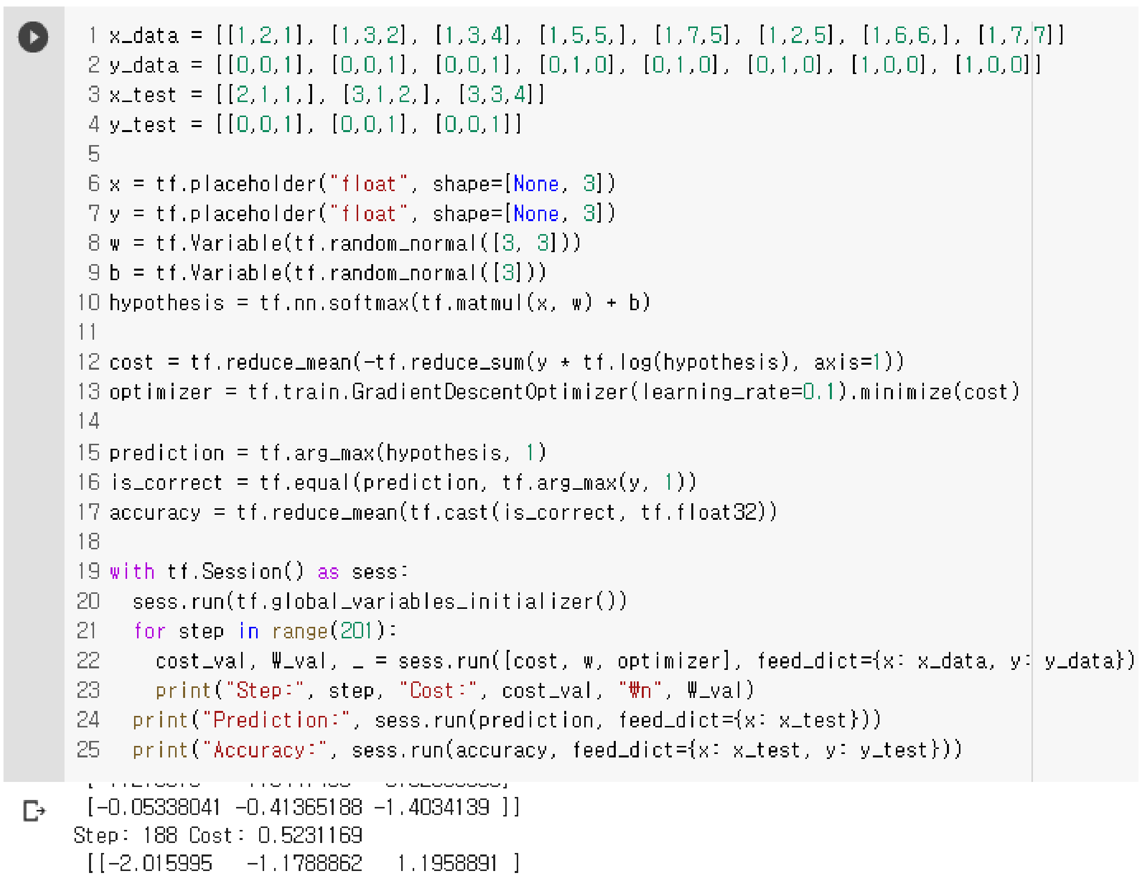
ML_Lab 07-1.

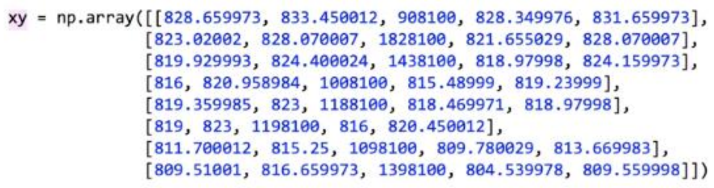
Training & Test Test.



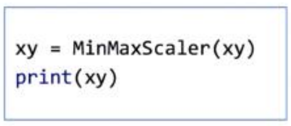
```
x_data = [[1,2,1], [1,3,2], [1,3,4], [1,5,5,], [1,7,5], [1,2,5], [1,6,6,], [1,7,7]]
y_data = [[0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0], [1,0,0], [1,0,0]]
x_test = [[2,1,1,], [3,1,2,], [3,3,4]]
y_test = [[0,0,1], [0,0,1], [0,0,1]]

x = tf.placeholder("float", shape=[None, 3])
y = tf.placeholder("float", shape=[None, 3])
w = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))
hypothesis = tf.nn.softmax(tf.matmul(x, w) + b)

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

prediction = tf.arg_max(hypothesis, 1)
is_correct = tf.equal(prediction, tf.arg_max(y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for step in range(201):
    cost_val, ₩_val, _ = sess.run([cost, w, optimizer], feed_dict={x: x_data, y: y_data})
    print("Step:", step, "Cost:", cost_val, "₩n", ₩_val)
  print("Prediction:", sess.run(prediction, feed_dict={x: x_test}))
  print("Accuracy:", sess.run(accuracy, feed_dict={x: x_test, y: y_test}))
```

```
 [-0.05338041 -0.41365188 -1.4034139 ]]
Step: 188 Cost: 0.5231169
 [[-2.015995   -1.1788862   1.1958891 ]
```

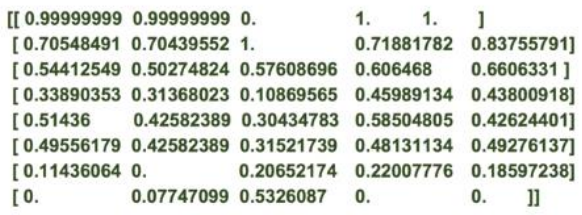
Normalized Inputs Test.



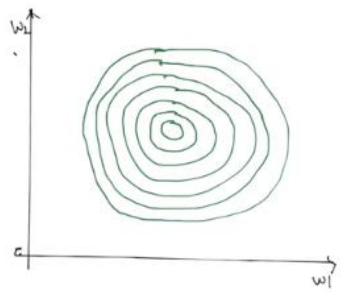
```
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```

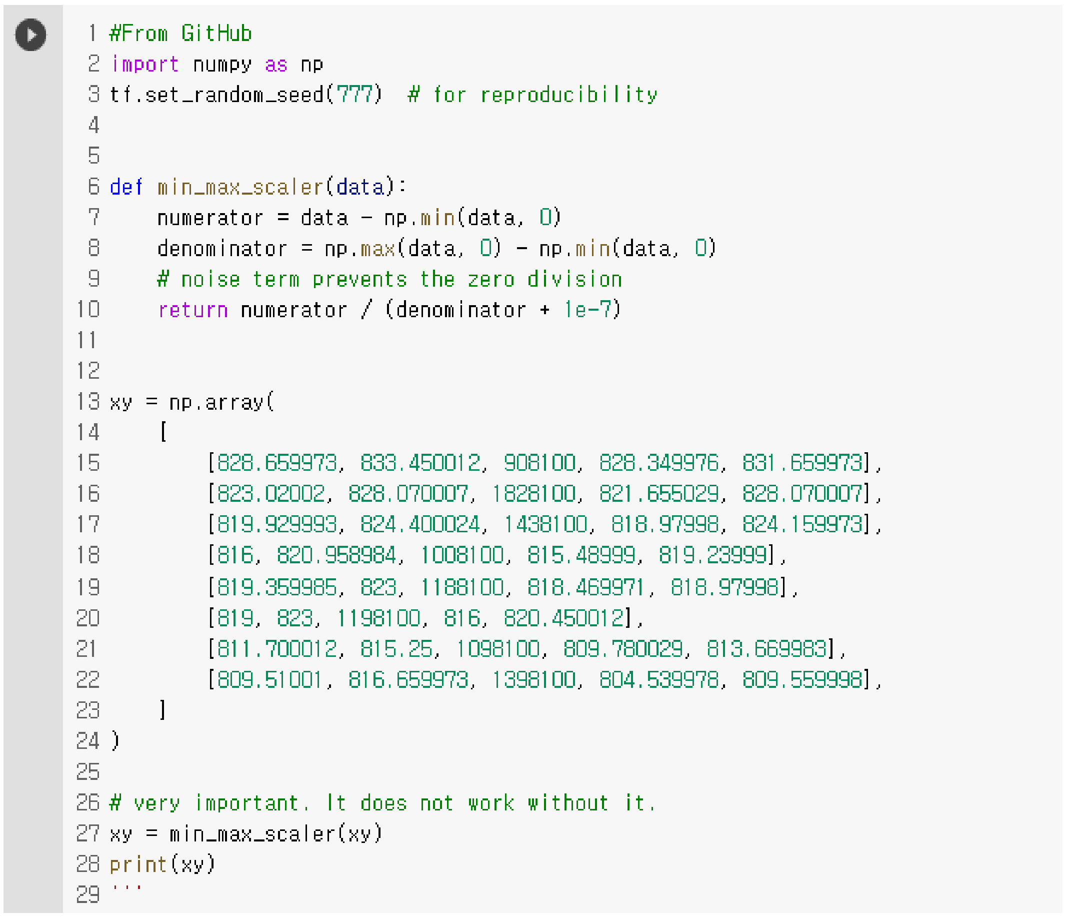


```
xy = MinMaxScaler(xy)
print(xy)
```

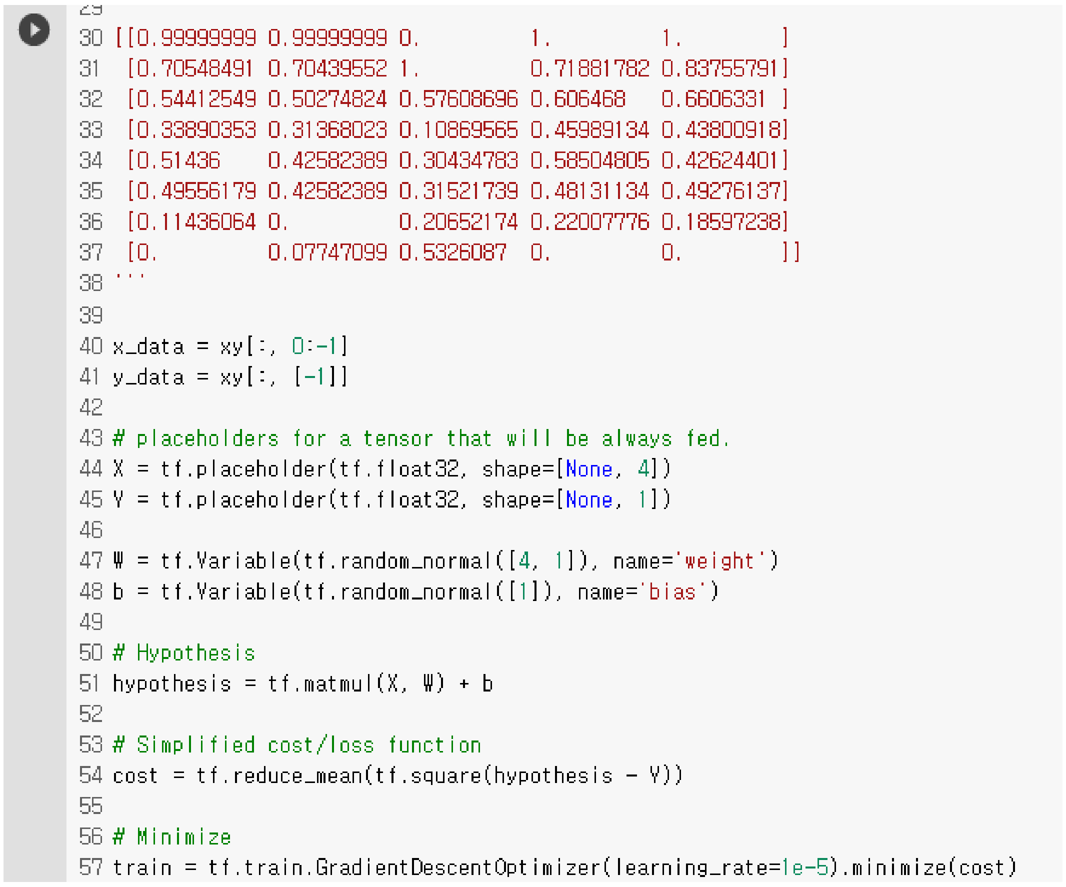


```
[[ 0.99999999  0.99999999  0.          1.          1.        ]
 [ 0.70548491  0.70439552  1.          0.71881782  0.83755791]
 [ 0.54412549  0.50274824  0.57608696  0.606468    0.6606331 ]
 [ 0.33890353  0.31368023  0.10869565  0.45989134  0.43800918]
 [ 0.51436     0.42582389  0.30434783  0.58504805  0.42624401]
 [ 0.49556179  0.42582389  0.31521739  0.48131134  0.49276137]
 [ 0.11436064  0.          0.20652174  0.22007776  0.18597238]
 [ 0.          0.07747099  0.5326087   0.          0.        ]]
```

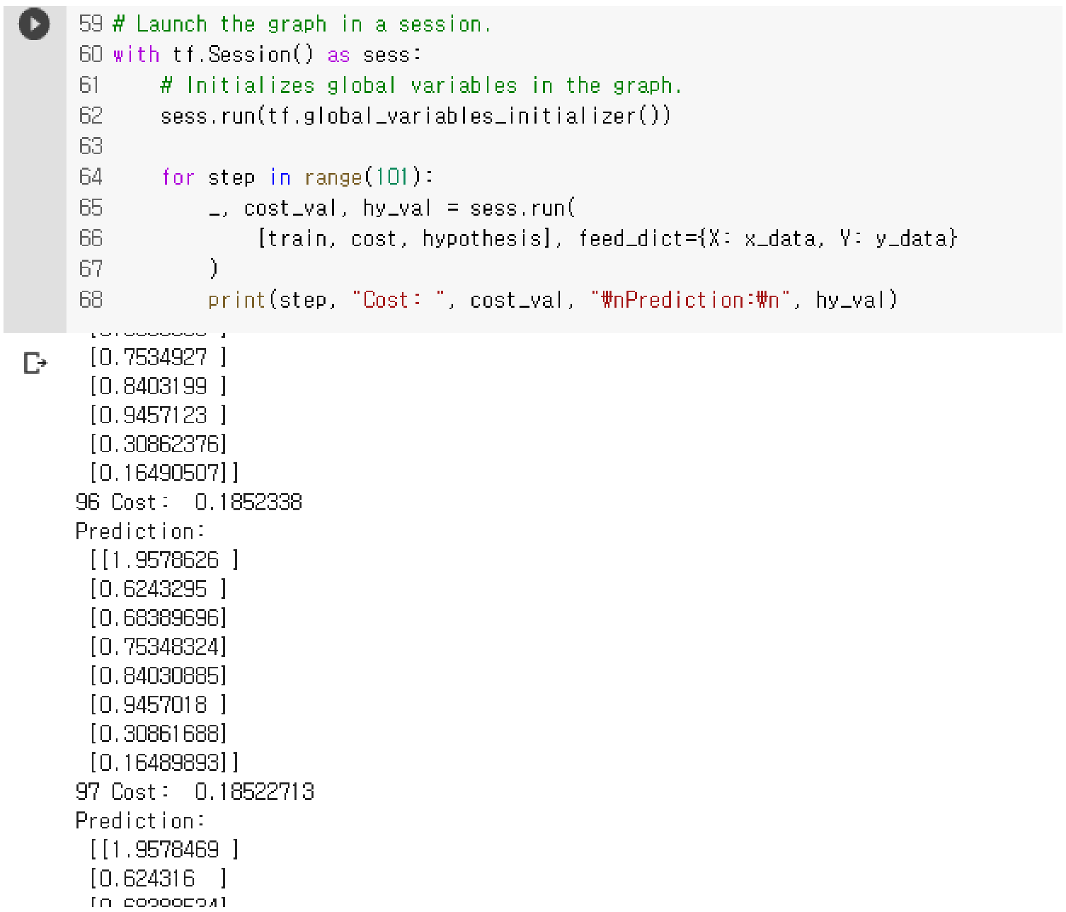




```python
#From GitHub
import numpy as np
tf.set_random_seed(777)  # for reproducibility


def min_max_scaler(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    # noise term prevents the zero division
    return numerator / (denominator + 1e-7)


xy = np.array(
    [
        [828.659973, 833.450012, 908100, 828.349976, 831.659973],
        [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
        [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
        [816, 820.958984, 1008100, 815.48999, 819.23999],
        [819.359985, 823, 1188100, 818.469971, 818.97998],
        [819, 823, 1198100, 816, 820.450012],
        [811.700012, 815.25, 1098100, 809.780029, 813.669983],
        [809.51001, 816.659973, 1398100, 804.539978, 809.559998],
    ]
)

# very important. It does not work without it.
xy = min_max_scaler(xy)
print(xy)
'''
```



```python
[[0.99999999 0.99999999 0.         1.         1.        ]
 [0.70548491 0.70439552 1.         0.71881782 0.83755791]
 [0.54412549 0.50274824 0.57608696 0.606468   0.6606331 ]
 [0.33890353 0.31368023 0.10869565 0.45989134 0.43800918]
 [0.51436    0.42582389 0.30434783 0.58504805 0.42624401]
 [0.49556179 0.42582389 0.31521739 0.48131134 0.49276137]
 [0.11436064 0.         0.20652174 0.22007776 0.18597238]
 [0.         0.07747099 0.5326087  0.         0.        ]]
'''


x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([4, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

