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.

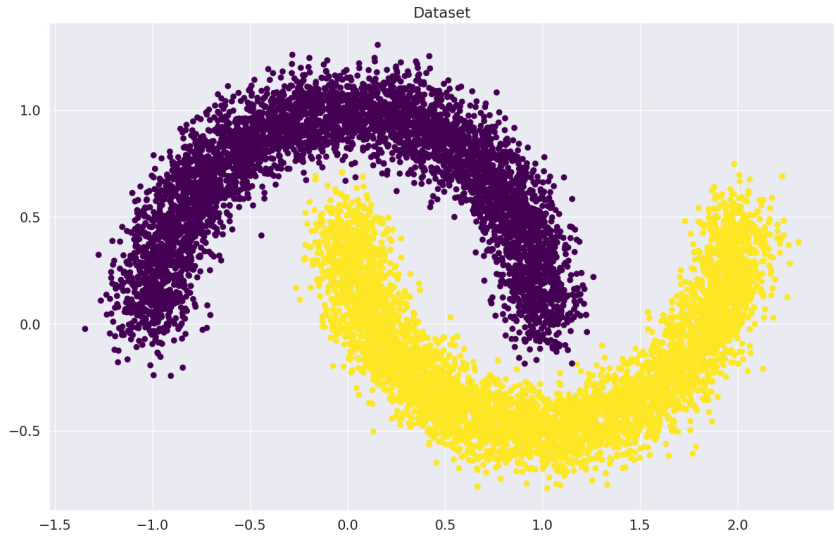
```
import numpy as np

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.tensor(X_train, dtype=torch.float32)
y_train_t = torch.tensor(y_train, dtype=torch.float32)
X_val_t = torch.tensor(X_val, dtype=torch.float32)
y_val_t = torch.tensor(y_val, dtype=torch.float32)
```

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(in_features, out_features))
        self.bias = bias
        if bias:
            self.bias_term = nn.Parameter(torch.randn(out_features))

    def forward(self, x):
        x = x.matmul(self.weights)
        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.

```
def count_trainable_parameters(model):
    total_parameters = 0
    for parameter in model.parameters():
        total_parameters += torch.prod(torch.tensor(parameter.shape)).item()
    return total_parameters

trainable_parameters = count_trainable_parameters(linear_regression)
print("Total number of trainable parameters:", trainable_parameters)
```

```
Total number of trainable parameters: 3
```

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.squeeze(), y_batch.type(torch.float32))
        loss.backward()
        losses.append(loss.detach().flatten()[0])
        optimizer.step()
        probabilities = torch.sigmoid(outp).squeeze()
        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
```

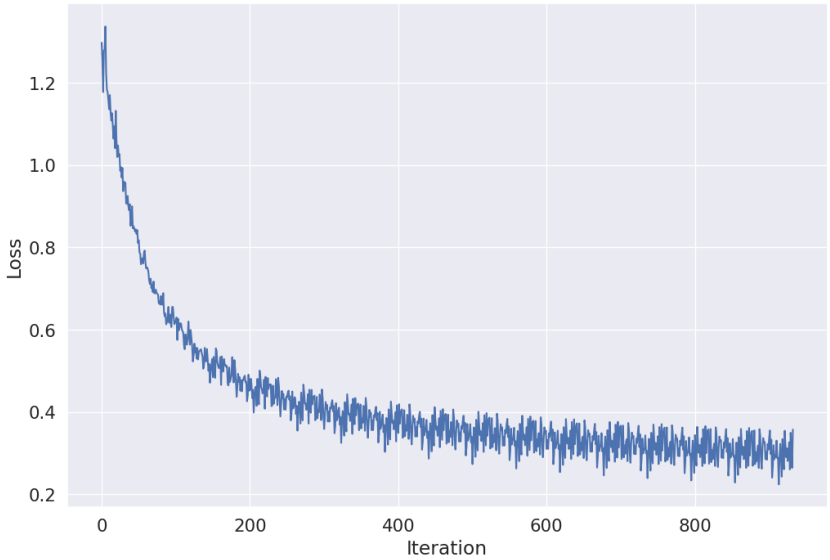
```
Iteration: 0
Batch accuracy: 0.15625
Iteration: 100
Batch accuracy: 0.578125
Iteration: 200
Batch accuracy: 0.7578125
Iteration: 300
Batch accuracy: 0.8203125
Iteration: 400
Batch accuracy: 0.8046875
Iteration: 500
Batch accuracy: 0.8671875
Iteration: 600
Batch accuracy: 0.8515625
Iteration: 700
Batch accuracy: 0.796875
Iteration: 800
Batch accuracy: 0.8125
Iteration: 900
Batch accuracy: 0.8125

