Школа глубокого обучения ФПМИ МФТИ

Домашнее задание. Полносвязные и свёрточные нейронные сети

В этом занятии вам предстоит потренироваться построению нейронных сетей с помощью библиотеки Pytorch. Делать мы это будем на нескольких датасетах.

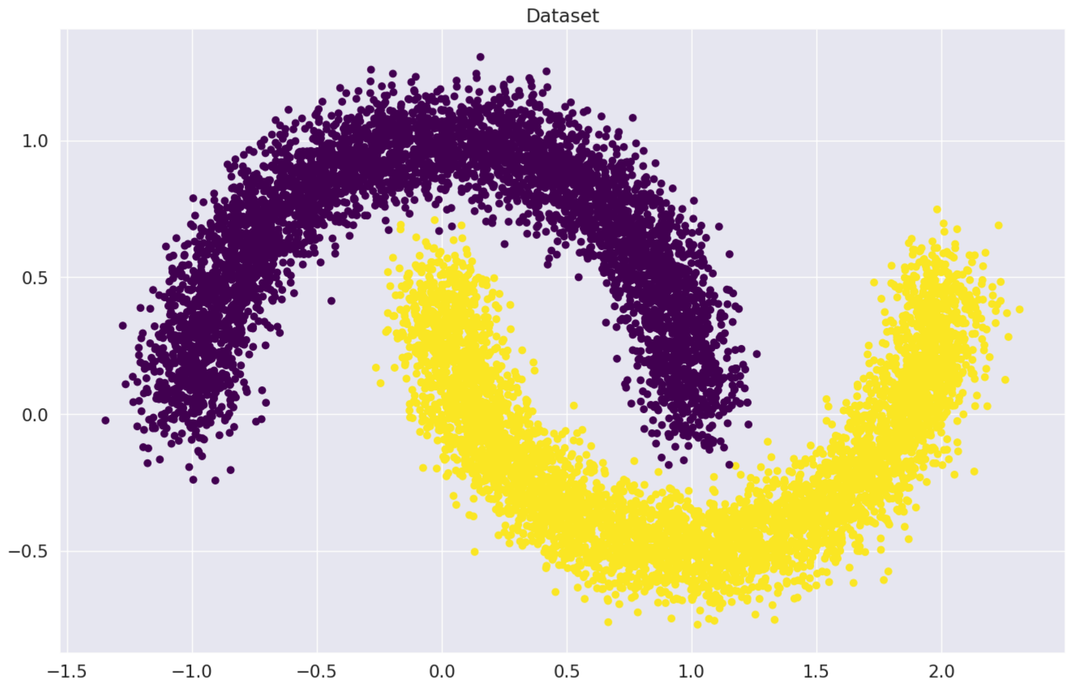
```
import numpy as np

import seaborn as sns
from matplotlib import pyplot as plt

from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

import torch
from torch import nn
from torch.nn import functional as F

from torch.utils.data import TensorDataset, DataLoader

sns.set(style="darkgrid", font_scale=1.4)
```

Часть 1. Датасет moons

Давайте сгенерируем датасет и посмотрим на него!

```
X, y = make_moons(n_samples=10000, random_state=42, noise=0.1)
```

```
plt.figure(figsize=(16, 10))
plt.title("Dataset")
plt.scatter(X[:, 0], X[:, 1], c=y, cmap="viridis")
plt.show()
```



Сделаем train/test split

```
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=42)
```

Загрузка данных

В PyTorch загрузка данных как правило происходит налету (иногда датасеты не помещаются в оперативную память). Для этого используются две сущности `Dataset` и `DataLoader`.

1. `Dataset` загружает каждый объект по отдельности.
2. `DataLoader` группирует объекты из `Dataset` в батчи.

Так как наш датасет достаточно маленький мы будем использовать `TensorDataset`. Все, что нам нужно, это перевести из массива numpy в тензор с типом `torch.float32`.

Задание. Создайте тензоры с обучающими и тестовыми данными

```
X_train_t =  torch.FloatTensor(X_train)
y_train_t =  torch.FloatTensor(y_train).view(-1, 1)
X_val_t =  torch.FloatTensor(X_val)
y_val_t =  torch.FloatTensor(y_val).view(-1, 1)
```

Создаем `Dataset` и `DataLoader`.

```
train_dataset = TensorDataset(X_train_t, y_train_t)
val_dataset = TensorDataset(X_val_t, y_val_t)
train_dataloader = DataLoader(train_dataset, batch_size=128)
val_dataloader = DataLoader(val_dataset, batch_size=128)
```

Logistic regression is my profession

Напоминание Давайте вспоним, что происходит в логистической регрессии. На входе у нас есть матрица объект-признак X и столбец-вектор y – метки из $\{0, 1\}$ для каждого объекта. Мы хотим найти такую матрицу весов W и смещение b (bias), что наша модель $XW + b$ будет каким-то образом предсказывать класс объекта. Как видно на выходе наша модель может выдавать число в интервале от $(-\infty; \infty)$. Этот выход как правило называют "логитами" (logits). Нам необходимо перевести его на интервал от $[0; 1]$ для того, чтобы он выдавал нам вероятность принадлежности объекта к кассу один, также лучше, чтобы эта функция была монотонной, быстро считалась, имела производную и на $-\infty$ имела значение 0, а на $+\infty$ имела значение 1. Такой класс функций называется сигмоидью. Чаще всего в качестве сигмоида берут

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

Задание. Реализация логистической регрессии

Вам необходимо написать модуль на PyTorch реализующий $logits = XW + b$, где W и b – параметры (`nn.Parameter`) модели.
Иначе говоря, здесь мы реализуем своими руками модуль `nn.Linear` (в этом пункте его использование запрещено).
Инициализируйте веса нормальным распределением (`torch.randn`).

```
class LinearRegression(nn.Module):
    def __init__(self, in_features: int, out_features: int, bias: bool = True):
        super().__init__()
        self.weights = nn.Parameter(torch.randn(out_features,in_features,requires_grad=True,dtype=torch.float32))
        self.bias = bias
        if bias:
            self.bias_term = nn.Parameter(torch.randn(out_features,requires_grad=True,dtype=torch.float32))

    def forward(self, x):
        x =  x @ self.weights.t()
        if self.bias:
            x += self.bias_term
        return x
```

```
linear_regression = LinearRegression(2, 1)
loss_function = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(linear_regression.parameters(), lr=0.05)
```

Вопрос 1. Сколько обучаемых параметров у получившейся модели? Имеется в виду суммарное количество отдельных числовых переменных, а не количество тензоров.

```
list(linear_regression.parameters())
```

```
    [Parameter containing:
     tensor([[ 1.7111, -1.4783]], requires_grad=True),
     Parameter containing:
     tensor([-0.0394], requires_grad=True)]
```

Ответ 1: Обучаемых параметра два:

```
* weights

* bias_term
```

Train loop

Вот псевдокод, который поможет вам разобраться в том, что происходит во время обучения

```
for epoch in range(max_epochs):  # <---------- итерируемся по датасету несколько раз
    for x_batch, y_batch in dataset:  # <------ итерируемся по датасету. Так как мы используем SGD а не GD, то берем батчи заданного
        optimizer.zero_grad()  # <------------- обуляем градиенты модели
        outp = model(x_batch)  # <------------- получаем "логиты" из модели
        loss = loss_func(outp, y_batch)  # <--- считаем "лосс" для логистической регрессии
        loss.backward()  # <------------------- считаем градиенты
        optimizer.step()  # <------------------ делаем шаг градиентного спуска
```

```
        if convergence:  # <---------------------- в случае сходимости выходим из цикла
            break
```

В коде ниже добавлено логирование `accuracy` и `loss`.

Задание. Реализация цикла обучения

```
tol = 1e-3
losses = []
max_epochs = 100
prev_weights = torch.zeros_like(linear_regression.weights)
stop_it = False
for epoch in range(max_epochs):
    for it, (X_batch, y_batch) in enumerate(train_dataloader):
        optimizer.zero_grad()
        outp =  linear_regression(X_batch)
        loss =  loss_function(outp,y_batch)
        loss.backward()
        losses.append(loss.detach().flatten()[0])
        optimizer.step()
        probabilities =  torch.sigmoid(outp)
        preds = (probabilities > 0.5).type(torch.long)
        batch_acc = (preds.flatten() == y_batch).type(torch.float32).sum() / y_batch.size(0)

        if (it + epoch * len(train_dataloader)) % 100 == 0:
            print(f"Iteration: {it + epoch * len(train_dataloader)}\nBatch accuracy: {batch_acc}")
        current_weights = linear_regression.weights.detach().clone()
        if (prev_weights - current_weights).abs().max() < tol:
            print(f"\nIteration: {it + epoch * len(train_dataloader)}.Convergence. Stopping iterations.")
            stop_it = True
            break
        prev_weights = current_weights
    if stop_it:
        break
```

