

Глубокое обучение 2025

Занятие 2. Глубокое обучение с Keras

Создание виртуального окружения для Jupyter Notebook

Последовательность действий:

- `conda create -n <имя> python=3.11.9` или через Anaconda Navigator
- `conda activate <имя>` или через Anaconda Navigator
- `conda install <пакеты>` или через Anaconda Navigator или `pip install <пакеты>`
- `conda install ipykernel` или через Anaconda Navigator
- `python -m ipykernel install --user --name=<имя>`

Убираем предупреждения и импортируем библиотеки:

```
In [1]: # убираем предупреждающие сообщения
from silence_tensorflow import silence_tensorflow
silence_tensorflow()
```

```
In [2]: # импорт библиотек
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras
from keras import datasets
```

Пример

Будем использовать набор данных `Fashion-MNIST`, который состоит из изображений размером 28x28 пикселей для 10 классов модных товаров.

```
In [3]: (X_train, y_train), (X_test, y_test) = datasets.fashion_mnist.load_data()
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
Out[3]: ((60000, 28, 28), (10000, 28, 28), (60000,), (10000,))
```

```
In [4]: fmnist_classes = {0: "T-shirt/top", 1: "Trouser", 2: "Pullover",
                          3: "Dress", 4: "Coat", 5: "Sandal",
                          6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle boot"}
```

```
# Выберем случайные изображения
from random import randint
fig, axes = plt.subplots(1, 5, figsize=(10, 5))
for i in range(5):
    n = randint(0, 60000) # 70000
    axes[i].imshow(X_train[n], cmap=plt.cm.gray_r) # .reshape(28, 28)
    axes[i].set_xticks([])
    axes[i].set_yticks([])
    axes[i].set_xlabel("{} ".format(fmnist_classes[y_train[n]]))
plt.show();
```



Препроцессинг

Изменение формы (reshaping)

Чтобы иметь возможность передавать эти данные через нейронную сеть, форма входных данных должна соответствовать форме входного слоя.

Для обычных плотных слоев это будет плоский массив. Можно использовать:

- `numpy.reshape()`
- слои Keras, например, `Flatten`

Изменение масштаба (rescaling)

Изменение масштаба данных помогает при обучении нейронной сети и приведет к более быстрой сходимости. Вы можете использовать минимальное и максимальное масштабирование до $[0,1]$ или стандартизацию (среднее значение 0, стандартное отклонение 1). Используем здесь простое деление на максимально возможное значение.

```
In [5]: X_train = X_train.astype('float32') / 255
X_test  = X_test.astype('float32') / 255
```

Форматирование меток

Для многоклассовой классификации наш выходной слой обычно будет иметь один выходной нейрон для каждого класса. Поэтому нам необходимо выполнить прямое кодирование меток (one-hot-encoding). Например, классу '4' соответствует вектор `[0,0,0,0,1,0,...]` с единицей на пятой позиции (метки кодируются с нуля).

Для этого в Keras есть вспомогательная функция [to_categorical](#).

```
In [6]: y_train
```

```
Out[6]: array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

```
In [7]: from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test  = to_categorical(y_test)
y_train
```

```
Out[7]: array([[0., 0., 0., ..., 0., 0., 1.],
               [1., 0., 0., ..., 0., 0., 0.],
               [1., 0., 0., ..., 0., 0., 0.],
               ...,
               [0., 0., 0., ..., 0., 0., 0.],
               [1., 0., 0., ..., 0., 0., 0.],
               [0., 0., 0., ..., 0., 0., 0.]])
```

```
In [8]: X_train.shape, y_train.shape
```

```
Out[8]: ((60000, 28, 28), (60000, 10))
```

```
In [9]: X_train = X_train.reshape((-1, 28*28))
X_test  = X_test.reshape((-1, 28*28))
```

```
In [10]: X_train.shape
```

```
Out[10]: (60000, 784)
```

Разделение на обучающую, тестовую и проверочную выборки

Если наборы данных достаточно велики, то обычно используются простые алгоритмы разделения. Для небольших наборов данных также можно использовать перекрестную валидацию, но можно столкнуться с высокой дисперсией результатов.

Разделим обучающую выборку на набор для обучения и проверки (валидации).

```
In [11]: from sklearn.model_selection import train_test_split
Xf_train, X_val, yf_train, y_val = \
    train_test_split(X_train, y_train, train_size=50000,
                    shuffle=True, stratify=y_train, random_state=0)
```

```
In [12]: Xf_train.shape, yf_train.shape, X_val.shape, y_val.shape, X_test.sh
```

```
Out[12]: ((50000, 784),
          (50000, 10),
          (10000, 784),
          (10000, 10),
          (10000, 784),
          (10000, 10))
```

Построение последовательных моделей нейронных сетей

- **Последовательные** модели **Tensorflow** — это самый простой вид нейронных сетей. Они состоят из ряда слоев, идущих один за другим.
- В **Tensorflow** определено **много типов слоев**
- Пока будем использовать только плотные **Dense** (полносвязные) слои. В плотных слоях имеется несколько важных настроек:
 - **units**: количество узлов (нейронов)
 - **activation**: **функция активации**
 - **kernel_initializer**: как **инициализировать веса**
 - **kernel_regularizer**: применять ли L1/L2 **регуляризацию**

```
keras.layers.Dense(
    units, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Рассмотрим следующую простую сеть с одним скрытым слоем:

- метод **Sequential.add()** добавляет слой в сеть
- также можно передать в конструктор массив слоев:
Sequential([layers])
- используется активация **ReLU** для скрытого слоя и **SoftMax** для выходного слоя

```
In [13]: from keras import models
         from keras import layers

         model = models.Sequential()
         model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28
         model.add(layers.Dense(10, activation='softmax'))
```

```
/opt/anaconda3/envs/dlrfm218/lib/python3.12/site-packages/keras/src/
layers/core/dense.py:92: UserWarning: Do not pass an `input_shape`/`
input_dim` argument to a layer. When using Sequential models, prefer
using an `Input(shape)` object as the first layer in the model inste
ad.
```

```
super().__init__(activity_regularizer=activity_regularizer, **kwar
gs)
```

```
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
```

```
I0000 00:00:1757928226.242420 3670825 pluggable_device_factory.cc:30
5] Could not identify NUMA node of platform GPU ID 0, defaulting to
0. Your kernel may not have been built with NUMA support.
```

```
I0000 00:00:1757928226.242472 3670825 pluggable_device_factory.cc:27
1] Created TensorFlow device (/job:localhost/replica:0/task:0/devic
e:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, na
me: METAL, pci bus id: <undefined>)
```

