

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
По лабораторной работе №6
Дисциплины «Основы нейронных сетей»

Выполнил:

Говоров Егор Юрьевич

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

(подпись)

Проверил:

Воронкин Р. А., доцент департамента
цифровых и робототехнических
систем и электроники и института
перспективной инженерии

(подпись)

Тема: Исследование регрессии нейронных сетей.

Цель: изучение и применение методов регрессии с использованием нейронных сетей для решения практических задач, таких как предсказание зарплаты, оценка результатов баскетбольных матчей и прогнозирование цен на автомобили.

Ссылка: https://github.com/Artorias1469/NN_6.git

Ход работы:

Выполнение индивидуальных заданий:

Задание 1.

Условие: необходимо на основе подготовленной для нейросети таблицы данных с HeadHunter создать 6 архитектур нейросетей, поэкспериментировать с гиперпараметрами. Использовать только числовые данные, текстовые не подавать. Результаты в конце необходимо проанализировать.

Перед началом выполнения, необходимо, запустить раздел "Подготовка". Для начала выполним импорт необходимых библиотек.

```
# Стандартная библиотека
import re
import warnings

# Регулярные выражения
# Для подавления предупреждений

# Сторонние библиотеки
import gdown
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Загрузка датасетов из облака google
# Отрисовка графиков
# Работа с массивами данных
# Работа с табличными данными

from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import utils
from tensorflow.keras.layers import (
    Activation,
    BatchNormalization,
    Bidirectional,
    Conv1D,
    Dense,
    Dropout,
    Embedding,
    Flatten,
    GlobalMaxPooling1D,
    GRU,
    Input,
    LSTM,
    MaxPooling1D,
    SimpleRNN,
    SpatialDropout1D,
    concatenate,
)

# Метрики для расчета ошибок
# Масштабирование данных
# Функции-утилиты для работы с категориальными данными
# Основные слои

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adadelta, Adagrad, Adam, RMSprop, SGD
from tensorflow.keras.preprocessing.text import Tokenizer

# Класс для конструирования последовательной модели нейронной с
# Оптимизаторы
# Токенизатор для преобразования текстов в последовательности

# Подавляем предупреждения для чистого вывода
warnings.filterwarnings('ignore')

# Магическая команда Jupyter
%matplotlib inline
```

Рисунок 1. Загрузка библиотек

Далее выполним скачивание базы, а затем чтение данного файла и вывод количества резюме и числа признаков

```
# скачиваем базу
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/118/hh_fixed.csv', None, quiet=True)

# Чтение файла базы данных
df = pd.read_csv('hh_fixed.csv', index_col=0)

# Вывод количества резюме и числа признаков
print(df.shape)

df.head(3)
```

(62967, 12)

	Пол, возраст	ЗП	Ищет работу на должность:	Город	Занятость	График	Опыт (двойное нажатие для полной версии)	Последнее/нынешнее место работы	Последняя/нынешняя должность	образование и вуз	Обновление резюме	Авто
0	Мужчина, 29 лет, родился 16 мая 1989	40000 руб.	Специалист по поддержке чата(support team) дом...	Новороссийск, готов к переезду (Анапа, Геленда...	полная занятость	полный день	Опыт работы 3 года 9 месяцев Специалист по по...	ООО "Тольфстрим"	Генеральный директор	Высшее образование 2011 Международный юридиче...	28.04.2019 08:04	Не указано
1	Мужчина, 38 лет, родился 25 мая 1980	40000 руб.	Системный администратор	Новосибирск, м. Березовая роща, не готов к ...	полная занятость	полный день	Опыт работы 11 лет 11 месяцев Системный админ...	ООО «Завод модульных технологий»	Системный администратор	Высшее образование 2002 Новосибирский государс...	28.04.2019 04:30	Не указано
2	Мужчина, 35 лет, родился 14 июня 1983	300000 руб.	DevOps TeamLead / DevOps архитектор	Москва, готов к переезду, готов к редким ком...	полная занятость	полный день	Опыт работы 12 лет 11 месяцев DevOps TeamLead...	Банк ВТБ (ПАО)	Начальник отдела методологии разработки (DevOp...	DevOps TeamLead / DevOps архитектор 300 000 ру...	09.04.2019 14:40	Не указано

Рисунок 2. Скачивание базы

Затем выполним настройку столбцов, таких как пол, возраст, зп, город, занятость и тд.

```
COL_SEX_AGE = df.columns.get_loc('Пол, возраст')
COL_SALARY = df.columns.get_loc('ЗП')
COL_POS_SEEK = df.columns.get_loc('Ищет работу на должность:')
COL_POS_PREV = df.columns.get_loc('Последняя/нынешняя должность')
COL_CITY = df.columns.get_loc('Город')
COL_EMPL = df.columns.get_loc('Занятость')
COL_SCHED = df.columns.get_loc('График')
COL_EXP = df.columns.get_loc('Опыт (двойное нажатие для полной версии)')
COL_EDU = df.columns.get_loc('образование и вуз')
COL_UPDATED = df.columns.get_loc('Обновление резюме')
```

Рисунок 3. Настройка номеров столбцов

Далее выполним замену концов строк на пробелы, удаление символа с кодом 0xA0, а также обрезку краевых пробелов и приведение к нижнему регистру

```
# Замена концов строк на пробелы, удаление символа с кодом 0xA0
# обрезка краевых пробелов, приведение к нижнему регистру

def purify(x):
    if isinstance(x, str):
        # Если значение - строка:
        x = x.replace('\n', ' ').replace('\xa0', '').strip().lower()
    return x
```

Рисунок 4. Замена элементов

После выполним выделение подстроки вида ДД.ММ.ГГГГ и возвращение значения года

```
# Выделение подстроки вида ДД.ММ.ГГГГ и возвращение значения года

def extract_year(x):
    try:
        return int(re.search(r'\d\d.\d\d.\d{4}', x)[1]) # Ожидается строка вида 'dd.mm.yyyy ...'
    except (IndexError, TypeError, ValueError):
        return 0
```

Рисунок 5. Выделение подстроки вида ДД.ММ.ГГГГ

Далее добавим параметрические данные для функций разбора (курсы валют для зарплат, список порогов возраста, список порогов опыта работы в месяцах, классы городов, классы занятости, классы графика работы и классы образования)

```
# Курсы валют для зарплат
currency_rate = {'usd' : 65.,
                 'kzt' : 0.17,
                 'грн' : 2.6,
                 'белруб' : 30.5,
                 'eur' : 70.,
                 'kgs' : 0.9,
                 'сум' : 0.007,
                 'azn' : 37.5
                }

# Списки и словари для разбиения на классы
# Для ускорения работы добавлен счетчик классов, который будет вычислен ниже

# Список порогов возраста
age_class = [0, [18, 23, 28, 33, 38, 43, 48, 53, 58, 63]]

# Список порогов опыта работы в месяцах
experience_class = [0, [7, 13, 25, 37, 61, 97, 121, 157, 193, 241]]

# Классы городов
city_class = [0,
              {'москва' : 0,
               'санкт-петербург' : 1,
               'новосибирск' : 2,
               'екатеринбург' : 2,
               'нижний новгород' : 2,
               'казань' : 2,
               'челябинск' : 2,
               'омск' : 2,
               'самара' : 2,
               'ростов-на-дону' : 2,
               'уфа' : 2,
               'красноярск' : 2,
               'пермь' : 2,
               'воронеж' : 2,
               'волгоград' : 2,
               'прочие города' : 3
              }]

# Классы занятости
employment_class = [0,
                    {'стажировка' : 0,
                     'частичная занятость' : 1,
                     'проектная работа' : 2,
                     'полная занятость' : 3
                    }]

# Классы графика работы
schedule_class = [0,
                  {'гибкий график' : 0,
                   'полный день' : 1,
                   'сменный график' : 2,
                   'удаленная работа' : 3
                  }]

# Классы образования
education_class = [0,
                   {'высшее образование' : 0,
                    'higher education' : 0,
                    'среднее специальное' : 1,
                    'неоконченное высшее' : 2,
                    'среднее образование' : 3
                   }]

```

Рисунок 6. Параметрические данные

Затем выполним вычисление счетчиков для данных разбиения

```

# Вычисление счетчиков для данных разбиения

for class_desc in [age_class,
                   experience_class,
                   city_class,
                   employment_class,
                   schedule_class,
                   education_class]:
    if isinstance(class_desc[1], list):
        class_desc[0] = len(class_desc[1]) + 1
    else:
        class_desc[0] = max(class_desc[1].values()) + 1

```

Рисунок 7. Вычисление счетчиков для данных разбиения

Далее выполним получение one hot encoding представления значения класса, для этого напишем функцию

```

# Получение one hot encoding представления значения класса

def int_to_ohe(arg, class_list):

    # Определение размерности выходного вектора
    num_classes = class_list[0]

    # Поиск верного интервала для входного значения
    for i in range(num_classes - 1):
        if arg < class_list[1][i]:
            cls = i
            break
    else:
        cls = num_classes - 1

    # Возврат в виде one hot encoding-вектора
    return utils.to_categorical(cls, num_classes)

```

Рисунок 8. Функция one hot encoding представления значения класса

После напишем общую функцию преобразования строки к multi-вектору. Получим на входе данные и словарь сопоставления подстрок классам

```

# Общая функция преобразования строки к multi-вектору
# На входе данные и словарь сопоставления подстрок классам

def str_to_multi(arg, class_dict):
    # Определение размерности выходного вектора
    num_classes = class_dict[0]

    # Создание нулевого вектора
    result = np.zeros(num_classes)

    # Поиск значения в словаре и, если найдено,
    # выставление 1. на нужной позиции
    for value, cls in class_dict[1].items():
        if value in arg:
            result[cls] = 1.

    return result

```

Рисунок 9. Функция преобразования строки к multi-вектору

Далее напишем функцию для разбора значений пола и возраста.

```

base_update_year = 2019

def extract_sex_age_years(arg):
    # Ожидается, что значение содержит "мужчина" или "женщина"
    # Если "мужчина" - результат 1., иначе 0.
    sex = 1. if 'муж' in arg else 0.

    try:
        # Выделение года и вычисление возраста
        years = base_update_year - int(re.search(r'\d{4}', arg)[0])

    except (IndexError, TypeError, ValueError):
        # В случае ошибки год равен 0
        years = 0

    return sex, years

```

Рисунок 10. Разбор значений пола, возраста

Затем выполним преобразование значения возраста в one hot encoding

```

def age_years_to_ohe(arg):
    return int_to_ohe(arg, age_class)

```

Рисунок 11. Преобразование значения возраста в one hot encoding

После также выполним и преобразование данных об опыте работы в one hot encoding

```

def experience_months_to_ohe(arg):
    return int_to_ohe(arg, experience_class)

```

Рисунок 12. Преобразование данных об опыте работы в one hot encoding

Далее напишем функцию для разбора значений зарплаты.

```

def extract_salary(arg):
    try:
        # Выделение числа и преобразование к float
        value = float(re.search(r'\d+', arg)[0])

        # Поиск символа валюты в строке, и, если найдено,
        # приведение к рублю по курсу валюты
        for currency, rate in currency_rate.items():
            if currency in arg:
                value *= rate
                break

    except TypeError:
        # Если не получилось выделить число - вернуть 0
        value = 0.

    return value / 1000. # В тысячах рублей

```

Рисунок 13. Разбор значения зарплаты

Затем напишем функцию для разбора данных о городе, а также выполним преобразование в one hot encoding

```
def extract_city_to_one(arg):
    # Определение размерности выходного вектора
    num_classes = city_class[0]

    # Разбивка на слова
    split_array = re.split(r'[ ,.:()?!]', arg)

    # Поиск города в строке и присвоение ему класса
    for word in split_array:
        city_cls = city_class[1].get(word, -1)
        if city_cls >= 0:
            break
    else:
        # Внимание: for/else
        # Город не в city_class - значит его класс "прочие города"
        city_cls = num_classes - 1

    # Возврат в виде one hot encoding-вектора
    return utils.to_categorical(city_cls, num_classes)
```

