Лабораторная работа "Нейронные сети"

В рамках этой работы должны быть выполнены следующие этапы.

1. Создание нейронной сети для бинарной классификации.
2. Создание нейронной сети для многоклассовой классификации.
3. Создание нейронной сети для регрессии.

Описание базовой структуры нейронной сети

Полносвязная нейросеть состоит из k слоев, причем i-тый слой имеет u_i нейронов и функцию активации f_i. Нейросеть представляет собой функцию, отображающую вектор входных данных X в вектор Z_k. Значение Z_k вычисляется итеративно по формуле:

$$Z_i = \begin{cases} F_i(W_i Z_{i-1} + B_i), & i > 0, \\ X, & i = 0, \end{cases}$$

где W_i — матрица весов размера $u_i \times u_{i-1}$, B_i — вектор сдвигов размером u_i, F_i — функция, применяющая f_i покомпонентно.

Веса W_i и B_i — это обучаемые параметры, которые находятся градиентным спуском с помощью обратного распространения ошибки. Обычно начальные веса инициализируются случайно.

Количество слоёв сети k, количества нейронов u_i и вид функции активации — это гиперпараметры. От выбора их значений зависит скорость и качество обучения.

В качестве функций активации часто используют сигмоидальную функцию

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

гиперболический тангенс

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

или функцию RELU, задаваемую формулой

$$RELU(x) = \begin{cases} x, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

В задаче бинарной классификации разумно взять количество нейронов на выходном слое $u_k = 1$. Тогда выход сети Z_k без функции активации — то есть $f_k(x) = x$ —

можно интерпретировать по знаку. Если в качестве f_k взять сигмоидальную функцию, то выход $Z_k \in [0; 1]$ можно интерпретировать по знаку выражения $Z_k - 0.5$. Также значение Z_k можно интерпретировать как вероятность того, что наблюдение пренадлежит к одному из классов.

Для многоклассовой классификации обычно используют "one-hot-coding": на выходном слое число нейронов соответствует числу классов, при этом каждый нейрон принимает значение в промежутке $[0; 1]$. Это значение обозначает степень уверенности модели в том, что перед ней соответствующий класс. Значение нейронов на выходном слое с произвольной функцией активации можно преобразовать в промежуток $[0; 1]$, чтобы в сумме все значения давали 1, с помощью функции "softmax":

$$Softmax(x_1, x_2, \ldots, x_n) = \left(\frac{e^{x_1}}{\sum_i^n e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_i^n e^{x_i}} \right).$$

Это позволит воспринимать выход нейросети как некое распределение вероятностей.

Для задачи регрессии имеет смысл взять u_k по количеству ожидаемых выходов, при этом функция активации может отсутствовать (то есть быть тождественной). Тогда значение на выходе не будет ограничено множеством значений функции активации. Впрочем, функцию активации можно и добавить.

Примеры

В данном разделе приведены примеры реализации нейронных сетей для классификации и регрессии на языке Python.

Для реализации нейронных сетей использовался фреймворк Tensorflow, в частности его модуль Keras, который предоставляет интерфейс для создания нейронных сетей, программного составления графа вычислений для обратного распространения ошибки и прочего.

Для анализа данных использовались библиотеки pandas и numpy, для визуализации использовались библиотеки matplotlib и seaborn.

В качестве среды выполнения использовался сервис Google Colab, который предоставляет бесплатную возможность запуска файлов Jupiter Notebook на виртуальных машинах компании Google.

Для выполнения работы не обязательно использовать именно эту среду выполнения.

Бинарная классификация

В качестве примера для бинарной классификации возьмем набор данных диагностирования рака груди.

Этот датасет имеет следующую структуру.

- Поле 1: идентификатор.
- Поле 2: метка, является ли опухоль злокачественной или доброкачественной:
  - `M` — злокачественная опухоль;
  - `B` — доброкачественная опухоль.
- Поля 3-32: признаки для трех клеточных ядер, 3-12 для первого, 13-22 для второго, 23-32 для третьего:
  - Поле 3: радиус (среднее значение расстояний от центра до точек по периметру)
  - Поле 4: текстура (стандартное отклонение значений шкалы серого);
  - Поле 5: периметр;
  - Поле 6: площадь;
  - Поле 7: гладкость (локальное изменение длин радиусов);
  - Поле 8: компактность (периметр ^2 / площадь - 1,0);
  - Поле 9: вогнутость (выраженность вогнутых участков контура);
  - Поле 10: вогнутые точки (количество вогнутых участков контура);
  - Поле 11: симметрия;
  - Поле 12: фрактальная размерность ("приближение береговой линии" - 1).

В наборе данных нет пропущенных значений.

Данные имеют следующее распределение по классам:

- 357 доброкачественных опухолей,
- 212 злокачественных опухолей.

Сначала нужно подключить все необходимые библиотеки.

