

## ML lec 09-1

- ✓ 그 동안 풀지 못했던 XOR 난제를 Sigmoid로 풀게 됨



$$\begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ 5 \end{bmatrix} + -8 = 0 + 0 - 8 = -8, \text{ sigmoid}(-8) = 0$$

$$\begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} + 3 = 0 + 0 + 3 = 3, \text{ sigmoid}(3) = 1$$

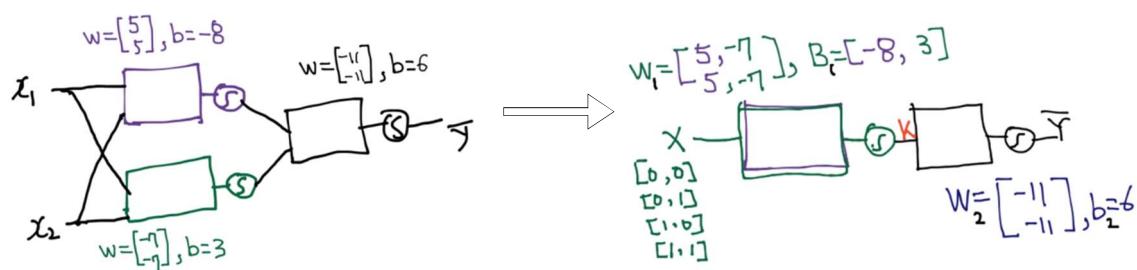
$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} -11 \\ -11 \end{bmatrix} + 6 = 0 + -11 + 6 = -5$$

$x_1$	$x_2$	$y_1$	$y_2$	$\bar{y}$	XOR
0	0	0	1	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	0	0	0

- ✓ 그 방법을 간단하게 도식화하면 다음과 같고, 이것은 하나의 Neural Network(NN)이다

Forward propagation

NN



- TF로의 구현법

```

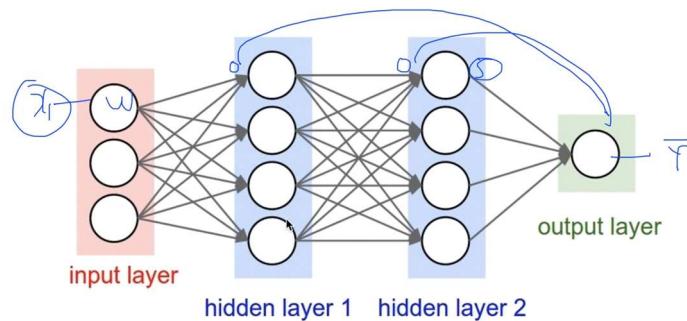
1 # NN
2 K = tf.sigmoid(tf.matmul(X, W1) + b1)
3 hypothesis = tf.sigmoid(tf.matmul(K, W2) + b2)

```

## ML lec 09-2

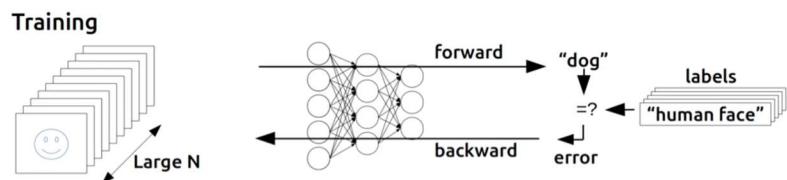
- ✓ Cost 함수에 경사하강법을 적용하기 위해서는 미분이 필요함
  - 하지만 NN이 되면서 미분하기엔 복잡해짐

### Derivation



- ✓ Backpropagation이 해결할 수 있다

Backpropagation  
(1974, 1982 by Paul Werbos, 1986 by Hinton)



### Back propagation (chain rule)

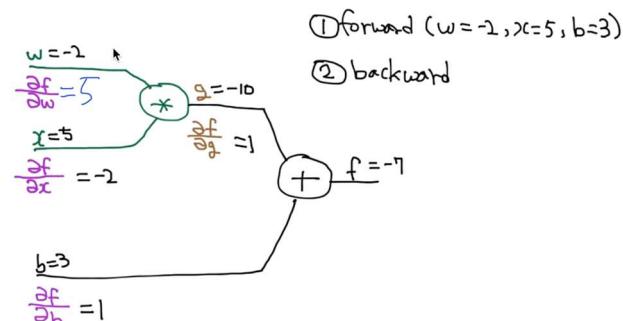
$$f = w \cdot x + b, g = w \cdot x, f = g + b$$

$$\frac{\partial f}{\partial w} = 1, \frac{\partial f}{\partial b} = 1$$

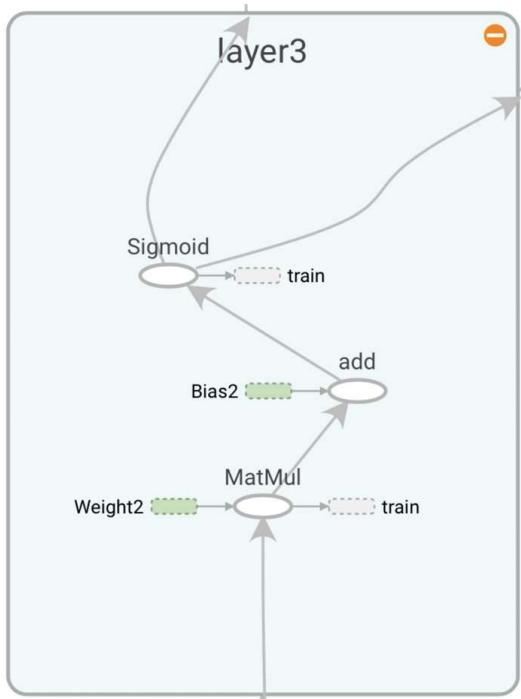
$$\frac{\partial g}{\partial w} = x, \frac{\partial g}{\partial x} = w$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} = 1 \cdot w = -2$$

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial w} = 1 \cdot x = 5$$

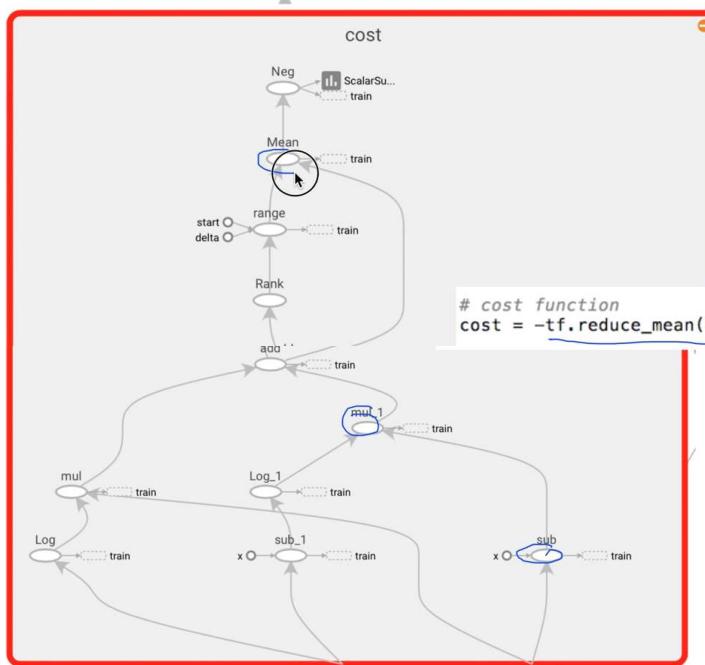


- 그 Backpropagation의 과정은 TensorFlow의 기능 중 하나인 TensorBoard를 통해 확인할 수 있다



## Back propagation in TensorFlow TensorBoard

```
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```



## Back propagation i TensorFlow TensorBoard

```
# cost function
cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))
```

# ML lab 09-1

## ✓ XOR을 TF로 나타내기

```
1 x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
2 y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
3
```

$x_1$	0	0	1	1
$x_2$	0	1	0	1
$Y$	0	1	1	0

## ✓ XOR with Logistic Regression

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays 'temp.py' with the following content:

```
1 # Lab 9 XOR
2 import tensorflow as tf
3 import numpy as np
4
5 tf.set_random_seed(777) # for reproducibility
6
7 x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
8 y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
9
10 X = tf.placeholder(tf.float32, [None, 2])
11 Y = tf.placeholder(tf.float32, [None, 1])
12
13 W = tf.Variable(tf.random_normal([2, 1]), name="weight")
14 b = tf.Variable(tf.random_normal([1]), name="bias")
15
16 # Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
17 hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
18
19 # cost/loss function
20 cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
21 train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
22
23 # Accuracy computation
24 # True if hypothesis>0.5 else False
25 predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
26 accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
27
28 # Launch graph
29 with tf.Session() as sess:
30     # Initialize TensorFlow variables
31     sess.run(tf.global_variables_initializer())
32
33     for step in range(10001):
34         _, cost_val, w_val = sess.run(
35             [train, cost, W], feed_dict={X: x_data, Y: y_data}
36         )
37         if step % 100 == 0:
38             print(step, cost_val, w_val)
39
40     # Accuracy report
41     h, c, a = sess.run(
42         [hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data}
43     )
44     print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```

On the right, the Python console window shows the output of the code execution. It includes the printed hypothesis values, the correct values, and the calculated accuracy.

```
8000 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8100 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8200 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8300 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8400 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8500 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8600 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8700 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8800 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
8900 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9000 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9100 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9200 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9300 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9400 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9500 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9600 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9700 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9800 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
9900 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
10000 0.6931472 [[1.222877e-07]
[1.2214579e-07]]
```

Hypothesis: [[0.5]
[0.5]
[0.5]
[0.5]]
Correct: [[0.]
[0.]
[0.]
[0.]]
Accuracy: 0.5

- 알던대로 작동하지 않는다

## ✓ 해결방법은 NN이다

```
1 W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
2 b1 = tf.Variable(tf.random_normal([2]), name='bias1')
3 layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
4
5 W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
6 b2 = tf.Variable(tf.random_normal([1]), name='bias2')
7 hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
```

>>> 성공적인 실행 화면

The screenshot shows the Spyder Python 3.5 IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar below has icons for file operations like Open, Save, and Run. The main window displays the code for 'XOR for NN.py' and its execution output.

**Code Content:**

```
1 # Lab 9 XOR
2 import tensorflow as tf
3 import numpy as np
4
5 tf.set_random_seed(777) # for reproducibility
6
7 x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
8 y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
9
10 X = tf.placeholder(tf.float32, [None, 2])
11 Y = tf.placeholder(tf.float32, [None, 1])
12
13 W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
14 b1 = tf.Variable(tf.random_normal([2]), name='bias1')
15 layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
16
17 W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
18 b2 = tf.Variable(tf.random_normal([1]), name='bias2')
19 hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
20
21 # cost/loss function
22 cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
23 train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
24
25 # Accuracy computation
26 # True if hypothesis>0.5 else False
27 predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
28 accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
29
30 # Launch graph
31 with tf.Session() as sess:
32     # Initialize TensorFlow variables
33     sess.run(tf.global_variables_initializer())
34
35     for step in range(10001):
36         _, cost_val = sess.run([train, cost], feed_dict={X: x_data, Y: y_data})
37         if step % 100 == 0:
38             print(step, cost_val)
39
40     # Accuracy report
41     h, p, a = sess.run(
42         [hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data}
43     )
44
45     print("\nHypothesis:\n{}\nPredicted:\n{}\nAccuracy:\n{}".format(h, p, a))
```

**Execution Output:**

```
IPython console
Console 1/A
6200 0.043852367
6300 0.04218307
6400 0.040628236
6500 0.03917697
6600 0.037819687
6700 0.0365479
6800 0.03535401
6900 0.034231417
7000 0.033174098
7100 0.03217671
7200 0.031234514
7300 0.030343082
7400 0.029498689
7500 0.028697733
7600 0.027937133
7700 0.027213924
7800 0.026525486
7900 0.025869496
8000 0.02524376
8100 0.024646249
8200 0.02407522
8300 0.02352893
8400 0.023005854
8500 0.022504618
8600 0.022023894
8700 0.021562472
8800 0.02111922
8900 0.020693151
9000 0.02028333
9100 0.01988883
9200 0.019508727
9300 0.019142445
9400 0.01878915
9500 0.018448206
9600 0.018119005
9700 0.017800948
9800 0.017493473
9900 0.017196104
10000 0.0169083

Hypothesis:
[[0.01368038]
 [0.98169047]
 [0.9817343 ]
 [0.01679076]]
Predicted:
[[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:
1.0
```