Входной слой

Обратите внимание, что входной слой может быть определен с помощью параметра `input_shape`. В качестве альтернативы также можно добавить явный входной слой `InputLayer` с параметром `shape`. В нашем случае данные представляют собой плоский массив из 28*28 входов.

```
In [14]: model = models.Sequential()
model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Слои активации

Большинство функций активации, инициализаторов и регуляризаторов можно указать в виде ключевых слов. Если нужен больший контроль над нейронной сетью, то можно указать активацию как отдельный слой. Тогда плотный слой будет использовать линейную активацию, а в следующем слое будет применяться выбранная активация.

```
In [15]: model = models.Sequential()
model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(512))
model.add(layers.ReLU(negative_slope=0.1)) # слой leaky ReLU
model.add(layers.Dense(10, activation='softmax'))
```

Краткое описание модели

- вызов `model.summary()` выводит краткое описание модели по слоям
 - скрытый слой 1: $(28 * 28 + 1) * 512 = 401920$
 - скрытый слой 2: $(512 + 1) * 512 = 262656$

- выходной слой: $(512 + 1) * 10 = 5130$

```
In [16]: ## добавим дополнительный скрытый слой для лучшей производительности
model = models.Sequential()
model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
In [17]: model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	
dense_6 (Dense)	(None, 512)	
dense_7 (Dense)	(None, 512)	
dense_8 (Dense)	(None, 10)	

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

Выбор функции потерь, оптимизатора, метрик

Вызов `model.compile()` указывает, как модель должна обучаться, т. е. какую функцию потерь и оптимизатор использовать и какие показатели оценки качества модели вычислять.

- **Функция потерь** [см. обзор](#)
 - Кросс-энтропия (логарифмические потери) для многоклассовой классификации (метка y_{true} должна иметь прямое кодирование (one-hot encoded))
 - Используйте бинарную кросс-энтропию для задач бинарной классификации (один выходной нейрон)
 - Используйте разреженную категориальную кросс-энтропию, если выход y_{true} закодирован метками 0,1,2,3,...
- **Оптимизатор** [см. обзор](#)
 - Любой из доступных оптимизаторов. `RMSprop` и `Adam` обычно работают хорошо.
- **Метрики** [см. обзор](#)
 - Для мониторинга производительности во время обучения и тестирования, например, доля верных ответов (accuracy)

Значения всех объектов в методе `compile()` можно указать как текст:

```
In [18]: # коротко
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'rmsprop',
              metrics = ['accuracy'])
```

Для большего контроля можно передать названия фактических функций (с параметрами):

```
In [19]: from keras.optimizers import RMSprop
from keras.losses import CategoricalCrossentropy
from keras.metrics import Accuracy

# детально
model.compile(loss = CategoricalCrossentropy(label_smoothing=0.01),
              optimizer = RMSprop(learning_rate=0.001, momentum=0.0),
              metrics = [Accuracy()])
```

Обучение (подгонка)

Функция `fit` обучает сеть и возвращает историю потерь при обучении и проверке, а также значения всех метрик за эпоху.

```
network.fit(X_train, y_train, epochs=3, batch_size=64)
```

Имеется два важных гиперпараметра:

- **Количество эпох** (epochs): должно быть достаточно, чтобы обеспечить сходимость
 - Слишком много: модель начинает переобучаться (или просто теряет время)
- **Размер пакета** (batch_size): часто предпочтительнее небольшие пакеты (например, 32, 64 и т. д.)
 - «Зашумленные» обучающие данные снижают вероятность переобучения
 - Большие пакеты хуже обобщаются
 - Требуют меньше памяти (особенно в графических процессорах)
 - Большие пакеты ускоряют обучение и сходимость достигается за меньшее количество эпох

Повторяющееся обучение

Вызов `model.fit` несколько раз не воссоздает модель с нуля (как это делается в `scikit-learn`), а просто продолжает обучение с сохраненными весами. Для обучения с нуля, например, с разными гиперпараметрами, необходимо воссоздать модель, например, оформив создание модели как функцию `create_model`.

```
In [20]: def create_model():
model = models.Sequential()
```

```

model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(512, activation='relu', kernel_initializer=
model.add(layers.Dense(512, activation='relu', kernel_initializer=
model.add(layers.Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

return model

```

Отслеживание прогресса обучения

Вызов `fit` обеспечивает вывод прогресса для каждой эпохи и возвращает объект `history`, содержащий все потери и показатели метрик оценки.

```

In [21]: model = create_model()
         history = model.fit(Xf_train, yf_train, epochs=3, batch_size=64);

```

```

Epoch 1/3
782/782 ————— 7s 7ms/step – accuracy: 0.7198 – loss:
1.1345
Epoch 2/3
782/782 ————— 6s 7ms/step – accuracy: 0.7267 – loss:
2.4327
Epoch 3/3
782/782 ————— 6s 7ms/step – accuracy: 0.7246 – loss:
5.4512

```

```

In [22]: model.to_json()

