всех

тренеровачных данных, за 50 эпох и оценим ее. В листинге 4 показана инициализация и обучение итоговой модели.

```
model=build_model()
model.fit(train_data,train_targets,epochs=50, batch_size=1,verbose=0)
val_mse,val_mae=model.evaluate(test_data,test_targets,verbose=0)
print(val_mae)
```

Листинг 4 - Итоговая модель

В результате средняя ошибка составляет около 2 317 долларов.

Выводы.

В ходе выполнения лабораторной работы исследовано влияние количества эпох на решение моделью задачи регрессии. Отработан метод перекрестной проверки для корректировки параметров обучения сети. Построена итоговая модель показывающая неплохой результат на достаточно небольной выборке.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
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

(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 = 4

num_val_samples = len(train_data) // k
num_epochs = 100

all_val_mae_history=[]
all_mae_history=[]
all_val_loss_history=[]
all_loss_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,
validation_data=(val_data,val_targets),epochs=num_epochs, batch_size=1,
verbose=0).history



    val_mae_history=history['val_mae']
    val_loss_history=history['val_loss']

    all_val_loss_history.append(val_loss_history)
    all_val_mae_history.append(val_mae_history)


    plt.plot(range(num_epochs), val_mae_history,'b',label='Val mae')
    plt.title('k=' + str(i))
    plt.xlabel('epochs')
    plt.ylabel('acc')
    plt.show()

def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 -
factor))
        else:
            smoothed_points.append(point)
    return smoothed_points


val_mae=[np.mean([x[i] for x in all_val_mae_history]) for i in
range(num_epochs)]
smooth_val_mae_history = smooth_curve(val_mae[10:])


plt.plot(range(1,
len(smooth_val_mae_history)+1),smooth_val_mae_history,'b',label='Val
mae')
plt.title('mean acc')
```

```
plt.xlabel('epochs')
plt.ylabel('mae')
plt.show()

val_loss=[np.mean([x[i] for x in all_val_loss_history]) for i in
range(num_epochs)]

plt.plot(range(len(val_loss)),val_loss,'b',label='Val mae')
plt.title('loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.show()

model=build_model()
model.fit(train_data,train_targets,epochs=50, batch_size=1,verbose=0)
val_mse,val_mae=model.evaluate(test_data,test_targets,verbose=0)
print(val_mae)
```