## ✓ 더 넓고 기잎게

```
1 W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
2 ...
3 hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4) # 4스택
```

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** The code is located in a file named "XOR for NN more deep and wide.py".

```
1 # Lab 9 XOR
2 import tensorflow as tf
3 import numpy as np
4
5 tf.set_random_seed(777) # for reproducibility
6
7 x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
8 y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
9
10 X = tf.placeholder(tf.float32, [None, 2])
11 Y = tf.placeholder(tf.float32, [None, 1])
12
13 W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
14 b1 = tf.Variable(tf.random_normal([10]), name='bias1')
15 layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
16
17 W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
18 b2 = tf.Variable(tf.random_normal([10]), name='bias2')
19 layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)
20
21 W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
22 b3 = tf.Variable(tf.random_normal([10]), name='bias3')
23 layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)
24
25 W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
26 b4 = tf.Variable(tf.random_normal([1]), name='bias4')
27 hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)
28
29 # cost/loss function
30 cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
31 train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
32
33 # Accuracy computation
34 # True if hypothesis>0.5 else False
35 predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
36 accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
37
38 # Launch graph
39 with tf.Session() as sess:
40     # Initialize TensorFlow variables
41     sess.run(tf.global_variables_initializer())
42
43     for step in range(10001):
44         _, cost_val = sess.run([train, cost], feed_dict={X: x_data, Y: y_data})
45         if step % 100 == 0:
46             print(step, cost_val)
47
48     # Accuracy report
49     h, c, a = sess.run(
50         [hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data}
51     )
52     print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```
- Console:** The output shows the cost values for each step from 5900 to 10000, followed by the final hypothesis, correct predictions, and accuracy.

- 결과가 이전보다 더 정확해졌다

## ✓ Exercise

Spyder (Python 3.5)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - D:\Documents\ML\MNIST\Number Recognition\_Deeper Wider.py XOR for NN more deep and wide.py

```
1 # ML Lab 09-1
2 import tensorflow as tf
3 import matplotlib.pyplot as plt
4 import random
5
6 tf.set_random_seed(777) # for reproducibility
7
8 from tensorflow.examples.tutorials.mnist import input_data
9
10 # Check out https://www.tensorflow.org/get_started/mnist/beginners for
11 # more information about the mnist dataset
12 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
13
14 nb_classes = 10
15
16 # MNIST data image of shape 28 * 28 = 784
17 X = tf.placeholder(tf.float32, [None, 784])
18 # 0 - 9 digits recognition = 10 classes
19 Y = tf.placeholder(tf.float32, [None, nb_classes])
20
21 W1 = tf.Variable(tf.random_normal([784, 256]), name='weight1')
22 b1 = tf.Variable(tf.random_normal([256]), name='bias1')
23 layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
24
25 W2 = tf.Variable(tf.random_normal([256, 256]), name='weight2')
26 b2 = tf.Variable(tf.random_normal([256]), name='bias2')
27 layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)
28
29 W3 = tf.Variable(tf.random_normal([256, 256]), name='weight3')
30 b3 = tf.Variable(tf.random_normal([256]), name='bias3')
31 layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)
32
33 W4 = tf.Variable(tf.random_normal([256, nb_classes]), name='weight4')
34 b4 = tf.Variable(tf.random_normal([nb_classes]), name='bias4')
35
36 # Hypothesis (using softmax)
37 hypothesis = tf.nn.softmax(tf.matmul(layer3, W4) + b4)
38
39 cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
40 train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
41
42 # Test model
43 is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
44 # Calculate accuracy
45 accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
46
47 # parameters
48 num_epochs = 15
49 batch_size = 100
50 num_iterations = int(mnist.train.num_examples / batch_size)
51
52 with tf.Session() as sess:
53     # Initialize TensorFlow variables
54     sess.run(tf.global_variables_initializer())
55     # Training cycle
56     for epoch in range(num_epochs):
57         avg_cost = 0
58
59         for i in range(num_iterations):
60             batch_xs, batch_ys = mnist.train.next_batch(batch_size)
61             _, cost_val = sess.run([train, cost], feed_dict={X: batch_xs, Y: batch_ys})
62             avg_cost += cost_val / num_iterations
63
64     print("Epoch: {0}, Cost: {1:.9f} ".format(epoch+1, cost_val))
65
66     print("Accuracy: {0:.3f} ".format(accuracy.eval({X:mnist.validation.images, Y:mnist.validation.labels})))
67
68     print("Label: {0} ".format(mnist.validation.labels[0]))
69     print("Prediction: {0} ".format(hypothesis.eval({X:mnist.validation.images})))
70
71     print("In [12]: |")
72
73     print("In [13]: |")
```

IPython console

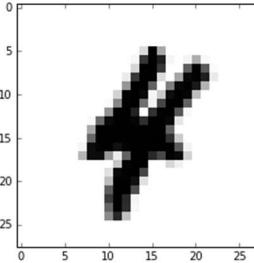
Console 1/A

```
[9.9877375e-01]
[9.9876839e-01]
[1.86780069e-03]
Correct: [[0.]
[1.]
[1.]
[0.]]
Accuracy: 1.0
```

In [12]: runfile('D:/Documents/ML/MNIST/Number Recognition\_Deeper.py')

Extracting MNIST\_data/train-images-idx3-ubyte.gz  
Extracting MNIST\_data/train-labels-idx1-ubyte.gz  
Extracting MNIST\_data/t10k-images-idx3-ubyte.gz  
Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz

Epoch: 0001, Cost: 1.613946109  
Epoch: 0002, Cost: 0.767379145  
Epoch: 0003, Cost: 0.611400311  
Epoch: 0004, Cost: 0.524184886  
Epoch: 0005, Cost: 0.466777018  
Epoch: 0006, Cost: 0.422271322  
Epoch: 0007, Cost: 0.388455596  
Epoch: 0008, Cost: 0.361273837  
Epoch: 0009, Cost: 0.337292241  
Epoch: 0010, Cost: 0.316751879  
Epoch: 0011, Cost: 0.298987155  
Epoch: 0012, Cost: 0.282859016  
Epoch: 0013, Cost: 0.269032453  
Epoch: 0014, Cost: 0.255886539  
Epoch: 0015, Cost: 0.243981089  
Learning finished  
Accuracy: 0.8887  
Label: [4]  
Prediction: [4]



In [13]: |

- 이전 결과보다 더 발전한 모습이다

## ML lab 09-2

- ✓ Old Fashion : print, print, print....
- ✓ New Way!

- 1 From TF graph, decide which tensors you want to log

```
w2_hist = tf.summary.histogram("weights2", W2)
cost_summ = tf.summary.scalar("cost", cost)
```

- 2 Merge all summaries

```
summary = tf.summary.merge_all()
```

- 3 Create writer and add graph

```
# Create summary writer
writer = tf.summary.FileWriter('./logs')
writer.add_graph(sess.graph)
```

- 4 Run summary merge and add\_summary

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
```

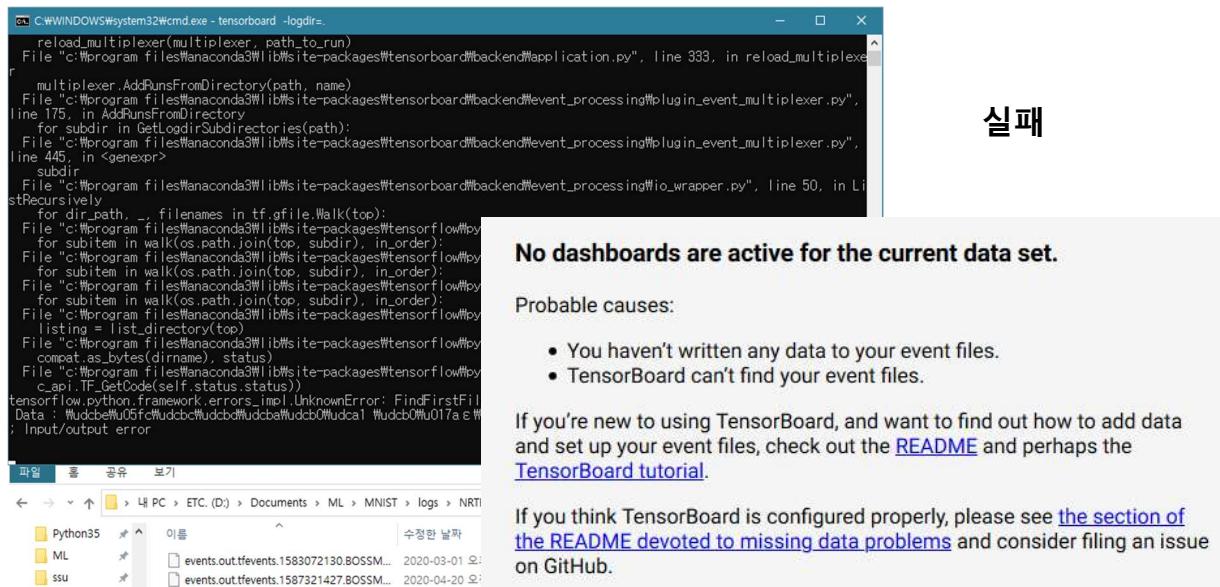
- 5 Launch TensorBoard

```
tensorboard --logdir=./logs
```

