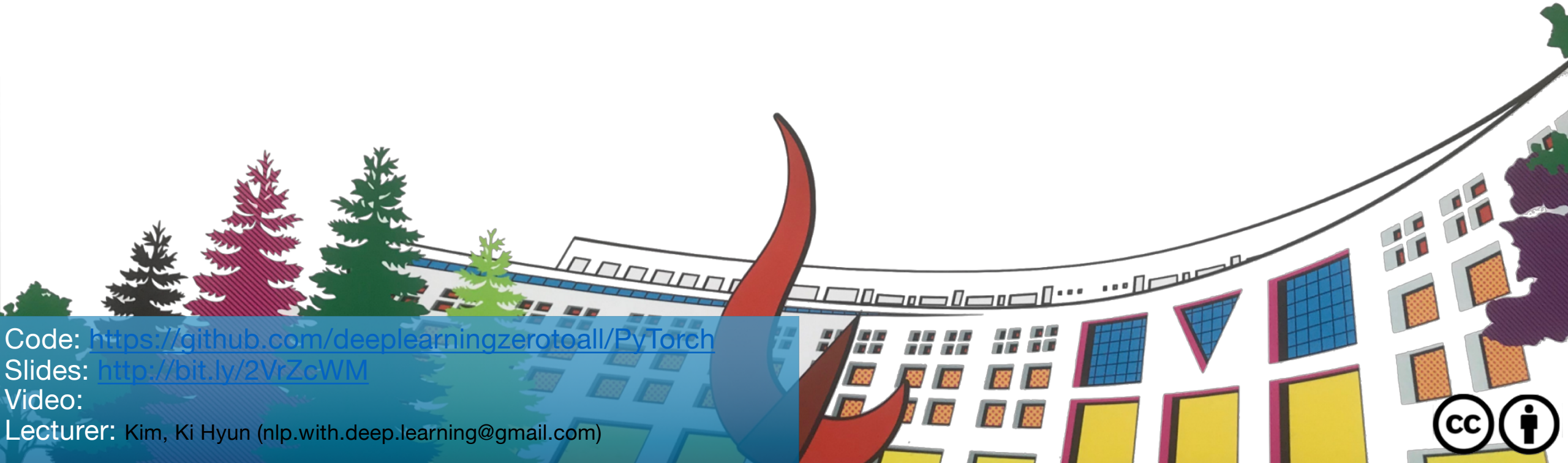
ML/DL for Everyone Season2

with PYTORCH

PyTorch Basic Tensor Manipulation I

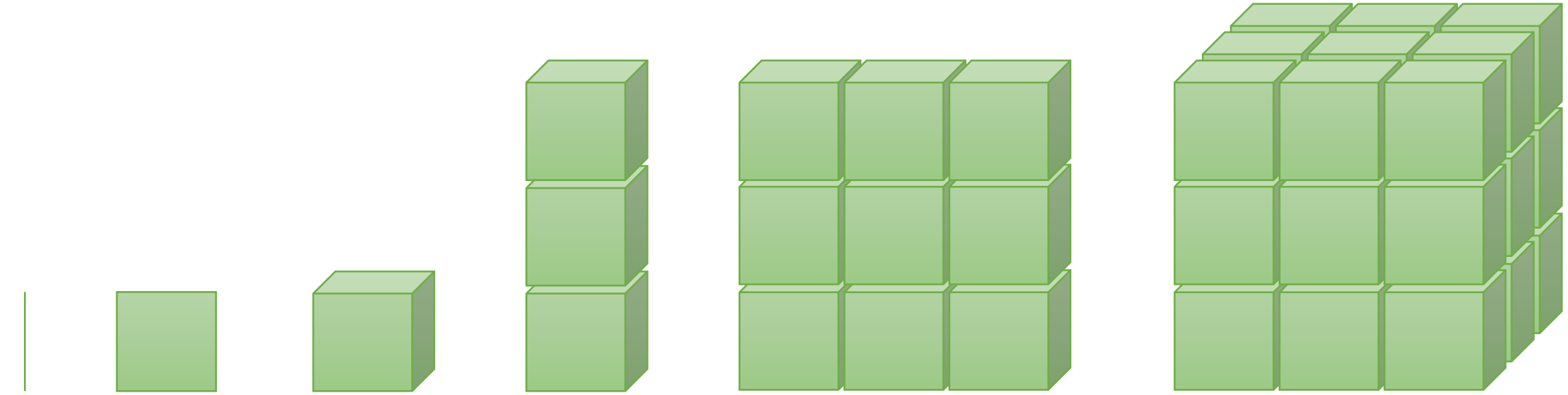


Code: https://github.com/deeplearningzerotoall/PyTorch
Slides: http://bit.ly/2VrZcWM
Video:
Lecturer: Kim, Ki Hyun (nlp.with.deep.learning@gmail.com)