```



```

Out[22]: '{"module": "keras", "class_name": "Sequential", "config": {"name": "sequential_4", "trainable": true, "dtype": {"module": "keras", "class_name": "DTypePolicy", "config": {"name": "float32"}, "registered_name": null}, "layers": [{"module": "keras.layers", "class_name": "InputLayer", "config": {"batch_shape": [null, 784], "dtype": "float32", "sparse": false, "ragged": false, "name": "input_layer_4"}, "registered_name": null}, {"module": "keras.layers", "class_name": "Dense", "config": {"name": "dense_9", "trainable": true, "dtype": {"module": "keras", "class_name": "DTypePolicy", "config": {"name": "float32"}, "registered_name": null}, "units": 512, "activation": "relu", "use_bias": true, "kernel_initializer": {"module": "keras.initializers", "class_name": "HeNormal", "config": {"seed": null}, "registered_name": null}, "bias_initializer": {"module": "keras.initializers", "class_name": "Zeros", "config": {}, "registered_name": null}, "kernel_regularizer": null, "bias_regularizer": null, "kernel_constraint": null, "bias_constraint": null}, {"module": "keras.layers", "class_name": "Dense", "config": {"name": "dense_10", "trainable": true, "dtype": {"module": "keras", "class_name": "DTypePolicy", "config": {"name": "float32"}, "registered_name": null}, "units": 512, "activation": "relu", "use_bias": true, "kernel_initializer": {"module": "keras.initializers", "class_name": "HeNormal", "config": {"seed": null}, "registered_name": null}, "bias_initializer": {"module": "keras.initializers", "class_name": "Zeros", "config": {}, "registered_name": null}, "kernel_regularizer": null, "bias_regularizer": null, "kernel_constraint": null, "bias_constraint": null}, {"module": "keras.layers", "class_name": "Dense", "config": {"name": "dense_11", "trainable": true, "dtype": {"module": "keras", "class_name": "DTypePolicy", "config": {"name": "float32"}, "registered_name": null}, "units": 10, "activation": "softmax", "use_bias": true, "kernel_initializer": {"module": "keras.initializers", "class_name": "GlorotUniform", "config": {"seed": null}, "registered_name": null}, "bias_initializer": {"module": "keras.initializers", "class_name": "Zeros", "config": {}, "registered_name": null}, "kernel_regularizer": null, "bias_regularizer": null, "kernel_constraint": null, "bias_constraint": null}, {"module": "keras.layers", "class_name": "Dense", "config": {"name": "dense_12", "trainable": true, "dtype": {"module": "keras", "class_name": "DTypePolicy", "config": {"name": "float32"}, "registered_name": null}, "units": 10, "activation": "softmax", "use_bias": true, "kernel_initializer": {"module": "keras.initializers", "class_name": "GlorotUniform", "config": {"seed": null}, "registered_name": null}, "bias_initializer": {"module": "keras.initializers", "class_name": "Zeros", "config": {}, "registered_name": null}, "kernel_regularizer": null, "bias_regularizer": null, "kernel_constraint": null, "bias_constraint": null}], "build_config": {"input_shape": [null, 784]}, "build_input_shape": [null, 784]}, {"module": "keras.optimizers", "class_name": "RMSprop", "config": {"name": "rmsprop", "learning_rate": 0.001, "weight_decay": null, "clipnorm": null, "global_clipnorm": null, "clipvalue": null, "use_ema": false, "ema_momentum": 0.99, "ema_overwrite_frequency": null, "loss_scale_factor": null, "gradient_accumulation_steps": null, "rho": 0.9, "momentum": 0.0, "epsilon": 1e-07, "centered": false}, "registered_name": null}, {"module": "keras.metrics", "class_name": "CategoricalAccuracy", "config": {"name": "categorical_accuracy", "loss_weights": null, "metrics": ["accuracy"], "weighted_metrics": null, "run_eagerly": false, "steps_per_execution": 1, "jit_compile": false}}}'

```

Можно также указать проверочную (валидационную) выборку, чтобы также возвращались показатели потерь и доли верных ответов на проверочной выборке. Параметр `verbose=0` заглушает вывод данных.

```
In [23]: model = create_model()
history = model.fit(Xf_train, yf_train, epochs=3, batch_size=32, validation_data=(X_val, y_val))
```

Возвращенная история обучения (объект `history`) содержит данные оценки качества модели (потери и метрики) для каждой эпохи.

```
In [24]: history.history
```

```
Out[24]: {'accuracy': [0.7127000093460083, 0.7134199738502502, 0.7135599851608276],
          'loss': [1.591084361076355, 5.873380184173584, 14.718507766723633],
          'val_accuracy': [0.6679999828338623, 0.6330999732017517, 0.6855999827384949],
          'val_loss': [4.702043533325195, 15.511051177978516, 27.070091247558594]}
```

Прогнозы и оценки

Теперь можно вызывать `predict` для генерации прогнозов и оценить качество обученной модели на всем тестовом наборе при помощи `evaluate`.

```
network.predict(X_test)
test_loss, test_acc = network.evaluate(X_test, y_test)
```

```
In [25]: predictions = model.predict(X_test)

# Visualize one of the predictions
sample_id = 0
print('Прогнозируемые вероятности меток:\n', predictions[sample_id])

np.set_printoptions(precision=7)
fig, axes = plt.subplots(1, 1, figsize=(4, 4))
axes.imshow(X_test[sample_id].reshape(28, 28), cmap=plt.cm.gray_r)
axes.set_xlabel("Истинная метка: {}".format(y_test[sample_id]))
axes.set_xticks([])
axes.set_yticks([]);
```

313/313 ————— 0s 1ms/step

Прогнозируемые вероятности меток:

```
[0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 3.2934842e-11 0.0000000e+00 1.7857913e-13 0.0000000e+00 1.0000000e+00]
```



Истинная метка: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]

```
In [26]: test_loss, test_acc = model.evaluate(X_test, y_test)
print('Доля верных ответов (accuracy) на тестовой выборке:', test_a
```

