

# Машинное обучение, DS-поток

## Домашнее задание 8

### Правила:

- Дедлайн **30 апреля 23:59**. После дедлайна работы не принимаются кроме случаев наличия уважительной причины.
- Выполненную работу нужно отправить на почту `mipt.stats@yandex.ru`, указав тему письма "[ml] Фамилия Имя - задание 8". Квадратные скобки обязательны. Если письмо дошло, придет ответ от автоответчика.
- Прислать нужно ноутбук и его pdf-версию (без архивов). Названия файлов должны быть такими: `name.N.ipynb` и `name.N.pdf`, где `N` - ваш номер из таблицы с оценками, а `name` - имя ноутбука.
- Решения, размещенные на каких-либо интернет-ресурсах не принимаются. Кроме того, публикация решения в открытом доступе может быть приравнена к предоставлению возможности списать.
- Для выполнения задания используйте этот ноутбук в качестве основы, ничего не удаляя из него.
- Код из данного задания при проверке может быть запущен.

Welcome в Глубокое обучение!



В этом домашнем задании вам предстоит самостоятельно реализовать самые важные компоненты нейросети, **используя только NumPy**.

Структура задания подразумевает выполнение 3-х jupyter notebook 'ов:

**[0]task8\_train\_model.ipynb** : текущий ноутбук. В нем поясняется суть задания, описаны остальные ноутбуки и именно этот ноутбук нужно запустить, когда все слои уже реализованы;

**[1]task8\_modules.ipynb** : ноутбук, в котором нужно непосредственно реализовать слои нейронной сети. А именно:

В этом ноутбуке вам предстоит реализовать слои нейронной сети, используя только библиотеку NumPy:

Базовые концепции:

- ☒ **Module** — абстрактный класс для компонент нейронной сети;
- ☐ (2 балла) **Sequential** — класс, содержащий в себе последовательность объектов класса **Module**.

Слои:

- ☐ (2 балла) **Linear** — линейный слой;
- ☐ (3 балла) **SoftMax** — слой, вычисляющий операцию *softmax*;
- ☒ **LogSoftMax** — слой, вычисляющий операцию *log(softmax)*;
- ☐ (2 балла) **Dropout** — слой дропаута;
- ☐ (3 балла) **BatchNormalization** — слой для работы слоя батч-нормализации;
- ☒ **Scaling** — слой для работы слоя батч-нормализации;
- ☒ **Flatten** — слой, который просто "разворачивает" тензор любой размерности в одномерный вектор.

Функции активации (тоже являются слоями, но выделены в отдельную секцию для удобства):

- ☒ **ReLU** — функция активации *Rectified Linear Unit*;
- ☐ (1 балл) **LeakyReLU** — функция активации *Leaky Rectified Linear Unit*;
- ☐ (1 балл) **ELU** — функция активации *Exponential Linear Unit*;
- ☐ (1 балл) **Softplus** — функция активации *Softplus*.

Функции потерь:

- ☒ **Criterion** — абстрактный класс для функций потерь;
- ☐ (1 балл) **MSECriterion** — среднеквадратичная функция потерь;
- ☐ (1 балл) **NLLCriterionUnstable** — negative log-likelihood функция потерь (нестабильная версия, возможны числовые переполнения);
- ☒ (1 балл) **NLLCriterion** — negative log-likelihood функция потерь (стабильная версия).

Оптимизаторы:

- ☐ (2 балла) SGD — метод стохастического градиентного спуска (включая momentum).

**Галочками** помечены те слои, которые **уже реализованы за вас**. Таким образом, задание состоит в реализации оставшихся слоёв (мы заботимся о вашем здоровье 🍀).

**[2]task8\_test\_modules.ipynb** : ноутбук с юнит-тестами, который следует использовать для отладки ноутбука **[1]task8\_modules.ipynb** .

