Санкт-Петербургский государственный морской технический университет

Отчёт

по практической работе №2

«Прогнозирование цены на жильё с помощью глубоких нейронных сетей»

Выполнил: Студент 2 курса

Факультета Цифровых

Промышленных Технологий

группы 20221 Хохлов Д.Р.

Преподаватель:

Кафедра киберфизических систем

Кайнова Татьяна Денисовна

Санкт-Петербург

2025

Оглавление

[Введение 2](#_Toc191637353)

[Постановка задачи 3](#_Toc191637354)

[Действия для выполнения задачи 3](#_Toc191637355)

[Датасет «Boston Housing» 3](#_Toc191637356)

[Конфигурации нейросетей 4](#_Toc191637357)

[Результаты обучения и анализ моделей 5](#_Toc191637358)

[Заключение 7](#_Toc191637359)

[Приложение А 8](#_Toc191637360)

# Введение

Цель работы: изучение и применение глубоких нейронных сетей для прогнозирования цен на жилье с использованием набора данных по жилью в Бостоне. В ходе работы необходимо модифицировать исходный код, добавляя регуляризацию, дропаут и изменяя топологию сети, а также провести анализ результатов.

## Постановка задачи

Задачей лабораторной работы являлось создание и модификация нейронной сети для прогнозирования цен на жилье в Бостоне. Требовалось:

1. Упростить исходную модель до одного нейрона и сравнить результаты.
2. Добавить регуляризацию L2, Dropout, увеличить количество слоёв и нейронов, а также проанализировать влияние изменений на ошибку обучения и теста.

## Действия для выполнения задачи

Для решения поставленной задачи необходимо:

1. Загрузка и стандартизация данных Boston Housing.
2. Построение исходной модели с двумя скрытыми слоями.
3. Последовательная модификация модели:
   1. Упрощение до одного нейрона.
   2. Добавление L2-регуляризации (λ = 0.1).
   3. Включение Dropout (коэффициент 0.2 → 0.3).
   4. Увеличение количества слоёв и нейронов (128/128/64).
4. Обучение и оценка моделей.
5. Сравнение ошибок (MSE) на обучающей и тестовой выборках.

## 

## Датасет «Boston Housing»

Датасет содержит 506 записей с 13 признаками (например, уровень преступности, среднее число комнат), а также целевую переменную — медианную стоимость жилья в тысячах долларов. Данные разделены на обучающую и тестовую выборки.

Датасет содержит данные (Рисунок 1):

* Уровень преступности на душу населения по городу (CRIM)
* Доля жилой земли, зонированной для участков более 25 000 кв. футов (ZN)
* Доля неторговых бизнес-акров на город (INDUS)
* Фиктивная переменная реки Чарльз (CHAS)
* Концентрация нитритов (NOX)
* Среднее количество комнат на жилище (RM)
* Доля домов, построенных до 1940 года. (AGE)
* **В**звешенные расстояния до пяти Boston employment centers (DIS)
* **И**ндекс доступности к радиальным шоссе (RAD)
* **П**олная ставка налога на имущество на 10 000 долларов (TAX)
* Соотношение учеников и учителей по городу (PYRATIO)
* Доля чернокожих жителей по городу (B)
* Процент населения с низким статусом (LSTAT)
* Медианная стоимость домовладельческих домов в 1000 долларов (MEDV)