313/313 ————— 2s 6ms/step – accuracy: 0.6867 – loss: 27.9076

Доля верных ответов (accuracy) на тестовой выборке: 0.6866999864578247

Проверка кривых обучения

Есть несколько способов проверить кривые обучения

- Подождите, пока обучение завершится, затем нарисуйте кривые обучения из возвращенной истории
- Добавьте обратный вызов (callback) в функцию `fit`, который перерисовывает кривые обучения в режиме реального времени при каждом обновлении (пример реализации приведен ниже)
- Используйте внешний инструмент, например [TensorBoard](#)

```
In [27]: from IPython.display import clear_output

# For plotting the learning curve in real time
class TrainingPlot(keras.callbacks.Callback):

    # This function is called when the training begins
    def on_train_begin(self, logs={}):
        # Initialize the lists for holding the logs, losses and acc
        self.losses = []
        self.acc = []
        self.val_losses = []
        self.val_acc = []
        self.logs = []
```

```

self.max_acc = 0

# This function is called at the end of each epoch
def on_epoch_end(self, epoch, logs={}):

    # Append the logs, losses and accuracies to the lists
    self.logs.append(logs)
    self.losses.append(logs.get('loss'))
    self.acc.append(logs.get('accuracy'))
    self.val_losses.append(logs.get('val_loss'))
    self.val_acc.append(logs.get('val_accuracy'))
    self.max_acc = max(self.max_acc, logs.get('val_accuracy'))

    # Before plotting ensure at least 2 epochs have passed
    if len(self.losses) > 1:

        # Clear the previous plot
        clear_output(wait=True)
        N = np.arange(0, len(self.losses))

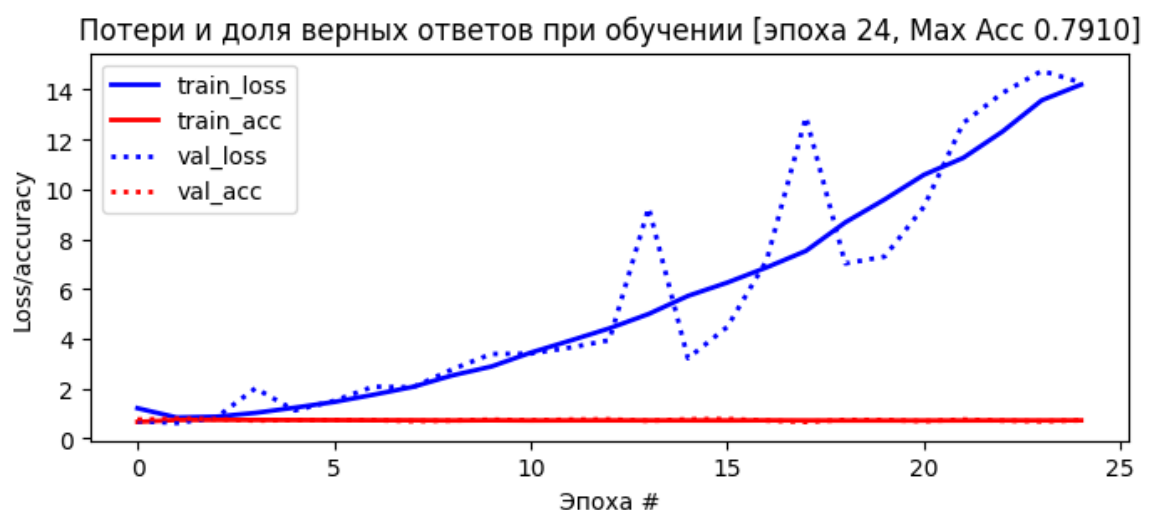
        # Plot train loss, train acc, val loss and val acc again
        plt.figure(figsize=(8,3))
        plt.plot(N, self.losses, lw=2, c="b", linestyle="--", label="train_loss")
        plt.plot(N, self.acc, lw=2, c="r", linestyle="--", label="train_acc")
        plt.plot(N, self.val_losses, lw=2, c="b", linestyle=":", label="val_loss")
        plt.plot(N, self.val_acc, lw=2, c="r", linestyle=":", label="val_acc")
        plt.title("Потери и доля верных ответов при обучении [Эпоха #]")
        plt.xlabel("Эпоха #")
        plt.ylabel("Loss/accuracy")
        plt.legend()
        plt.show()

```

```

In [28]: plot_losses = TrainingPlot()
model = create_model()
history = model.fit(Xf_train, yf_train, epochs=25, batch_size=512,
                    validation_data=(X_val, y_val), callbacks=[plot_losses])

```



Ранняя остановка (early stopping)

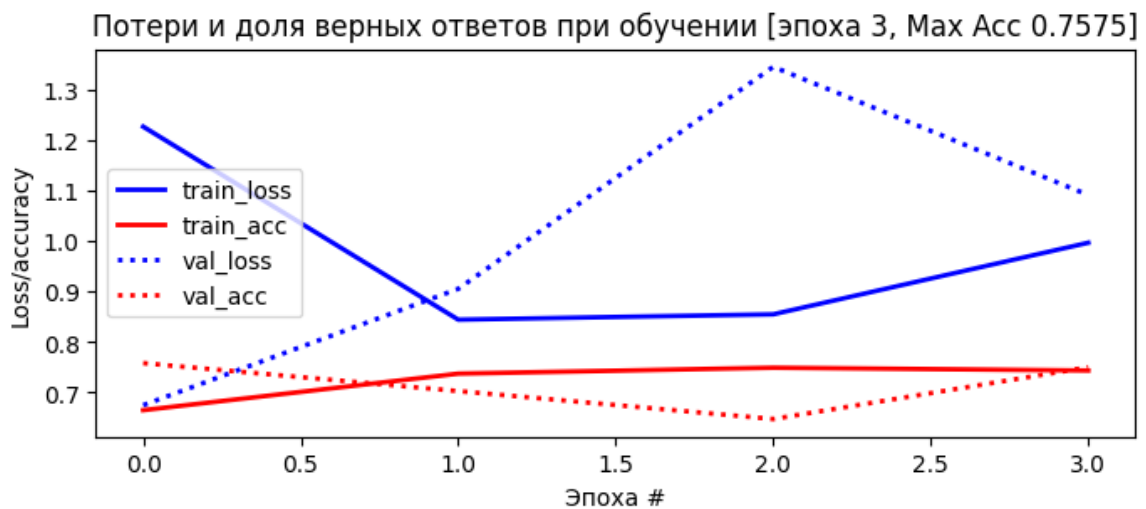
- Нужно прекратить обучение, когда потери на валидационной