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

# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
train = tf.train.GradientDescentOptimizer(learning_rate=1e-5).minimize(cost)
```



```python
# Launch the graph in a session.
with tf.Session() as sess:
    # Initializes global variables in the graph.
    sess.run(tf.global_variables_initializer())

    for step in range(101):
        _, cost_val, hy_val = sess.run(
            [train, cost, hypothesis], feed_dict={X: x_data, Y: y_data}
        )
        print(step, "Cost: ", cost_val, "₩nPrediction:₩n", hy_val)
```

```
 [0.7534927 ]
 [0.8403199 ]
 [0.9457123 ]
 [0.30862376]
 [0.16490507]]
96 Cost:  0.1852338
Prediction:
 [[1.9578626 ]
 [0.6243295 ]
 [0.68389696]
 [0.75348324]
 [0.84030885]
 [0.9457018 ]
 [0.30861688]
 [0.16489893]]
97 Cost:  0.18522713
Prediction:
 [[1.9578469 ]
 [0.624316  ]
```

ML_Lab 07-2.

Training Epoch & Batch.

1. One Epoch : One forward pass and one backward pass of all the training examples
2. Batch Size : The number of training examples in one forward/backward pass. The higher the Batch Size, the more memory space you'll need.
3. Number of Iterations : number of passes, each pass using [Batch Size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example : If you have 1000 training examples, and your Batch Size is 500, then it will take 2 Iterations to complete 1 Epoch.

MNIST Dataset Test.

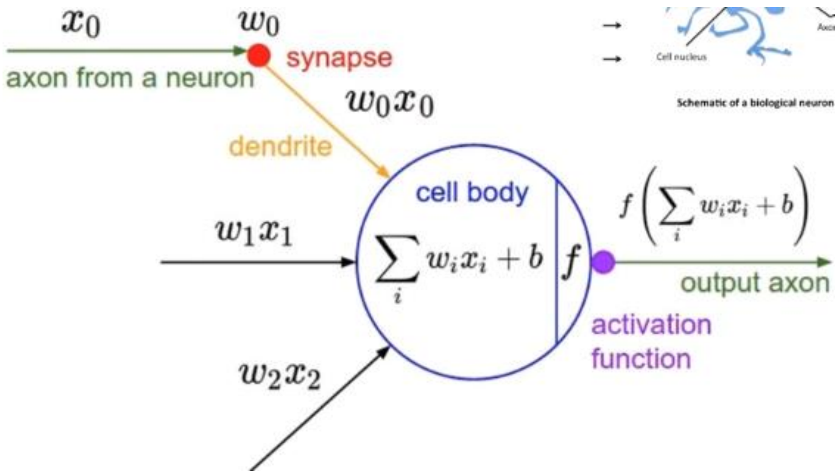
```python
# Lab 7 Learning rate and Evaluation
import tensorflow as tf
import matplotlib.pyplot as plt
import random

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
    print(
        "Accuracy: ",
        accuracy.eval(
            session=sess, feed_dict={X: mnist.test.images, Y: mnist.test.labels}
        ),
    )

    # Get one and predict
    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label: ", sess.run(tf.argmax(mnist.test.labels[r : r + 1], 1)))
    print(
        "Prediction: ",
        sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r : r + 1]}),
    )

    plt.imshow(
        mnist.test.images[r : r + 1].reshape(28, 28),
        cmap="Greys",
        interpolation="nearest",
    )
    plt.show()


'''
Epoch: 0001, Cost: 2.826302672
Epoch: 0002, Cost: 1.061668952
Epoch: 0003, Cost: 0.838061315
Epoch: 0004, Cost: 0.733232745
Epoch: 0005, Cost: 0.669279885
Epoch: 0006, Cost: 0.624611836
Epoch: 0007, Cost: 0.591160344
Epoch: 0008, Cost: 0.563868987
Epoch: 0009, Cost: 0.541745171
Epoch: 0010, Cost: 0.522673578
Epoch: 0011, Cost: 0.506782325
```

//From GitHub, Memo.보류

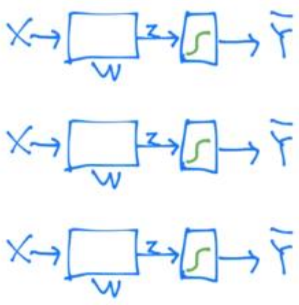
ML_Lec 08-1.

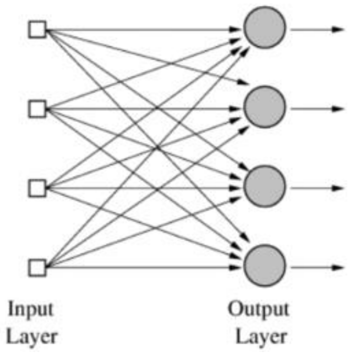
Activation Functions. //From a Neuron



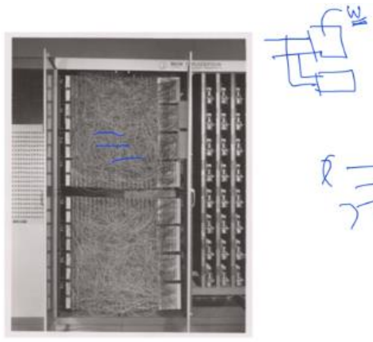
Logistic Regression Units.

전에 언급한 내용을 통해 여러 개의 출력을 동시에 낼 수 있다.





Hardware Implementations.

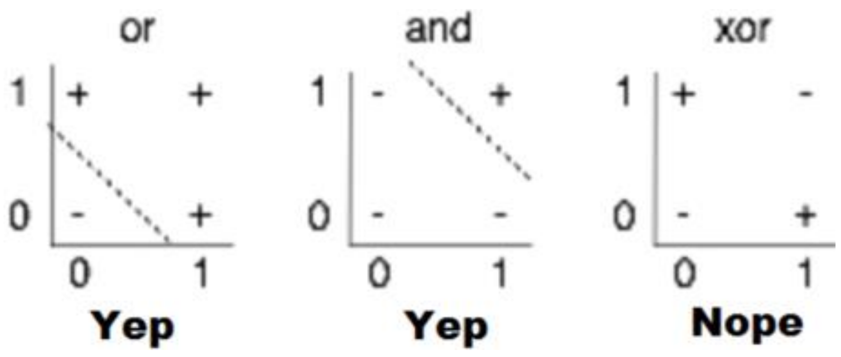


Frank Rosenblatt, ~1957: Perceptron

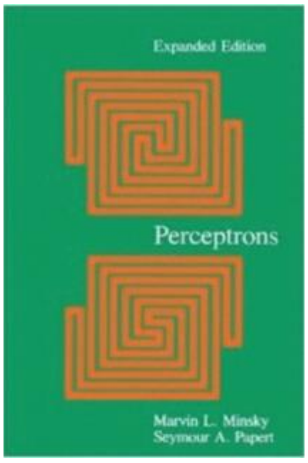
Widrow and Hoff, ~1960: Adaline/Madaline

AND/OR/XOR Problem: Linearly Separable? (Simple).

AND/OR 은 되나 XOR 은 불가능하다.



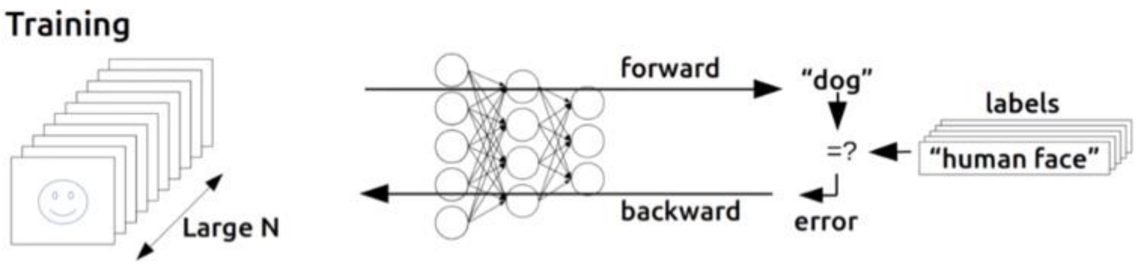
Perceptrons. //1969 by Marvin Minsky, founder of the MIT AI Lab



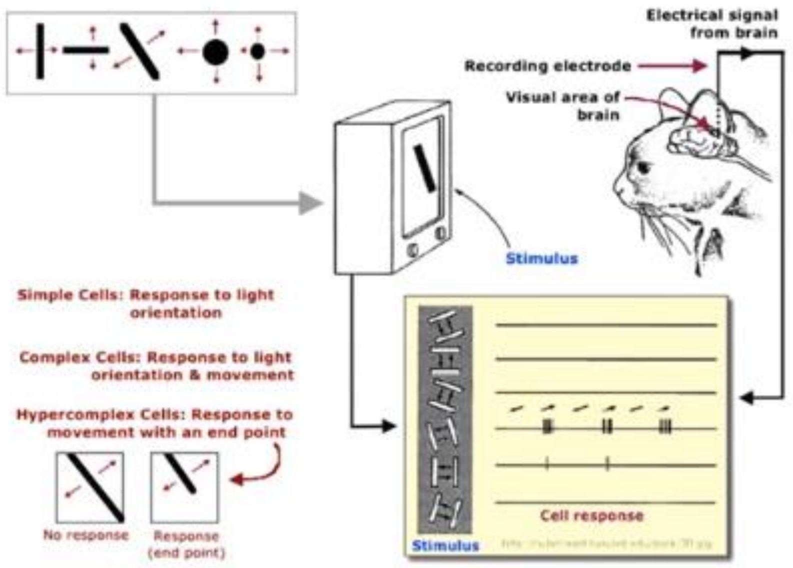
We need to use MLP, multilayer perceptrons (multilayer neural nets)

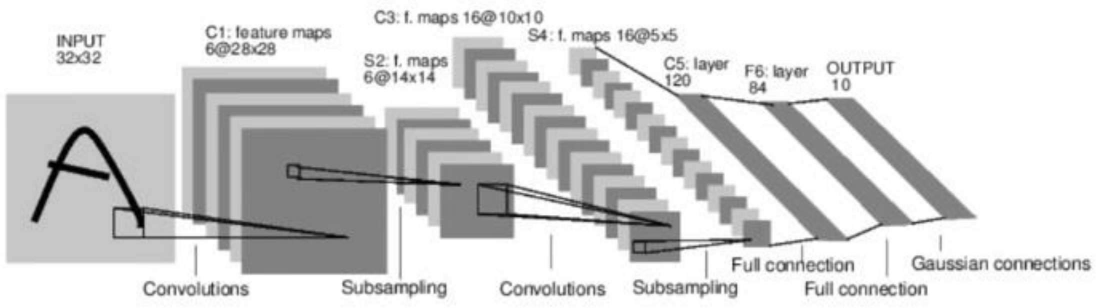
No one on earth had found a viable way to train MLPs good enough to learn such simple functions.

Backpropagtion. //역전파 //1974, 1982 by Paul Werbos, 1986 by Hinton

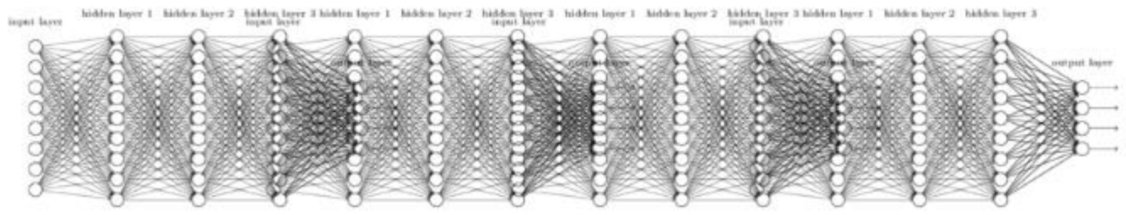


Convolutional Neural Networks. //1959 by Hubel & Wiesel

1. 
Electrical signal from brain
Recording electrode
Visual area of brain
Stimulus
Simple Cells: Response to light orientation
Complex Cells: Response to light orientation & movement
Hypercomplex Cells: Response to movement with an end point
No response
Response (end point)
Cell response
Stimulus

2. 

A BIG Problem.



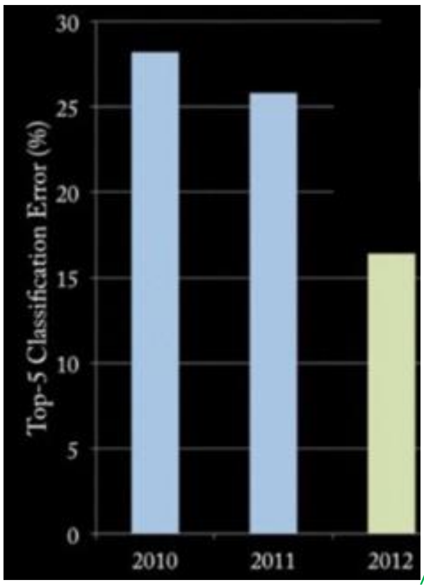
1. Backpropagation just did not work well for normal neural nets with many layers
2. Other rising machine learning algorithms: SVM, RandomForest, etc.
3. 1995 “Comparison of Learning Algorithms For Handwritten Digit Recognition” by LeCun et al.found that this new approach worked better.

ML_Lec 08-2.

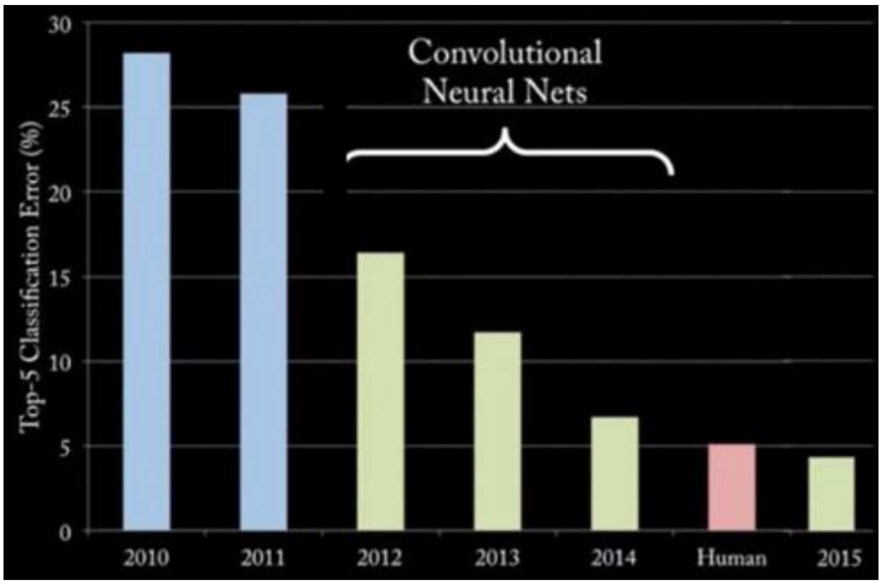
Breakthrough. //In 2006 and 2007 by Hinton and Bengio

1. Neural networks with many layers really could be trained well, if the weights are initialized in a clever way rather than randomly.
2. Deep machine learning methods are more efficient for difficult problems than shallow methods.
3. Rebranding to Deep Nets, Deep Learning.

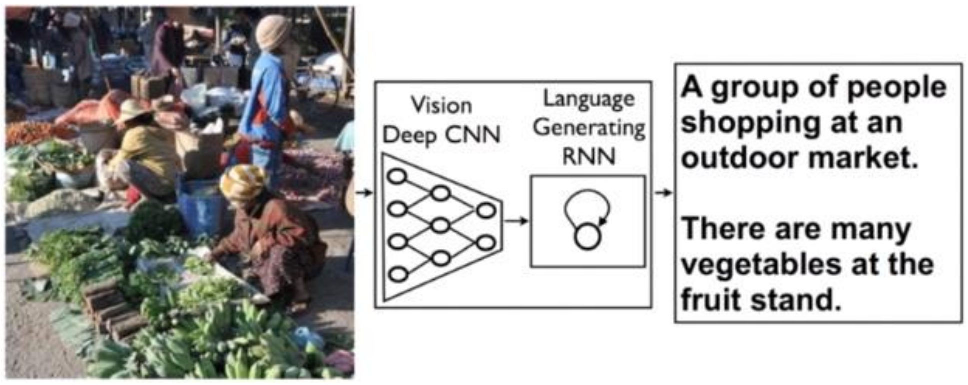
ImageNet Classification. //2010 – 2015



// Error 26.2% to 15.3%



Neural Networks That Can Explain Photos.


Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Deep API Learning.

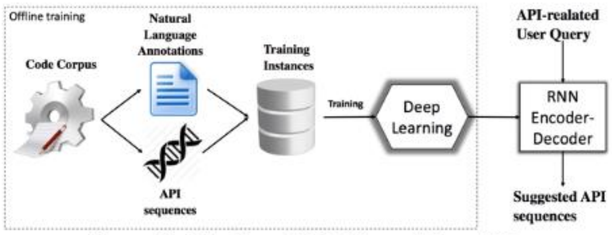
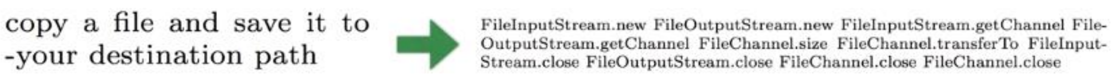


Figure 3: The Overall Workflow of DeepAPI


copy a file and save it to -your destination path
FileInputStream.new FileOutputStream.new FileInputStream.getChannel File-OutputStream.getChannel FileChannel.size FileChannel.transferTo FileInput-Stream.close FileOutputStream.close FileChannel.close FileChannel.close

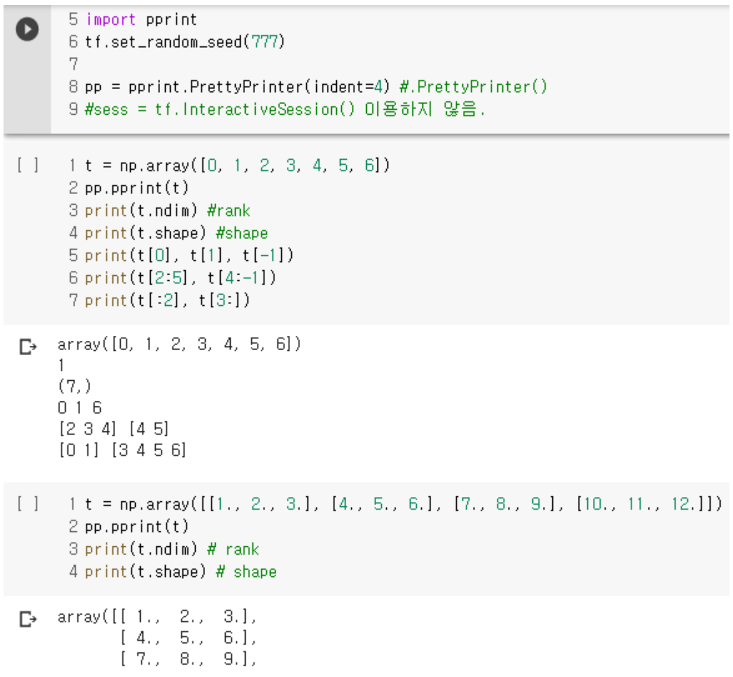
Geoffrey Hinton’s Summary of Findings Up to Today.

1. Our labeled datasets were thousands of times too small.
2. Our computers were millions of times too slow.
3. We initialized the weights in a stupid way.
4. We used the wrong type of non-linearity.

ML_Lab 08.

Tensor Manipulation.

Simple ID Array and Slicing Test.