PyTorch Basic Tensor Manipulation

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

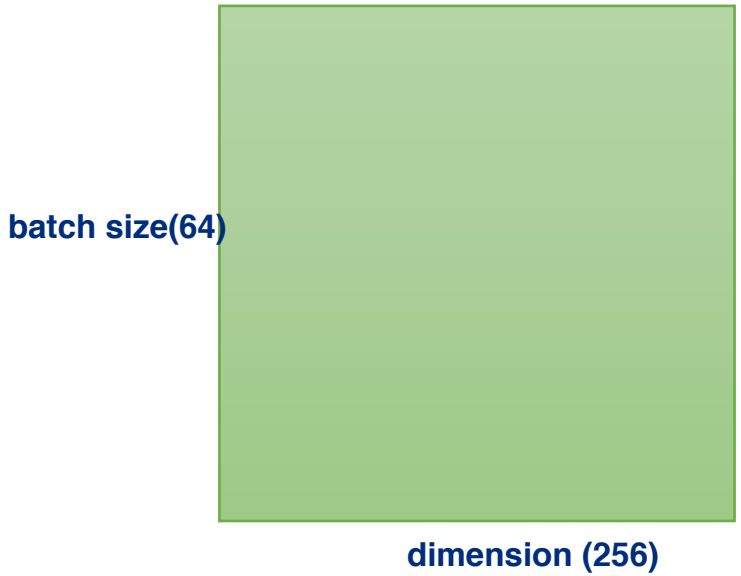
Vector, Matrix and Tensor



PyTorch Tensor Shape Convention

- 2D Tensor (Typical Simple Setting)
  - $|t| = (batch\ size, dim)$

"크기를 어떻게 구하는가"

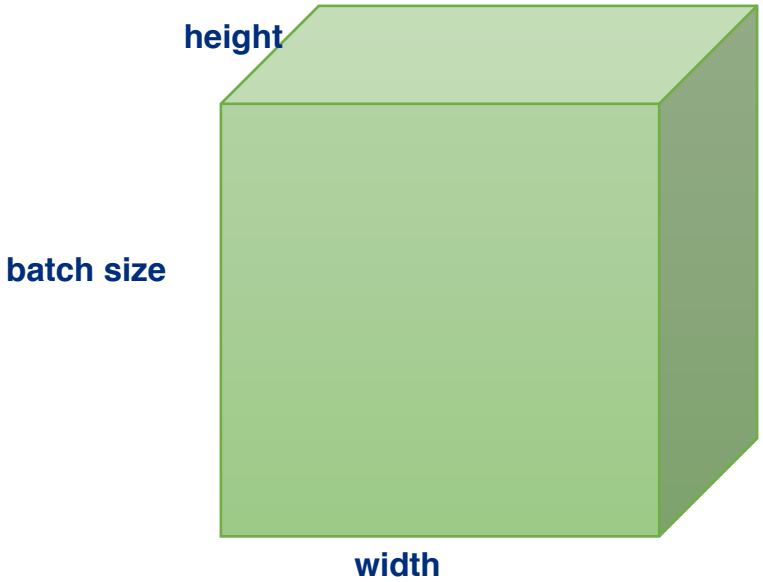


64 * 256이 대부분

PyTorch Tensor Shape Convention

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

여러 장의 이미지를 3차원의 텐서로 나타낸다.

