**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 |  | Вологдин М.Д. |
| 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.**

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

**Ход работы.**

Задача классификации сводится к определению класса объекта по его характеристикам. Необходимо заметить, что в этой задаче множество классов, к которым может быть отнесен объект, заранее известно.

Задача регрессии, подобно задаче классификации, позволяет определить по известным характеристикам объекта значение некоторого его параметра. В отличие от задачи классификации значением параметра является не конечное множество классов, а множество действительных чисел.

Посмотрим на результаты нейронной сети на данных по умолчанию – на 4 блоках и 100 эпохах. Графики представлены на рис. 1, 2.

|  |  |
| --- | --- |
|  |  |
| а | б |
|  |  |
| в | г |

Рисунок 1 – График оценки МАЕ для блока а–1, б–2, в–3, г–4.

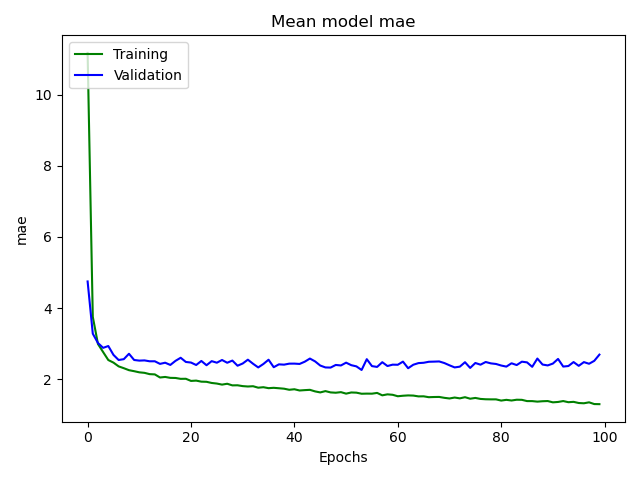


Рисунок 2 – График среднего значения МАЕ

Заметим, что оценки МАЕ на тестовых данных начинают возрастать после ~18 эпохи, значит следует убавить количество эпох до этого значения во избежание переобучения.

Рассмотрим модели с 18 эпохами на 2, 4, 6 и 8 блоках. Графики представлены на рис. 3.

|  |  |
| --- | --- |
|  |  |
| а | б |
|  |  |
| в | г |

Рисунок 3 – Графики среднего значения МАЕ для модели с количеством блоков: а–2, б–4, в–6, г–8.

Из графиков видим, что наилучшей сходимостью и наименьшей средней ошибкой обладает модель с 6 блоками.

**Выводы.**

В ходе выполнения данной работы была изучена задача регрессии и ее отличие от задачи классификации c помощью библиотеки Keras. Также было изучено влияние количества эпох и числа блоков на результат обучения сети.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД**

import numpy as np

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential

from tensorflow.keras.datasets import boston\_housing

import matplotlib.pyplot as plt

def build\_model():

model1 = Sequential()

model1.add(Dense(64, activation='relu', input\_shape=(train\_data.shape[1],)))

model1.add(Dense(64, activation='relu'))

model1.add(Dense(1))

model1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

return model1

(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 = 6

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

num\_epochs = 18

all\_scores = []

mean\_loss = []

mean\_mae = []

mean\_val\_loss = []

mean\_val\_mae = []

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)

mean\_val\_mae.append(history.history['val\_mean\_absolute\_error'])

mean\_mae.append(history.history['mean\_absolute\_error'])

plt.plot(history.history['mean\_absolute\_error'], 'g')

plt.plot(history.history['val\_mean\_absolute\_error'], 'b')

plt.title('Mean absolute error' + ', i = ' + str(i + 1))

plt.ylabel('mae')

plt.xlabel('Epochs')

plt.legend(['Training', 'Validation'], loc='upper left')

plt.show()

mean\_val\_loss.append(history.history['val\_loss'])

mean\_loss.append(history.history['loss'])

plt.plot(history.history['loss'], 'g')

plt.plot(history.history['val\_loss'], 'b')

plt.title('Model loss' + ', i = ' + str(i + 1))

plt.ylabel('loss')

plt.xlabel('Epochs')

plt.legend(['Training', 'Validation'], loc='upper left')

plt.show()

plt.plot(np.mean(mean\_mae, axis=0), 'g')

plt.plot(np.mean(mean\_val\_mae, axis=0), 'b')

plt.title('Mean model mae')

plt.ylabel('mae')

plt.xlabel('Epochs')

plt.legend(['Training', 'Validation'], loc='upper left')

plt.show()

plt.plot(np.mean(mean\_loss, axis=0), 'g')

plt.plot(np.mean(mean\_val\_loss, axis=0), 'b')

plt.title('Mean model loss')

plt.ylabel('loss')

plt.xlabel('Epochs')

plt.legend(['Training', 'Validation'], loc='upper left')

plt.show()