выборке (или доля верных ответов на валидационной выборке) больше не улучшаются

- При этом нужно учитывать, что потери могут быть неровным: используйте скользящее среднее или подождите k шагов без улучшения

```
earlystop = callbacks.EarlyStopping(monitor='val_loss',  
patience=3)  
model.fit(x_train, y_train, epochs=25, batch_size=512,  
callbacks=[earlystop])
```

```
In [29]: from keras import callbacks  
  
earlystop = callbacks.EarlyStopping(monitor='val_loss', patience=3)  
model = create_model()  
history = model.fit(Xf_train, yf_train, epochs=25, batch_size=512, validation_data=(X_val, y_val), callbacks=[plot])
```



Регуляризация

Есть несколько способов регуляризации моделей в случае переобучения:

- Получить больше обучающих данных
- Уменьшить сеть (например, использовать меньше нейронов в слоях или использовать меньшее количество слоев)
- Регуляризовать веса модели (например, с помощью регуляризации L1/L2)
- Использовать технику исключения нейронов (dropout)
- Пакетная нормализация (batch normalization) также обладает эффектом регуляризации

Регуляризация весов (уменьшение весов)

- Регуляризацию весов можно применять к слоям с помощью

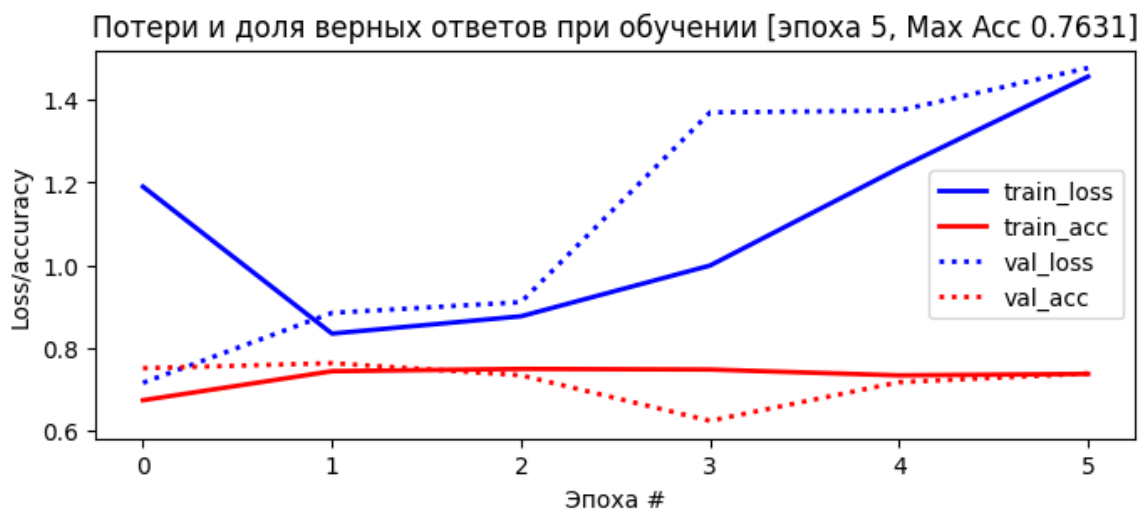
регуляризатора

- Регуляризация L1: приводит к *разреженным сетям* со многими весами, равными нулю
- Регуляризация L2: приводит к большому количеству очень маленьких весов

```
In [30]: from keras import regularizers

model = models.Sequential()
model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(512, activation='relu', kernel_initializer='l1',
                        kernel_regularizer='l1'))
model.add(layers.Dense(512, activation='relu', kernel_initializer='l1',
                        kernel_regularizer='l1'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

earlystop = callbacks.EarlyStopping(monitor='val_loss', patience=5)
model = create_model()
history = model.fit(Xf_train, yf_train, epochs=50, batch_size=512,
                    validation_data=(X_val, y_val), callbacks=[plot])
```



Исключение (отсев) нейронов (dropout)

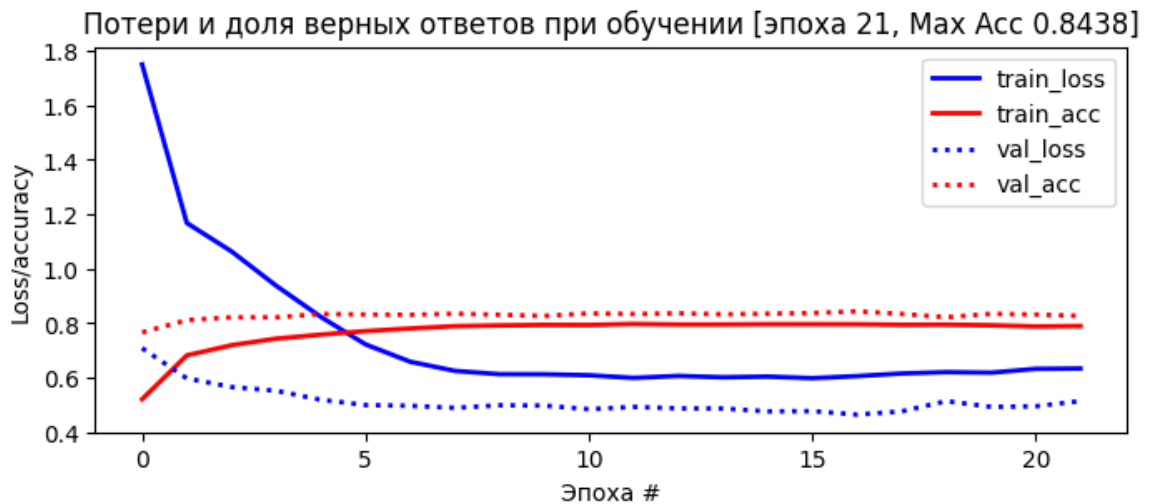
- Механизм dropout случайным образом устанавливает некоторое количество функций активации в слое равными нулю. Это позволяет избежать запоминания моделью несущественных связей в данных
- Механизм dropout добавляется в модель через слой [Dropout](#).
- Коэффициент отсева (dropout rate) или доля обнулённых выходных значений обычно составляет от 0.1 до 0.5, но этот параметр должен быть настроен на конкретную задачу
- Слой dropout может быть добавлен после любого плотного слоя

```
In [31]: model = models.Sequential()
```

```

model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
plot_losses = TrainingPlot())
earlystop = callbacks.EarlyStopping(monitor='val_loss', patience=5)
history = model.fit(Xf_train, yf_train, epochs=50, batch_size=512,
validation_data=(X_val, y_val), callbacks=[plot_losses, earlystop])