In [ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, losses
import matplotlib.pyplot as plt
```

Теперь скачаем данные и подготовим их.

In [ ]:
```
# ссылка на скачивание датасета
link = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-can

# сформируем имена колонок
# имена признаков, которые повторяются три раза
features = [
    "radius",
    "texture",
```

```
    "perimeter",
    "area",
    "smoothness",
    "compactness",
    "concavity",
    "concave_points",
    "symmetry",
    "fractal_dimension"
]

# повторяем имена признаков три раза с разными индексами
# и добавляем имена первых двух колонок с помощью
# контатенации списков оператором `+`

# ["id", "class"] + ["radius-1", "area-1", ...] + ["radius-2", "area-2", ...
names = sum(([f"{f}-{i}" for f in features] for i in range(1, 4)),
            ['id', 'class'])

raw_data = pd.read_csv(link, names=names)

# посмотрим на результат
raw_data
```

In [ ]:
```
# Слегка преобработаем данные

# 1. Уберем колонку id
data = raw_data.drop(["id"], axis=1)

# 2. Закодируем метку числом 0 и 1
data["class"], class_labels = pd.factorize(data["class"])

print("Class labels:", class_labels)

# посмотрим на результат
data
```

In [ ]:
```
# Разделим на обучающую и тестовую выборку

# используем уже реализованную функцию из sklearn
# see: https://scikit-learn.org/stable/modules/generated/sklearn.model_selec

from sklearn.model_selection import train_test_split

train_x, test_x = train_test_split(data, test_size=0.2, random_state=2023)
train_y, test_y = train_x.pop("class"), test_x.pop("class")

# посмотрим на результат
train_x
```

Теперь создадим нейросеть. Сначала опишем ее структуру.

1. На вход должны поступать признаки. Их 30 штук.
2. Нейросеть состоит из нескольких скрытых слоев. Стоит попробовать разное количество скрытых слоев с разным количеством нейронов и разными функциями

активациями.

3. Последний слой должен содержать один нейрон. В качестве функции активации выберем сигмоид.

Создадим базовую модель со следующими параметрами:

- три скрытых слоя с числом нейронов `[16, 8, 6]`;
- функции активации — `RELU`.

In [ ]:
```
# создаем модель keras
base_model = tf.keras.Sequential(
    layers=[                        # описываем слои нейронной сети
        layers.Dense(                    # создаем полносвязный слой
            16,                          # количество нейронов
            activation='relu',           # функция активации
            input_dim=train_x.shape[1]),  # количество нейронов на входе,
                                         # нужно тольно для первого слоя
        layers.Dense(8, activation='relu'), # так же задаем второй слой
        layers.Dense(6, activation='relu'),
        layers.Dense(1, activation='sigmoid') # выходной слой
    ],
    name="breast-cancer-base-model",
)

# посмотрим описание модели
base_model.summary()
```

```
Model: "breast-cancer-base-model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 16)                496

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 6)                 54

 dense_6 (Dense)             (None, 1)                 7

=================================================================
Total params: 693
Trainable params: 693
Non-trainable params: 0
_________________________________________________________________
```

Теперь надо выбрать функцию потерь и алгоритм оптимизации.

В качестве функции потерь можно взять перекрестную энтропию, среднюю квадратичную ошибку или среднюю абсолютную ошибку. Тут выберем перекрестную энтропию.

В качестве оптимизатор выберем алгоритм Adam.

Также можно включить метрики, которые мы хотим отслеживать в процессе обучения. Для примера будем отслеживать точность.

Метод `compile` сбросит текущие веса, построит граф вычислений, после чего модель будет готова к обучению.

In [ ]:
```
base_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'])
```

Теперь запустим обучение модели. Для обучения нужно выбрать количество эпох, размер пакета для градиентного спуска.

Также добавим тестовую выборку для отслеживания качества работы модели.

Также измерим время обучения модели. Для этого вставим директиву `%%time` в начало ячейки.

In [ ]:
```
%%time

