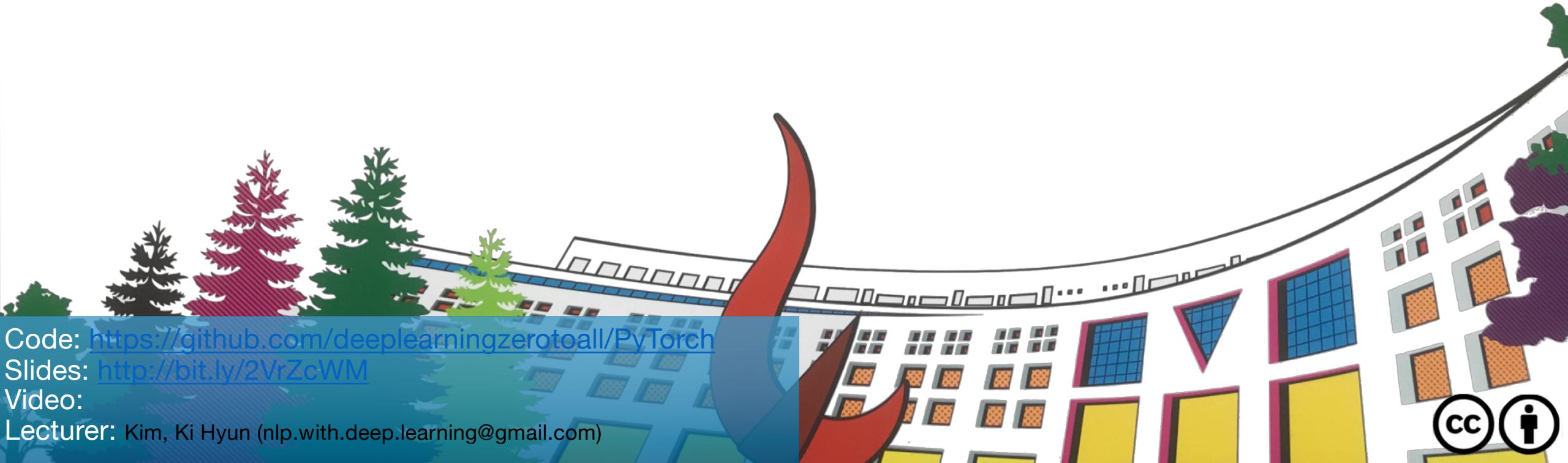


# ML/DL for Everyone Season2

with **PYTORCH**

## PyTorch Basic Tensor Manipulation I



Code: <https://github.com/deeplearningzerotoall/PyTorch>

Slides: <http://bit.ly/2VrZcWM>

Video:

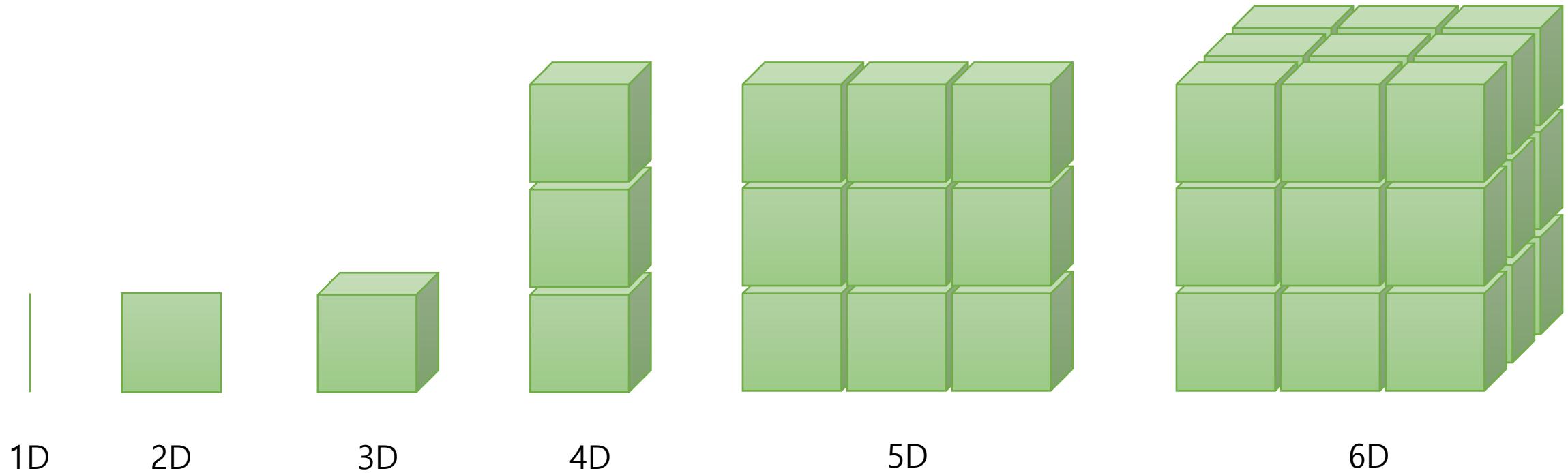
Lecturer: Kim, Ki Hyun (nlp.with.deep.learning@gmail.com)



# PyTorch Basic Tensor Manipulation

- Vector, Matrix and Tensor
- NumPy Review
- PyTorch Tensor Allocation
- Matrix Multiplication
- Other Basic Ops

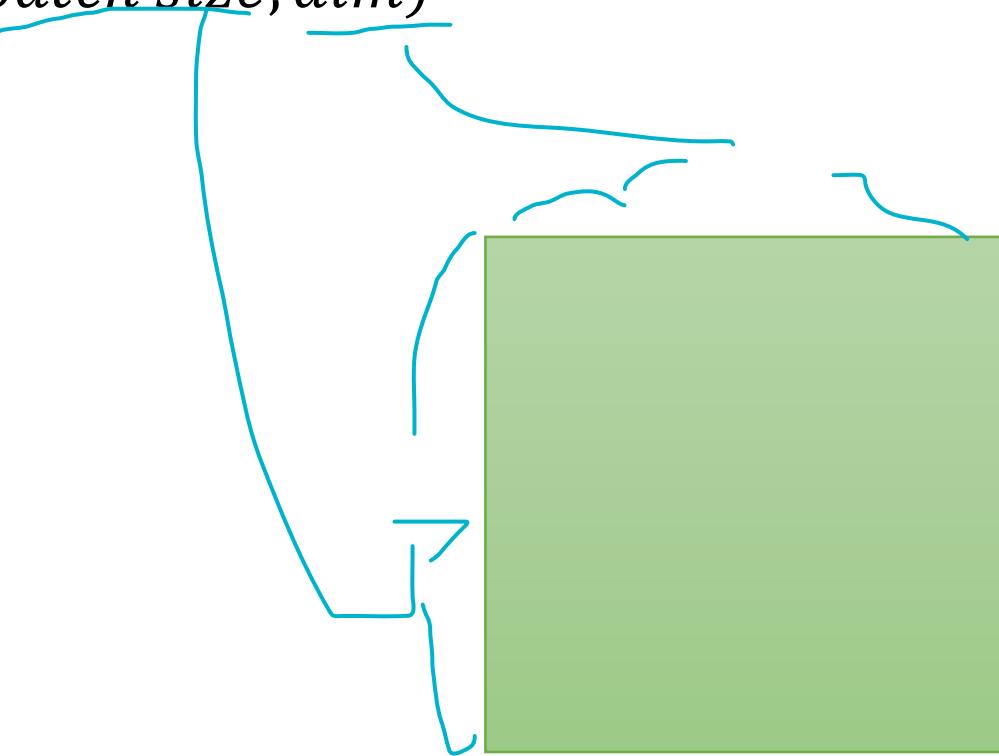
# Vector, Matrix and Tensor



# PyTorch Tensor Shape Convention

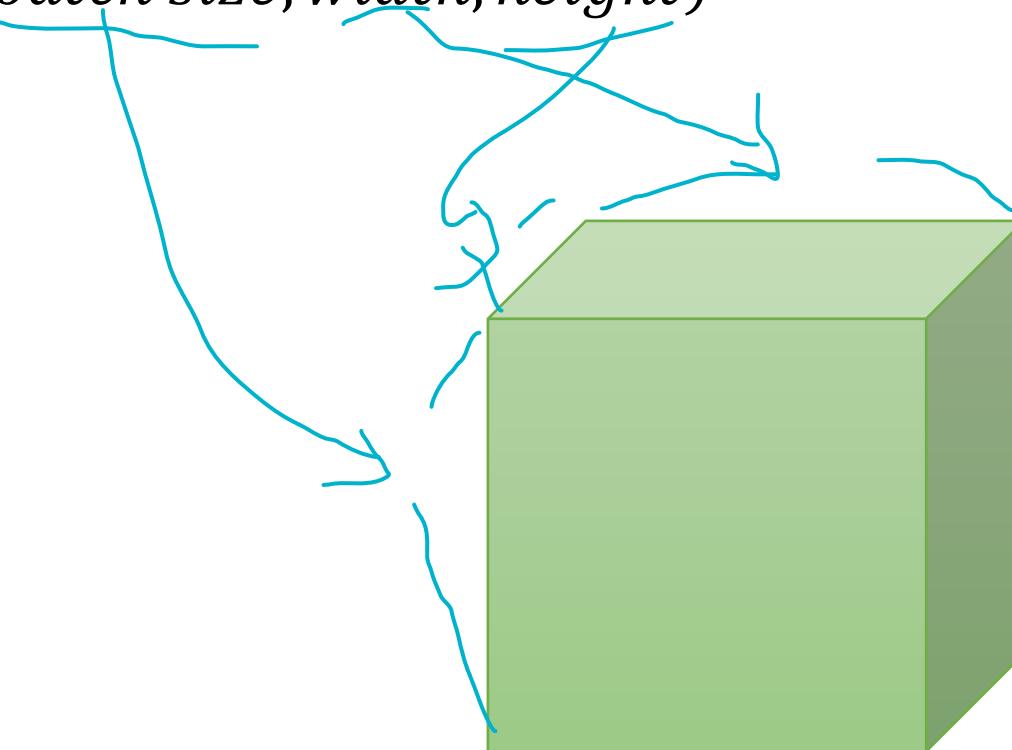
- 2D Tensor (Typical Simple Setting)

- $|t| = (\text{batch size}, \text{dim})$



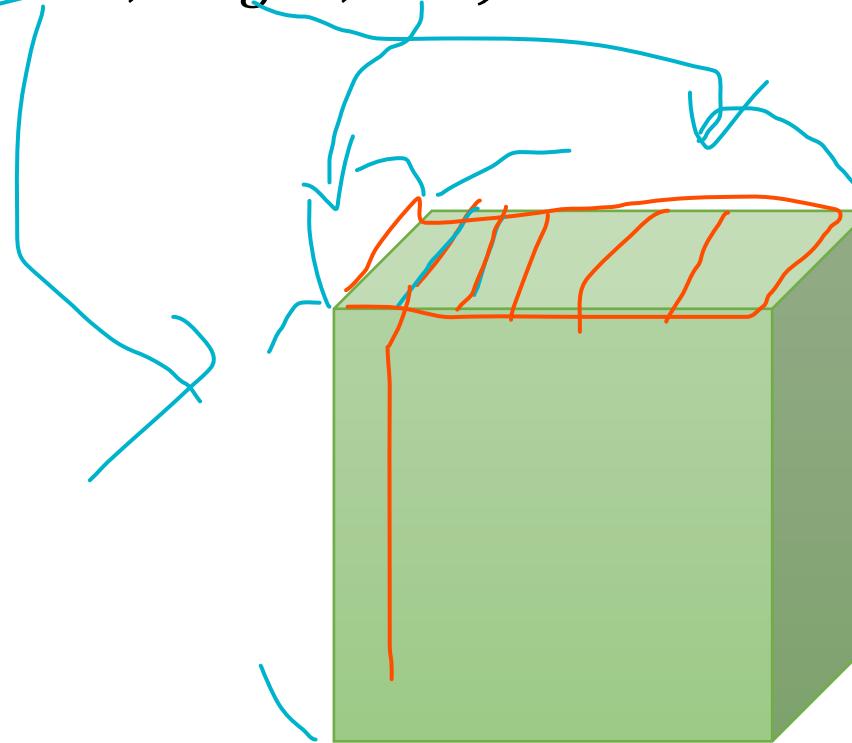
# PyTorch Tensor Shape Convention

- 3D Tensor (Typical Computer Vision)
  - $|t| = (\text{batch size}, \text{width}, \text{height})$



# PyTorch Tensor Shape Convention

- 3D Tensor (Typical Natural Language Processing)
  - $|t| = (\underline{batch\ size}, \underline{length}, \underline{dim})$



# Import

Run `pip install -r requirements.txt` in terminal to install all required Python packages.

```
import numpy as np
import torch
```

# NumPy Review

## 1D Array with NumPy

```
t = np.array([0., 1., 2., 3., 4., 5., 6.])
print(t)
```

```
[ 0.  1.  2.  3.  4.  5.  6.]
```

```
print('Rank of t: ', t.ndim)
print('Shape of t: ', t.shape)
```

```
Rank of t: 1
```

```
Shape of t: (7,)
```

```
print('t[0] t[1] t[-1] = ', t[0], t[1], t[-1]) # Element
print('t[2:5] t[4:-1] = ', t[2:5], t[4:-1])    # Slicing
print('t[:2] t[3:]     = ', t[:2], t[3:])       # Slicing
```

```
t[0] t[1] t[-1] =  0.0 1.0 6.0
```

```
t[2:5] t[4:-1] = [ 2.  3.  4.] [ 4.  5.]
```

```
t[:2] t[3:]     = [ 0.  1.] [ 3.  4.  5.  6.]
```

# NumPy Review

## 2D Array with NumPy

```
t = np.array([[1., 2., 3.], [4., 5., 6.], [7., 8., 9.], [10., 11., 12.]])  
print(t)
```

```
[[ 1.  2.  3.]  
 [ 4.  5.  6.]  
 [ 7.  8.  9.]  
 [10. 11. 12.]]
```

```
print('Rank of t: ', t.ndim)  
print('Shape of t: ', t.shape)
```

```
Rank of t: 2  
Shape of t: (4, 3)
```

# PyTorch Tensor

## 1D Array with PyTorch

```
t = torch.FloatTensor([0., 1., 2., 3., 4., 5., 6.])
print(t)
```

```
tensor([0., 1., 2., 3., 4., 5., 6.])
```

```
print(t.dim()) # rank
print(t.shape) # shape
print(t.size()) # shape
print(t[0], t[1], t[-1]) # Element
print(t[2:5], t[4:-1]) # Slicing
print(t[:2], t[3:]) # Slicing
```

```
1
torch.Size([7])
torch.Size([7])
tensor(0.) tensor(1.) tensor(6.)
tensor([2., 3., 4.]) tensor([4., 5.])
tensor([0., 1.]) tensor([3., 4., 5., 6.])
```

# PyTorch Tensor

## 2D Array with PyTorch

```
t = torch.FloatTensor([[1., 2., 3.],
                      [4., 5., 6.],
                      [7., 8., 9.],
                      [10., 11., 12.]])
print(t)
```

```
tensor([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.],
        [10., 11., 12.]])
```

```
print(t.dim()) # rank
print(t.size()) # shape
print(t[:, 1])
print(t[:, 1].size())
print(t[:, :-1])
```

```
2
torch.Size([4, 3])
tensor([ 2.,  5.,  8., 11.])
torch.Size([4])
tensor([[ 1.,  2.],
        [ 4.,  5.],
        [ 7.,  8.],
        [10., 11.]])
```

