202001555 지은미

In [1]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import transforms, datasets
```

In [2]:

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

In [3]:

```
EPOCHS = 50
BATCH_SIZE = 64
```

In [4]:

```
loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data',
                   train=True,
                   download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.ToTensor()])),
    batch_size=32,
    num_workers=0,
    shuffle=False)

mean = torch.mean(loader.dataset.data.float()/255.0)
std = torch.std(loader.dataset.data.float()/255.0)

print("The mean is ", mean)
print("The standard deviation is ", std)
```

```
The mean is  tensor(0.1307)
The standard deviation is  tensor(0.3081)
```

In [5]:

```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./.data',
                   train=True,
                   download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.ToTensor(),
                       transforms.Normalize((mean,), (std,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./.data',
                   train=False,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((mean,), (std,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
```

In [6]:

```
class Net(nn.Module):
    def __init__(self, dropout_p=0.2):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        # 드롭아웃 확률
        self.dropout_p = dropout_p

    def forward(self, x):
        x = x.view(-1, 784)
        x = F.relu(self.fc1(x))
        # 드롭아웃 추가
        x = F.dropout(x, training=self.training,
                      p=self.dropout_p)
        x = F.relu(self.fc2(x))
        # 드롭아웃 추가
        x = F.dropout(x, training=self.training,
                      p=self.dropout_p)
        x = self.fc3(x)
        return x
```

In [7]:

```
model        = Net(dropout_p=0.2).to(DEVICE)
optimizer    = optim.SGD(model.parameters(), lr=0.01)
```

In [8]:

```
def train(model, train_loader, optimizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(DEVICE), target.to(DEVICE)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
```

In [9]:

```
def evaluate(model, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)
            test_loss += F.cross_entropy(output, target,
                                         reduction='sum').item()

            # 맞춘 갯수 계산
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

In [10]:

```
for epoch in range(1, EPOCHS + 1):
    train(model, train_loader, optimizer)
    test_loss, test_accuracy = evaluate(model, test_loader)

    print('[{}] Test Loss: {:.4f}, Accuracy: {:.2f}%'.format(
          epoch, test_loss, test_accuracy))
```

```
[1] Test Loss: 0.5446, Accuracy: 82.67%
```

```
[2] Test Loss: 0.4270, Accuracy: 86.39%
[3] Test Loss: 0.3507, Accuracy: 89.04%
[4] Test Loss: 0.2938, Accuracy: 91.05%
[5] Test Loss: 0.2561, Accuracy: 92.15%
[6] Test Loss: 0.2257, Accuracy: 93.43%
[7] Test Loss: 0.2016, Accuracy: 93.83%
[8] Test Loss: 0.1896, Accuracy: 94.19%
[9] Test Loss: 0.1751, Accuracy: 94.62%
[10] Test Loss: 0.1657, Accuracy: 94.87%
[11] Test Loss: 0.1546, Accuracy: 95.28%
[12] Test Loss: 0.1490, Accuracy: 95.41%
[13] Test Loss: 0.1442, Accuracy: 95.59%
[14] Test Loss: 0.1397, Accuracy: 95.64%
[15] Test Loss: 0.1346, Accuracy: 95.71%
[16] Test Loss: 0.1322, Accuracy: 95.76%
[17] Test Loss: 0.1299, Accuracy: 95.90%
[18] Test Loss: 0.1239, Accuracy: 96.07%
[19] Test Loss: 0.1202, Accuracy: 96.31%
[20] Test Loss: 0.1161, Accuracy: 96.43%
[21] Test Loss: 0.1160, Accuracy: 96.38%
[22] Test Loss: 0.1144, Accuracy: 96.45%
[23] Test Loss: 0.1114, Accuracy: 96.38%
[24] Test Loss: 0.1117, Accuracy: 96.41%
[25] Test Loss: 0.1082, Accuracy: 96.67%
[26] Test Loss: 0.1085, Accuracy: 96.47%
[27] Test Loss: 0.1049, Accuracy: 96.69%
[28] Test Loss: 0.1032, Accuracy: 96.70%
[29] Test Loss: 0.0992, Accuracy: 96.94%
[30] Test Loss: 0.1001, Accuracy: 96.80%
[31] Test Loss: 0.0983, Accuracy: 96.94%
[32] Test Loss: 0.0959, Accuracy: 96.92%
[33] Test Loss: 0.0973, Accuracy: 96.89%
[34] Test Loss: 0.0951, Accuracy: 96.93%
[35] Test Loss: 0.0937, Accuracy: 97.10%
[36] Test Loss: 0.0979, Accuracy: 96.88%
[37] Test Loss: 0.0914, Accuracy: 97.11%
[38] Test Loss: 0.0932, Accuracy: 97.03%
[39] Test Loss: 0.0932, Accuracy: 96.95%
[40] Test Loss: 0.0898, Accuracy: 97.05%
[41] Test Loss: 0.0889, Accuracy: 97.20%
[42] Test Loss: 0.0892, Accuracy: 97.10%
[43] Test Loss: 0.0888, Accuracy: 97.20%
[44] Test Loss: 0.0889, Accuracy: 97.17%
[45] Test Loss: 0.0879, Accuracy: 97.21%
[46] Test Loss: 0.0859, Accuracy: 97.24%
[47] Test Loss: 0.0852, Accuracy: 97.18%
[48] Test Loss: 0.0847, Accuracy: 97.30%
[49] Test Loss: 0.0839, Accuracy: 97.31%
[50] Test Loss: 0.0835, Accuracy: 97.37%
```

202001555 지은미