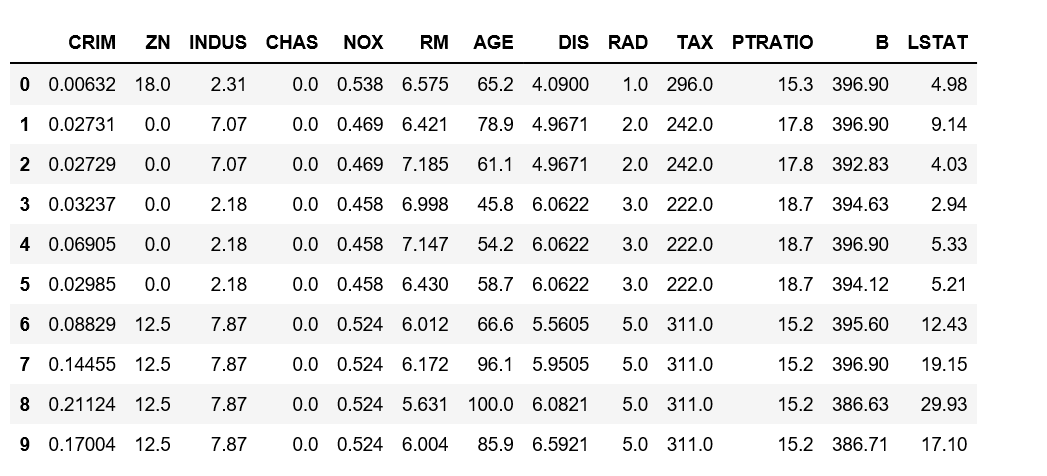
н

Рисунок 1 – Пример данных из датасета

Установка датасета уже встроена в библиотеку Tensorflow, она импортируется через встроенные методы.

## Конфигурации нейросетей

Было построено 6 моделей нейросетей:

* 2 слоя по 64 нейрона и 1 выходной
* 2 слоя по 1 нейрону
* 2 слоя по 64 нейрона и 1 выходной, регуляризация L2
* 2 слоя по 64 нейрона и 1 выходной, регуляризация L2 и Dropout
* 2 слоя по 128 нейронов, 1 слой – 64 нейрона и 1 выходной, регуляризация L2 и Dropout
* 2 слоя по 128 нейронов, 1 слой – 64 нейрона и 1 выходной, регуляризация L2 и Dropout с изменённым коэффициентом отсева

Нейросети строятся при помощи библиотек Tensorflow и Keras.

## Результаты обучения и анализ моделей

Все 6 моделей были обучены и результаты их ошибок во время обучения и теста были записаны для дальнейшего анализа (Таблица 1).

Таблица 1 – Результаты обучения моделей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Конфигурация | Топология | Регуляризация | Ошибка  обучения | Ошибка  теста |
| Оригинальная NN | 64/64/1 | - | 0.8120 | 12.0941 |
| Упрощённая NN  (1 слой - 1 нейрон) | 1/1 | - | 116.2056 | 123.0750 |
| Оригинальная NN, регуляризация (L2) | 64/64/1 | L2(λ = 0.1) | **0.7097** | **11.7362** |
| Оригинальная NN, регуляризация (L2, Dropout) | 64/64/1 | L2(λ = 0.1),  Dropout(λ = 0.2) | 10.1573 | 14.8412 |
| Расширенная NN,  Регуляризация(L2, Dropout) | 128/128/64/1 | L2(λ = 0.1),  Dropout(λ = 0.2) | 7.9811 | 12.8703 |
| Расширенная NN,  Регуляризация(L2, Dropout) | 128/128/64/1 | L2(λ = 0.1),  Dropout(λ = 0.3) | 8.0546 | 10.8399 |

Из таблицы видно, что меньшим коэффициентом ошибки обучения и теста обладает «Оригинальная модель с регуляризацией L2», однако у данной модели есть большая вероятность быть переобученной, что и наблюдается, т. к. коэффициент ошибки обучения сильно меньше, чем коэффициент ошибки теста. Такая модель не будет хорошо справляться со своей задачей, поскольку она просто «запомнила» данные из обучающего набора.

Лучшей нейросетью по функциональности является «Расширенная модель с регуляризацией L2 и Dropout». Она обладает хорошими коэффициентами ошибок и не переобучена на тренировочном наборе, т. к. во время обучения случайно отключалось 30% нейронов.

# Заключение

В ходе практической работы было проведено сравнение нескольких нейронных моделей с различными конфигурациями, включающих себя способы регуляризации и устранение переобучения нейросетей. Был проведён анализ на лучшую конфигурацию из обученных моделей.

# Приложение А

Листинг программы

EPOCHS = 500  
BATCH\_SIZE = 16  
DEBUG = False  
#%% md  
## Установка/Импортирование библиотек  
#%%  
!pip install tensorflow  
!pip install numpy  
#%%  
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Dropout  
from tensorflow.keras.regularizers import L2  
import numpy as np  
import logging  
  
