201600282 엄기산

4.3 오버피팅과 정규화 (Overfitting and Regularization)

In [7]:

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

In [8]:

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

In [9]:

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

데이터셋에 노이즈 추가하기

In [10]:

```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./.data',
                   train=True,
                   download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3015,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./.data',
                   train=False,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3015,))
                   ])),
    batch_size=BATCH_SIZE, shuffle=True)
```

뉴럴넷으로 Fashion MNIST 학습하기

In [11]:

```
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
```

모델 준비하기

In [12]:

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

학습하기

In [13]:

```
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()
```

테스트하기

In [14]:

```
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
```

코드 돌려보기

In [15]:

```
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: 0.5422, Accuracy: 82.76%
[2] Test Loss: 0.4180, Accuracy: 87.08%
[3] Test Loss: 0.3497, Accuracy: 89.20%
[4] Test Loss: 0.2941, Accuracy: 91.06%
[5] Test Loss: 0.2531, Accuracy: 92.35%
[6] Test Loss: 0.2219, Accuracy: 93.47%
[7] Test Loss: 0.2021, Accuracy: 94.03%
```

```
[8] Test Loss: 0.1895, Accuracy: 94.26%
[9] Test Loss: 0.1762, Accuracy: 94.74%
[10] Test Loss: 0.1661, Accuracy: 95.00%
[11] Test Loss: 0.1582, Accuracy: 95.03%
[12] Test Loss: 0.1526, Accuracy: 95.17%
[13] Test Loss: 0.1428, Accuracy: 95.58%
[14] Test Loss: 0.1417, Accuracy: 95.60%
[15] Test Loss: 0.1324, Accuracy: 95.81%
[16] Test Loss: 0.1312, Accuracy: 95.90%
[17] Test Loss: 0.1266, Accuracy: 96.06%
[18] Test Loss: 0.1237, Accuracy: 96.15%
[19] Test Loss: 0.1204, Accuracy: 96.25%
[20] Test Loss: 0.1177, Accuracy: 96.28%
[21] Test Loss: 0.1175, Accuracy: 96.23%
[22] Test Loss: 0.1158, Accuracy: 96.37%
[23] Test Loss: 0.1117, Accuracy: 96.47%
[24] Test Loss: 0.1097, Accuracy: 96.59%
[25] Test Loss: 0.1074, Accuracy: 96.67%
[26] Test Loss: 0.1036, Accuracy: 96.78%
[27] Test Loss: 0.1047, Accuracy: 96.73%
[28] Test Loss: 0.1028, Accuracy: 96.64%
[29] Test Loss: 0.1020, Accuracy: 96.65%
[30] Test Loss: 0.0987, Accuracy: 96.90%
[31] Test Loss: 0.0972, Accuracy: 96.97%
[32] Test Loss: 0.0956, Accuracy: 97.09%
[33] Test Loss: 0.0973, Accuracy: 96.94%
[34] Test Loss: 0.0977, Accuracy: 96.97%
[35] Test Loss: 0.0948, Accuracy: 97.09%
[36] Test Loss: 0.0975, Accuracy: 96.94%
[37] Test Loss: 0.0932, Accuracy: 97.17%
[38] Test Loss: 0.0926, Accuracy: 97.13%
[39] Test Loss: 0.0945, Accuracy: 96.98%
[40] Test Loss: 0.0924, Accuracy: 97.17%
[41] Test Loss: 0.0928, Accuracy: 97.12%
[42] Test Loss: 0.0914, Accuracy: 97.10%
[43] Test Loss: 0.0916, Accuracy: 97.21%
[44] Test Loss: 0.0895, Accuracy: 97.28%
[45] Test Loss: 0.0892, Accuracy: 97.25%
[46] Test Loss: 0.0890, Accuracy: 97.21%
[47] Test Loss: 0.0883, Accuracy: 97.35%
[48] Test Loss: 0.0874, Accuracy: 97.28%
[49] Test Loss: 0.0866, Accuracy: 97.42%
[50] Test Loss: 0.0858, Accuracy: 97.39%
```

201600282 엄기산

```
In [ ]:
```