base_model_history = base_model.fit(
    train_x, train_y,
    epochs=40,
    batch_size=20,
    validation_data=(test_x, test_y))
```

Теперь можем посмотреть информацию о процессе обучения в объекте, который вернула функция `fit`.

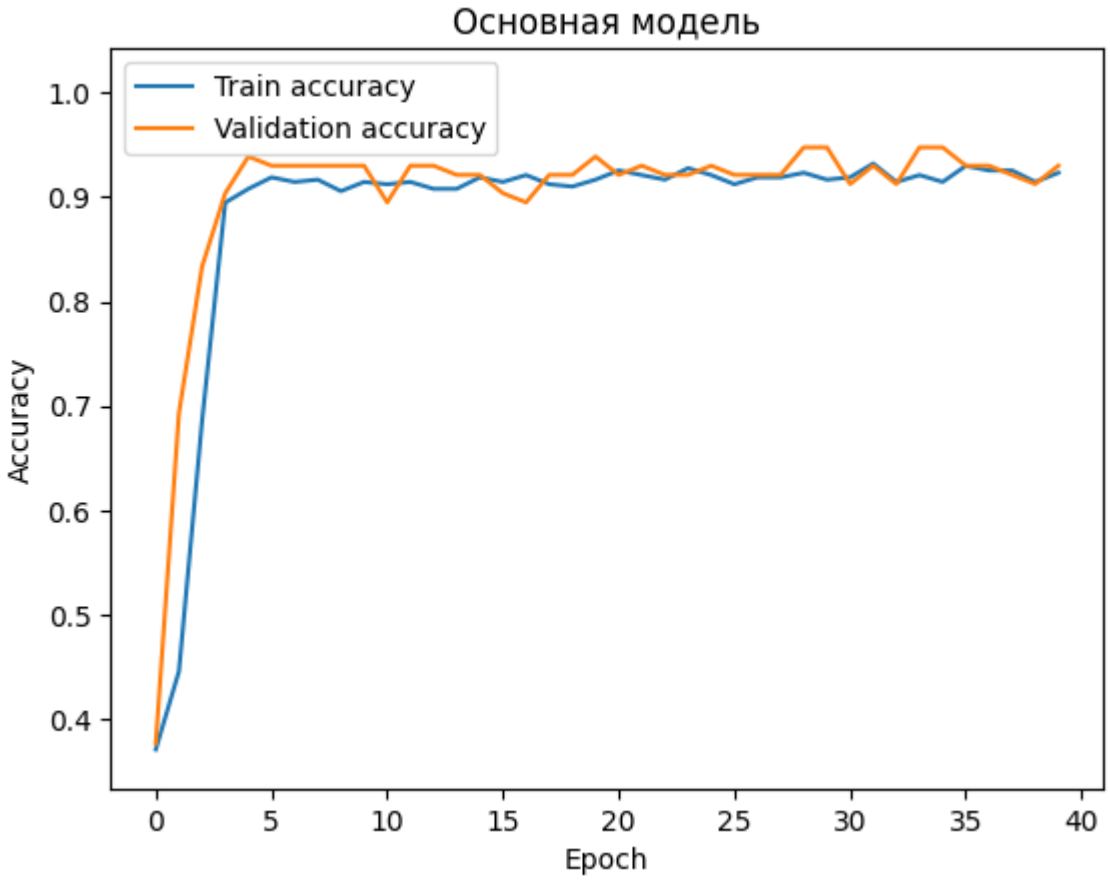
В его поле `history` лежат значения функции потерь и собранные метрики за каждую эпоху.

Напишем функцию для визуализации этих значений.

In [ ]:
```
from typing import Dict, List, Tuple

def plot_history(
    history: Dict[str, List[float]],
    title: str = "",
    metric_name: str = 'loss',
    ylim: Tuple[float, float] = None):

    train_values = history[metric_name]
    plt.plot(train_values, label=f'Train {metric_name}')
    try:
        val_values = history['val_' + metric_name]
        plt.plot(val_values, label=f'Validation {metric_name}')
    except KeyError:
        val_values = []
    plt.title(title)
```

```
    all_values = train_values + val_values
    ylim = ylim or (0.9 * min(all_values), 1.1 * max(all_values))
    plt.ylim(ylim)

    plt.ylabel(metric_name.capitalize())
    plt.xlabel("Epoch")
    plt.legend(loc='best')
    plt.show()
```

In [ ]:
```
plot_history(base_model_history.history, "Основная модель", "accuracy")
```



Многоклассовая классификация

Для примера многоклассовой классификации используем датасет MNIST. Его мы загрузим из модуля `tensorflow.keras.datasets`.

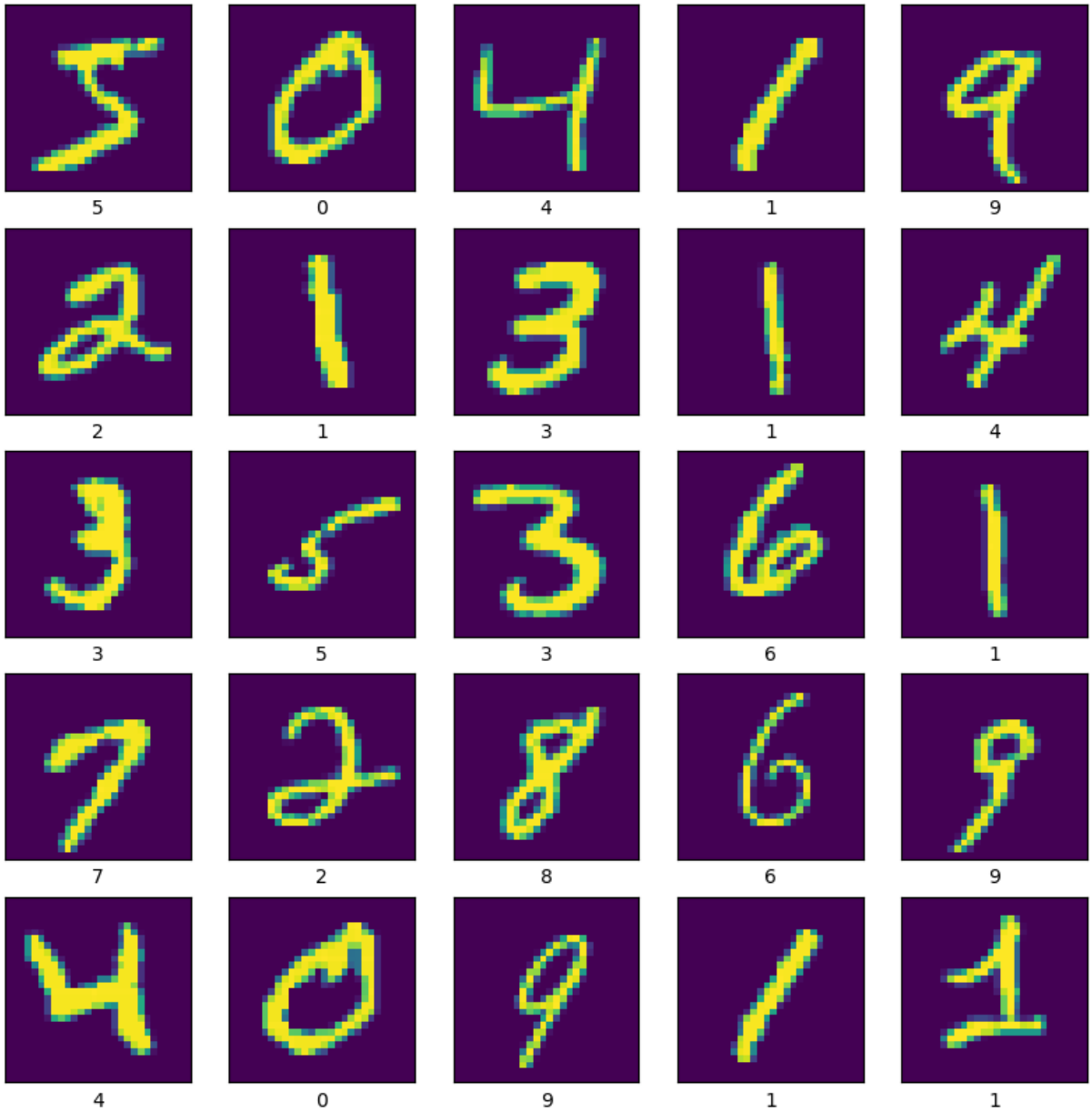
Этот датасет содержит изображения рукописных цифр от 0 до 9. Изображения имеют размер 28 на 28 пикселов в одном канале, каждый пиксель принимает значения от 0 до 255.

In [ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
```

```
from tensorflow.keras import layers, losses, datasets
import matplotlib.pyplot as plt
```