# Broadcasting

```
# Same shape
m1 = torch.FloatTensor([[3, 3]])
m2 = torch.FloatTensor([[2, 2]])
print(m1 + m2)
```

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

```
# Vector + scalar
m1 = torch.FloatTensor([[1, 2]])
m2 = torch.FloatTensor([3]) # 3 -> [[3, 3]]
print(m1 + m2)
```

```
tensor([[4., 5.]])
```

```
# 2 x 1 Vector + 1 x 2 Vector
m1 = torch.FloatTensor([[1, 2]])
m2 = torch.FloatTensor([[3], [4]])
print(m1 + m2)
```

```
tensor([[4., 5.],
       [5., 6.]])
```

# Multiplication vs Matrix Multiplication

```
print()
print('-----')
print('Mul vs Matmul')
print('-----')
m1 = torch.FloatTensor([[1, 2], [3, 4]])
m2 = torch.FloatTensor([[1], [2]])
print('Shape of Matrix 1: ', m1.shape) # 2 x 2
print('Shape of Matrix 2: ', m2.shape) # 2 x 1
print(m1.matmul(m2)) # 2 x 1

m1 = torch.FloatTensor([[1, 2], [3, 4]])
m2 = torch.FloatTensor([[1], [2]])
print('Shape of Matrix 1: ', m1.shape) # 2 x 2
print('Shape of Matrix 2: ', m2.shape) # 2 x 1
print(m1 * m2) # 2 x 2
print(m1.mul(m2))
```

```
-----
Mul vs Matmul
-----
Shape of Matrix 1:  torch.Size([2, 2])
Shape of Matrix 2:  torch.Size([2, 1])
tensor([[ 5.],
       [11.]])
Shape of Matrix 1:  torch.Size([2, 2])
Shape of Matrix 2:  torch.Size([2, 1])
tensor([[1., 2.],
       [6., 8.]])
tensor([[1., 2.],
       [6., 8.]])
```

# Mean

```
t = torch.FloatTensor([1, 2])
print(t.mean())

tensor(1.5000)
```

```
# Can't use mean() on integers
t = torch.LongTensor([1, 2])
try:
    print(t.mean())
except Exception as exc:
    print(exc)
```

Can only calculate the mean of floating types. Got Long instead.

You can also use `t.mean` for higher rank tensors to get mean of all elements, or mean by particular dimension.

```
t = torch.FloatTensor([[1, 2], [3, 4]])
print(t)

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

```
print(t.mean())
print(t.mean(dim=0))
print(t.mean(dim=1))
print(t.mean(dim=-1))
```

```
tensor(2.5000)
tensor([2., 3.])
tensor([1.5000, 3.5000])
tensor([1.5000, 3.5000])
```

# Sum

```
t = torch.FloatTensor([[1, 2], [3, 4]])  
print(t)
```

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

```
print(t.sum())  
print(t.sum(dim=0))  
print(t.sum(dim=1))  
print(t.sum(dim=-1))
```

```
tensor(10.)  
tensor([4., 6.])  
tensor([3., 7.])  
tensor([3., 7.])
```

# Max and Argmax

```
t = torch.FloatTensor([[1, 2], [3, 4]])
print(t)

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

The `max` operator returns one value if it is called without an argument.

```
print(t.max()) # Returns one value: max

tensor(4.)
```

The `max` operator returns 2 values when called with dimension specified. The first value is the maximum value, and the second value is the argmax: the index of the element with maximum value.

```
print(t.max(dim=0)) # Returns two values: max and argmax
print('Max: ', t.max(dim=0)[0])
print('Argmax: ', t.max(dim=0)[1])

(tensor([3., 4.]), tensor([1, 1]))
Max:  tensor([3., 4.])
Argmax:  tensor([1, 1])
```

```
print(t.max(dim=1))
print(t.max(dim=-1))

(tensor([2., 4.]), tensor([1, 1]))
(tensor([2., 4.]), tensor([1, 1]))
```