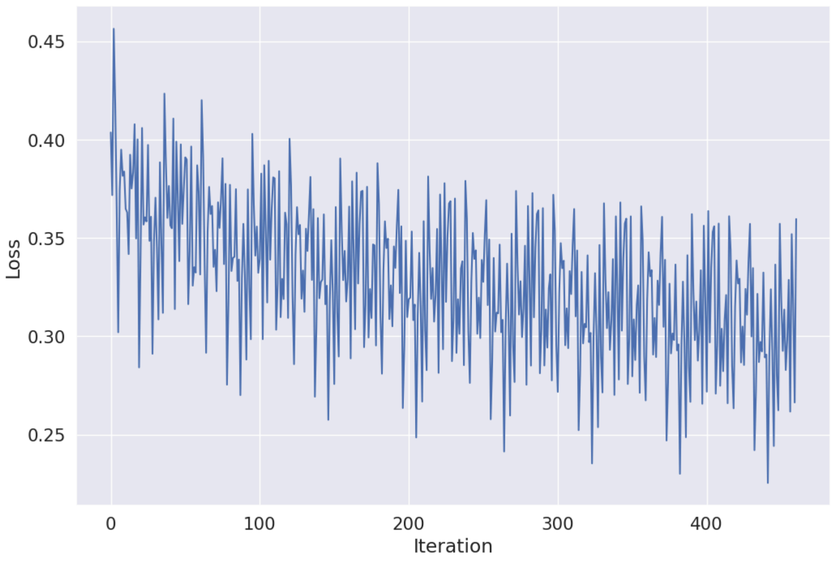
```
Iteration: 0
Batch accuracy: 63.8125
Iteration: 100
Batch accuracy: 64.75
Iteration: 200
Batch accuracy: 64.46875
Iteration: 300
Batch accuracy: 64.9375
Iteration: 400
Batch accuracy: 63.953125

Iteration: 460.Convergence. Stopping iterations.
```

Вопрос 2. Сколько итераций потребовалось, чтобы алгоритм сошелся?

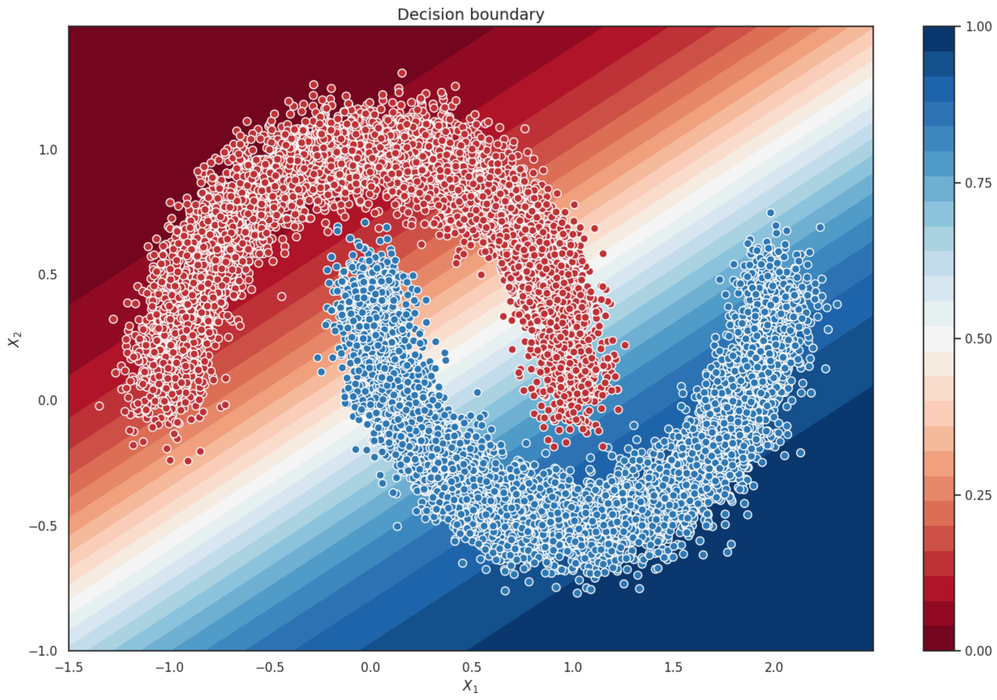
Ответ 2: Потребовалось 460 итераций, чтобы алгоритм сошелся

Визуализируем результаты

```
plt.figure(figsize=(12, 8))
plt.plot(range(len(losses)), losses)
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.show()
```



```
import numpy as np

sns.set(style="white")

xx, yy = np.mgrid[-1.5:2.5:.01, -1.:1.5:.01]
grid = np.c_[xx.ravel(), yy.ravel()]
batch = torch.from_numpy(grid).type(torch.float32)
with torch.no_grad():
    probs = torch.sigmoid(linear_regression(batch).reshape(xx.shape))
    probs = probs.numpy().reshape(xx.shape)

f, ax = plt.subplots(figsize=(16, 10))
ax.set_title("Decision boundary", fontsize=14)
contour = ax.contourf(xx, yy, probs, 25, cmap="RdBu",
                      vmin=0, vmax=1)
ax_c = f.colorbar(contour)
ax_c.set_label("$P(y = 1)$")
ax_c.set_ticks([0, .25, .5, .75, 1])

ax.scatter(X[100:,0], X[100:, 1], c=y[100:], s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1)

ax.set(xlabel="$X_1$", ylabel="$X_2$")
plt.show()
```



Задание. Реализуйте predict и посчитайте accuracy на test.

```
@torch.no_grad()
def predict(dataloader, model):
    model.eval()
    predictions = np.array([])
    for x_batch, _ in dataloader:
        outp = model(x_batch).flatten()
        preds = torch.sigmoid(outp)
        predictions = np.hstack((predictions, preds.numpy().flatten()))
    return predictions.flatten()
```

```
from sklearn.metrics import accuracy_score

y_val_pred = np.where(predict(val_dataloader, linear_regression)>0.5,1,0)
accuracy_score(y_val,y_val_pred)
```

```
0.8576
```

Вопрос 3

Какое `accuracy` получается после обучения?

Ответ 3: На тестовой выборке accuracy_score равен ~85.8%

Часть 2. Датасет MNIST

Датасет MNIST содержит рукописные цифры. Загрузим датасет и создадим DataLoader-ы. Пример можно найти в семинаре по полносвязным нейронным сетям.

```
import os
from torchvision.datasets import MNIST
from torchvision import transforms as tfs


data_tfs = tfs.Compose([
    tfs.ToTensor(),
    tfs.Normalize((0.5), (0.5))
])

# install for train and test
root = './'
train_dataset = MNIST(root, train=True,  transform=data_tfs, download=True)
val_dataset  = MNIST(root, train=False, transform=data_tfs, download=True)

train_dataloader =  DataLoader(train_dataset, batch_size=128, shuffle=True, num_workers=2)
valid_dataloader =  DataLoader(val_dataset, batch_size=128, shuffle=False, num_workers=2)
```

Часть 2.1. Полносвязные нейронные сети

Сначала решим MNIST с помощью полносвязной нейронной сети.

```
class Identical(nn.Module):
    def forward(self, x):
        return x
```

Задание. Простая полносвязная нейронная сеть

Создайте полносвязную нейронную сеть с помощью класса Sequential. Сеть состоит из:

- Уплощения матрицы в вектор (nn.Flatten);
- Двух скрытых слоёв из 128 нейронов с активацией nn.ELU;
- Выходного слоя с 10 нейронами.

Задайте лосс для обучения (кросс-энтропия).

```
train_dataloader.dataset.train_data.shape
```

```
    /usr/local/lib/python3.10/dist-packages/torchvision/datasets/mnist.py:75: UserWarning: train_data has been renamed data
      warnings.warn("train_data has been renamed data")
    torch.Size([60000, 28, 28])
```

```
valid_dataloader.dataset.test_data.shape
```

```
    /usr/local/lib/python3.10/dist-packages/torchvision/datasets/mnist.py:80: UserWarning: test_data has been renamed data
      warnings.warn("test_data has been renamed data")
    torch.Size([10000, 28, 28])
```

```
28*28
```

```
    784
```

```
activation = nn.ELU()

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 128),
    activation,
    nn.Linear(128,64),
    activation,
    nn.Linear(64,10)
)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

loaders = {"train": train_dataloader, "valid": valid_dataloader}
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
device
```

```
    'cuda'
```

Train loop (seriously)

Давайте разберемся с кодом ниже, который подойдет для 90% задач в будущем.

```
for epoch in range(max_epochs):  # <---------------- итерируемся по датасету несколько раз
    for k, dataloader in loaders.items():  # <----- несколько dataloader для train / valid / test
        for x_batch, y_batch in dataloader:  # <--- итерируемся по датасету. Так как мы используем SGD а не GD, то берем батчи заданн
            if k == "train":
                model.train()  # <------------------- переводим модель в режим train
                optimizer.zero_grad()  # <---------- обнуляем градиенты модели
                outp = model(x_batch)
                loss = criterion(outp, y_batch) # <-считаем "лосс" для логистической регрессии
                loss.backward()  # <---------------- считаем градиенты
                optimizer.step()  # <--------------- делаем шаг градиентного спуска
            else:  # <------------------------------- test/eval
                model.eval()  # <------------------- переводим модель в режим eval
                with torch.no_grad():  # <---------- НЕ считаем градиенты
                    outp = model(x_batch)  # <--------------- получаем "логиты" из модели
            count_metrics(outp, y_batch)  # <---------------- считаем метрики
```

Задание. Дополните цикл обучения.

```

```

```
max_epochs = 10
accuracy = {"train": [], "valid": []}
for epoch in range(max_epochs):
    for k, dataloader in loaders.items():
        epoch_correct = 0
        epoch_all = 0
        for x_batch, y_batch in dataloader:
            if k == "train":
                 # YOUR CODE. Set model to ``train`` mode and calculate outputs. Don't forget zero_grad!
                optimizer.zero_grad()
                outp = model(x_batch)
            else:
                 # YOUR CODE. Set model to ``eval`` mode and calculate outputs
                model.eval()
                with torch.no_grad():
                    outp = model(x_batch)
            preds = outp.argmax(-1)
            correct =  (preds == y_batch).sum() # YOUR CODE GOES HERE
            all =  y_batch.size(0) # YOUR CODE GOES HERE
            epoch_correct += correct.item()
            epoch_all += all
            if k == "train":
                loss = criterion(outp, y_batch)
                # YOUR CODE. Calculate gradients and make a step of your optimizer
                loss.backward()
                optimizer.step()

        if k == "train":
            print(f"Epoch: {epoch+1}")
        print(f"Loader: {k}. Accuracy: {epoch_correct/epoch_all}")
        accuracy[k].append(epoch_correct/epoch_all)
```

```
Epoch: 1
Loader: train. Accuracy: 0.88335
Loader: valid. Accuracy: 0.9313
Epoch: 2
Loader: train. Accuracy: 0.9413666666666667
Loader: valid. Accuracy: 0.9569
Epoch: 3
Loader: train. Accuracy: 0.9592833333333334
Loader: valid. Accuracy: 0.9621
Epoch: 4
Loader: train. Accuracy: 0.9681166666666666
Loader: valid. Accuracy: 0.9708
Epoch: 5
Loader: train. Accuracy: 0.9729166666666667
Loader: valid. Accuracy: 0.969
Epoch: 6
Loader: train. Accuracy: 0.9774166666666667
Loader: valid. Accuracy: 0.9732
Epoch: 7
Loader: train. Accuracy: 0.9794166666666667
Loader: valid. Accuracy: 0.9726
Epoch: 8
Loader: train. Accuracy: 0.9812
Loader: valid. Accuracy: 0.9728
Epoch: 9
Loader: train. Accuracy: 0.98415
Loader: valid. Accuracy: 0.9752
Epoch: 10
Loader: train. Accuracy: 0.9861166666666666
Loader: valid. Accuracy: 0.9752
```

Задание. Протестируйте разные функции активации.

Попробуйте разные функции активации. Для каждой функции активации посчитайте массив validation accuracy. Лучше реализовать это в виде функции, берущей на вход активацию и получающей массив из accuracies.

```
elu_accuracy = accuracy["valid"]
```

```
# YOUR CODE. Do the same thing with other activations (it's better to wrap into a function that returns a list of accuracies)

from tqdm import tqdm_notebook

def test_activation_function(activation):
    #YOUR CODE

    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, 128),
        activation(),
        nn.Linear(128,64),
        activation(),
        nn.Linear(64,10)
    )

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters())

    loaders = {"train": train_dataloader, "valid": valid_dataloader}

    device = "cuda" if torch.cuda.is_available() else "cpu"

    max_epochs = 10
    accuracy = {"train": [], "valid": []}
    for epoch in tqdm_notebook(range(max_epochs)):
        for k, dataloader in loaders.items():
            epoch_correct = 0
            epoch_all = 0
            for x_batch, y_batch in dataloader:
                if k == "train":
                     # YOUR CODE. Set model to ``train`` mode and calculate outputs. Don't forget zero_grad!
                    optimizer.zero_grad()
                    outp = model(x_batch)
                else:
                     # YOUR CODE. Set model to ``eval`` mode and calculate outputs
                    model.eval()
                    with torch.no_grad():
                        outp = model(x_batch)
                preds = outp.argmax(-1)
                correct =  (preds == y_batch).sum() # YOUR CODE GOES HERE
                all =  y_batch.size(0) # YOUR CODE GOES HERE
                epoch_correct += correct.item()
                epoch_all += all
                if k == "train":
                    loss = criterion(outp, y_batch)
                    # YOUR CODE. Calculate gradients and make a step of your optimizer
                    loss.backward()
                    optimizer.step()
            accuracy[k].append(epoch_correct/epoch_all)
    return accuracy
```

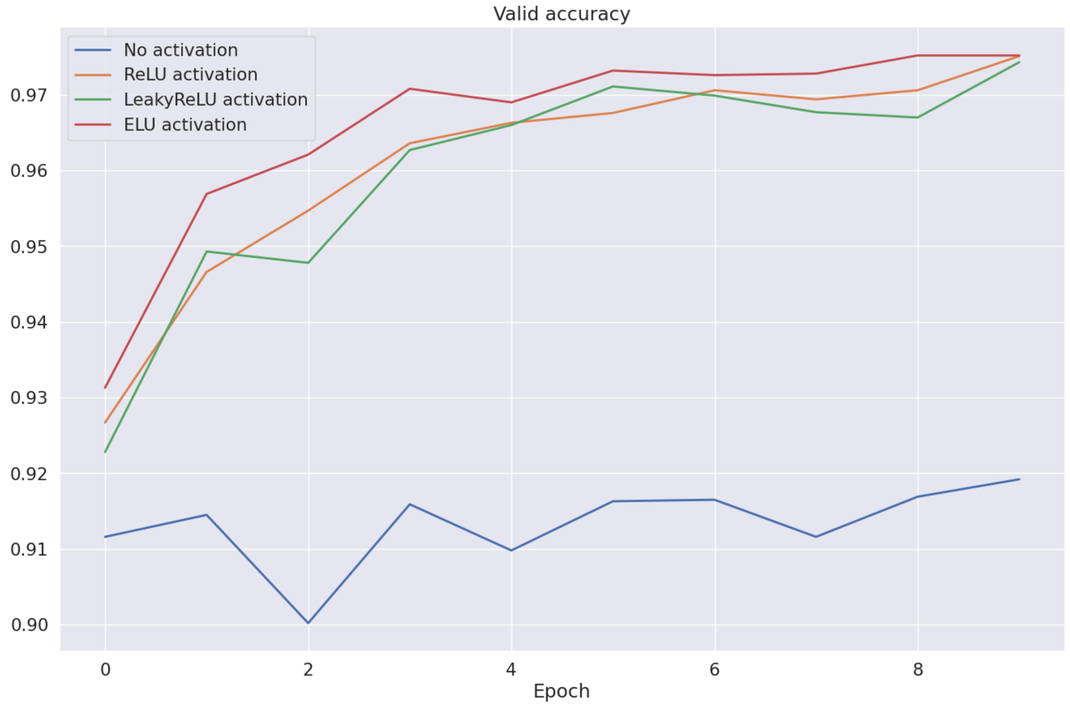
```
plain_accuracy = test_activation_function(Identical)['valid']
relu_accuracy = test_activation_function(nn.ReLU)['valid']
leaky_relu_accuracy = test_activation_function(nn.LeakyReLU)['valid']
```

```
<ipython-input-26-ade5565a123e>:26: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for epoch in tqdm_notebook(range(max_epochs)):
100%                                                10/10 [02:22<00:00, 14.21s/it]

100%                                                10/10 [02:23<00:00, 14.32s/it]

100%                                                10/10 [02:25<00:00, 14.56s/it]
```

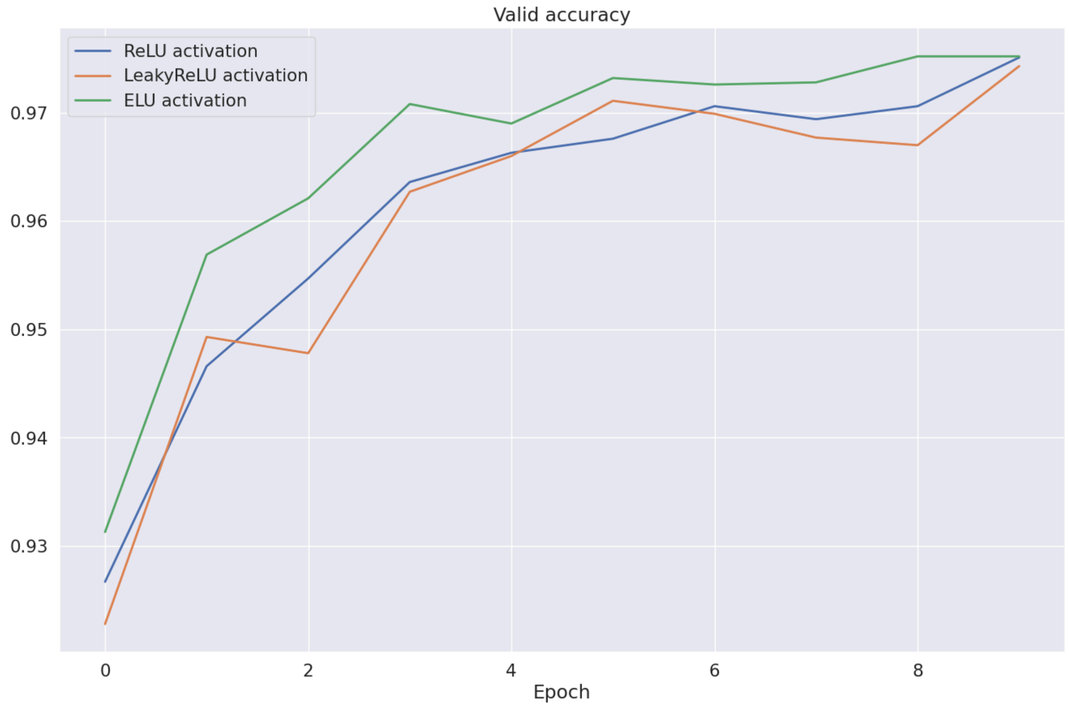
Accuracy

Построим график accuracy/epoch для каждой функции активации.

```
sns.set(style="darkgrid", font_scale=1.4)

plt.figure(figsize=(16, 10))
plt.title("Valid accuracy")
plt.plot(range(max_epochs), plain_accuracy, label="No activation", linewidth=2)
plt.plot(range(max_epochs), relu_accuracy, label="ReLU activation", linewidth=2)
plt.plot(range(max_epochs), leaky_relu_accuracy, label="LeakyReLU activation", linewidth=2)
plt.plot(range(max_epochs), elu_accuracy, label="ELU activation", linewidth=2)
plt.legend()
plt.xlabel("Epoch")
plt.show()
```




```
plt.figure(figsize=(16, 10))
plt.title("Valid accuracy")
plt.plot(range(max_epochs), relu_accuracy, label="ReLU activation", linewidth=2)
plt.plot(range(max_epochs), leaky_relu_accuracy, label="LeakyReLU activation", linewidth=2)
plt.plot(range(max_epochs), elu_accuracy, label="ELU activation", linewidth=2)
plt.legend()
plt.xlabel("Epoch")
plt.show()
```



```
from collections import Counter

dict_of_activation = {'No_activation':plain_accuracy,
                      'ReLU_activation':relu_accuracy,
                      'Leacky_ReLU_activation':leaky_relu_accuracy,
                      'ELU_activation':elu_accuracy}
max_values = dict()
for i in dict_of_activation:
    max_values[i] = max(dict_of_activation[i])

Counter(max_values).most_common()
```

```
[('ELU_activation', 0.9752),
 ('ReLU_activation', 0.9751),
 ('Leacky_ReLU_activation', 0.9743),
 ('No_activation', 0.9192)]
```

```
dict_of_activation['ELU_activation'].index(0.9752)
```

```
8
```

Вопрос 4. Какая из активаций показала наивысший `accuracy` к концу обучения?