```



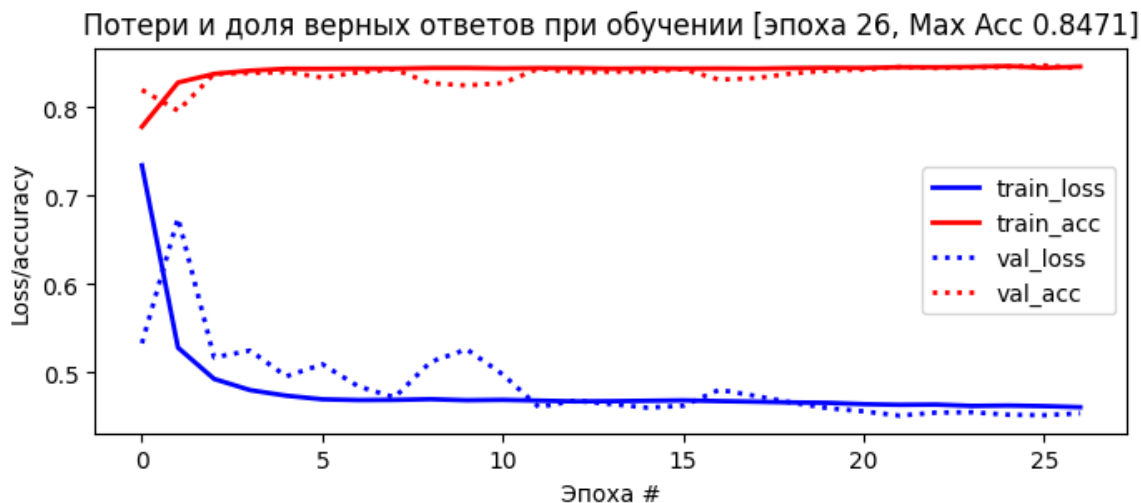
Пакетная нормализация (batch normalization)

- Пакетная нормализация нормализует выходы предыдущего слоя для каждого пакета
 - Внутри пакета установите среднюю активацию, близкую к 0, и стандартное отклонение, близкое к 1
 - При переходе к обработке другого пакета используется экспоненциальное скользящее среднее и дисперсия пакетов
 - Пакетная нормализация позволяет более глубоким сетям быть менее склонными к исчезновению или взрыву градиентов

```

In [32]: model = models.Sequential()
model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(265, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
plot_losses = TrainingPlot())
earlystop = callbacks.EarlyStopping(monitor='val_loss', patience=5)
history = model.fit(Xf_train, yf_train, epochs=50, batch_size=512,
validation_data=(X_val, y_val), callbacks=[plot_losses, earlystop])

```



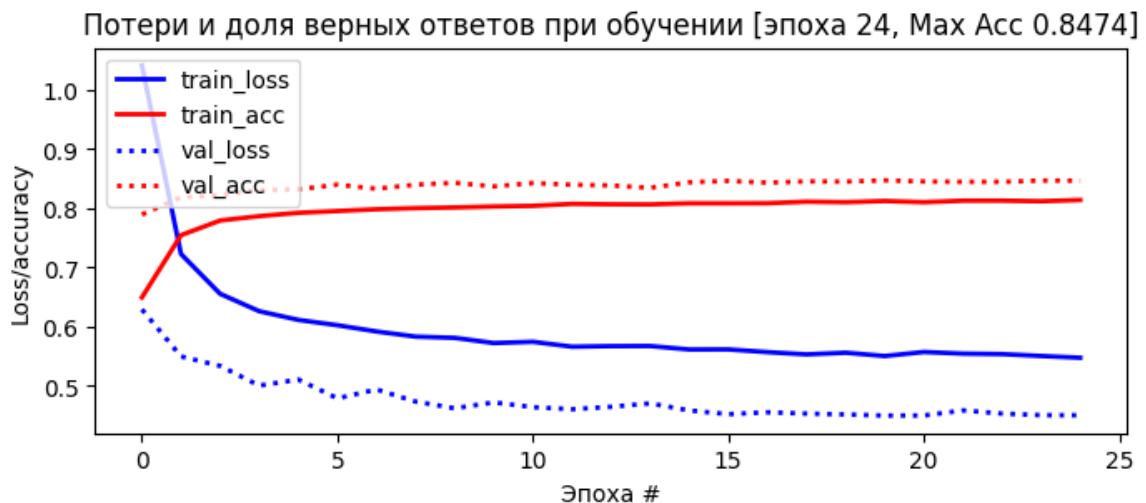
Комбинирование нескольких регуляризаторов

Ведутся споры о том, имеет ли смысл объединять несколько регуляризаторов и в каком порядке. Что работает (или нет) зависит от структуры и размера сети и имеющегося набора данных.

Например, поскольку пакетная нормализация уже выполняет некоторую регуляризацию, Dropout может не понадобиться. Однако иногда такая комбинация действительно помогает. Иногда помогает использование Dropout после пакетной нормализации только на самых глубоких уровнях.