from tensorflow.python.saved\_model.pywrap\_saved\_model.metrics import kFingerprintError  
#%% md  
## Работа с датасетом  
#%%  
bostondata = keras.datasets.boston\_housing  
(raw\_x\_train, y\_train), (raw\_x\_test,y\_test) = bostondata.load\_data()  
x\_mean = np.mean(raw\_x\_train, axis=0)  
x\_stddev = np.std(raw\_x\_train, axis=0)  
#%%  
x\_train =(raw\_x\_train -x\_mean) / x\_stddev  
x\_test =(raw\_x\_test - x\_mean) / x\_stddev  
#%% md  
## Model (Simplified)  
#%%  
model\_simple = Sequential()  
model\_simple.add(Dense(1, activation='linear', input\_shape=[13]))  
model\_simple.compile(loss="mse", optimizer='adam',metrics = ["mae"])  
model\_simple.summary() if DEBUG == True else None  
#%%  
#Fit model\_simple  
history\_simple = model\_simple.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1, shuffle=True)  
#%% md  
## Original Model  
### 64/64/1  
  
  
model\_orig = Sequential()  
model\_orig.add(Dense(64, activation='relu', input\_shape=[13]))  
model\_orig.add(Dense(64, activation='relu'))  
model\_orig.add(Dense(1, activation= "linear"))  
model\_orig.compile(loss="mse", optimizer='adam',metrics = ["mae"])  
model\_orig.summary() if DEBUG == True else None  
  
#Fit model\_orig\_l2\_drop  
history\_orig = model\_orig.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1, shuffle=True)  
#%%  
model\_orig\_l2 = Sequential()  
model\_orig\_l2.add(Dense(64, activation='relu', input\_shape=[13]))  
  
model\_orig\_l2.add(Dense(64, activation='relu', bias\_regularizer=L2(0.1)))  
  
model\_orig\_l2.add(Dense(1, activation="linear"))  
model\_orig\_l2.compile(loss="mse", optimizer='adam', metrics=["mae"])  
model\_orig\_l2.summary() if DEBUG == True else None  
  
#Fit model\_orig\_l2  
history\_orig\_l2 = model\_orig\_l2.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=EPOCHS, batch\_size=BATCH\_SIZE,  
 verbose=1, shuffle=True)  
#%% md  
## Original Model  
### 64/64/1, l2 = 0.1, dropout = 20%  
#%%  
model\_orig\_l2\_drop = Sequential()  
model\_orig\_l2\_drop.add(Dense(64, activation='relu', input\_shape=[13]))  
model\_orig\_l2\_drop.add(Dense(64, activation='relu', bias\_regularizer=L2(0.1)))  
model\_orig\_l2\_drop.add(Dense(1, activation= "linear"))  
model\_orig\_l2\_drop.compile(loss="mse", optimizer='adam',metrics = ["mae"])  
model\_orig\_l2\_drop.summary() if DEBUG == True else None  
  
#%%  
#Fit model\_orig\_l2\_drop  
history\_orig\_l2\_drop = model\_orig\_l2\_drop.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1, shuffle=True)  
#%% md  
## Extended model  
### 128/128/64/1, l2 = 0.1, dropout = 20%  
#%%  
model\_extend = Sequential()  
model\_extend.add(Dense(128, activation='relu', input\_shape=[13]))  
model\_extend.add(Dropout(0.2))  
model\_extend.add(Dense(128, activation='relu', bias\_regularizer=L2(0.1)))  
model\_extend.add(Dropout(0.2))  
model\_extend.add(Dense(64, activation="relu", bias\_regularizer=L2(0.1)))  
model\_extend.add(Dropout(0.2))  
model\_extend.add(Dense(1, activation="linear"))  
model\_extend.compile(loss="mse", optimizer='adam', metrics=["mae"])  
model\_extend.summary() if DEBUG == True else None  
  
#%%  
history\_extend = model\_extend.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=EPOCHS,  
 batch\_size=BATCH\_SIZE, verbose=1, shuffle=True)  
#%% md  
## Extended model  
### 128/128/64/1, l2 = 0.1, dropout = 30%  
#%%  
model\_extend2 = Sequential()  
model\_extend2.add(Dense(128, activation='relu', input\_shape=[13]))  
model\_extend2.add(Dropout(0.3))  
model\_extend2.add(Dense(128, activation='relu', bias\_regularizer=L2(0.1)))  
model\_extend2.add(Dropout(0.3))  
model\_extend2.add(Dense(64, activation= "relu"))  
model\_extend2.add(Dropout(0.3))  
model\_extend2.add(Dense(1, activation="linear"))  
model\_extend2.compile(loss="mse", optimizer='adam', metrics=["mae"])  
model\_extend2.summary() if DEBUG == True else None  
#%%  
history\_extend2 = model\_extend2.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1, shuffle=True)