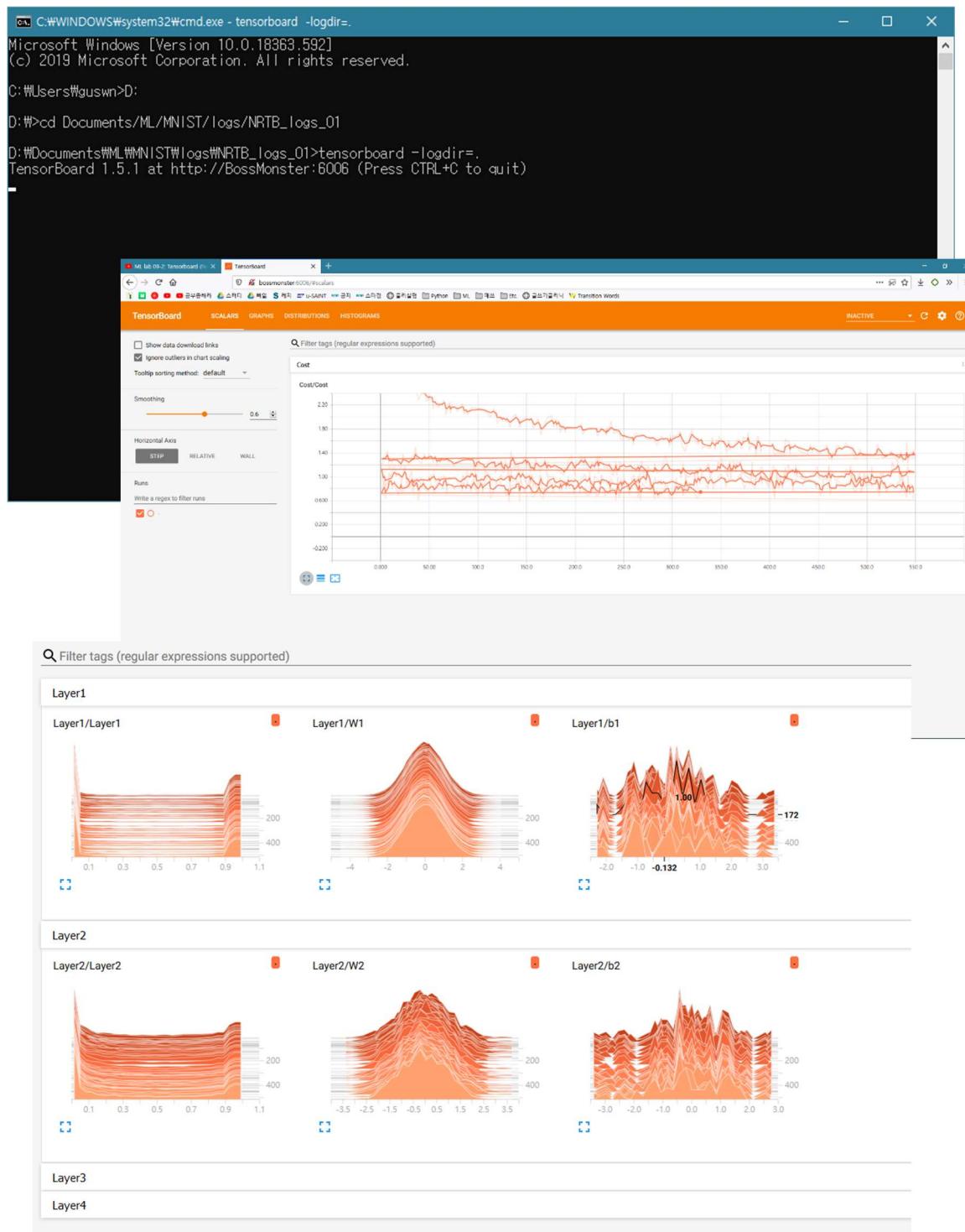
- 과정은 더 복잡해진 것 같다

- ✓ Exercise 첫 시도

## 5 steps of using TensorBoard

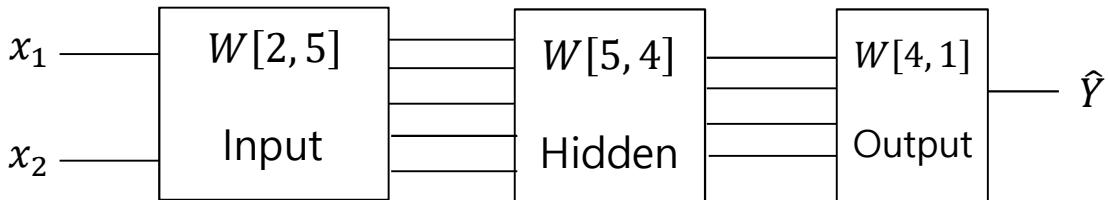


- 검색 끝에 겨우 성공(폴더명에 한국어 금지, 경로 직접 이동)



# ML lec 10-1

## ✓ Neural Network의 구조



- 9단까지도 가능

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")
W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight2")
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight3")
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight4")
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight5")
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight6")
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight7")
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight8")
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight9")
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0), name = "Weight10")
W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0), name = "Weight11")
b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")
b11 = tf.Variable(tf.zeros([1]), name="Bias11")

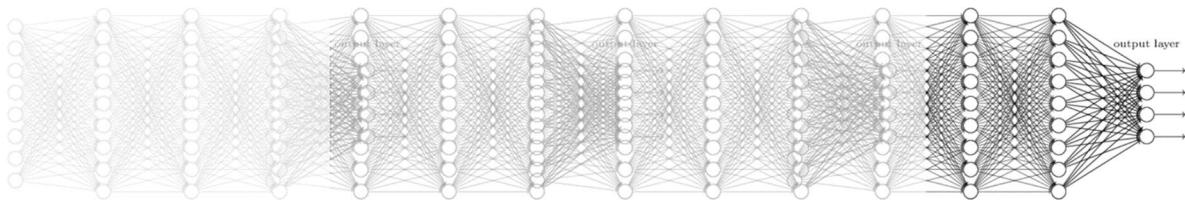
# Our hypothesis
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)
L7 = tf.sigmoid(tf.matmul(L6, W7) + b7)
L8 = tf.sigmoid(tf.matmul(L7, W8) + b8)
L9 = tf.sigmoid(tf.matmul(L8, W9) + b9)
L10 = tf.sigmoid(tf.matmul(L9, W10) + b10)

hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```

- 그런데 나타나는 저조한 결과

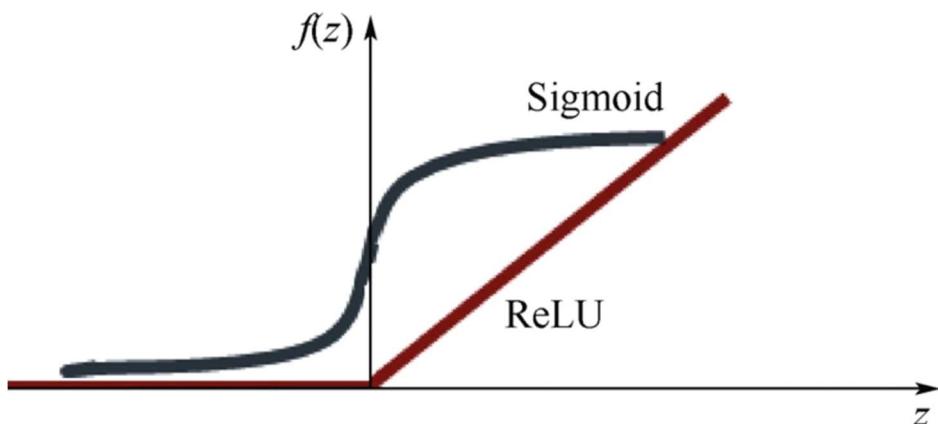
```
196000 [0.69314718, array([[ 0.49999988],
   [ 0.50000006],
   [ 0.49999982],
   [ 0.5        ]], dtype=float32)]
198000 [0.69314718, array([[ 0.49999988],
   [ 0.50000006],
   [ 0.49999982],
   [ 0.5        ]], dtype=float32)]
[array([[ 0.49999988],
   [ 0.50000006],
   [ 0.49999982],
   [ 0.5        ]], dtype=float32), array([[ 0.],
   [ 1.],
   [ 0.],
   [ 1.]], dtype=float32)]
Accuracy: 0.5
```

## ✓ Vanishing Gradient



- 그렇게 또 다시 Neural Network에는 침체기가 찾아온다

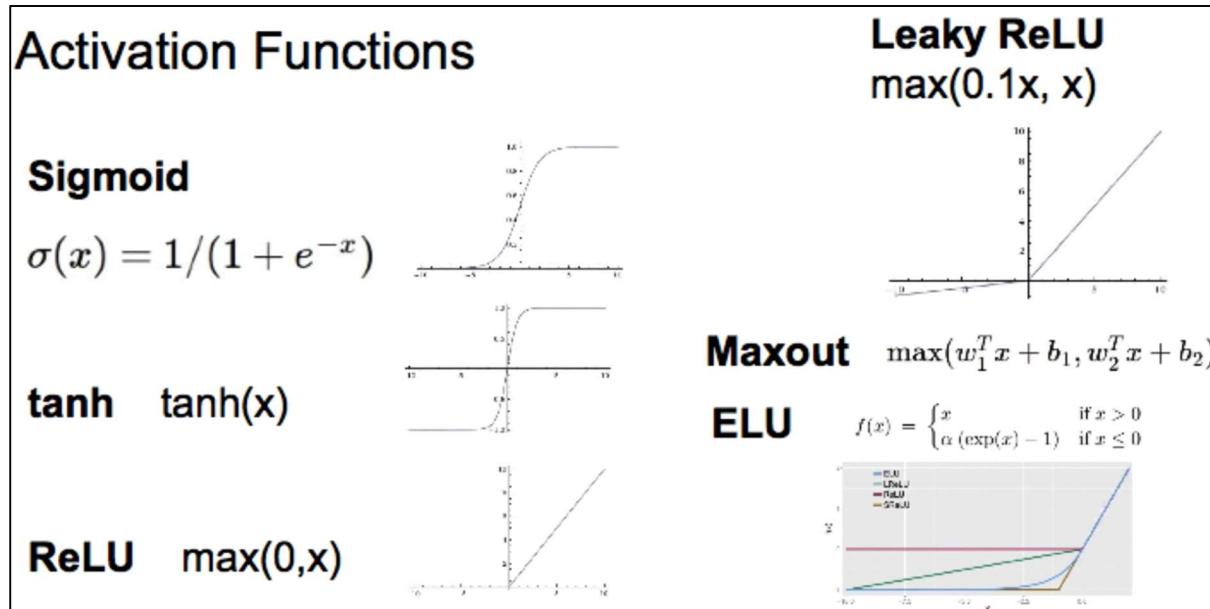
## ✓ ReLU : Rectified Linear Unit(0이하는 전부 제거)의 탄생



- 구현법

```
1 L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
196000 [2.6226094e-06, array([[ 2.59195826e-06],
   [ 9.9999642e-01],
   [ 9.99994874e-01],
   [ 2.43454133e-06]], dtype=float32)]
198000 [2.607708e-06, array([[ 2.55822852e-06],
   [ 9.9999642e-01],
   [ 9.99994874e-01],
   [ 2.40260101e-06]], dtype=float32),
[array([[ 2.52509381e-06],
   [ 9.9999642e-01],
   [ 9.99994874e-01],
   [ 2.37124709e-06]], dtype=float32), array([[ 0.],
   [ 1.],
   [ 1.],
   [ 0.]], dtype=float32)]
Accuracy: 1.0
```

- 다양한 ReLU의 변형식이 존재한다



- Sigmoid에 비해 확연히 차이나는 정확도

maxout	ReLU	VLReLU	tanh	Sigmoid
<b>93.94</b>	<b>92.11</b>	92.97	89.28	n/c
93.78	91.74	92.40	89.48	n/c
—	91.93	<b>93.09</b>	—	n/c
<b>91.75</b>	90.63	92.27	<b>89.82</b>	n/c
n/c†	90.91	92.43	89.54	n/c

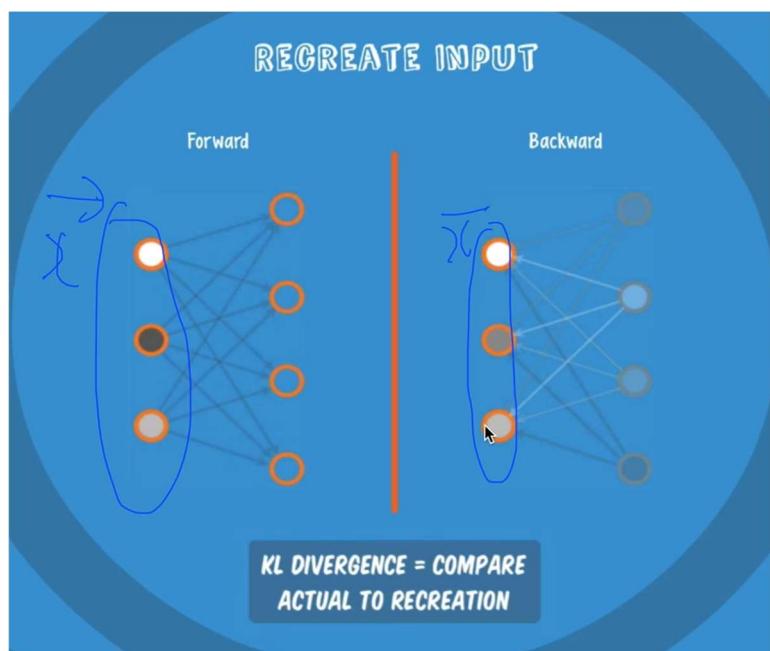
## ML lec 10-2

- ✓ Sigmoid가 안되는 이유 중 하나 : 잘못된 방식의 초기값 설정

- 제대로 두는 방법 :

  - Not all 0's
  - Challenging issue
  - Restricted Boatman Machine(RBM)

- ✓ RBM?



