



Fashion-MNIST Project

<input checked="" type="radio"/> 상태	DNN
👤 담당자	

다루어야 할 데이터 정보

모델 아웃라인

모델 설계

패키지 로드

하이퍼파라미터 세팅

Dataset 및 DataLoader 할당

뉴럴 네트 설계

Weight Initialization

모델 생성

학습

Loss function 및 Optimizer 정의

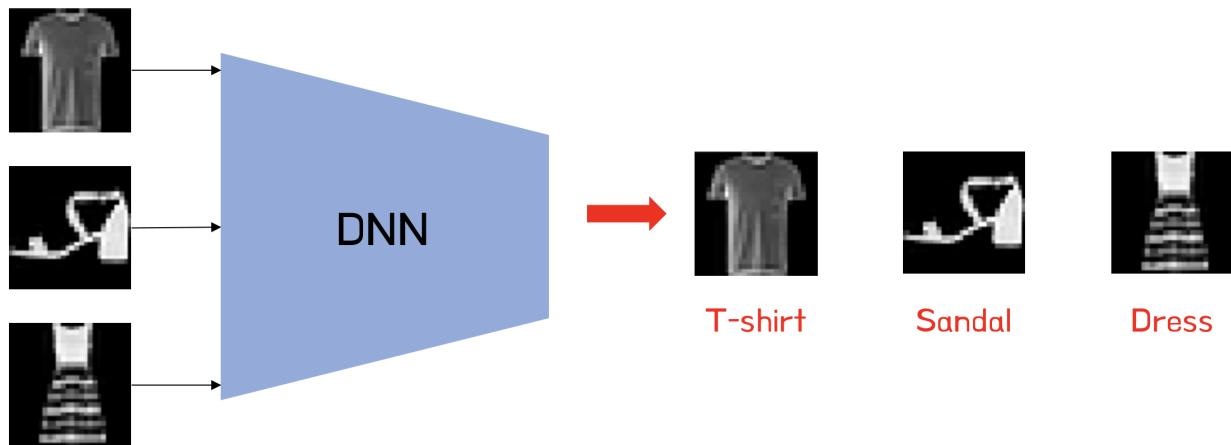
학습

평가

다루어야 할 데이터 정보

Fashion-MNIST 데이터는 $1 \times 28 \times 28$ (채널 x 이미지 높이 x 이미지 너비) 의 이미지 데이터이다. 일반적인 이미지가 R, G, B의 3채널로 구성된 것과 달리 gray scale이라는 점이 특징이다.

모델 아웃라인



- $1 \times 28 \times 28 \rightarrow \text{DNN} \rightarrow 10\text{차원(의류 종류)}$

모델 설계

패키지 로드

```
import torch
import torch.nn as nn
import torchvision.datasets as dset
import torchvision.transforms as transforms
```

하이퍼파라미터 세팅

```
batch_size = 100
num_epochs = 5
learning_rate = 0.001
```

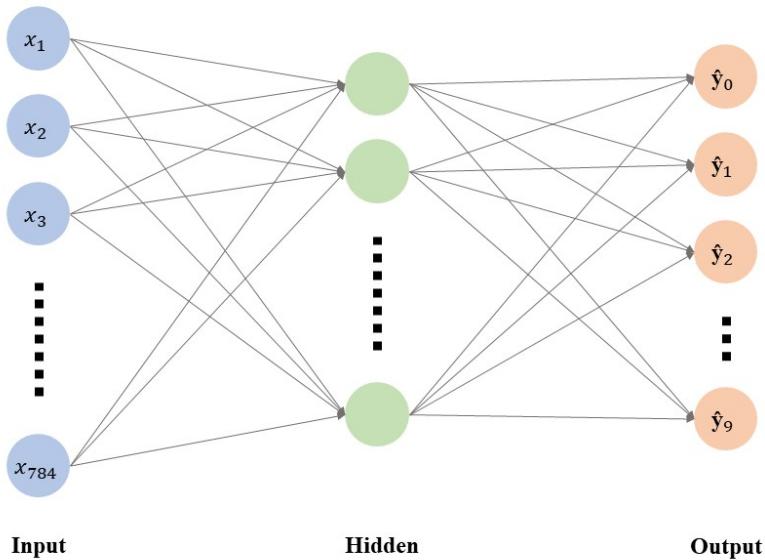
Dataset 및 DataLoader 할당

```
from torch.utils.data import DataLoader

root = './data'
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean=(0.5,), std=(0.5,))])
train_data = dset.FashionMNIST(root=root, train=True, transform=transform, download=True)
test_data = dset.FashionMNIST(root=root, train=False, transform=transform, download=True)
## 코드 시작 ##
train_loader = DataLoader(dataset=train_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=batch_size, shuffle=False)
## 코드 종료 ##
```

- **transform** : 각종 전처리(이미지 픽셀 값 정규화, 텐서 형태로 변경 등)

뉴럴 네트 설계



Multilayer Perceptron (MLP)

이번 예제에서는 fully connected neural net을 사용한다.

```
class DNN(nn.Module):
    def __init__(self, num_classes=10):
        super(DNN, self).__init__()
        self.layer1 = nn.Sequential(
            ## 코드 시작 ##
            nn.Linear(28*28, 512),      # Linear_1 해당하는 층
            nn.BatchNorm1d(512),         # BatchNorm_1 해당하는 층
            nn.ReLU()                   # ReLU_1 해당하는 층
            ## 코드 종료 ##
        )
        self.layer2 = nn.Sequential(
            ## 코드 시작 ##
            nn.Linear(512, 10)       # Linear_2 해당하는 층
            ## 코드 종료 ##
        )

    def forward(self, x):
        x = x.view(x.size(0), -1) # flatten
        x_out = self.layer1(x)
        x_out = self.layer2(x_out)
        return x_out
```

Weight Initialization

```
def weights_init(m):
    if isinstance(m, nn.Linear): # 모델의 모든 MLP 레이어에 대해서
        nn.init.xavier_normal_(m.weight) # Weight를 xavier_normal로 초기화
        print(m.weight)
```

모델 생성

`model.apply()`을 통해 위에서 정의한 `weights_init` 함수를 적용할 수 있고, weight를 일괄적으로 초기화 할 수 있다.

```
torch.manual_seed(7777) # 일관된 weight initialization을 위한 random seed 설정
model = DNN().to(device)
model.apply(weights_init) # 모델에 weight_init 함수를 적용하여 weight를 초기화
```

학습

Loss function 및 Optimizer 정의

Binary classification 문제를 해결하고자 하므로, Cross Entropy Error를 사용한다. Adam optimizer는 많은 경우에 잘 작동하므로 이를 사용한다.

```
## 코드 시작 ##
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
## 코드 종료 ##
```

학습

```
for epoch in range(num_epochs):
    for i, (imgs, labels) in enumerate(train_loader):
        imgs, labels = imgs.to(device), labels.to(device)
        ## 코드 시작 ##
        outputs = model(imgs) # 위의 설명 1. 을 참고하여 None을 채우세요.
        loss = criterion(outputs, labels) # 위의 설명 2. 을 참고하여 None을 채우세요.

        optimizer.zero_grad() # Clear gradients: 위의 설명 3. 을 참고하여 None을 채우세요.
        loss.backward() # Gradients 계산: 위의 설명 4. 을 참고하여 None을 채우세요.
        optimizer.step() # Parameters 업데이트: 위의 설명 5. 을 참고하여 None을 채우세요.
        ## 코드 종료 ##

        _, argmax = torch.max(outputs, 1)
```

```

accuracy = (labels == argmax).float().mean()

if (i+1) % 100 == 0:
    print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Accuracy: {:.2f}%'.format(
        epoch+1, num_epochs, i+1, len(train_loader), loss.item(), accuracy.item() * 100))

```

평가

```

model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for i, (imgs, labels) in enumerate(test_loader):
        imgs, labels = imgs.to(device), labels.to(device)
        outputs = model(imgs)
        _, argmax = torch.max(outputs, 1) # max()를 통해 최종 출력이 가장 높은 class 선택
        total += imgs.size(0)
        correct += (labels == argmax).sum().item()

print('Test accuracy for {} images: {:.2f}%'.format(total, correct / total * 100))

```

▼  해왔던 방식에 세련된 weight initialization 방식 적용

```

class DNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(28*28, 512)
        self.layer2 = nn.Linear(512, 10)
        self.relu = nn.ReLU()
        self.model = nn.Sequential(self.layer1, self.relu,
                                  self.layer2)

    def forward(self, x):
        return model(x)

```

```

def weights_init(m):
    if isinstance(m, nn.Linear): # 모델의 모든 MLP 레이어에 대해서
        nn.init.xavier_normal_(m.weight) # Weight를 xavier_normal로 초기화
        print(m.weight)

```

```

torch.manual_seed(7777) # 일관된 weight initialization을 위한 random seed 설정
model = DNN().to(device)
model.apply(weights_init) # 모델에 weight_init 함수를 적용하여 weight를 초기화

```