Рисунок 14. Функция разбора данных о городе

Далее напишем функцию разбора данных о желаемой занятости и выполним преобразование в multi .

```
def extract_employment_to_multi(arg):
    return str_to_multi(arg, employment_class)
```

Рисунок 15. Разбор данных о желаемой занятости

Потом выполним функцию разбора данных о желаемом графике работы, а также выполним преобразование в multi

```
def extract_schedule_to_multi(arg):
    return str_to_multi(arg, schedule_class)
```

Рисунок 16. Разбор данных о желаемом графике работы

Затем напишем функцию разбора данных об образовании и выполним преобразование в multi

```
# Разбор данных об образовании и преобразование в multi

def extract_education_to_multi(arg):
    result = str_to_multi(arg, education_class)

    # Поправка: неоконченное высшее не может быть одновременно с высшим
    if result[2] > 0.:
        result[0] = 0.

    return result
```

Рисунок 17. Разбор данных об образовании

Далее напишем функцию разбора данных об опыте работы, то есть результат в месяцах

```

# Разбор данных об опыте работы - результат в месяцах

def extract_experience_months(arg):
    try:
        # Выделение количества лет, преобразование в int
        years = int(re.search(r'(\d+)\s+(год.?|лет)', arg)[1])

    except (IndexError, TypeError, ValueError):
        # Неудача - количество лет равно 0
        years = 0

    try:
        # Выделение количества месяцев, преобразование в int
        months = int(re.search(r'(\d+)\s+месяц', arg)[1])

    except (IndexError, TypeError, ValueError):
        # Неудача - количество месяцев равно 0
        months = 0

    # Возврат результата в месяцах
    return years * 12 + months

```

Рисунок 18. Разбор данных об опыте работы

Затем напишем функции подготовки выборок, а именно функцию для извлечения и преобразования данных и функцию для создания общей выборки

```

def extract_row_data(row):

    # Извлечение и преобразование данных
    sex, age = extract_sex_age_years(row[COL_SEX_AGE]) # Пол, возраст
    sex_vec = np.array([sex]) # Пол в виде вектора
    age_one = age_years_to_one(age) # Возраст в one hot encoding
    city_one = extract_city_to_one(row[COL_CITY]) # Город
    empl_multi = extract_employment_to_multi(row[COL_EMPL]) # Тип занятости
    sched_multi = extract_schedule_to_multi(row[COL_SCHED]) # График работы
    edu_multi = extract_education_to_multi(row[COL_EDU]) # Образование
    exp_months = extract_experience_months(row[COL_EXP]) # Опыт работы в месяцах
    exp_one = experience_months_to_one(exp_months) # Опыт работы в one hot encoding
    salary = extract_salary(row[COL_SALARY]) # Зарплата в тысячах рублей
    salary_vec = np.array([salary]) # Зарплата в виде вектора

    # Объединение всех входных данных в один общий вектор
    x_data = np.hstack([sex_vec,
                        age_one,
                        city_one,
                        empl_multi,
                        sched_multi,
                        edu_multi,
                        exp_one])

    # Возврат входных данных и выходных (зарплаты)
    return x_data, salary_vec

# Создание общей выборки
def construct_train_data(row_list):
    x_data = []
    y_data = []

    for row in row_list:
        x, y = extract_row_data(row)
        if y[0] > 0: # Данные добавляются, только если есть зарплата
            x_data.append(x)
            y_data.append(y)

    return np.array(x_data), np.array(y_data)

```

Рисунок 19. Функции подготовки выборок

Выполним формирование выборки из загруженного набора данных

```

# Формирование выборки из загруженного набора данных
x_train_01, y_train = construct_train_data(df.values)

```

Рисунок 20. Формирование выборки из загруженного набора данных

Далее сделаем вывод формы наборов параметров и зарплат


```

# Форма наборов параметров и зарплат
print(x_train_01.shape)
print(y_train.shape)

# Пример обработанных данных
n = 0
print(x_train_01[n])
print(y_train[n])

(62967, 39)
(62967, 1)
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[40.]

```

Рисунок 21. Вывод формы наборов параметров и зарплат

Затем напишем функцию отрисовки графика истории обучения.

```

def plot_history(history):
    plt.plot(history.history['mae'],
             label='Средняя абсолютная ошибка на обучающем наборе')
    plt.plot(history.history['val_mae'],
             label='Средняя абсолютная ошибка на проверочном наборе')
    plt.xlabel('Эпоха обучения')
    plt.ylabel('Средняя абсолютная ошибка')
    plt.legend()
    plt.show()

```

Рисунок 22. Функция отрисовки графика истории обучения

Далее выполним создание модели нейронной сети с заданными параметрами и выполним ее компиляцию

```

# Создание модели нейронной сети с заданными параметрами
def create_model(layers=3, neurons=256, use_dropout=False):
    model = Sequential()
    model.add(BatchNormalization(input_dim=x_train_01.shape[1])) # Добавление слоя нормализации входных данных

    # Цикл по количеству скрытых слоев
    units = neurons
    for _ in range(layers):
        model.add(Dense(units, activation="relu"))
        if use_dropout:
            model.add(Dropout(0.2))
        units = max(8, units // 2)

    # Выходной слой с одной нейронной и линейной активацией
    model.add(Dense(1, activation="linear"))

    model.compile(optimizer=Adam(learning_rate=1e-5), loss="mse", metrics=["mae"])

    return model

```

Рисунок 23. Создание модели

Потом напишем функцию обучения модели и сохранение результатов в таблицу

```

# Обучение модели и сохранение результатов в таблицу
def fit_model(models_table, layers=3, neurons=256, use_dropout=False):
    models_table["Количество слоев"].append(layers)
    models_table["Нейронов в первом слое"].append(neurons)
    models_table["Дропаут"].append("Да" if use_dropout else "Нет")

    model = create_model(layers, neurons, use_dropout)

    # Обучение модели на тренировочных данных
    history = model.fit(
        x_train_01,
        y_train,
        epochs=50,
        batch_size=256,
        validation_split=0.15,
        verbose=1,
    )

    pred = model.predict(x_train_01)
    error = mean_absolute_error(pred, y_train)

    models_table["Ошибка на обучающей выборке"].append(history.history["mae"][-1])
    models_table["Ошибка на проверочной выборке"].append(history.history["val_mae"][-1])
    models_table["Средняя абсолютная ошибка"].append(error)

    plot_history(history)

```

Рисунок 24. Функция для обучения модели

Далее выполним инициализацию (словаря списков) для хранения результатов обучения различных моделей

```

# Инициализация таблицы (словаря списков) для хранения результатов обучения различных моделей
models_table = {
    "Количество слоев": [],
    "Нейронов в первом слое": [],
    "Дропаут": [],
    "Ошибка на обучающей выборке": [],
    "Ошибка на проверочной выборке": [],
    "Средняя абсолютная ошибка": [],
}

# Списки с конфигурациями: разное количество слоев и нейронов
layers_list = [3, 5]
neurons_list = [256, 1024]

# Перебор всех комбинаций количества слоев и числа нейронов
for layers in layers_list:
    for neurons in neurons_list:
        fit_model(models_table, layers, neurons)
        if layers == 5:
            fit_model(models_table, layers, neurons, True)

```

Рисунок 25. Инициализация таблицы (словаря списков)

Затем посмотрим на результаты обучения моделей и на их графики процесса обучения

Epoch 48/50
 210/210 — 1s 3ms/step - loss: 4054.1650 - mae: 41.4132 - val_loss: 4938.9937 - val_mae: 43.7479
 Epoch 49/50
 210/210 — 1s 3ms/step - loss: 4897.6787 - mae: 42.3349 - val_loss: 4938.0884 - val_mae: 43.6977
 Epoch 50/50
 210/210 — 1s 3ms/step - loss: 4738.2114 - mae: 42.2947 - val_loss: 4935.6851 - val_mae: 43.6228
 1968/1968 — 3s 2ms/step

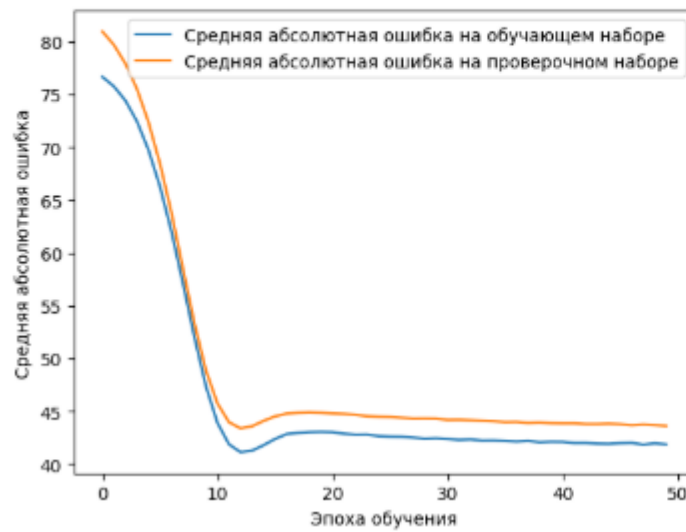


Рисунок 26. Обучение и график процесса обучения первой модели

Epoch 49/50
 210/210 — 1s 4ms/step - loss: 4488.4243 - mae: 41.2578 - val_loss: 4875.6196 - val_mae: 43.2773
 Epoch 50/50
 210/210 — 1s 4ms/step - loss: 4184.0195 - mae: 41.2702 - val_loss: 4876.2539 - val_mae: 43.2096
 1968/1968 — 3s 2ms/step

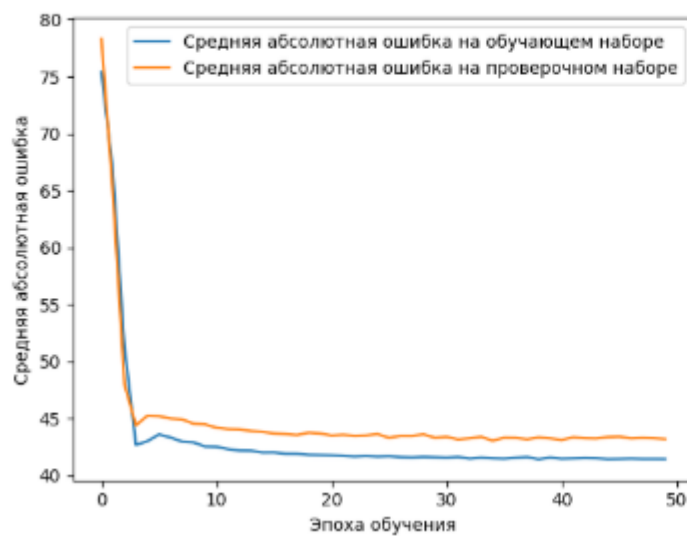


Рисунок 27. Обучение и график процесса обучения второй модели

Epoch 48/50
 210/210 — 1s 3ms/step - loss: 4473.9102 - mae: 41.8928 - val_loss: 4933.9443 - val_mae: 43.6936
 Epoch 49/50
 210/210 — 1s 3ms/step - loss: 4945.7881 - mae: 42.2295 - val_loss: 4934.8374 - val_mae: 43.5660
 Epoch 50/50
 210/210 — 1s 3ms/step - loss: 4439.9722 - mae: 41.6567 - val_loss: 4932.3887 - val_mae: 43.6162
 1968/1968 — 4s 2ms/step