In [ ]:
```
# загружаем данные
(train_x, train_y), (test_x, test_y) = datasets.mnist.load_data()
```

In [ ]:
```
# немного посмотрим на данные

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_x[i])
    plt.xlabel(str(train_y[i]))
plt.show()
```



Теперь создадим нейронную сеть. Она будет иметь следующую структуру.

1. Входной слой мы вытянем в один вектор размером 784. Также его можно привести к диапазону $[0; 1]$ делением каждого значения на 255.
2. Несколько полносвязных скрытых слоев. Стоит попробовать разные размеры скрытых слоев и функции активации.
3. Выходной слой будет иметь 10 нейронов — по количеству классов. Чтобы получить вектор, который можно интерпретировать как распределение вероятности, оставим этот слой без функции активации, но добавим в конце функцию `softmax`.

Сделаем базовую модель с двумя скрытыми слоями размерами 128 и 32 и функцией активации `RELU`.

In [ ]:
```
base_model = tf.keras.Sequential([
        layers.Flatten(input_shape=(28, 28, 1)),
        layers.Rescaling(1 / 255),
        layers.Dense(128, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(10),
        layers.Softmax()
    ],
    name="mnist-base-model",
)

# посмотрим описание модели
base_model.summary()
```

```
Model: "mnist-base-model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 rescaling (Rescaling)       (None, 784)               0

 dense_7 (Dense)             (None, 128)               100480

 dense_8 (Dense)             (None, 32)                4128

 dense_9 (Dense)             (None, 10)                330

 softmax (Softmax)           (None, 10)                0

=================================================================
Total params: 104,938
Trainable params: 104,938
Non-trainable params: 0
_________________________________________________________________
```

В качестве функции активации будем использовать перекрестную энтропию для нескольких классов.

В нашем случае мы имеем метки класса числом от 0 до 9. Обычно для использования функции потерь нужно сделать "one-hot-coding", то есть превратить каждую метку класса в вектор длиной 10, в котором будут нули везде, кроме нужного класса, в котором будет 1.

Однако в `tensorflow` есть реализация перекрестной энтропии, которая работает с кодировкой класса одним числом, как в наших `train_y` и `test_y`. Воспользуемся ей.

