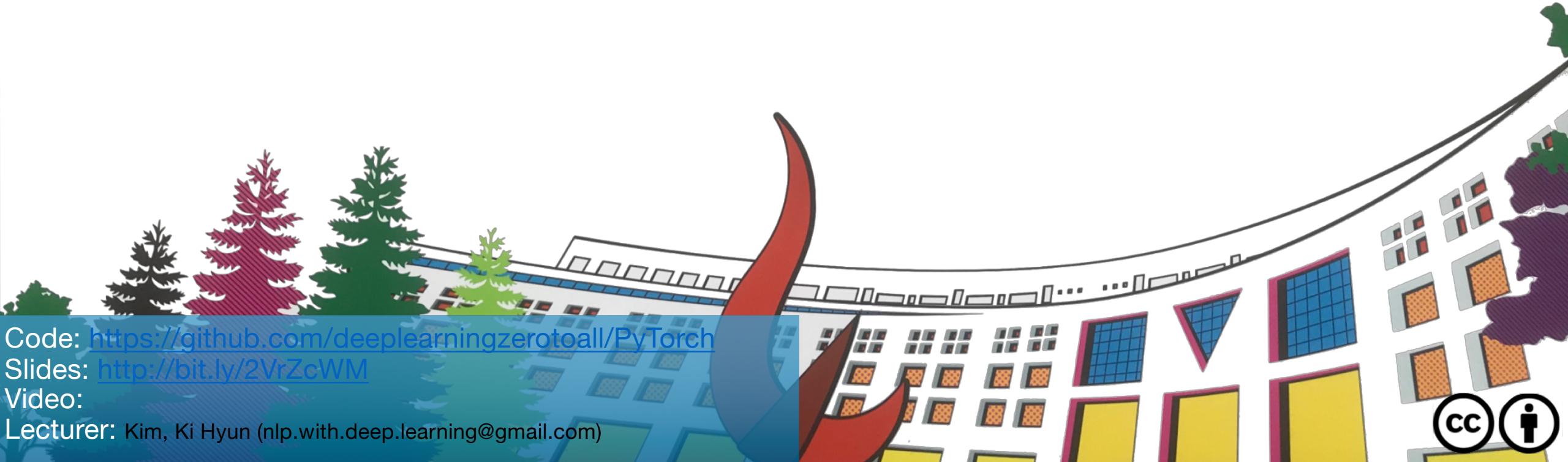


ML/DL for Everyone Season2

with 

PyTorch Basic Tensor Manipulation II



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

View (Reshape)

reshape => 텐서의 모양을 다시 만들어 주겠다.
중요함!!!!!! !shape를 자유자재로 바꿀 수 있어야 한다

```
t = np.array([[0, 1, 2],  
             [3, 4, 5]],  
  
             [[6, 7, 8],  
              [9, 10, 11]])  
ft = torch.FloatTensor(t)  
print(ft.shape)
```

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

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

```
print(ft.view([-1, 3])) 모양을 바꿔줘. 두개의 차원인데 앞에 모양은 모르겠고(-1), 뒤에는 3으로 만들어줘.  
print(ft.view([-1, 3]).shape)
```

```
tensor([[ 0.,  1.,  2.],  
        [ 3.,  4.,  5.],  
        [ 6.,  7.,  8.],  
        [ 9., 10., 11.]]) (2,2,3) -> (2x2, 3) = (4,3)  
torch.Size([4, 3])
```

```
print(ft.view([-1, 1, 3])) => 선언하는 숫자들의 곱(-1제외 )이 기존의 곱과 같으면 됨  
print(ft.view([-1, 1, 3]).shape)
```

```
tensor([[[ 0.,  1.,  2.]],  
       [[ 3.,  4.,  5.]],  
       [[ 6.,  7.,  8.]],  
       [[ 9., 10., 11.]]]) (2, 2,3) -> (2x2, 1, 3) = (4, 1, 3)  
torch.Size([4, 1, 3]) 4 x 1 x 3 = 12
```

Squeeze

en) 쥐어 짜다.

view와 달리 자동으로 원하는 dimension 혹은 전체에서 dimension이 1인 element를 없애줌

```
ft = torch.FloatTensor([[0], [1], [2]])
print(ft)
print(ft.shape)
```

```
tensor([[0.],
       [1.],
       [2.]])
torch.Size([3, 1])
```

```
print(ft.squeeze())
print(ft.squeeze().shape)
```

```
tensor([0., 1., 2.])           ft, squeeze(dim= 1) = 없어짐
torch.Size([3])
```

Unsqueeze

원하는 dimension에 1을 넣어줌. 반드시 dim 명시 필요함

```
ft = torch.Tensor([0, 1, 2])
print(ft.shape)
torch.Size([3])  
lftl = (3,) => (1,3)  
print(ft.unsqueeze(0))
print(ft.unsqueeze(0).shape)  
  
tensor([[0., 1., 2.]])
torch.Size([1, 3])  
  
print(ft.view(1, -1))
print(ft.view(1, -1).shape)  
  
tensor([[0., 1., 2.]])
torch.Size([1, 3])  
  
print(ft.unsqueeze(1))
print(ft.unsqueeze(1).shape)    lftl = (3, ) => (3,1)  
  
tensor([[0.],
       [1.],
       [2.]])
torch.Size([3, 1])            dim=-1 => dim=1  
  
print(ft.unsqueeze(-1))
print(ft.unsqueeze(-1).shape)  lftl = (3, ) => (3,1)  
  
tensor([[0.],
       [1.],
       [2.]])
torch.Size([3, 1])
```

Type Casting

tensor의 타입을 바꿔주겠다.

```
lt = torch.LongTensor([1, 2, 3, 4])  
print(lt)
```

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

```
print(lt.float())
```

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

```
bt = torch.ByteTensor([True, False, False, True])
```

```
print(bt)
```

연산을 활용하여 true/false 값을 나타낼 수 있다.

bt = (lt == 3)
-> bt = (0,0,1,0)

```
tensor([1, 0, 0, 1], dtype=torch.uint8)
```

```
print(bt.long())
```

```
print(bt.float())
```

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

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

Concatenate

이어 붙이다.

bt = (lt == 3)
-> bt = (0,0,1,0)

```
x = torch.FloatTensor([[1, 2], [3, 4]])  
y = torch.FloatTensor([[5, 6], [7, 8]])      같은 사이즈의 tensor
```

```
print(torch.cat([x, y], dim=0))  
print(torch.cat([x, y], dim=1))
```

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

Stacking

concat을 편리하게 이용할 수 있음. 몇 가지 기능을 단축시켜 놓음

```
x = torch.FloatTensor([1, 4])
y = torch.FloatTensor([2, 5])           |x|=|y|=|z|=(2,)
z = torch.FloatTensor([3, 6])
```

```
print(torch.stack([x, y, z]))  텐서들을 쌓아라! -> 새로운 차원이 생기게 됨
print(torch.stack([x, y, z], dim=1))
```

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

```
print(torch.cat([x.unsqueeze(0), y.unsqueeze(0), z.unsqueeze(0)], dim=0))
(1,2)          (1,2)          (1,2)
tensor([[1., 4.],
        [2., 5.],
        [3., 6.]])
```

Ones and Zeros

똑같은 shape의 1/0으로 가득 찬 텐서를 리턴
cpu / gpu .. 장비..

```
x = torch.FloatTensor([[0, 1, 2], [2, 1, 0]]))  
print(x)
```

```
tensor([[0., 1., 2.],  
       [2., 1., 0.]])
```

$|x| = (2, 3)$

```
print(torch.ones_like(x))  
print(torch.zeros_like(x))
```

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

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

In-place Operation

메모리에 의존하지 말고 새로운 값을 넣어라.

=> 새로운 텐서를 할당하지 않고도 값을 변경한다.

속도면에서 조금 빠르지 않을까.. 추측 (gc안돌아도 됨)
(but, gc가 잘 설계되어 있음)

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

```
print(x.mul(2.)) <=x * 2
print(x)
print(x.mul_(2.))
print(x)
```

```
tensor([[2., 4.],
        [6., 8.]])
tensor([[1., 2.],
        [3., 4.]])
tensor([[2., 4.],
        [6., 8.]])
tensor([[2., 4.],
        [6., 8.]])
```