- encode, decode를 통해 값이 같아지도록 weight을 조절

✓ 굳이 복잡한 RBM 쓰지말고 Xavier/He Initialization을 써라

```
1 # Xavier initialization  
2 W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```

```
1 # He initialization  
2 W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

- 효과만점

Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	<b>93.94</b>	<b>92.11</b>	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	<b>93.09</b>	–	n/c
Xavier	91.75	90.63	92.27	<b>89.82</b>	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

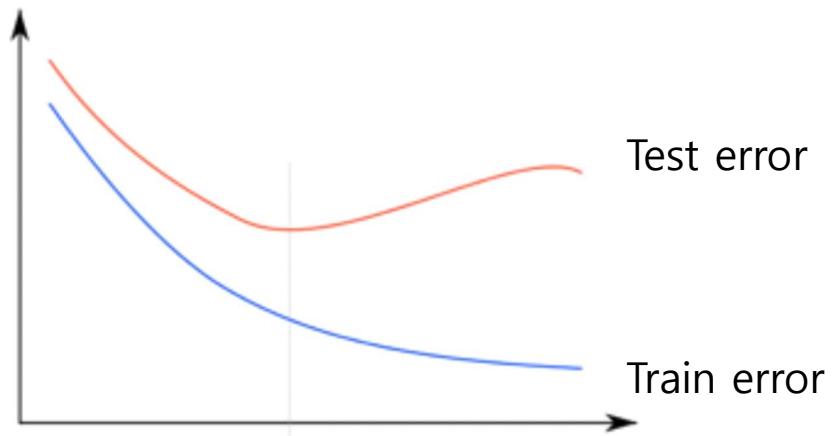
✓ 하지만 이 분야는 여전히 연구 중에 있다

- 따라서 여러 번 직접 시도해보는 것도 좋은 방법

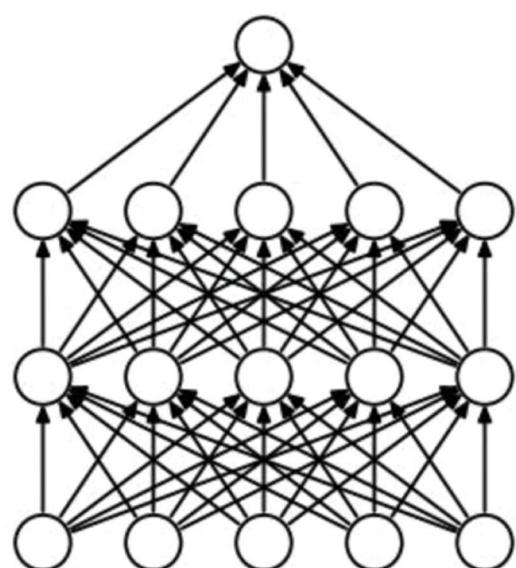
## ML lec 10-3

- ✓ 여전히 골칫거리인 Overfitting

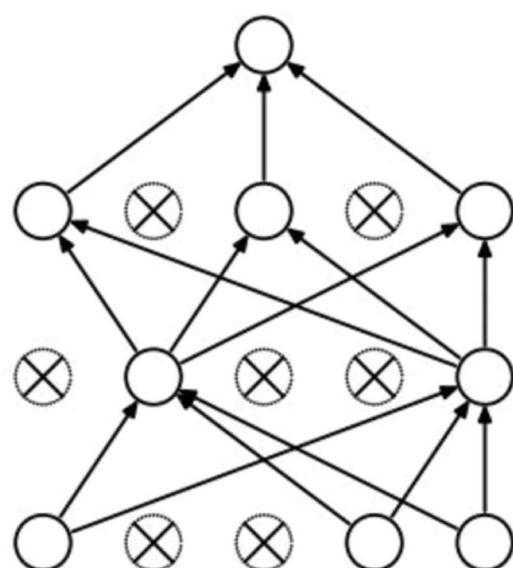
### Am I overfitting?



- 한 가지 방법으로 언급했던 Regularization( $l_2$ reg)
- ✓ NN에는 그 외의 방법이 더 있다
  - Dropout(그만두기)



(a) Standard Neural Net



(b) After applying dropout.

- 의외로 실행이 잘 된다

```

1 dropout_rate = tf.placeholder("float")
2 _L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), b1))
3 L1 = tf.nn.dropout(_L1, dropout_rate) # 보통 rate = 0.5

```

- 대신 Train, Evaluation할 땐 변경시켜 줘야 함

#### TRAIN:

```
1 sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys, dropout_rate: 0.7})
```

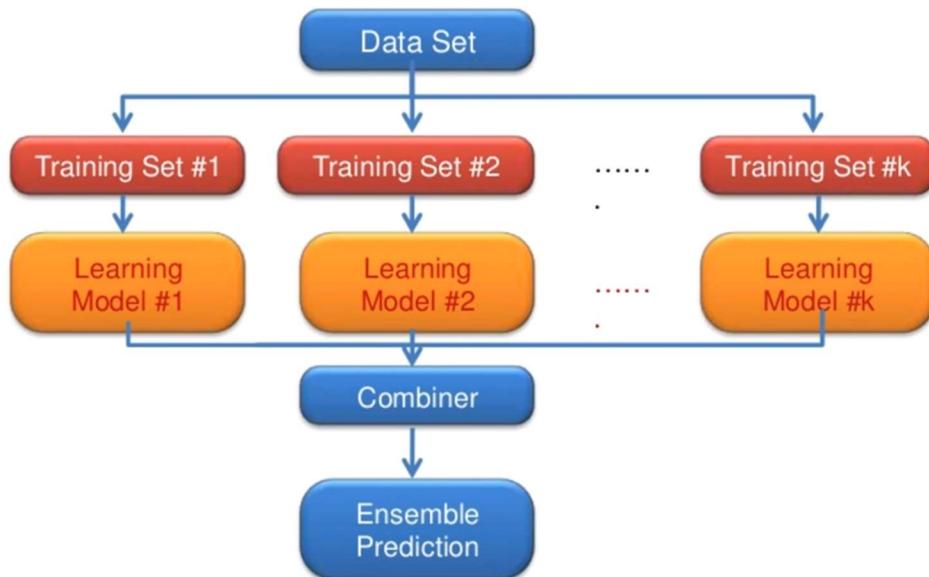
#### EVALUATION:

```

1 print "Accuracy:", accuracy.eval({X: mnist.test.images, Y:mnist.test.labels,
2                                     dropout_rate: 1})

```

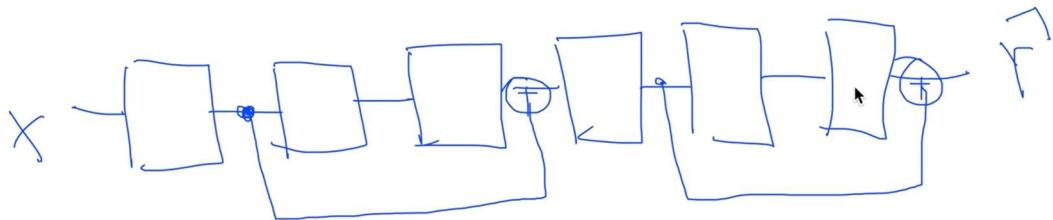
## ✓ Ensemble



- 사용하면 최대 4-5%까지 성능이 향상 됨

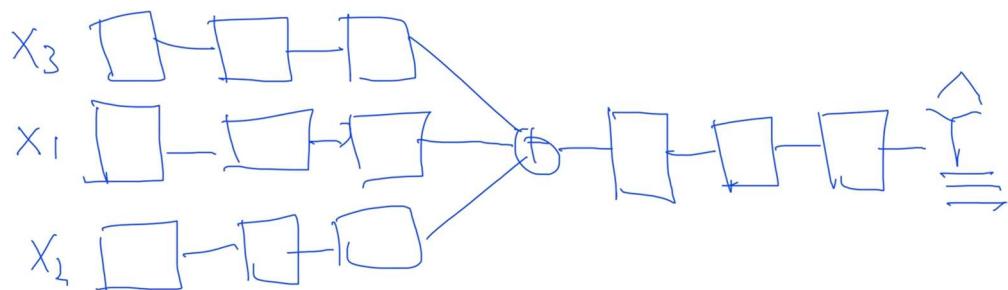
## ML lego 10-4

### ✓ Fast Forward

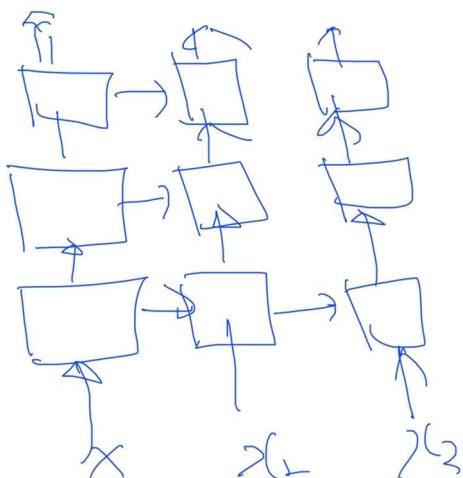


- 뽑아서 앞으로 옮김

### ✓ Split & Merge



### ✓ Recurrent Network(RNN)



- ✓ 이처럼 네트워크를 마음대로 레고처럼 조립할 수 있다

# ML lab 10

## ✓ MNIST for Softmax : 0.8951

The screenshot shows a Jupyter Notebook interface with two panes. The left pane is the 'Editor' containing Python code for training a softmax model on the MNIST dataset. The right pane is the 'IPython console' displaying the execution results.

**Editor - D:\Documents\ML\W\Number Recognition.py:**

```
1 # ML Lab 07-2
2 import tensorflow as tf
3 import matplotlib.pyplot as plt
4 import random
5
6 tf.set_random_seed(777) # for reproducibility
7
8 from tensorflow.examples.tutorials.mnist import input_data
9
10 # Check out https://www.tensorflow.org/get_started/mnist/beginners for
11 # more information about the mnist dataset
12 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
13
14 nb_classes = 10
15
16 # MNIST data image of shape 28 * 28 = 784
17 X = tf.placeholder(tf.float32, [None, 784])
18 # 0 - 9 digits recognition = 10 classes
19 Y = tf.placeholder(tf.float32, [None, nb_classes])
20
21 W = tf.Variable(tf.random_normal([784, nb_classes]))
22 b = tf.Variable(tf.random_normal([nb_classes]))
23
24 # Hypothesis (using softmax)
25 hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)
26
27 cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
28 train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
29
30 # Test model
31 is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
32 # Calculate accuracy
33 accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
34
35 # parameters
36 num_epochs = 15
37 batch_size = 100
38 num_iterations = int(mnist.train.num_examples / batch_size)
39
40 with tf.Session() as sess:
41     # Initialize TensorFlow variables
42     sess.run(tf.global_variables_initializer())
43     # Training cycle
44     for epoch in range(num_epochs):
45         avg_cost = 0
46
47         for i in range(num_iterations):
48             batch_xs, batch_ys = mnist.train.next_batch(batch_size)
49             _, cost_val = sess.run([train, cost], feed_dict={X: batch_xs, Y: batch_ys})
50             avg_cost += cost_val / num_iterations
51
52         print("Epoch: {:04d}, Cost: {:.9f}".format(epoch + 1, avg_cost))
53
54     print("Learning finished")
55
56 # Test the model. Using test set
```

**IPython console:**

```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default,
Type "copyright", "credits" or "license" for more
information
IPython 5.1.0 -- An enhanced Interactive Python.
?           --> Introduction and overview of IPython
%quickref --> Quick reference.
help       --> Python's own help system.
object?   --> Details about 'object', use 'object?'
```

In [1]:

```
In [1]: runfile('D:/Documents/ML/MNIST/Number Recognition.py', wdir='D:/Documents/ML/MNIST')
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Epoch: 0001, Cost: 2.826302660
Epoch: 0002, Cost: 1.061668948
Epoch: 0003, Cost: 0.838061307
Epoch: 0004, Cost: 0.733232732
Epoch: 0005, Cost: 0.669279880
Epoch: 0006, Cost: 0.624611828
Epoch: 0007, Cost: 0.591160339
Epoch: 0008, Cost: 0.563868978
Epoch: 0009, Cost: 0.541745167
Epoch: 0010, Cost: 0.522673571
Epoch: 0011, Cost: 0.506782322
Epoch: 0012, Cost: 0.492447640
Epoch: 0013, Cost: 0.479955830
Epoch: 0014, Cost: 0.468893666
Epoch: 0015, Cost: 0.458703479
Learning finished
Accuracy: 0.8951
Label: [8]
Prediction: [8]
```

A plot showing a handwritten digit '8' on a 28x28 grid, with axes ranging from 0 to 25.