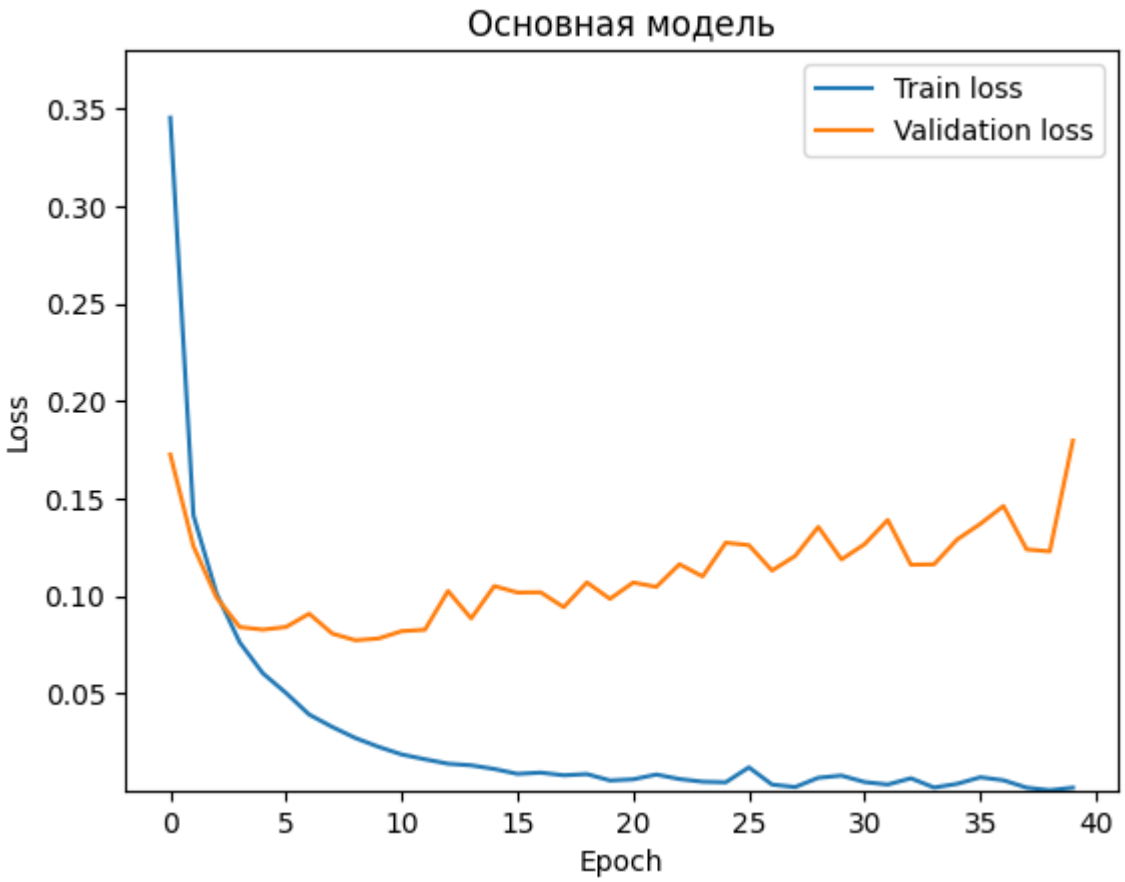
In [ ]:
```
base_model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Теперь начнем обучение модели.

In [ ]:
```
%%time

base_model_history = base_model.fit(
    train_x, train_y,
    epochs=40,
    batch_size=100,
    validation_data=(test_x, test_y))
```

Воспользуемся нашей функцией, чтобы отобразить динамику обучения.

In [ ]:
```
plot_history(base_model_history.history, "Основная модель")
```



Видно, что модель переобучилась: значение функции потерь на тестовой выборке перестало убывать.

Однако, точность модели получилась хорошая.

In [ ]:
```
base_model_history.history['val_accuracy'][-1]
```

Out[ ]:
```
0.972599983215332
```

Задача регрессии

Для задачи регрессии возьмем набор данных расхода топлива Auto MPG.

Набор данных имеет следующие колонки.

1. MPG (расход топлива, миль на галлон): непрерывное значение.
2. Количество цилиндров: дискретное значение.
3. Объем двигателя: непрерывное значение.
4. Количество лошадиных сил: непрерывное значение.
5. Вес: непрерывное значение.
6. Ускорение: непрерывное значение.
7. Год выпуска: дискретное значение.
8. Страна производства: 1 — США, 2 — Европа, 3 — Япония.

9. Название автомобиля: строка (уникальная для каждого экземпляра).

In [ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
# Загружаем данные из сети интернет

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/aut
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',
                'Acceleration', 'Model Year', 'Origin']

raw_dataset = pd.read_csv(url, names=column_names,
                          na_values='?', comment='\t',
                          sep=' ', skipinitialspace=True)
```

Обратим внимание, что в колонке с количеством лошадиных сил есть пропущенные значения. Заменим их средними значениями.

In [ ]:
```
print(raw_dataset.isna().sum())
filled_dataset = raw_dataset.fillna(raw_dataset["Horsepower"].mean())
filled_dataset.isna().sum()
```