Ответ 4: Наивысший "accuracy" показала ELU_activation на 8ой эпохе!

Часть 2.2 Сверточные нейронные сети

Ядра

Сначала немного поработам с самим понятием ядра свёртки.

```
!wget https://img.the-village.kz/the-village.com.kz/post-cover/5x5-I6oiwjmq79dMCZMEbA-default.jpg -O sample_photo.jpg
```

```
--2024-03-24 23:55:27--  https://img.the-village.kz/the-village.com.kz/post-cover/5x5-I6oiwjmq79dMCZMEbA-default.jpg
Resolving img.the-village.kz (img.the-village.kz)... 5.9.226.237
Connecting to img.the-village.kz (img.the-village.kz)|5.9.226.237|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 49337 (48K) [image/jpeg]
Saving to: ‘sample_photo.jpg’

sample_photo.jpg    100%[===================>]  48.18K   160KB/s    in 0.3s

2024-03-24 23:55:28 (160 KB/s) - ‘sample_photo.jpg’ saved [49337/49337]
```

```
import cv2
sns.set(style="white")
img = cv2.imread("sample_photo.jpg")
RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(12, 8))
plt.imshow(RGB_img)
plt.show()
```

Попробуйте посмотреть как различные свертки влияют на фото. Например, попробуйте А)

```
[0, 0, 0],
[0, 1, 0],
[0, 0, 0]
```

Б)

```
[0, 1, 0],
[0, -2, 0],
[0, 1, 0]
```

В)

```
[0, 0, 0],
[1, -2, 1],
[0, 0, 0]
```

Г)

```
[0, 1, 0],
[1, -4, 1],
[0, 1, 0]
```

Д)

```
[0, -1, 0],
[-1, 5, -1],
[0, -1, 0]
```

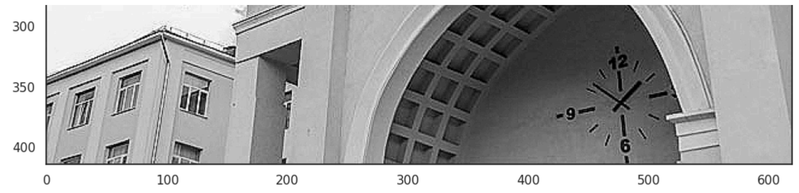
Е)

```
[0.0625, 0.125, 0.0625],
[0.125, 0.25, 0.125],
[0.0625, 0.125, 0.0625]
```

Не стесняйтесь пробовать свои варианты!

```
kernels = [
    torch.tensor([
    [0, 0, 0],
    [0, 1, 0],
    [0, 0, 0]
    ]).reshape(1, 1, 3, 3).type(torch.float32),
    torch.tensor([
    [0, 1, 0],
    [0, -2, 0],
    [0, 1, 0]
    ]).reshape(1, 1, 3, 3).type(torch.float32),
    torch.tensor([
    [0, 0, 0],
    [1, -2, 1],
    [0, 0, 0]
    ]).reshape(1, 1, 3, 3).type(torch.float32),
    torch.tensor([
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]
    ]).reshape(1, 1, 3, 3).type(torch.float32),
    torch.tensor([
    [0, -1, 0],
    [-1, 5, -1],
    [0, -1, 0]
    ]).reshape(1, 1, 3, 3).type(torch.float32),
    torch.tensor([
    [0.0625, 0.125, 0.0625],
    [0.125, 0.25, 0.125],
    [0.0625, 0.125, 0.0625]
    ]).reshape(1, 1, 3, 3).type(torch.float32)
          ]
```

```
for kernel in kernels:
  print(f'Conv is {kernel}')
  img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)

  kernel = kernel.repeat(3, 3, 1, 1)
  img_t = img_t.permute(0, 3, 1, 2)  # [BS, H, W, C] -> [BS, C, H, W]
  img_t = nn.ReflectionPad2d(1)(img_t)  # Pad Image for same output size

  result = F.conv2d(img_t, kernel)[0]  #

  plt.figure(figsize=(12, 8))
  result_np = result.permute(1, 2, 0).numpy() / 256 / 3

  plt.imshow(result_np)
  plt.show()
  print('-'*100)
```



```
Conv is tensor([[[[0.0625, 0.1250, 0.0625],
          [0.1250, 0.2500, 0.1250],
          [0.0625, 0.1250, 0.0625]]]])
```