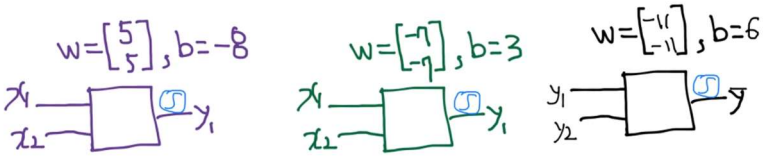
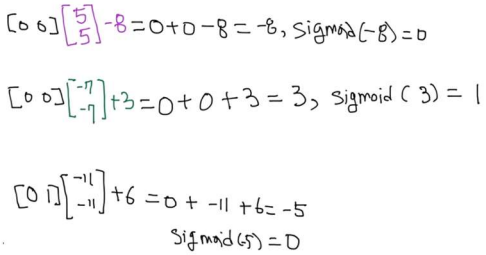
ML lec 09-1

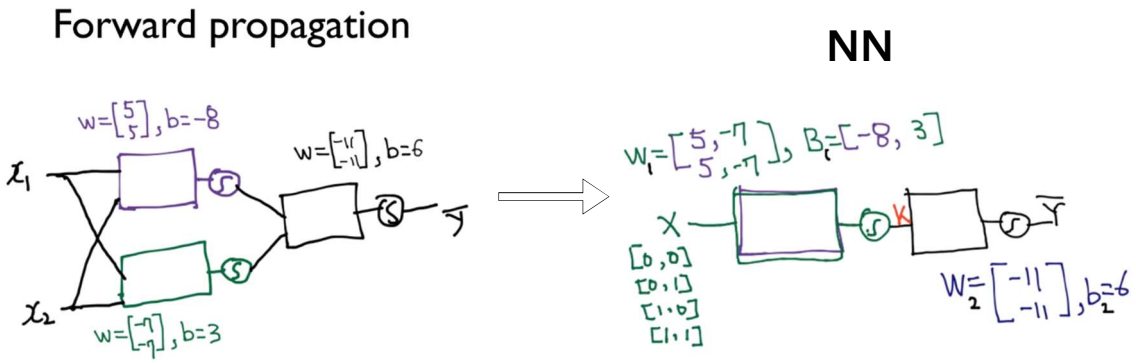
✓ 그 동안 풀지 못했던 XOR 난제를 Sigmoid로 풀게 됨





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

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



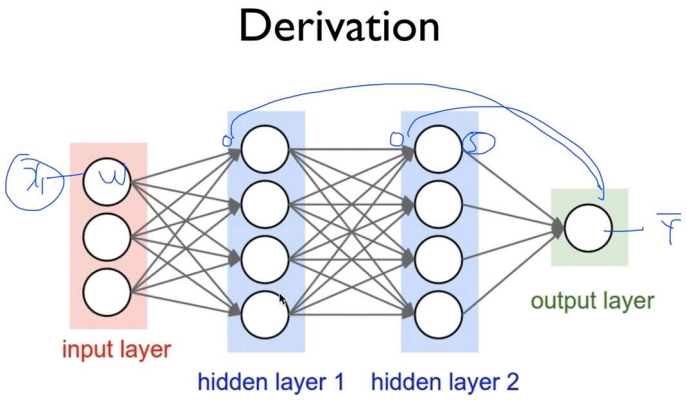
- TF로의 구현법

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

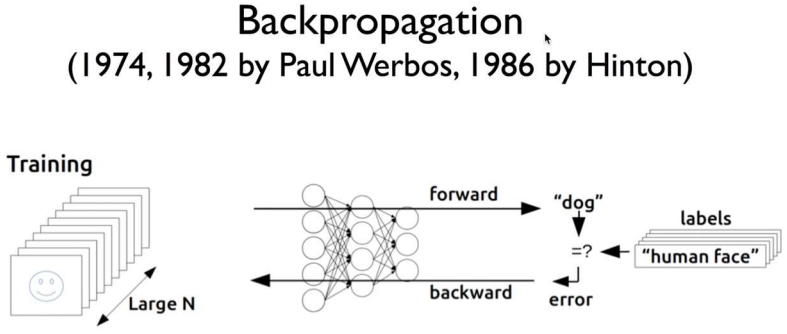
ML lec 09-2

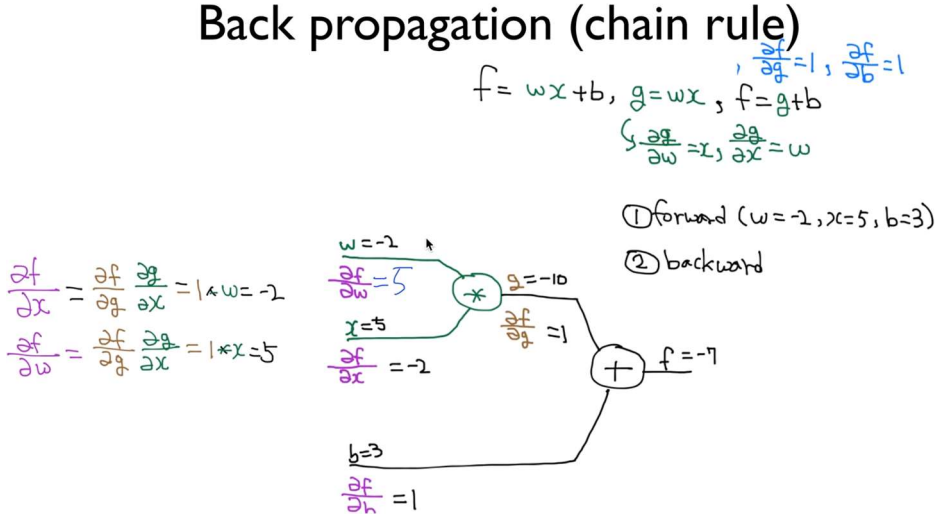
✓ **Cost 함수에 경사하강법을 적용하기 위해서는 미분이 필요함**

- 하지만 NN이 되면서 미분하기엔 복잡해짐

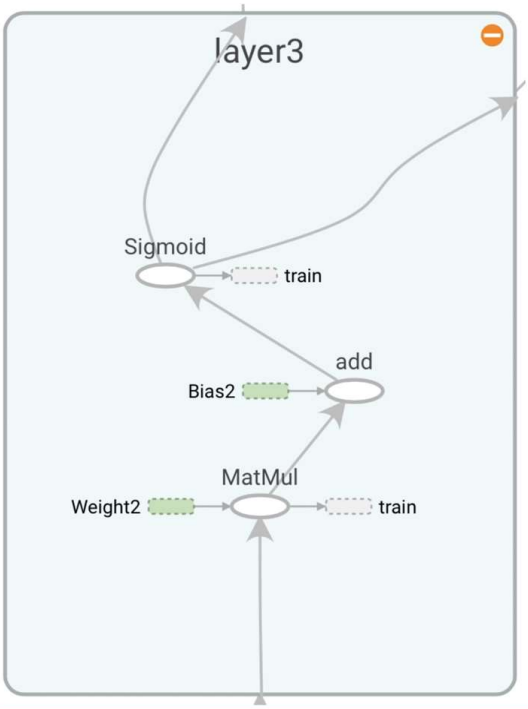


✓ **Backpropagation이 해결할 수 있다**



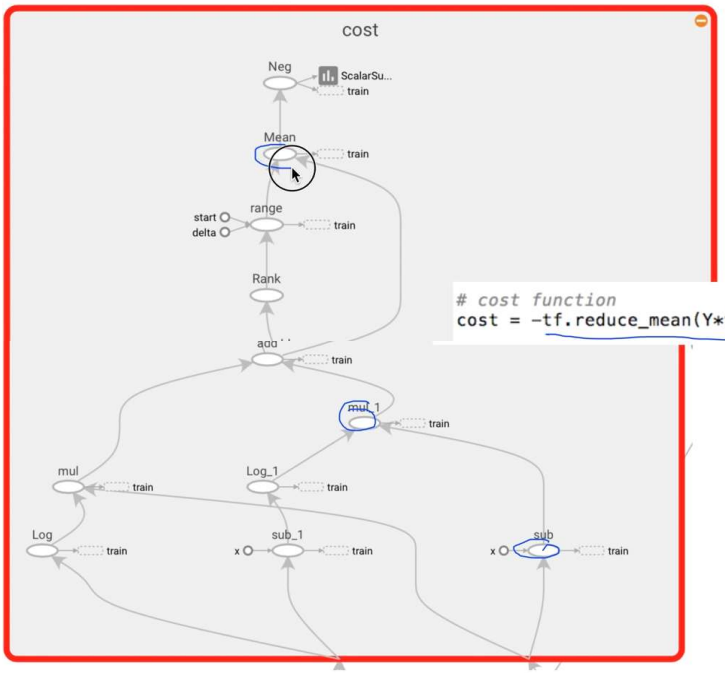


· 그 Backpropagation의 과정은 TensorFlow의 기능 중 하나인 TensorBoard를 통해 확인할 수 있다



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-hypothesi
```

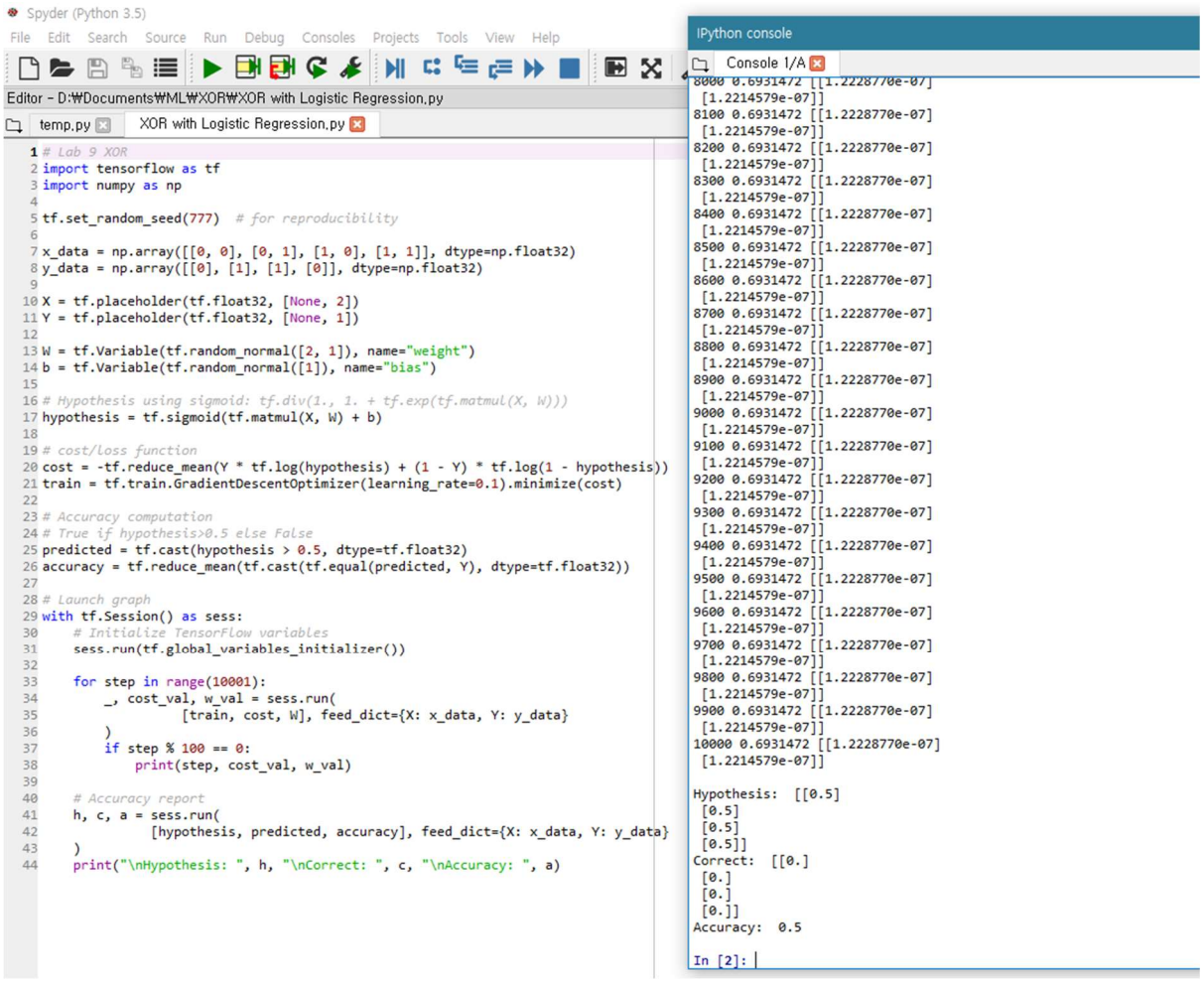
ML lab 09-1

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

✓ XOR을 TF로 나타내기

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

```

✓ XOR with Logistic Regression

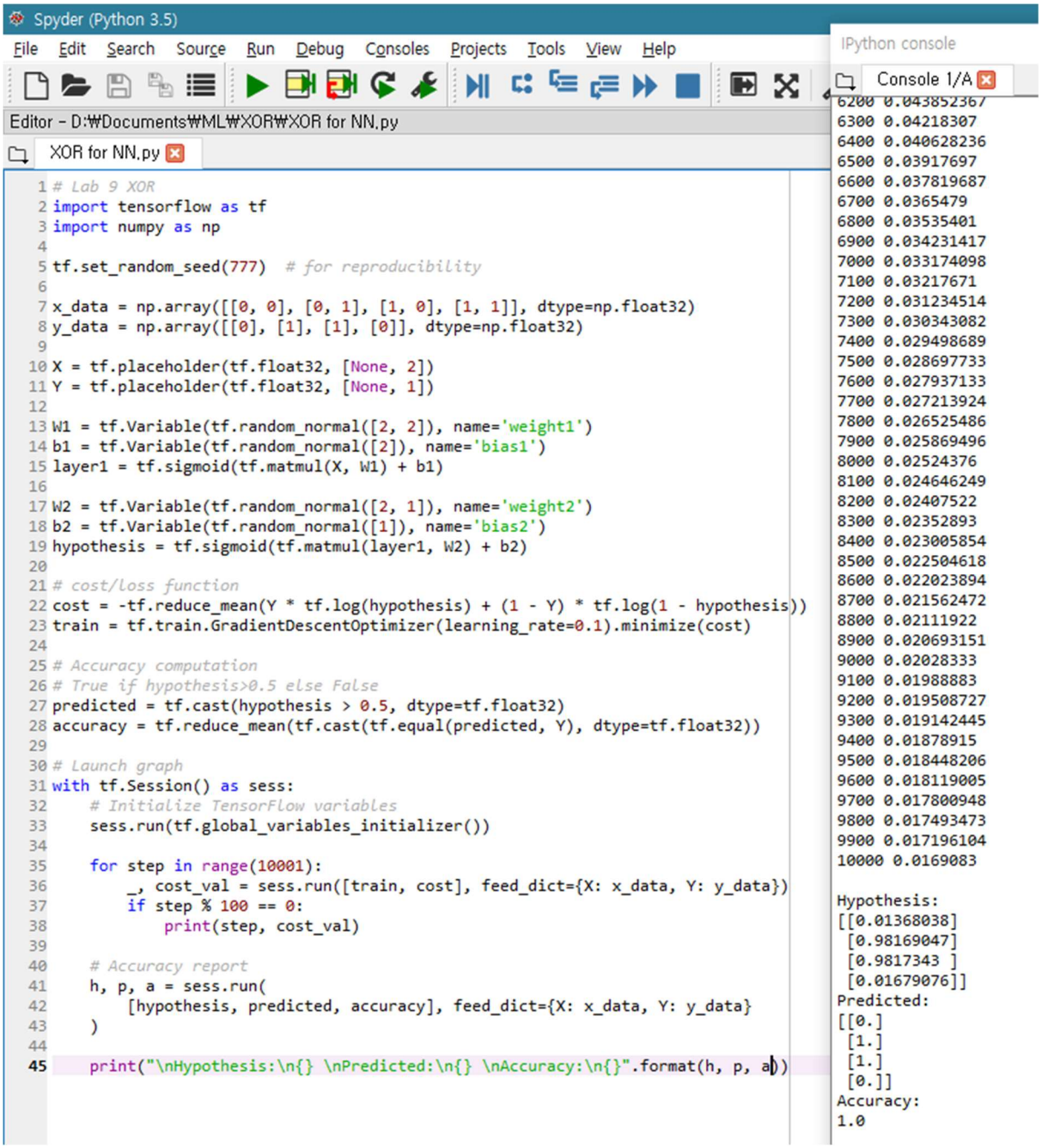


- 알던대로 작동하지 않는다

✓ 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)
```

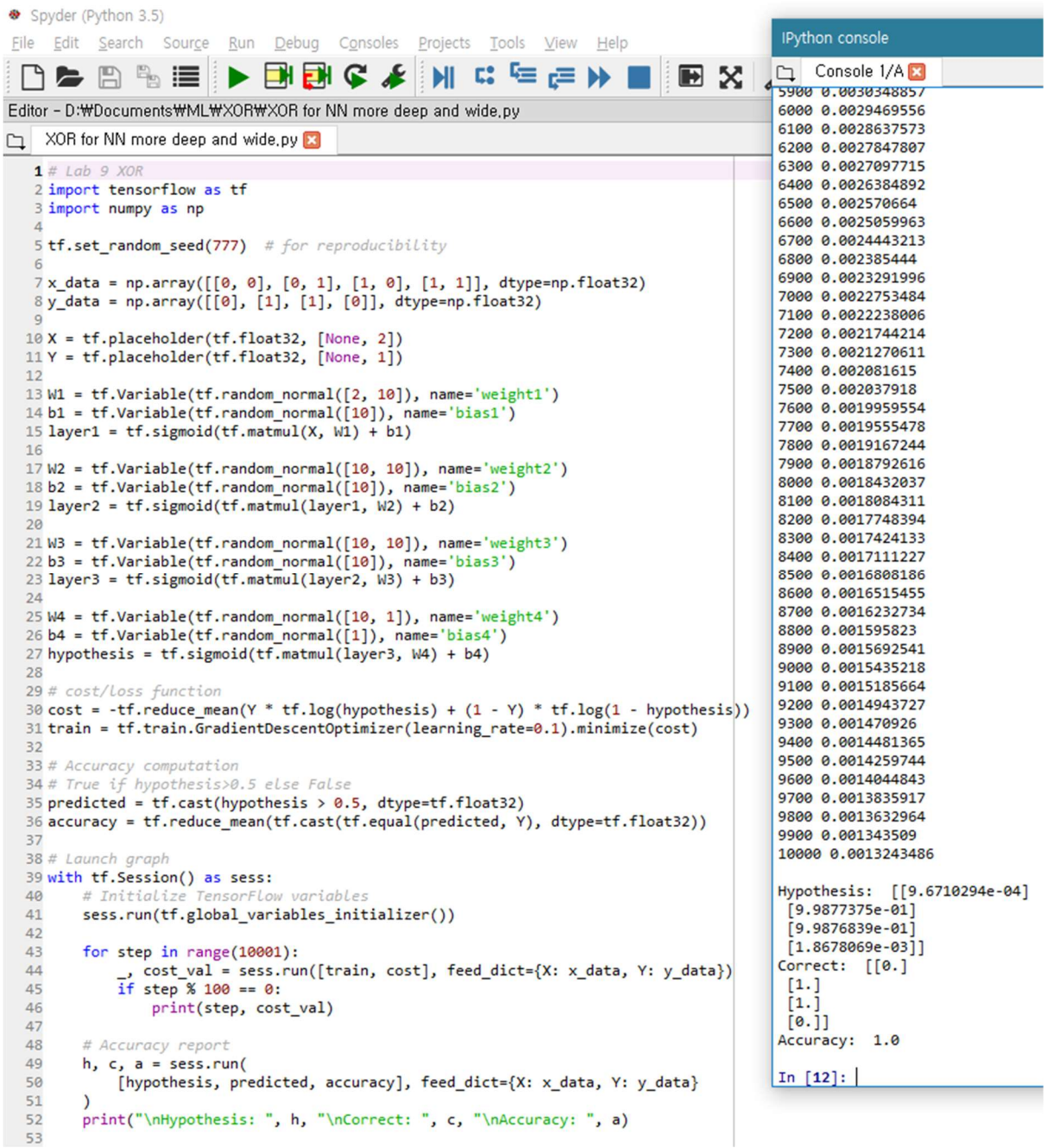
>>> 성공적인 실행 화면




```
# Lab 9 XOR
import tensorflow as tf
import numpy as np

tf.set_random_seed(777)  # for reproducibility

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

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        _, cost_val = sess.run([train, cost], feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, cost_val)

    # Accuracy report
    h, p, a = sess.run(
        [hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data}
    )

    print("\nHypothesis:\n{} \nPredicted:\n{} \nAccuracy:\n{}".format(h, p, a))
```

✓ 더 너엏고 기잎게

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



- 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

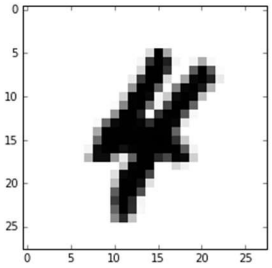
Number Recognition_Deeper Wider.py | XOR for NN more deep and wide.py

```python
# ML lab 09-1
import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777) # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data

# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W1 = tf.Variable(tf.random_normal([784, 256]), name='weight1')
b1 = tf.Variable(tf.random_normal([256]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]), name='weight2')
b2 = tf.Variable(tf.random_normal([256]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 256]), name='weight3')
b3 = tf.Variable(tf.random_normal([256]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([256, nb_classes]), name='weight4')
b4 = tf.Variable(tf.random_normal([nb_classes]), name='bias4')

# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(layer3, W4) + b4)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
num_epochs = 15
batch_size = 100
num_iterations = int(mnist.train.num_examples / batch_size)

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(num_epochs):
        avg_cost = 0

        for i in range(num_iterations):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            _, cost_val = sess.run([train, cost], feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += cost_val / num_iterations
```

IPython console

Console 1/A

```
 [9.9877375e-01]
 [9.9876839e-01]
 [1.8678069e-03]]
Correct:  [[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:  1.0

In [12]: runfile('D:/Documents/ML/MNIST/Number Recognition_Deeper Wider.p
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: 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)
```

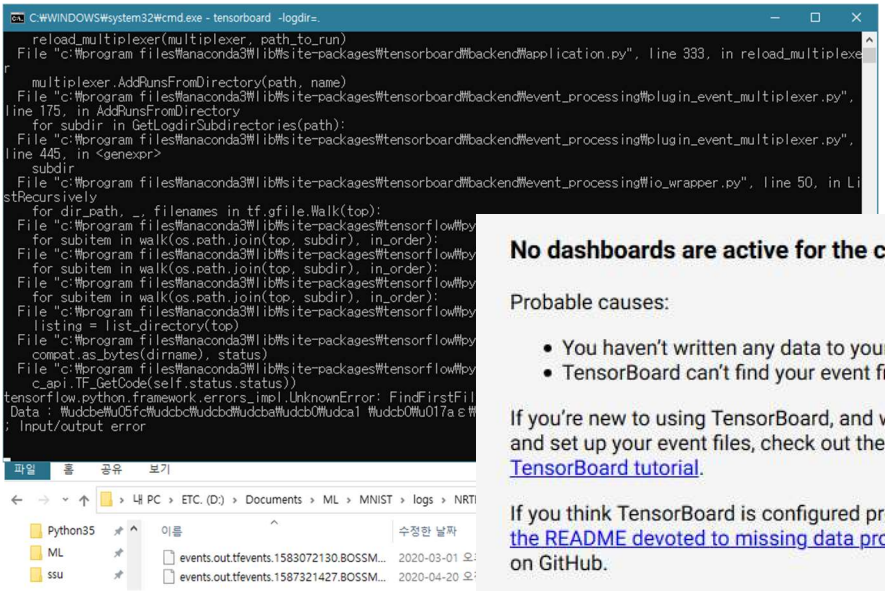
5. Launch TensorBoard

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

5 steps of using TensorBoard

- 과정은 더 복잡해진 것 같다

✓ **Exercise 첫 시도**



실패

No dashboards are active for the current data set.

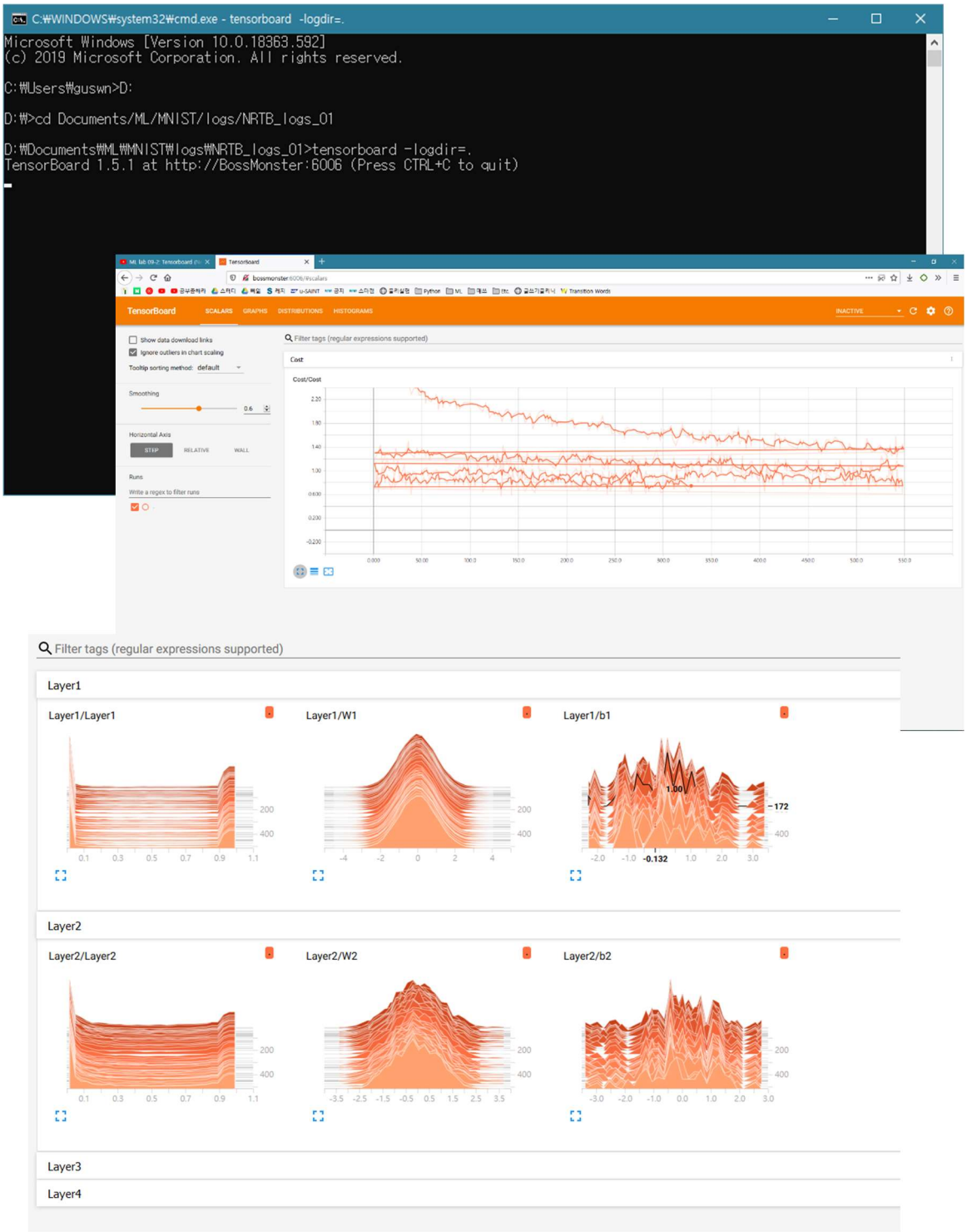
Probable causes:

- You haven't written any data to your event files.
- TensorBoard can't find your event files.

If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the README and perhaps the TensorBoard tutorial.

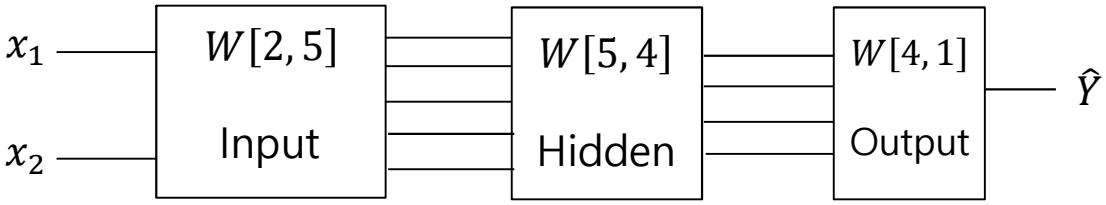
If you think TensorBoard is configured properly, please see the section of the README devoted to missing data problems and consider filing an issue on GitHub.

- 검색 끝에 겨우 성공(폴더명에 한국어 금지, 경로 직접 이동)



ML lec 10-1

✓ Neural Network의 구조



- 9단까지도 가능

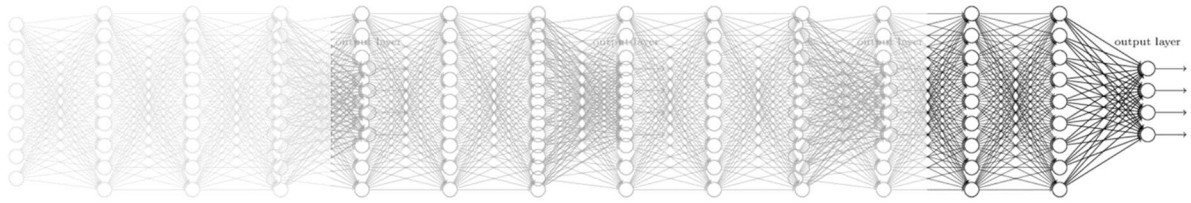
```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0), name = "Weight1")

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**

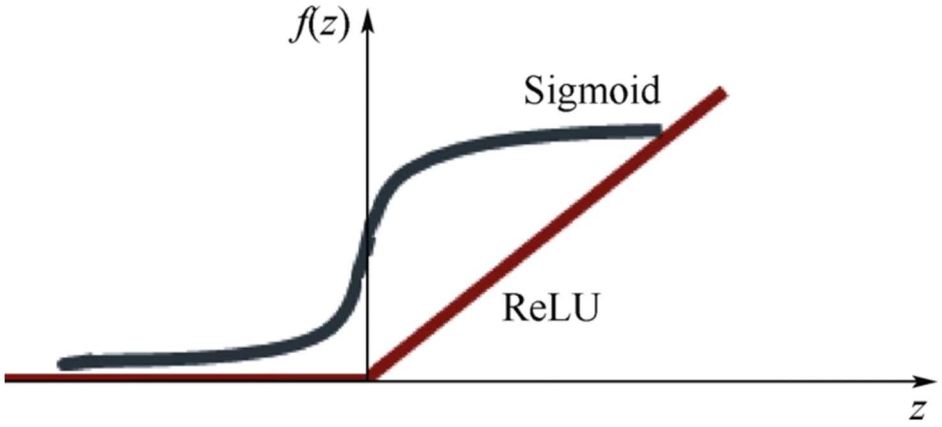
```
196000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
198000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
[array([[ 0.49999988],
       [ 0.50000006],
       [ 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

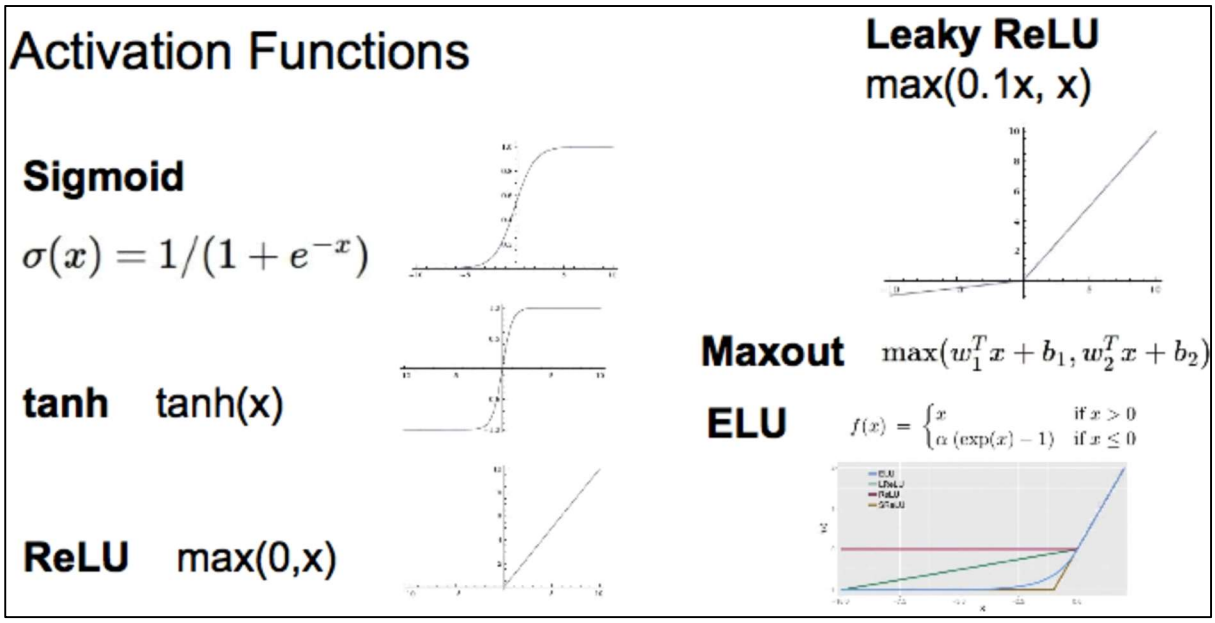


- 구현법

```
1   L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
```

```
196000 [2.6226094e-06, array([[  2.59195826e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.43454133e-06]], dtype=float32)]
198000 [2.607708e-06, array([[  2.55822852e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.40260101e-06]], dtype=float32)]
[array([[  2.52509381e-06],
       [  9.99999642e-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

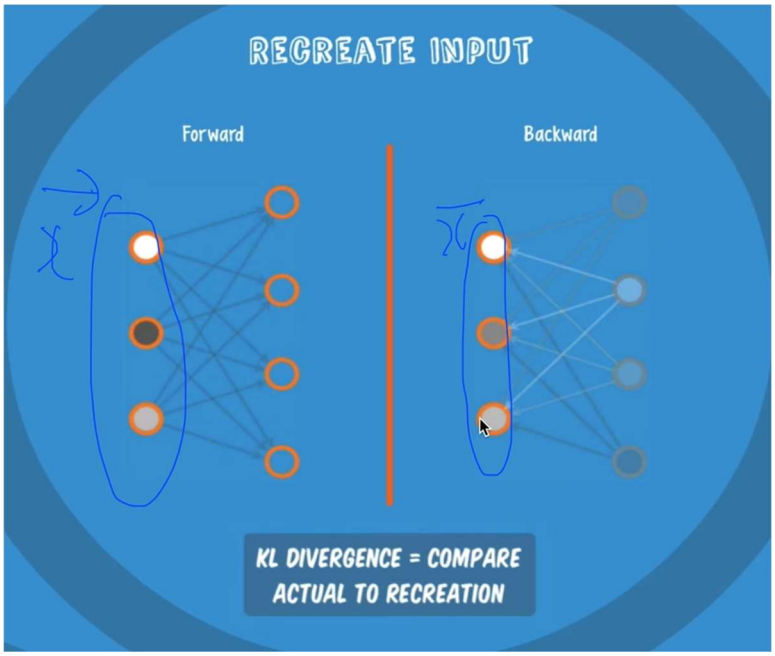
maxout	ReLU	VLReLU	tanh	Sigmoid
93.94	**92.11**	92.97	89.28	n/c
93.78	91.74	92.40	89.48	n/c
–	91.93	**93.09**	–	n/c
91.75	90.63	92.27	**89.82**	n/c
n/c†	90.91	92.43	89.54	n/c

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**

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

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

- 효과만점

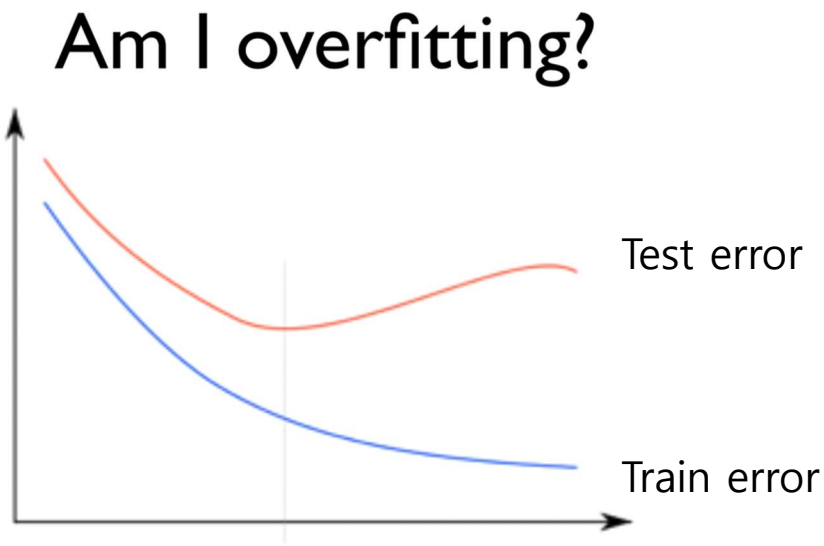
Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	**93.09**	–	n/c
Xavier	91.75	90.63	92.27	**89.82**	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

✓ **하지만 이 분야는 여전히 연구 중에 있다**

- 따라서 여러 번 직접 시도해보는 것도 좋은 방법

ML lec 10-3

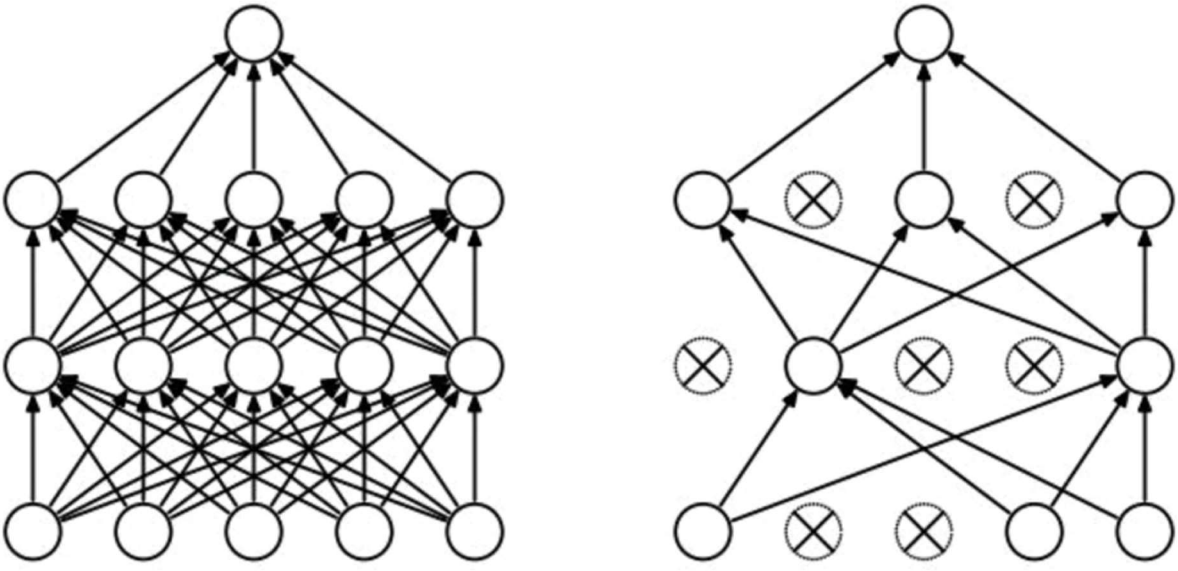
✓ **여전히 골칫거리인 Overfitting**



- 한 가지 방법으로 언급했던 Regularization(l2reg)

✓ **NN에는 그 외의 방법이 더 있다**

- Dropout(그만두기)



(a) Standard Neural Net (b) After applying dropout.

- 의외로 실행이 잘 된다

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

- 대신 Train, Evaluation할 땐 변경시켜 줘야 함

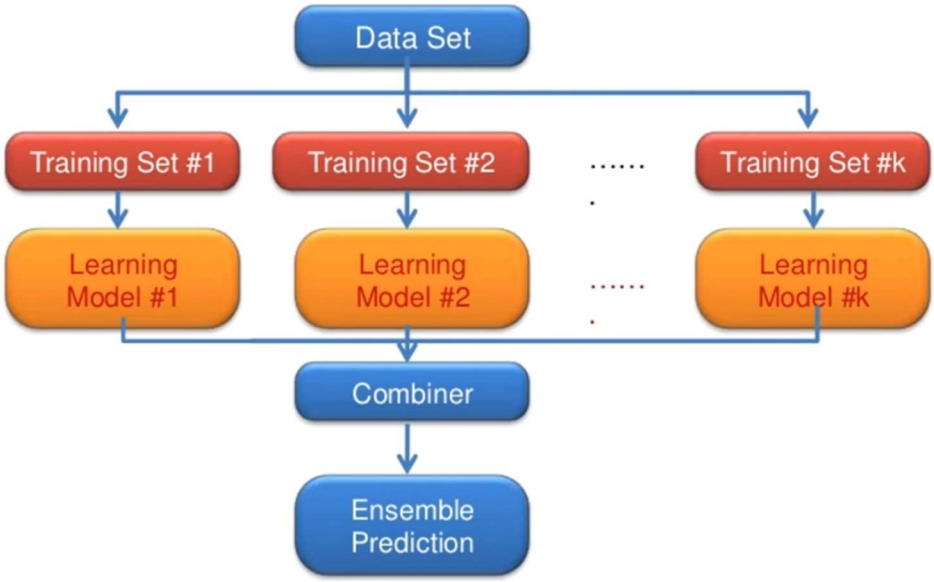
TRAIN:

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

EVALUATION:

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

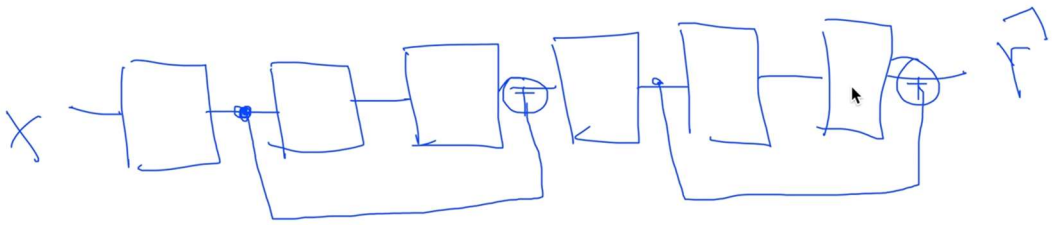
✓ Ensemble



- 사용하면 최대 4-5%까지 성능이 향상 됨

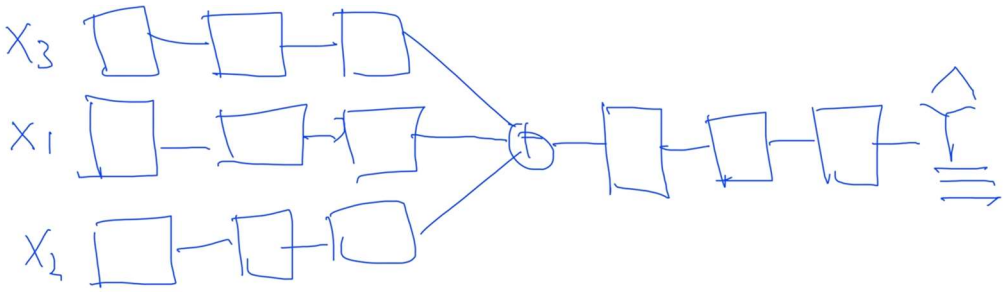
ML lego 10-4

✓ **Fast Forward**

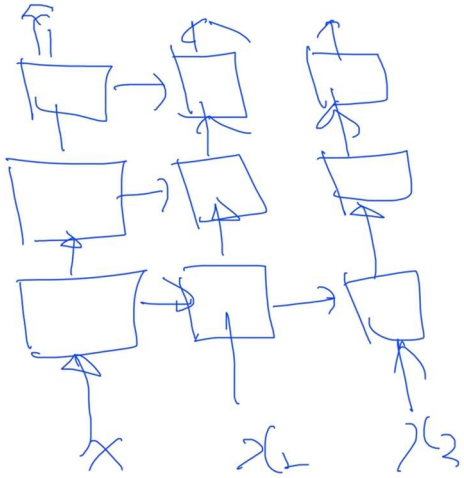


- 뽑아서 앞으로 옮김

✓ **Split & Merge**



✓ **Recurrenct Network(RNN)**



✓ 이처럼 네트워크를 마음대로 레고처럼 조립할 수 있다

ML lab 10

✓ MNIST for Softmax : 0.8951

Editor - D:₩Documents₩ML₩MNIST₩Number Recognition.py

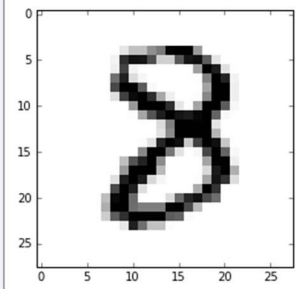
Number Recognition.py | Number Recognition_Deeper Wider.py | Number Recognition_ReLU.py

```python
# ML lab 07-2
import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777) # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data

# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))

# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
num_epochs = 15
batch_size = 100
num_iterations = int(mnist.train.num_examples / batch_size)

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(num_epochs):
        avg_cost = 0

        for i in range(num_iterations):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            _, cost_val = sess.run([train, cost], feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += cost_val / num_iterations

        print("Epoch: {:04d}, Cost: {:.9f}".format(epoch + 1, avg_cost))

    print("Learning finished")

    # Test the model using test sets
```

IPython console

Console 1/A

```
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default,
Type "copyright", "credits" or "license" for more

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]:

In [1]: runfile('D:/Documents/ML/MNIST/Number Rec
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]
```



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

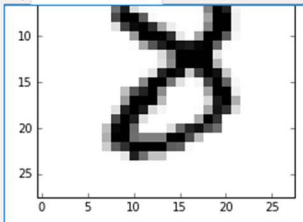
Editor - D:₩Documents₩ML₩MNIST₩Number Recognition_Deeper Wider.py

Number Recognition.py | Number Recognition_Deeper Wider.py | Number Recognition_ReLU.py

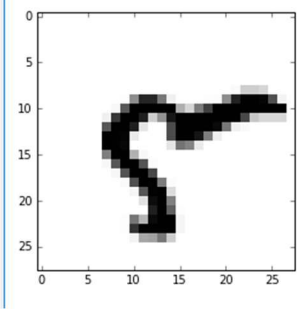
```python
# ML lab 09-1
import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777) # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data

# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W1 = tf.Variable(tf.random_normal([784, 256]), name='weight1')
b1 = tf.Variable(tf.random_normal([256]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]), name='weight2')
b2 = tf.Variable(tf.random_normal([256]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 256]), name='weight3')
b3 = tf.Variable(tf.random_normal([256]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([256, nb_classes]), name='weight4')
b4 = tf.Variable(tf.random_normal([nb_classes]), name='bias4')

# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(layer3, W4) + b4)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
num_epochs = 15
batch_size = 100
num_iterations = int(mnist.train.num_examples / batch_size)

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(num_epochs):
        avg_cost = 0
```

IPython console

Console 1/A



```
In [2]: runfile('D:/Documents/ML/MNIST/Number Re
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: 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: 0.267029808
Epoch: 0013, Cost: 0.254337992
Epoch: 0014, Cost: 0.242623737
Epoch: 0015, Cost: 0.231251723
Learning finished
Accuracy:  0.8951
Label:  [5]
Prediction:  [9]
```



✓ MNIST for ReLU : 0.955!

Editor - D:₩Documents₩ML₩MNIST₩Number Recognition_ReLU.py

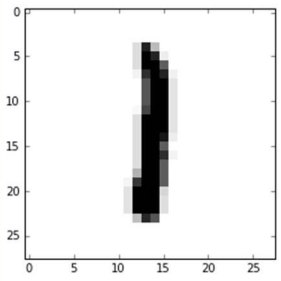
mber Recognition.py | Number Recognition_Deeper Wider.py | Number Recognition_ReLU.py

```
# ML lab 09-1
import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777) # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data

# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W1 = tf.Variable(tf.random_normal([784, 256]), name='weight1')
b1 = tf.Variable(tf.random_normal([256]), name='bias1')
layer1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]), name='weight2')
b2 = tf.Variable(tf.random_normal([256]), name='bias2')
layer2 = tf.nn.relu(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 256]), name='weight3')
b3 = tf.Variable(tf.random_normal([256]), name='bias3')
layer3 = tf.nn.relu(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([256, nb_classes]), name='weight4')
b4 = tf.Variable(tf.random_normal([nb_classes]), name='bias4')

# Hypothesis (using softmax)
hypothesis = tf.matmul(layer3, W4) + b4

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
train = tf.train.AdamOptimizer(learning_rate=0.01).minimize(cost)

# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
num_epochs = 15
batch_size = 100
num_iterations = int(mnist.train.num_examples / batch_size)

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
```

IPython console

Console 1/A

```
In [3]: runfile('D:/Documents/ML/MNIST/Number Recogn
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
softmax_cross_entropy_with_logits (from tensorflow.p
in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradi
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

Editor - D:₩Documents₩ML₩MNIST₩Number Recognition_Xavier.py

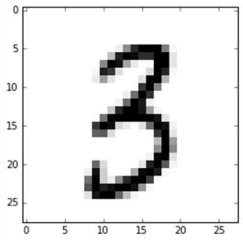
ecognition_Deeper Wider.py | Number Recognition_ReLU.py | Number Recognition_Xavier.py

```
# ML lab 09-1
import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777) # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data

# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W1 = tf.get_variable("W1", shape=[784, 256],
                     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]), name='bias1')
layer1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[256, 256],
                     initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([256]), name='bias2')
layer2 = tf.nn.relu(tf.matmul(layer1, W2) + b2)

W3 = tf.get_variable("W3", shape=[256, 256],
                     initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([256]), name='bias3')
layer3 = tf.nn.relu(tf.matmul(layer2, W3) + b3)

W4 = tf.get_variable("W4", shape=[256, nb_classes],
                     initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([nb_classes]), name='bias4')

# Hypothesis (using softmax)
hypothesis = tf.matmul(layer3, W4) + b4 # 'tf.nn.softmax' deleted.

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
train = tf.train.AdamOptimizer(learning_rate=0.01).minimize(cost) # Gradient -> Adam

# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
num_epochs = 15
batch_size = 100
num_iterations = int(mnist.train.num_examples / batch_size)

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(num_epochs):
```

IPython console

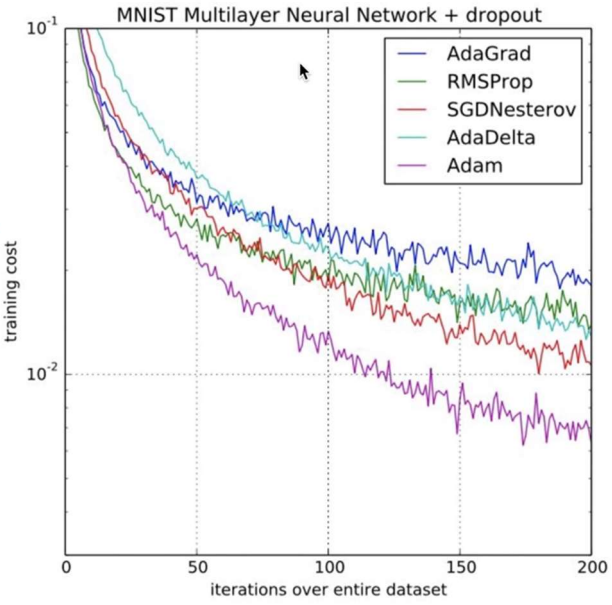
Console 1/A

```
In [4]: runfile('D:/Documents/ML/MNIST/Number Recog
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: 0.294849021
Epoch: 0002, Cost: 0.156128212
Epoch: 0003, Cost: 0.134834543
Epoch: 0004, Cost: 0.117306856
Epoch: 0005, Cost: 0.106162803
Epoch: 0006, Cost: 0.107378627
Epoch: 0007, Cost: 0.095509183
Epoch: 0008, Cost: 0.091734687
Epoch: 0009, Cost: 0.091129726
Epoch: 0010, Cost: 0.079916846
Epoch: 0011, Cost: 0.077588749
Epoch: 0012, Cost: 0.072744891
Epoch: 0013, Cost: 0.066493722
Epoch: 0014, Cost: 0.069562955
Epoch: 0015, Cost: 0.062439037
Learning finished
Accuracy:  0.971
Label:  [3]
Prediction:  [3]
```



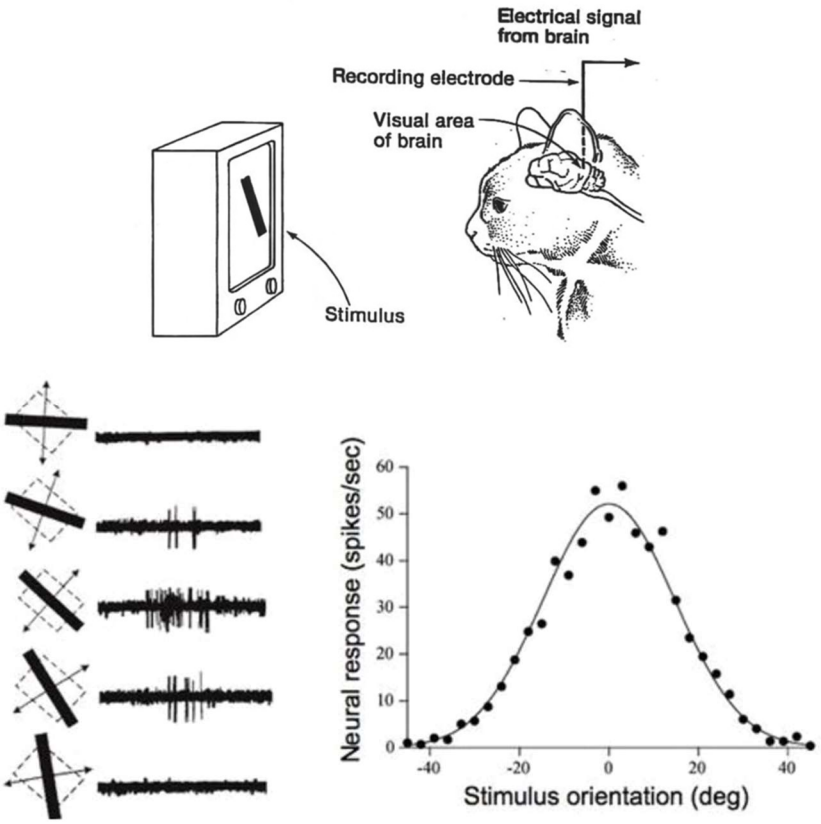
```
In [5]:
```

+ AdamOptimizer의 성능이 좋다

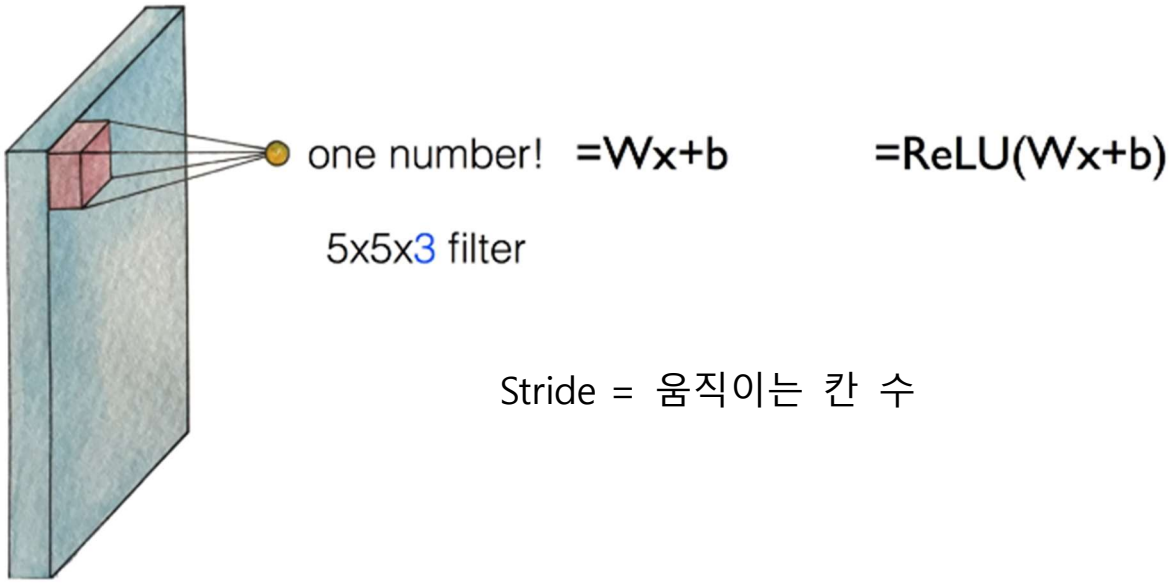


ML lec 11-1

✓ **CNN의 시초 : 고양이 실험**



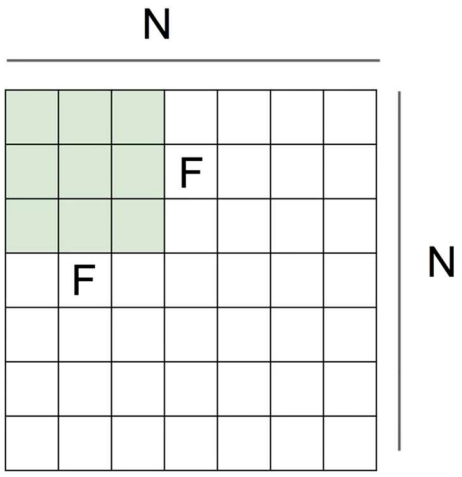
✓ **Filter**(빨간 블록)를 사용해서 하나의 숫자를 추출하자



Stride = 움직이는 칸 수

· 이것을 반복하면 몇 개의 숫자를 얻을 수 있을까?

✓ **다음과 같이 계산할 수 있다**

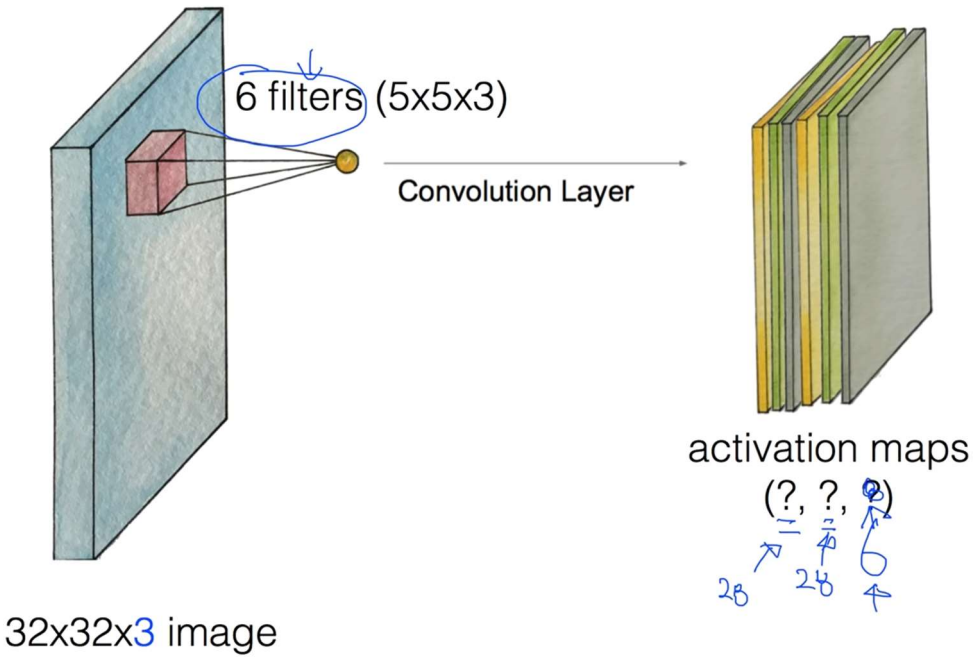


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

· 대신 결과값은 정수가 나와야 한다

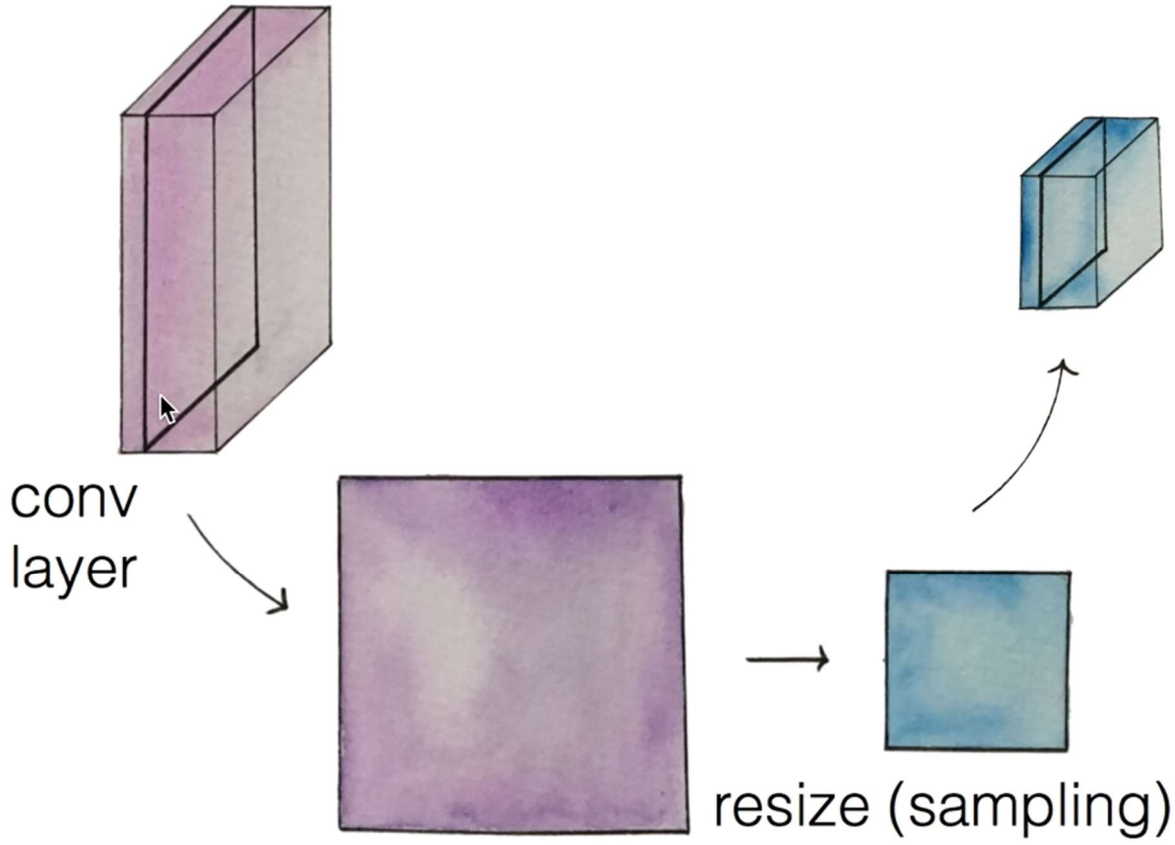
✓ 이런 식으로 필터들을 통해 activation maps를 만들 수 있다



(32 – 5)/1 + 1 = 28

ML lec 11-2

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



· 그 후에 Max Pooling(2x2 filters and stirde 2로)

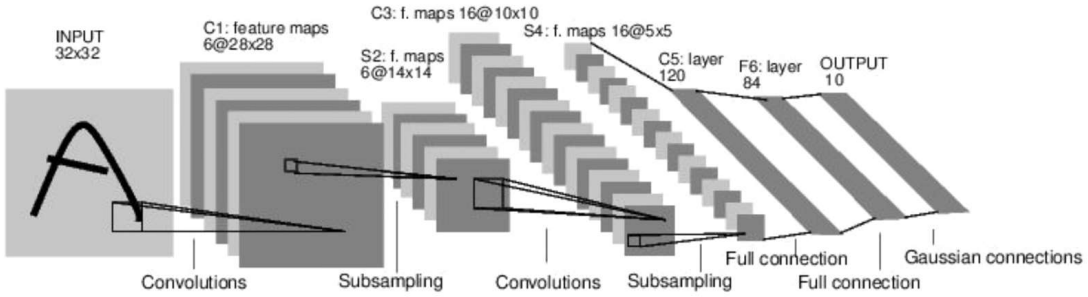
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

→

6	8
3	4

ML lec 11-3

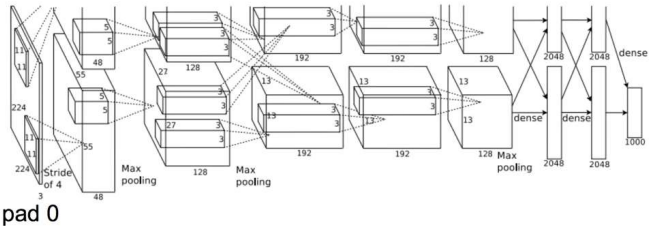
✓ ConvNet의 활용예



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]

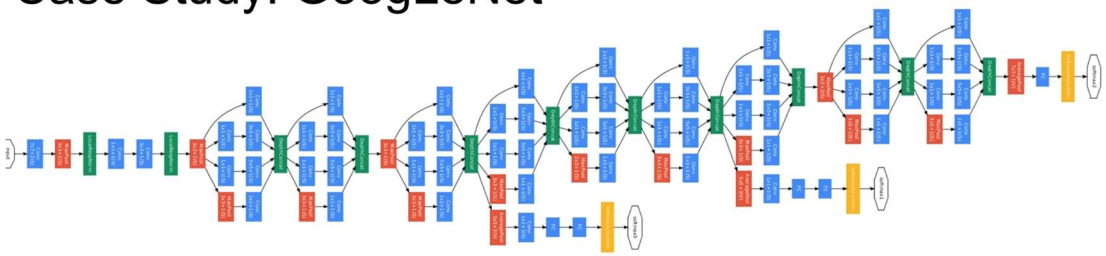
Case Study: AlexNet

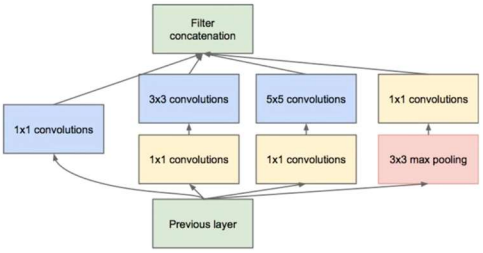
[Krizhevsky et al. 2012]



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]*



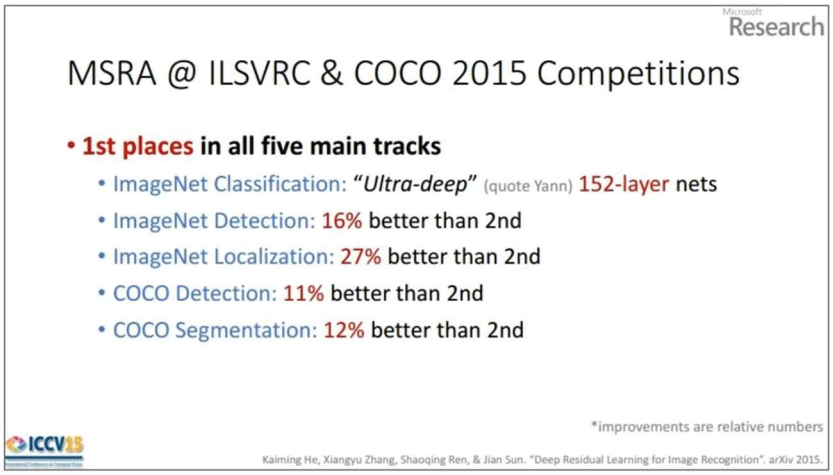


Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Case Study: ResNet *[He et al., 2015]*

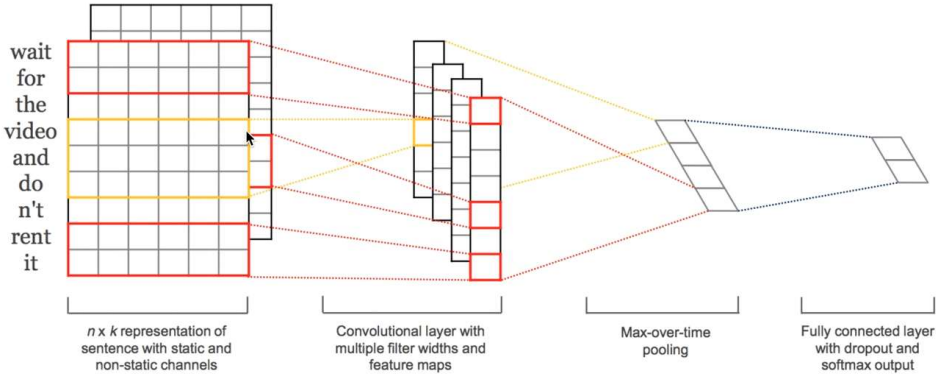
ILSVRC 2015 winner (3.6% top 5 error)



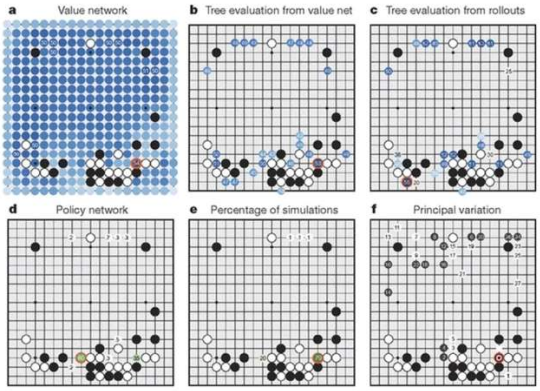
Fast Forward 사용

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

Convolutional Neural Networks for Sentence Classification [Yoon Kim, 2014]

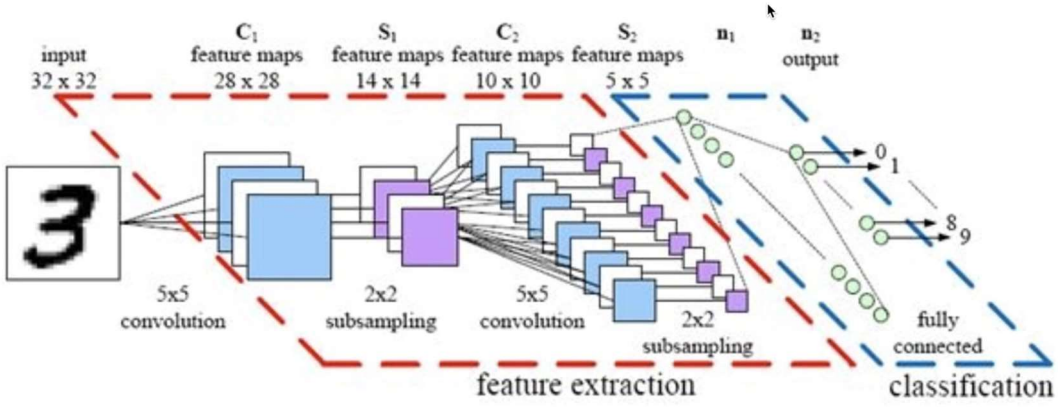


Case Study Bonus: DeepMind's AlphaGo



ML lab 11-1

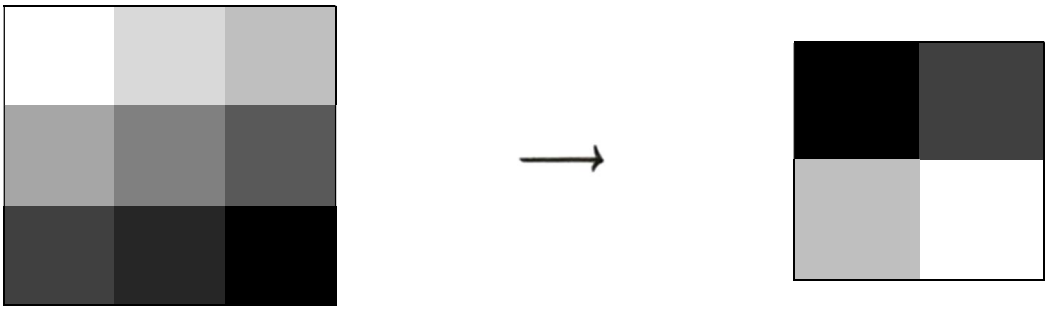
✓ **CNN의 구조**



- 구현

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

