

With TF 1.0!



# Lab 8

## Tensor Manipulation

Sung Kim <[hunkim+ml@gmail.com](mailto:hunkim+ml@gmail.com)>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



# Call for comments

Please feel free to add comments directly on these slides

Other slides: <https://goo.gl/jPtVNt>



With TF 1.0!



# Lab 8

## Tensor Manipulation

Sung Kim <[hunkim+ml@gmail.com](mailto:hunkim+ml@gmail.com)>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>



<https://github.com/hunkim/DeepLearningZeroToAll/>



zeran4

1 commit / 5 ++ / 4 --

#11



jennykang

19 commits / 940 ++ / 253 --

#2



GzuPark

14 commits / 41 ++ / 31 --

#3



kkweon

5 commits / 372 ++ / 296 --

#4



BlueMelon715

4 commits / 45 ++ / 34 --

#5



jin-chong

2 commits / 4 ++ / 4 --

#6



FuZer

2 commits / 37 ++ / 30 --

#7



cynthia

1 commit / 28 ++ / 28 --

#8



keon

1 commit / 3 ++ / 3 --

#9



allieus

1 commit / 55 ++ / 59 --

#10

# Simple 1D array and slicing



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

# Simple 1D array and slicing



```
t = np.array([0., 1., 2., 3., 4., 5., 6.])
pp.pprint(t)
print(t.ndim) # rank
print(t.shape) # shape
print(t[0], t[1], t[-1])
print(t[2:5], t[4:-1])
print(t[:2], t[3:])
```

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.])
1
(7,)
0.0 1.0 6.0
[ 2.  3.  4.] [ 4.  5.]
[ 0.  1.] [ 3.  4.  5.  6.]
```

# 2D Array

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

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

2
(4, 3)
```

# Shape, Rank, Axis

```
t = tf.constant([1,2,3,4])  
tf.shape(t).eval()
```

```
array([4], dtype=int32)
```

```
t = tf.constant([[1,2],  
                 [3,4]])  
tf.shape(t).eval()
```

```
array([2, 2], dtype=int32)
```

```
t = tf.constant([[[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]],  
                 [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]]])  
tf.shape(t).eval()
```

```
array([1, 2, 3, 4], dtype=int32)
```



# Shape, Rank, Axis

```
t = tf.constant([[[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]],  
                  [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]]])  
tf.shape(t).eval()
```

```
array([1, 2, 3, 4], dtype=int32)
```

```
[  
  [  
    [  
      [1, 2, 3, 4],  
      [5, 6, 7, 8],  
      [9, 10, 11, 12]  
    ],  
    [  
      [13, 14, 15, 16],  
      [17, 18, 19, 20],  
      [21, 22, 23, 24]  
    ]  
  ]  
]
```

# Matmul VS multiply

```
matrix1 = tf.constant([[1., 2.], [3., 4.]])  
matrix2 = tf.constant([[1.],[2.]])  
print("Metrix 1 shape", matrix1.shape)  
print("Metrix 2 shape", matrix2.shape)  
tf.matmul(matrix1, matrix2).eval()
```

Metrix 1 shape (2, 2)

Metrix 2 shape (2, 1)

```
array([[ 5.],  
       [11.]], dtype=float32)
```

# Matmul VS multiply

```
matrix1 = tf.constant([[1., 2.], [3., 4.]])  
matrix2 = tf.constant([1.],[2.])  
print("Metrix 1 shape", matrix1.shape)  
print("Metrix 2 shape", matrix2.shape)  
tf.matmul(matrix1, matrix2).eval()
```

```
Metrix 1 shape (2, 2)
```

```
Metrix 2 shape (2, 1)
```

```
array([[ 5.],  
       [11.]], dtype=float32)
```

```
(matrix1*matrix2).eval()
```

```
array([[ 1.,  2.],  
       [ 6.,  8.]], dtype=float32)
```

# Broadcasting



**WARNING**

```
# Operations between the same shapes
```

```
matrix1 = tf.constant([[3., 3.]])
```

```
matrix2 = tf.constant([[2., 2.]])
```

```
(matrix1 + matrix2).eval()
```

```
array([[ 5.,  5.]], dtype=float32)
```

# Broadcasting



```
matrix1 = tf.constant([[1., 2.]])  
matrix2 = tf.constant(3.)  
(matrix1+matrix2).eval()
```

```
array([[ 4.,  5.]], dtype=float32)
```

```
matrix1 = tf.constant([[1., 2.]])  
matrix2 = tf.constant([3., 4.])  
(matrix1+matrix2).eval()
```

```
array([[ 4.,  6.]], dtype=float32)
```

```
matrix1 = tf.constant([[1., 2.]])  
matrix2 = tf.constant([[3.],[4.]])  
(matrix1+matrix2).eval()
```

```
array([[ 4.,  5.],  
       [ 5.,  6.]], dtype=float32)
```

