Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Домашнее Задание №1
по дисциплине
«Методы машинного обучения»
на тему

«Распознавание рукописных цифр MNIST с помощью PyTorch»

Выполнил:
студент группы ИУ5-23М
Ся Тунтун

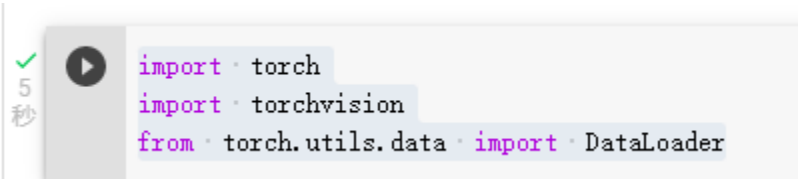
Москва — 2021 г.

1. Задаче обучения

В этой статье мы построим простую сверточную нейронную сеть в PyTorch и обучим ее распознаванию рукописных цифр на основе набора данных MNIST. 70 000 изображений рукописных цифр включены в MNIST: 60 000 для обучения и 10 000 для тестирования. Изображения имеют серый масштаб, 28x28 пикселей, и центрированы для уменьшения предварительной обработки и ускорения работы.

2. Настройка среды

Сначала перейдите на официальный сайт и, следуя руководству, установите среду PyTorch на свой ПК, а затем ознакомьтесь с библиотекой.



3.Подготовка набора данных

Когда импорт готов, мы можем приступить к подготовке данных, которые мы будем использовать. Но перед этим мы определим гиперпараметры, которые будем использовать для экспериментов. Здесь число эпох определяет количество циклов, в течение которых мы будем проходить весь набор данных для обучения, а скорость обучения и импульс - это гиперпараметры оптимизатора, который мы будем использовать позже.

```
[2]  n_epochs = 3
     batch_size_train = 64
     batch_size_test = 1000
     learning_rate = 0.01
     momentum = 0.5
     log_interval = 10
     random_seed = 1
     torch.manual_seed(random_seed)

     <torch._C.Generator at 0x7f0551d95730>
```

Теперь нам также нужен загрузчик данных для набора данных, и здесь в игру вступает TorchVision. Это позволяет нам загружать набор данных MNIST удобным способом. Мы будем использовать batch_size=64 для обучения и size=1000 для тестирования на этом наборе данных. Значения 0,1307 и 0,3081, используемые в преобразовании Normalize() ниже, являются глобальным средним и стандартным отклонением набора данных MNIST, и здесь мы принимаем их как данность.

TorchVision предлагает ряд удобных преобразований, таких как обрезка или нормализация.

```
train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('./data/', train=True, download=True,
                               transform=torchvision.transforms.Compose([
                                   torchvision.transforms.ToTensor(),
                                   torchvision.transforms.Normalize(
                                       (0.1307,), (0.3081,))
                               ])),
    batch_size=batch_size_train, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('./data/', train=False, download=True,
                               transform=torchvision.transforms.Compose([
                                   torchvision.transforms.ToTensor(),
                                   torchvision.transforms.Normalize(
                                       (0.1307,), (0.3081,))
                               ])),
    batch_size=batch_size_test, shuffle=True)
```

```
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 ./data/MNIST/raw/train-images-idx3-ubyte.gz
9913344/? [00:00<00:00, 16573223.05it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw
```

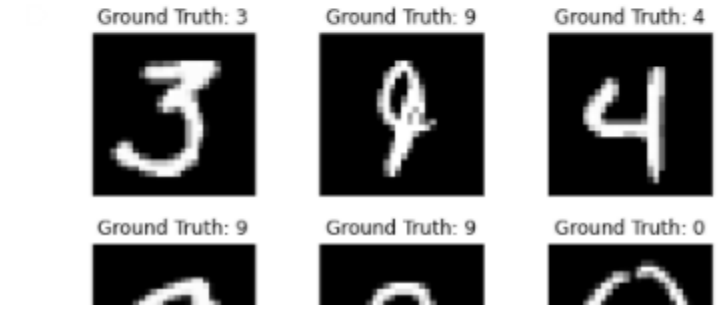
example_targets - это фактические числовые метки, соответствующие изображениям.

```
[4]  examples = enumerate(test_loader)
     batch_idx, (example_data, example_targets) = next(examples)
     print(example_targets)
     print(example_data.shape)
```

```
tensor([3, 9, 4, 9, 9, 0, 8, 3, 1, 2, 3, 9, 1, 3, 6, 6, 4, 4, 9, 7, 3, 7, 6, 3,
        4, 8, 4, 6, 8, 6, 1, 1, 1, 1, 0, 0, 1, 8, 4, 0, 1, 2, 7, 9, 3, 2, 3, 8,
        3, 2, 0, 4, 6, 6, 5, 5, 3, 0, 3, 7, 2, 4, 1, 6, 7, 5, 4, 1, 0, 8, 5, 9,
        0, 9, 6, 1, 8, 0, 9, 3, 5, 7, 8, 5, 6, 4, 2, 2, 2, 1, 8, 4, 4, 2, 1, 5
```

Мы можем использовать matplotlib для построения графика некоторых из них

```
[5]  import matplotlib.pyplot as plt
     fig = plt.figure()
     for i in range(6):
         plt.subplot(2,3,i+1)
         plt.tight_layout()
         plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
         plt.title("Ground Truth: {}".format(example_targets[i]))
         plt.xticks([])
         plt.yticks([])
     plt.show()
```



4.Создание сети

Теперь давайте начнем строить нашу сеть. Мы будем использовать два двухмерных конволюционных слоя, а затем два полностью связанных (или линейных) слоя. В качестве функции активации мы выберем выпрямленные линейные единицы (сокращенно ReLU), а в качестве средства регуляризации будем использовать два отсеивающих слоя. Хорошим способом построения сети в PyTorch является создание нового класса для сети, которую мы хотим построить. Давайте импортируем здесь некоторые подмодули, чтобы получить более читабельный код.

```
[6]  import torch.nn as nn
     import torch.nn.functional as F
     import torch.optim as optim

     class Net(nn.Module):
         def __init__(self):
             super(Net, self).__init__()
             self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
             self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
             self.conv2_drop = nn.Dropout2d()
             self.fc1 = nn.Linear(320, 50)
             self.fc2 = nn.Linear(50, 10)
         def forward(self, x):
             x = F.relu(F.max_pool2d(self.conv1(x), 2))
             x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
             x = x.view(-1, 320)
             x = F.relu(self.fc1(x))
             x = F.dropout(x, training=self.training)
             x = self.fc2(x)
             return F.log_softmax(x)
```

Инициализация сети и оптимизатора.

```
[7]  network = Net()
     optimizer = optim.SGD(network.parameters(), lr=learning_rate,
                           momentum=momentum)
```

5.Модельное обучение

Пришло время установить наш тренировочный цикл. Во-первых, мы убедимся, что наша сеть находится в режиме обучения. Затем для каждой эпохи выполняется одна итерация всех обучающих данных. Загрузка отдельных пакетов осуществляется с помощью DataLoader.

Во-первых, нам нужно вручную установить градиент на ноль с помощью optimizer.zero_grad(), поскольку PyTorch накапливает градиенты по умолчанию. Затем мы генерируем выход сети (прямой проход) и вычисляем отрицательную логарифмическую потерю вероятности между выходом и истинной меткой. Теперь мы собираем новый набор градиентов и распространяем их обратно на каждый параметр сети с помощью optimizer.step().

```
[8]  train_losses = []
     train_counter = []
     test_losses = []
     test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]
```

Перед началом обучения мы запустим тестовый цикл, чтобы посмотреть, сколько точности/потерь можно получить, используя только случайно инициализированные параметры сети.

```
[9]  def train(epoch):
         network.train()
         for batch_idx, (data, target) in enumerate(train_loader):
             optimizer.zero_grad()
             output = network(data)
             loss = F.nll_loss(output, target)
             loss.backward()
             optimizer.step()
             if batch_idx % log_interval == 0:
                 print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                     epoch, batch_idx * len(data), len(train_loader.dataset),
                     100. * batch_idx / len(train_loader), loss.item()))
                 train_losses.append(loss.item())
                 train_counter.append(
                     (batch_idx*64) + ((epoch-1)*len(train_loader.dataset)))
                 torch.save(network.state_dict(), './model.pth')
                 torch.save(optimizer.state_dict(), './optimizer.pth')

     train(1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: UserWarning: Implicit dimension
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.319280
Train Epoch: 1 [640/60000 (1%)] Loss: 2.290954
Train Epoch: 1 [1280/60000 (2%)]        Loss: 2.318535
```

Модули нейронных сетей, а также оптимизаторы могут сохранять и загружать свои внутренние состояния с помощью функции .state_dict(). Таким образом, при необходимости мы можем продолжить обучение на основе ранее сохраненного диктата состояния - просто вызвав .load_state_dict(state_dict).

Теперь войдите в цикл тестирования. Здесь мы суммируем потери при тестировании и отслеживаем количество правильных классификаций, чтобы рассчитать точность сети.

```
[10] def test():
         network.eval()
         test_loss = 0
         correct = 0
         with torch.no_grad():
             for data, target in test_loader:
                 output = network(data)
                 test_loss += F.nll_loss(output, target, size_average=False).item()
                 pred = output.data.max(1, keepdim=True)[1]
                 correct += pred.eq(target.data.view_as(pred)).sum()
         test_loss /= len(test_loader.dataset)
         test_losses.append(test_loss)
         print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
             test_loss, correct, len(test_loader.dataset),
             100. * correct / len(test_loader.dataset)))

     test()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: UserWarning: Implicit dime
/usr/local/lib/python3.7/dist-packages/torch/nn/_reduction.py:42: UserWarning: size_average
  warnings.warn(warning.format(ret))

Test set: Avg. loss: 0.1922, Accuracy: 9440/10000 (94%)
```

Используя контекстный менеджер no_grad(), мы можем избежать хранения вычислительных результатов вывода генеративной сети в вычислительном графе.

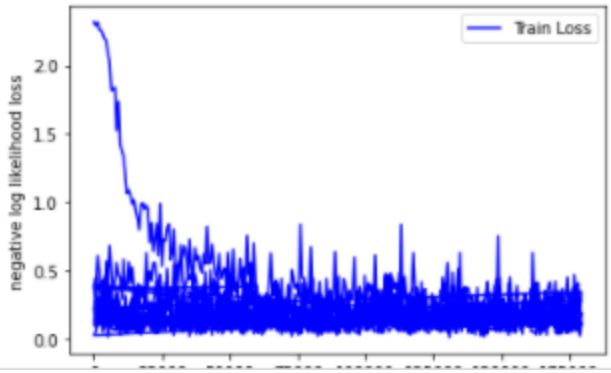
Пора начинать тренировки! Мы вручную добавим вызов test() перед циклом через n_epochs, чтобы оценить нашу модель, используя случайно инициализированные параметры.

```
[21] test()
     for epoch in range(1, n_epochs + 1):
         train(epoch)
         test()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.
/usr/local/lib/python3.7/dist-packages/torch/nn/_reduction
  warnings.warn(warning.format(ret))

Test set: Avg. loss: 0.0380, Accuracy: 9876/10000 (99%)

Train Epoch: 1 [0/60000 (0%)]   Loss: 0.392204
Train Epoch: 1 [640/60000 (1%)] Loss: 0.202859
Train Epoch: 1 [1280/60000 (2%)]        Loss: 0.133428
Train Epoch: 1 [1920/60000 (3%)]        Loss: 0.154377
```

Нарисуйте кривую обучения

```
[25] import matplotlib.pyplot as plt
     fig = plt.figure()
     plt.plot(train_counter, train_losses, color='blue')

     plt.legend(['Train Loss', 'Test Loss'], loc='upper right')
     plt.xlabel('number of training examples seen')
     plt.ylabel('negative log likelihood loss')
     plt.show()
```



Продолжить обучение

```
[26] examples = enumerate(test_loader)
     batch_idx, (example_data, example_targets) = next(examples)
     with torch.no_grad():
         output = network(example_data)
     fig = plt.figure()
     for i in range(6):
         plt.subplot(2,3,i+1)
         plt.tight_layout()
         plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
         plt.title("Prediction: {}".format(
             output.data.max(1, keepdim=True)[1][i].item()))
         plt.xticks([])
         plt.yticks([])
     plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: UserWarning
```

6.Постоянное обучение сотрудников контрольно-пропускных пунктов

Теперь давайте перейдем к обучению сети или посмотрим, как можно продолжить обучение на основе state_dicts, сохраненных после первого запуска обучения. Мы инициализируем новый набор сетей и оптимизаторов.

```
[27] continued_network = Net()
     continued_optimizer = optim.SGD(network.parameters(), lr=learning_rate,
                                                           momentum=momentum)
```

Используя .load_state_dict(), мы можем загрузить внутренние состояния сети и оптимизировать их на момент последнего сохранения.

```
[28] network_state_dict = torch.load('model.pth')
     continued_network.load_state_dict(network_state_dict)
     optimizer_state_dict = torch.load('optimizer.pth')
     continued_optimizer.load_state_dict(optimizer_state_di
```

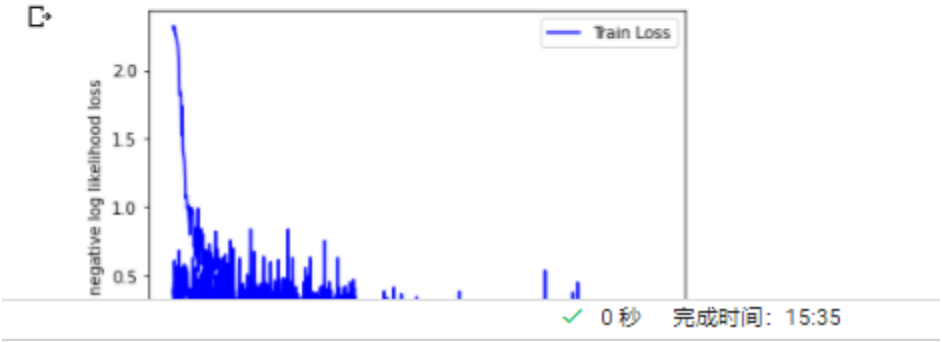
Опять же, запуск тренировочного цикла должен немедленно возобновить нашу предыдущую тренировку. Чтобы проверить это, мы просто используем тот же список, что и раньше, для отслеживания значений потерь. Из-за того, как мы построили счетчик количества обучающих примеров для теста, здесь нам пришлось добавить его вручную

```
[29] for i in range(4,9):
       test_counter.append(i*len(train_loader.dataset))
       train(i)
       test()
```

```
Train Epoch: 4 [3840/60000 (6%)]        Loss: 0.014168
Train Epoch: 4 [4480/60000 (7%)]        Loss: 0.224046
Train Epoch: 4 [5120/60000 (9%)]        Loss: 0.091797
Train Epoch: 4 [5760/60000 (10%)]       Loss: 0.147034
Train Epoch: 4 [6400/60000 (11%)]       Loss: 0.089756
Train Epoch: 4 [7040/60000 (12%)]       Loss: 0.038675
```

И снова мы видим улучшение точности тестового набора от одной эпохи к другой (работает медленнее, намного медленнее). Давайте воспользуемся изображениями для дальнейшей проверки хода обучения.

```
fig = plt.figure()
plt.plot(train_counter, train_losses, color='blue')

plt.legend(['Train Loss', 'Test Loss'], loc='upper right')
plt.xlabel('number of training examples seen')
plt.ylabel('negative log likelihood loss')
plt.show()
```



Список литературы

[1]Гапанюк Ю. Е. Домашнее задание (вариант 1) «решение задачи обучения с учителем» [Электронный ресурс] // GitHub. — 2021. — Режим доступа:

[2] Li Yandong, Hao Zongbo, Lei Hang. Обзор исследований в области конволюционных нейронных сетей[J]. Computer Applications,2016(9):2508-2515,2565.https://github.com/ugapanyuk/ml_course_2021/wiki/DZ_MMO__SUPERVISED