PyTorch로 딥러닝 제대로 배우기

- 기초편 -

Part3. 텐서(Tensor)

강사: 김 동 희

목차

I. Tensor

1. Tensor

❑ Tensor의 정의와 특징

- 텐서는 인공지능을 위한 데이터의 특수한 구조로써 배열과 행렬과 매우 유사
- PyTorch에서는 텐서를 사용하여 모델의 입력과 출력 뿐만 아니라 모델의 매개 변수를 인코딩
- 텐서는 GPU나 다른 하드웨어 가속기에서 실행할 수 있다는 점을 제외하고는 NumPy의 ndarrays와 유사
- 텐서와 NumPy 배열은 종종 동일한 기본 메모리를 공유 가능하므로 데이터를 복사할 필요성이 없음
- 텐서는 또한 Auto Diff에 최적화되어 있음

2. Tensor 생성

❑ 직접 생성

- 데이터로부터 직접적으로 텐서 생성이 가능
- 자료형은 자동으로 유추되어 적용
- 단, specific한 자료형이 필요한 경우, 사용자가 정의 가능

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
```

❑ Numpy 배열로부터 생성

- Numpy array로부터 텐서 생성 가능
- 텐서에서 Numpy array로 역변환도 가능
  (Bridge with Numpy 참고)

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

2. Tensor 생성

❑ 다른 텐서로부터 생성

- 새로운 텐서는 명시적으로 재정의되지 않는 한 인수 텐서의 속성(모양, 데이터 유형)을 유지

```
x_ones = torch.ones_like(x_data) # retains the properties of x_data
print(f"Ones Tensor: \n {x_ones} \n")

x_rand = torch.rand_like(x_data, dtype=torch.float) # overrides the datatype of x_data
print(f"Random Tensor: \n {x_rand} \n")
```

```
Ones Tensor:
 tensor([[1, 1],
         [1, 1]])

Random Tensor:
 tensor([[0.7577, 0.3862],
         [0.5864, 0.0733]])
```

2. Tensor 생성

❑ 랜덤하게 또는 상수 변수

- shape은 튜플로 표시된 텐서의 차원 정보

```
shape = (2,3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)

print(f"Random Tensor: \n {rand_tensor} \n")
print(f"Ones Tensor: \n {ones_tensor} \n")
print(f"Zeros Tensor: \n {zeros_tensor}")
```

```
Random Tensor:
 tensor([[0.5535, 0.0687, 0.8074],
        [0.6330, 0.1156, 0.9921]])

Ones Tensor:
 tensor([[1., 1., 1.],
        [1., 1., 1.]])

Zeros Tensor:
 tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

3. Attributes of a Tensor

❑ 텐서 속성은 모양, 데이터 유형 및 저장된 디바이스(CPU or GPU)를 설명

```
tensor = torch.rand(3,4)

print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Device tensor is stored on: {tensor.device}")
```

```
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

4. Tensor 연산

❑ Tensor 연산에서 주의할 점

- 기본적으로, 텐서는 CPU에서 생성
- GPU를 사용하고자 할 경우, 텐서를 GPU로 명시적으로 이동해야 함
- 디바이스에서 큰 텐서를 복사하는 것은 시간과 메모리 측면에서 비효율적
- Tensor의 연산은 같은 디바이스 내에서만 가능

```
# We move our tensor to the GPU if available
if torch.cuda.is_available():
    tensor = tensor.to("cuda")
```

4. Tensor 연산

❑ **Indexing and Slicing**

- Numpy 환경과 비슷하게 데이터를 인덱싱하거나 슬라이싱 할 수 있음

❑ **Joining**

- 다른 두 텐서를 합칠 수 있음

❑ **Arithmetic Operation**

- 텐서를 통해서 산술 연산을 할 수 있음

❑ **Single-element tensors**

- 단일 텐서인 경우, .item() 함수를 활용하여 python 자료형으로 변경 가능

```
agg = tensor.sum()
agg_item = agg.item()
print(agg_item, type(agg_item))
```

```
12.0 <class 'float'>
```

4. Tensor 연산

❑ In-place operations

- In-place: 결과를 피연산자에 저장하는 작업
- Pytorch에서 접미사 _로 표시된 함수들은 inplace 작업을 수행. 예: x.copy_(y), x.t_()

```
print(f"{tensor} \n")
tensor.add_(5)
print(tensor)
```

```
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])

tensor([[6., 5., 6., 6.],
        [6., 5., 6., 6.],
        [6., 5., 6., 6.],
        [6., 5., 6., 6.]])
```

5. Bridge with Numpy

❑ Tensor to Numpy array

```
t = torch.ones(5)
print(f"t: {t}")
n = t.numpy()
print(f"n: {n}")
```

```
t: tensor([1., 1., 1., 1., 1.])
n: [1. 1. 1. 1. 1.]
```

- 주의할 점은 텐서가 numpy array에 영향을 줄 수 있음

```
t.add_(1)
print(f"t: {t}")
print(f"n: {n}")
```

```
t: tensor([2., 2., 2., 2., 2.])
n: [2. 2. 2. 2. 2.]
```

5. Bridge with Numpy

❑ Numpy array to Tensor

```
n = np.ones(5)
t = torch.from_numpy(n)
```

- 반대 경우에도 성립

```
np.add(n, 1, out=n)
print(f"t: {t}")
print(f"n: {n}")
```

```
t: tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
n: [2. 2. 2. 2. 2.]
```

감사합니다.