```
import pprint
tf.set_random_seed(777)

pp = pprint.PrettyPrinter(indent=4) #.PrettyPrinter()
#sess = tf.InteractiveSession() 이용하지 않음.
```

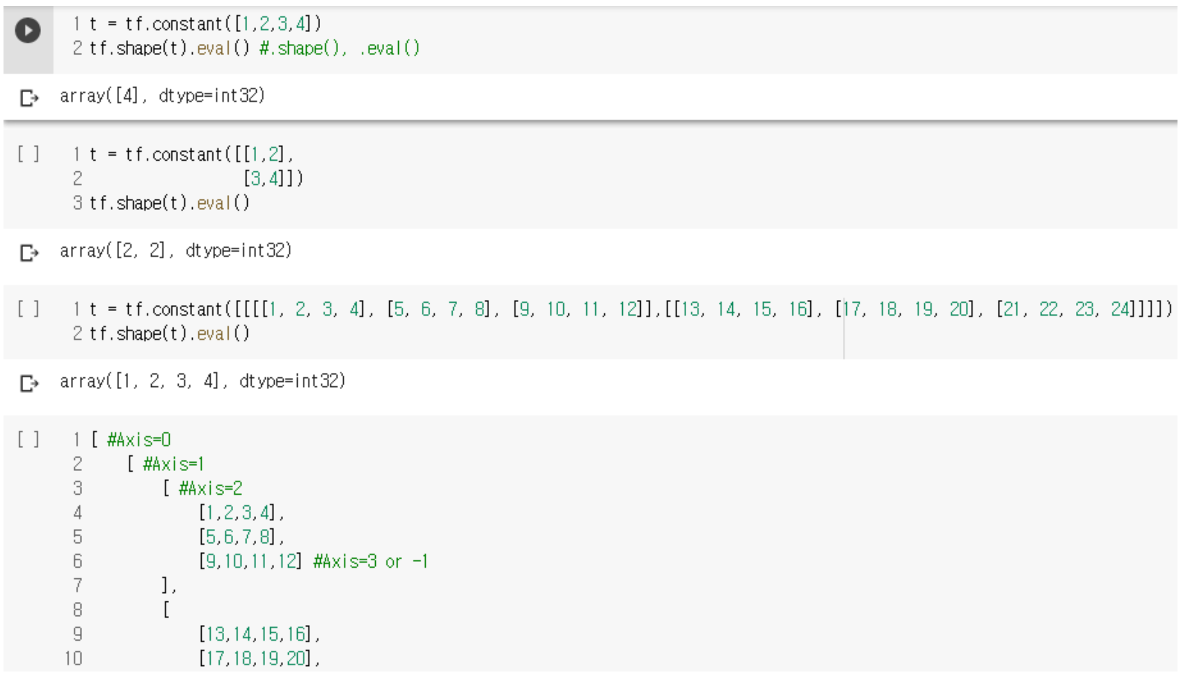
```
t = np.array([0, 1, 2, 3, 4, 5, 6])
pp.pprint(t)
print(t.ndim) #rank
print(t.shape) #shape
print(t[0], t[1], t[-1])
print(t[2:5], t[4:-1])
print(t[:2], t[3:])
```

```
array([0, 1, 2, 3, 4, 5, 6])
1
(7,)
0 1 6
[2 3 4] [4 5]
[0 1] [3 4 5 6]
```

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

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

Shape, Rank, Axis Test.



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

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

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

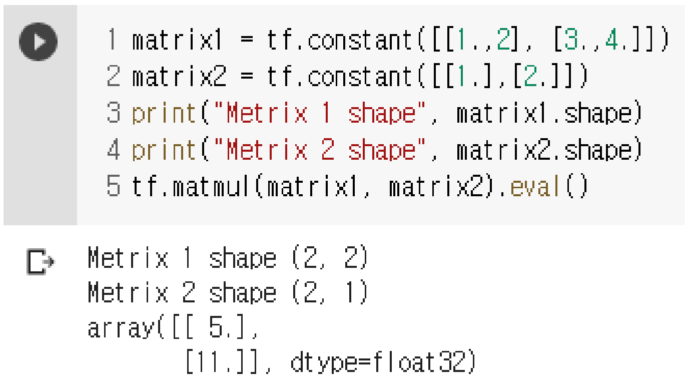
```
array([2, 2], dtype=int32)
```

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

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

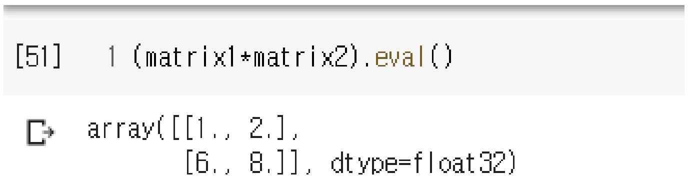
```
[ #Axis=0
    [ #Axis=1
        [ #Axis=2
            [1,2,3,4],
            [5,6,7,8],
            [9,10,11,12] #Axis=3 or -1
        ],
        [
            [13,14,15,16],
            [17,18,19,20],
```

Matmul & Multiply Test.



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

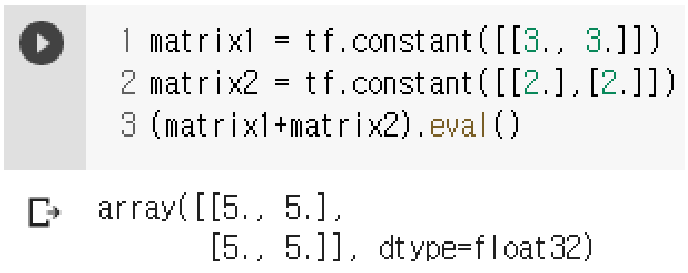
```
Metrix 1 shape (2, 2)
Metrix 2 shape (2, 1)
array([[ 5.],
       [11.]], dtype=float32)
```



```
[51]  (matrix1*matrix2).eval()
```

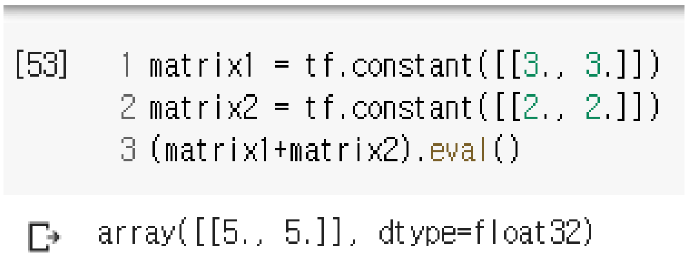
```
array([[1., 2.],
       [6., 8.]], dtype=float32)
```

Broadcasting Test.



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

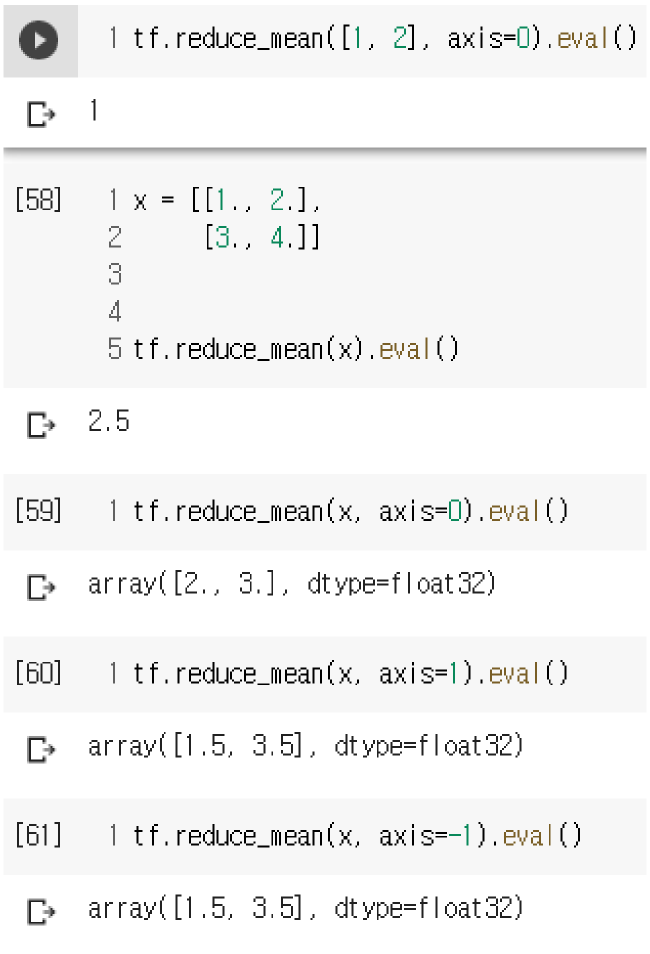
```
array([[5., 5.],
       [5., 5.]], dtype=float32)
```



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

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

Reduce_mean Test.



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

```
1
```

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


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

```
2.5
```

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

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

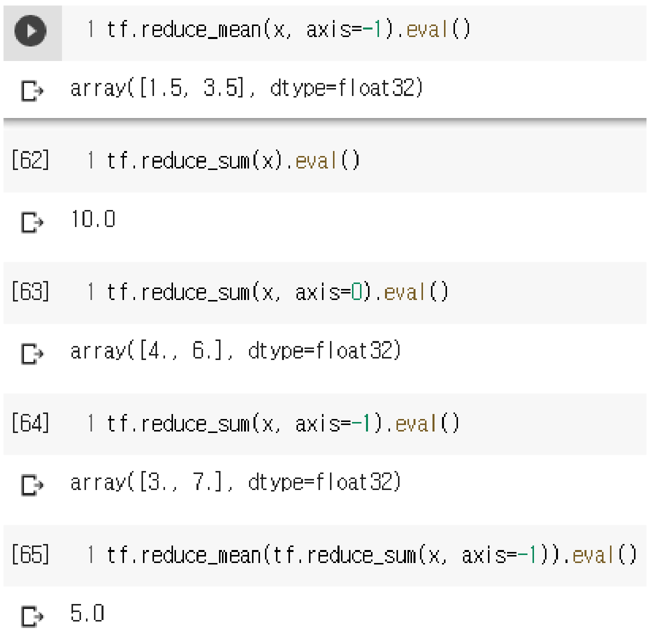
```
[60]  tf.reduce_mean(x, axis=1).eval()
```

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

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

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

Reduce_sum Test.



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

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

```
[62]  tf.reduce_sum(x).eval()
```

```
10.0
```

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

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

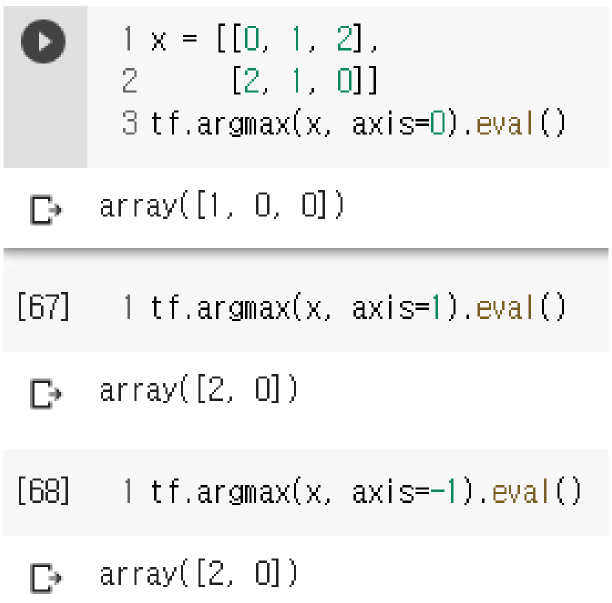
```
[64]  tf.reduce_sum(x, axis=-1).eval()
```

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

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

```
5.0
```

Argmax Test.



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

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

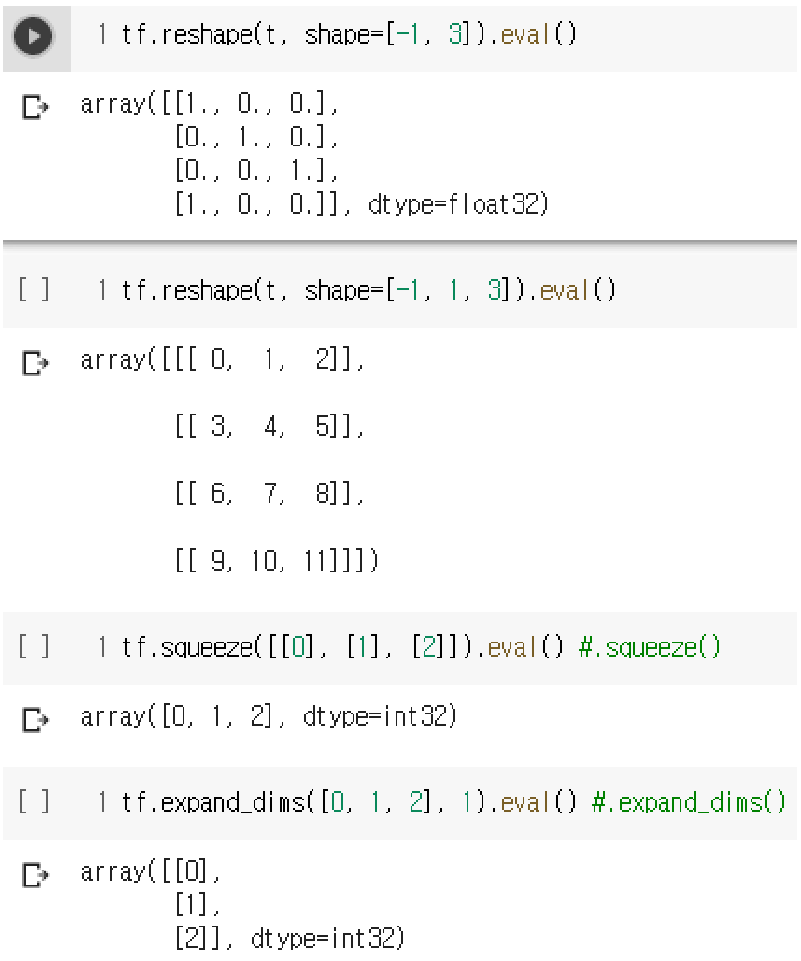
```
tf.argmax(x, axis=1).eval()
```

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

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

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

Reshape Test.



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

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

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

