Домашнее задание: основы DL

Задание 1

Вспомним 1ое зянятие - мы узнали, что в DL мы можем подбирать оптимальные параметры для любой дифференцируемой модели, считая градиенты по обучаемым параметрам.

В этом задании вам предстоит руками сделать backpropagation для совсем простой модели, чтобы понять, что в torch не происходит никакой магии

Пусть:

$x = [1, 1]^T$

$y = [1, -1]^T$

$z = [-1, 2]^T$

$l = sum(max(0, x * y)) + prod(x + z^2)$

где $sum, prod$ - поэлементные сложения и умножения соответственно

В ответ вам нужно указать $\frac{\partial l}{\partial x}$ и $\frac{\partial l}{\partial z}$, а в ноутбуке отобразить, как вы аналитически получили это значение

Для проверки, что вы правильно поняли идею backpropagation, можете подсчитать значение $\frac{\partial l}{\partial y}$, оно должно проходить assert

In [392…

```
dl_dy = [1, 0]
assert np.all(dl_dy == [1, 0]), "Пока что неверное, попробуйте еще раз и в
```

```
dl_dx = [2, 4]
```

```
dl_dz = [-4, 16]
```

In [393…

```
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)
```

Задание 2

На занятиях мы часто говорили про линейные модели, что это просто. Давайте реализуем ее сами и попробуем обучать такую модель для задачи классификации (то есть реализуем логистическую регрессию)

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 2.0

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`).

In [394…

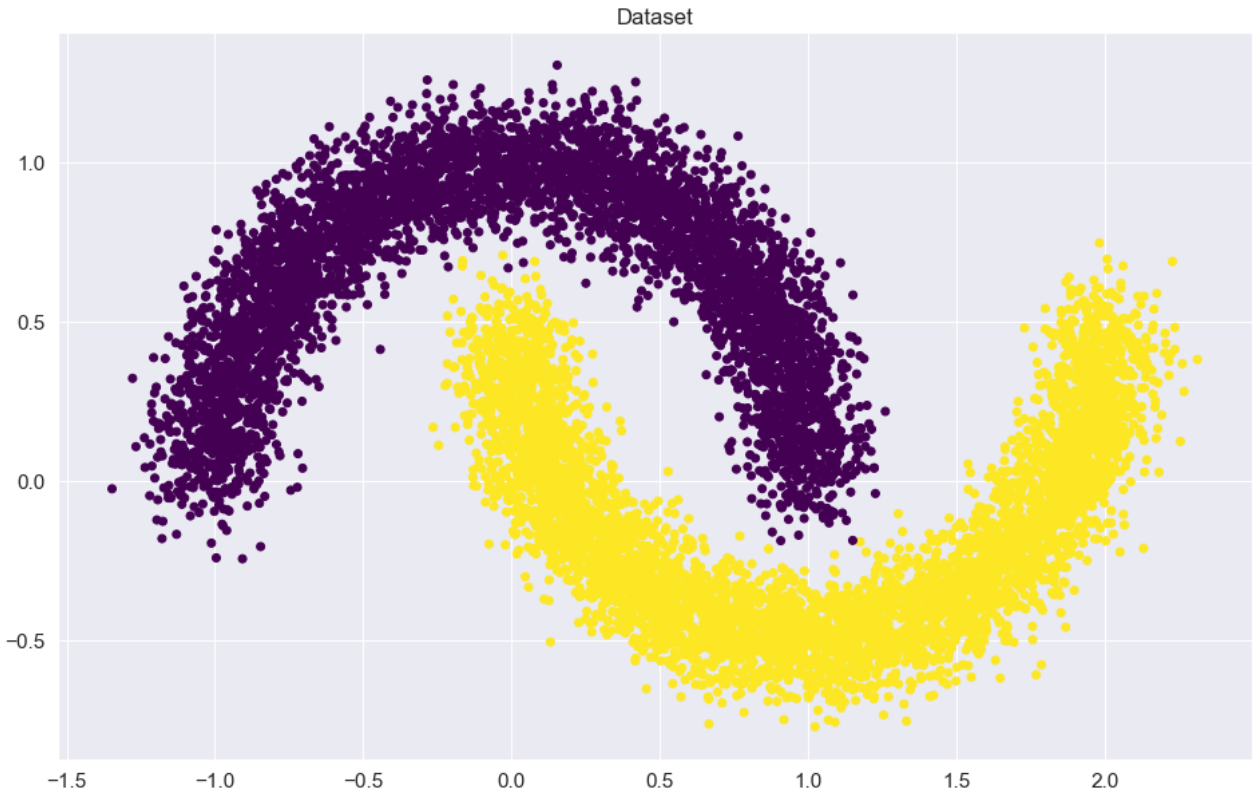
```
class LinearRegression(nn.Module):
    def __init__(self, in_features: int, out_features: int, bias: bool = Tr
        super().__init__()
        self.weights = nn.Parameter(torch.randn(in_features, out_features))
        self.bias = bias
        if bias:
            self.bias_term = nn.Parameter(torch.randn(out_features))

    def forward(self, x):
        res = torch.matmul(x, self.weights)
        if self.bias:
            res += self.bias_term
        return res
```