**Всего за задание можно получить:**

- **20 баллов** за **[1]task8\_modules.ipynb**
- **10 баллов** за **[0]task8\_train\_model.ipynb** (этот ноутбук)
- Суммарно: **30 баллов**

---

## 1. Использование NumPy-фреймворка (10 баллов = 5 + 5)

Следующая ячейка делает `from [1]task8_modules.ipynb import *`, таким образом позволяя этому ноутбуку видеть все слои:

In [ ]:

```
1 %run [1]task8_modules.ipynb
```

После того, как все слои в `[1]task8_modules.ipynb` протестированы и работают корректно (напомним, что для этого нужно пройти все `assert` 'ы в **[2]task8\_test\_modules.ipynb** ), проверим нейросеть сначала на наборах синтетических данных, а потом на датасете `FashionMNIST` .

In [ ]:

```
1 import time
2 from time import time, sleep
3 import warnings
4
5 from itertools import cycle, islice
6
7 from IPython import display
8
9 import numpy as np
10 import pandas as pd
11
12 import scipy.stats as sps
13
14 import matplotlib.pyplot as plt
15 from matplotlib.colors import ListedColormap
16 %matplotlib inline
17
18 import seaborn as sns
19 sns.set(font_scale=1.5)
20
21 cm = plt.cm.RdBu
22 cm_bright = ListedColormap(['#FF3300', '#00CC66'])
23
24 from sklearn import cluster, datasets
25 from sklearn.metrics import accuracy_score
26 from sklearn.preprocessing import StandardScaler
27
28 import torch
29 import torch.nn as nn
30 from torch.optim import SGD as torch_sgd
31
32 RANDOM_SEED = 42
33
34 np.random.seed(RANDOM_SEED)
```

## 1.1. Синтетические данные (5 баллов = 3 + 2)

### Линейно разделимая выборка (3 балла)

Обучим однослойную нейронную сеть решать линейно-разделимую классификацию на 2 класса:

In [ ]:

```
1  n_samples = 1024
2  n_classes = 2
3
4  ▾ varied_blobs = datasets.make_blobs(
5      n_samples=n_samples,
6      n_features=2,
7      centers=n_classes,
8      cluster_std=[1.0, 2.5],
9      random_state=RANDOM_SEED
10 )
11
12 sample, labels = varied_blobs
```

In [ ]:

```
1  plt.figure(figsize=(12,7))
2  plt.title('Линейно разделимая выборка')
3  plt.xlabel('Признак 1')
4  plt.ylabel('Признак 2')
5  plt.scatter(sample[:,0], sample[:,1], c=labels, cmap=cm_bright, alpha=0.6);
```

Генератор батчей (помним, что нейросети обучаются итеративно — по батчам):

In [ ]:

```
1  ▾ def train_generator(sample, labels, batch_size):
2      ...
3      Генератор батчей.
4      На каждом шаге возвращает `batch_size` объектов из `sample` и их
5      меток из `labels`.
6      ...
7
8      n_samples = sample.shape[0]
9      # Перемешиваем в случайном порядке в начале эпохи
10     indices = np.arange(n_samples)
11     np.random.shuffle(indices)
12
13     # Обратите внимание на yield вместо return
14     # (если раньше не сталкивались с генераторами)
15  ▾     for start in range(0, n_samples, batch_size):
16         end = min(start + batch_size, n_samples)
17         batch_idx = indices[start:end]
18         yield sample[batch_idx], labels[batch_idx]
```

Функция для удобного обучения модели:

In [ ]:

```
1  ▾ def train_model(  
2      model,  
3      sample, y,  
4      criterion,  
5      opt_params,  
6      opt_state,  
7      n_epoch,  
8      batch_size  
9  ▾ ):  
10     ...  
11     Обучает модель из вашего мини-фреймворка.  
12     Возвращает обученную модель, историю значений функции потерь  
13     и метрики качества.  
14  
15     :param `model`: модель из вашего мини-фреймворка  
16     :param `sample`: матрица объектов  
17     :param `y`: вектор истинных меток объектов  
18     :param `criterion`: функция потерь  
19     :param `opt_params`: гиперпараметры оптимизатора  
20  ▾ :param `opt_state`: текущая информация о градиентах,  
21         хранящаяся в оптимизаторе  
22     :param `n_epoch`: количество эпох  
23     :param `batch_size`: размер одного батча  
24     ...  
25  
26     loss_history = []  
27  
28  ▾ for i in range(n_epoch):  
29  ▾     for x_batch, y_batch in train_generator(sample, y, batch_size):  
30         # Обнуляем градиенты с предыдущей итерации  
31         # Forward pass  
32         # Backward pass  
33         <ВАШ КОД ЗДЕСЬ>  
34         # Обновление весов  
35  ▾     SGD(model.get_parameters(),  
36           model.get_grad_params(),  
37           opt_params,  
38           opt_state)  
39         loss_history.append(loss)  
40  
41         display.clear_output(wait=True)  
42  
43         plt.figure(figsize=(8, 6))  
44         plt.title("Функция потерь на train")  
45         plt.xlabel("итерация")  
46         plt.ylabel("лосс")  
47         plt.plot(loss_history, 'b')  
48         plt.show()  
49         print('Current loss: %f' % loss)  
50  
51     return model, loss_history
```

Построим однослойную нейросеть для классификации: размер слоя  $2 \times 2$ , так как 2 признака и 2 класса. В качестве последнего слоя рекомендуется использовать `LogSoftMax`.

In [ ]:

```
1 net = <ВАШ КОД ЗДЕСЬ>
2
3 print(net)
```

Объявим оптимизируемую функцию потерь и гиперпараметры:

In [ ]:

```
1 ▾ # Функция потерь
2 criterion = <ВАШ КОД ЗДЕСЬ>
3
4 # Гиперпараметры оптимизатора
5 optimizer_config = {'learning_rate': 1e-2, 'momentum': 0.9}
6 optimizer_state = {}
7
8 # Гиперпараметры цикла обучения и генератора
9 n_epoch = 30
10 batch_size = 128
```

Проверим, что кодирование производится верно:

In [ ]:

```
1 y = np.hstack([1-labels[:,None], labels[:,None]])
2 print(y.shape)
3 print(labels[:10])
4 print(y[:10])
```

Обучим модель с помощью функции `train_model` :

In [ ]:

```
1 ▾ net, loss_history = train_model(
2     net,
3     sample, y,
4     criterion,
5     optimizer_config,
6     optimizer_state,
7     n_epoch,
8     batch_size
9 )
```

**Упражнение:** Попробуйте поменять гиперпараметр `learning_rate` в `optimizer_config` на `1e-1`, `1e-2`, `1e-3` и `1e-4`. Как это влияет на обучение?

Батч-генератор для тестовой выборки:

In [ ]:

```
1 ▾ def generate_grid(sample, h=0.02):
2     ''' Генерирует двумерную сетку. '''
3
4     x_min, x_max = sample[:, 0].min() - .5, sample[:, 0].max() + .5
5     y_min, y_max = sample[:, 1].min() - .5, sample[:, 1].max() + .5
6 ▾   xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
7                           np.arange(y_min, y_max, h))
8
9     return xx, yy
10
11 ▾ def test_generator(sample):
12     ''' Батч-генератор для тестовых данных (без меток). '''
13
14     n_samples = sample.shape[0]
15     indices = np.arange(n_samples)
16 ▾   for start in range(0, n_samples, batch_size):
17       end = min(start + batch_size, n_samples)
18       batch_idx = indices[start:end]
19       yield sample[batch_idx]
```

Функции для удобного тестирования модели:

In [ ]:

```
1 ▾ def test_model(model, test_sample):
2     ''' Тестирует модель на тестовой выборке. '''
3
4     preds = []
5     model.evaluate()
6 ▾   for test_batch in test_generator(test_sample):
7       batch = model.forward(test_batch)
8       batch = batch.argmax(axis=1).reshape(-1,1)
9       preds.append(batch)
10    preds = np.vstack(preds)
11    return preds
```