```
array([[[ 0,  1,  2]],

       [[ 3,  4,  5]],

       [[ 6,  7,  8]],

       [[ 9, 10, 11]]])
```

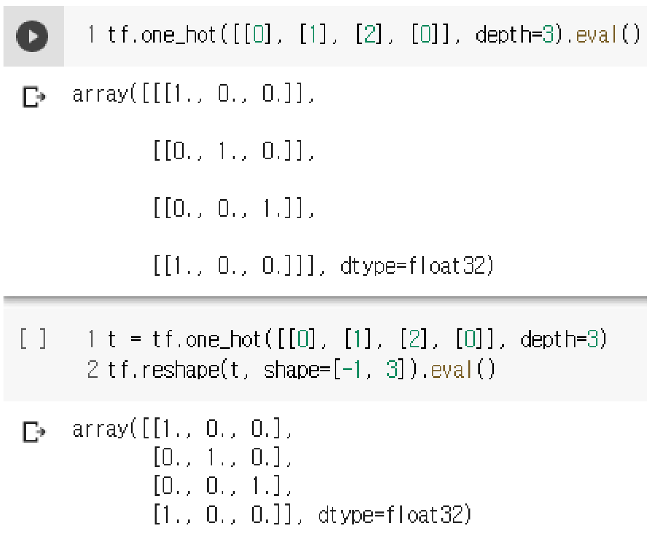
```
tf.squeeze([[0], [1], [2]]).eval() #.squeeze()
```

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

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

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

One_hot Test.



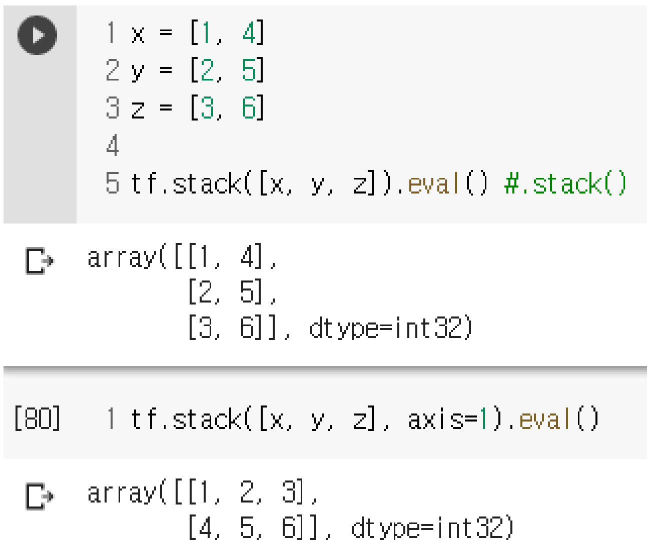
```
tf.one_hot([[0], [1], [2], [0]], depth=3).eval()
```

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

       [[0., 1., 0.]],

       [[0., 0., 1.]],

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

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

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

Stack Test.



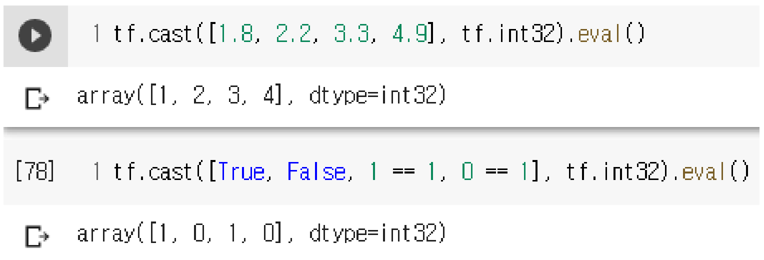
```
x = [1, 4]
y = [2, 5]
z = [3, 6]

tf.stack([x, y, z]).eval() #.stack()
```

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

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

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

Casting Test.



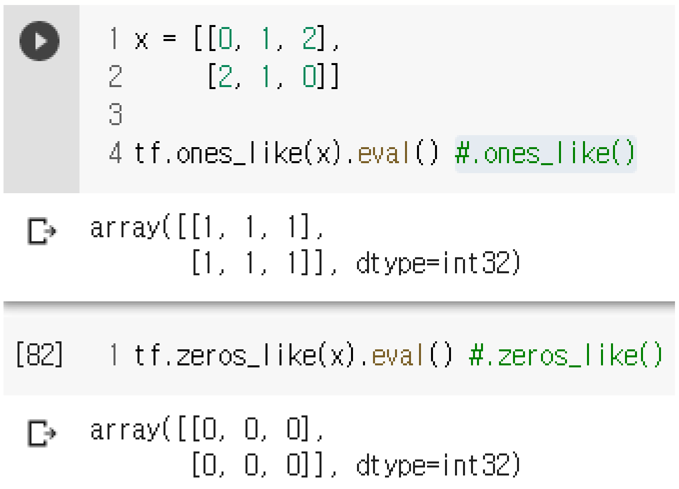
```
tf.cast([1.8, 2.2, 3.3, 4.9], tf.int32).eval()
```

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

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

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

Ones_like & Zeros_like Test.



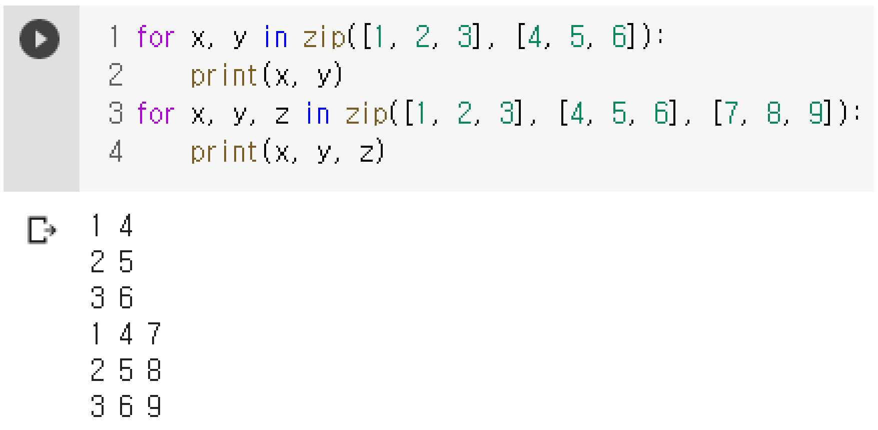
```python
x = [[0, 1, 2],
     [2, 1, 0]]

tf.ones_like(x).eval() #.ones_like()
```

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

```python
tf.zeros_like(x).eval() #.zeros_like()
```

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

Zip Test.



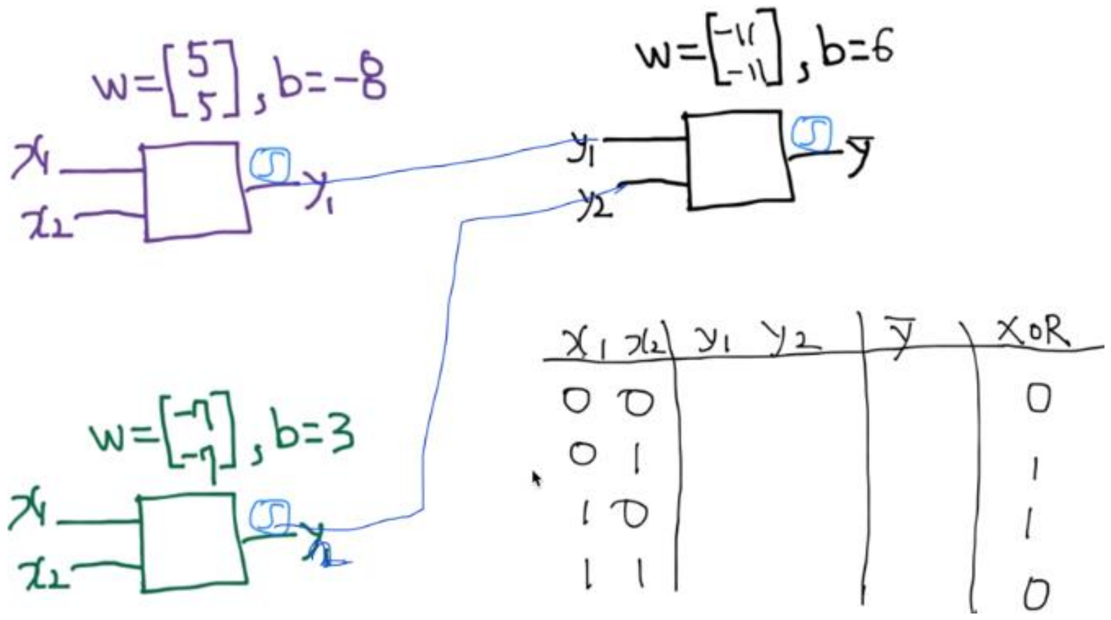
```python
for x, y in zip([1, 2, 3], [4, 5, 6]):
    print(x, y)
for x, y, z in zip([1, 2, 3], [4, 5, 6], [7, 8, 9]):
    print(x, y, z)
```

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

ML_Lec 09-1.

XOR Problem Using Neural Net(NN).

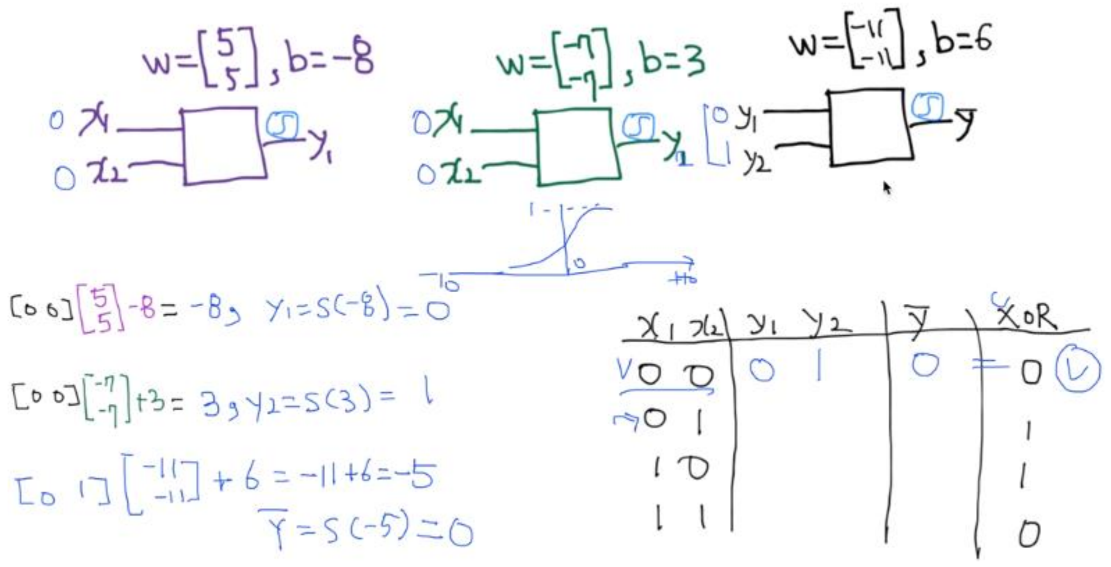
1. Multiple Logistic Regression Units.



$w=\begin{bmatrix}5\\5\end{bmatrix}, b=-8$
$w=\begin{bmatrix}-7\\-7\end{bmatrix}, b=3$
$w=\begin{bmatrix}-11\\-11\end{bmatrix}, b=6$
x_1
x_2
y_1
y_2
$\bar{y}$

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

2. 계산 과정.



$w=\begin{bmatrix}5\\5\end{bmatrix}, b=-8$
$w=\begin{bmatrix}-7\\-7\end{bmatrix}, b=3$
$w=\begin{bmatrix}-11\\-11\end{bmatrix}, b=6$
0 x_1
0 x_2
y_1
y_2
$\bar{y}$
-10
+10

$[0\ 0]\begin{bmatrix}5\\5\end{bmatrix}-8=-8,\ y_1=S(-8)=0$

$[0\ 0]\begin{bmatrix}-7\\-7\end{bmatrix}+3=3,\ y_2=S(3)=1$

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

$\bar{Y}=S(-5)=0$

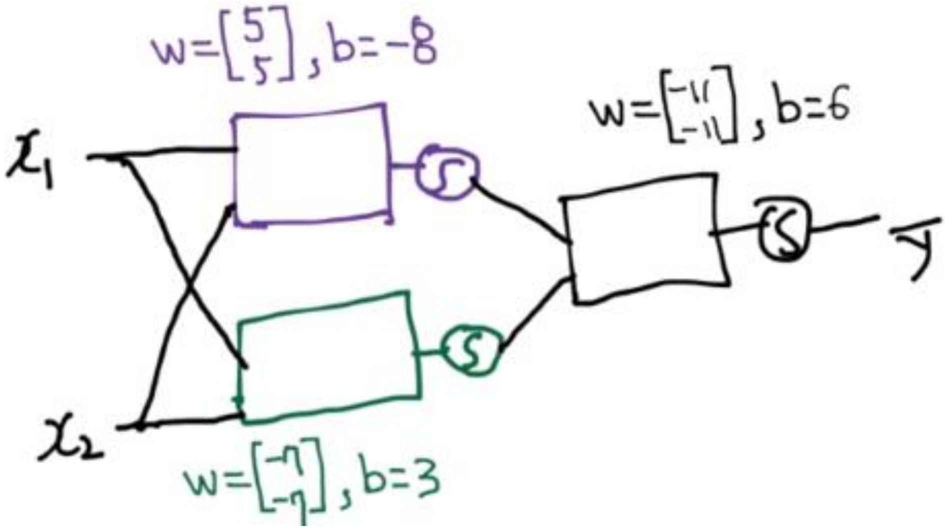
x_1	x_2	y_1	y_2	$\bar{y}$	XOR
0	0	0	1	0	0
0	1				1
1	0				1
1	1				0

1) x_1, x_2를 대입하여 Sigmoid Function 으로 y_1, y_2를 얻는다.

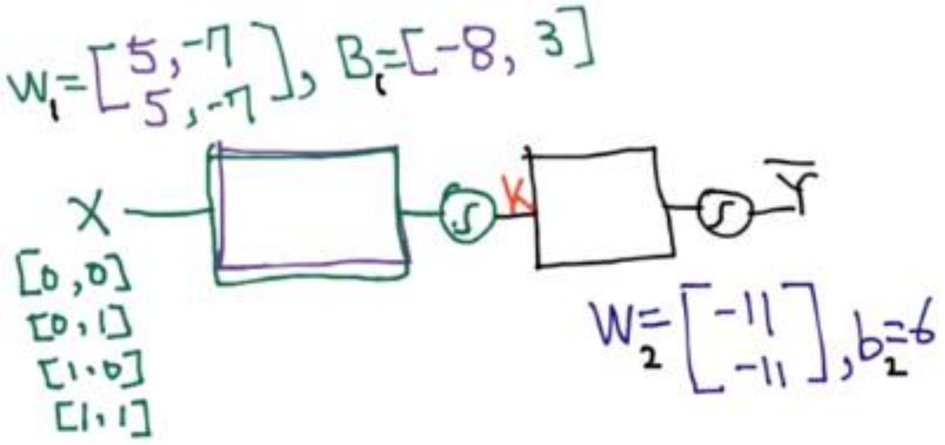
2) y_1, y_2를 대입하여 Sigmoid Function 으로 $\bar{y}$를 얻는다.

3) $\bar{y}$와 XOR 의 값을 확인한다.

Forward Propagation. //순전파

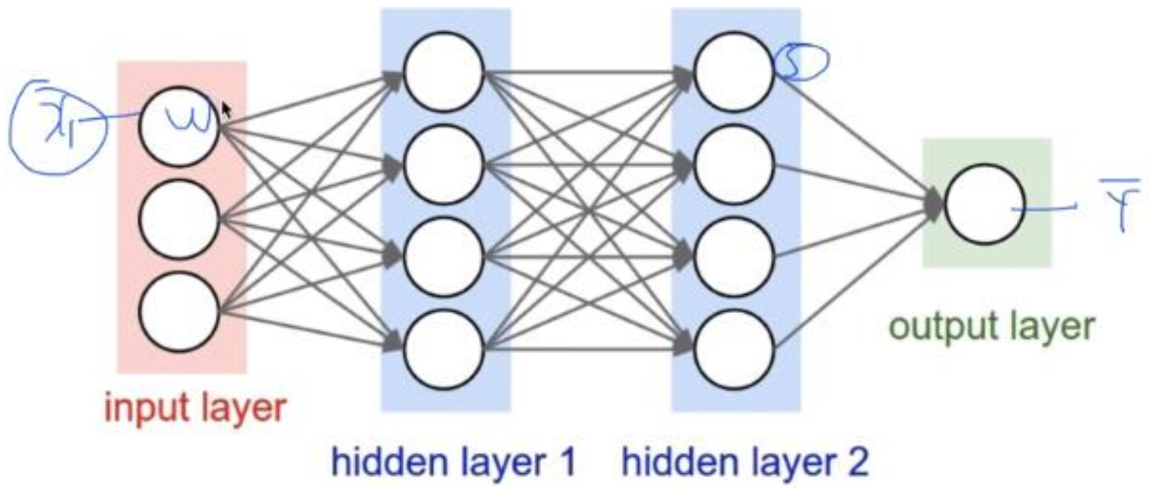


↓↓↓↓↓↓↓↓↓↓

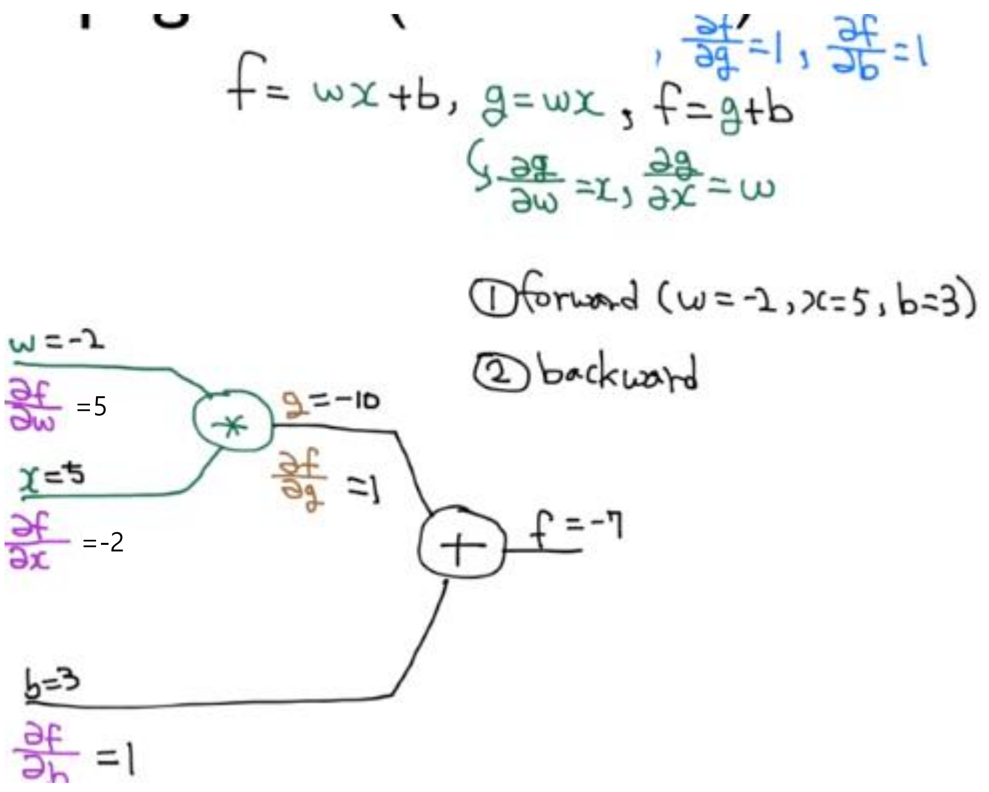


ML_Lec 09-2.

Derivation.



Backpropagation(Chain Rule). //역전파(합성함수의 미분)



$f = wx + b, \quad g = wx, \quad f = g + b$

$\frac{\partial f}{\partial g} = 1, \quad \frac{\partial f}{\partial b} = 1$

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

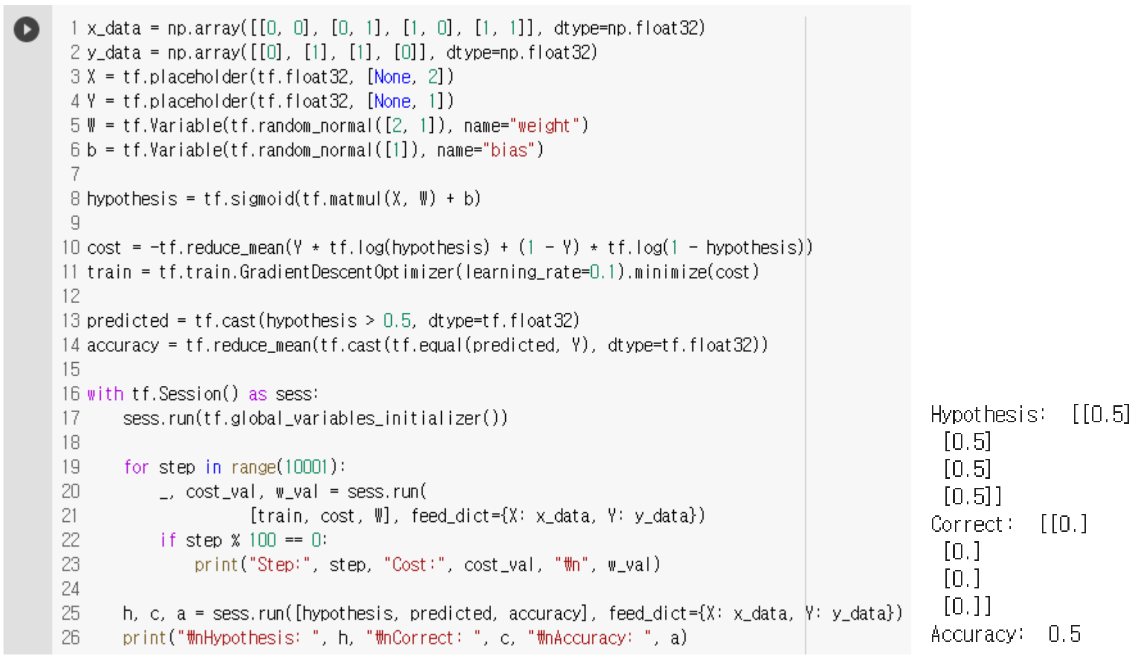
① forward $(w=-2, x=5, b=3)$

② backward

$w=-2$
$\frac{\partial f}{\partial w}$ =5
$x=5$
$\frac{\partial f}{\partial x}$ =-2
$*$
$g=-10$
$\frac{\partial f}{\partial g}=1$
$+$
$f=-7$
$b=3$
$\frac{\partial f}{\partial b}=1$

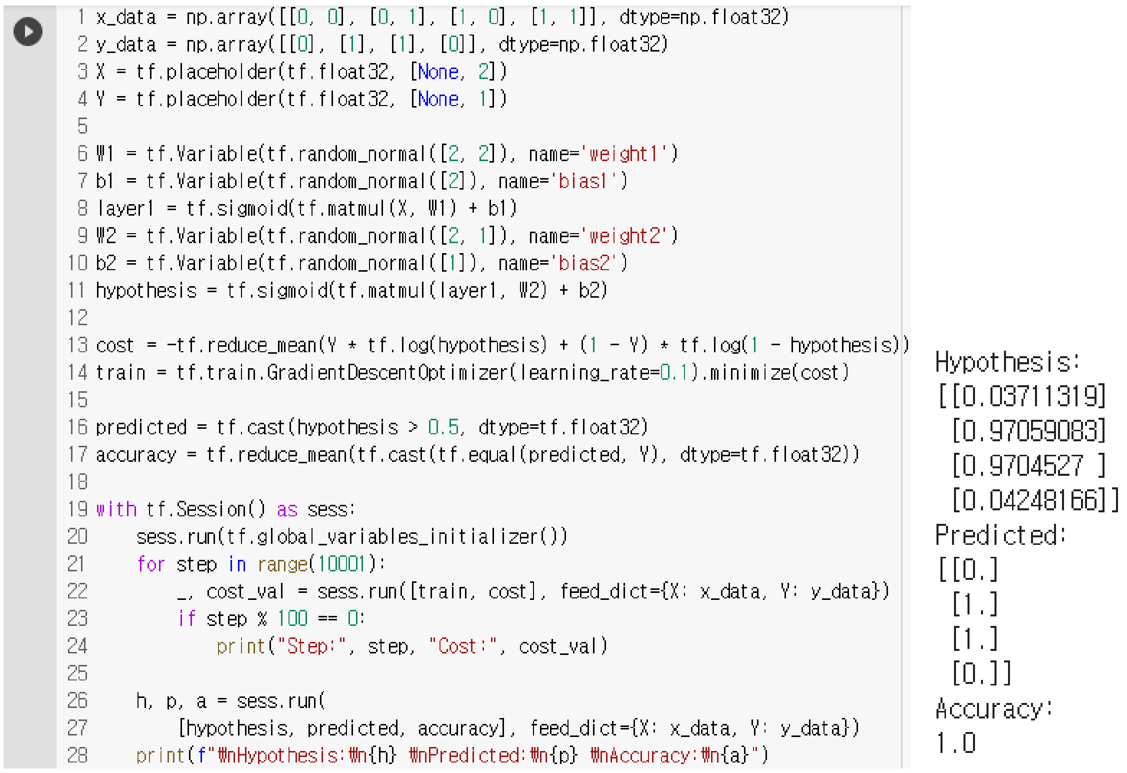
ML_Lab 09-1.

Test.