## ✓ MNIST for NN : 0.8951 (이건 글씨체가 잘못했다)

The screenshot shows a Jupyter Notebook interface with several tabs:

- Editor - D:\Documents\ML\MNIST\Number Recognition\_Deeper Wider.py
- Number Recognition.py
- Number Recognition\_Deeper Wider.py
- Number Recognition\_ReLU.py
- IPython console
- Console 1/A

The IPython console tab displays the output of the script, which includes:

- Extracting MNIST\_data/train-images-idx3-ubyte.gz
- Extracting MNIST\_data/train-labels-idx1-ubyte.gz
- Extracting MNIST\_data/t10k-images-idx3-ubyte.gz
- Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz
- Epoch: 0001, Cost: 1.705338788
- Epoch: 0002, Cost: 0.739673085
- Epoch: 0003, Cost: 0.579599479
- Epoch: 0004, Cost: 0.494572282
- Epoch: 0005, Cost: 0.437653043
- Epoch: 0006, Cost: 0.396837625
- Epoch: 0007, Cost: 0.364425630
- Epoch: 0008, Cost: 0.338112633
- Epoch: 0009, Cost: 0.316955047
- Epoch: 0010, Cost: 0.297604143
- Epoch: 0011, Cost: 0.282338359
- Epoch: 0012, Cost: 0.267029888
- Epoch: 0013, Cost: 0.254337992
- Epoch: 0014, Cost: 0.242623737
- Epoch: 0015, Cost: 0.231251723
- Learning finished
- Accuracy: 0.8951
- Label: [5]
- Prediction: [9]

The Notebook also contains the following Python code:

```
1 # ML Lab 09-1
2 import tensorflow as tf
3 import matplotlib.pyplot as plt
4 import random
5
6 tf.set_random_seed(777) # for reproducibility
7
8 from tensorflow.examples.tutorials.mnist import input_data
9
10 # Check out https://www.tensorflow.org/get_started/mnist/beginners for
11 # more information about the mnist dataset
12 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
13
14 nb_classes = 10
15
16 # MNIST data image of shape 28 * 28 = 784
17 X = tf.placeholder(tf.float32, [None, 784])
18 # 0 - 9 digits recognition = 10 classes
19 Y = tf.placeholder(tf.float32, [None, nb_classes])
20
21 W1 = tf.Variable(tf.random_normal([784, 256]), name='weight1')
22 b1 = tf.Variable(tf.random_normal([256]), name='bias1')
23 layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
24
25 W2 = tf.Variable(tf.random_normal([256, 256]), name='weight2')
26 b2 = tf.Variable(tf.random_normal([256]), name='bias2')
27 layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)
28
29 W3 = tf.Variable(tf.random_normal([256, 256]), name='weight3')
30 b3 = tf.Variable(tf.random_normal([256]), name='bias3')
31 layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)
32
33 W4 = tf.Variable(tf.random_normal([256, nb_classes]), name='weight4')
34 b4 = tf.Variable(tf.random_normal([nb_classes]), name='bias4')
35
36 # Hypothesis (using softmax)
37 hypothesis = tf.nn.softmax(tf.matmul(layer3, W4) + b4)
38
39 cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
40 train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
41
42 # Test model
43 is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
44 # Calculate accuracy
45 accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
46
47 # parameters
48 num_epochs = 15
49 batch_size = 100
50 num_iterations = int(mnist.train.num_examples / batch_size)
51
52 with tf.Session() as sess:
53     # Initialize TensorFlow variables
54     sess.run(tf.global_variables_initializer())
55     # Training cycle
56     for epoch in range(num_epochs):
57         ave_cost = 0
```

Two plots are shown in the Notebook:

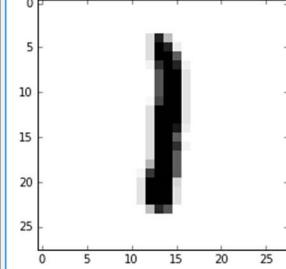
- A plot of a handwritten digit '8' on a 28x28 grid.
- A plot of a handwritten digit '5' on a 28x28 grid.

## ✓ MNIST for ReLU : 0.955!

Editor - D:\Documents\ML\MNIST\Number Recognition\_ReLU.py    IPython console

File Number Recognition.py    Number Recognition\_Deeper Wider.py    Number Recognition\_ReLU.py    In [3]: runfile('D:/Documents/ML/MNIST/Number Recognition\_ReLU.py', wdir='D:/Documents/ML/MNIST')  
Extracting MNIST\_data/train-images-idx3-ubyte.gz  
Extracting MNIST\_data/train-labels-idx1-ubyte.gz  
Extracting MNIST\_data/t10k-images-idx3-ubyte.gz  
Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz  
WARNING:tensorflow:From D:/Documents/ML/MNIST/Number Recognition\_ReLU.py:13: softmax\_cross\_entropy\_with\_logits (from tensorflow.python.ops.losses\_ops) is deprecated and will be removed in a future version.  
Instructions for updating:  
Future major versions of TensorFlow will allow gradients to flow into the labels input on backprop by default.  
See tf.nn.softmax\_cross\_entropy\_with\_logits\_v2.  
Epoch: 0001, Cost: 414.200646147  
Epoch: 0002, Cost: 74.375877354  
Epoch: 0003, Cost: 40.029818622  
Epoch: 0004, Cost: 29.177955821  
Epoch: 0005, Cost: 22.679409503  
Epoch: 0006, Cost: 19.757297865  
Epoch: 0007, Cost: 15.772126528  
Epoch: 0008, Cost: 13.670529195  
Epoch: 0009, Cost: 11.524907183  
Epoch: 0010, Cost: 8.961469026  
Epoch: 0011, Cost: 6.812244155  
Epoch: 0012, Cost: 6.444425043  
Epoch: 0013, Cost: 5.211466259  
Epoch: 0014, Cost: 3.662808030  
Epoch: 0015, Cost: 2.750344535  
Learning finished  
Accuracy: 0.955  
Label: [1]  
Prediction: [1]

In [4]: |



## ✓ MNIST for Xavier : 0.971!!!

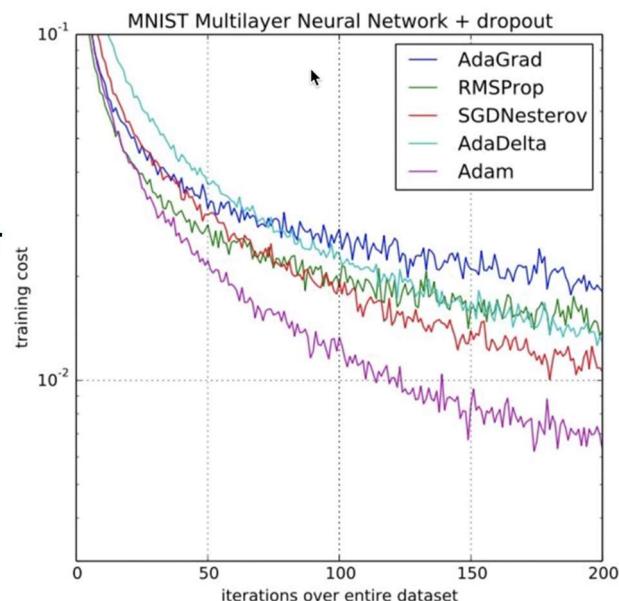
- Xavier은 초기값부터 바로 잘 찾음

The screenshot shows a Jupyter Notebook interface with two tabs: 'ecognition\_Deeper Wider.py' and 'Number Recognition\_ReLU.py'. The 'Number Recognition\_Xavier.py' tab is active, displaying Python code for a neural network. The code includes imports for TensorFlow, matplotlib, and random, and defines placeholders X and Y, variables W1 through W4 and b1 through b4, and a hypothesis function using softmax. It also includes a cost calculation using tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits) and a training step with AdamOptimizer. The 'In [4]' cell shows the command to run the file and the resulting training log from epoch 0001 to 0015, showing decreasing cost and learning finished at epoch 0010. The 'In [5]' cell shows a 28x28 pixel image of a handwritten digit '3'.

```

1 # ML Lab 09-1
2 import tensorflow as tf
3 import matplotlib.pyplot as plt
4 import random
5
6 tf.set_random_seed(777) # for reproducibility
7
8 from tensorflow.examples.tutorials.mnist import input_data
9
10 # Check out https://www.tensorflow.org/get_started/mnist/beginners for
11 # more information about the mnist dataset
12 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
13
14 nb_classes = 10
15
16 # MNIST data image of shape 28 * 28 = 784
17 X = tf.placeholder(tf.float32, [None, 784])
18 # 0 - 9 digits recognition = 10 classes
19 Y = tf.placeholder(tf.float32, [None, nb_classes])
20
21 W1 = tf.get_variable("W1", shape=[784, 256],
22                      initializer=tf.contrib.layers.xavier_initializer())
23 b1 = tf.Variable(tf.random_normal([256]), name='bias1')
24 layer1 = tf.nn.relu(tf.matmul(X, W1) + b1)
25
26 W2 = tf.get_variable("W2", shape=[256, 256],
27                      initializer=tf.contrib.layers.xavier_initializer())
28 b2 = tf.Variable(tf.random_normal([256]), name='bias2')
29 layer2 = tf.nn.relu(tf.matmul(layer1, W2) + b2)
30
31 W3 = tf.get_variable("W3", shape=[256, 256],
32                      initializer=tf.contrib.layers.xavier_initializer())
33 b3 = tf.Variable(tf.random_normal([256]), name='bias3')
34 layer3 = tf.nn.relu(tf.matmul(layer2, W3) + b3)
35
36 W4 = tf.get_variable("W4", shape=[256, nb_classes],
37                      initializer=tf.contrib.layers.xavier_initializer())
38 b4 = tf.Variable(tf.random_normal([nb_classes]), name='bias4')
39
40 # Hypothesis (using softmax)
41 hypothesis = tf.matmul(layer3, W4) + b4 # 'tf.nn.softmax' deleted.
42
43 cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
44    logits=hypothesis, labels=Y))
45 train = tf.train.AdamOptimizer(learning_rate=0.01).minimize(cost) # Gradient -> Adam
46
47 # Test model
48 is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
49 # Calculate accuracy
50 accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
51
52 # parameters
53 num_epochs = 15
54 batch_size = 100
55 num_iterations = int(mnist.train.num_examples / batch_size)
56
57 with tf.Session() as sess:
58     # Initialize TensorFlow variables
59     sess.run(tf.global_variables_initializer())
60     # Training cycle
61     for epoch in range(num_epochs):

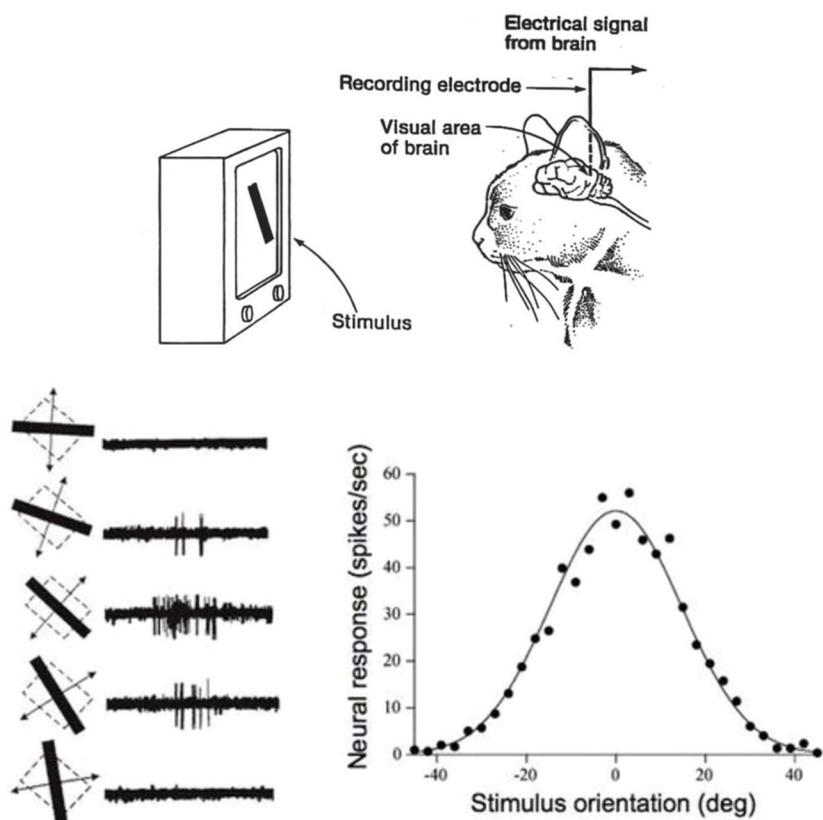
```



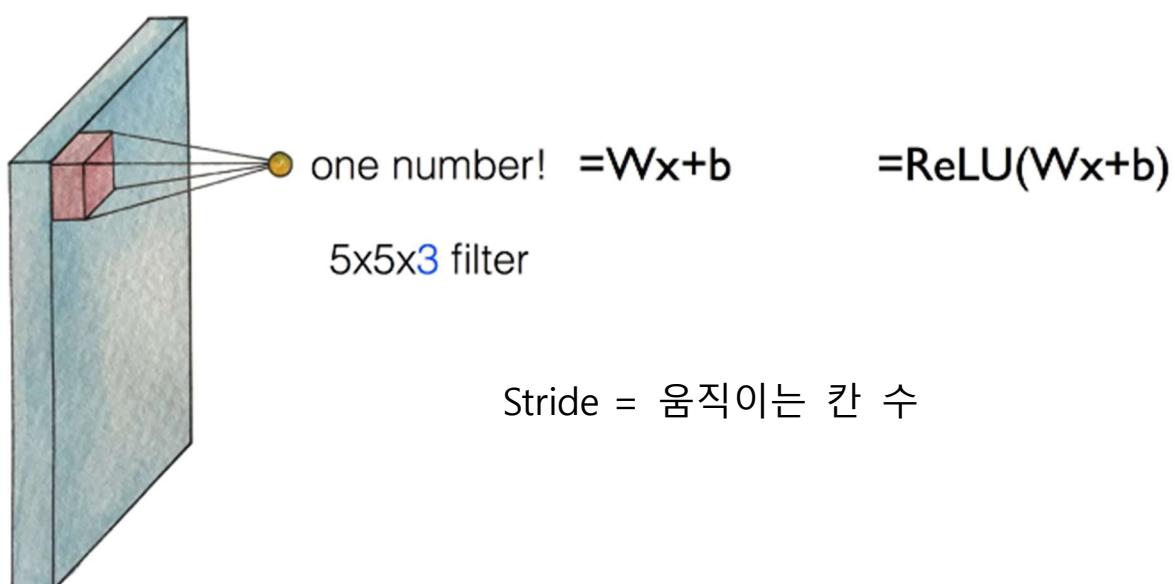
+ AdamOptimizer의 성능이 좋다

## ML lec 11-1

- ✓ CNN의 시초 : 고양이 실험

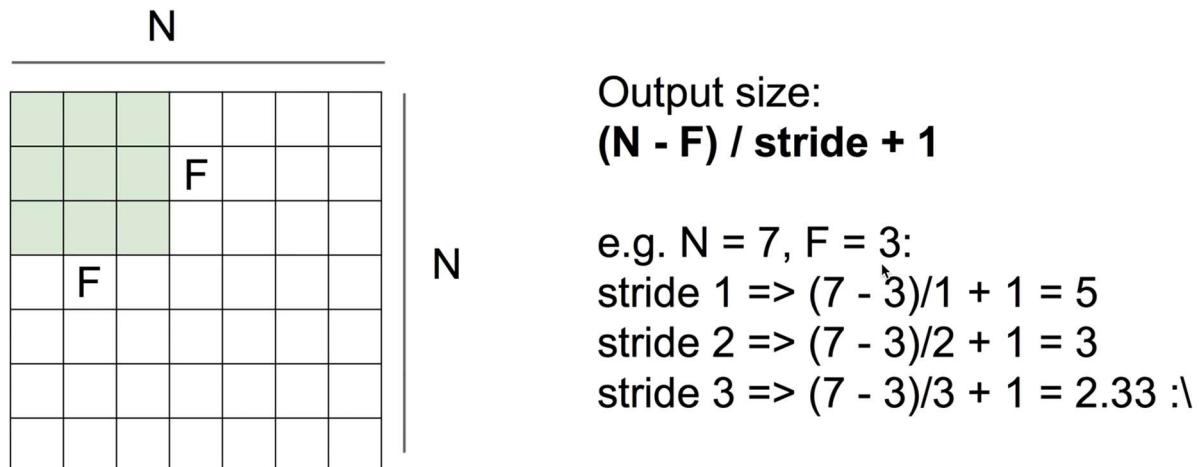


- ✓ Filter(빨간 블록)를 사용해서 하나의 숫자를 추출하자

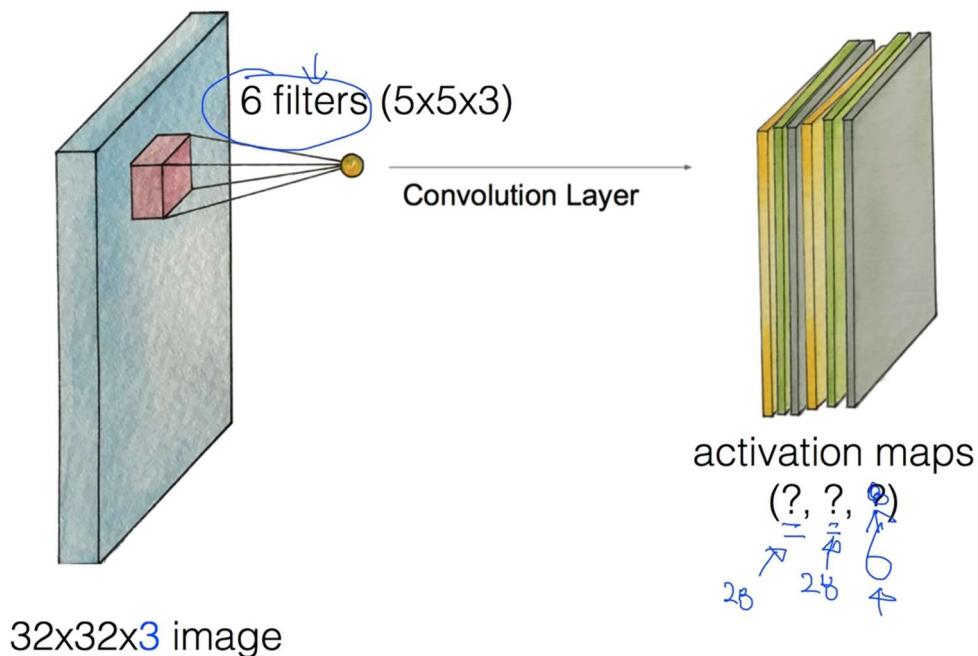


- 이것을 반복하면 몇 개의 숫자를 얻을 수 있을까?

✓ 다음과 같이 계산할 수 있다



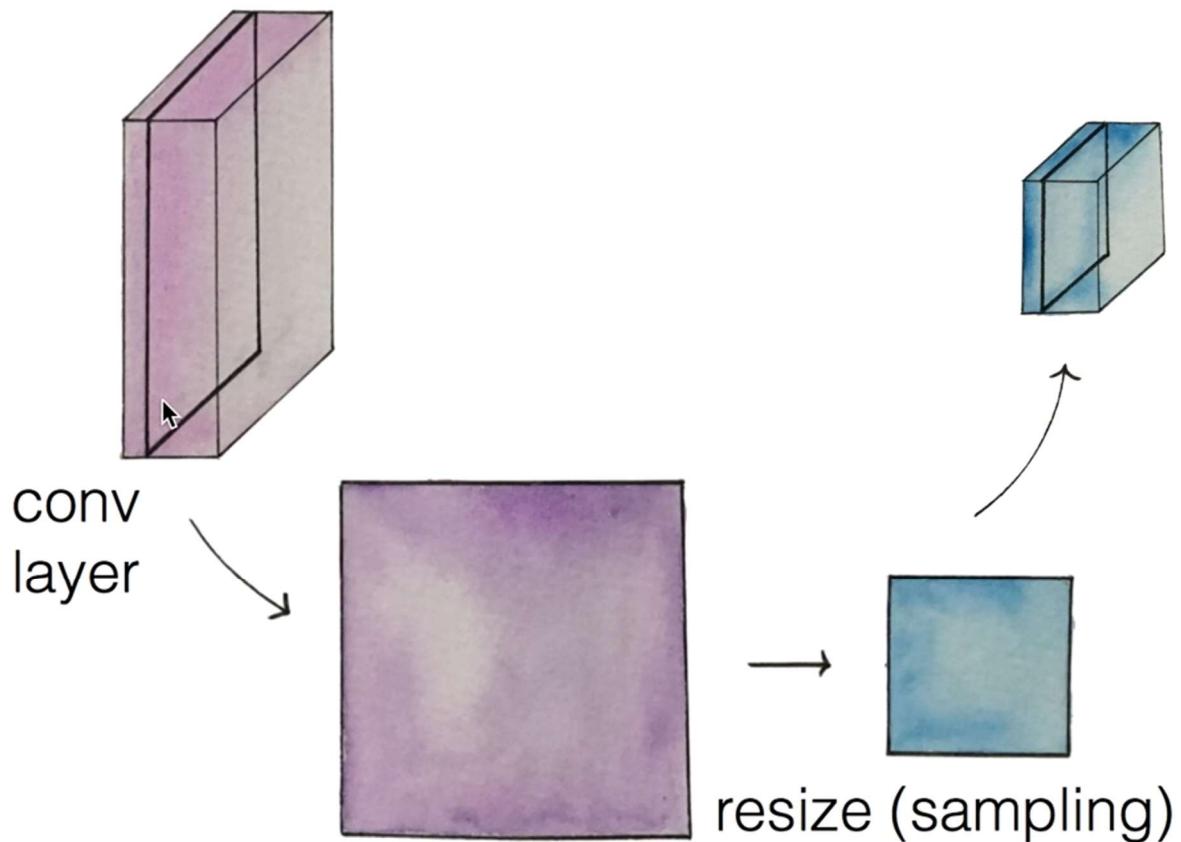
- 대신 결과값은 정수가 나와야 한다
- ✓ 이런 식으로 필터들을 통해 activation maps를 만들 수 있다



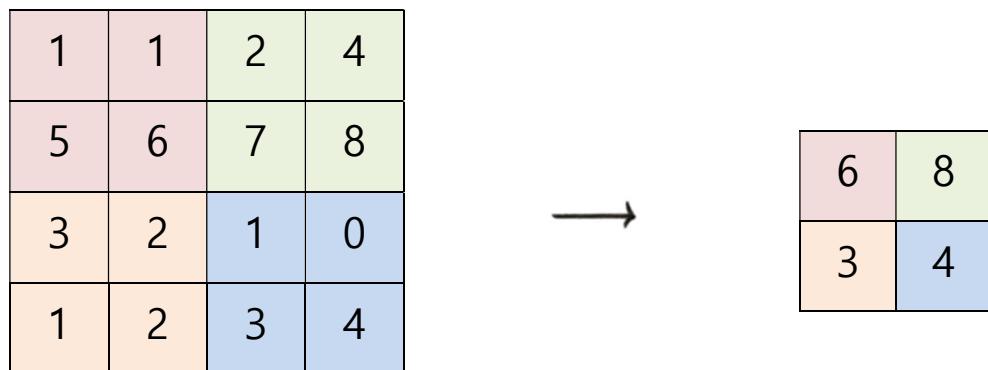
$$(32 - 5)/1 + 1 = 28$$

## ML lec 11-2

- ✓ **Pooling layer(sampling)** : 한 레이어만 뽑아 내

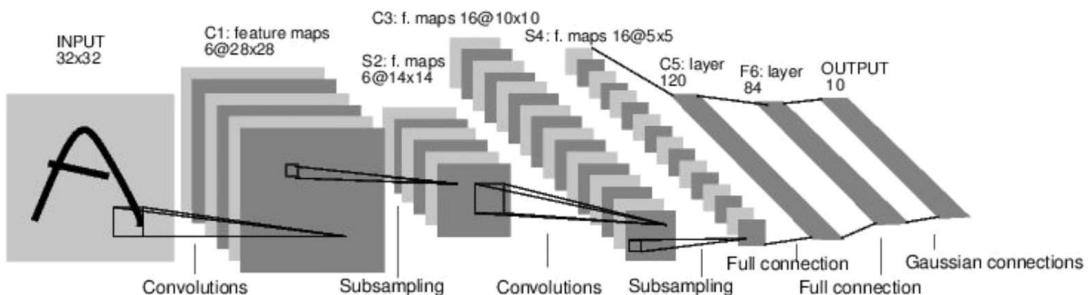


- 그 후에 Max Pooling(2x2 filters and stride 2로)



