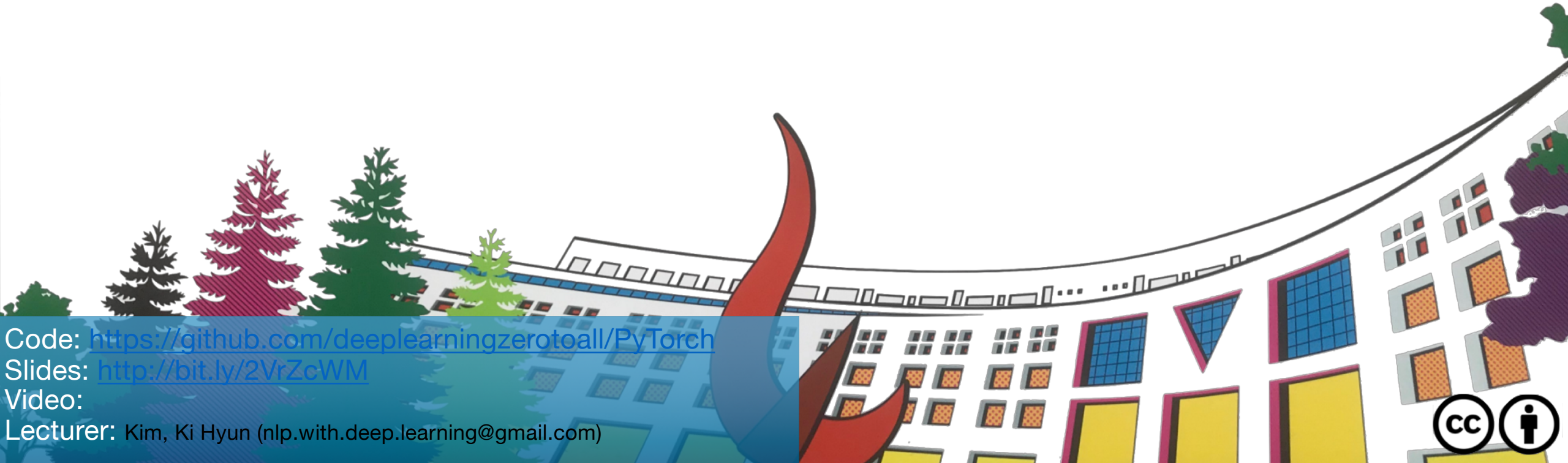
ML/DL for Everyone Season2

with PYTORCH

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)

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

              [[6, 7, 8],
               [9, 10, 11]]])
ft = torch.FloatTensor(t)
print(ft.shape)
```

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

```
print(ft.view([-1, 3]))
print(ft.view([-1, 3]).shape)
```

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

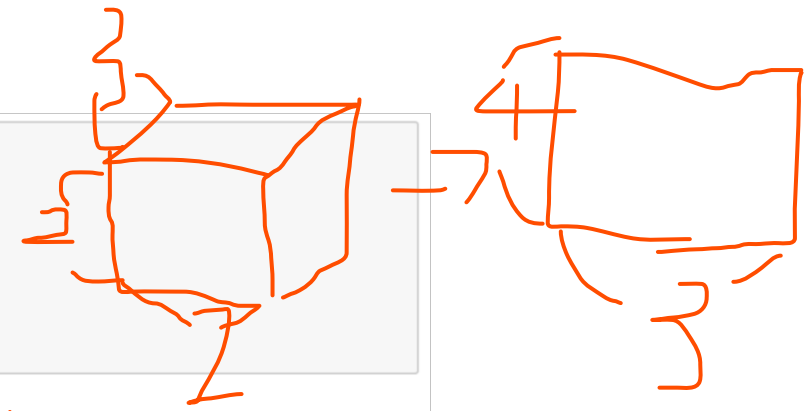
```
print(ft.view([-1, 1, 3]))
print(ft.view([-1, 1, 3]).shape)
```

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

        [[ 3.,  4.,  5.]],

        [[ 6.,  7.,  8.]],

        [[ 9., 10., 11.]]])
torch.Size([4, 1, 3])
```



ft.shape = [2, 2, 3] 3×2×3=12

⇓

[2×2, 3] = [4, 3] 4×3=12

⇓

[2×2, 1, 3] = [4, 1, 3]

4×1×3=12

Squeeze

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

Unsqueeze

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

```
torch.Size([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)
```

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

```
print(ft.unsqueeze(-1))
print(ft.unsqueeze(-1).shape)
```

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

Type Casting

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

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

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

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

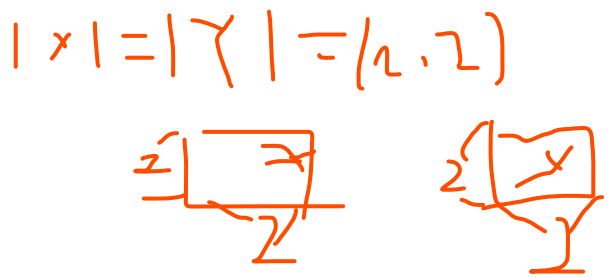
```
bt = torch.ByteTensor([True, False, False, True])
print(bt)
```

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

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

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

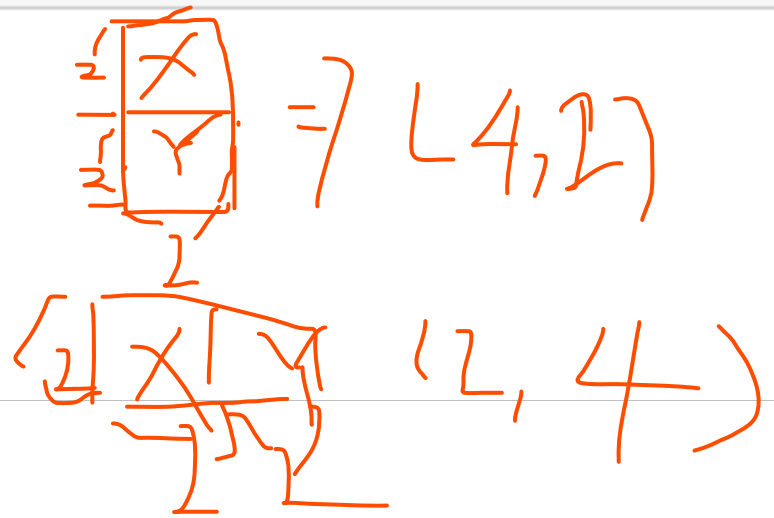
Concatenate



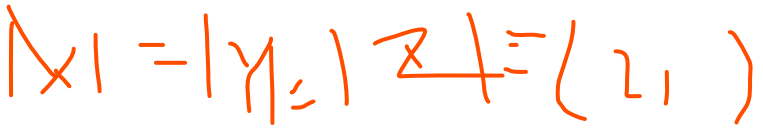
```
x = torch.FloatTensor([[1, 2], [3, 4]])
y = torch.FloatTensor([[5, 6], [7, 8]])
```

```
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.]])
```

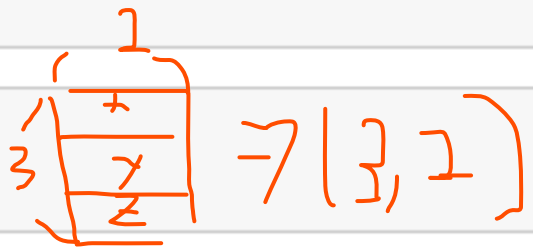


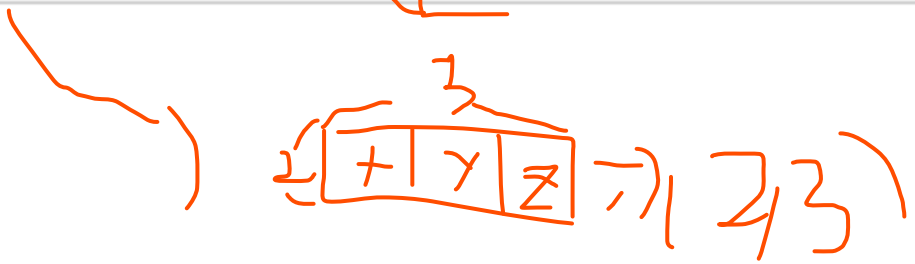
Stacking



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

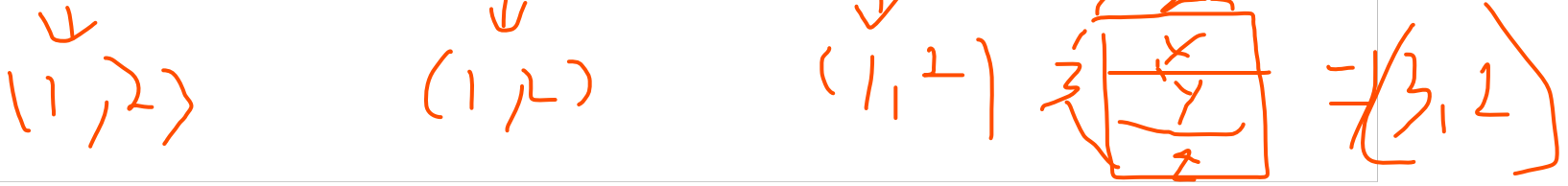
```
print(torch.stack([x, y, z]))
print(torch.stack([x, y, z], dim=1))
```





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

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

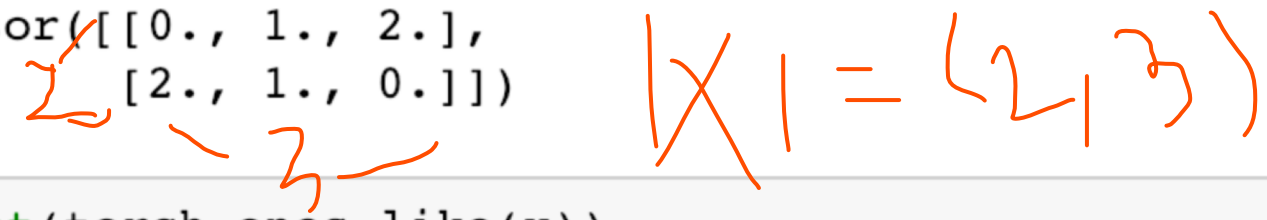


```
tensor([[1., 4.],
        [2., 5.],
        [3., 6.]])
```

Ones and Zeros

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

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




```
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

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

```
print(x.mul(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.]])
```