```
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])
W = tf.Variable(tf.random_normal([2, 1]), name="weight")
b = tf.Variable(tf.random_normal([1]), name="bias")

hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

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

predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

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

    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print("₩nHypothesis: ", h, "₩nCorrect: ", c, "₩nAccuracy: ", a)
```

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

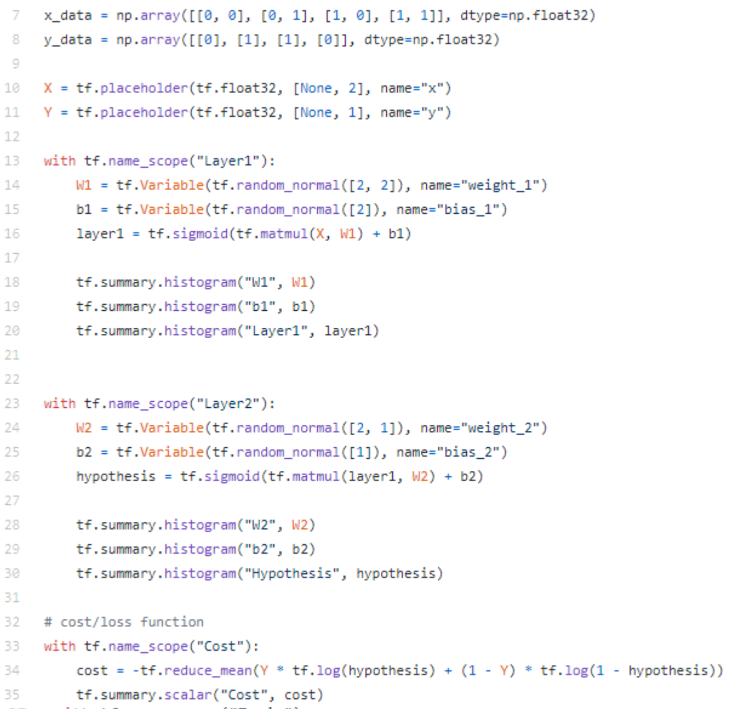


```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

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

predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

with tf.Session() as sess:
    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:", step, "Cost:", cost_val)

    h, p, a = sess.run(
        [hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print(f"₩nHypothesis:₩n{h} ₩nPredicted:₩n{p} ₩nAccuracy:₩n{a}")
```

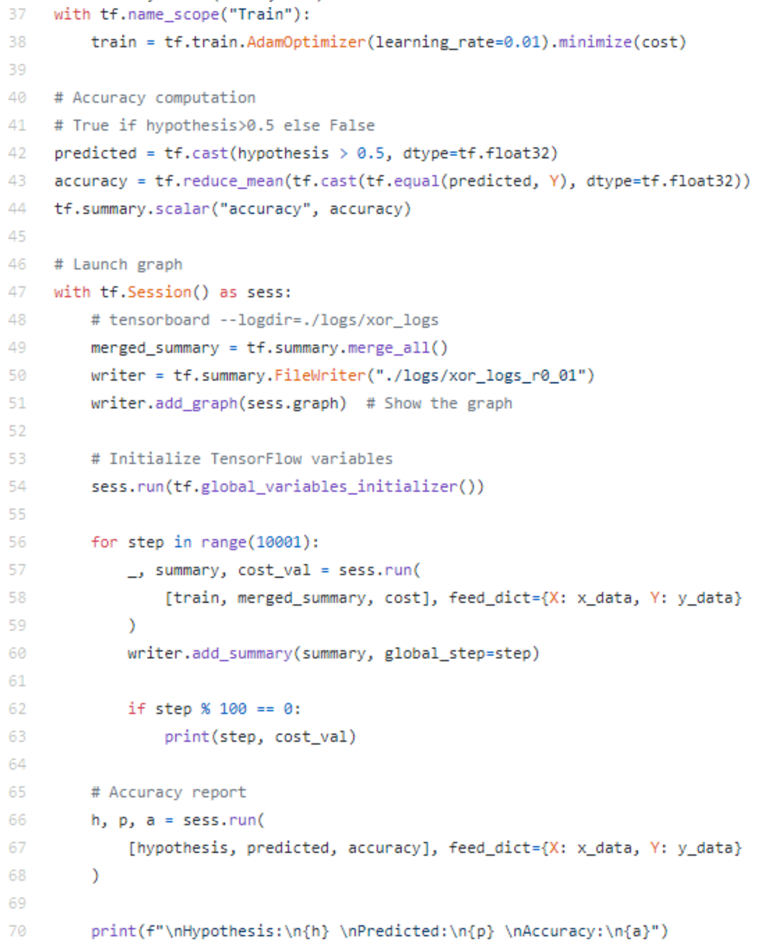
```
Hypothesis:
[[0.03711319]
 [0.97059083]
 [0.9704527 ]
 [0.04248166]]
Predicted:
[[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:
1.0
```

ML_Lab 09-2.

Test.



```python
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], name="x")
Y = tf.placeholder(tf.float32, [None, 1], name="y")

with tf.name_scope("Layer1"):
    W1 = tf.Variable(tf.random_normal([2, 2]), name="weight_1")
    b1 = tf.Variable(tf.random_normal([2]), name="bias_1")
    layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

    tf.summary.histogram("W1", W1)
    tf.summary.histogram("b1", b1)
    tf.summary.histogram("Layer1", layer1)


with tf.name_scope("Layer2"):
    W2 = tf.Variable(tf.random_normal([2, 1]), name="weight_2")
    b2 = tf.Variable(tf.random_normal([1]), name="bias_2")
    hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

    tf.summary.histogram("W2", W2)
    tf.summary.histogram("b2", b2)
    tf.summary.histogram("Hypothesis", hypothesis)

# cost/loss function
with tf.name_scope("Cost"):
    cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
    tf.summary.scalar("Cost", cost)
```

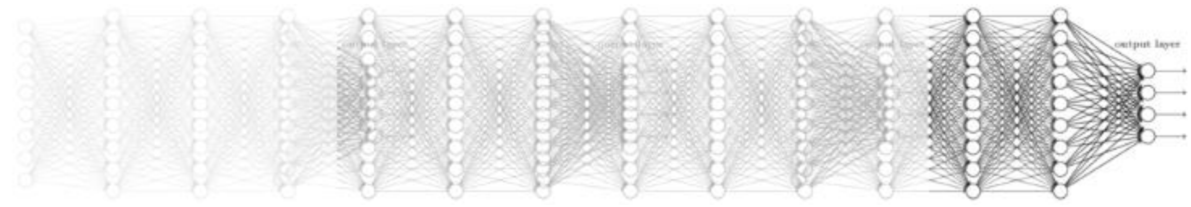


```python
with tf.name_scope("Train"):
    train = tf.train.AdamOptimizer(learning_rate=0.01).minimize(cost)

# 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))
tf.summary.scalar("accuracy", accuracy)

# Launch graph
with tf.Session() as sess:
    # tensorboard --logdir=./logs/xor_logs
    merged_summary = tf.summary.merge_all()
    writer = tf.summary.FileWriter("./logs/xor_logs_r0_01")
    writer.add_graph(sess.graph)  # Show the graph

    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        _, summary, cost_val = sess.run(
            [train, merged_summary, cost], feed_dict={X: x_data, Y: y_data}
        )
        writer.add_summary(summary, global_step=step)