```
# print("imag:\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')
```

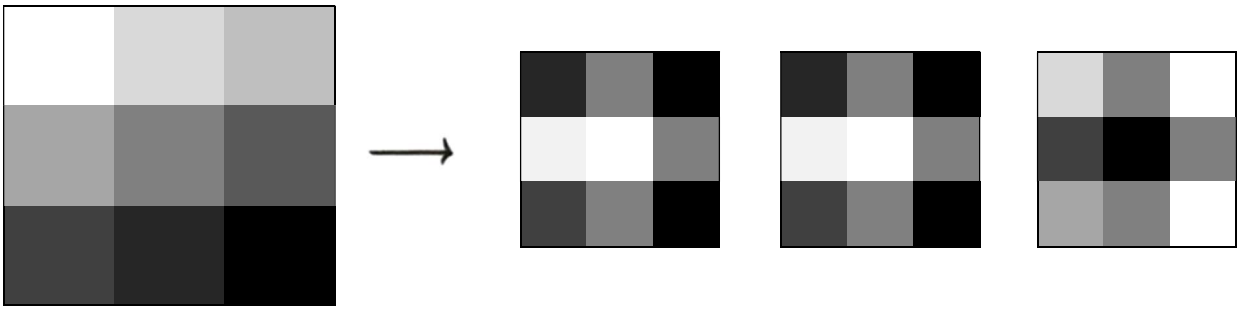


✓ Padding

```
# print("imag:\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("imag:\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')
```



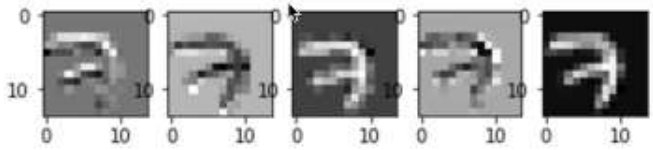
✓ Max Pooling

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

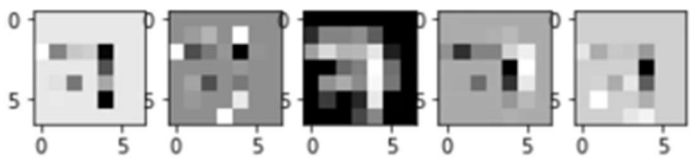
```
Tensor("Conv2D_1:0", shape=(1, 14, 14, 5), dtype=float32)
```



✓ MNIST Max Pooling

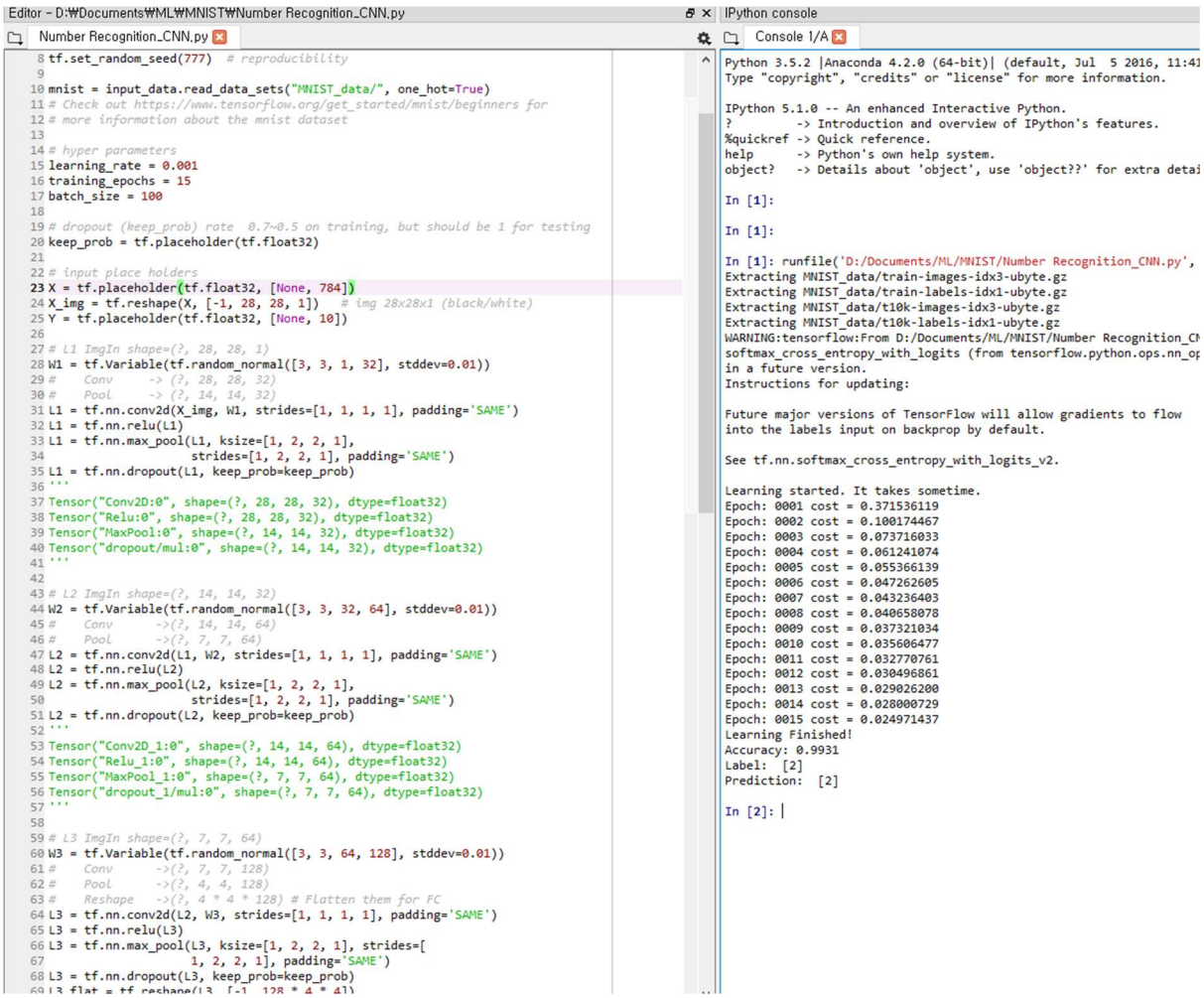
```
pool = tf.nn.max_pool(conv2d, ksize=[1, 2, 2, 1], strides=[
                      1, 2, 2, 1], padding='SAME')
print(pool)
sess.run(tf.global_variables_initializer())
pool_img = pool.eval()
pool_img = np.swapaxes(pool_img, 0, 3)
for i, one_img in enumerate(pool_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(7,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

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



ML lab 11-2

- ✓ **MNIST with CNN : 0.9931**



Editor - D:₩Documents₩ML₩MNIST₩Number Recognition_CNN.py

Number Recognition_CNN.py

```
tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# hyper parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# dropout (keep_prob) rate  0.7~0.5 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])

# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv      -> (?, 28, 28, 32)
#    Pool      -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
'''
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
Tensor("dropout/mul:0", shape=(?, 14, 14, 32), dtype=float32)
'''

# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("dropout_1/mul:0", shape=(?, 7, 7, 64), dtype=float32)
'''

# L3 ImgIn shape=(?, 7, 7, 64)
W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
#    Conv      ->(?, 7, 7, 128)
#    Pool      ->(?, 4, 4, 128)
#    Reshape   ->(?, 4 * 4 * 128) # Flatten them for FC
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
L3 = tf.nn.relu(L3)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
                    1, 2, 2, 1], padding='SAME')
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
L3_flat = tf.reshape(L3, [-1, 128 * 4 * 4])
```

IPython console

Console 1/A

```
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 detai

In [1]:

In [1]:

In [1]: runfile('D:/Documents/ML/MNIST/Number Recognition_CNN.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
WARNING:tensorflow:From D:/Documents/ML/MNIST/Number Recognition_CN
softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_op
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_Ensemble.py

Number Recognition_Ensemble.py

```python
training_epochs = 20
batch_size = 100


class Model:

    def __init__(self, sess, name):
        self.sess = sess
        self.name = name
        self._build_net()

    def _build_net(self):
        with tf.variable_scope(self.name):
            # dropout (keep_prob) rate  0.7~0.5 on training, but should be 1
            # for testing
            self.training = tf.placeholder(tf.bool)

            # input place holders
            self.X = tf.placeholder(tf.float32, [None, 784])

            # img 28x28x1 (black/white), Input Layer
            X_img = tf.reshape(self.X, [-1, 28, 28, 1])
            self.Y = tf.placeholder(tf.float32, [None, 10])

            # Convolutional Layer #1
            conv1 = tf.layers.conv2d(inputs=X_img, filters=32, kernel_size=[3, 3],
                                     padding="SAME", activation=tf.nn.relu)
            # Pooling Layer #1
            pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
                                            padding="SAME", strides=2)
            dropout1 = tf.layers.dropout(inputs=pool1,
                                         rate=0.3, training=self.training)

            # Convolutional Layer #2 and Pooling Layer #2
            conv2 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3],
                                     padding="SAME", activation=tf.nn.relu)
            pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
                                            padding="SAME", strides=2)
            dropout2 = tf.layers.dropout(inputs=pool2,
                                         rate=0.3, training=self.training)

            # Convolutional Layer #3 and Pooling Layer #3
            conv3 = tf.layers.conv2d(inputs=dropout2, filters=128, kernel_size=[3, 3],
                                     padding="SAME", activation=tf.nn.relu)
            pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2],
                                            padding="SAME", strides=2)
            dropout3 = tf.layers.dropout(inputs=pool3,
                                         rate=0.3, training=self.training)

            # Dense Layer with 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)

            # Logits (no activation) Layer: L5 Final FC 625 inputs -> 10 outputs
            self.logits = tf.layers.dense(inputs=dropout4, units=10)

        # define cost/loss & optimizer
        self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
            logits=self.logits, labels=self.Y))
```

IPython console

Console 1/A

```
In [3]: runfile('D:/Documents/ML/MNIST/Number Recog
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 = [0.04005741 0.04080501]
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 = [0.03109086 0.02964267]
Epoch: 0013 cost = [0.02744721 0.02809541]
Epoch: 0014 cost = [0.02813654 0.02901832]
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]:
```