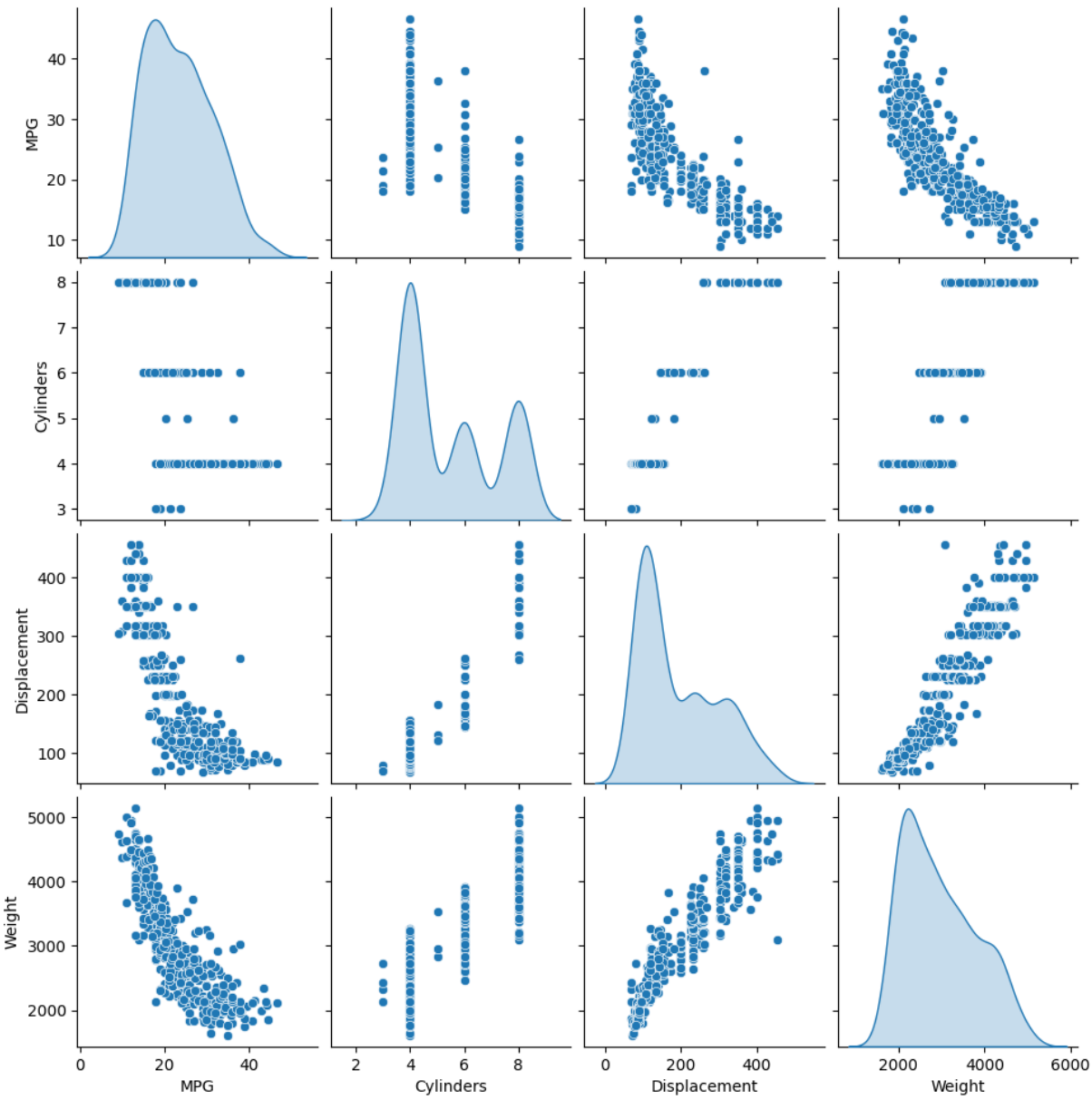
У нас есть категориальные данные в колонке "Origin". Для них нужно сделать "one hot coding". Сделать это можно функцией `get_dummies` из `pandas`.

In [ ]:
```
# Сделаем "one hot coding" для категориальной колонки "Origin"
dataset = filled_dataset.copy()
dataset["Origin"] = dataset["Origin"].map({ 1: "USA", 2: "Europe",
                                            3: "Japan" })
dataset = pd.get_dummies(dataset, columns=["Origin"])
dataset
```

In [ ]:
```
# Визуализируем данные при помощи библиотеки seaborn
sns.pairplot(dataset[['MPG', 'Cylinders', 'Displacement', 'Weight']],
             diag_kind='kde')
dataset.describe().transpose()
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	ma
MPG	398.0	23.514573	7.815984	9.0	17.500	23.0	29.000	46
Cylinders	398.0	5.454774	1.701004	3.0	4.000	4.0	8.000	8
Displacement	398.0	193.425879	104.269838	68.0	104.250	148.5	262.000	455
Horsepower	398.0	104.469388	38.199187	46.0	76.000	95.0	125.000	230
Weight	398.0	2970.424623	846.841774	1613.0	2223.750	2803.5	3608.000	5140
Acceleration	398.0	15.568090	2.757689	8.0	13.825	15.5	17.175	24
Model Year	398.0	76.010050	3.697627	70.0	73.000	76.0	79.000	82
Origin_Europe	398.0	0.175879	0.381197	0.0	0.000	0.0	0.000	1
Origin_Japan	398.0	0.198492	0.399367	0.0	0.000	0.0	0.000	1
Origin_USA	398.0	0.625628	0.484569	0.0	0.000	1.0	1.000	1



Наша цель — предсказать расход топлива по остальным признакам.

Разделим данные на обучающую и тестовую выборку, а также отделим наблюдаемое значение от признаков.

In [ ]:
```
# Разделим на две выборки, 80% – обучающая

from sklearn.model_selection import train_test_split

# установим random_state, чтобы получать одинаковые
# результаты при отключении ноутбука
train_x, test_x = train_test_split(dataset, test_size=0.2,
                                   random_state=2023)
train_y, test_y = train_x.pop("MPG"), test_x.pop("MPG")
```

Теперь построим модель. Она будет состоять из следующих частей.

1. На входе у нас будет 10 нейронов — 8 признаков, один из который мы отобразили в 3 признака с помощью "one hot coding".
2. Нормализация. В этот раз добавим нормализацию, которое сделает с каждым признаком следующее линейное преобразование:

$$\frac{x_i - \bar{x}}{\sqrt{s^2}},$$

где $\bar{x}$ — среднее значение признака x_i в обучающей выборке, а s^2 — выборочная дисперсия. Таким обазом все признаки в обучающей выборке будут иметь нулевое среднее и единичную выборочную дисперсию. 2. Несколько скрытых слоев с функциями активации. 3. Последний слой будет состоять из одного нейрона. Тут можно не ставить функцию активацию, а можно поставить функцию `RELU` или `ELU`, так как наше наблюдаемое значение всегда положительное.

В качестве базовой модели возьмем следующую модель.

1. Слой нормализации.
2. Два скрытых слоя размером 32 с функцией активации `RELU`.
3. Выходной слой с одним нейроном без функцией активации.

В качестве функции потерь используем среднюю квадратичную ошибку.

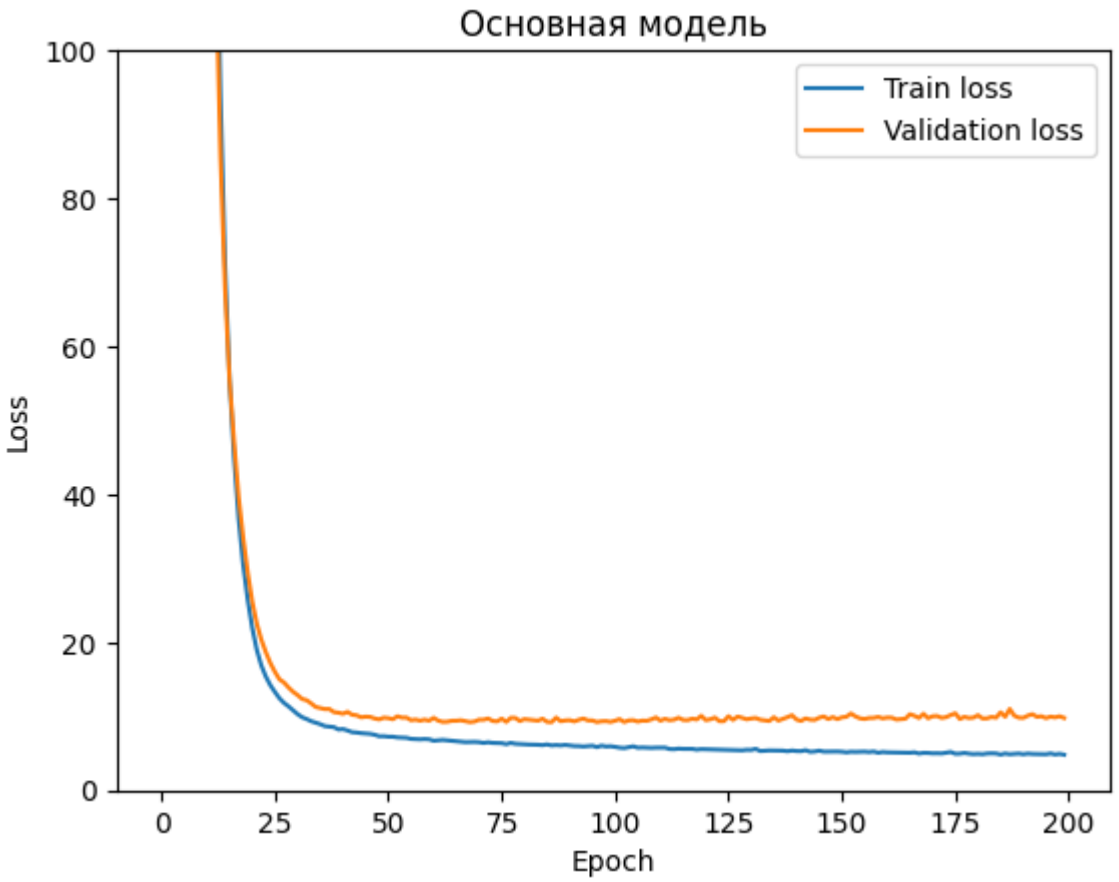
In [ ]:
```
normalizer = layers.Normalization()
normalizer.adapt(train_x)

base_model = tf.keras.Sequential([
    normalizer,
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1),
])