# Reduce mean

```
tf.reduce_mean([1, 2], axis=0).eval()
```

1

```
x = [[1., 2.],  
      [3., 4.]]
```

```
tf.reduce_mean(x).eval()
```

2.5

```
tf.reduce_mean(x, axis=0).eval()
```

```
array([ 2.,  3.], dtype=float32)
```

```
tf.reduce_mean(x, axis=1).eval()
```

```
array([ 1.5,  3.5], dtype=float32)
```

```
tf.reduce_mean(x, axis=-1).eval()
```

```
array([ 1.5,  3.5], dtype=float32)
```

# Reduce sum

```
x = [[1., 2.],  
      [3., 4.]]
```

```
tf.reduce_sum(x).eval()
```

```
10.0
```

```
tf.reduce_sum(x, axis=0).eval()
```

```
array([ 4.,  6.], dtype=float32)
```

```
tf.reduce_sum(x, axis=-1).eval()
```

```
array([ 3.,  7.], dtype=float32)
```

```
tf.reduce_mean(tf.reduce_sum(x, axis=-1)).eval()
```

```
5.0
```

# Argmax

```
x = [[0, 1, 2],  
      [2, 1, 0]]  
tf.argmax(x, axis=0).eval()
```

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

```
tf.argmax(x, axis=1).eval()
```

```
array([2, 0])
```

```
tf.argmax(x, axis=-1).eval()
```

```
array([2, 0])
```



# Reshape\*\*

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

```
(2, 2, 3)
```

```
tf.reshape(t, shape=[-1, 3]).eval()
```

```
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11]])
```

```
tf.reshape(t, shape=[-1, 1, 3]).eval()
```

```
array([[[ 0,  1,  2]],  
       [[ 3,  4,  5]],  
       [[ 6,  7,  8]],  
       [[ 9, 10, 11]])
```

# Reshape (squeeze, expand)

```
tf.squeeze([[0], [1], [2]]).eval()
```

```
array([0, 1, 2], dtype=int32)
```

```
tf.expand_dims([0, 1, 2], 1).eval()
```

```
array([[0],  
       [1],  
       [2]], dtype=int32)
```

# One hot



```
tf.one_hot([[0], [1], [2], [0]], depth=3).eval()
```

```
array([[[ 1.,  0.,  0.],  
        [ 0.,  1.,  0.],  
        [ 0.,  0.,  1.],  
        [ 1.,  0.,  0.]], dtype=float32)
```

```
t = tf.one_hot([[0], [1], [2], [0]], depth=3)  
tf.reshape(t, shape=[-1, 3]).eval()
```

```
array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.],  
       [ 1.,  0.,  0.]], dtype=float32)
```

# Casting

```
tf.cast([1.8, 2.2, 3.3, 4.9], tf.int32).eval()
```

```
array([1, 2, 3, 4], dtype=int32)
```

```
tf.cast([True, False, 1 == 1, 0 == 1], tf.int32).eval()
```

```
array([1, 0, 1, 0], dtype=int32)
```

# Stack

```
x = [1, 4]  
y = [2, 5]  
z = [3, 6]
```

```
# Pack along first dim.  
tf.stack([x, y, z]).eval()
```

```
array([[1, 4],  
       [2, 5],  
       [3, 6]], dtype=int32)
```

```
tf.stack([x, y, z], axis=1).eval()
```

```
array([[1, 2, 3],  
       [4, 5, 6]], dtype=int32)
```

# Ones and Zeros like

```
x = [[0, 1, 2],  
      [2, 1, 0]]  
  
tf.ones_like(x).eval()  
  
array([[1, 1, 1],  
       [1, 1, 1]], dtype=int32)
```

```
tf.zeros_like(x).eval()  
  
array([[0, 0, 0],  
       [0, 0, 0]], dtype=int32)
```

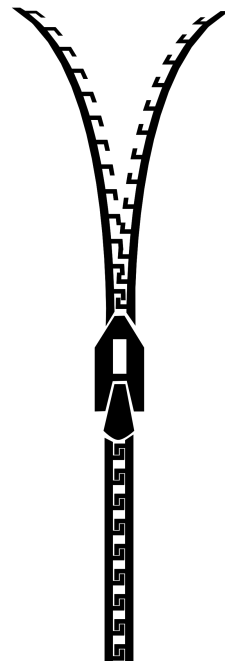
# Zip

```
for x, y in zip([1, 2, 3], [4, 5, 6]):  
    print(x, y)
```

```
1 4  
2 5  
3 6
```

```
for x, y, z in zip([1, 2, 3], [4, 5, 6], [7, 8, 9]):  
    print(x, y, z)
```

```
1 4 7  
2 5 8  
3 6 9
```





One step  
at a time...



With TF 1.0!



# Lab 9-I

## NN for XOR

Sung Kim <[hunkim+ml@gmail.com](mailto:hunkim+ml@gmail.com)>

Code: <https://github.com/hunkim/DeepLearningZeroToAll/>