        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(f"\nHypothesis:\n{h} \nPredicted:\n{p} \nAccuracy:\n{a}")
```

//From GitHub, Memo.보류

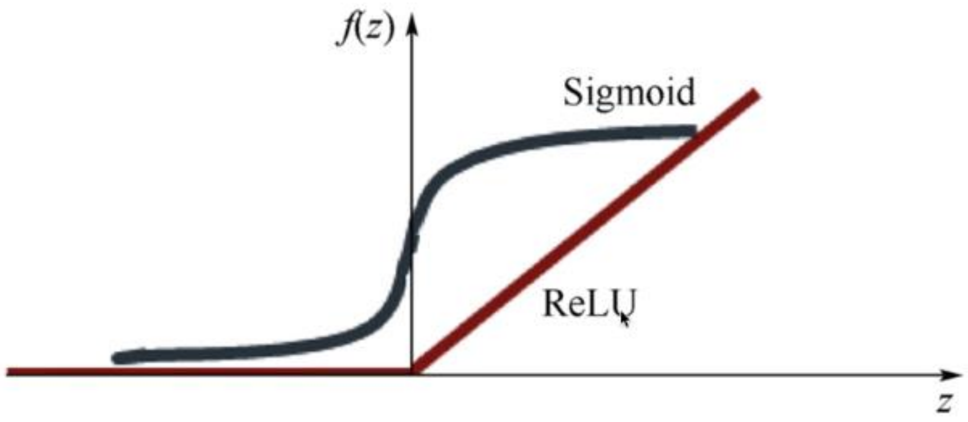
ML_Lec 10-1.

Vanishing gradient. //NN winter2: 1986-2006.

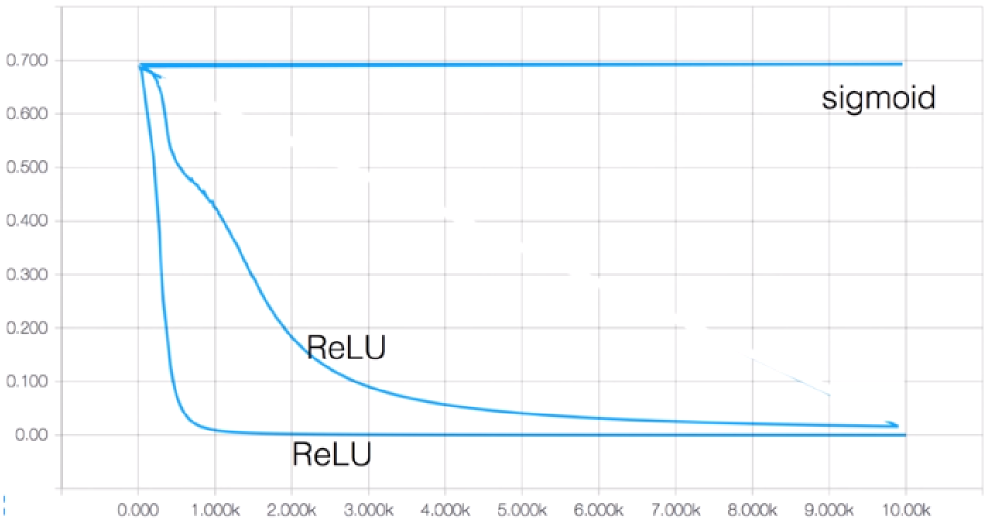


ReLU: Rectified Linear Unit.

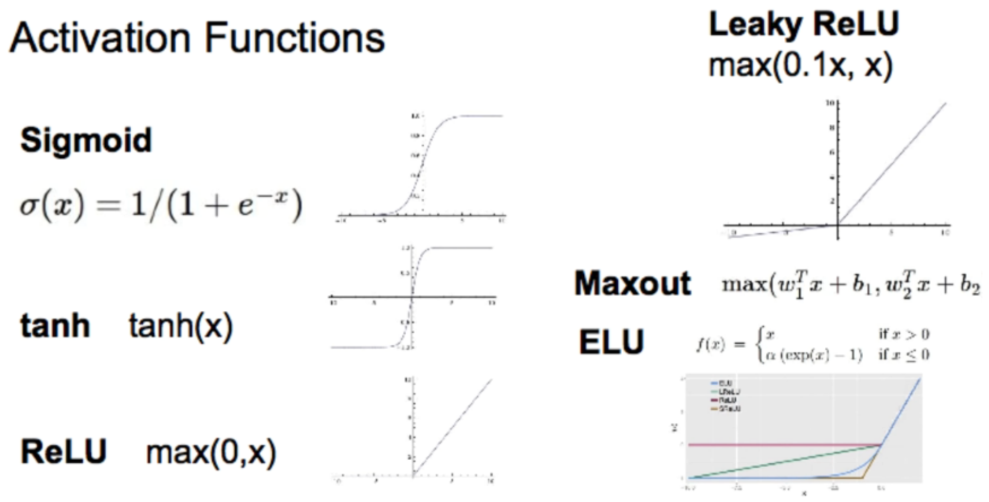
Code: L1 = tf.nn.relu(tf.matmul(X, W1) + b1)



Cost Function.



Other Activation Functions.



Activation Functions

Sigmoid

$\sigma(x) = 1/(1 + e^{-x})$

tanh tanh(x)

ReLU max(0,x)

Leaky ReLU
max(0.1x, x)

Maxout $\max(w_1^T x + b_1, w_2^T x + b_2$

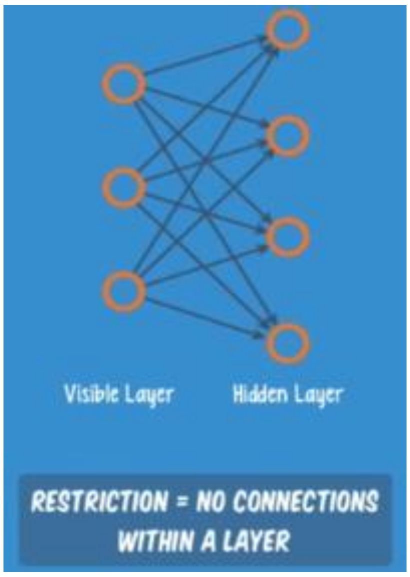
ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$

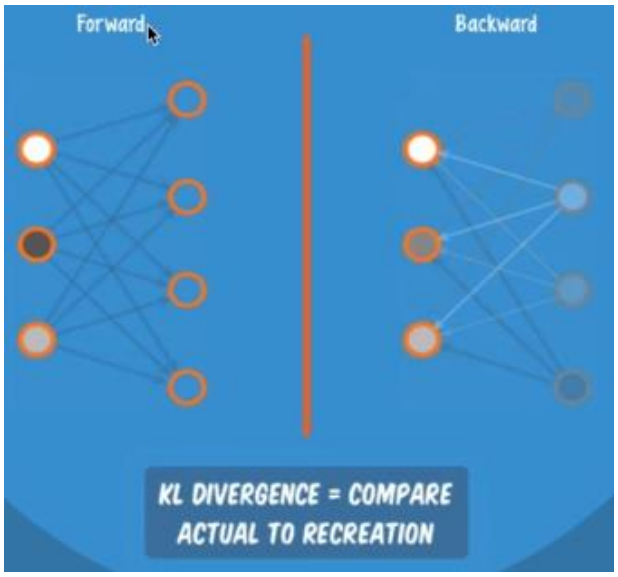
ML_Lec 10-2.

Need to Set the Initial Weight Values Wisely.

1. Not all 0's
2. Challenging issue
3. Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"-RBM

RBM(Restricted Boatman Machine) Structure.





1. Forward 를 통한 x 값과 Backward 를 통한 x 를 비교한다.
2. 가장 차이가 적도록 Weight 를 조정한다.

Xavier/He Initialization.

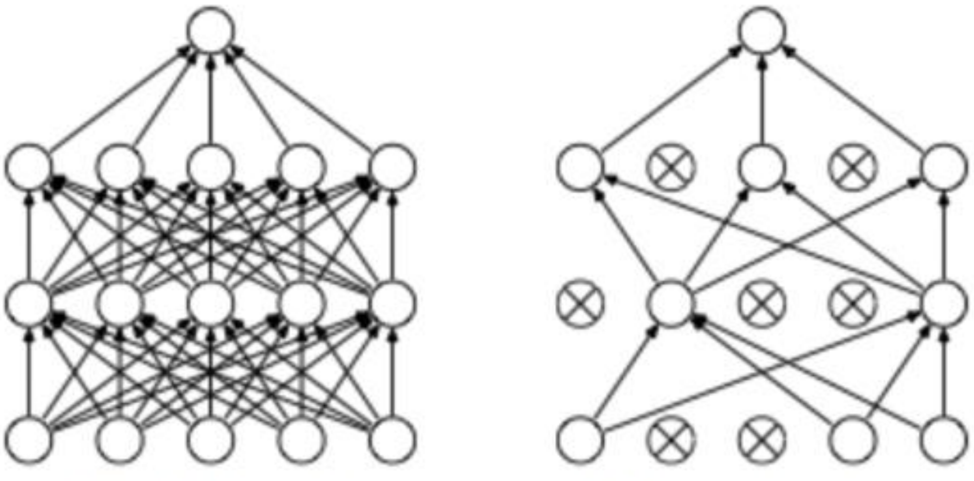
1. Makes sure the weights are 'just right', not too small, not to big
2. Using number of input (fan in) and output (fan out)
3. Code: W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in) //by 2015. (fan_in/2)

ML_Lec 10-3.

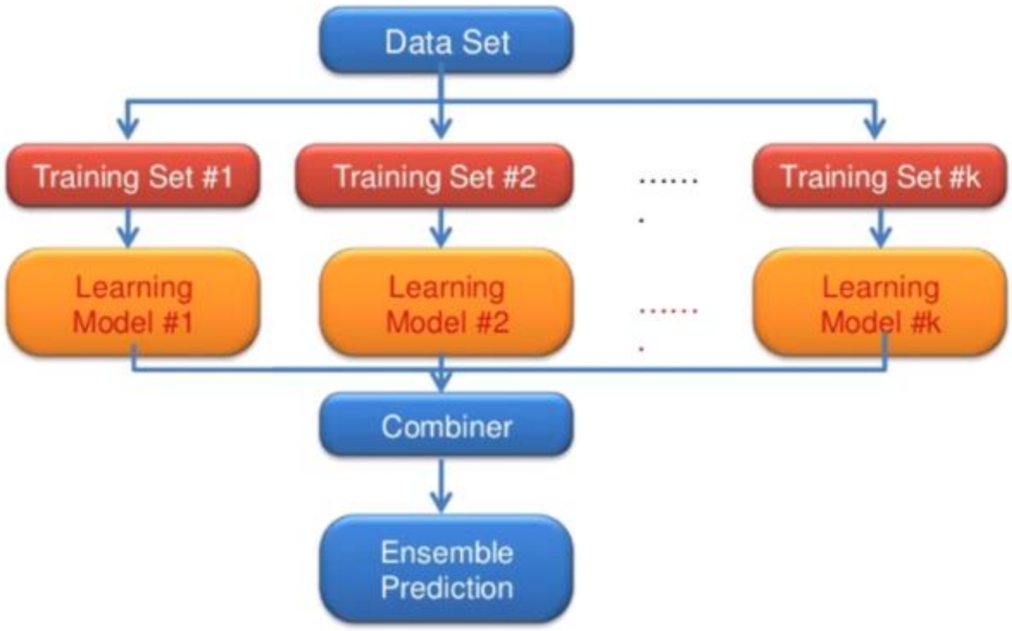
Dropout. //A Simple Way to Prevent NN from Overfitting (Srivastava et al.2014)

"randomly set some neurons to zero in the forward pass."

Code: L1 = tf.nn.dropout(_L1, dropout_rate)



Ensemble.

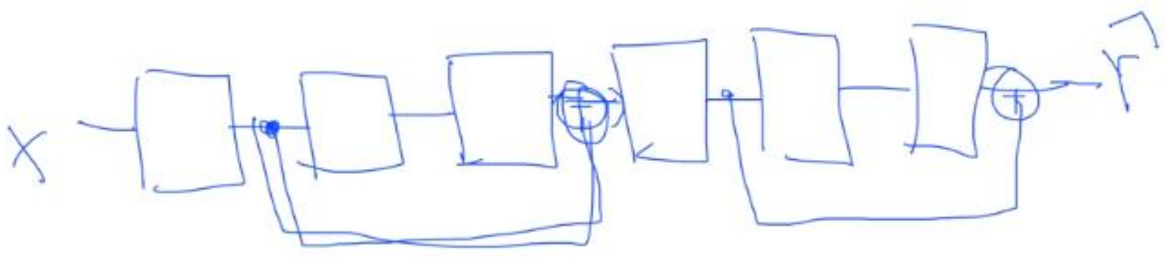


ML_Lec 10-4.

NN LEGO Play.

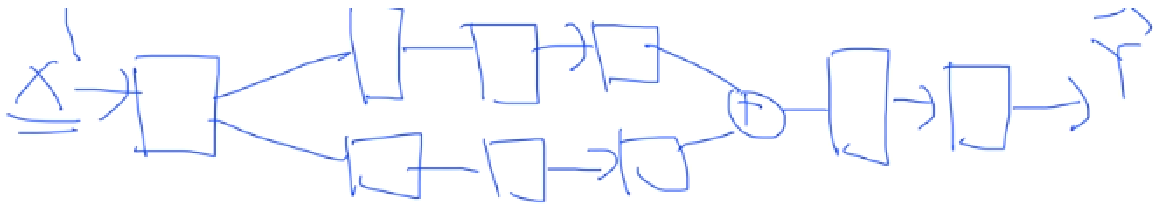
Fast Forward.

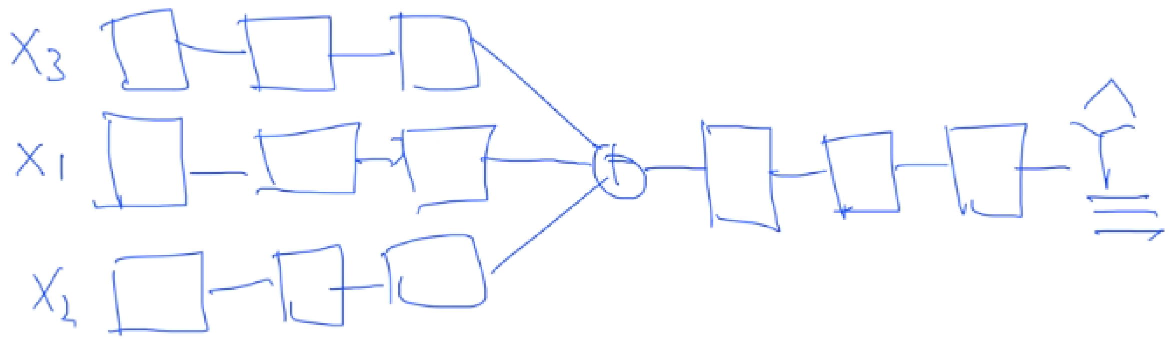
신호를 앞으로 당겨 계산한다.



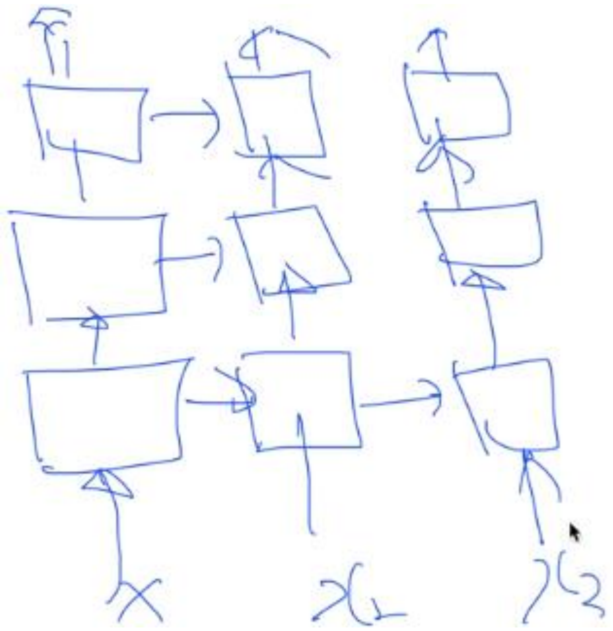
Split & Merge.

나뉘어 계산을 하고 나중에 모인다.



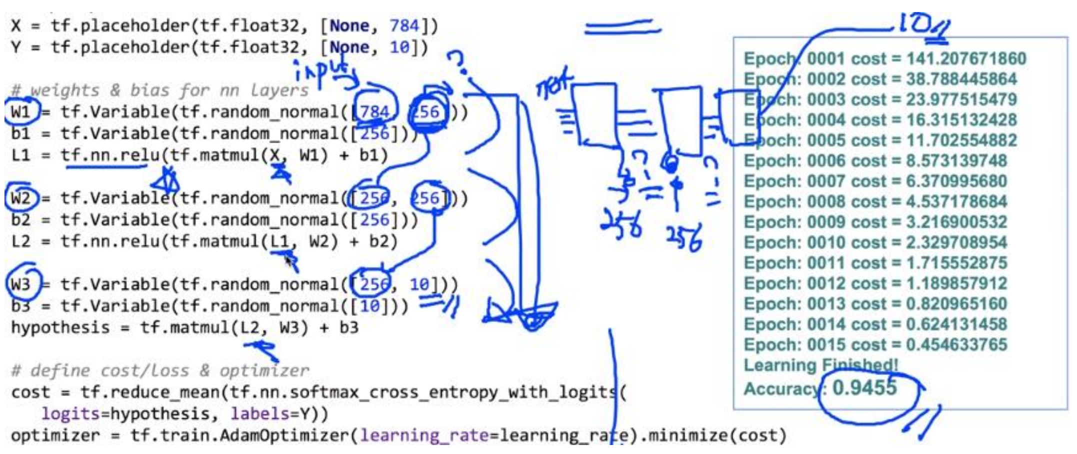


Recurrent Network(RNN).



ML_Lab 10.

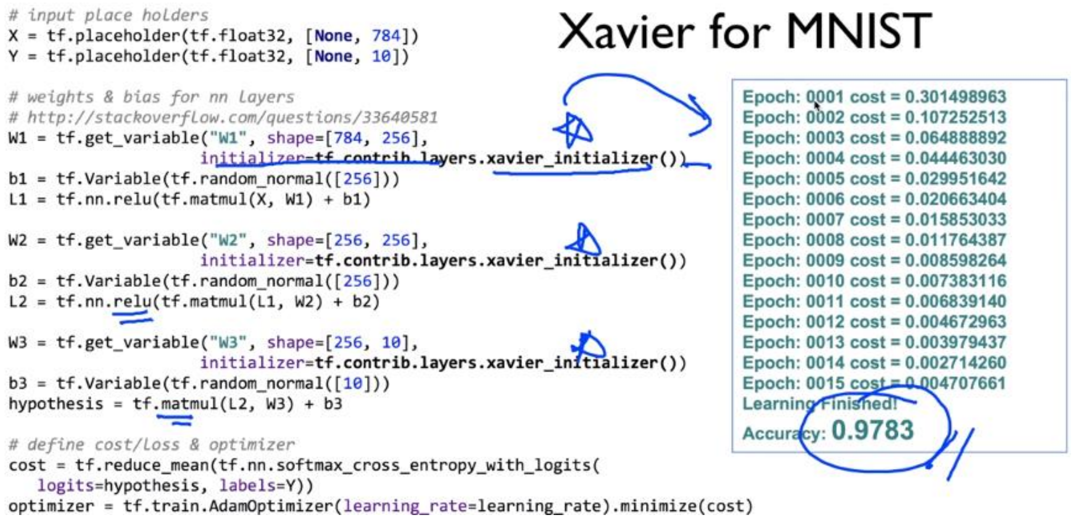
NN for MNIST Test. //Memo.보류



```
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
Epoch: 0001 cost = 141.207671860
Epoch: 0002 cost = 38.788445864
Epoch: 0003 cost = 23.977515479
Epoch: 0004 cost = 16.315132428
Epoch: 0005 cost = 11.702554882
Epoch: 0006 cost = 8.573139748
Epoch: 0007 cost = 6.370995680
Epoch: 0008 cost = 4.537178684
Epoch: 0009 cost = 3.216900532
Epoch: 0010 cost = 2.329708954
Epoch: 0011 cost = 1.715552875
Epoch: 0012 cost = 1.189857912
Epoch: 0013 cost = 0.820965160
Epoch: 0014 cost = 0.624131458
Epoch: 0015 cost = 0.454633765
Learning Finished!
Accuracy: 0.9455
```

Xavier for MNIST Test. //Memo.보류

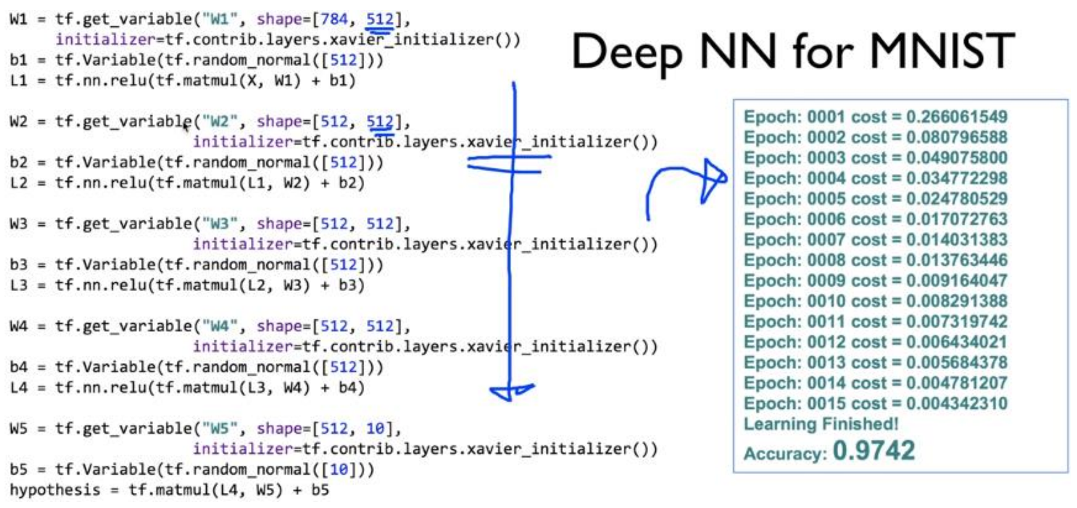


Xavier for MNIST

```
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581
W1 = tf.get_variable("W1", shape=[784, 256],
           initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]))
L1 = 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]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[256, 10],
           initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
Epoch: 0001 cost = 0.301498963
Epoch: 0002 cost = 0.107252513
Epoch: 0003 cost = 0.064888892
Epoch: 0004 cost = 0.044463030
Epoch: 0005 cost = 0.029951642
Epoch: 0006 cost = 0.020663404
Epoch: 0007 cost = 0.015853033
Epoch: 0008 cost = 0.011764387
Epoch: 0009 cost = 0.008598264
Epoch: 0010 cost = 0.007383116
Epoch: 0011 cost = 0.006839140
Epoch: 0012 cost = 0.004672963
Epoch: 0013 cost = 0.003979437
Epoch: 0014 cost = 0.002714260
Epoch: 0015 cost = 0.004707661
Learning Finished!
Accuracy: 0.9783
```

Deep NN for MNIST Test. //Memo.보류

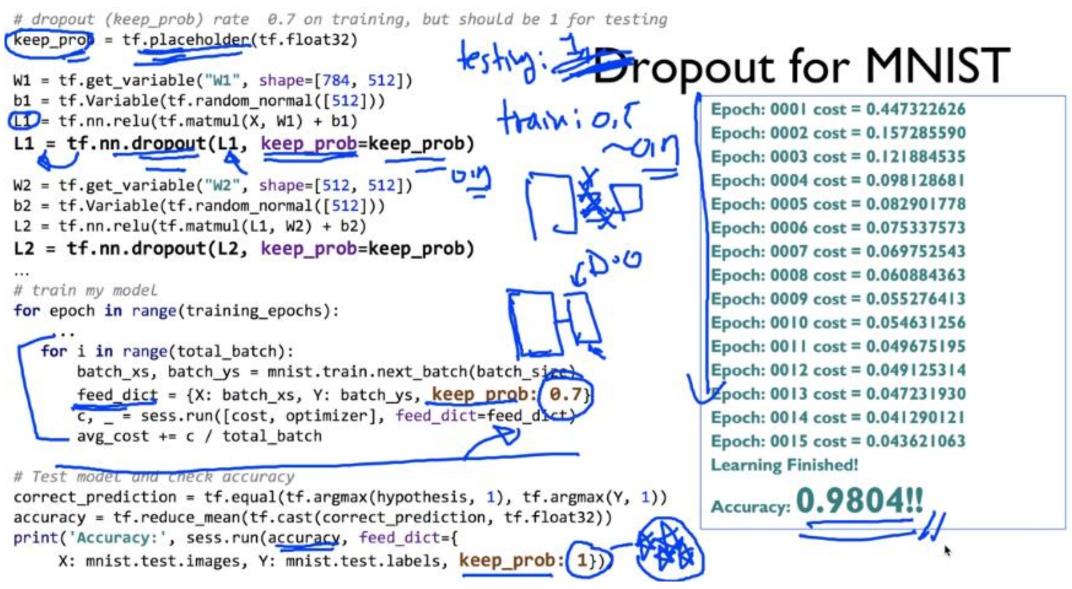


Deep NN for MNIST

```
W1 = tf.get_variable("W1", shape=[784, 512],
     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[512, 512],
           initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[512, 512],
           initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)

W4 = tf.get_variable("W4", shape=[512, 512],
           initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)

W5 = tf.get_variable("W5", shape=[512, 10],
           initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5
```

```
Epoch: 0001 cost = 0.266061549
Epoch: 0002 cost = 0.080796588
Epoch: 0003 cost = 0.049075800
Epoch: 0004 cost = 0.034772298
Epoch: 0005 cost = 0.024780529
Epoch: 0006 cost = 0.017072763
Epoch: 0007 cost = 0.014031383
Epoch: 0008 cost = 0.013763446
Epoch: 0009 cost = 0.009164047
Epoch: 0010 cost = 0.008291388
Epoch: 0011 cost = 0.007319742
Epoch: 0012 cost = 0.006434021
Epoch: 0013 cost = 0.005684378
Epoch: 0014 cost = 0.004781207
Epoch: 0015 cost = 0.004342310
Learning Finished!
Accuracy: 0.9742
```

Dropout for MNIST Test. //Memo.보류



Dropout for MNIST

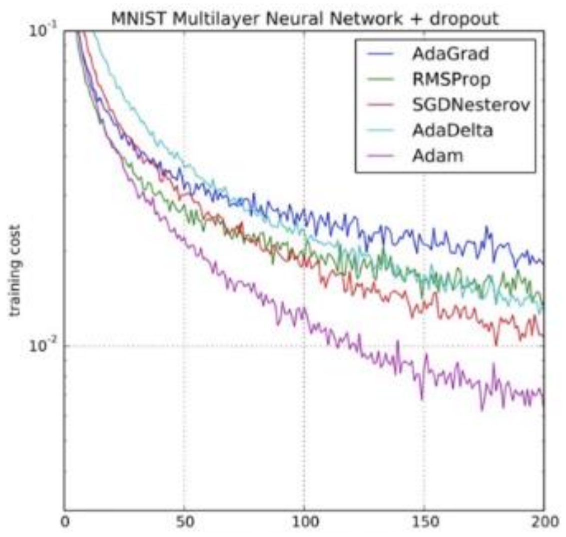
```
# dropout (keep_prob) rate  0.7 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

W1 = tf.get_variable("W1", shape=[784, 512])
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

W2 = tf.get_variable("W2", shape=[512, 512])
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
...
# train my model
for epoch in range(training_epochs):
    ...
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
```

```
Epoch: 0001 cost = 0.447322626
Epoch: 0002 cost = 0.157285590
Epoch: 0003 cost = 0.121884535
Epoch: 0004 cost = 0.098128681
Epoch: 0005 cost = 0.082901778
Epoch: 0006 cost = 0.075337573
Epoch: 0007 cost = 0.069752543
Epoch: 0008 cost = 0.060884363
Epoch: 0009 cost = 0.055276413
Epoch: 0010 cost = 0.054631256
Epoch: 0011 cost = 0.049675195
Epoch: 0012 cost = 0.049125314
Epoch: 0013 cost = 0.047231930
Epoch: 0014 cost = 0.041290121
Epoch: 0015 cost = 0.043621063
Learning Finished!
Accuracy: 0.9804!!
```

Optimizer.

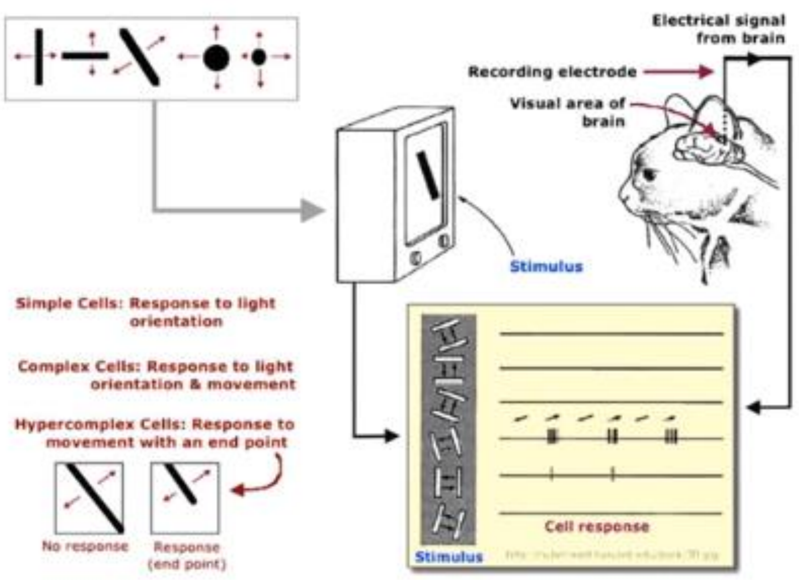
- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer

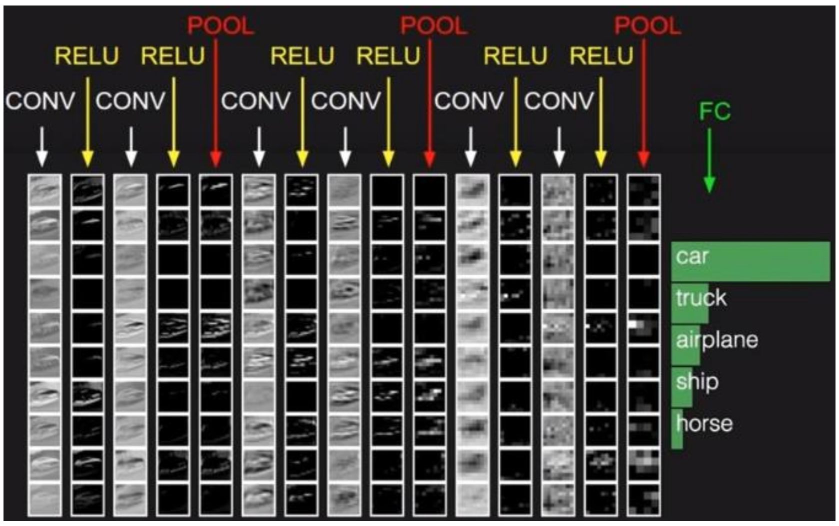
ADAM. //A Method for Stochastic Optimaization (Kingma et al.2015)



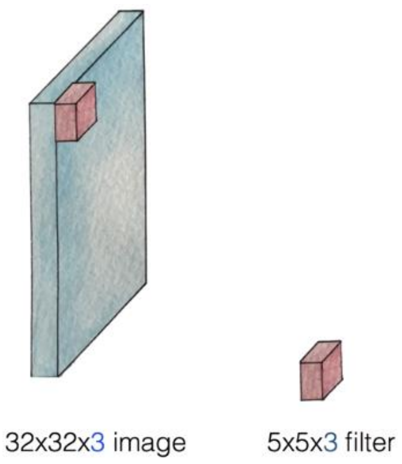
ML_Lec 11-1.

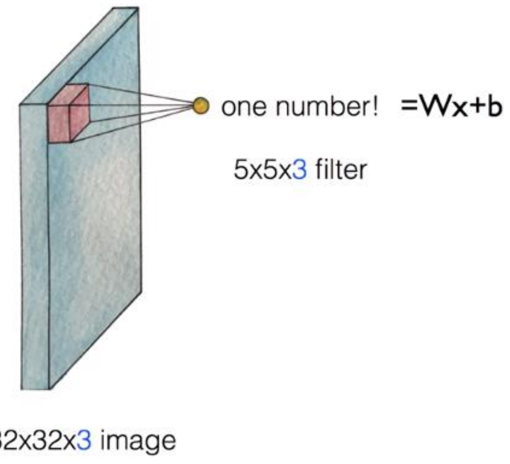
Convolutional Neural Networks. //1959 by Hubel & Wiesel

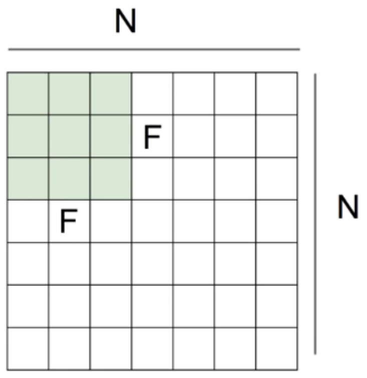
1. 
Electrical signal from brain
Recording electrode
Visual area of brain
Stimulus
Simple Cells: Response to light orientation
Complex Cells: Response to light orientation & movement
Hypercomplex Cells: Response to movement with an end point
No response
Response (end point)
Cell response
Stimulus

2. 

ConvNet. CNN.

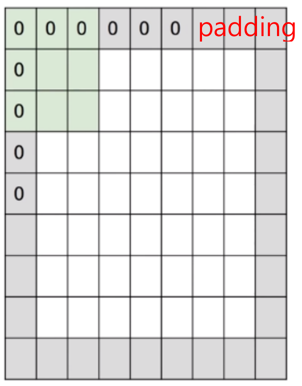
1. 

2. 

3. 

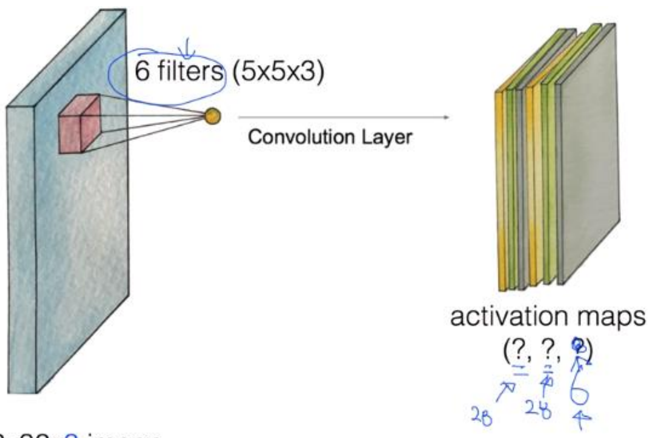
Output size:
(N - F) / stride + 1

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 :\

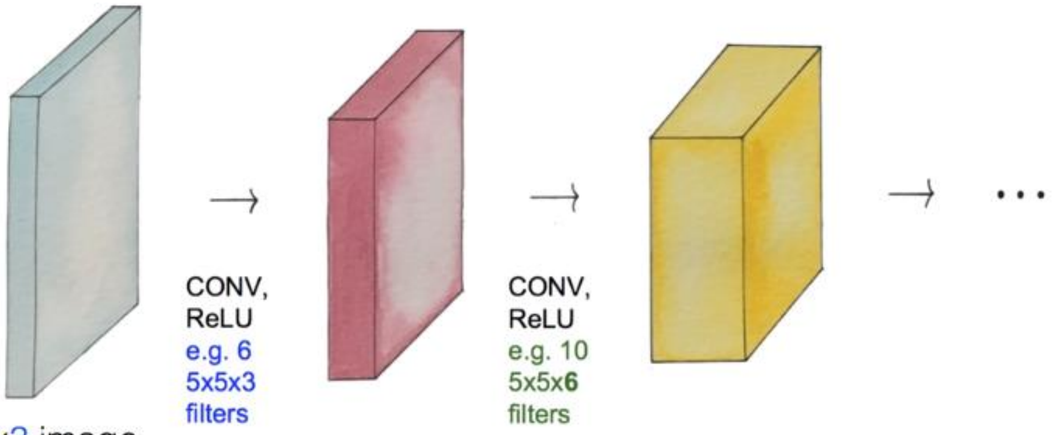
4. 

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3



5. 32x32x3 image //(32-5)/1+1=28

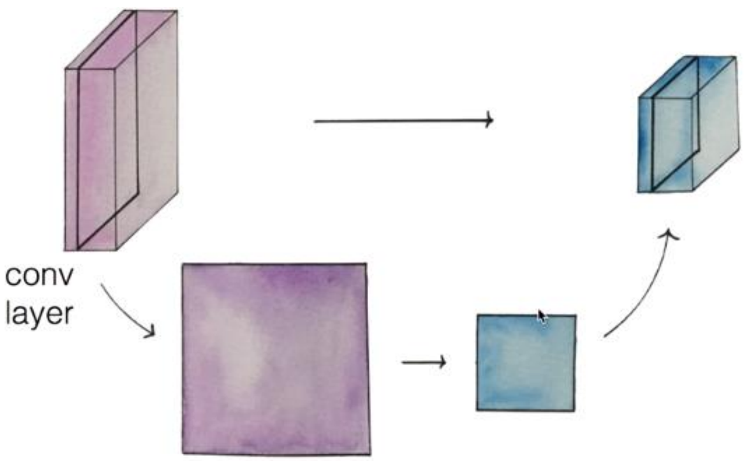


6. 32x32x3 image

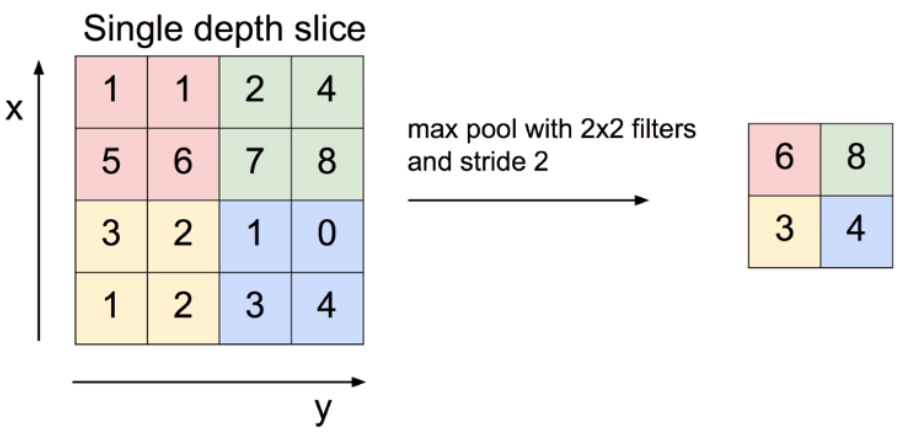
//w=5x5x3x6

ML_Lec 11-2.

Pooling Layer (Sampling).



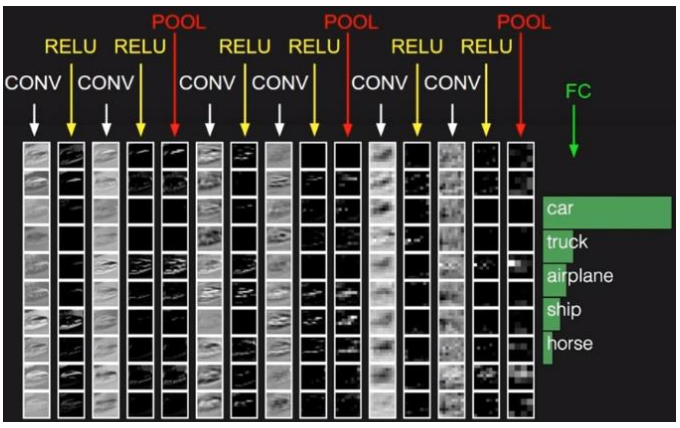
Max Pooling.



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

6	8
3	4

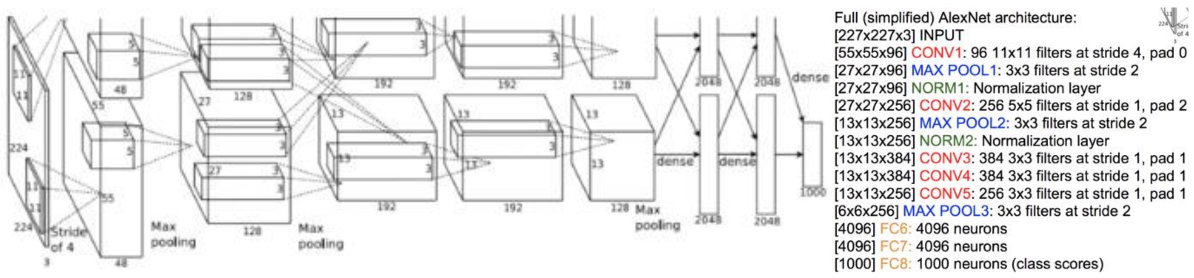
Fully Connected(FC) Layer.



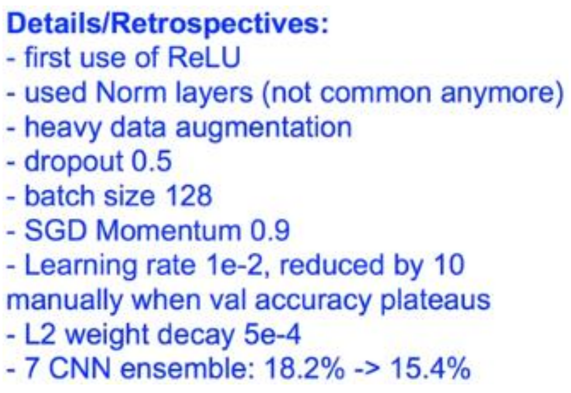
ML_Lec 11-3.

ConvNet 의 활용.

AlexNet. //Krizhevsky et al.2012

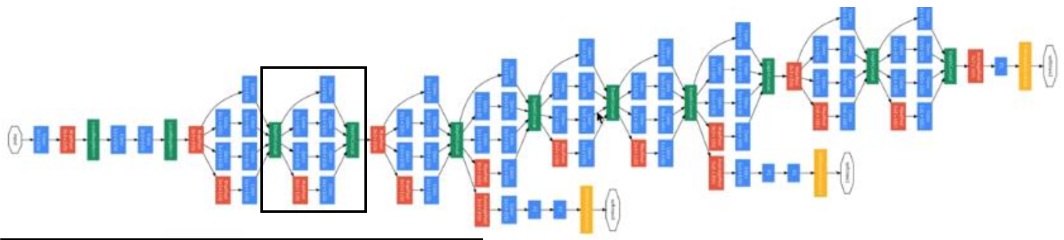


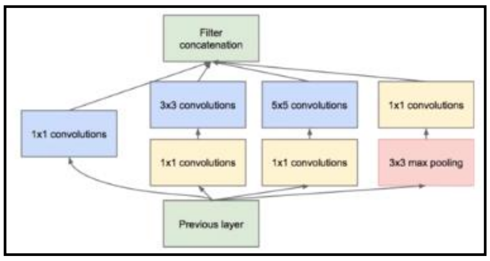
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)

1. Input: 227x227x3 Images
2. First Layer(CONV): 96 11x11 Filters applied at Stride 4
   1) Output Volume: 55x55x96
   2) Parameters: (11*11*3)*96 = 35K
3. Second Layer(FOOL): 3x3 Filters applied at Stride 2
   1) Output Volume: 27x27x96
   2) Parameters: 0!
4. NORM: Normalization Layer
5. 

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

GoogLeNet. //Szegedy et al.2014





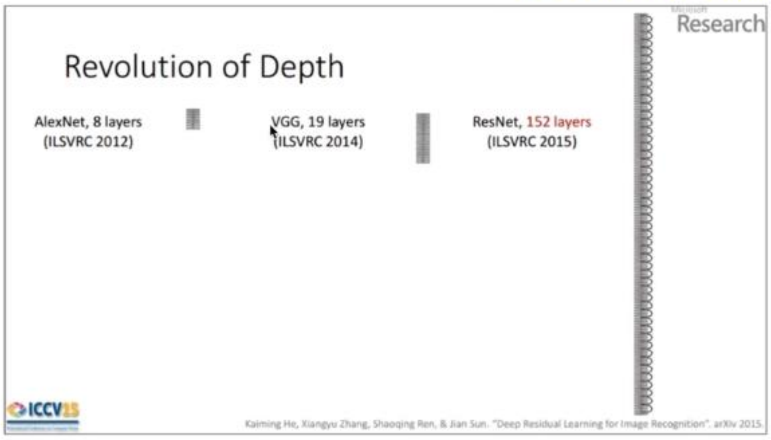
Inception module

ILSVRC 2014 winner (6.7% top 5 error)

ResNet //He et al.2015

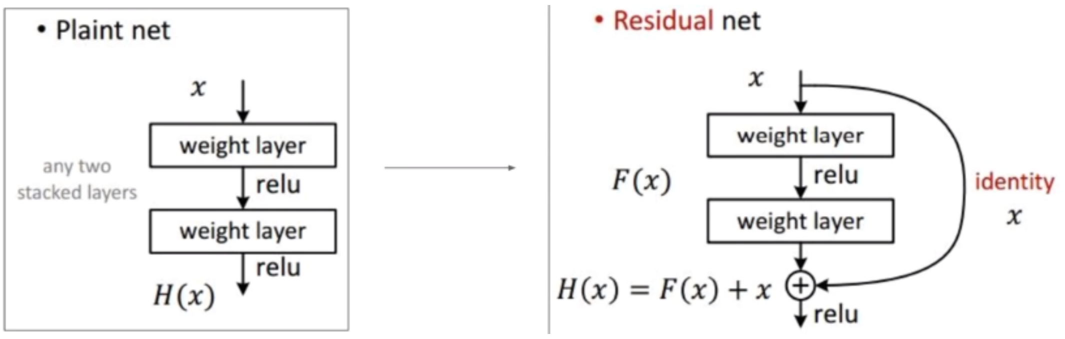
Case Study: ResNet [He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)

1. 
Research
Revolution of Depth
AlexNet, 8 layers (ILSVRC 2012)
VGG, 19 layers (ILSVRC 2014)
ResNet, 152 layers (ILSVRC 2015)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

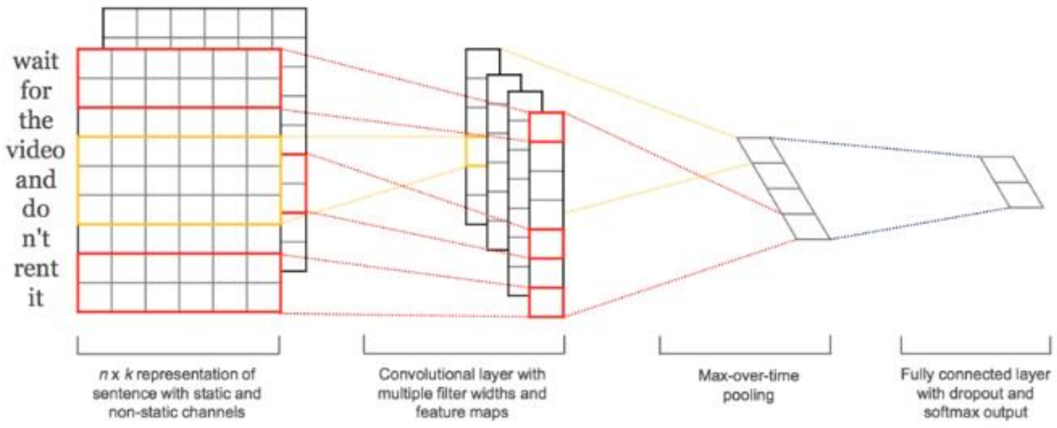
2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

2. 

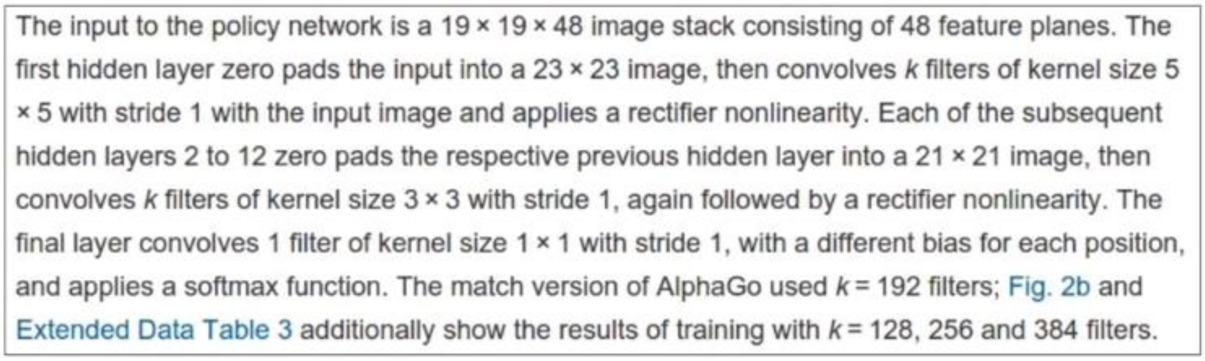
//Fast Forward

Sentence Classification. //Yoon Kim, 2014



wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

DeepMind's AlphaGo.



The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

Policy Network:

1. [19x19x45] Input
2. CONV1: 192 5x5 Filters, Stride1, Pad2, [19x19x192]
3. CONV2...12: 192 3x3 Filters, Stride1, Pad1, [19x19x192]
4. CONV: 1 1x1 Filters, Stride1 ,Pad0, [19x19]

Memo.

.constant()	//상수
.Session()	//tensor에 데이터를 넣어 흐르게 함.
.run()	//실행
.add()	//더하기
.placeholeder(), feed_dict={a:a_data}	//변수, 값을 나중에 할당.
.Variable()	//변수, 자동으로 업데이트.
.random_normal(Shapes)	//랜덤 값 반환
.reduce_mean()	//평균
.square()	//제곱
.GradientDescentOptimizer()	//미니 배치 확률적 경사하강법(SGD) 구현.
.minimize()	//최소화
.global_variables_initializer()	//.Variable()를 초기화.
.append()	//append
.plot()	//plot
.show()	//show
.reduce_sum()	//총합
.assign()	//.Variable()의 값 변경.
.compute_gradients()	//compute_gradients
.apply_gradients()	//apply_gradients
.matmul()	//matmul
.loadtext()	//text 불러오기.
.set_random_seed()	//랜덤 값 시드, 다른 환경에서도 같다.
.string_input_producer()	//Queue, text 를 Filename Queue 에 쌓기.
.TextLineReader()	//Queue, text 를 Reader 로 연결.
.read()	//Queue, text 읽기.

.decode_csv()	//Queue, text decode
.batch()	//Queue, text batch
.Coordinator()	//Queue, Coordinator 생성.
.start_queue_runners()	//Queue, Queue 를 Thread 로 시작.
.request_stop()	//Queue, 중지
.join()	//Queue, 대기
.sigmoid()	//S 자 곡선
.log()	//로그
.cast()	//새로운 자료형
.equal()	//값이 같은지
.softmax()	//softmax
.arg_max()	//arg_max
.one_hot()	//one_hot
.reshape()	//reshape
.softmax_cross_entropy_with_logits()	//softmax_cross_entropy_with_logits
.format()	//format
.flatten()	//flatten
.PrettyPrinter()	//PrettyPrinter
.InteractiveSession()	//InteractiveSession
.array()	//Array
.shape()	//Shape
.eval()	//Eval
.squeeze()	//Array 정리
.expand_dims()	//Array 정렬
.stack()	//Array 쌓기
.ones_like()	//One 으로 바꿈.

.zeros_like()	//Zero 로 바꿈.
zip()	//Zip
.nn.relu()	//Relu
.random.randn()	//지정 범위 내 랜덤 값 반환
.nn.dropout()	//Dropout

//Lab으로 연결 https://colab.research.google.com/drive/1gaTpEufmhoK2CsEsNyfDDtyynQ_HRpSu

//14폰트, 12폰트, 10폰트

//1.   1)   a.   *♠

//0.71   1.34