In [ ]:

```
1 ▾ def plot_grid_preds(sample, labels, xx, yy, preds, title):
2     ''' Функция для удобной отрисовки предсказаний
3         нейросети на двумерной сетке. '''
4
5     plt.figure(figsize=(12,7))
6     plt.title(title)
7     plt.xlabel('Признак 1')
8     plt.ylabel('Признак 2')
9 ▾   plt.scatter(sample[:,0], sample[:,1], c=labels,
10               cmap=cm_bright, alpha=0.55)
11   plt.contourf(xx, yy, preds, alpha=.2, cmap=cm)
12   plt.show();
```

Выведем предсказания модели на двумерной сетке:



In [ ]:

```
1  xx, yy = <ВАШ КОД ЗДЕСЬ>
2  test_sample = <ВАШ КОД ЗДЕСЬ>
3
4  test_predictions = <ВАШ КОД ЗДЕСЬ>
5  test_predictions = <ВАШ КОД ЗДЕСЬ>
6
7  plot_grid_preds(sample, labels, xx, yy,
8                  test_predictions, 'Линейно разделимая выборка')
```

Сравним с точно такой же моделью, но на PyTorch:

*Подсказка:* названия слоев идентичны, если брать их из `torch.nn`. Отличия только в оптимизаторе и названии функции потерь.

In [ ]:

```
1  net_torch = <ВАШ КОД ЗДЕСЬ>
2
3  print(net_torch)
```

In [ ]:

```
1  ▼ def train_model_torch(
2      model,
3      sample, y,
4      criterion,
5      optimizer,
6      n_epoch,
7      batch_size
8  ▼ ):
9      ...
10     Обучает модель из PyTorch.
11     Возвращает обученную модель, историю значений функции потерь
12     и метрики качества.
13
14     :param `model`: модель из PyTorch
15     :param `sample`: матрица объектов
16     :param `y`: вектор истинных меток объектов
17     :param `criterion`: функция потерь
18     :param `opt_params`: гиперпараметры оптимизатора
19  ▼   :param `opt_state`: текущая информация о градиентах,
20       хранящаяся в оптимизаторе
21     :param `n_epoch`: количество эпох
22     :param `batch_size`: размер одного батча
23     ...
24
25     loss_history = []
26
27  ▼   for i in range(n_epoch):
28  ▼       for x_batch, y_batch in train_generator(sample, y, batch_size):
29           # Обнуляем градиенты с предыдущей итерации
30           # Forward pass
31           # Backward pass
32           # Обновление весов
33           <ВАШ КОД ЗДЕСЬ>
34
35           loss_history.append(loss.data)
36
37           display.clear_output(wait=True)
38
39           plt.figure(figsize=(8, 6))
40           plt.title("Функция потерь на train")
41           plt.xlabel("итерация")
42           plt.ylabel("лосс")
43           plt.plot(loss_history, 'b')
44           plt.show()
45           print('Current loss: %f' % loss)
46
47     return model, loss_history
```

In [ ]:

```
1 criterion = <ВАШ КОД ЗДЕСЬ>
2 optimizer = torch_sgd(<ВАШ КОД ЗДЕСЬ>)
3
4 net_torch, loss_history = train_model_torch(
5     net_torch, sample, y,
6     criterion,
7     optimizer,
8     n_epoch,
9     batch_size
10 )
```

**Для самопроверки:** Значение функции потерь (лосса) должны быть примерно одинаковые у вашей модели и у сети из PyTorch. Каким именно на этом датасете — см таблицу ниже.

Архитектура	learning_rate	Значение лосса
Linear(2,2)	1e-2	0.1 и ниже

In [ ]:

```
1 def test_torch(model, test_sample):
2     '''
3     Тестирует модель из PyTorch на тестовой выборке.
4     '''
5
6     preds = []
7     model.eval()
8     with torch.no_grad():
9         for test_batch in test_generator(test_sample):
10             <ВАШ КОД ЗДЕСЬ>
11             <ВАШ КОД ЗДЕСЬ>
12         return preds
```

In [ ]:

```
1 xx, yy = <ВАШ КОД ЗДЕСЬ>
2 test_sample = <ВАШ КОД ЗДЕСЬ>
3
4 test_predictions = <ВАШ КОД ЗДЕСЬ>
5 test_predictions = <ВАШ КОД ЗДЕСЬ>
6
7 plot_grid_preds(
8     sample,
9     labels,
10    xx, yy,
11    test_predictions,
12    'Линейно разделимая выборка'
13 )
```

Рисунки с предсказаниями должны практически совпадать у вашей модели и у модели на PyTorch.

## Вложенные окружности (2 балла)

Проверим работу нейросети на более сложной выборке, которая линейно не разделима в исходном пространстве признаков:

In [ ]:

```
1 n_samples = 1024
2
3 noisy_circles = datasets.make_circles(
4     n_samples=n_samples,
5     factor=.5,
6     noise=.05
7 )
8
9 sample, labels = noisy_circles
```

In [ ]:

```
1 plt.figure(figsize=(12,7))
2 plt.title('Вложенные окружности')
3 plt.xlabel('Признак 1')
4 plt.ylabel('Признак 2')
5 plt.scatter(sample[:,0], sample[:,1], c=labels,
6             cmap=cm_bright, alpha=0.55);
```

Сначала попробуем обучить и протестировать однослойную неросеть:

Архитектура и гиперпараметры:

In [ ]:

```
1 net = <ВАШ КОД ЗДЕСЬ>
2 print(net)
3
4 # Функция потерь
5 criterion = <ВАШ КОД ЗДЕСЬ>
6
7 # Гиперпараметры оптимизатора
8 optimizer_config = {'learning_rate' : 1e-2, 'momentum': 0.9}
9 optimizer_state = {}
10
11 # Гиперпараметры цикла обучения и генератора
12 n_epoch = 30
13 batch_size = 128
```

In [ ]:

```
1 y = np.hstack([1-labels[:,None], labels[:,None]])
2 print(y.shape)
3 print(labels[:10])
4 print(y[:10])
```

Обучение модели:

In [ ]:

```
1 net, loss_history = <ВАШ КОД ЗДЕСЬ>
```

Предсказание на двумерной сетке:

In [ ]:

```
1 <ВАШ КОД ЗДЕСЬ>
2
3 plot_grid_preds(sample, labels, xx, yy,
4                  test_predictions, 'Вложенные окружности')
```

Вряд ли у вас получился лосс ниже 0.68 и адекватный рисунок. Это говорит о том, что один слой не может решить задачу (что логично).

Попробуем сделать **двухслойную** нейросеть. Не жалейте нейронов в скрытый слой:

*Подсказка:* не забудьте про нелинейности.

Архитектура и гиперпараметры:

In [ ]:

```
1 net = <ВАШ КОД ЗДЕСЬ>
2 print(net)
3
4 criterion = <ВАШ КОД ЗДЕСЬ>
5
6 optimizer_config = {'learning_rate' : 1e-2, 'momentum': 0.9}
7 optimizer_state = {}
8
9 n_epoch = 30
10 batch_size = 128
11
12 y = np.hstack([1-labels[:,None], labels[:,None]])
13 print(y.shape)
```

Обучение модели:

In [ ]:

```
1 net, loss_history = <ВАШ КОД ЗДЕСЬ>
```

Тестирование модели:

In [ ]:

```
1 <ВАШ КОД ЗДЕСЬ>
2
3 plot_grid_preds(sample, labels, xx, yy,
4                  test_predictions, 'Вложенные окружности')
```

**Упражнение:** Попробуйте менять количество нейронов в скрытом слое. Начиная с какого количества нейронов окружности начинают хорошо разделяться?

И снова сравним с PyTorch:

In [ ]:

```
1 net_torch = <ВАШ КОД ЗДЕСЬ>
2
3 print(net_torch)
4
5 criterion = <ВАШ КОД ЗДЕСЬ>
6 optimizer = torch_sgd(<ВАШ КОД ЗДЕСЬ>)
7
8 net_torch, loss_history = <ВАШ КОД ЗДЕСЬ>
```

In [ ]:

```
1 <ВАШ КОД ЗДЕСЬ>
2
3 plot_grid_preds(sample, labels, xx, yy,
4                  test_predictions, 'Вложенные окружности')
```

**Для самопроверки:** Значение функции потерь (лосса) должны быть примерно одинаковые у вашей модели и у сети из PyTorch. Каким именно на этом датасете — см. таблицу ниже.

Архитектура	learning_rate	Значение лосса
(2,100) -> ReLU -> (100,2)	1e-2	0.2 и ниже

Оба рисунка должны примерно совпадать и адекватно разделять окружности между собой (одна область должна быть "внутри" другой).

Выводы: ...

## 1.2. Многоклассовая классификация: FashionMNIST (5 баллов = 2 + 2 + 1)



FashionMNIST — это датасет, аналогичный MNIST'у (датасету рукописных цифр), но по своей сути чуть интереснее — вместо цифр здесь элементы одежды. Это, конечно, не [DeepFashion](https://github.com/switchablenorms/DeepFashion2) (<https://github.com/switchablenorms/DeepFashion2>), но для проверки работы слоев подойдет.

Вместе с заданием идет архив `fashionmnist.zip`, распакуйте его в папку с этим ноутбуком (архив

изначально скачан с [Kaggle Datasets \(https://www.kaggle.com/zalando-research/fashionmnist\)](https://www.kaggle.com/zalando-research/fashionmnist)).

In [2]:

```
1 ▾ # !unzip fashionmnist.zip
```

В данном случае уже есть .csv -файлы, содержащие в себе "развернутые" в вектор-строку картинки (признаки) и столбец `label` (метки). Каждая картинка имеет размер  $28 \times 28$ . Значения (интенсивности цвета) в каждом пикселе — целые числа от 0 до 255.

Значения меток классов:

- 0. T-shirt/top
- 1. Trouser
- 2. Pullover
- 3. Dress
- 4. Coat
- 5. Sandal
- 6. Shirt
- 7. Sneaker
- 8. Bag
- 9. Ankle boot

In [ ]:

```
1 train_df = pd.read_csv('fashion-mnist_train.csv')
2 train_df
```

In [ ]:

```
1 test_df = pd.read_csv('fashion-mnist_test.csv')
2 test_df
```

В трейне 60к картинок, в тесте 10к. В глубоком обучении и нейросетях в целом обычно не делают кросс-валидацию, потому что это долго и, следовательно, вычислительно затратно. Обычно выделяют `train`, `val` и `test` сеты: на `train` обучают, на `val` валидируют модели на предмет переобучения и подбирают гиперпараметры, на `test` измеряют итоговое качество.

Убедимся, что данные уже перемешаны:

In [ ]:

```
1 train_df['label'].values[:10]
```

In [ ]:

```
1 test_df['label'].values[:10]
```

Как видим, данные уже перемешаны, так что просто выделим под валидацию 1/5 часть трейна:

In [ ]:

```
1 val_start_idx = -int(len(train_df)/5)
2
3 train_df = train_df[:val_start_idx]
4 val_df = train_df[val_start_idx:]
5
6 print(train_df.shape, val_df.shape)
```

Разделим признаки и метки:

In [ ]:

```
1 train_sample = train_df.drop(axis=1, labels=['label']).values
2 train_labels = train_df['label'].values
3 val_sample = val_df.drop(axis=1, labels=['label']).values
4 val_labels = val_df['label'].values
5 test_sample = test_df.drop(axis=1, labels=['label']).values
6 test_labels = test_df['label'].values
7
8 print('Train:', train_sample.shape, train_labels.shape)
9 print('Val:', val_sample.shape, val_labels.shape)
10 print('Test:', test_sample.shape, test_labels.shape)
```

Сразу преобразуем метки в OneHot:

In [ ]:

```
1 ▾ def make_onehot(labels):
2     '''
3     Практикуемся делать OneHot-кодирование на PyTorch
4     '''
5
6     num_classes = len(np.unique(labels))
7     y = torch.LongTensor(labels.astype('int')).view(-1, 1)
8     y_ohe = torch.FloatTensor(y.shape[0], num_classes)
9     y_ohe.zero_()
10    y_ohe.scatter_(1, y, 1)
11    return y_ohe
```

In [ ]:

```
1 y_train_ohe = make_onehot(train_labels).numpy()
2 y_val_ohe = make_onehot(val_labels).numpy()
3 y_test_ohe = make_onehot(test_labels).numpy()
4
5 print(train_labels.shape, y_train_ohe.shape)
6 print(train_labels[:10])
7 print(y_train_ohe[:10])
```

Поскольку мы получили `np.array` 'и, то можем использовать уже написанные функции `train_model` и `test_model` . Осталось только объявить архитектуру нейросети, оптимизатор и функцию потерь.

## Задание (2 балла)

Добейтесь точности не менее 0.85 на тестовой выборке. Используйте уже реализованные для синтетических данных функции `train_model` и `test_model` . Попробуйте добавить в архитектуру:



- Dropout
- BatchNorm
- LeakyReLU , ELU , SoftPlus
- Попробуйте разные связки выходного слоя и лосса: LogSoftmax->NLLCriterion и Softmax->NLLCriterionUnstable

Архитектура модели и гиперпараметры:

In [ ]:

```
1 num_features = train_sample.shape[1]
2 num_classes = y_train_ohe.shape[1]
3
4 net = <ВАШ КОД ЗДЕСЬ>
5 print(net)
6
7 criterion = <ВАШ КОД ЗДЕСЬ>
8
9 optimizer_config = {'learning_rate' : 1e-4, 'momentum': 0.9}
10 optimizer_state = {}
11
12 n_epoch = 10
13 batch_size = 128
```

Обучение модели:

In [ ]:

```
1 net, loss_history = <ВАШ КОД ЗДЕСЬ>
```

Предскажем на валидационной и тестовой выборке:

In [ ]:

```
1 val_predictions_my = <ВАШ КОД ЗДЕСЬ>
2 print(val_predictions_my.shape)
3 test_predictions_my = <ВАШ КОД ЗДЕСЬ>
4 print(test_predictions_my.shape)
```

In [ ]:

```
1 print(
2     f'Качество на Val моей собственной нейронной сети, \
3     обученной в течение {n_epoch} эпох: \
4     {accuracy_score(val_labels, val_predictions_my):.3}'
5 )
```

In [ ]:

```
1 print(
2     f'Качество на Test моей собственной нейронной сети, \
3     обученной в течение {n_epoch} эпох: \
4     {accuracy_score(test_labels, test_predictions_my):.3}'
5 )
```

Сравним с качеством аналогичной нейросети на PyTorch. Сначала обучим аналогичную torch-модель:

In [ ]:

```
1 net_torch = <ВАШ КОД ЗДЕСЬ>
2 print(net_torch)
3
4 criterion = <ВАШ КОД ЗДЕСЬ>
5 optimizer = torch_sgd(
6     <ВАШ КОД ЗДЕСЬ>
7 )
8
9 net_torch, loss_history = <ВАШ КОД ЗДЕСЬ>
```

Сделаем предсказания и посчитаем метрику качества:

In [ ]:

```
1 val_predictions_torch = <ВАШ КОД ЗДЕСЬ>
2 print(val_predictions_torch.shape)
3 test_predictions_torch = <ВАШ КОД ЗДЕСЬ>
4 print(test_predictions_torch.shape)
```

In [ ]:

```
1 print(
2     f'Качество на Val нейронной сети из PyTorch, \
3     в течение {n_epoch} эпох: \
4     {accuracy_score(val_labels, val_predictions_torch):.3}'
5 )
```

In [ ]:

```
1 print(
2     f'Качество на Test нейронной сети из PyTorch, \
3     обученной в течение {n_epoch} эпох: \
4     {accuracy_score(test_labels, test_predictions_torch):.3}'
5 )
```

**Для самопроверки:** Значение функции потерь (лосса) должны быть примерно одинаковые у вашей модели и у сети из PyTorch. Каким именно на этом датасете — см. таблицу ниже.

