

pytorch fashion-mnist 실습

학습 목표

- pytorch를 활용하여 모델을 구축해 본다.
- 데이터 셋은 fashion-mnist를 활용한다.

목차

[01 데이터 및 라이브러리 불러오기](#)

[02. 데이터를 배치 단위로 가져오기 - DataLoader](#)

[03. 모델 정의 및 구축](#)

[04. 모델 학습](#)

[05. 모델 평가](#)

[06. 모델 학습 및 검증](#)

01 데이터 및 라이브러리 불러오기

[목차로 이동하기](#)

- torchvision
 - torchvision 패키지는 컴퓨터 비전을 위한 데이터 셋, 모델 구조, 컴퓨터 비전을 위한 여러가지 기능 패키지
 - torchvision을 이용하여 CIFAR10 훈련, 테스트 데이터 셋을 불러오고 정규화한다.
 - 설치 : pip install torchvision

In [1]:

```
import numpy as np

import torch
import torchvision

from torch.utils.data import Dataset
from torchvision import datasets, transforms

print(torch.__version__)
print(torchvision.__version__)
```

1.13.0+cpu

0.14.0+cpu

- 데이터 목록 확인
 - <https://pytorch.org/vision/stable/datasets.html> (<https://pytorch.org/vision/stable/datasets.html>)

Image Transform

- torchvision의 transforms를 활용하여 정규화를 적용가능
- transforms.ToTensor()
 - 정규화(Normalize)한 결과가 0~1 범위로 변환

In [2]:

```
transform = transforms.Compose([
    transforms.ToTensor(),
])
```

Fashion MNIST DataSet



Fashion MNIST 데이터셋 로드

- 데이터 출처
 - <https://github.com/zalando-research/fashion-mnist> (<https://github.com/zalando-research/fashion-mnist>)

학습용 및 테스트용 데이터 셋 가져오기

In [3]:

```
train_data = datasets.FashionMNIST(root='data',  
                                   train=True,      # 학습용 데이터셋 설정(True)  
                                   download=True,  
                                   transform=transform # 정규화  
                                   )
```

In [4]:

```
test_data = datasets.FashionMNIST(root='data',  
                                  train=False,    # 검증용 데이터셋 설정(False)  
                                  download=True,  
                                  transform=transform  
                                  )
```

데이터 시각화

In [5]:

```
import matplotlib.pyplot as plt
```

In [6]:

```
class_names = {  
    0: "t-shirt/top",  
    1: "trouser",  
    2: "pullover",  
    3: "dress",  
    4: "coat",  
    5: "sandal",  
    6: "shirt",  
    7: "sneaker",  
    8: "bag",  
    9: "ankle boot",  
}
```

In [7]:

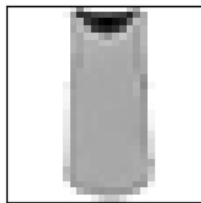
```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1) # 그래프의 표시 위치
    img, label = train_data[i]
    plt.xticks([])
    plt.yticks([])
    plt.grid(False) # 그리드선
    plt.imshow(torch.permute(img, (1, 2, 0)), cmap=plt.cm.binary)
    plt.xlabel(class_names[label])
plt.show()
```



ankle boot



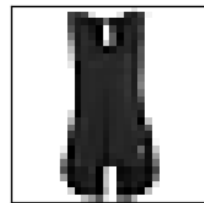
t-shirt/top



t-shirt/top



dress



t-shirt/top



pullover



sneaker



pullover



sandal



sandal



t-shirt/top



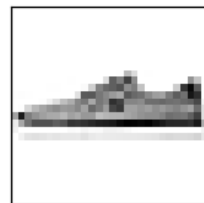
ankle boot



sandal



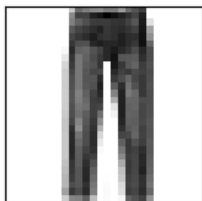
sandal



sneaker



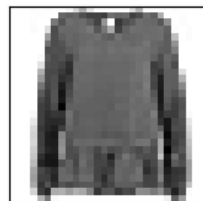
ankle boot



trouser



t-shirt/top



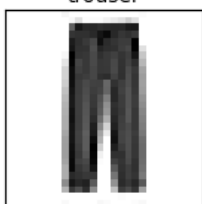
shirt



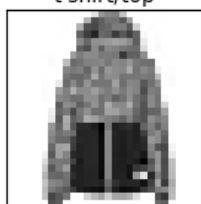
coat



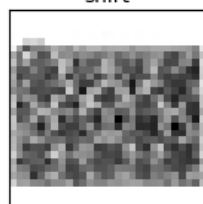
dress



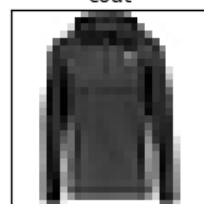
trouser



coat



bag



coat

02. 데이터를 배치 단위로 가져오기 - DataLoader

[목차로 이동하기](#)

In [8]:

```
import os
os.cpu_count()
```

Out[8]:

12

In [9]:

```
batch_size = 32 # batch_size 지정
num_workers = 8 # Thread 숫자 지정
```

In [10]:

```
train_loader = torch.utils.data.DataLoader(train_data,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=num_workers)
```

In [11]:

```
test_loader = torch.utils.data.DataLoader(test_data,
                                           batch_size=batch_size,
                                           shuffle=False,
                                           num_workers=num_workers)
```

train_loader 활용하여 하나의 배치 확인

In [12]:

```
# 1개의 배치 추출 후 Image, label의 shape 출력
img, lbl = next(iter(train_loader))
img.shape, lbl.shape
```

Out[12]:

(torch.Size([32, 1, 28, 28]), torch.Size([32]))

- 배치 사이즈가 32, 채널(1), 세로(28), 가로(28)

03. 모델 정의 및 구축

목차로 이동하기

- cuda 설정이 있다면 cuda
- cpu 설정이 있다면 cpu로 학습 진행

In [13]:

```
torch.cuda.is_available()
```

Out[13]:

False

In [14]:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")  
print(device)
```

cpu

- GPU가 지정되어 있다면 cuda:0, 2대는 cuda:1로 지정

In [15]:

```
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim
```

- torch.nn.functional : <https://pytorch.org/docs/stable/nn.functional.html>
(<https://pytorch.org/docs/stable/nn.functional.html>).

In [16]:

```
class DNNModel(nn.Module):  
    def __init__(self):  
        super(DNNModel, self).__init__()  
        self.fc1 = nn.Linear(28*28, 128)  
        self.fc2 = nn.Linear(128, 32)  
        self.output = nn.Linear(32, 10)  
  
    def forward(self, x):  
        # 텐서는 같지만 새로운 텐서 반환(모양 변환)  
        x = x.view(-1, 28*28)  
        x = F.relu(self.fc1(x))  
        x = F.relu(self.fc2(x))  
        x = self.output(x)  
        return x
```

In [17]:

```
model = DNNModel() # Model 생성
model.to(device)   # device 로드
```

Out[17]:

```
DNNModel(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=32, bias=True)
  (output): Linear(in_features=32, out_features=10, bias=True)
)
```

최적화함수(optimizer) 및 손실함수(loss function)

- 옵티마이저는 `model.parameters()`를 지정해야 한다.
- 다항분류이므로 `CrossEntropy` 손실을 지정한다.

In [18]:

```
optimizer = optim.Adam(model.parameters(), lr=0.0005)
loss_fn = nn.CrossEntropyLoss()
```

04. 모델 학습

목차로 이동하기

In [19]:

```
from tqdm import tqdm
```

- `model.train()`
 - 모델을 학습 모드로 설정한다. 학습 모드일때 Gradient가 업데이트가 가능하다. 반드시 `train()`로 변경 필요.
- `optimizer.zero_grad()`
 - Gradient를 초기화
 - 한번의 학습이 완료가 되면, `gradients`를 항상 0으로 만들어 주어야 한다.
 - 만약 0으로 초기화가 되지 않으면 의도한 방향과 다른 방향으로 가게 될 가능성이 있다.
- `loss = loss_fn(output, lbl)`
 - 손실함수를 이용하여 손실계산
- `loss.backward()` : 예측 손실을 역전파. Pytorch는 각 매개변수에 대한 손실의 변화도를 저장.
- `optimizer.step()` : 역전파 단계에서 Gradient를 업데이트 시키기.
- `total_batch_loss += loss.item() * img.size(0)`
 - `loss.item()`은 1개 배치의 평균 손실(loss)
 - `img.size(0)`은 배치사이즈(batch size)
 - 1개 배치의 전체 손실을 구해서 계속 더해준다.
- `acc = total_num / len(data_loader.dataset)`

- `total_num` : 전체 누적된 잘 맞춘 개수
- `len(data_loader.dataset)` : 전체 데이터 개수

In [20]:

```
def model_train(model, data_loader, loss_fn, optimizer, device):
    model.train()

    # 초기화
    total_batch_loss = 0    # 총 손실
    total_num = 0          # 정답 개수

    progress_bar = tqdm(data_loader)

    # mini-batch 학습
    for img, lbl in progress_bar:
        # image, label 데이터를 device에 올리기.
        img, lbl = img.to(device), lbl.to(device)

        optimizer.zero_grad() # 그래디언트 초기화
        output = model(img)    # Forward Propagation을 진행. 결과 획득.

        loss = loss_fn(output, lbl) # 손실 계산
        loss.backward()          # 오차역전파(Back Propagation) 진행. 미분 값을 계산
        optimizer.step()        # 가중치 업데이트

        # output의 max(dim=1)은 max probability와 max index를 반환.
        # max probability는 무시하고, max index는 pred에 저장
        _, pred = output.max(dim=1)

        # pred.eq(lbl).sum() 은 정확히 맞춘 label의 합계를 계산합니다.
        # item()은 tensor에서 값을 추출.
        total_num += pred.eq(lbl).sum().item() #

        # 이를 누적한 뒤 Epoch 종료시 전체 데이터셋의 개수로 나누어 평균 loss를 산출합니다.
        # total_batch_loss에 1개 배치의 전체 loss를 더하기
        total_batch_loss += loss.item() * img.size(0)

    acc = total_num / len(data_loader.dataset) # 정확도 계산

    # 평균 손실(loss)과 정확도를 반환합니다.
    # train_loss, train_acc
    return total_batch_loss / len(data_loader.dataset), acc
```

05. 모델 평가

목차로 이동하기

- `model.eval()`
 - 모델을 평가모드로 변경합니다.
 - dropout와 같은 layer의 역할 변경을 위해 evaluation 진행시 꼭 필요한 절차이다.
- `with torch.no_grad():`
 - Pytorch는 autograd engine를 off를 시킨다.
 - 자동으로 gradient를 트래킹하지 않겠다라는 의미.

- 주된 목적은 autograd를 off를 시켜 메모리 사용량을 줄이고, 연산속도의 향상을 가져온다.
- `total_num += torch.sum(pred.eq(lbl)).item()`
 - `pred.eq(lbl)` : 정확하게 맞추어는가?
 - `torch.sum(pred.eq(lbl)).item()` : 맞춘 개수
- `total_batch_loss += loss_fn(output, lbl).item() * img.size(0)`
 - `loss_fn(output, lbl).item()`은 1개 배치의 평균 손실(loss)
 - `img.size(0)`은 배치사이즈(batch size)
 - 1개 배치의 전체 손실을 구해서 계속 더해준다.

In [21]:

```
def model_evaluate(model, data_loader, loss_fn, device):
    model.eval() # model.eval()은 모델을 평가모드로 설정 변경

    with torch.no_grad():
        # 손실과 정확도 계산을 위한 초기화
        total_num = 0
        total_batch_loss = 0

        # 배치별 evaluation을 진행
        for img, lbl in data_loader:

            # device에 데이터를 올리기
            img, lbl = img.to(device), lbl.to(device)

            output = model(img) # Forward Propagation을 진행. 결과 획득.

            # output의 max(dim=1)은 max probability와 max index를 반환.
            _, pred = output.max(dim=1)
            total_num += torch.sum(pred.eq(lbl)).item() # 정확한 것 개수 더하기(누적)

            # 이를 누적한 뒤 Epoch 종료시 전체 데이터셋의 개수로 나누어 평균 loss를 산출합니다.
            total_batch_loss += loss_fn(output, lbl).item() * img.size(0)

    acc = total_num / len(data_loader.dataset) # 정확도 계산

    # 결과를 반환 - val_loss, val_acc
    return total_batch_loss / len(data_loader.dataset), acc
```

06. 모델 학습 및 검증

[목차로 이동하기](#)

%%time

모델의 가중치를 가져와 검증 손실과 검증 정확도를 계산

In [23]:

```
# 가중치 로드
model.load_state_dict(torch.load('DNNModel.pth'))

# 최종 검증 손실(validation loss)와 검증 정확도(validation accuracy)를 계산
loss, acc = model_evaluate(model, test_loader, loss_fn, device)
print(f'검증 손실: {loss:.5f}, 평가 정확도: {acc:.5f}')
```

검증 손실: 0.32401, 평가 정확도: 0.88990

REF

- 데이터 목록 확인
 - <https://pytorch.org/vision/stable/datasets.html> (<https://pytorch.org/vision/stable/datasets.html>).
- torch.nn.functional : <https://pytorch.org/docs/stable/nn.functional.html> (<https://pytorch.org/docs/stable/nn.functional.html>).
- https://qiita.com/sugulu_Ogawa_ISID/items/62f5f7adee083d96a587 (https://qiita.com/sugulu_Ogawa_ISID/items/62f5f7adee083d96a587).