# по умолчанию оптимизируемся стохастическим ГС
base_model.compile(loss="mean_squared_error")

base_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normalizat  (None, 9)                19
 ion)

 dense (Dense)               (None, 32)                320

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,428
Trainable params: 1,409
Non-trainable params: 19
_________________________________________________________________
```

In [ ]:
```
%%time

base_model_history = base_model.fit(
    train_x, train_y,
    validation_data=(test_x, test_y),
    epochs=200)
```

Построим график функции потерь с помощью нашей функции.

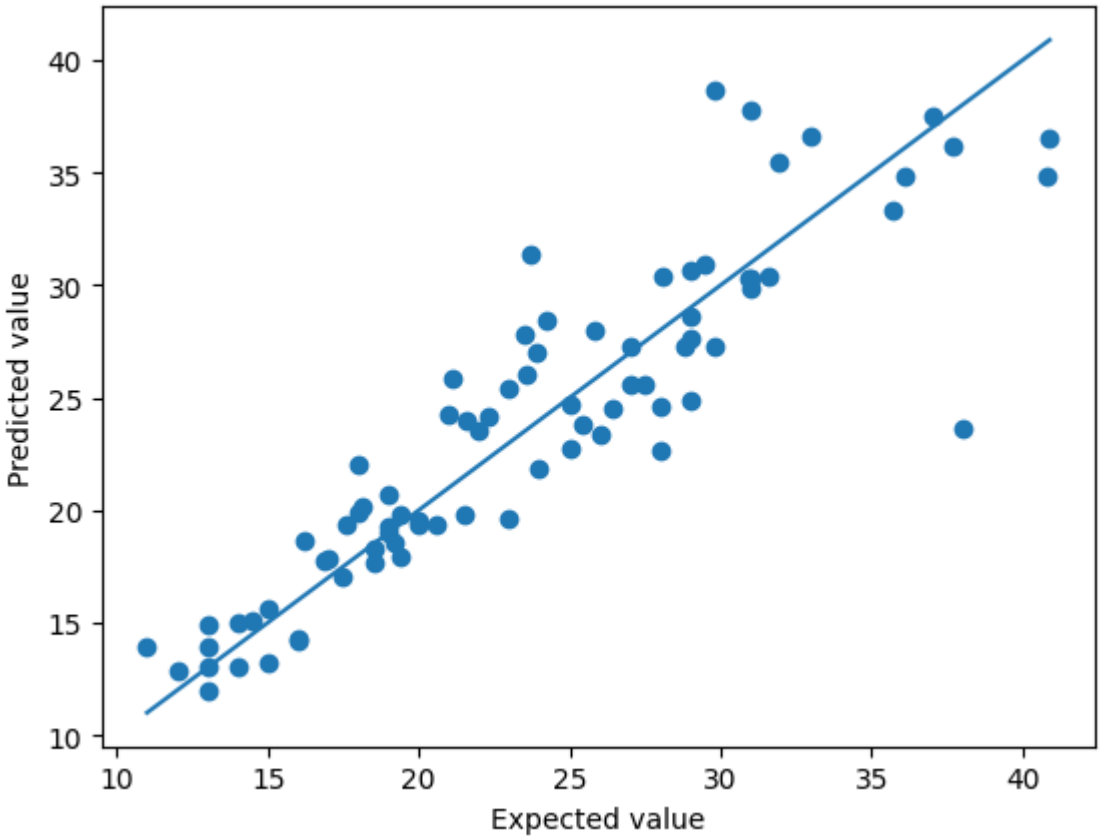
In [ ]:
```
plot_history(base_model_history.history, "Основная модель", ylim=[0, 100])
```



Визуализируем полученный результат на графике. По оси OX отложим ожидаемое значение, а по оси OY — результат регрессии. Чем ближе точки к линии $y = x$, тем лучше предсказание.

In [ ]:
```
predicted_y = base_model.predict(test_x, verbose=0)
plt.scatter(test_y, predicted_y)
plt.xlabel("Expected value")
plt.ylabel("Predicted value")
lims = [min(*test_y, *predicted_y), max(*test_y, *predicted_y)]
plt.plot(lims, lims)
print("Mean squared error:", base_model_history.history["val_loss"][-1])
```

```
Mean squared error: 9.77734375
```



Задания

1. Попробуйте другие архитектуры нейронных сетей для бинарной классификации на наборе данных из примера. Попробуйте изменить количество слоев, количество нейронов и функции активации. Сравните результаты. Оцените время обучения моделей.
2. Постройте нейронную сеть для бинарной классификации для набора данных о сердечной недостаточности. Сравните эффективность нейросетей с различными параметрами.
3. Попробуйте другие архитектуры нейронных сетей для многоклассовой классификации на наборе данных из примера. Попробуйте изменить количество слоев, количество нейронов и функции активации. Сравните результаты. Оцените время обучения моделей.
4. Постройте нейронную сеть для многоклассовой классификации для набора данных Fashion MNIST. Сравните эффективность нейросетей с различными параметрами.
5. Попробуйте другие архитектуры нейронных сетей для регрессии на наборе данных из примера. Попробуйте изменить количество слоев, количество нейронов и функции активации. Оцените влияние нормализации. Сравните результаты. Оцените время обучения моделей.
6. Постройте нейронную сеть для регрессии для набора данных цен на недвижимость в Бостоне. Сравните эффективность нейросетей с различными

параметрами.