Архитектура	learning_rate	Значение лосса
(784,128)->ReLU->(128,128)->ReLU->(128,10)	1e-4	0.5 и ниже

Ассигу должно примерно совпадать и быть больше 0.85.

**Упражнение:** Сейчас мы подавали в нейросеть "сырые" значения в пикселях. В компьютерном зрении обычно их масштабируют (нормализуют), чтобы значения во входном тензоре были от 0 до 1. Попробуйте поделить обучающую выборку на максимальное значение интенсивности пикселя и обучить сети (не забудьте отмасштабировать значения в тестовых тензорах тоже). Улучшилось/ухудшилось итоговое качество нейросетей? Как вы думаете, почему?

Также, как и всегда ранее, полезно поиграться со значениями гиперпараметров (количество нейронов в скрытом слое, learning\_rate, momentum).

## Задание (2 балла)

Является ли разница в качестве вашей нейросети и сети на PyTorch статистически значимой? Проверьте, написав код/формулы в ячейках ниже.

In [ ]:

```
1 <ВАШ КОД ЗДЕСЬ>
```

In [ ]:

```
1 ...
```

Вывод: ...

## Задание (1 балл)

Выберите случайные 5 картинок и предскажите для них вероятности. Нарисуйте для каждой картинке в строчку:

1. Саму картинку
2. Гистограмму оценок вероятностей, которые получаются на выходе

*Примечание:* Удобно вывести по оси X названия классов вместо чисел. Соответствие классов номерам дано в ячейке ниже.

In [ ]:

```
1 id_to_name = {
2     0: 'T-shirt',
3     1: 'Trouser',
4     2: 'Pullover',
5     3: 'Dress',
6     4: 'Coat',
7     5: 'Sandal',
8     6: 'Shirt',
9     7: 'Sneaker',
10    8: 'Bag',
11    9: 'Ankle boot'
12 }
13
14 def plot_image_hist(pixel_vector, probas, true_label=None, figsize=(20,4)):
15     <ВАШ КОД ЗДЕСЬ>
```

In [ ]:

```
1 <ВАШ КОД ЗДЕСЬ>
```

## 2. Послесловие

Возможно, вам кажется, что вы построили какую-то игрушечную нейросеть, которая неспособна работать в "реальных" задачах. **Это не так.**

То, что вы реализовали, по сути составляет основную часть фреймворка PyTorch. Да, там реализовано еще много трюков для более эффективных вычислений, `autograd` и работа с GPU, но суть та же — модули, где каждый предоставляет `forward` и `backward` (с помощью `autograd`).

В этом задании вы оперировали в основном полносвязными ( Linear ) слоями, чаще их называют Fully-Connected (FC). Они активно применяются и по сей день:

- В задачах компьютерного зрения (CV): как слои классификации в "голове" сверточных нейросетей
- В задачах обработки естественного языка (NLP): как слои attention'a в Transformer'e
- В Reinforcement Learning (RL) иногда делают архитектуры исключительно из FC-слоев

### 3. Полезные ссылки

При составлении этого Домашнего задания авторы вдохновлялись [заданием](https://github.com/yandexdataschool/Practical_DL/tree/spring2019/homework01) ([https://github.com/yandexdataschool/Practical\\_DL/tree/spring2019/homework01](https://github.com/yandexdataschool/Practical_DL/tree/spring2019/homework01)) из курса "Deep Learning" Школы Анализа Данных.

- [Заметки главы ИИ в Tesla Andrej Karpathy по обучению нейросетей](http://karpathy.github.io/2019/04/25/recipe/) (<http://karpathy.github.io/2019/04/25/recipe/>)
- [Backpropagation на brilliant.org](https://brilliant.org/wiki/backpropagation/) (<https://brilliant.org/wiki/backpropagation/>)
- См. ссылки в `[1]task8_modules.ipynb`