Датасет

Давайте сгенерируем датасет и посмотрим на него!

In [395…
```
X, y = make_moons(n_samples=10000, random_state=42, noise=0.1)
```

In [396…
```
plt.figure(figsize=(16, 10))
plt.title("Dataset")
plt.scatter(X[:, 0], X[:, 1], c=y, cmap="viridis")
plt.show()
```



Сделаем train/test split

In [397…
```
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=42)
```

In [398…
```
X_train.shape
```

Out[398… `(7500, 2)`

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`.

In [399…
```
X_train_t =  torch.from_numpy(X_train.astype(np.float32))
y_train_t =  torch.from_numpy(y_train.astype(np.float32))
X_val_t =  torch.from_numpy(X_val.astype(np.float32))
y_val_t =  torch.from_numpy(y_val.astype(np.float32))
```

Создаем `Dataset` и `DataLoader`.

In [400…
```
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)
```

In [401…
```
# catalyst.utils.set_global_seed(42)
linear_regression = LinearRegression(2, 1, True)
loss_function = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(linear_regression.parameters(), lr=0.05)
```

In [402…
```
total_params = sum(p.numel() for p in linear_regression.parameters())
print(total_params)
```

```
3
```

Задание 2.1 Сколько обучаемых параметров у получившейся модели?

Ответ: 3

Задание 2.2

Теперь обучим эту модель

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 размера
        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` .

In [403…

```
tol = 1e-3
losses = []
max_epochs = 100
prev_weights = torch.zeros_like(linear_regression.weights)
prev_bias = torch.zeros_like(linear_regression.bias_term)
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(1), y_batch)
        loss.backward()
        losses.append(loss)
        optimizer.step()
        sigm = nn.Sigmoid()
        probabilities =  sigm(outp)
        preds = (probabilities>0.5).type(torch.long)
        batch_acc = (preds.flatten() == y_batch).type(torch.float32).sum()/
        if it % 500000 == 0:
            print(f"Iteration: {it + epoch*len(train_dataset)}\nBatch accu
        current_weights = linear_regression.weights.detach().clone()
        current_bias = linear_regression.bias_term.detach().clone()
        if (prev_weights - current_weights).abs().max() < tol and (prev_bia
            print(f"\nIteration: {it + epoch*len(train_dataset)}.Convergenc
            stop_it = True
            break
        prev_weights = current_weights
        prev_bias = current_bias
    if stop_it:
        break
```

```
Iteration: 0
Batch accuracy: 0.2734375
Iteration: 7500
Batch accuracy: 0.828125
Iteration: 15000
Batch accuracy: 0.8203125
Iteration: 22500
Batch accuracy: 0.8125
Iteration: 30000
Batch accuracy: 0.8125
Iteration: 37500
Batch accuracy: 0.8125
Iteration: 45000
Batch accuracy: 0.8125
Iteration: 52500
Batch accuracy: 0.8203125
Iteration: 60000
Batch accuracy: 0.8203125
Iteration: 67500
Batch accuracy: 0.8203125
Iteration: 75000
Batch accuracy: 0.8203125
Iteration: 82500
Batch accuracy: 0.8203125
Iteration: 90000
Batch accuracy: 0.8203125
Iteration: 97500
Batch accuracy: 0.828125
Iteration: 105000
Batch accuracy: 0.8359375

Iteration: 105054.Convergence. Stopping iterations.
```

Задание 2.2

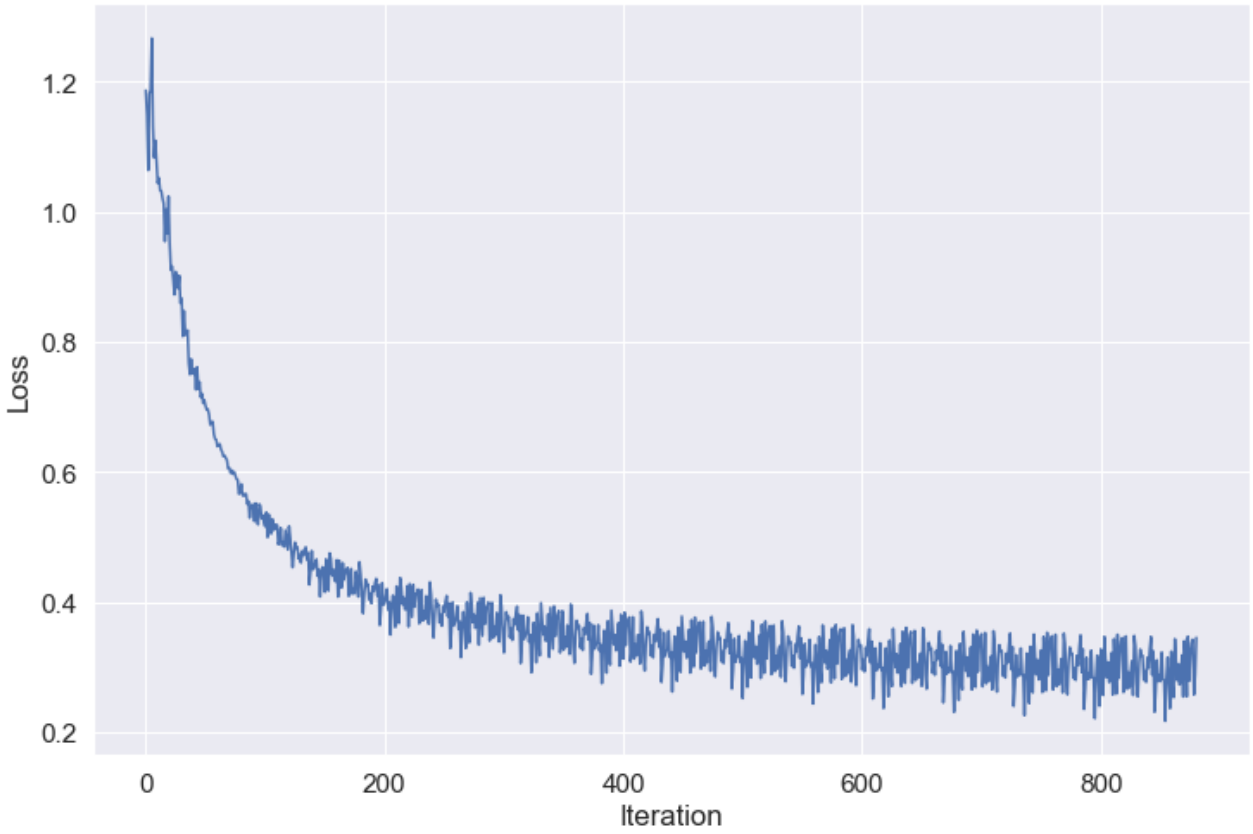
Сколько итераций потребовалось, чтобы алгоритм сошелся?

Ответ: 105054

Визуализируем результаты

In [404…

```
plt.figure(figsize=(12, 8))
losses = [loss.detach().numpy() for loss in losses]
plt.plot(range(len(losses)), losses)
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.show()
```



In [405…

```
@torch.no_grad()
def predict(dataloader, model):
    model.eval()
    predictions = np.array([])
    for x_batch, _ in dataloader:
        outp = model(x_batch)
        probs = torch.sigmoid(outp)
        preds = (probs > 0.5).type(torch.long)
        predictions = np.hstack((predictions, preds.numpy().flatten()))
    predictions = predictions
    return predictions.flatten()
```

In [406…

```
from sklearn.metrics import accuracy_score

out = predict(val_dataloader, linear_regression)
print(accuracy_score(out, y_val_t))
```

```
0.8696
```

Задание 2.3

Какое `accuracy` получается после обучения?

Ответ: 0.8696

Задание 3

Теперь перейдем к датасету MNIST!

На 2ом и 3ем семинаре мы работали с этим датасетом, поэтому ваша задача взять код от туда и исследовать такой интересный вопрос: какая функция активации лучше всего подходит под эту задачу?

Вам необходимо обучить 4 раза модель ниже и сравнить качество для различных функций активаций (или их отсутствия)

In [423…

```
import torch.nn.functional as F
import torchvision
from tqdm.auto import tqdm
from torch import nn
```

In [424…

```
transform = torchvision.transforms.Compose(
    [
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize((0.1307,), (0.3081,)),
    ]
)

mnist_train = torchvision.datasets.MNIST(
    "./mnist/", train=True, download=True, transform=transform
)
mnist_val = torchvision.datasets.MNIST(
    "./mnist/", train=False, download=True, transform=transform
)


train_dataloader = torch.utils.data.DataLoader(mnist_train, batch_size=64,
val_dataloader = torch.utils.data.DataLoader(mnist_val, batch_size=64, shu
```

In [425…

```
massive = [[],[],[],[],[],[],[],[],[],[]]
```

In [426…

```
def train(model, optimizer, n_epochs=10):
    for epoch in range(1,n_epochs+1):
        # тренировка
        for x_train, y_train in tqdm(train_dataloader):
            y_pred = model(x_train)
            loss = F.cross_entropy(y_pred, y_train)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        # валидация
        val_loss = []
        val_accuracy = []
        with torch.no_grad():
            for x_val, y_val in tqdm(val_dataloader):
                y_pred = model(x_val)
                loss = F.cross_entropy(y_pred, y_val)
                val_loss.append(loss.numpy())
                val_accuracy.extend(
                    (torch.argmax(y_pred, dim=-1) == y_val).numpy().tolist
                )

        # печатаем метрики
        print(
            f"Epoch: {epoch}, loss: {np.mean(val_loss)}, accuracy: {np.mea
        )
        massive[epoch-1].append(np.mean(val_accuracy))
```

In [427…

```
# activations = [nn.ReLU, nn.LeakyReLU, nn.ELU]
# for i in range(len(activations)):
#     print(f"{activation}")
#     activation = activations[i]
#     model = nn.Sequential(
#         nn.Flatten(),
#         nn.Linear(28*28, 128),
#         activation(),
#         nn.Linear(128, 128),
#         activation(),
#         nn.Linear(128, 10))
#     optimizer = torch.optim.SGD(model.parameters(), lr = 0.05)
#     print(f"Training model with {sum([x[1].numel() for x in model.named_
#     train(model, optimizer)
```

In [428…

```
activations = [nn.ReLU]
for i in range(len(activations)):
    print("ReLU")
    activation = activations[i]
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(28*28, 128),
        activation(),
        nn.Linear(128, 128),
        activation(),
        nn.Linear(128, 10))
    optimizer = torch.optim.SGD(model.parameters(), lr = 0.05)
    print(f"Training model with {sum([x[1].numel() for x in model.named_pa
    train(model, optimizer)
```

```
ReLU
Training model with 118282 parameters

Epoch: 1, loss: 0.2075415700674057, accuracy: 0.9348

Epoch: 2, loss: 0.13317056000232697, accuracy: 0.9593

Epoch: 3, loss: 0.10534357279539108, accuracy: 0.9655

Epoch: 4, loss: 0.09274303168058395, accuracy: 0.9686

Epoch: 5, loss: 0.089905746281147, accuracy: 0.9716

Epoch: 6, loss: 0.07960105687379837, accuracy: 0.9729

Epoch: 7, loss: 0.07332146912813187, accuracy: 0.9758

Epoch: 8, loss: 0.06781353056430817, accuracy: 0.9778

Epoch: 9, loss: 0.07110472768545151, accuracy: 0.9777

Epoch: 10, loss: 0.07153356075286865, accuracy: 0.9774
```

In [429…

```
print(massive)
```

```
[[0.9348], [0.9593], [0.9655], [0.9686], [0.9716], [0.9729], [0.9758], [0.9
778], [0.9777], [0.9774]]
```

In [430…

```
activations = [nn.LeakyReLU]
for i in range(len(activations)):
    print("LeakyReLU")
    activation = activations[i]
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(28*28, 128),
        activation(),
        nn.Linear(128, 128),
        activation(),
        nn.Linear(128, 10))
    optimizer = torch.optim.SGD(model.parameters(), lr = 0.05)
    print(f"Training model with {sum([x[1].numel() for x in model.named_pa
    train(model, optimizer)
```

```
LeakyReLU
Training model with 118282 parameters

Epoch: 1, loss: 0.1835421621799469, accuracy: 0.945

Epoch: 2, loss: 0.13064631819725037, accuracy: 0.9595

Epoch: 3, loss: 0.10273808240890503, accuracy: 0.967

Epoch: 4, loss: 0.09327074885368347, accuracy: 0.9714

Epoch: 5, loss: 0.09373459964990616, accuracy: 0.972

Epoch: 6, loss: 0.08878692239522934, accuracy: 0.9717

Epoch: 7, loss: 0.07721471786499023, accuracy: 0.9764

Epoch: 8, loss: 0.07701306790113449, accuracy: 0.9772

Epoch: 9, loss: 0.06934937089681625, accuracy: 0.9792

Epoch: 10, loss: 0.06627160310745239, accuracy: 0.9789
```

In [431…

```
activations = [nn.ELU]
for i in range(len(activations)):
    print("ELU")
    activation = activations[i]
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(28*28, 128),
        activation(),
        nn.Linear(128, 128),
        activation(),
        nn.Linear(128, 10))
    optimizer = torch.optim.SGD(model.parameters(), lr = 0.05)
    print(f"Training model with {sum([x[1].numel() for x in model.named_pa
    train(model, optimizer)
```

```
ELU
Training model with 118282 parameters

Epoch: 1, loss: 0.22106294333934784, accuracy: 0.9369

Epoch: 2, loss: 0.14428435266017914, accuracy: 0.958

Epoch: 3, loss: 0.10980455577373505, accuracy: 0.966

Epoch: 4, loss: 0.10095846652984619, accuracy: 0.9683

Epoch: 5, loss: 0.08892668783664703, accuracy: 0.972

Epoch: 6, loss: 0.07777231186628342, accuracy: 0.9733

Epoch: 7, loss: 0.08269988745450974, accuracy: 0.9727

Epoch: 8, loss: 0.07225136458873749, accuracy: 0.9775

Epoch: 9, loss: 0.07366837561130524, accuracy: 0.9766

Epoch: 10, loss: 0.07101350277662277, accuracy: 0.9777
```

In [432…

```
activations = [None]
for i in range(len(activations)):
    print("None")
    activation = activations[i]
    model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(28*28, 128),
#         activation(),
        nn.Linear(128, 128),
#         activation(),
        nn.Linear(128, 10))
    optimizer = torch.optim.SGD(model.parameters(), lr = 0.05)
    print(f"Training model with {sum([x[1].numel() for x in model.named_pa
    train(model, optimizer)
```

```
None
Training model with 118282 parameters

Epoch: 1, loss: 0.30732864141464233, accuracy: 0.9097

Epoch: 2, loss: 0.29346221685409546, accuracy: 0.9155

Epoch: 3, loss: 0.3049066364765167, accuracy: 0.9102

Epoch: 4, loss: 0.2917293310165405, accuracy: 0.916

Epoch: 5, loss: 0.3359886705875397, accuracy: 0.9001

Epoch: 6, loss: 0.30344608426094055, accuracy: 0.9132

Epoch: 7, loss: 0.2918868064880371, accuracy: 0.9165

Epoch: 8, loss: 0.3113236427307129, accuracy: 0.9124

Epoch: 9, loss: 0.28542596101760864, accuracy: 0.9195

Epoch: 10, loss: 0.28569263219833374, accuracy: 0.9205
```

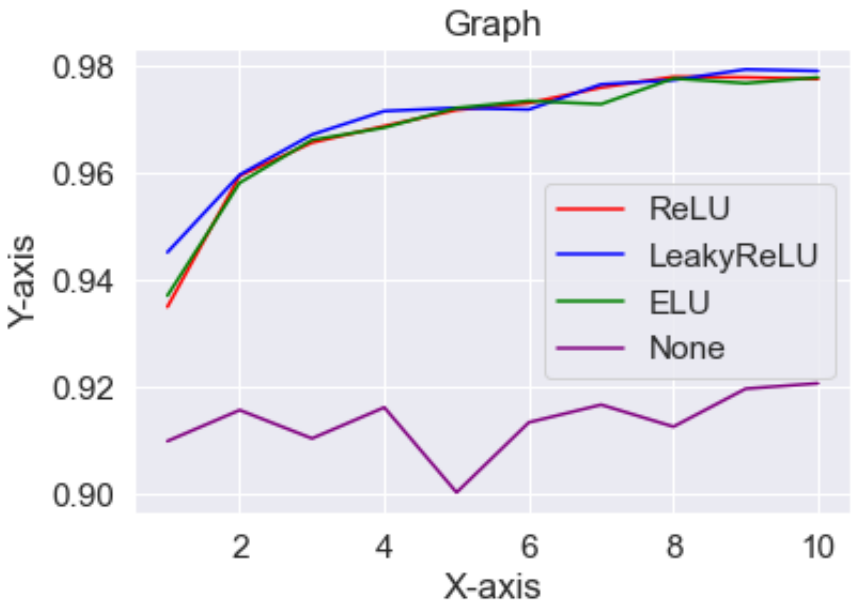
In [433…

```
print(massive)
```

```
[[0.9348, 0.945, 0.9369, 0.9097], [0.9593, 0.9595, 0.958, 0.9155], [0.9655,
0.967, 0.966, 0.9102], [0.9686, 0.9714, 0.9683, 0.916], [0.9716, 0.972, 0.9
72, 0.9001], [0.9729, 0.9717, 0.9733, 0.9132], [0.9758, 0.9764, 0.9727, 0.9
165], [0.9778, 0.9772, 0.9775, 0.9124], [0.9777, 0.9792, 0.9766, 0.9195], [
0.9774, 0.9789, 0.9777, 0.9205]]
```

In [435…

```
colors = ['red', 'blue', 'green', 'purple']

transposed_data = list(zip(*massive))

fig, ax = plt.subplots()

for i, line_data in enumerate(transposed_data):
    ax.plot(range(1, len(line_data)+1), line_data, color=colors[i], label=

ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Graph')

ax.legend(["ReLU", "LeakyReLU", "ELU", "None"])

plt.show()
```



Необходимо построить график: valid accuracy от номера эпохи (максимум 10 эпох) для разных функций активации и выбрать лучшую из них

Вопрос 3 Какая из активаций показала наивысший `accuracy` ?

Ответ: LeakyReLU

Задание 4

Теперь обучим архитектуру, которая использует операции nn.Conv2d. На семинарах мы наблюдали, что можем сильно увеличить качество решения.

Давайте посмотрим на архитектуру, предложенную еще в 1998 году - LeNet!

In [21]:
```
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.conv2 = nn.Conv2d(6, 16, 3)
        self.fc1 = nn.Linear(get_flat_features(), 120)  # YOUR CODE HERE
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))
        x = ... # YOUR CODE HERE
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In [22]:
```
model = LeNet().to(device)

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

loaders = {"train": trainloader, "valid": testloader}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-cf459d3ab72f> in <module>
----> 1 model = LeNet().to(device)
      2
      3 criterion = nn.CrossEntropyLoss()
      4 optimizer = torch.optim.Adam(model.parameters())
      5

<ipython-input-21-c4a82c10e464> in __init__(self)
      5         self.conv1 = nn.Conv2d(1, 6, 3)
      6         self.conv2 = nn.Conv2d(6, 16, 3)
----> 7         self.fc1 = nn.Linear(..., 120)  # YOUR CODE HERE
      8         self.fc2 = nn.Linear(120, 84)
      9         self.fc3 = nn.Linear(84, 10)

/opt/anaconda3/lib/python3.8/site-packages/torch/nn/modules/linear.py in __
init__(self, in_features, out_features, bias, device, dtype)
     94         self.in_features = in_features
     95         self.out_features = out_features
---> 96         self.weight = Parameter(torch.empty((out_features,
in_features), **factory_kwargs))
     97         if bias:
     98             self.bias = Parameter(torch.empty(out_features, **facto
ry_kwargs))

TypeError: empty(): argument 'size' must be tuple of ints, but found elemen
t of type ellipsis at pos 2
```

Необходимо обучить модель и сравнить дают ли сверточные слои прирост к качеству? Для этого ответить на вопрос

Задание 4

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

Ответ: необходимо указать в форме

In [ ]:

In [ ]: