BRFRAME
AI&AA framework

AI Prebuild Service &
Automated Machine Learning Leader

한양대 분석가 교육과정

Introduction to PyTorch

2021년 06월 16일

비알프레임

Machine Learning &
Predictive Analytics

01. Introduction to PyTorch

- **학습 목표**
  - 머신러닝 모델을 구축하는 데 사용되는 주요 개념 학습
  - Computer Vision 모델을 구축하는 방법 학습
  - PyTorch API를 사용하여 모델을 구축하기

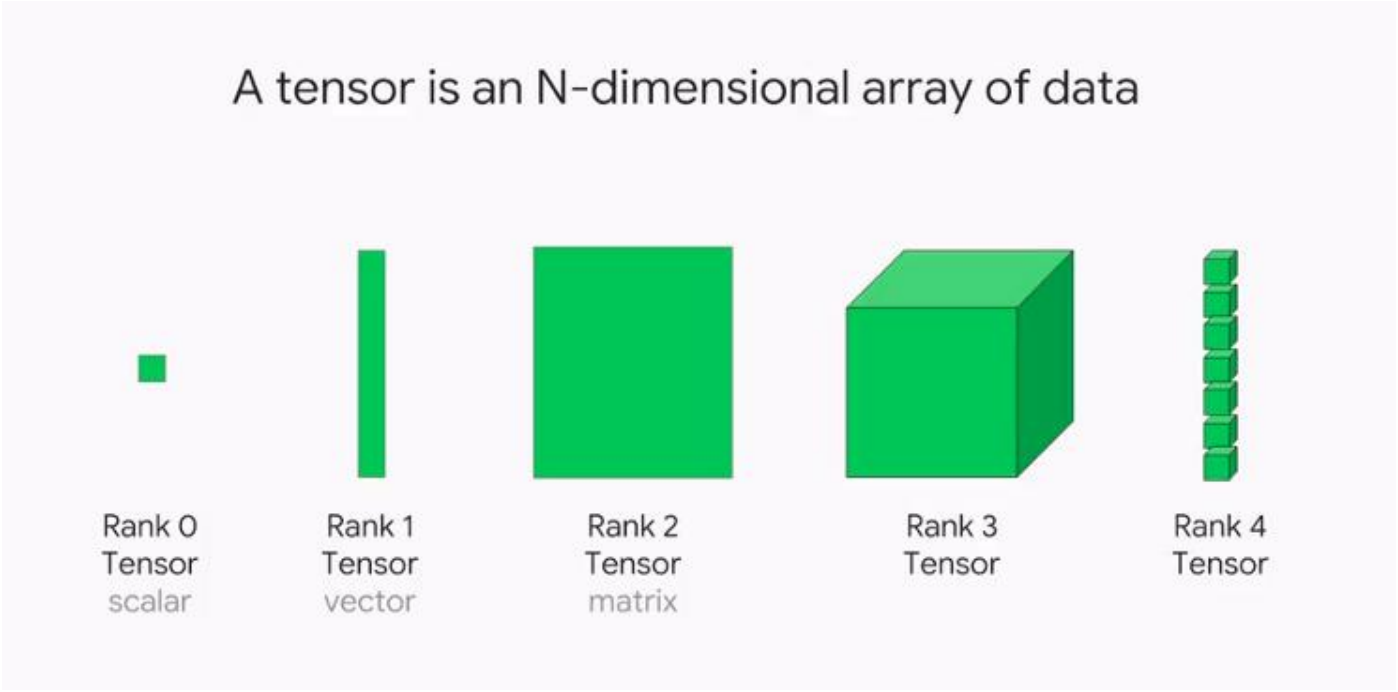
- **전제 조건**
  - 기본 Python 지식
  - Jupyter Notebooks 사용 방법에 대한 기본 지식

01. Introduction to PyTorch

- **1. 텐서(Tensors)란?**

  - 텐서(Tensors)는 배열(array) 및 행렬(matrix)과 매우 유사한 전문화된 데이터 구조이다.
    PyTorch에서는 텐서를 사용하여 모델의 입력 및 출력 뿐만 아니라 모델의 매개 변수를 인코딩한다.

  - 텐서는 GPU 또는 다른 하드웨어 가속기에서 실행할 수 있다는 점을 제외하면 Numpy의 ndarray와 유사하다.
    실제로 텐서와 Numpy의 배열은 동일한 기본 메모리를 공유할 수 있으므로 데이터를 복사할 필요가 없으며, 텐서는 자동 미분에 최적화되어 있다는 특징이 있다.



이미지 출처 : https://medium.com/xyzeelearn/machine-learning-pytorch-590b09409236

01. Introduction to PyTorch

- **2. 텐서(Tensors) 생성**

1) 데이터에서 직접 만드는 방법
➔ torch.tensor()

2) Numpy의 배열(array)에서 텐서를 생성하는 방법
➔ torch.from_numpy()

3) 또 다른 텐서로부터 텐서를 생성하는 방법 (이전 텐서의 모양과 데이터 타입 속성은 유지)
➔ torch.ones_like() 또는 torch.rand_like()

4) 임의의 모양 또는 차원에 대하여 랜덤 또는 상수 값의 텐서를 생성하는 방법
➔ torch.rand(), torch.ones(), torch.zeros()

01. Introduction to PyTorch

- **3. 텐서(Tensors)의 특성 및 연산**

  - 텐서의 속성 : 모양(shape), 데이터 타입(dtype), 데이터가 저장되어 있는 장치(device)로 구성

  - 텐서의 연산 : indexing, slicing, joining 등 100여가지가 넘는 다양한 연산을 수행할 수 있으며, 이러한 작업은 GPU에서 수행하여 CPU보다 더욱 빠른 속도로 연산할 수 있다.

    ※ 기본적으로 텐서는 CPU에서 생성되며, GPU의 가용성을 확인한 후 .to 메서드를 사용하여 텐서를 GPU로 이동

```
if torch.cuda.is_available():
    tensor = tensor.to('cuda')
```

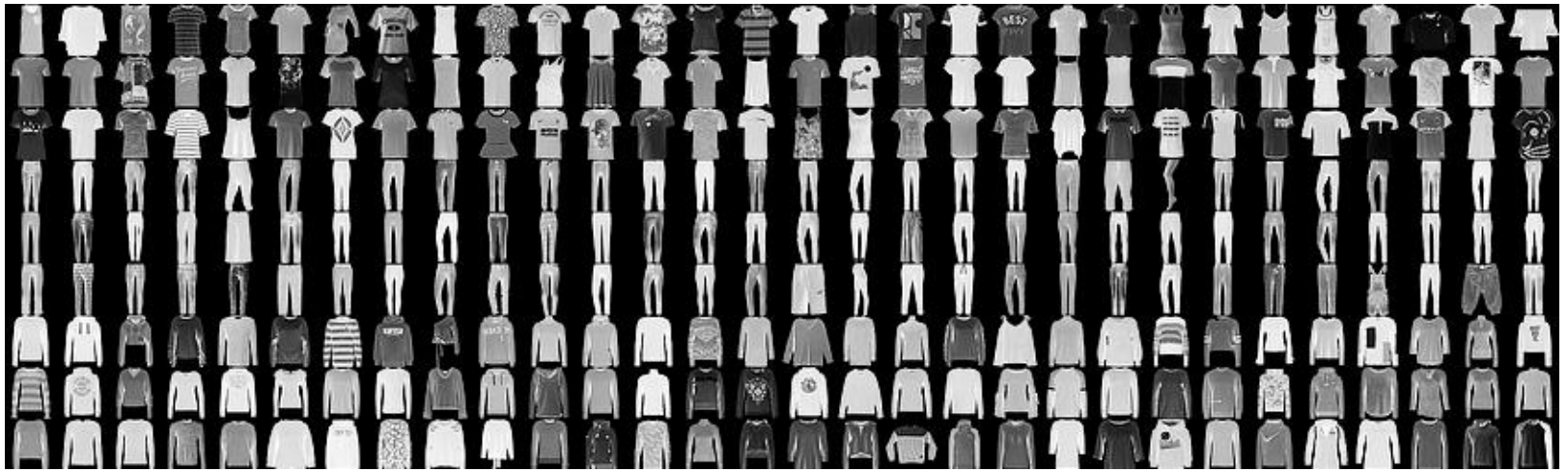
01. Introduction to PyTorch

- **4. 데이터셋과 데이터로더**

  - 데이터셋(dataset) : 데이터 샘플과 이에 해당하는 라벨을 함께 저장하고 호출 시 특정 인덱스의 데이터를 입력 텐서 형태로 전달해주는 역할
  - 데이터로더(dataloader) : 학습시 Iteration마다 데이터셋으로부터 입력 텐서를 불러오는 역할이며, 데이터셋 샘플로의 접근이 용이

※ 예제 데이터셋 : Fashion-MNIST

- Fashion MNIST은 옷, 바지 등 10개 클래스의 의류로 구성된 이미지 데이터셋이다. 이 데이터는 28x28 사이즈의 그레이스케일 이미지이며, 학습 데이터셋은 6만장, 테스트 데이터셋은 1만장으로 구성되어 있다.



01. Introduction to PyTorch

- **4. 데이터셋과 데이터로더**

  - 아래 파라미터를 이용하여 Fashion MNIST 데이터셋 불러오기
  - root : 데이터가 저장되는 경로
  - train : 사용할 데이터셋의 종류 지정 (True로 지정하면 학습 데이터셋, False로 지정하면 테스트 데이터셋)
  - download : 인터넷으로 데이터셋을 다운로드 받을지 여부
  - transform : feature와 label 변환 방법 지정

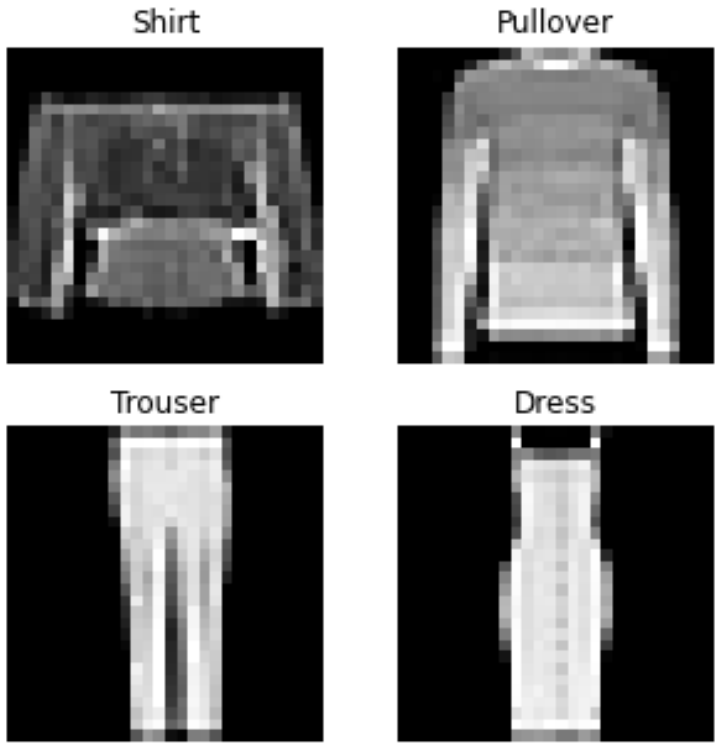
```
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
```

01. Introduction to PyTorch

- **4. 데이터셋과 데이터로더**
  - matplotlib 패키지를 이용하여 Fashion MNIST 데이터셋 시각화

```
import matplotlib.pyplot as plt

figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(training_data), size=(1,)).item()
    img, label = training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```



01. Introduction to PyTorch

- **4. 데이터셋과 데이터로더**

  - 사용자 파일로 부터 커스텀 데이터셋 생성

  - 커스텀 데이터셋 클래스는 반드시 __init__, __len__, __getitem__ 함수를 구현해야 한다.

    1) __init__ : 데이터셋의 전처리를 진행해주는 부분

    2) __len__ : 데이터셋의 길이. 즉, 총 샘플의 수를 적어주는 부분

    3) __getitem__ : 데이터셋에서 특정 인덱스의 샘플을 가져오는 함수

```python
class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = tvio.read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        sample = {"image": image, "label": label}
        return sample
```

01. Introduction to PyTorch

- **4. 데이터셋과 데이터로더**

  • 데이터로더를 사용하여 학습을 위한 데이터 준비

  - 데이터로더를 통해 모델이 훈련하는 동안 미니배치에서 샘플을 통과시키고, 모델의 과적합을 방지하기 위해서 매 훈련 epoch마다 데이터를 변경하여 데이터의 검색 속도를 높이고자 한다.

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

01. Introduction to PyTorch

- **5. 데이터 변환(Transform)**

• 데이터는 머신러닝 알고리즘을 학습하는 데 필요한 최종 형태로 처리되어 항상 제공되지 않는다. Transform을 사용하여 데이터의 일부 조작을 수행하고 모델 학습에 적합하게 만든다.
• 아래 예시에서는 데이터셋을 불러오는 부분에서 사용되는 transform(features)과 target_transform(labels) 파라미터에서 torchvision.transforms 모듈에서 제공되는 ToTensor와 Lambda를 사용하여 데이터 변환 수행
- ToTensor : PIL 이미지 또는 Numpy의 ndarray를 FloatTensor로 변환하고 픽셀 강도 값을 [0, 1] 사이의 값으로 조정
- Lambda : 사용자 정의 lambda 함수를 적용하며, Fashion-MNIST의 10개 label을 원-핫 인코딩 텐서로 변환 수행

```
ds = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
    target_transform=Lambda(lambda y: torch.zeros(10, dtype=torch.float).scatter_(0, torch.tensor(y), value=1))
)
```

01. Introduction to PyTorch

- **6. 신경망 구축**

• 신경망(Neural network)은 데이터 연산을 수행하는 레이어와 모듈로 구성되어 있다. torch.nn 패키지는 신경망을 구축하는데 필요한 모든 구성요소를 제공한다.

• nn.Module은 신경망 모듈로서, 신경망 레이어를 생성하는 __init__과 출력값을 반환하도록 입력값을 연산하는 forward 메서드로 구성되어 있다.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

01. Introduction to PyTorch

- **7. 자동 미분(Automatic differentiation)**

• 신경망을 훈련시킬 때, 가장 많이 사용되는 알고리즘은 역전파(back-propagation)이다. 역전파는 손실 함수(Loss function)의 기울기를 계산하여 파라미터(모델의 가중치)를 조정하게 되는데 PyTorch에서는 torch.autograd로 불리는 빌트인 미분 엔진을 사용한다. 이는 어떠한 연산 그래프에서도 자동 기울기 계산을 지원한다.

• 일반적인 신경망에서 w, b와 같은 매개변수는 손실 함수의 기울기를 계산하여 최적화할 수 있도록 해당 텐서의 requires_grad=True로 설정

```
x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```

•loss.backward()를 수행하여 w와 b의 미분 값을 계산

```
loss.backward()
print(w.grad)
print(b.grad)
```

```
tensor([[0.2739, 0.0490, 0.3279],
        [0.2739, 0.0490, 0.3279],
        [0.2739, 0.0490, 0.3279],
        [0.2739, 0.0490, 0.3279],
        [0.2739, 0.0490, 0.3279]])
tensor([0.2739, 0.0490, 0.3279])
```

01. Introduction to PyTorch

- **8. 최적화 루프(optimization loop)**

  - 이제 모델 및 데이터를 확보했으므로 데이터에 대한 매개 변수를 최적화하여 모델을 학습, 검증 및 테스트해야 합니다.

<모델 학습의 반복 과정>
1) 각 반복에서 모델은 예측값을 출력
2) 예측값(손실)에서 오류를 계산
3) 파라미터와 관련된 오차의 미분값을 계산
4) 기울기 하강법(gradient descent)를 사용하여 파라미터 최적화 수행

```
for batch, (X, y) in enumerate(dataloader):
    # Compute prediction and loss
    pred = model(X)
    loss = loss_fn(pred, y)

    # Backpropagation
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

- optimizer.zero_grad() : 모델 파라미터의 기울기를 0으로 초기화
- loss.backward() : 역전파를 위하여 실제값과 예측값의 오차의 미분 계산
- optimizer.step() : 계산된 기울기 값을 통하여 파라미터 최적화 수행

01. Introduction to PyTorch

- **8. 최적화 루프(optimization loop)**

  - 전체 학습 과정에서의 최적화 루프 구현 코드

  - 하이퍼파라미터 세팅
    - learning_rate = 1e-3
    - batch_size = 64
    - epochs = 10

  - 손실 함수 및 최적화 루프 설정
    - Loss function : Cross Entropy
    - optimizer : Stochastic Gradient Descent

```python
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}₩n-------------------------------")
    train_loop(train_dataloader, model, loss_fn, optimizer)
print("Done!")
```

01. Introduction to PyTorch

- **9. 모델 저장, 불러오기, 예측 수행**

• 모델 저장은 torch.save 메서드를 이용하여 수행. PyTorch 모델에 저장된 학습된 파라미터 state_dict와 저장 경로를 설정하여 모델 저장 수행

```
model = models.vgg16(pretrained=True)
torch.save(model.state_dict(), 'data/model_weights.pth')
```

• load_state_dict() 메서드를 이용하여 모델의 가중치를 불러오기
※ 같은 모델의 인스턴스를 생성한 이후에 가중치 불러와야 함

```
model = models.vgg16() # we do not specify pretrained=True, i.e. do not load default weights
model.load_state_dict(torch.load('data/model_weights.pth'))
```

• 모델 전체 구조와 함께 저장 및 불러오기

```
torch.save(model, 'data/vgg_model.pth')
model = torch.load('data/vgg_model.pth')
```

End of document

Website

http://www.brframe.com/

Contact Point

E-mail admin@brframe.com