Iteration: 932. Convergence. Stopping iterations.
```

Вопрос 2. Сколько итераций потребовалось, чтобы алгоритм сошелся?

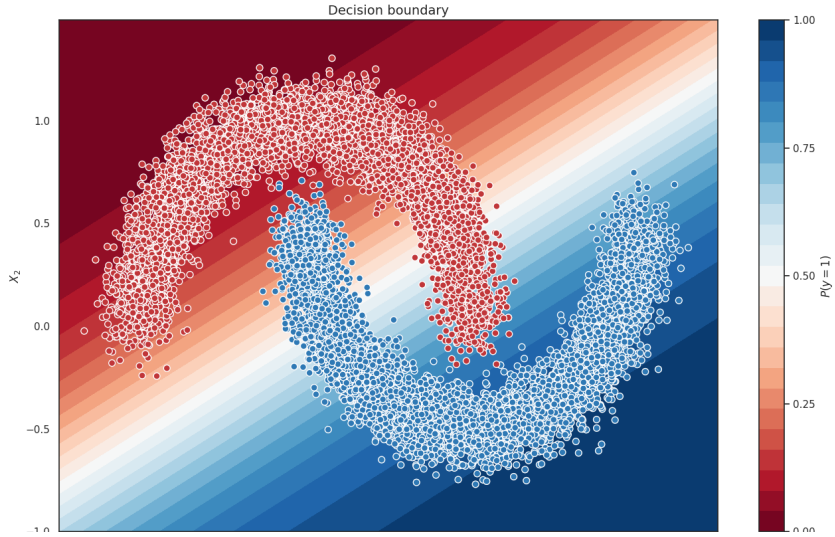
Ответ: 932

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)
        probabilities = torch.sigmoid(outp).squeeze()
        preds = (probabilities > 0.5).type(torch.long)
        predictions = np.hstack((predictions, preds.numpy().flatten()))
    return predictions.flatten()
```

```
from sklearn.metrics import accuracy_score

val_dataset = TensorDataset(X_val_t, y_val_t)
val_dataloader = DataLoader(val_dataset, batch_size=64, shuffle=False)
predictions = predict(val_dataloader, linear_regression)
accuracy = accuracy_score(y_val, predictions)
print("Total accuracy:", accuracy)
```

```
Total accuracy: 0.86
```

Вопрос 3

Какое `accuracy` получается после обучения?

Ответ: 0.86

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
import torchvision.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=64, shuffle=True)
valid_dataloader = DataLoader(val_dataset, batch_size=64, shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./MNIST/raw/train-images-idx3-ubyte.gz
100.0%
Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
100.0%Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./MNIST/raw/train-labels-idx1-ubyte.gz
Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz

2.0%Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./MNIST/raw/t10k-images-idx3-ubyte.gz
100.0%
Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./MNIST/raw/t10k-labels-idx1-ubyte.gz
100.0%Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw
```

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).

```
activation = nn.ELU

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(28 * 28, 128),
    nn.ELU(),
    nn.Linear(128, 128),
    nn.ELU(),
    nn.Linear(128, 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"
```

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":
                model.train()
                optimizer.zero_grad()
                outp = model(x_batch)
            else:
                model.eval()
                with torch.no_grad():
                    outp = model(x_batch)
            preds = outp.argmax(-1)
            correct = (preds == y_batch).sum()
            all = y_batch.size(0)
            epoch_correct += correct.item()
            epoch_all += all
            if k == "train":
                loss = criterion(outp, y_batch)
                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: 0.9012333333333333
Loader: valid. Accuracy: 0.9385
Epoch: 2
Loader: train. Accuracy: 0.9548
Loader: valid. Accuracy: 0.9577
Epoch: 3
Loader: train. Accuracy: 0.9671333333333333
Loader: valid. Accuracy: 0.9662
Epoch: 4
Loader: train. Accuracy: 0.9732166666666666
Loader: valid. Accuracy: 0.9652
Epoch: 5
Loader: train. Accuracy: 0.9768833333333333
Loader: valid. Accuracy: 0.9669
Epoch: 6
Loader: train. Accuracy: 0.9798333333333333
Loader: valid. Accuracy: 0.9724
Epoch: 7
```

```
Loader: train. Accuracy: 0.9815666666666667
Loader: valid. Accuracy: 0.9726
Epoch: 8
Loader: train. Accuracy: 0.9836333333333334
Loader: valid. Accuracy: 0.973
Epoch: 9
Loader: train. Accuracy: 0.9859166666666667
Loader: valid. Accuracy: 0.9775
Epoch: 10
Loader: train. Accuracy: 0.9867
Loader: valid. Accuracy: 0.9646
```

Задание. Протестируйте разные функции активации.

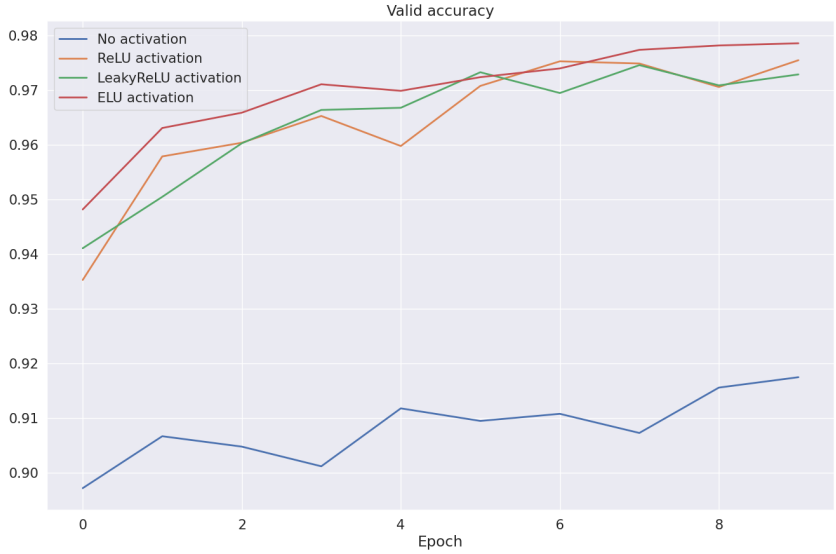
Попробуйте разные функции активации. Для каждой функции активации посчитайте массив validation accuracy. Лучше реализовать это в виде функции, берущей на вход активацию и получающей массив из accuracies.

```
elu_accuracy = accuracy["valid"]


def test_activation_function(activation):
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(28 * 28, 128),
        activation(),
        nn.Linear(128, 128),
        activation(),
        nn.Linear(128, 10)
    )

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

    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":
                    model.train()
                    optimizer.zero_grad()
                    outp = model(x_batch)
                else:
                    model.eval()
                    with torch.no_grad():
                        outp = model(x_batch)
                preds = outp.argmax(-1)
                correct = (preds == y_batch).sum()
                all = y_batch.size(0)
                epoch_correct += correct.item()
                epoch_all += all
                if k == "train":
                    loss = criterion(outp, y_batch)
                    loss.backward()
                    optimizer.step()
            accuracy[k].append(epoch_correct/epoch_all)
    return accuracy["valid"]

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


plain_accuracy = test_activation_function(Identical)
relu_accuracy = test_activation_function(nn.ReLU)
leaky_relu_accuracy = test_activation_function(nn.LeakyReLU)
```

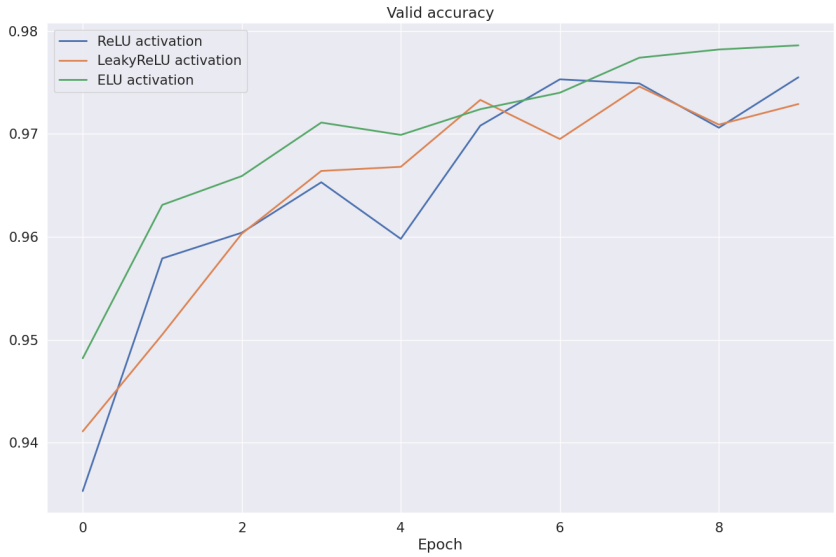
Accuracy

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()
```



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

Ответ: ELU activation

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
```

```
--2023-04-02 11:47:08--  https://img.the-village.kz/the-village.com.kz/post-cover/5x5-I6oiwjmq79dMCZMEbA-default.jpg
Resolving img.the-village.kz (img.the-village.kz)... 144.76.208.75
Connecting to img.the-village.kz (img.the-village.kz)|144.76.208.75|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://img.the-village-kz.com/the-village.com.kz/post-cover/5x5-I6oiwjmq79dMCZMEbA-default.jpg [following]
--2023-04-02 11:47:09--  https://img.the-village-kz.com/the-village.com.kz/post-cover/5x5-I6oiwjmq79dMCZMEbA-default.jpg
Resolving img.the-village-kz.com (img.the-village-kz.com)... 104.21.1.92, 172.67.128.246, 2606:4700:3036::ac43:80f6, ...
Connecting to img.the-village-kz.com (img.the-village-kz.com)|104.21.1.92|: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  --.-KB/s    in 0.07s

2023-04-02 11:47:10 (678 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]
```

E)

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

Не стесняйтесь пробовать свои варианты!

```
img_t = torch.from_numpy(RGB_img).type(torch.float32).unsqueeze(0)
kernel = 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)

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()
```

Вопрос 5. Как можно описать действия ядер, приведенных выше? Сопоставьте для каждой буквы число.

1) Размытие

2) Увеличение резкости

3) Тождественное преобразование

4) Выделение вертикальных границ

5) Выделение горизонтальных границ

6) Выделение границ

Ответ: А-3, Б-4, В-3, Г-5, Д-6, Е-1

Задание. Реализуйте LeNet

Если мы сделаем параметры сверток обучаемыми, то можем добиться хороших результатов для задач компьютерного зрения. Реализуйте архитектуру LeNet, предложенную еще в 1998 году! На этот раз используйте модульную структуру (без помощи класса Sequential).

Наша нейронная сеть будет состоять из

- Свёртки 3x3 (1 карта на входе, 6 на выходе) с активацией ReLU;
- MaxPooling-а 2x2;
- Свёртки 3x3 (6 карт на входе, 16 на выходе) с активацией ReLU;
- MaxPooling-а 2x2;
- Уплощения (nn.Flatten);
- Полносвязного слоя со 120 нейронами и активацией ReLU;
- Полносвязного слоя с 84 нейронами и активацией ReLU;
- Выходного слоя из 10 нейронов.

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square conv kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 3)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = nn.Flatten()(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


model = LeNet().to(device)

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

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

Задание. Обучите CNN

Используйте код обучения, который вы написали для полносвязной нейронной сети.

```
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":
                model.train()
                optimizer.zero_grad()
                outp = model(x_batch)
            else:
                model.eval()
```

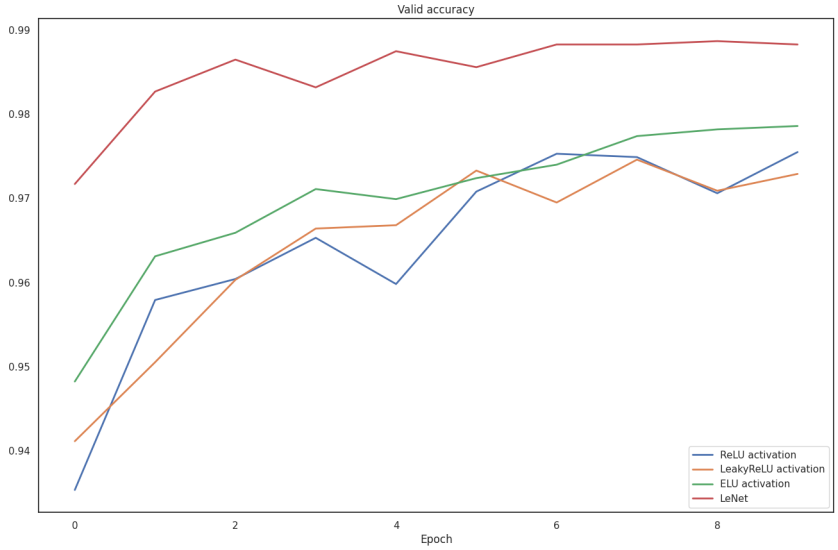
```
                with torch.no_grad():
                    outp = model(x_batch)
            preds = outp.argmax(-1)
            correct = (preds == y_batch).sum()
            all = y_batch.size(0)
            epoch_correct += correct.item()
            epoch_all += all
            if k == "train":
                loss = criterion(outp, y_batch)
                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: 0.8991666666666667
Loader: valid. Accuracy: 0.9717
Epoch: 2
Loader: train. Accuracy: 0.9741
Loader: valid. Accuracy: 0.9827
Epoch: 3
Loader: train. Accuracy: 0.98215
Loader: valid. Accuracy: 0.9865
Epoch: 4
Loader: train. Accuracy: 0.9863
Loader: valid. Accuracy: 0.9832
Epoch: 5
Loader: train. Accuracy: 0.9884666666666667
Loader: valid. Accuracy: 0.9875
Epoch: 6
Loader: train. Accuracy: 0.9899833333333333
Loader: valid. Accuracy: 0.9856
Epoch: 7
Loader: train. Accuracy: 0.9916666666666667
Loader: valid. Accuracy: 0.9883
Epoch: 8
Loader: train. Accuracy: 0.9932
Loader: valid. Accuracy: 0.9883
Epoch: 9
Loader: train. Accuracy: 0.9932666666666666
Loader: valid. Accuracy: 0.9887
Epoch: 10
Loader: train. Accuracy: 0.9943333333333333
Loader: valid. Accuracy: 0.9883
```

```
lenet_accuracy = accuracy["valid"]
```

Сравним с предыдущем пунктом

```
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.plot(range(max_epochs), lenet_accuracy, label="LeNet", linewidth=2)
plt.legend()
plt.xlabel("Epoch")
plt.show()
```



Вопрос 6 Какое `accuracy` получается после обучения с точностью до двух знаков после запятой?

Ответ: 0.99