# ML lec 11-3

## ✓ ConvNet의 활용 예

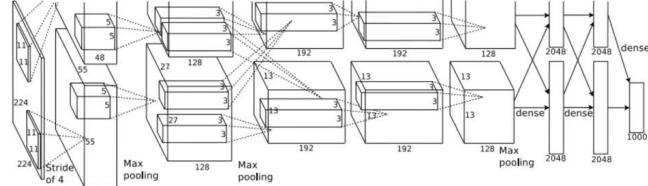


Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2x2 applied at stride 2  
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

## Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

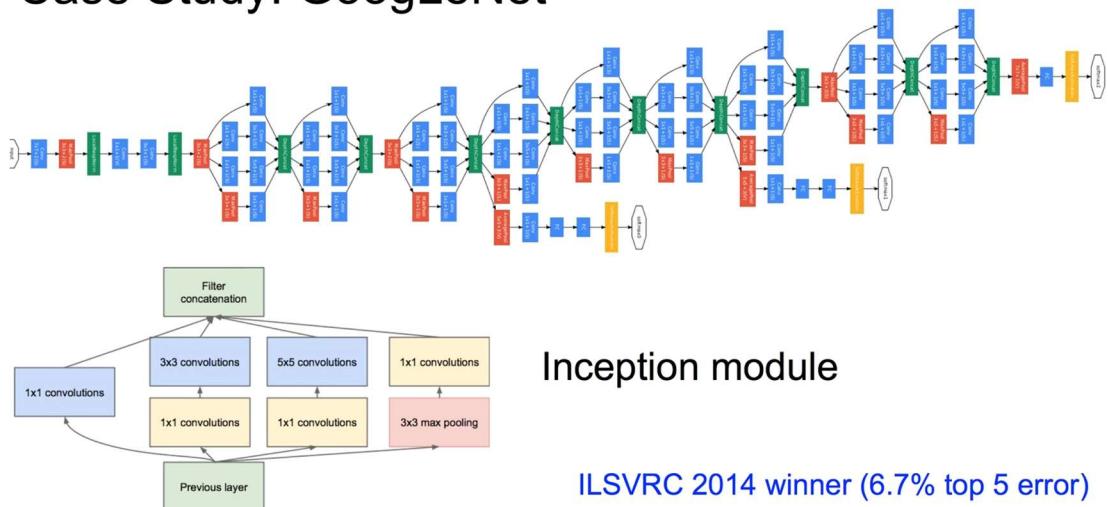
[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)

## Case Study: GoogLeNet

[Szegedy et al., 2014]



## Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)

Microsoft Research

### MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
  - ImageNet Detection: **16%** better than 2nd
  - ImageNet Localization: **27%** better than 2nd
  - COCO Detection: **11%** better than 2nd
  - COCO Segmentation: **12%** better than 2nd

\*Improvements are relative numbers

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

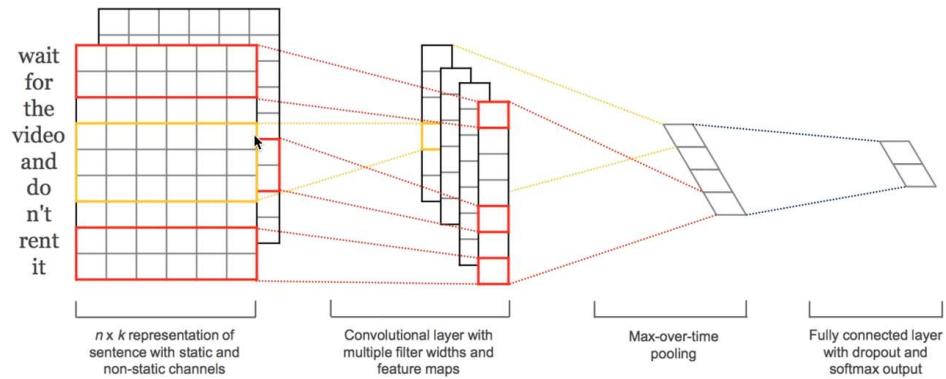
ICCV15

Fast Forward 사용

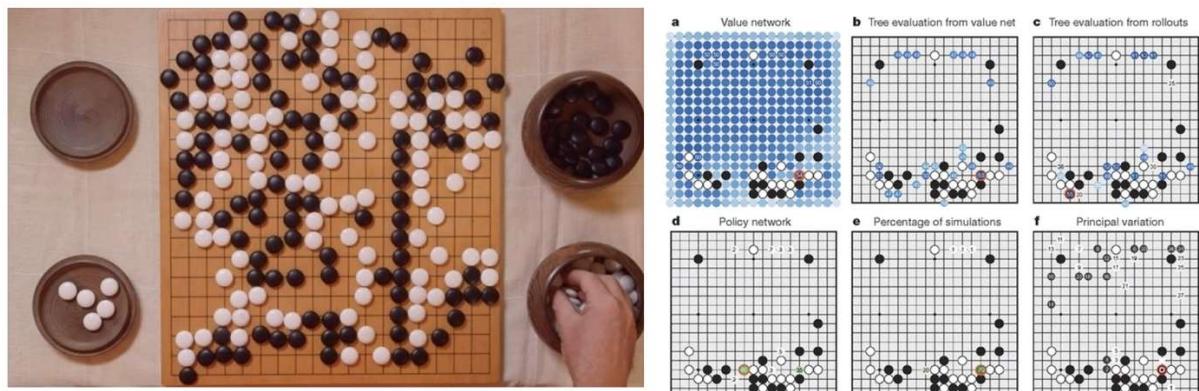
Slide from Kaiming He's recent presentation <https://www.youtube.com/watch?v=1PGLj-uKT1w>

# Convolutional Neural Networks for Sentence Classification

[Yoon Kim, 2014]

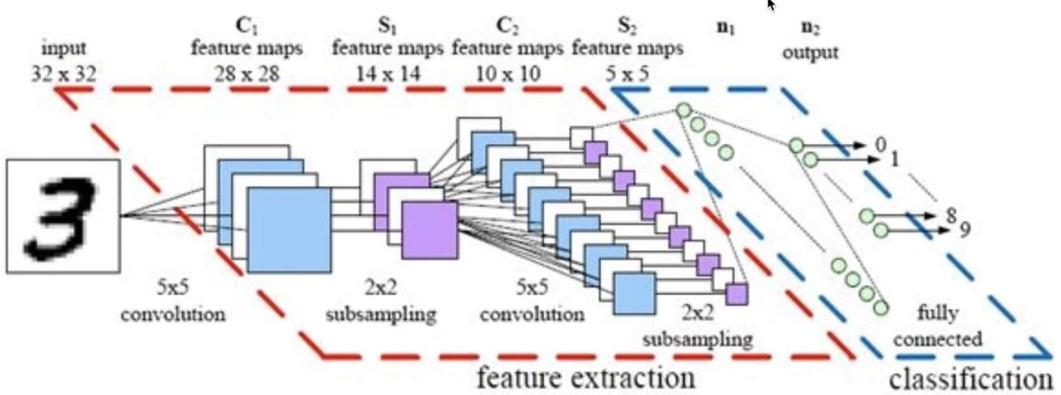


## Case Study Bonus: DeepMind's AlphaGo



## ML lab 11-1

- ✓ CNN의 구조



- 구현

```

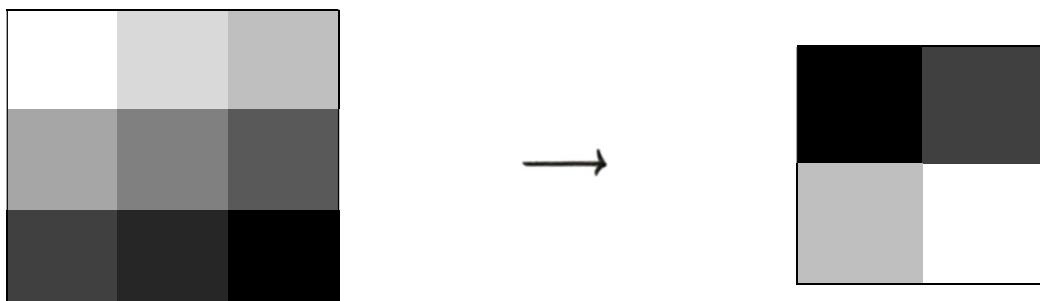
1 sess = tf.InteractiveSession()
2 image = np.array([[[[1],[2],[3]],
3                   [[4],[5],[6]],
4                   [[7],[8],[9]]]], dtype=np.float32)
5 print(image.shape)
6 plt.imshow(image.reshape(3,3), cmap='Greys')

```

```

# print("image:\n", image)
print("image.shape", image.shape)
weight = tf.constant([[[[1.],[[1.]],
                      [[1.],[[1.]]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(2,2))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray')

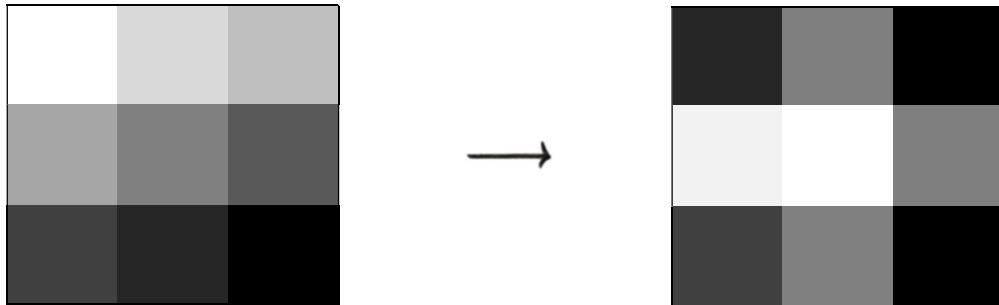
```



## ✓ Padding

```
# print("image:\n", image)
print("image.shape", image.shape)

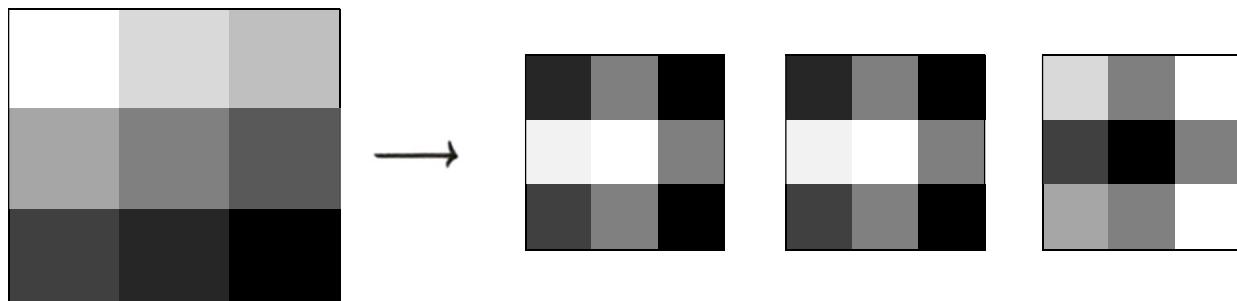
weight = tf.constant([[[[1.]], [[1.]]],  
                     [[[1.]], [[1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```



- 물론 filter가 하나일 필요는 없다(3개)

```
# print("image:\n", image)
print("image.shape", image.shape)

weight = tf.constant([[[[1.,10.,-1.]], [[1.,10.,-1.]]],  
                     [[[1.,10.,-1.]], [[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```



## ✓ Max Pooling

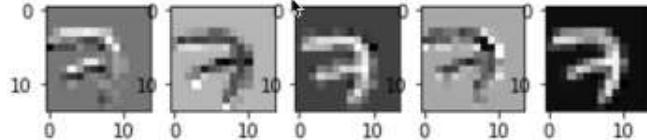
```
1 image = np.array([[[[4],[3]],
2                   [[2],[1]]]], dtype=np.float32)
3 pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
4                       strides=[1, 1, 1, 1], padding='SAME')
5 print(pool.shape) # padding='SAME': L1 출력값은 입력의 이미지 사이즈와 같다
6 print(pool.eval())
```

## ✓ MNIST Convolution Layer

```
sess = tf.InteractiveSession()

img = img.reshape(-1,28,28,1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 5], stddev=0.01))
conv2d = tf.nn.conv2d(img, W1, strides=[1, 2, 2, 1], padding='SAME')
print(conv2d)
sess.run(tf.global_variables_initializer())
conv2d_img = conv2d.eval()
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(14,14), cmap='gray')

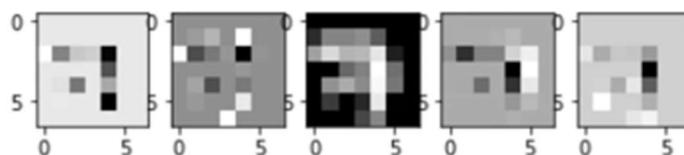
Tensor("Conv2D_1:0", shape=(1, 14, 14, 5), dtype=float32)
```



## ✓ MNIST Max Pooling

```
pool = tf.nn.max_pool(conv2d, ksize=[1, 2, 2, 1], strides=[
1, 2, 2, 1], padding='SAME')
print(pool)
sess.run(tf.global_variables_initializer())
pool_img = pool.eval()
pool_img = np.swapaxes(pool_img, 0, 3)
for i, one_img in enumerate(pool_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(7, 7), cmap='gray')

Tensor("MaxPool_2:0", shape=(1, 7, 7, 5), dtype=float32)
```



# ML lab 11-2

## ✓ MNIST with CNN : 0.9931

The screenshot shows a Jupyter Notebook interface with two panes. The left pane is the code editor containing the Python script `Number Recognition_CNN.py`. The right pane is the IPython console showing the execution of the code.

**Code Editor (Left):**

```
Editor - D:\Documents\ML\Number Recognition_CNN.py
Number Recognition_CNN.py
8 tf.set_random_seed(777) # reproducibility
9
10 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
11 # Check out https://www.tensorflow.org/get_started/mnist/beginners for
12 # more information about the mnist dataset
13
14 # hyper parameters
15 learning_rate = 0.001
16 training_epochs = 15
17 batch_size = 100
18
19 # dropout (keep_prob) rate 0.7-0.5 on training, but should be 1 for testing
20 keep_prob = tf.placeholder(tf.float32)
21
22 # input place holders
23 X = tf.placeholder(tf.float32, [None, 784])
24 X_img = tf.reshape(X, [-1, 28, 28, 1]) # img 28x28x1 (black/white)
25 Y = tf.placeholder(tf.float32, [None, 10])
26
27 # L1 ImgIn shape=(?, 28, 28, 1)
28 W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
29 #    Conv      -> (?, 28, 28, 32)
30 #    Pool      -> (?, 14, 14, 32)
31 L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
32 L1 = tf.nn.relu(L1)
33 L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
34                      strides=[1, 2, 2, 1], padding='SAME')
35 L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
36 ...
37 Tensor("Conv2D_0", shape=(?, 28, 28, 32), dtype=float32)
38 Tensor("Relu_0", shape=(?, 28, 28, 32), dtype=float32)
39 Tensor("MaxPool_0", shape=(?, 14, 14, 32), dtype=float32)
40 Tensor("dropout/mul_0", shape=(?, 14, 14, 32), dtype=float32)
41 ...
42
43 # L2 ImgIn shape=(?, 14, 14, 32)
44 W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
45 #    Conv      -> (?, 14, 14, 64)
46 #    Pool      -> (?, 7, 7, 64)
47 L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
48 L2 = tf.nn.relu(L2)
49 L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
50                      strides=[1, 2, 2, 1], padding='SAME')
51 L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
52 ...
53 Tensor("Conv2D_1", shape=(?, 14, 14, 64), dtype=float32)
54 Tensor("Relu_1", shape=(?, 14, 14, 64), dtype=float32)
55 Tensor("MaxPool_1", shape=(?, 7, 7, 64), dtype=float32)
56 Tensor("dropout_1/mul_0", shape=(?, 7, 7, 64), dtype=float32)
57 ...
58
59 # L3 ImgIn shape=(?, 7, 7, 64)
60 W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
61 #    Conv      -> (?, 7, 7, 128)
62 #    Pool      -> (?, 4, 4, 128)
63 #    Reshape   -> (?, 4 * 4 * 128) # Flatten them for FC
64 L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
65 L3 = tf.nn.relu(L3)
66 L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
67                      padding='SAME')
68 L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
69 L3.flat = tf.reshape(L3, [-1, 128 * 4 * 4])
```

**IPython Console (Right):**

```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul 5 2016, 11:41)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details

In [1]:
In [1]: runfile('D:/Documents/ML/MNIST/Number Recognition_CNN.py', wdir='D:/Documents/ML/MNIST')
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From D:/Documents/ML/MNIST/Number Recognition_CNN.py:16: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow into the labels input on backprop by default.

See tf.nn.softmax_cross_entropy_with_logits_v2.

Learning started. It takes sometime.
Epoch: 0001 cost = 0.371536119
Epoch: 0002 cost = 0.100174467
Epoch: 0003 cost = 0.073716033
Epoch: 0004 cost = 0.061241074
Epoch: 0005 cost = 0.055366139
Epoch: 0006 cost = 0.047262605
Epoch: 0007 cost = 0.043236403
Epoch: 0008 cost = 0.040658078
Epoch: 0009 cost = 0.037321034
Epoch: 0010 cost = 0.035606477
Epoch: 0011 cost = 0.032770761
Epoch: 0012 cost = 0.030496861
Epoch: 0013 cost = 0.029026200
Epoch: 0014 cost = 0.028000729
Epoch: 0015 cost = 0.024971437
Learning Finished!
Accuracy: 0.9931
Label: [2]
Prediction: [2]

In [2]:
```

## ML lab 11-3

- ✓ `tf.layers` : 한 줄로 간단하게 정리 가능

```
# Convolutional Layer #1
conv1 = tf.layers.conv2d(inputs=X_img,filters=32,kernel_size=[3,3],padding="SAME",activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], padding="SAME", strides=2)
dropout1 = tf.layers.dropout(inputs=pool1,rate=0.7, training=self.training)

# Convolutional Layer #2
conv2 = tf.layers.conv2d(inputs=dropout1,filters=64,kernel_size=[3,3],padding="SAME",activation=tf.nn.relu)
...
flat = tf.reshape(dropout3, [-1, 128 * 4 * 4])
dense4 = tf.layers.dense(inputs=flat, units=625, activation=tf.nn.relu)
dropout4 = tf.layers.dropout(inputs=dense4, rate=0.5, training=self.training)
...
```

- ✓ MNIST with Class & Ensemble : **0.9947**

Editor - D:\Documents\ML\MNIST\Number Recognition\_Engsemble.py

Number Recognition\_Engsemble.py

```

16 training_epochs = 20
17 batch_size = 100
18
19
20 class Model:
21
22     def __init__(self, sess, name):
23         self.sess = sess
24         self.name = name
25         self._build_net()
26
27     def _build_net(self):
28         with tf.variable_scope(self.name):
29             # dropout (keep_prob) rate 0.7~0.5 on training, but should be 1
30             # for testing
31             self.training = tf.placeholder(tf.bool)
32
33             # input place holders
34             self.X = tf.placeholder(tf.float32, [None, 784])
35
36             # img 28x28x1 (black/white), Input Layer
37             X_img = tf.reshape(self.X, [-1, 28, 28, 1])
38             self.Y = tf.placeholder(tf.float32, [None, 10])
39
40             # Convolutional Layer #1
41             conv1 = tf.layers.conv2d(inputs=X_img, filters=32, kernel_size=[3, 3],
42                                    padding="SAME", activation=tf.nn.relu)
43             # Pooling Layer #1
44             pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
45                                             padding="SAME", strides=2)
46             dropout1 = tf.layers.dropout(inputs=pool1,
47                                         rate=0.3, training=self.training)
48
49             # Convolutional Layer #2 and Pooling Layer #2
50             conv2 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3],
51                                    padding="SAME", activation=tf.nn.relu)
52             pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
53                                             padding="SAME", strides=2)
54             dropout2 = tf.layers.dropout(inputs=pool2,
55                                         rate=0.3, training=self.training)
56
57             # Convolutional Layer #3 and Pooling Layer #3
58             conv3 = tf.layers.conv2d(inputs=dropout2, filters=128, kernel_size=[3, 3],
59                                    padding="SAME", activation=tf.nn.relu)
50             pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2],
61                                    padding="SAME", strides=2)
62             dropout3 = tf.layers.dropout(inputs=pool3,
63                                         rate=0.3, training=self.training)
64
65             # Dense Layer with Relu
66             flat = tf.reshape(dropout3, [-1, 128 * 4 * 4])
67             dense4 = tf.layers.dense(inputs=flat,
68                                    units=625, activation=tf.nn.relu)
69             dropout4 = tf.layers.dropout(inputs=dense4,
70                                         rate=0.5, training=self.training)
71
72             # Logits (no activation) Layer: L5 Final FC 625 inputs -> 10 outputs
73             self.logits = tf.layers.dense(inputs=dropout4, units=10)
74
75             # define cost/loss & optimizer
76             self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
77                             logits=self.logits, labels=self.Y))

```

In [3]: runfile('D:/Documents/ML/MNIST/Number Recog', 1)

Extracting MNIST\_data/train-images-idx3-ubyte.gz  
Extracting MNIST\_data/train-labels-idx1-ubyte.gz  
Extracting MNIST\_data/t10k-images-idx3-ubyte.gz  
Extracting MNIST\_data/t10k-labels-idx1-ubyte.gz  
Learning Started!

Epoch: 0001 cost = [0.28760711 0.27969753]  
Epoch: 0002 cost = [0.08866234 0.08411827]  
Epoch: 0003 cost = [0.0673912 0.06638912]  
Epoch: 0004 cost = [0.05474671 0.05475383]  
Epoch: 0005 cost = [0.04744834 0.04691222]  
Epoch: 0006 cost = [0.04609834 0.04257706]  
Epoch: 0007 cost = [0.04005742 0.04008050]  
Epoch: 0008 cost = [0.03795883 0.03850874]  
Epoch: 0009 cost = [0.03355184 0.03483846]  
Epoch: 0010 cost = [0.03436217 0.03329869]  
Epoch: 0011 cost = [0.03122814 0.03083786]  
Epoch: 0012 cost = [0.03109886 0.02964267]  
Epoch: 0013 cost = [0.02744721 0.02809541]  
Epoch: 0014 cost = [0.02813654 0.0291832]  
Epoch: 0015 cost = [0.0269899 0.02644391]  
Epoch: 0016 cost = [0.02588537 0.02502248]  
Epoch: 0017 cost = [0.02367517 0.02500959]  
Epoch: 0018 cost = [0.02291008 0.02542166]  
Epoch: 0019 cost = [0.02393062 0.02175492]  
Epoch: 0020 cost = [0.02276768 0.02220519]  
Learning Finished!  
0 Accuracy: 0.9945  
1 Accuracy: 0.9928  
Ensemble accuracy: 0.9947

In [4]: |