파이토치에 앞서 알아보기

학습 목표

- 텐서를 만들어봅니다. (스칼라~4차텐서)
- torch에서 사용하는 기본 함수에 대해 알아봅니다.
  - item(), view(), max() 등

목차

01. 텐서와 기본함수에 대해 알아보기

목차로 이동하기

- 파이토치는 텐서(Tensor)라는 고유 클래스로 데이터를 표현.
- 파이토치늬 연산 대상 데이터는 모두 '텐서(Tensor)'이라는 파이토치의 고유한 형식으로 되어 있어야 한다.

기본 함수 알아보기

- view() 함수 - 텐서의 계수(rank)를 변환 - (reshape)의 역할.
- item() 함수 - 손실을 계산한 후, 텐서값을 가져올 때 사용.
- max() 함수 - 다중 분류의 예측 결과로부터 예측 라벨 값을 가져오는 데 사용.

02. 0차원~4차원 텐서 만들기

목차로 이동하기

In [6]:
```
import torch
import numpy as np

print(torch.__version__)
print(np.__version__)
```

```
2.3.1+cu121
1.25.2
```

0차원 텐서(스칼라) 만들기

In [7]:
```
# 0차원 텐서(스칼라)
r0 = torch.tensor(5.0).float()
```

```
# type 확인
print(type(r0))

# dtype 확인
print(r0.dtype)
```

```
<class 'torch.Tensor'>
torch.float32
```

shape과 데이터 확인

In [8]:
```
# dimension 확인
print(r0.ndim)

# shape과 데이터 내용 확인
print(r0.shape)
print(r0.data)
```

```
0
torch.Size([])
tensor(5.)
```

- [텐서].ndim을 이용하여 현재 차원을 확인 가능하다.
- 텐서도 shape이라는 속성을 갖는다.
- 0차원 텐서는 torch.Size([])와 같이 표기된다.
- 순수하게 수치를 얻고 싶을 때는 data 속성을 이용하면 된다.

1차원 텐서(벡터) 만들기

In [9]:
```
np_1 = np.array([1,2,3,4,5])
print(np_1.shape)
```

```
(5,)
```

In [10]:
```
# 넘파이 배열을 이용하여 텐서 만들기
tensor01 = torch.tensor(np_1).float()

# 자료형, shape, 데이터 확인
print(tensor01.ndim)
print(tensor01.dtype)
print(tensor01.shape)
print(tensor01.data)
```

```
1
torch.float32
torch.Size([5])
tensor([1., 2., 3., 4., 5.])
```

2차원 텐서(행렬) 만들기

In [11]:
```
np_2 = np.array([ [1,5,6],
                  [14,13,12] ])

# 넘파이 shape 확인
print(np_2.shape)
```

```
# 넘파이에서 텐서로 변환
tensor02 = torch.tensor(np_2).float()

# 자료형, shape, 데이터 확인
tensor02 = torch.tensor(np_2).float()

print(tensor02.ndim)
print(tensor02.shape)
print(tensor02.data)
```

```
(2, 3)
2
torch.Size([2, 3])
tensor([[ 1.,  5.,  6.],
        [14., 13., 12.]])
```

3차원 텐서 만들기

In [12]:
```
torch.manual_seed(123)

# shape=[3,2,2]의 정규 분포 텐서 작성
tensor03 = torch.randn(  (3, 2, 2))

# shape, 데이터 확인
print(tensor03.ndim)
print(tensor03.shape, tensor03.data)
```

```
3
torch.Size([3, 2, 2]) tensor([[[-0.1115,  0.1204],
         [-0.3696, -0.2404]],

        [[-1.1969,  0.2093],
         [-0.9724, -0.7550]],

        [[ 0.3239, -0.1085],
         [ 0.2103, -0.3908]]])
```

4차원 텐서 만들기

In [13]:
```
tensor04 = torch.ones( (2, 3, 2, 2) )

# 자료형, 데이터 확인
print(tensor04.ndim)
print(tensor04.shape, tensor04.data)
```

```
4
torch.Size([2, 3, 2, 2]) tensor([[[[1., 1.],
          [1., 1.]],

         [[1., 1.],
          [1., 1.]],

         [[1., 1.],
          [1., 1.]]],


        [[[1., 1.],
          [1., 1.]],

         [[1., 1.],
          [1., 1.]],

         [[1., 1.],
          [1., 1.]]]])
```

- numpy 배열에서 텐서를 만드는 것보다 torch.randn(), torch.ones()함수를 이용하는 것이 더 쉽다.

03. 기본텐서 및 기본 함수

목차로 이동하기

- 파이토치의 대부분의 계산은 부동소수점 수치형(dtype=float32)을 사용.
  - 한가지 예외, '다중 분류'에서 사용하는 손실함수인 nn.CrossEntropyLoss와 nn.NLLoss는 손실함수 호출시에, 두번째 인수로 정수 타입 지정을 해야함.

In [14]:
```
tensor01.dtype
```

Out[14]:
```
torch.float32
```

In [15]:
```
tensor_int = tensor01.long()

# dtype, 값 확인
print(tensor_int.dtype)
print(tensor_int)
```

```
torch.int64
tensor([1, 2, 3, 4, 5])
```

view() 함수

- 데이터는 같지만 모양이 다른 새로운 텐서를 만들기

In [16]:
```
print(tensor03.shape)
```

```
torch.Size([3, 2, 2])
```

2차원 텐서로 변환

In [17]:
```
ts_02 = tensor03.view(3, -1)

print(ts_02.shape)
print(ts_02.data)
```

```
torch.Size([3, 4])
tensor([[-0.1115,  0.1204, -0.3696, -0.2404],
        [-1.1969,  0.2093, -0.9724, -0.7550],
        [ 0.3239, -0.1085,  0.2103, -0.3908]])
```

- tensor03.view(3, -1)의 인수의 값 중에 하나가 -1이다.-1이 넘어가면 나머지 인숫값들을 크기를 맞추고 -1의 부분은 자동으로 맞춰진다.

실습 - tensor03을 1차원 텐서로 변경해 보자.

In [18]:
```
ts_01 = tensor03.view(-1)

print(ts_01.shape)
print(ts_01.data)
```

```
torch.Size([12])
tensor([-0.1115,  0.1204, -0.3696, -0.2404, -1.1969,  0.2093, -0.9724, -0.7550,
         0.3239, -0.1085,  0.2103, -0.3908])
```

item() 함수

- 딥러닝 코드에서 텐서로 이루어진 loss 계산 결과에서 데이터 기록을 위한 값 추출에 자주 사용.

In [19]:
```
r0
```

Out[19]:
```
tensor(5.)
```

In [20]:
```
item = r0.item()

print( type(item))
print( item)
```

```
<class 'float'>
5.0
```

- 주의 : 1차원 텐서이상에서 item()은 사용을 하지않는다.

In [21]:
```
tensor01
```

Out[21]:
```
tensor([1., 2., 3., 4., 5.])
```

In [22]:
```
tensor01.item()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-22-e96a45902892> in <cell line: 1>()
----> 1 tensor01.item()

RuntimeError: a Tensor with 5 elements cannot be converted to Scalar
```

- 에러 발생.

max() 함수

- 텐서 클래스는 수치 연산도 가능하다.
- 최대값을 가져오는 max() 함수도 확인해 본다.

In [23]:
```
tensor02
```

Out[23]:
```
tensor([[ 1.,  5.,  6.],
        [14., 13., 12.]])
```

In [24]:
```
# 텐서 tensor02
print(tensor02)
```

```
tensor([[ 1.,  5.,  6.],
        [14., 13., 12.]])
```

- max 함수를 인수 없이 호출 시, 최댓값을 얻을 수 있음.

In [25]:
```
print(torch.max(tensor02, 1))
```

```
torch.return_types.max(
values=tensor([ 6., 14.]),
indices=tensor([2, 0]))
```

- torch.max(tensor02, 1) 두번째 인수 1은 행방향의 집계, 0은 열방향의 집계를 의미

In [26]:
```
print(torch.max(tensor02, 0))
```

```
torch.return_types.max(
values=tensor([14., 13., 12.]),
indices=tensor([1, 1, 1]))
```

In [27]:
```
torch.max(tensor02, 1)
```

Out[27]:
```
torch.return_types.max(
values=tensor([ 6., 14.]),
indices=tensor([2, 0]))
```

- 위의 코드에 [1]을 붙이면 indices만을 가져올 수 있음.

In [28]:
```
print(torch.max(tensor02, 1))
```

```
torch.return_types.max(
values=tensor([ 6., 14.]),
indices=tensor([2, 0]))
```

In [29]:
```
print( torch.max(tensor02, 1)[0] )   # 값
print( torch.max(tensor02, 1)[1] )   # 최대값이 위치하는 인덱스
```

```
tensor([ 6., 14.])
tensor([2, 0])
```

텐서를 numpy 로 변수로 바꾸기

In [30]:
```
ts2_np = tensor02.data.numpy()

print( type(ts2_np))

# 값 확인
print(ts2_np)
```

```
<class 'numpy.ndarray'>
[[ 1.  5.  6.]
 [14. 13. 12.]]
```