Пакетная нормализация иногда выполняется перед плотным слоем, но в целом она работает лучше, если применяется после плотного слоя. Аналогично, Dropout можно применить до или после пакетной нормализации. Однако использование Dropout перед пакетной нормализацией приведет к включению нулей в статистику нормализации, что нежелательно.

```
In [33]: network = models.Sequential()
network.add(layers.InputLayer(shape=(28 * 28,)))
network.add(layers.Dense(265, activation='relu'))
network.add(layers.BatchNormalization())
network.add(layers.Dropout(0.3))
network.add(layers.Dense(64, activation='relu'))
network.add(layers.BatchNormalization())
network.add(layers.Dropout(0.3))
network.add(layers.Dense(32, activation='relu'))
network.add(layers.BatchNormalization())
network.add(layers.Dropout(0.3))
network.add(layers.Dense(10, activation='softmax'))
network.compile(loss='categorical_crossentropy', optimizer='rmsprop',
plot_losses = TrainingPlot())
earlystop = callbacks.EarlyStopping(monitor='val_loss', patience=5)
history = network.fit(Xf_train, yf_train, epochs=50, batch_size=512,
validation_data=(X_val, y_val), callbacks=[earlystop])
```

Настройка множественных гиперпараметров

- Модуль Keras имеет соответствующую библиотеку настройки [keras-tuner](#) с несколькими методами настройки параметров:
- Случайный поиск (RandomSearch)
- Гипербанд (Hyperband)
- Байесовская оптимизация
- Sklearn (для настройки моделей scikit-learn)
- Модуль keras-tuner создает папку со всеми результатами для каждого проекта (параметр `project_name`). Нужно будет удалить папку или изменить имя проекта, чтобы запустить его снова.

```
In [34]: #!pip install -q -U keras-tuner
```

```
In [35]: from keras import optimizers
import keras_tuner as kt

def build_model(hp):
    model = models.Sequential()

    # Настроим число нейронов в плотных слоях.
    # Выберем оптимальное значение между 32-512.
    hp_units = hp.Int('units', min_value = 32, max_value = 512, step=16)

    model.add(keras.layers.Dense(units = hp_units, activation = 'relu',
                                  input_shape=(28 * 28,)))
    model.add(keras.layers.Dense(units = hp_units, activation = 'relu'))
    model.add(keras.layers.Dense(10))

    # Настроим шаг обучения для оптимизатора
    # Выберем оптимальное значение между 0.01, 0.001 или 0.0001
    hp_learning_rate = hp.Choice('learning_rate', values = [1e-2, 1e-3, 1e-4])

    model.compile(optimizer = optimizers.Adam(learning_rate = hp_learning_rate))
```

```

        loss = 'categorical_crossentropy',
        metrics = ['accuracy'])
    return model

tuner = kt.RandomSearch(build_model, max_trials=5, objective = 'val_
                        project_name='lab02')

```

```

In [36]: # выполнение кода может занять определенное время
tuner.search(Xf_train, yf_train, epochs = 10, validation_data = (X_
            callbacks = [TrainingPlot()])

# получить оптимальные гиперпараметры
best_hps = tuner.get_best_hyperparameters(num_trials = 1)[0]
best_hps.values

```

```

Trial 5 Complete [00h 02m 01s]
val_accuracy: 0.10000000149011612

```

```

Best val_accuracy So Far: 0.17910000681877136
Total elapsed time: 00h 10m 01s

```

```

Out[36]: {'units': 160, 'learning_rate': 0.0001}

```

- Вы можете обернуть модели Keras как модели scikit-learn, используя [KerasClassifier](#) и использовать любую технику настройки.

```

In [37]: #from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
# pip install scikeras --upgrade
from scikeras.wrappers import KerasClassifier

def build_model(var_activation='relu',var_optimizer='adam'):
    """ Для построения модели Keras используются аргументы функции.
    model = models.Sequential()
    model.add(layers.InputLayer(shape=(28 * 28,)))
    model.add(layers.Dense(64,activation=var_activation))
    model.add(layers.Dense(32,activation=var_activation))
    model.add(layers.Dense(16,activation=var_activation))
    model.add(layers.Dense(10,activation='softmax'))
    model.compile(loss="categorical_crossentropy",
                  optimizer=var_optimizer,
                  metrics=["accuracy"])
    return model

# пространство поиска
_activations=['tanh','relu','selu']
_optimizers=['sgd','adam']
_batch_size=[16,32,64]
params=dict(var_activation=_activations,
            var_optimizer=_optimizers,
            batch_size=_batch_size)

# обертка
model = KerasClassifier(model=build_model,epochs=4,batch_size=16,
                        var_optimizer='adam',var_activation='relu')

```

```
In [38]: from sklearn.model_selection import RandomizedSearchCV




















# выполнение кода занимает время
rscv = RandomizedSearchCV(model, param_distributions=params, cv=2,
                          n_iter=3, verbose=1, n_jobs=-1)
rscv_results = rscv.fit(Xf_train,yf_train)
```

Fitting 2 folds for each of 3 candidates, totalling 6 fits

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1757929030.539338 3696403 pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
I0000 00:00:1757929030.539374 3696403 pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1757929030.539509 3696410 pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
I0000 00:00:1757929030.539532 3696410 pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1757929030.543337 3696399 pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
I0000 00:00:1757929030.543368 3696399 pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1757929030.552322 3696414 pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
I0000 00:00:1757929030.552344 3696414 pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1757929030.552890 3696406 pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
I0000 00:00:1757929030.552907 3696406 pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1757929030.555793 3696393 pluggable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.
I0000 00:00:1757929030.555814 3696393 pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)

Epoch 1/4

Epoch 1/4

Epoch 1/4
Epoch 1/4
Epoch 1/4
Epoch 1/4
782/782  **12s** 15ms/step - accuracy: 0.7346 - loss: 0.7763
Epoch 2/4
782/782  **12s** 15ms/step - accuracy: 0.7446 - loss: 0.7578
Epoch 2/4
782/782  **13s** 15ms/step - accuracy: 0.7870 - loss: 0.6749
Epoch 2/4
782/782  **13s** 15ms/step - accuracy: 0.7831 - loss: 0.68103
Epoch 2/4
1563/1563  **16s** 10ms/step - accuracy: 0.6450 - loss: 0.9478
Epoch 2/4
1563/1563  **16s** 10ms/step - accuracy: 0.6568 - loss: 0.9654
Epoch 2/4
782/782  **11s** 15ms/step - accuracy: 0.8136 - loss: 0.5228
Epoch 3/4
782/782  **11s** 15ms/step - accuracy: 0.8153 - loss: 0.5246
Epoch 3/4
782/782  **12s** 15ms/step - accuracy: 0.8456 - loss: 0.4424
Epoch 3/4
782/782  **12s** 15ms/step - accuracy: 0.8486 - loss: 0.43966
141/782  **9s** 15ms/step - accuracy: 0.8169 - loss: 0.4876
Epoch 3/4
1563/1563  **16s** 10ms/step - accuracy: 0.7742 - loss: 0.6170
Epoch 3/4
1563/1563  **15s** 10ms/step - accuracy: 0.7855 - loss: 0.6022
Epoch 3/4
782/782  **11s** 15ms/step - accuracy: 0.8272 - loss: 0.47682
Epoch 4/4
782/782  **11s** 15ms/step - accuracy: 0.8319 - loss: 0.47899
Epoch 4/4
782/782  **12s** 15ms/step - accuracy: 0.8601 - loss: 0.3950
Epoch 4/4
782/782  **12s** 15ms/step - accuracy: 0.8595 - loss: 0.3918
Epoch 4/4
782/782  **11s** 15ms/step - accuracy: 0.8394 - loss: 0.4472
782/782  **11s** 15ms/step - accuracy: 0.8392 - loss:

```

s: 0.4504
1563/1563 ————— 15s 10ms/step - accuracy: 0.8079 - lo
ss: 0.5486
Epoch 4/4
1563/1563 ————— 15s 10ms/step - accuracy: 0.8000 - lo
ss: 0.5604
Epoch 4/4
782/782 ————— 3s 4ms/stepstep - accuracy: 0.8026 - lo
ss: 0.543
782/782 ————— 12s 15ms/step - accuracy: 0.8698 - los
s: 0.3646
782/782 ————— 12s 15ms/step - accuracy: 0.8731 - los
s: 0.35642
782/782 ————— 3s 4ms/stepstep - accuracy: 0.8138 - lo
ss: 0.
782/782 ————— 1s 1ms/stepstep - accuracy: 0.8140 - los
s: 0.527
782/782 ————— 1s 1ms/stepstep - accuracy: 0.8140 - los
s: 0.52
1563/1563 ————— 13s 8ms/step - accuracy: 0.8030 - los
s: 0.5740
1563/1563 ————— 13s 8ms/step - accuracy: 0.8098 - los
s: 0.5330
1563/1563 ————— 2s 1ms/step
1563/1563 ————— 2s 1ms/step
Epoch 1/4
1563/1563 ————— 17s 11ms/step - accuracy: 0.8119 - lo
ss: 0.5710
Epoch 2/4
1563/1563 ————— 16s 10ms/step - accuracy: 0.8590 - lo
ss: 0.3940
Epoch 3/4
1563/1563 ————— 16s 10ms/step - accuracy: 0.8720 - lo
ss: 0.3540
Epoch 4/4
1563/1563 ————— 17s 11ms/step - accuracy: 0.8791 - lo
ss: 0.3328

```

```
In [39]: print('Лучший результат равен: {} при использовании параметров {}'.format(
    rscv_results.best_score_, rscv_results.best_params_))
```

Лучший результат равен: 0.85978 при использовании параметров {'var_optimizer': 'adam', 'var_activation': 'tanh', 'batch_size': 32}

Функциональный интерфейс Keras

В модуле **Keras** имеется два интерфейса (API) для быстрого построения архитектур нейронных сетей: **последовательный интерфейс** (Sequential API) и **функциональный интерфейс** (Functional API).

Первый интерфейс позволяет строить только последовательные архитектуры нейронных сетей, в которых выход каждого слоя передается на вход следующего слоя.

При помощи функционального интерфейса можно задать нейронную сеть в виде произвольного направленного ациклического графа (DAG или directed acyclic graph), что дает намного больше возможностей для построения сложных моделей. **Направленный ациклический граф** (DAG) — это ориентированный граф, в котором отсутствуют циклы, но могут быть «параллельные» пути, выходящие из одного узла и разным образом приходящие в конечный узел. В частности, функциональный интерфейс может обрабатывать модели с нелинейной топологией, модели с общими слоями, и модели с несколькими входами или выходами.

В качестве примера рассмотрим простую нейронную сеть, созданную при помощи последовательного интерфейса:

```
In [40]: model = models.Sequential()
model.add(layers.InputLayer(shape=(28 * 28,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

или через список слоев:

```
In [41]: model = models.Sequential([
    layers.InputLayer(shape=(28 * 28,)),
    layers.Dense(512, activation='relu'),
    layers.Dense(512, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Воссоздадим эту нейронную сеть при помощи функционального интерфейса. Сначала создаем входные данные нейронной сети:

```
In [42]: inputs = layers.Input(shape=(28 * 28,))
```

Здесь указывается размерность данных, при этом количество данных всегда опускается. Переменная `inputs` содержит информацию о размерах и типе данных которые будут передаваться в модель:

```
In [43]: inputs.shape, inputs.dtype
```

```
Out[43]: ((None, 784), 'float32')
```

Создаем новый слой в графе слоев с `inputs` в качестве входных данных:

```
In [44]: x = layers.Dense(512, activation='relu')(inputs)
```

Добавим еще один слой в граф слоев:

```
In [45]: x = layers.Dense(512, activation='relu')(x)
```

Наконец, добавим последний слой:

```
In [46]: outputs = layers.Dense(10, activation='softmax', name='OutputLayer')
```

Теперь создаем модель, указав ее входы и выходы в графе слоев:

```
In [47]: model2 = keras.Model(inputs=inputs, outputs=outputs)
```

Сравним две модели:

```
In [48]: model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	
dense_10 (Dense)	(None, 512)	
dense_11 (Dense)	(None, 512)	
dense_12 (Dense)	(None, 10)	

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

```
In [49]: model2.summary()
```

Model: "functional_11"

Layer (type)	Output Shape	
input_layer_4 (InputLayer)	(None, 784)	
dense_13 (Dense)	(None, 512)	
dense_14 (Dense)	(None, 512)	
OutputLayer (Dense)	(None, 10)	

Total params: 669,706 (2.55 MB)

Trainable params: 669,706 (2.55 MB)

Non-trainable params: 0 (0.00 B)

Задание 2 по теме №1

Загрузите из `keras.datasets` набор данных California Housing price regression dataset (https://keras.io/api/datasets/california_housing/), обучите нейронную сеть прогнозировать медианную цену домов в зависимости

от количества комнат в доме, визуализируйте процесс обучения.

In []:

