PyTorch로 딥러닝 제대로 배우기

- 기초편 -

Part3. 텐서(Tensor)

강사: 김 동 희

목차

II. Operation on Tensors

1. Tensor 연산

❑ is_tensor(input)

- 텐서 여부를 확인하는 함수

```
>>> x=torch.tensor([1,2,3])
>>> torch.is_tensor(x)
True
```

❑ is_nonzero(input)

- Zero 여부 확인

```
>>> torch.is_nonzero(torch.tensor([0.]))
False
>>> torch.is_nonzero(torch.tensor([1.5]))
True
>>> torch.is_nonzero(torch.tensor([False]))
False
>>> torch.is_nonzero(torch.tensor([3]))
True
>>> torch.is_nonzero(torch.tensor([1, 3, 5]))
Traceback (most recent call last):
...
RuntimeError: bool value of Tensor with more than one value is ambiguous
>>> torch.is_nonzero(torch.tensor([]))
Traceback (most recent call last):
...
RuntimeError: bool value of Tensor with no values is ambiguous
```

2. Creation Operation

❑ **tensor**

- 텐서를 생성하는 함수

```
>>> torch.tensor([[0.1, 1.2], [2.2, 3.1], [4.9, 5.2]])
tensor([[ 0.1000,  1.2000],
        [ 2.2000,  3.1000],
        [ 4.9000,  5.2000]])
```

❑ **from_numpy**

- 텐서를 생성하는 함수

```
>>> a = numpy.array([1, 2, 3])
>>> t = torch.from_numpy(a)
>>> t
tensor([ 1,  2,  3])
```

❑ **zeros**

- 텐서를 생성하는 함수

```
>>> torch.zeros(2, 3)
tensor([[ 0.,  0.,  0.],
        [ 0.,  0.,  0.]])
```

❑ **ones**

- 텐서를 생성하는 함수

```
>>> torch.ones(2, 3)
tensor([[ 1.,  1.,  1.],
        [ 1.,  1.,  1.]])
```

3. Indexing, Slicing, Joining, Mutating Ops

❑ Indexing

```
>>> x = torch.randn(3, 4)
>>> x
tensor([[ 0.1427,  0.0231, -0.5414, -1.0009],
        [-0.4664,  0.2647, -0.1228, -1.1068],
        [-1.1734, -0.6571,  0.7230, -0.6004]])
>>> indices = torch.tensor([0, 2])
>>> torch.index_select(x, 0, indices)
tensor([[ 0.1427,  0.0231, -0.5414, -1.0009],
        [-1.1734, -0.6571,  0.7230, -0.6004]])
>>> torch.index_select(x, 1, indices)
tensor([[ 0.1427, -0.5414],
        [-0.4664, -0.1228],
        [-1.1734,  0.7230]])
```

3. Indexing, Slicing, Joining, Mutating Ops

❑ Slicing

```
tensor = torch.ones(4, 4)
print(f"First row: {tensor[0]}")
print(f"First column: {tensor[:, 0]}")
print(f"Last column: {tensor[..., -1]}")
tensor[:,1] = 0
print(tensor)
```

```
First row: tensor([1., 1., 1., 1.])
First column: tensor([1., 1., 1., 1.])
Last column: tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

3. Indexing, Slicing, Joining, Mutating Ops

❑ Joining

```
>>> x = torch.randn(2, 3)
>>> x
tensor([[ 0.6580, -1.0969, -0.4614],
        [-0.1034, -0.5790,  0.1497]])
>>> torch.cat((x, x, x), 0)
tensor([[ 0.6580, -1.0969, -0.4614],
        [-0.1034, -0.5790,  0.1497],
        [ 0.6580, -1.0969, -0.4614],
        [-0.1034, -0.5790,  0.1497],
        [ 0.6580, -1.0969, -0.4614],
        [-0.1034, -0.5790,  0.1497]])
>>> torch.cat((x, x, x), 1)
tensor([[ 0.6580, -1.0969, -0.4614,  0.6580, -1.0969, -0.4614,  0.6580,
         -1.0969, -0.4614],
        [-0.1034, -0.5790,  0.1497, -0.1034, -0.5790,  0.1497, -0.1034,
         -0.5790,  0.1497]])
```

3. Indexing, Slicing, Joining, Mutating Ops

❑ Mutating - permute

```
>>> x = torch.randn(2, 3, 5)
>>> x.size()
torch.Size([2, 3, 5])
>>> torch.permute(x, (2, 0, 1)).size()
torch.Size([5, 2, 3])
```

❑ Mutating - reshape

```
>>> a = torch.arange(4.)
>>> torch.reshape(a, (2, 2))
tensor([[ 0.,  1.],
        [ 2.,  3.]])
>>> b = torch.tensor([[0, 1], [2, 3]])
>>> torch.reshape(b, (-1,))
tensor([ 0,  1,  2,  3])
```

3. Indexing, Slicing, Joining, Mutating Ops

❑ Mutating - Transpose

```
>>> x = torch.randn(2, 3)
>>> x
tensor([[ 1.0028, -0.9893,  0.5809],
        [-0.1669,  0.7299,  0.4942]])
>>> torch.transpose(x, 0, 1)
tensor([[ 1.0028, -0.1669],
        [-0.9893,  0.7299],
        [ 0.5809,  0.4942]])
```

❑ Mutating - Squeeze

```
>>> x = torch.zeros(2, 1, 2, 1, 2)
>>> x.size()
torch.Size([2, 1, 2, 1, 2])
>>> y = torch.squeeze(x)
>>> y.size()
torch.Size([2, 2, 2])
>>> y = torch.squeeze(x, 0)
>>> y.size()
torch.Size([2, 1, 2, 1, 2])
>>> y = torch.squeeze(x, 1)
>>> y.size()
torch.Size([2, 2, 1, 2])
```

3. Random sampling

❑ **rand**

```
>>> torch.rand(4)
tensor([ 0.5204,  0.2503,  0.3525,  0.5673])
>>> torch.rand(2, 3)
tensor([[ 0.8237,  0.5781,  0.6879],
        [ 0.3816,  0.7249,  0.0998]])
```

4. Math operations

❑ Arithmetic operation

- 덧셈, 뺄셈, 곱셈, 나눗셈 등 기본 수학 연산 기능 제공

```
# This computes the matrix multiplication between two tensors. y1, y2, y3 will have the same value
y1 = tensor @ tensor.T
y2 = tensor.matmul(tensor.T)

y3 = torch.rand_like(y1)
torch.matmul(tensor, tensor.T, out=y3)


# This computes the element-wise product. z1, z2, z3 will have the same value
z1 = tensor * tensor
z2 = tensor.mul(tensor)

z3 = torch.rand_like(tensor)
torch.mul(tensor, tensor, out=z3)
```

4. Math operations

❑ Pointwise Ops

- abs

```
>>> torch.abs(torch.tensor([-1, -2, 3]))
tensor([ 1,  2,  3])
```

- add

```
>>> a = torch.randn(4)
>>> a
tensor([ 0.0202,  1.0985,  1.3506, -0.6056])
>>> torch.add(a, 20)
tensor([ 20.0202,  21.0985,  21.3506,  19.3944])
```

- cos

```
>>> a = torch.randn(4)
>>> a
tensor([ 1.4309,  1.2706, -0.8562,  0.9796])
>>> torch.cos(a)
tensor([ 0.1395,  0.2957,  0.6553,  0.5574])
```

- pow

```
>>> a = torch.randn(4)
>>> a
tensor([ 0.4331,  1.2475,  0.6834, -0.2791])
>>> torch.pow(a, 2)
tensor([ 0.1875,  1.5561,  0.4670,  0.0779])
```

4. Math operations

❑ Reduction Ops

- argmax

```
>>> a = torch.randn(4, 4)
>>> a
tensor([[ 1.3398,  0.2663, -0.2686,  0.2450],
        [-0.7401, -0.8805, -0.3402, -1.1936],
        [ 0.4907, -1.3948, -1.0691, -0.3132],
        [-1.6092,  0.5419, -0.2993,  0.3195]])
>>> torch.argmax(a)
tensor(0)
```

- argmin

```
>>> a = torch.randn(4, 4)
>>> a
tensor([[ 0.1139,  0.2254, -0.1381,  0.3687],
        [ 1.0100, -1.1975, -0.0102, -0.4732],
        [-0.9240,  0.1207, -0.7506, -1.0213],
        [ 1.7809, -1.2960,  0.9384,  0.1438]])
>>> torch.argmin(a)
tensor(13)
>>> torch.argmin(a, dim=1)
tensor([ 2,  1,  3,  1])
>>> torch.argmin(a, dim=1, keepdim=True)
tensor([[2],
        [1],
        [3],
        [1]])
```

4. Math operations

❑ Reduction Ops

- sum

```
>>> a = torch.randn(1, 3)
>>> a
tensor([[ 0.1133, -0.9567,  0.2958]])
>>> torch.sum(a)
tensor(-0.5475)
```

- unique

```
>>> output = torch.unique(torch.tensor([1, 3, 2, 3], dtype=torch.long))
>>> output
tensor([ 2,  3,  1])
```

4. Math operations

❑ Comparison Ops

- eq

```
>>> torch.eq(torch.tensor([[1, 2], [3, 4]]), torch.tensor([[1, 1], [4, 4]]))
tensor([[ True, False],
        [False, True]])
```

- equal

```
>>> torch.equal(torch.tensor([1, 2]), torch.tensor([1, 2]))
True
```

4. Math operations

❑ Comparison Ops

- isinf

```
>>> torch.isinf(torch.tensor([1, float('inf'), 2, float('-inf'), float('nan')]))
tensor([False,  True,  False,  True,  False])
```

- isnan

```
>>> torch.isnan(torch.tensor([1, float('nan'), 2]))
tensor([False, True, False])
```

5. Disabling gradient computation

```
>>> x = torch.zeros(1, requires_grad=True)
>>> with torch.no_grad():
...     y = x * 2
>>> y.requires_grad
False

>>> is_train = False
>>> with torch.set_grad_enabled(is_train):
...     y = x * 2
>>> y.requires_grad
False

>>> torch.set_grad_enabled(True)  # this can also be used as a function
>>> y = x * 2
>>> y.requires_grad
True

>>> torch.set_grad_enabled(False)
>>> y = x * 2
>>> y.requires_grad
False
```

감사합니다.