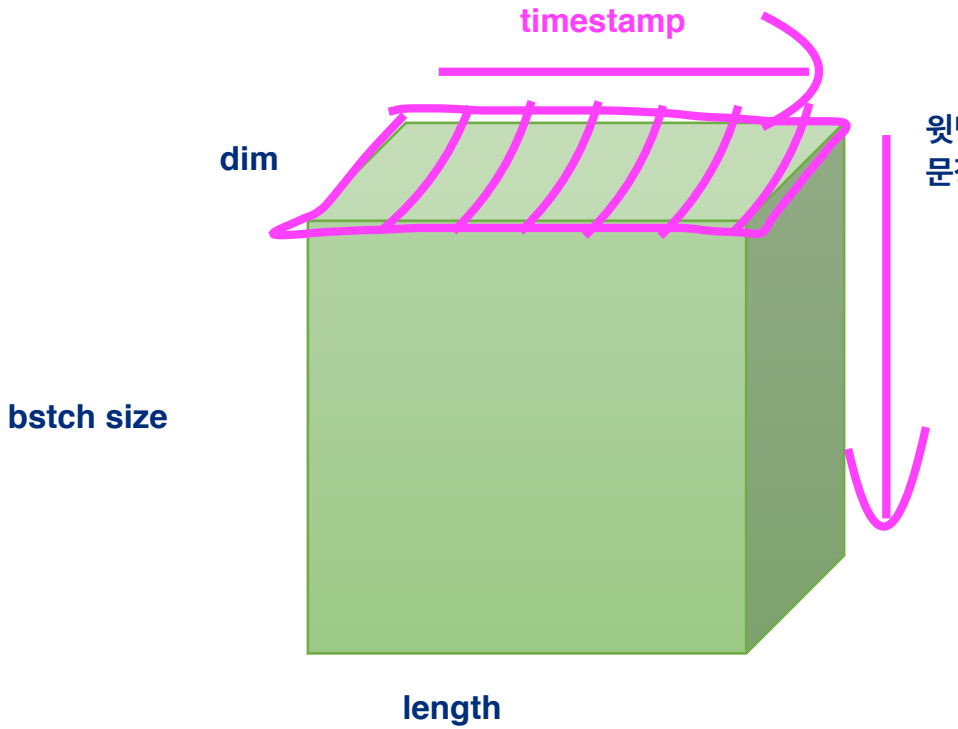


3차원 텐서를 이루는 pytorch의 컨밴션은 batch size, width, height 로 구성된다.

PyTorch Tensor Shape Convention

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

시계열 데이터, 시퀀셜 데이타(순차 정보)의 경우도 3차원 텐서를 다룬다.



윗면 = 하나의 문장
문장(혹은 데이터가)이 batch size만큼 존재한다!

Import

numpy 훑기

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.]])
```

4

3

```
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

행렬의 연산
=> 덧셈 뺄샘 : 행/열 사이즈가 같아야 한다.
=> 곱셈 : 마지막 차원과 첫번째 차원이 일치해야 한다.

하지만 다른 크기의 텐서를 더하거나 빼야 하는 상황이 발생한다.

pytorch의 기능(= broadcasting)이 자동으로 사이즈를 맞춰 연산을 해줌

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

|m1| = (1,2)
|m1| = |m2|

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

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

|m1| = (1,2)
|m2| = (1,) => (1,2)

```
tensor([[4., 5.]])
```
= [[1,2]] + [[3,3]]

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

|m1| = (1,2) => (2,2)
|m2| = (2, 1) => (2,2)

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

자동 으로 실행되기 때문에, 주의해서 사용해야 함
=> 의도하지 않은 동작을 했을 때
에러를 뱉는 것이 아닌, 자동으로 계산되기 때문에 디버깅이 힘듦

Multiplication vs Matrix Multiplication

elementwize 곱셈 : 사이즈가 같아야 함 => broadcasting
행렬 곱(inner products, dot products) => 마지막 열과 첫번째 행의 사이즈가 같아야 함

```
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))
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$$

(2x2) * (2x1) => (2x1)

=> 2X2
(broadcasting)

: [[1,2], [2,4]] X [[1,1], [2,2]]
=> [[1,2], [6,8]]

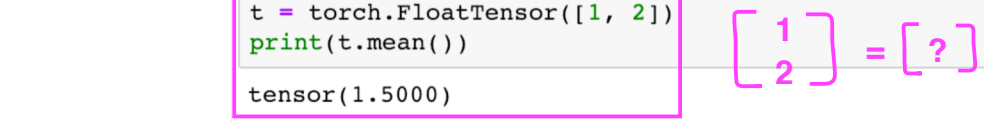
```
-------------
Mul vs Matmul
-------------
Shape of Matrix 1:  torch.Size([2, 2])
Shape of Matrix 2:  torch.Size([2, 1])
tensor([[ 5.],
        [11.]])
Shape of Matrix 1:  torch.Size([2, 2])
Shape of Matrix 2:  torch.Size([2, 1])
tensor([[1., 2.],
        [6., 8.]])
tensor([[1., 2.],
        [6.,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

Mean

평균

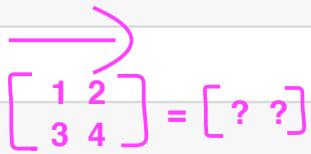
```
t = torch.FloatTensor([1, 2])
print(t.mean())
```

```
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.
```



(2x2) => (1x2) (vector가 됨)

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

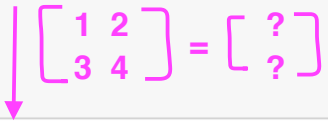
```
t = torch.FloatTensor([[1, 2], [3, 4]])
print(t)
```

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

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

dim 0을 없애겠어

dim 1을 없애겠어




```
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

max : 가장 큰 값을 찾아준다
argumax : 인덱스를 찾아줘

```
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]))
```