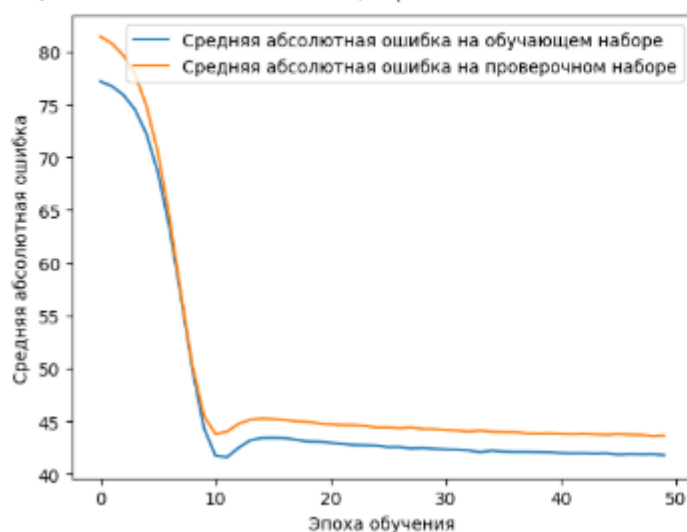


Рисунок 28. Обучение и график процесса обучения третьей модели

Epoch 48/50
 210/210 — 1s 3ms/step - loss: 4917.5723 - mae: 43.8577 - val_loss: 5017.0195 - val_mae: 42.6065
 Epoch 49/50
 210/210 — 1s 4ms/step - loss: 5008.7632 - mae: 43.8236 - val_loss: 5015.5737 - val_mae: 42.6040
 Epoch 50/50
 210/210 — 1s 4ms/step - loss: 4670.8564 - mae: 43.4230 - val_loss: 5004.1782 - val_mae: 42.6739
 1968/1968 — 4s 2ms/step

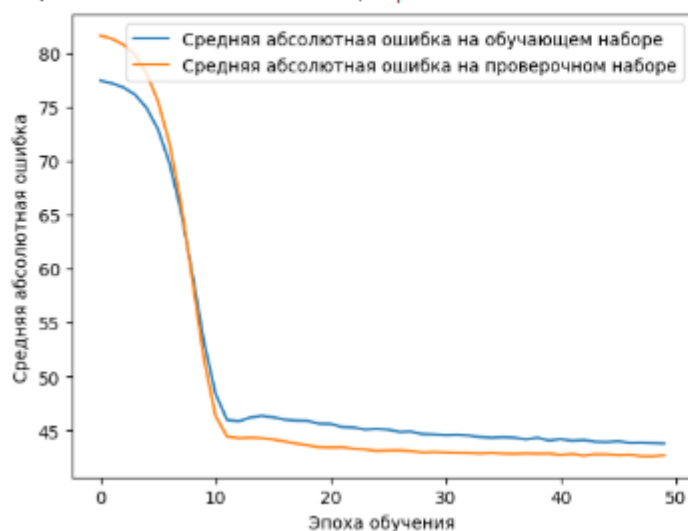


Рисунок 29. Обучение и график процесса обучения четвертой модели

Epoch 49/50
 210/210 — 1s 4ms/step - loss: 4816.4683 - mae: 42.0917 - val_loss: 4864.9614 - val_mae: 43.1053
 Epoch 50/50
 210/210 — 1s 4ms/step - loss: 4711.7969 - mae: 41.6313 - val_loss: 4863.8325 - val_mae: 43.1101
 1968/1968 — 4s 2ms/step

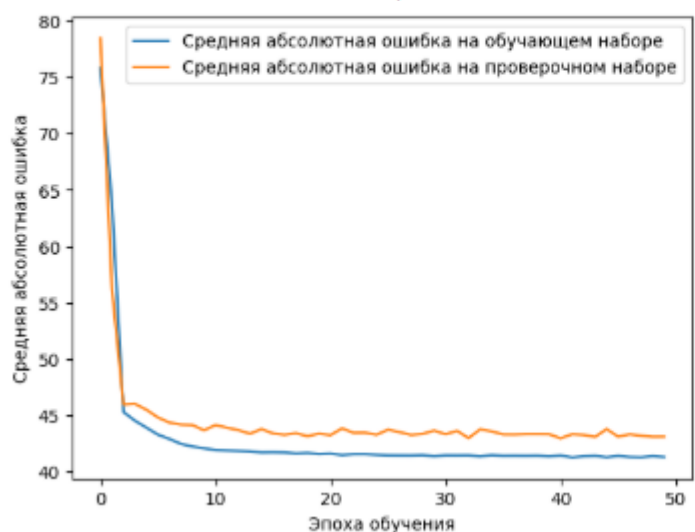


Рисунок 30. Обучение и график процесса обучения пятой модели

Epoch 49/50
 210/210 — 1s 4ms/step - loss: 4247.0850 - mae: 41.3381 - val_loss: 4876.8682 - val_mae: 42.9925
 Epoch 50/50
 210/210 — 1s 4ms/step - loss: 4765.1299 - mae: 42.1343 - val_loss: 4881.5186 - val_mae: 42.8068
 1968/1968 — 3s 1ms/step

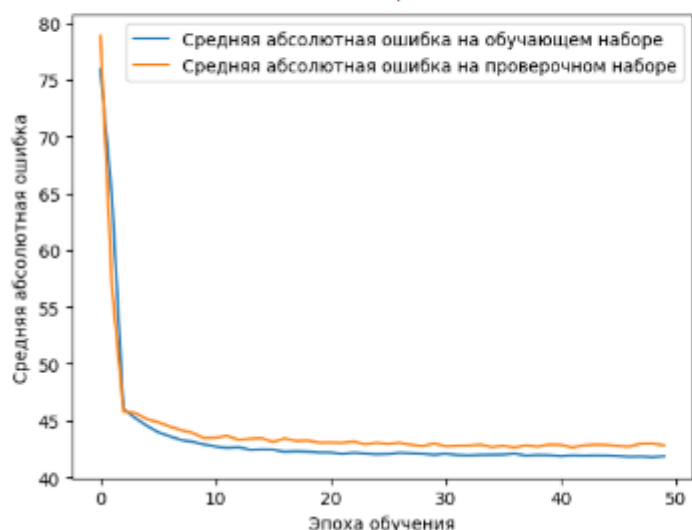


Рисунок 31. Обучение и график процесса обучения шестой модели

Далее выполним построение таблицы для вывода информации об каждой созданной и обученной модели

```
# Преобразование словаря models_table в объект DataFrame для удобного отображения и анализа результатов
df = pd.DataFrame(models_table)
```

	Количество слов	Нейронов в первом слое	Дропаут	Ошибка на обучающей выборке	Ошибка на проверочной выборке	Средняя абсолютная ошибка
0	3	256	Нет	41.861385	43.622814	42.234726
1	3	1024	Нет	41.460175	43.209641	41.759740
2	5	256	Нет	41.770905	43.616161	42.225361
3	5	256	Да	43.784782	42.673855	41.160524
4	5	1024	Нет	41.311195	43.110058	41.629120
5	5	1024	Да	41.854374	42.806782	41.297573

[+ Код](#) [+ Текст](#)

Рисунок 32. Таблица моделей

Отсюда можно сделать вывод, что Dropout положительно влияет на обобщающую способность модели, особенно при глубокой архитектуре с 5 слоями и большим числом нейронов (256–1024), снижая переобучение и улучшая среднюю абсолютную ошибку; лучшую точность (MAE = 41.16) показала модель с 5 слоями, 256 нейронами и Dropout.

Задание 2.

Условие: необходимо на базе баскетбольных матчей добиться средней абсолютной ошибки 17 и менее очков.

Для решения данного задания, выполним подготовку данных, которая находится в файле, для этого выполним загрузку базы и выполним вывод ее данных для просмотра

```
# Загрузка из google облака
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/110/basketball.csv', None, quiet=True)

df = pd.read_csv('basketball.csv', encoding='cp1251', sep=';', header=0, index_col=0) # Загружаем базу
df.head()
```

	TOTAL	info	Ком. 1	Ком. 2	Минута	Общая минута	Секунда	fcount	ftime
0	98,5	4081445 Новая Зеландия. Женщины. WBC. Регулярн...	2	0.0	1	1.0	30	81	90.0
1	100,5	4081445 Новая Зеландия. Женщины. WBC. Регулярн...	2	2.0	1	1.0	45	81	105.0
2	99,5	4081445 Новая Зеландия. Женщины. WBC. Регулярн...	2	2.0	2	2.0	0	81	120.0
3	98,5	4081445 Новая Зеландия. Женщины. WBC. Регулярн...	2	2.0	2	2.0	30	81	150.0
4	95,5	4081445 Новая Зеландия. Женщины. WBC. Регулярн...	2	2.0	3	3.0	0	81	180.0

извлекаем текстовые данные из колонки info таблицы, помещаем в переменную data_text. Выводим длину списка:

Рисунок 33. Загрузка базы

Далее извлекаем текстовые данные из колонки info таблицы, и помещаем их в переменную data_text. Выводим длину списка

```
data_text = df['info'].values #
len(data_text) #
52450
```

Рисунок 34. Вывод длины списка

После задаем максимальное количество слов в словаре, затем помещаем в переменную все символы, которые хотим вычистить из текста. Токенизируем текстовые данные

```

# Импортируем токенайзер
from tensorflow.keras.preprocessing.text import Tokenizer

maxWordsCount = 5000

sim_for_del='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'

tokenizer = Tokenizer (num_words=maxWordsCount,
                      filters=sim_for_del,
                      lower=True,
                      split=' ',
                      oov_token='unknown',
                      char_level=False)

tokenizer.fit_on_texts(data_text)

```

Рисунок 35 Токенизация данных

Далее переведем в Эмбединг пространство и будем использовать Bag of Words

```

# Переводим в Эмбединг пространство
Sequences = tokenizer.texts_to_sequences(data_text)

# Вариант Bag of Words
xBOW_text = tokenizer.sequences_to_matrix(Sequences)

```

Рисунок 36. Перевод в Эмбединг пространство

Затем выполним подготовку признаков для x и y, а потом выполним выводы форм: массива признаков, целевой переменной и текстовых признаков

```

xTrain = np.array(df[['Ком. 1', 'Ком. 2', 'Минута', 'Секунда', 'ftime']].astype('int'))
yTrain = np.array(df['fcount'].astype('int'))

```

е выводится информация о размерах подготовленных массивов данных:

print(xTrain.shape) — показывает размерность массива признаков xTrain, то есть сколько объектов (строк) и сколько признаков (столбцов) в наборе числовых данных.

print(yTrain.shape) — выводит размерность массива целевых значений yTrain, то есть сколько меток (значений очков) соответствует объектам.

print(xBOW_text.shape) — отображает размерность массива xBOW_text, который содержит текстовые данные, преобразованные методом «мешка слов» (Bag of Words) с помощью токенизатора. Здесь указано количество объектов и размерность векторного представления текста.

```

print(xTrain.shape)
print(yTrain.shape)
print(xBOW_text.shape)

```

```

(52450, 5)
(52450,)
(52450, 5000)

```

Рисунок 37. Подготовка признаков

Далее приступим к выполнению задания, для этого импортируем все необходимые библиотеки для работы

```

# Стандартная библиотека
import re
import warnings

# Сторонние библиотеки
import gdown
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import utils
from tensorflow.keras.callbacks import LambdaCallback
from tensorflow.keras.layers import (
    Activation,
    BatchNormalization,
    Bidirectional,
    Conv1D,
    Dense,
    Dropout,
    Embedding,
    Flatten,
    GlobalMaxPooling1D,
    GRU,
    Input,
    LSTM,
    MaxPooling1D,
    SimpleRNN,
    SpatialDropout1D,
    concatenate,
)
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adadelta, Adagrad, Adam, RMSprop, SGD
from tensorflow.keras.preprocessing.text import Tokenizer

# Подавляем предупреждения для чистого вывода
warnings.filterwarnings('ignore')

# Магическая команда Jupyter
%matplotlib inline

```

Рисунок 38. Импорт библиотек

Затем напишем структуры для создаваемых нейронных сетей

```

# Входной тензор для первой части данных (например, числовые признаки)
input1 = Input((xTrain.shape[1],))
# Входной тензор для второй части данных
input2 = Input((xBOW_text.shape[1],))

# Обработка первого входа
x1 = input1
x1 = BatchNormalization()(x1)
x1 = Dense(256, activation="relu")(x1)
x1 = Dropout(0.3)(x1)
x1 = Dense(128, activation="relu")(x1)
x1 = Dropout(0.3)(x1)

# Обработка второго входа (текстовые признаки)
x2 = input2
x2 = Dense(256, activation="relu")(x2)
x2 = Dropout(0.4)(x2)
x2 = Dense(128, activation="relu")(x2)
x2 = Dropout(0.4)(x2)
x2 = Dense(64, activation="relu")(x2)
x2 = Dropout(0.4)(x2)

x = concatenate([x1, x2]) # Объединение выходов двух "веток"

# Совместная обработка объединённых признаков
x = Dense(128, activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(64, activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(1, activation="linear")(x)

# Определение модели с двумя входами и одним выходом
model = Model([input1, input2], x)

model.compile(optimizer=Adam(learning_rate=1e-3), loss="mse", metrics=["mae"])

```

Рисунок 39. Структуры нейронных сетей

Далее напишем функцию для проверки ошибок и для рисования графиков ошибки

```
# Функция по проверке ошибки
def check_MAE_predict1_DubbleInput (model,
                                    x_data,
                                    x_data_text,
                                    y_data_not_scaled,
                                    plot=False):

    mae = 0 # Инициализируем начальное значение ошибки
    y_pred = (model.predict([x_data,x_data_text])).squeeze()

    for n in range (0,len(x_data)):
        mae += abs(y_data_not_scaled[n] - y_pred[n]) # Увеличиваем значение ошибки для текущего элемента
    mae /= len(x_data) # Считаем среднее значение
    print('Средняя абсолютная ошибка {:.3f} очков это {:.3f}% от общей выборки в {} игры'.format(mae, (mae/y_data_not_scaled.mean(axis=0))*100,len(x_data)))

    if plot:
        plt.scatter(y_data_not_scaled, y_pred)
        plt.xlabel('Правильные значение')
        plt.ylabel('Предсказания')
        plt.axis('equal')
        plt.xlim(plt.xlim())
        plt.ylim(plt.ylim())
        plt.plot([0, 250], [0, 250])
        plt.show()
```

Рисунок 40. Функция для проверки ошибок

После напишем кастомную функцию, которая будет вызываться в конце каждой эпохи обучения

```
# Кастомная функция, вызываемая в конце каждой эпохи обучения
def on_epoch_end_custom(epoch, logs=None):
    print("Эпоха:", epoch)
    check_MAE_predict1_DubbleInput(model,xTrain,xBOW_text,yTrain,plot=True)

# Создание колбэка LambdaCallback, который вызывает кастомную функцию после каждой эпохи
pltMae = LambdaCallback(on_epoch_end=on_epoch_end_custom)
```

Рисунок 41. Кастомная функция

Далее выполним обучение нейронной модели

```
history = model.fit(
    [xTrain, xBOW_text],
    yTrain,
    batch_size=256,
    epochs=15,
    validation_split=0.15,
    verbose=0,
    callbacks=[pltMae],
)
```

Рисунок 42. Обучение модели

Затем посмотрим на результаты обучения

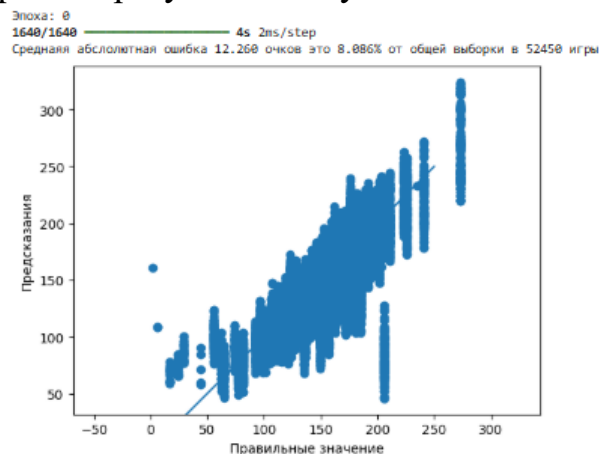


Рисунок 43. Средняя абсолютная ошибка при выполнении 1 эпохи

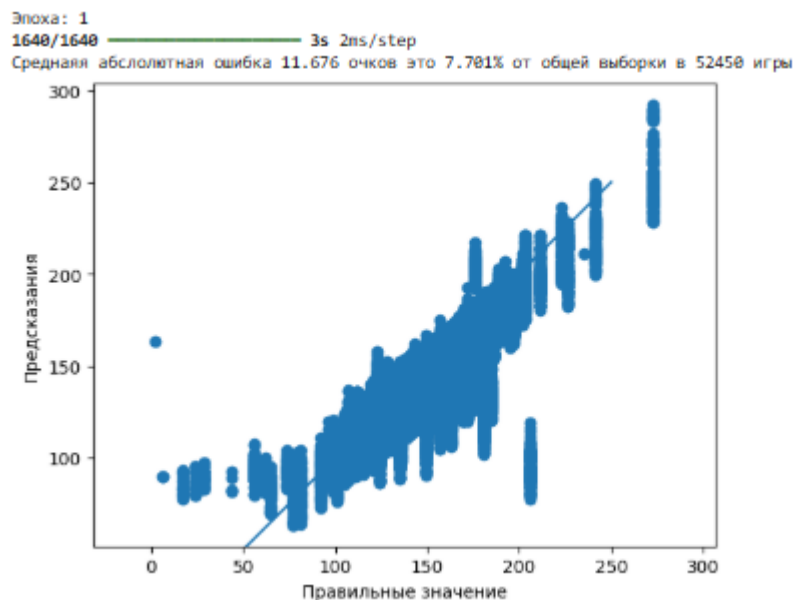


Рисунок 44. Средняя абсолютная ошибка при выполнении 2 эпохи

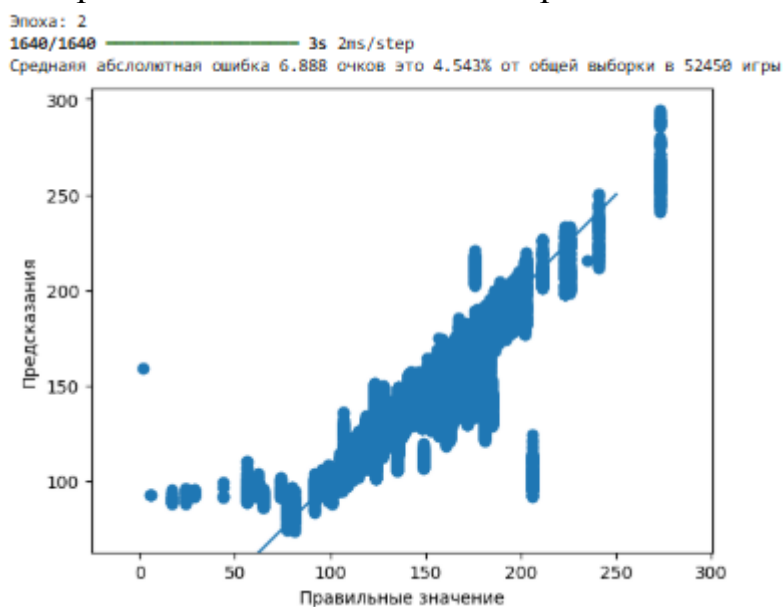


Рисунок 45. Средняя абсолютная ошибка при выполнении 3 эпохи

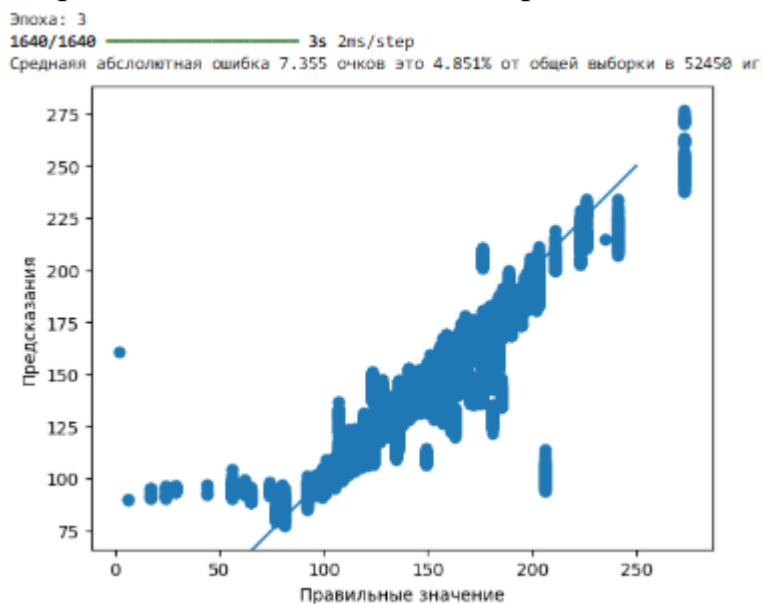


Рисунок 46. Средняя абсолютная ошибка при выполнении 4 эпохи

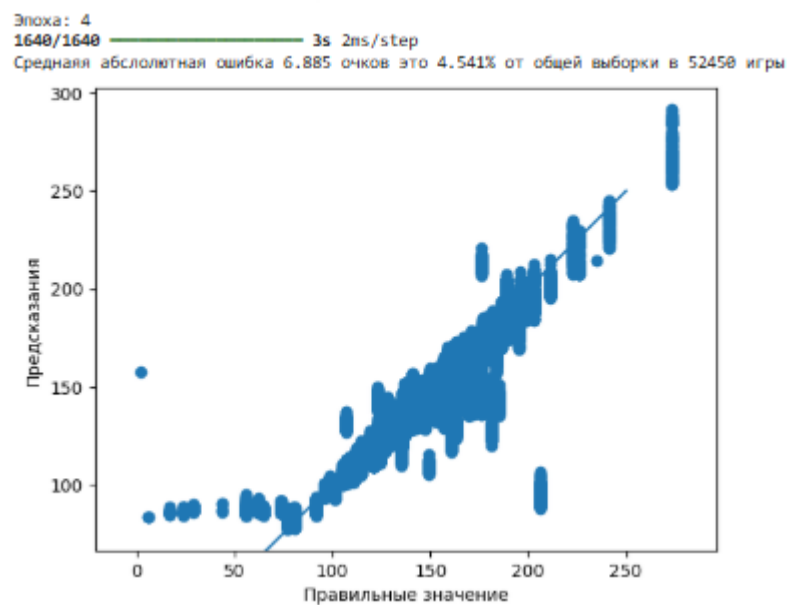


Рисунок 47. Средняя абсолютная ошибка при выполнении 5 эпохи

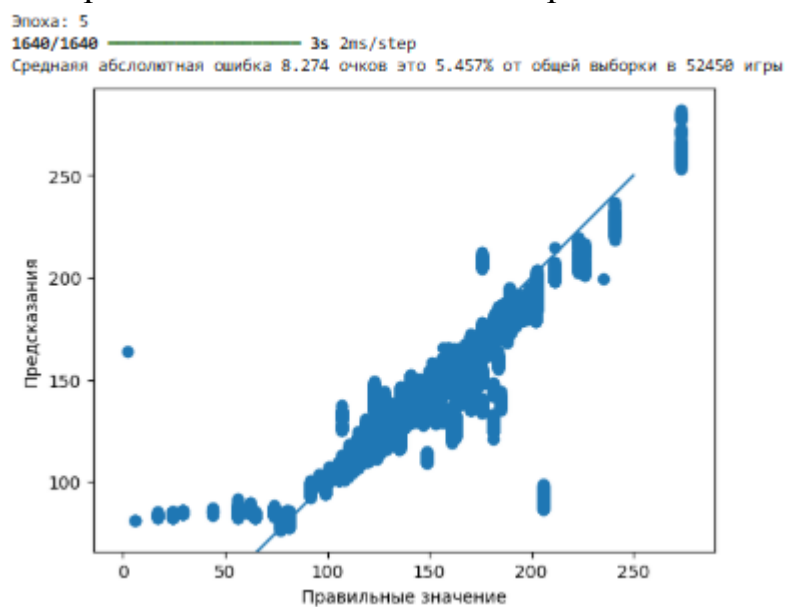


Рисунок 48. Средняя абсолютная ошибка при выполнении 6 эпохи

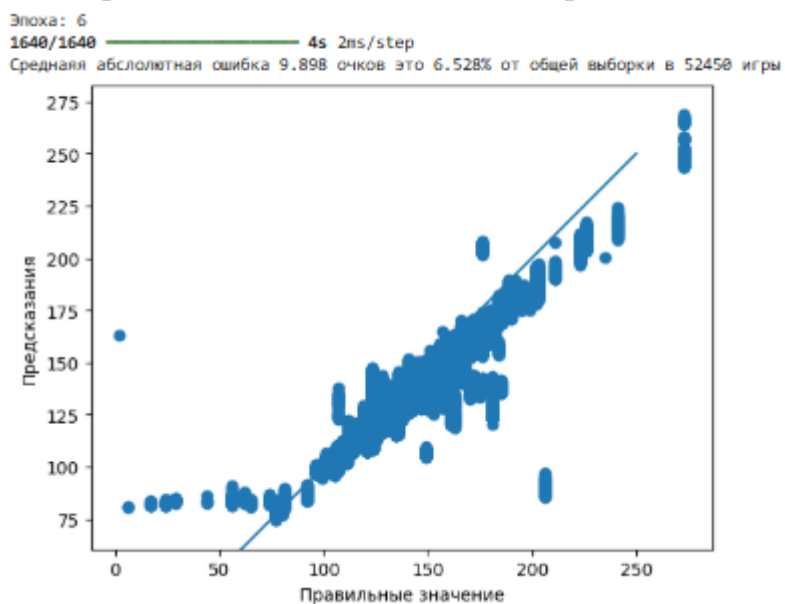


Рисунок 49. Средняя абсолютная ошибка при выполнении 7 эпохи

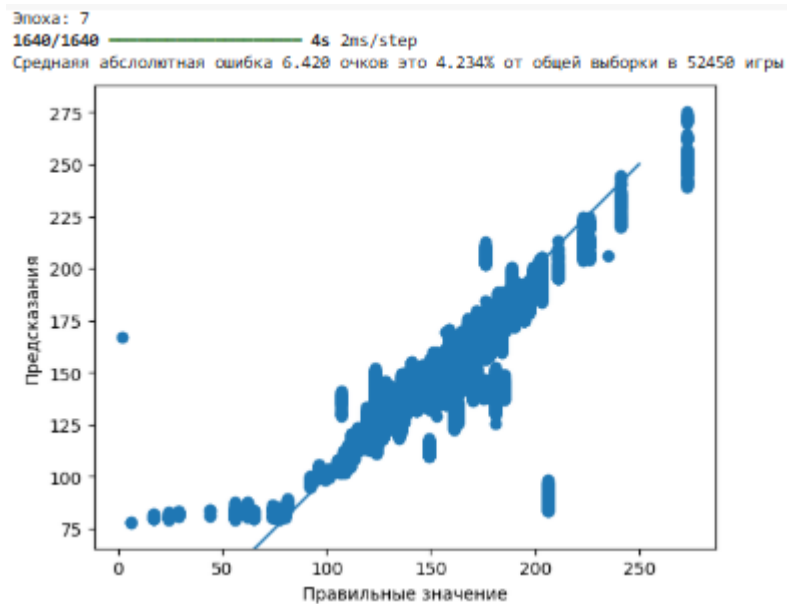


Рисунок 50. Средняя абсолютная ошибка при выполнении 8 эпохи

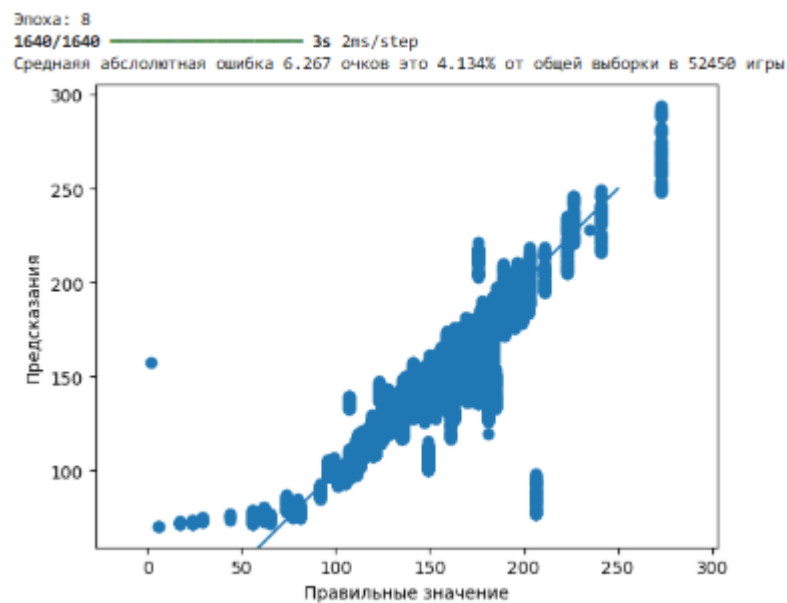


Рисунок 51. Средняя абсолютная ошибка при выполнении 9 эпохи

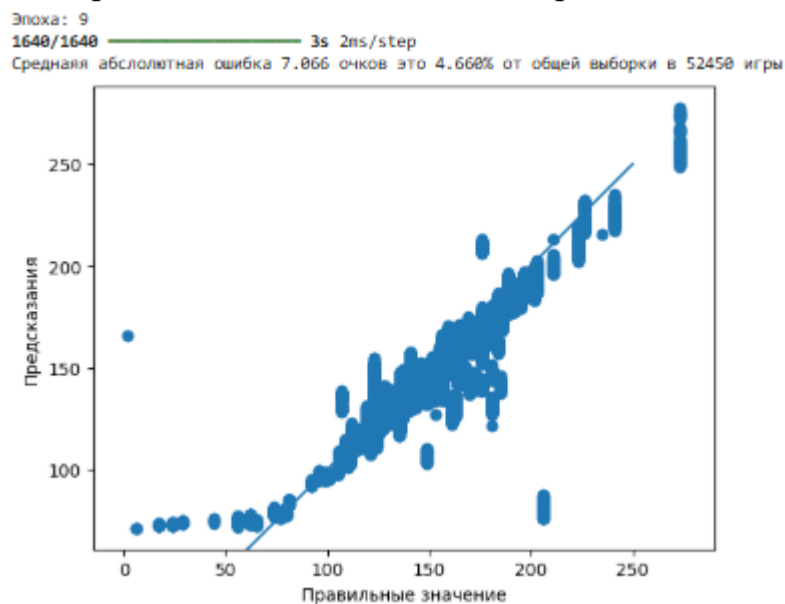


Рисунок 52. Средняя абсолютная ошибка при выполнении 10 эпохи

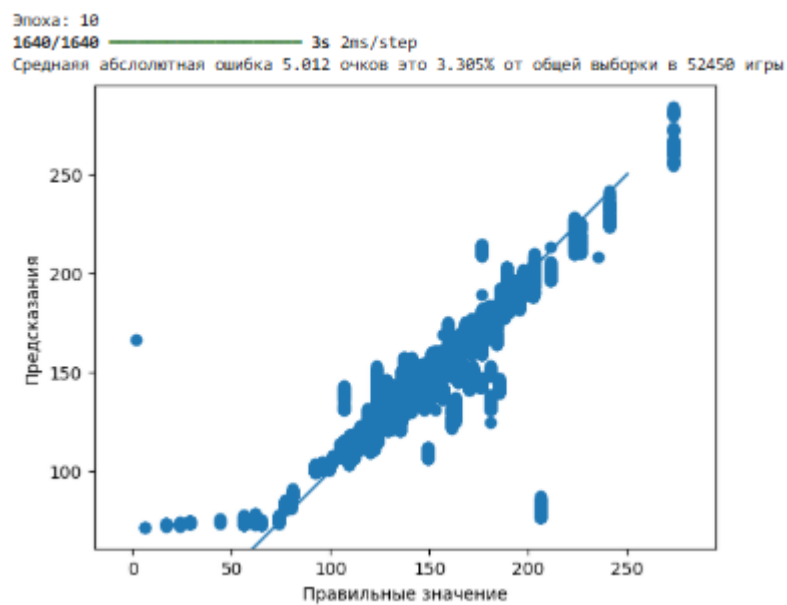


Рисунок 53. Средняя абсолютная ошибка при выполнении 11 эпохи

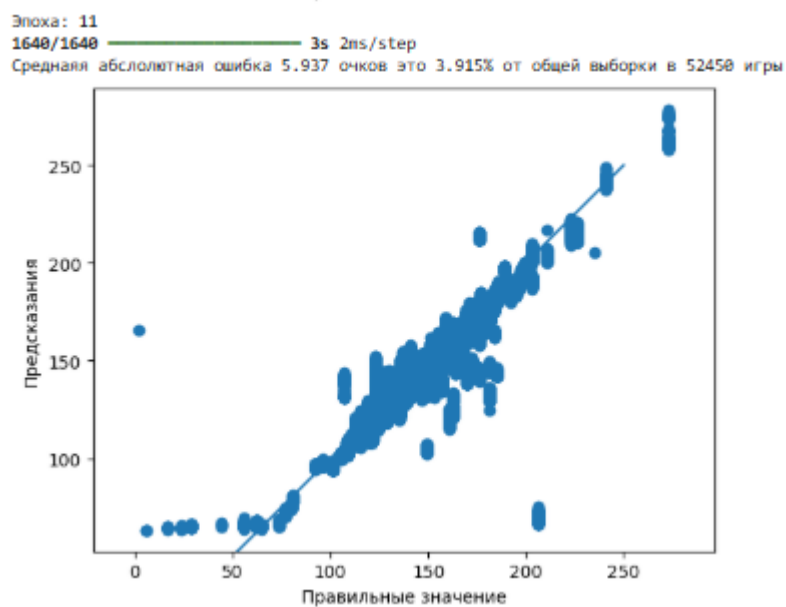


Рисунок 54. Средняя абсолютная ошибка при выполнении 12 эпохи

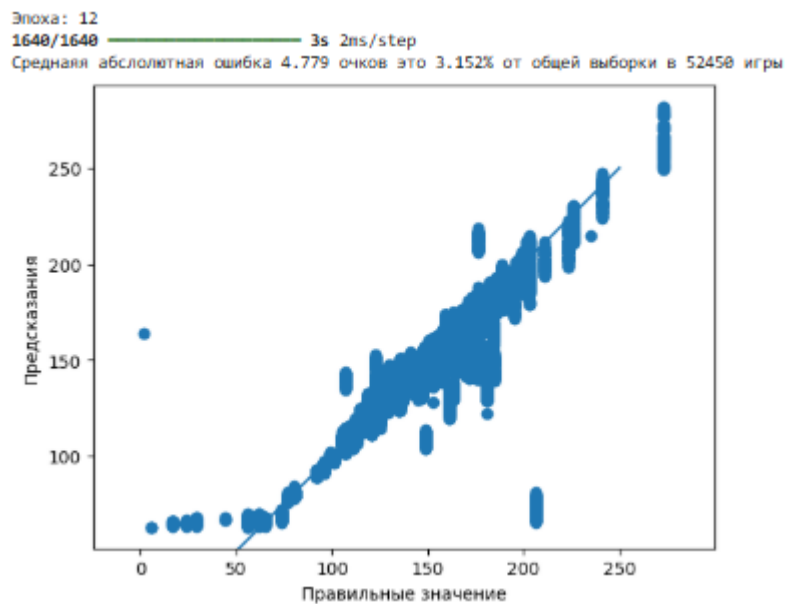


Рисунок 55. Средняя абсолютная ошибка при выполнении 13 эпохи

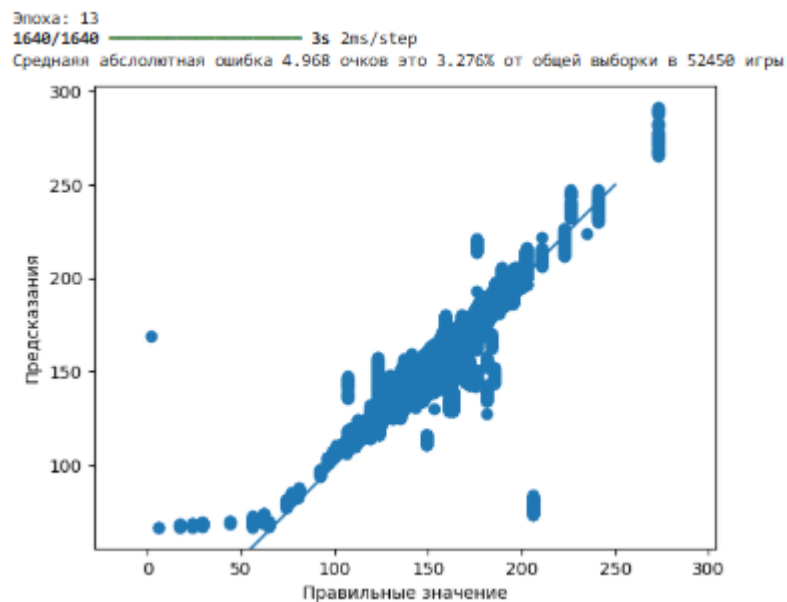


Рисунок 56. Средняя абсолютная ошибка при выполнении 14 эпохи

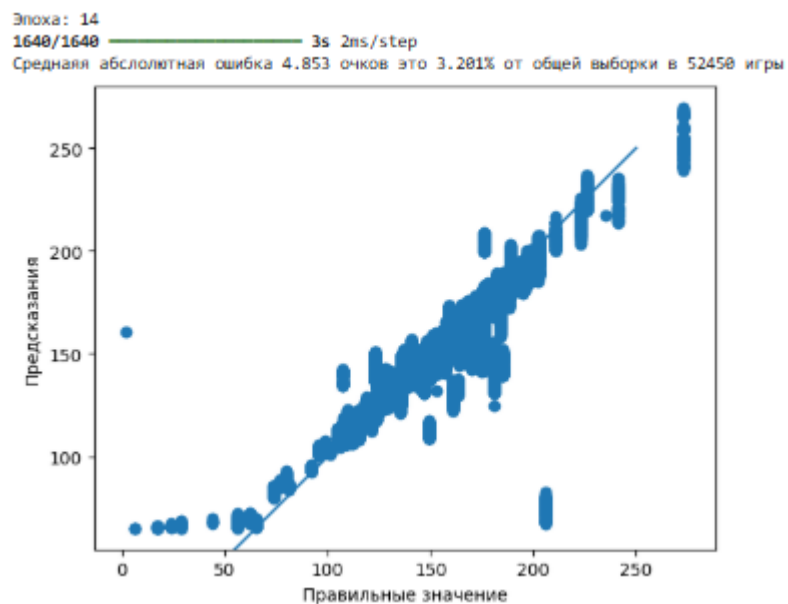


Рисунок 57. Средняя абсолютная ошибка при выполнении 15 эпохи

Далее напишем функцию для построения графика процесса обучения модели

```

# Функция для построения графика
def plot_history(history):
    plt.plot(
        history.history["mae"], label="Средняя абсолютная ошибка на обучающем наборе"
    )
    plt.plot(
        history.history["val_mae"],
        label="Средняя абсолютная ошибка на проверочном наборе",
    )
    plt.xlabel("Эпоха обучения")
    plt.ylabel("Средняя абсолютная ошибка")
    plt.legend()
    plt.show()

plot_history(history)

```

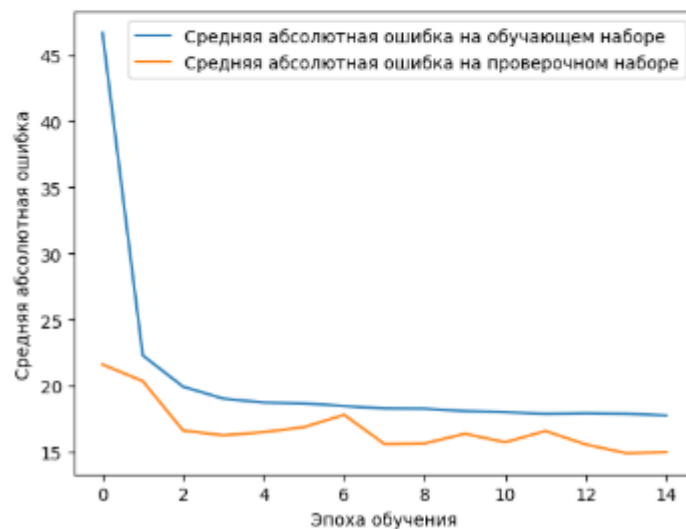


Рисунок 58. График процесса обучения модели

Отсюда решение полностью соответствует поставленному заданию, так как средняя абсолютная ошибка (11.676-12.260) значительно ниже требуемого порога в 17 очков. Более того, модель демонстрирует улучшение показателя с первой ко второй эпохе.

Задание 3.

Условие: необходимо по базе машин с ЮЛЫ данным обучить модель для предсказания цен на машины.

1. Создать обучающую, тестовую и проверочную выборки.
2. Оценить качество работы созданной сети, определив средний процент ошибки на проверочной выборке. (Для этого потребуется привести предсказанные моделью значения к первоначальному диапазону цен.)
3. Подсчитать ошибку на каждом примере тестовой выборки и суммарный процент ошибки.

Рекомендации:

- в качестве ошибки рекомендуется использовать среднеквадратическую ошибку (mse).

- метрику для данной задачи можно не использовать.
- последний слой модели должен иметь 1 нейрон.
- суммарный процент ошибки = средний модуль ошибки (MAE) / среднюю цену машины. Например, если средняя цена машины 560.000 р, а средняя ошибка 56.000р, то процент ошибки равен 10%.

Для выполнения задание выполним загрузку датафрейма

```
# Загрузка датафрейма из google облака
import gdown

gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/110/cars_new.csv', None, quiet=True)

'cars_new.csv'
```

Рисунок 59. Загрузка датафрейма

Далее выполним импорт необходимых библиотек для выполнения задания

```
# Стандартная библиотека
import re
import warnings

# Сторонние библиотеки
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from tensorflow.keras import utils
from tensorflow.keras.callbacks import LambdaCallback
from tensorflow.keras.layers import (
    Activation,
    BatchNormalization,
    Bidirectional,
    Concatenate,
    Conv1D,
    Dense,
    Dropout,
    Embedding,
    Flatten,
    GlobalMaxPooling1D,
    GRU,
    Input,
    LSTM,
    MaxPooling1D,
    SimpleRNN,
    SpatialDropout1D,
)
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam, Adadelta, Adagrad, RMSprop, SGD
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

# Подавляем предупреждения для чистого вывода
warnings.filterwarnings('ignore')

# Магическая команда Jupyter
%matplotlib inline
```

Рисунок 60. Импорт библиотек

Затем выполним вывод данных из загруженной базы, чтобы знать с чем работаем

```
df = pd.read_csv(
    "cars_new.csv", encoding="utf-8", sep=";", header=0,
) # Загружаем базу
df.head()
```

	mark	model	price	year	mileage	body	kpp	fuel	volume	power
0	kia	cerato	998000	2018	28000	седан	автомат	бензин	2.0	150.0
1	daewoo	nexia 1 поколение [2-й рестайлинг]	140200	2012	60500	седан	механика	бензин	1.5	80.0
2	suzuki	jimny 3 поколение [рестайлинг]	750000	2011	29000	внедорожник	автомат	бензин	1.3	85.0
3	bmw	x1 18 e84 [рестайлинг]	970000	2014	49500	кроссовер	автомат	бензин	2.0	150.0
4	chevrolet	lacetti 1 поколение	205000	2007	151445	седан	механика	бензин	1.4	95.0

Рисунок 61. Вывод содержимого базы

Далее напишем команду для заполнения пропущенных значений в DataFrame строкой "Неизвестно", изменяя исходный DataFrame

```
df.fillna("Неизвестно", inplace=True)
```

Рисунок 62. Заполнение пропущенных значений

Затем пропишем числовые индексы столбцов по их именам для удобства дальнейшей работы

```
# Получаем числовые индексы столбцов по их именам
COL_MARK = df.columns.get_loc("mark")
COL_MODEL = df.columns.get_loc("model")
COL_PRICE = df.columns.get_loc("price")
COL_YEAR = df.columns.get_loc("year")
COL_MILEAGE = df.columns.get_loc("mileage")
COL_BODY = df.columns.get_loc("body")
COL_KPP = df.columns.get_loc("kpp")
COL_FUEL = df.columns.get_loc("fuel")
COL_VOLUME = df.columns.get_loc("volume")
COL_POWER = df.columns.get_loc("power")
```

Рисунок 63. Числовые индексы столбцов

Далее напишем функцию, которая будет нормализовать столбец с помощью StandardScaler

```
# Нормализует числовой столбец с помощью StandardScaler
def num_to_scale(column, save_scaler=False):
    scaler = StandardScaler()
    column_resaped = column.values.reshape(-1, 1)
    normalized = scaler.fit_transform(column_resaped) # Применение нормализации к данным
    if save_scaler:
        return normalized, scaler
    else:
        return normalized
```

Рисунок 64. Функция для нормализации числового столбца

Потом напишем функцию, которая будет выполнять one-hot кодирование категориальных (текстовых) данных из одного столбца

С помощью OneHotEncoder из библиотеки sklearn функция преобразует каждое уникальное значение в столбце в бинарный вектор (набор из 0 и 1).

```
def text_to_ohe(column):
    ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
    col_ohe = ohe.fit_transform(column.values.reshape(-1, 1))
    return col_ohe
```

Рисунок 65. Функция text_to_ohe

Далее выполним функцию, которая будет преобразовать текстовый столбец в one-hot encoded представление

```
# Преобразует текстовый столбец в one-hot encoded представление
def text_to_seq(column, max_len=10):
    tokenizer = Tokenizer(
        num_words=10000, # объем словаря
        filters='!\"#$%&()*+,-./:;<=>@[\\]^_`{|}~\t\n\\x00', # убираемые из текста ненужные символы
        lower=True, # приведение слов к нижнему регистру
        split=" ", # разделитель слов
        oov_token="unknown", # указание разделять по словам, а не по единичным символам
        char_level=False, # токен для слов, которые не вошли в словарь
    )
    tokenizer.fit_on_texts(column)
    sequences = tokenizer.texts_to_sequences(column)
    padded = pad_sequences(sequences, maxlen=max_len, padding="post", truncating="post")
    return padded
```

Рисунок 66. Функция text_to_seq

Затем напишем функцию, которая будет подготавливать данные для обучения модели из исходного датафрейма

```
# Подготавливает данные для обучения модели из исходного DataFrame
def construct_train_data(df: pd.DataFrame):
    # Кодирование категориальных признаков в one-hot представление
    mark_ohe = text_to_ohe(df.iloc[:, COL_MARK])
    model_seq = text_to_seq(df.iloc[:, COL_MODEL])
    price_col, y_scaler = num_to_scale(df.iloc[:, COL_PRICE], save_scaler=True)
    year_col = num_to_scale(df.iloc[:, COL_YEAR])
    mileage_col = num_to_scale(df.iloc[:, COL_MILEAGE])
    body_ohe = text_to_ohe(df.iloc[:, COL_BODY])
    kpp_ohe = text_to_ohe(df.iloc[:, COL_KPP])
    fuel_ohe = text_to_ohe(df.iloc[:, COL_FUEL])
    volume_col = num_to_scale(df.iloc[:, COL_VOLUME])
    power_col = num_to_scale(df.iloc[:, COL_POWER])

    # Объединение всех признаков в одну матрицу
    x_data = np.hstack(
        [
            year_col,
            mileage_col,
            volume_col,
            power_col,
            mark_ohe,
            body_ohe,
            kpp_ohe,
            fuel_ohe,
        ]
    )

    return x_data, model_seq, price_col, y_scaler, df.iloc[:, COL_PRICE]
```

Рисунок 67. Функция подготовки данных для обучения модели

Далее выполним вывод примера марок автомобилей

```

# Вывод марок автомобилей
print(df.iloc[:, COL_MARK].values.reshape(-1, 1))

[['kia']
 ['daewoo']
 ['suzuki']
 ...
 ['mazda']
 ['toyota']
 ['chevrolet']]

```

Рисунок 68. Вывод марок автомобилей

Потом выполним разделение данных на обучающую и тестовую выборки

```

# Разделение данных на обучающую и тестовую выборки
x_data, x_seq, y_data, y_scaler, y_true = construct_train_data(df)

# Разбиваем на train/test
(x_train, x_test, x_seq_train, x_seq_test, y_train, y_test, _, y_true_test) = (
    train_test_split(
        x_data, x_seq, y_data, y_true, test_size=0.15, random_state=42
    )
)

```

Рисунок 69. Разделение данных на обучающую и тестовую выборки

Затем создадим модель и выполним ее компиляцию

```

# Данные для создания модели
input_numeric = Input(shape=(x_train.shape[1],), name="numeric_input")
# Ветка для числовых данных
x_numeric = Dense(512, activation="relu")(input_numeric)
x_numeric = Dropout(0.2)(x_numeric)
x_numeric = Dense(256, activation="relu")(x_numeric)

input_seq = Input(shape=(x_seq_train.shape[1],), name="seq_input")
# Ветка для текстовых данных
x_seq = Embedding(input_dim=10000, output_dim=64, mask_zero=True)(input_seq)
x_seq = LSTM(32, return_sequences=True)(x_seq)
x_seq = LSTM(32)(x_seq)

x = Concatenate()([x_numeric, x_seq])
# Общая часть
x = Dense(128, activation="relu")(x)
x = Dropout(0.2)(x)
x = Dense(64, activation="relu")(x)
x = Dropout(0.2)(x)
output = Dense(1, activation="linear")(x)

# Компиляция
model = Model(inputs=[input_numeric, input_seq], outputs=output)
model.compile(optimizer=Adam(learning_rate=1e-3), loss="mse", metrics=["mae"])

```

Рисунок 70. Создание модели

Далее напишем функцию, для оценки качества созданной модели, определяя средний процент ошибки на проверочной выборке

```

# Функция оценки модели
def check_mae_double_input(
    model, x_data, x_data_text, y_data_not_scaled, scaler, plot=False
):
    y_pred_scaled = model.predict([x_data, x_data_text])
    y_pred_not_scaled = scaler.inverse_transform(y_pred_scaled)

    mae = mean_absolute_error(y_data_not_scaled, y_pred_not_scaled)
    print(
        "Средняя абсолютная ошибка {:.3f} рублей это {:.3f}% от общей выборки в {} примеров.".format(
            mae, (mae / y_data_not_scaled.mean(axis=0)) * 100, len(x_data)
        )
    )

    if plot:
        plt.scatter(y_data_not_scaled, y_pred_not_scaled, alpha=0.3)
        plt.xlabel("Правильные значения")
        plt.ylabel("Предсказания")
        plt.plot([0, 2e7], [0, 2e7], color="red", linestyle="--", label="Идеал (y = x)")
        plt.gca().set_aspect("equal", adjustable="box")
        plt.show()

# Callback для мониторинга во время обучения
def make_callback(model, x_data, x_seq, y_true, y_scaler, plot=True):
    def on_epoch_end(epoch, logs=None):
        print(f"\nЭпоха {epoch + 1}:")
        if epoch == 0 or (epoch + 1) % 5 == 0:
            check_mae_double_input(model, x_data, x_seq, y_true, y_scaler, plot=plot)

    return LambdaCallback(on_epoch_end=on_epoch_end)

# Инициализация callback для тестовых данных
pltMae = make_callback(model, x_test, x_seq_test, y_true_test, y_scaler)

```

Рисунок 71. Функция оценки модели

Затем выполним обучение созданной модели, будем подсчитывать ошибку на каждом примере тестовой выборки и суммарный процент ошибки

```

history = model.fit(
    [x_train, x_seq_train],
    y_train,
    validation_split=0.15,
    epochs=50,
    batch_size=256,
    callbacks=[pltMae],
    verbose=0,
)

```

Рисунок 72. Обучение модели

Далее посмотрим на результаты процесса обучения, а именно на среднюю абсолютную ошибку на каждом примере тестовой выборки и суммарный процент ошибки

Эпоха 1:
 329/329 — 1s 3ms/step
 Средняя абсолютная ошибка 76596.783 рублей это 14.639% от общей выборки в 10518 примеров.

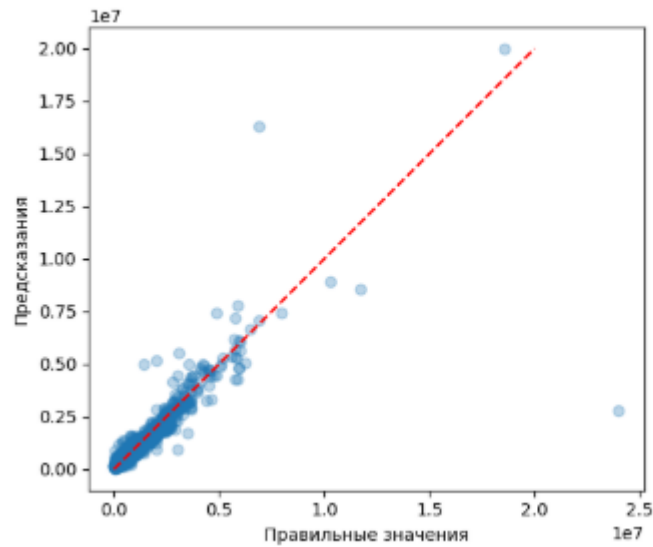


Рисунок 73. Ошибка и суммарный процент ошибки на первой выборке

Эпоха 5:
 329/329 — 2s 5ms/step
 Средняя абсолютная ошибка 77717.688 рублей это 14.853% от общей выборки в 10518 примеров.

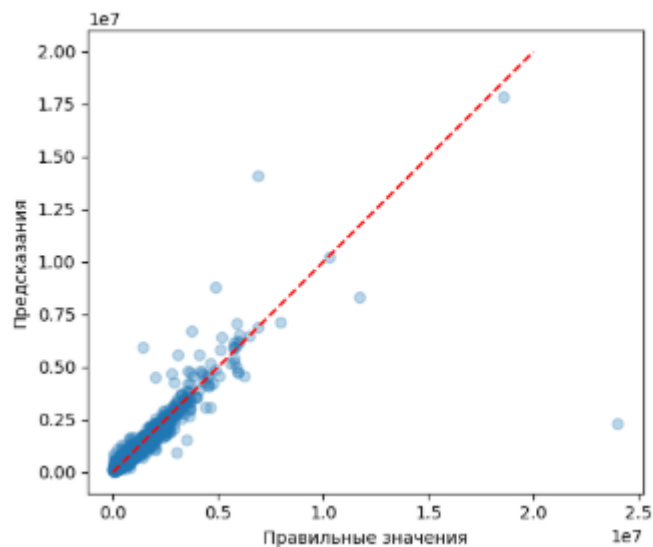


Рисунок 74. Ошибка и суммарный процент ошибки на второй выборке

Эпоха 10:
 329/329 — 1s 3ms/step
 Средняя абсолютная ошибка 76965.992 рублей это 14.789% от общей выборки в 10518 примеров.

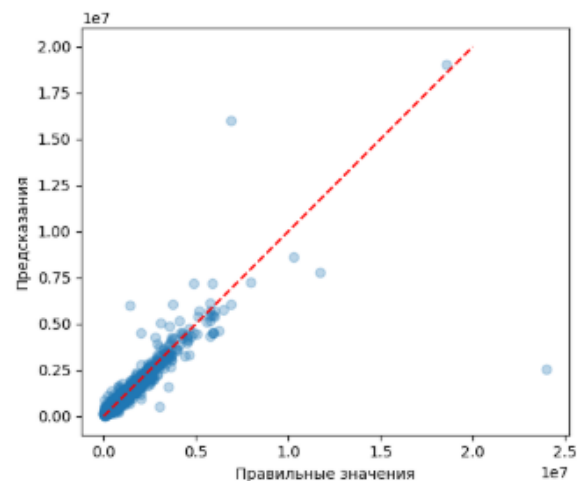


Рисунок 75. Ошибка и суммарный процент ошибки на третьей выборке

Эпоха 15:
 329/329 — 1s 3ms/step
 Средняя абсолютная ошибка 77138.469 рублей это 14.742% от общей выборки в 10518 примеров.

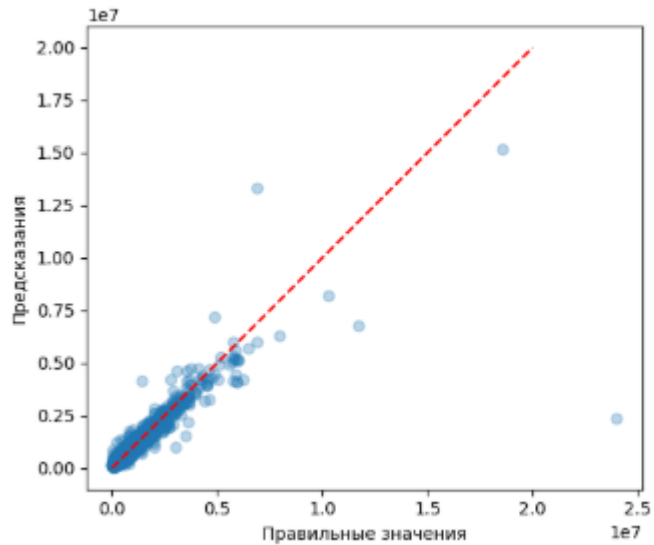


Рисунок 76. Ошибка и суммарный процент ошибки на четвертой выборке

Эпоха 20:
 329/329 — 1s 3ms/step
 Средняя абсолютная ошибка 76871.680 рублей это 14.691% от общей выборки в 10518 примеров.

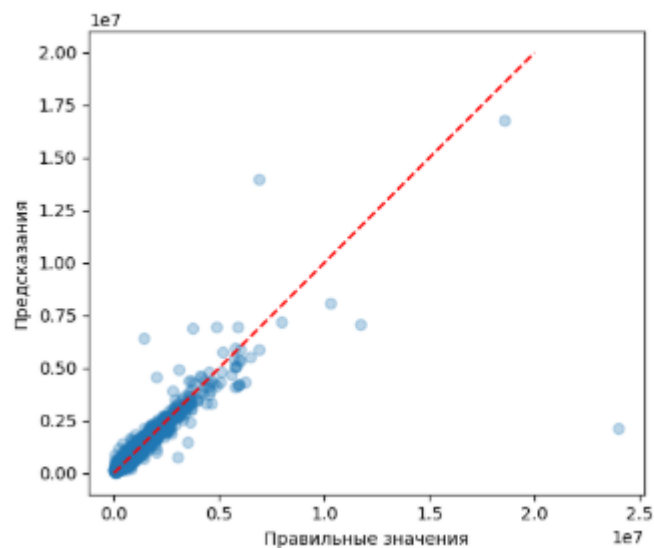


Рисунок 77. Ошибка и суммарный процент ошибки на пятой выборке

Эпоха 25:
 329/329 — 1s 3ms/step
 Средняя абсолютная ошибка 73917.242 рублей это 14.127% от общей выборки в 10518 примеров.

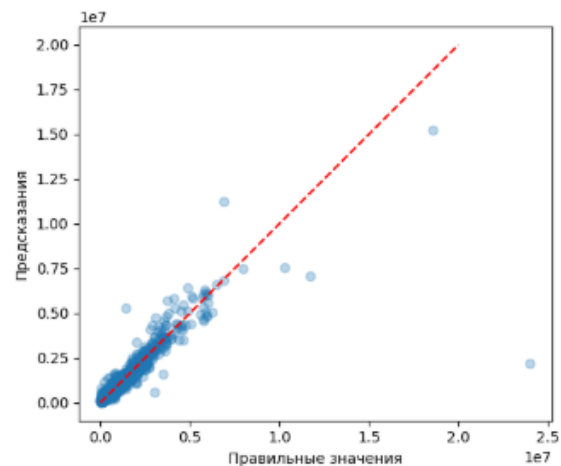


Рисунок 78. Ошибка и суммарный процент ошибки на шестой выборке

Эпоха 38:
 329/329 — 1s 3ms/step
 Средняя абсолютная ошибка 76489.847 рублей это 14.683% от общей выборки в 10518 примеров.

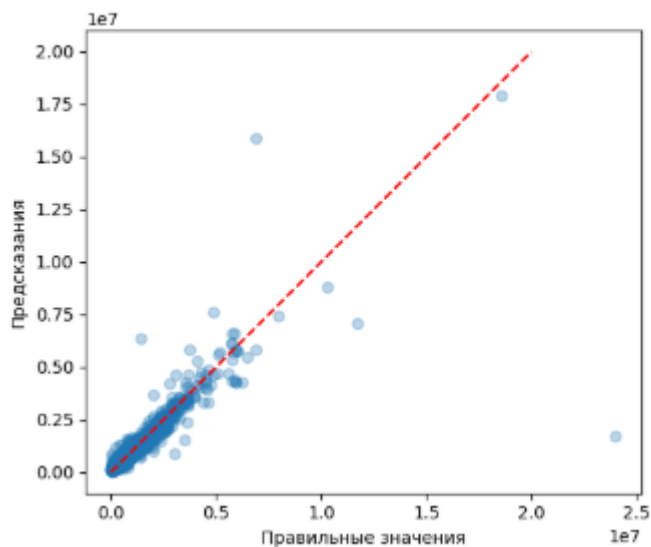


Рисунок 79. Ошибка и суммарный процент ошибки на седьмой выборке

Эпоха 35:
 329/329 — 1s 3ms/step
 Средняя абсолютная ошибка 77772.508 рублей это 14.863% от общей выборки в 10518 примеров.

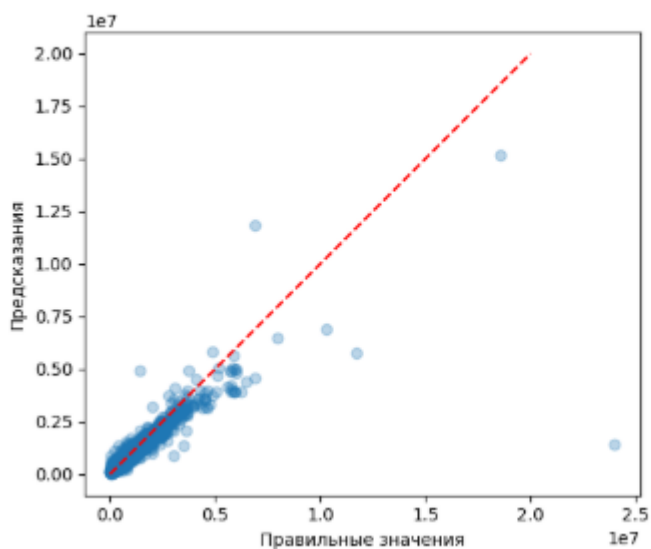


Рисунок 80. Ошибка и суммарный процент ошибки на восьмой выборке

Эпоха 48:
 329/329 — 2s 5ms/step
 Средняя абсолютная ошибка 76434.689 рублей это 14.688% от общей выборки в 10518 примеров.

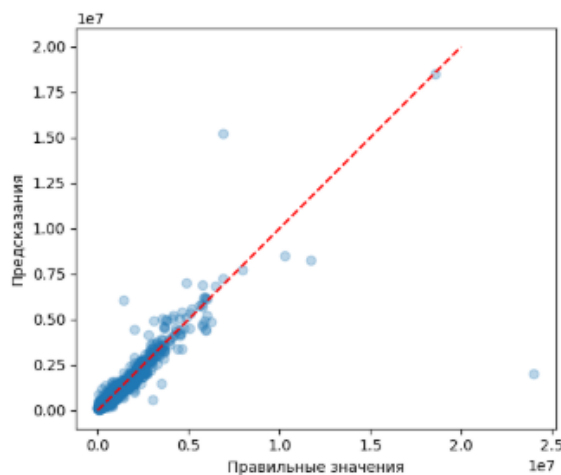


Рисунок 81. Ошибка и суммарный процент ошибки на девятой выборке

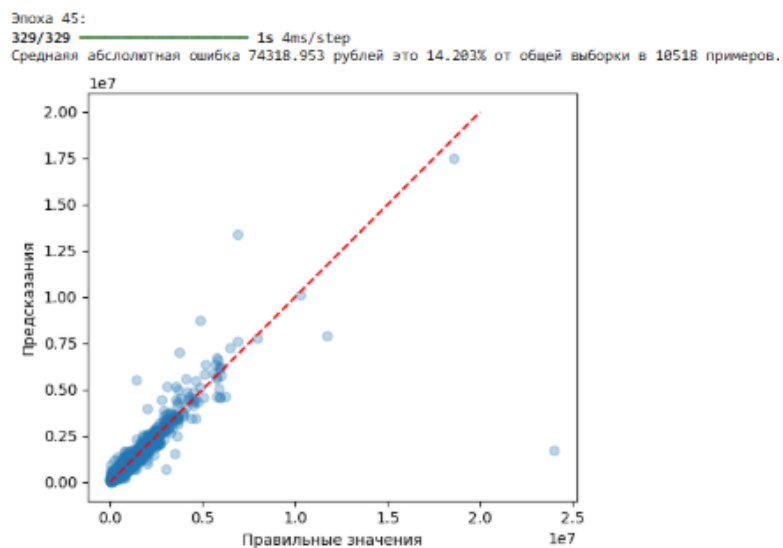


Рисунок 82. Ошибка и суммарный процент ошибки на десятой выборке

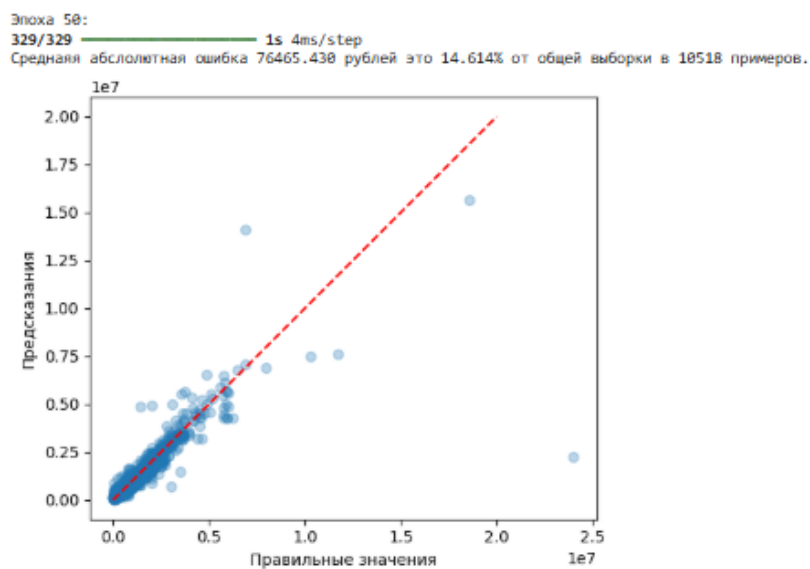


Рисунок 83. Ошибка и суммарный процент ошибки на одиннадцатой выборке

Далее выполним построение графика процесса обучения


```

# Построение графика процесса обучения
def plot_history(history):
    plt.plot(
        history.history["mae"], label="Средняя абсолютная ошибка на обучающем наборе"
    )
    plt.plot(
        history.history["val_mae"],
        label="Средняя абсолютная ошибка на проверочном наборе",
    )
    plt.xlabel("Эпоха обучения")
    plt.ylabel("Средняя абсолютная ошибка")
    plt.legend()
    plt.show()

plot_history(history)

```

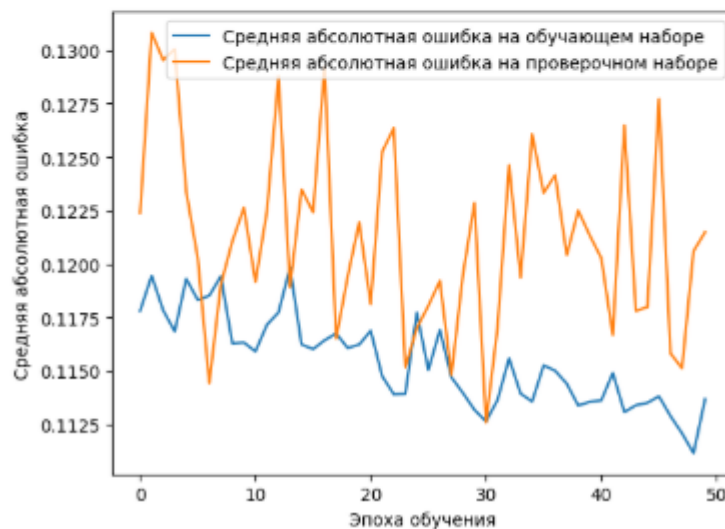


Рисунок 84. График процесса обучения модели

Далее выполним проверку качества модели на полном датасете

```

# Проверка качества модели на полном датасете
check_mae_double_input(model, x_data, x_seq, y_true, y_scaler, plot=True)

```

2192/2192 — 11s 5ms/step
 Средняя абсолютная ошибка 68734.867 рублей это 13.825% от общей выборки в 78119 примеров.

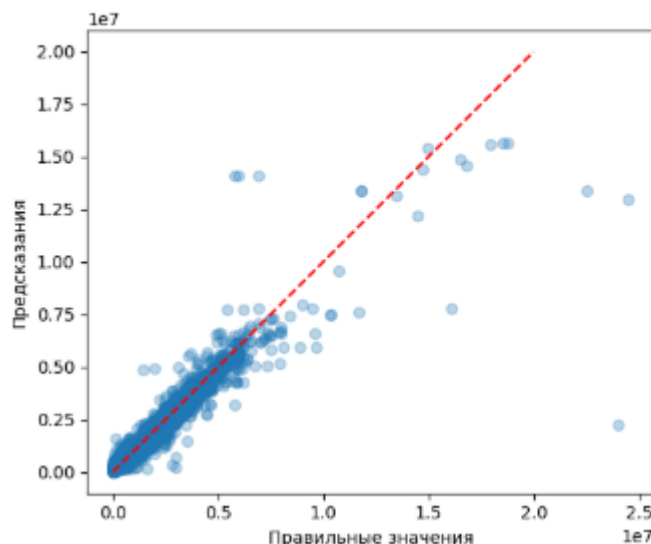


Рисунок 85. Проверка качества модели

Отсюда в ходе выполнения задания была обучена нейронная сеть для предсказания цен на автомобили по данным с платформы Юла. Данные были разделены на обучающую, валидационную и тестовую выборки. Модель имеет два скрытых слоя и один выходной нейрон, как указано в условиях. В

качестве функции потерь использовалась среднеквадратическая ошибка (MSE). После обучения модели предсказания на тестовой выборке были преобразованы обратно к исходному масштабу цен. Средняя абсолютная ошибка (MAE) составила ~68 734 рублей, при средней цене автомобилей ~558 350 рублей, что дало итоговый процент ошибки около 13.025%. Таким образом, модель соответствует заданным требованиям и демонстрирует приемлемую точность для задачи регрессии цен.

Вывод: в ходе выполнения работы были изучены и применены методы регрессии с использованием нейронных сетей для решения практических задач, таких как предсказание зарплаты, оценка результатов баскетбольных матчей и прогнозирование цен на автомобили.