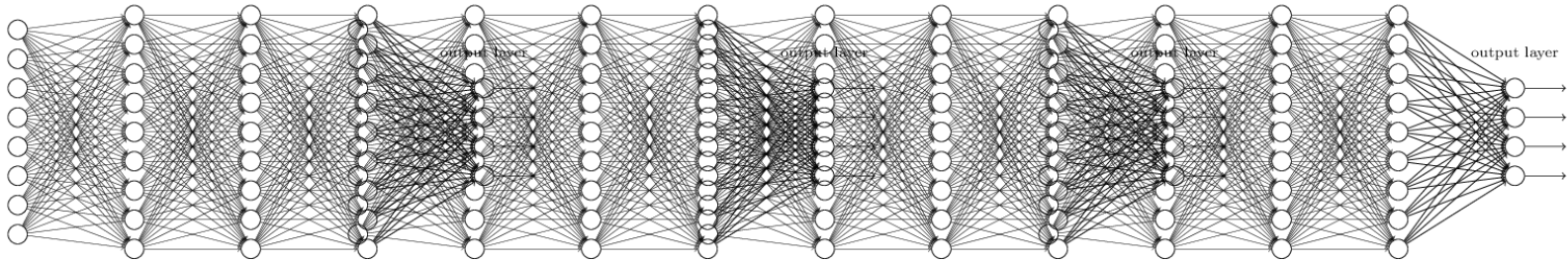




NN Backpropagation

Lecture 09

Back propagation : How to train?



Gradient descent algorithm

$$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

learning_rate

$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})x^{(i)}$$

Gradient descent algorithm : Tensorflow

$$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

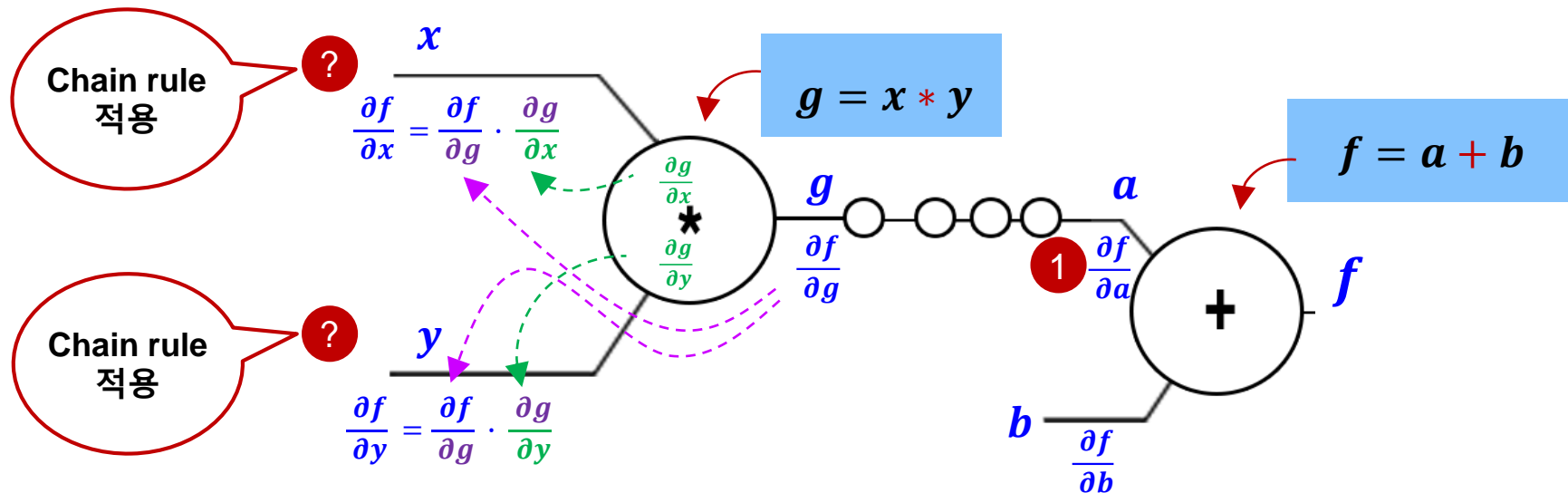
learning_rate

$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$




```
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```


Back propagation (chain rule)

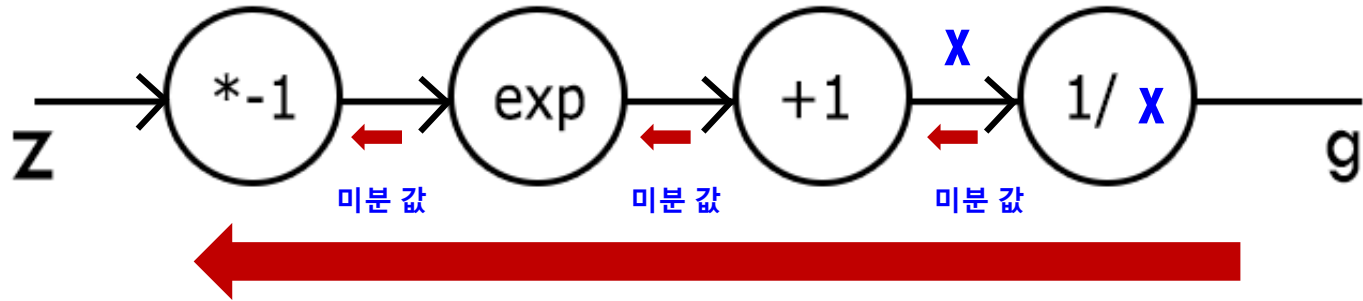


Back propagation (chain rule) : Sigmoid



































$$g(z) = \frac{1}{1 + e^{-z}}$$


? $\frac{\partial g}{\partial z}$

x




Animal classification

Birds	Insect	Fishes	Amphibians	Reptiles	Mammals
					
					
					
					
					
					
					
				Kayla	

x_data

0~6

1	0	0	1	0	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	0	2	1	0	0	1
1	0	0	1	0	0	0	0	1	1	1	0	0	4	1	0	1	0

Predicting animal type based on various features

```
xy = np.loadtxt('g:\\data-04-zoo.csv', delimiter=',', dtype=np.float32)
```

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

```
y_data = xy[:, [-1]]
```

```

import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility

# Predicting animal type based on various features
xy = np.loadtxt('g:\\data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1] # column : 16
N = x_data.shape[0]
y_data = xy[:, [-1]] # column : 1

# y_data has labels from 0 ~ 6
print("y has one of the following values")
print(np.unique(y_data)) # [0. 1. 2. 3. 4. 5. 6.] 출력됨

# X_data.shape = (101, 16) => 101 samples, 16 features
# y_data.shape = (101, 1) => 101 samples, 1 label
print("Shape of X data: ", x_data.shape) # (101, 16) 출력됨
print("Shape of y data: ", y_data.shape) # (101, 1) 출력됨

nb_classes = 7 # 0 ~ 6

X = tf.placeholder(tf.float32, [None, 16])
y = tf.placeholder(tf.int32, [None, 1]) # 0 ~ 6

target1 = tf.one_hot(y, nb_classes) # one hot
target2 = tf.reshape(target1, [-1, nb_classes])
target3 = tf.cast(target2, tf.float32)

W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

```

Backpropagation

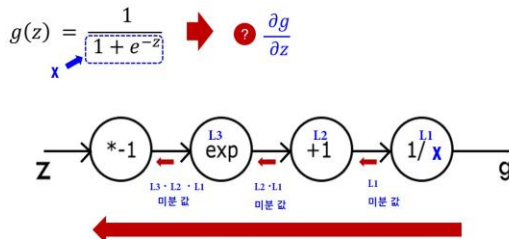
(Without using the
gradient descent
algorithm)

S

```
def sigma(x):
    # sigmoid function
    #  $\sigma(x) = 1 / (1 + \exp(-x))$ 
    return 1. / (1. + tf.exp(-x))
```

sp

```
def sigma_prime(x):
    # derivative of the sigmoid function
    #  $\sigma'(x) = \sigma(x) * (1 - \sigma(x))$ 
    return sigma(x) * (1. - sigma(x))
```



Backpropagation

(Without using the gradient descent algorithm)

Forward propagation

```
layer_1 = tf.matmul(X, W) + b
y_pred = sigma(layer_1) # g(z), sigmoid 출력
```

S

```
# Loss Function (end of forward propagation), target3 : one-hot (True value Y) , y_pred: sigmoid
loss_i = - target3 * tf.log(y_pred) - (1. - target3) * tf.log(1. - y_pred)
loss = tf.reduce_sum(loss_i)
```

Dimension Check

```
assert y_pred.shape.as_list() == target3.shape.as_list() # assert는 ==이 아니면 error 처리함
```

Back prop (chain rule)

How to derive? please read "Neural Net Backprop in one slide!"

```
d_loss = (y_pred - target3) / (y_pred * (1. - y_pred) + 1e-7)
```

```
d_sigma = sigma_prime(layer_1)
```

```
d_layer = d_loss * d_sigma
```

```
d_b = d_layer # backpropagation 의 b 값 계산
```

```
d_W = tf.matmul(tf.transpose(X), d_layer) # transpose(x)는 x를 전치함
# backpropagation 의 W 값 계산
```

Updating network using gradients (backpropagation 의 b, W 값 할당)

```
learning_rate = 0.01
```

```
train_step = [
```

```
    tf.assign(W, W - learning_rate * d_W), # W를 W - learning_rate * d_W으로 할당한다.
```

```
    tf.assign(b, b - learning_rate * tf.reduce_sum(d_b)), # b를 b-learning_rate * tf.reduce_sum(d_b)으로 할당한다.
```

```
]
```

Prediction and Accuracy

```
prediction = tf.argmax(y_pred, 1)
```

```
acct_mat = tf.equal(tf.argmax(y_pred, 1), tf.argmax(target3, 1))
```

```
acct_res = tf.reduce_mean(tf.cast(acct_mat, tf.float32))
```

Backpropagation

(Without using the
gradient descent
algorithm)

```
# Launch graph
```

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

```
for step in range(500):
    # Back prop (chain rule)를 적용한 w, b를 할당하기 위해서
    sess.run(train_step, feed_dict={X: x_data, y: y_data})

    if step % 10 == 0:
        # Within 300 steps, you should see an accuracy of 100%
        step_loss, acc = sess.run([loss, acct_res], feed_dict={X: x_data, y: y_data})
        print("Step: {:5}\t Loss: {:.10.5f}\t Acc: {:.2%}" .format(step, step_loss, acc))
```

```
# Let's see if we can predict
```

```
pred = sess.run(prediction, feed_dict={X: x_data})
line = 0
for p, y in zip(pred, y_data):
    line = line + 1
    msg = "[{}]\t Prediction: {:.d}\t True y: {:.d}"
    print("Line : " ,line, "-->",msg.format(p == int(y[0]), p, int(y[0])))
```

Backpropagation

(Without using the
gradient descent
algorithm)

```

# Predicting animal type based on various features
xy = np.loadtxt('g:\\data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1] # column : 16
N = x_data.shape[0] # [0]열의 행의 개수 101 이 출력됨 → N 은 101 이다.
Y_data = xy[:, [-1]] # column : 1

# y_data has labels from 0 ~ 6
Print("y has one of the following values ")
Print(np.unique(y_data)) # [0. 1. 2. 3. 4. 5. 6.] 출력됨

# X_data.shape = (101, 16) => 101 samples, 16 features
# y_data.shape = (101, 1) => 101 samples, 1 label
print("Shape of X data: ", x_data.shape) # (101, 16) 출력됨
print("Shape of y data: ", y_data.shape) # (101, 1) 출력됨

```



```

y has one of the following values
[0. 1. 2. 3. 4. 5. 6.]
Shape of X data:  (101, 16)
Shape of y data:  (101, 1)

```

```
target1 = tf.one_hot(y, nb_classes) # ex) [[1. 0. 0. 0. 0. 0. 0.]] 이렇게 one hot 형태로 101개 출력
target2 = tf.reshape(target1, [-1, nb_classes])
target3 = tf.cast(target2, tf.float32)
```

ex09_1 (propagation).ipynb

(Notes)

Launch graph

with tf.Session() as sess:

sess.run(tf.global_variables_initializer())

print("target1 :", sess.run(target1, feed_dict={X: x_data, y: y_data}))

print("target2 :", sess.run(target2, feed_dict={X: x_data, y: y_data}))

print("target3 :", sess.run(target3, feed_dict={X: x_data, y: y_data}))

추가해서
확인 !!



```
target1 = tf.one_hot(y, nb_classes)
```

```
target1 : [[[1. 0. 0. 0. 0. 0. 0.]]
            [[1. 0. 0. 0. 0. 0. 0.]]
            [[0. 0. 0. 1. 0. 0. 0.]]
            [[1. 0. 0. 0. 0. 0. 0.]]
            [[1. 0. 0. 0. 0. 0. 0.]]
            [[1. 0. 0. 0. 0. 0. 0.]]
```

:

```
target2 = tf.reshape(target1, [-1, nb_classes])
```

```
target2 : [[1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [0. 0. 0. 1. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [0. 0. 0. 1. 0. 0. 0.]
```

:

```
target3 = tf.cast(target2, tf.float32)
```

```
target3 : [[1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [0. 0. 0. 1. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
            [1. 0. 0. 0. 0. 0. 0.]
```

:

```
target1 = tf.one_hot(y, nb_classes) # ex) [[1. 0. 0. 0. 0. 0. 0.]] 이렇게 one hot 형태로 101개 출력
target2 = tf.reshape(target1, [-1, nb_classes])
target3 = tf.cast(target2, tf.float32)
```

Forward propagation

```
layer_1 = tf.matmul(X, W) + b
y_pred = sigma(layer_1) # Layer_1 각 행에서 표준편차
```

Loss Function (end of forward propagation)

```
loss_i = - target3 * tf.log(y_pred) - (1. - target3) * tf.log(1. - y_pred)
loss = tf.reduce_sum(loss_i)
```

Dimension Check

```
assert y_pred.shape.as_list() == target3.shape.as_list()
```

Prediction and Accuracy

```
prediction = tf.argmax(y_pred, 1)
acct_mat = tf.equal(tf.argmax(y_pred, 1), tf.argmax(target3, 1))
acct_res = tf.reduce_mean(tf.cast(acct_mat, tf.float32))
```

Launch graph

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

```
print("• layer_1 \n",sess.run(layer_1, feed_dict={X: x_data, y: y_data}))
print("• y_pred \n",sess.run(y_pred, feed_dict={X: x_data, y: y_data}))
```

• layer_1

```
[[ -1.03140700e+00 -7.79828429e-02  3.71772528e+00  6.74435520e+00
   -8.10283566e+00 -1.14475656e+00  8.27225494e+00]
 [ -3.32648945e+00  6.78884089e-01  3.46578169e+00  7.85745239e+00
   -8.47612953e+00 -3.18839216e+00  9.72614765e+00]
 [  2.57165432e+00  3.03211141e+00  4.04691076e+00  3.67466688e+00
   -7.77052164e-01 -4.31904793e-02  3.20639515e+00]
```

•
•
•

• y_pred

```
[[ 2.62811422e-01  4.80514139e-01  9.76286829e-01  9.98823822e-01
   3.02588043e-04  2.41448119e-01  9.99744594e-01]
 [ 3.46735418e-02  6.63489640e-01  9.69698310e-01  9.99613345e-01
   2.08340265e-04  3.96048911e-02  9.99940276e-01]
 [ 9.29014921e-01  9.54003930e-01  9.82823849e-01  9.75269258e-01
   3.14955562e-01  4.89204049e-01  9.61074293e-01]
```

•
•
•

추가해서
확인 !!



```
target1 = tf.one_hot(y, nb_classes) # ex) [[1. 0. 0. 0. 0. 0.]] 이렇게 one hot 형태로 101개 출력
target2 = tf.reshape(target1, [-1, nb_classes])
target3 = tf.cast(target2, tf.float32)
```

ex09_1 (propagation).ipynb

(Notes)

```
# Forward propagation
```

```
layer_1 = tf.matmul(X, W) + b
```

```
y_pred = sigma(layer_1) # layer_1 각 행에서 표준편차
```

$$C(H(x), y) = -y \log(H(x)) - (1 - y) \log(1 - H(x))$$

```
# Loss Function (end of forward propagation)
```

```
loss_i = - target3 * tf.log(y_pred) - (1. - target3) * tf.log(1. - y_pred)
```

```
loss = tf.reduce_sum(loss_i)
```

```
# Dimension Check
```

```
assert y_pred.shape.as_list() == target3.shape.as_list()
```

```
# Prediction and Accuracy
```

```
prediction = tf.argmax(y_pred, 1)
```

```
acct_mat = tf.equal(tf.argmax(y_pred, 1), tf.argmax(target3, 1))
```

```
acct_res = tf.reduce_mean(tf.cast(acct_mat, tf.float32))
```

```
# Launch graph
```

```
with tf.Session() as sess:
```

```
    sess.run(tf.global_variables_initializer())
```

```
    for step in range(500):
```

```
        sess.run(train_step, feed_dict={X: x_data, y: y_data})
```

```
        if step % 10 == 0:
```

```
            # Within 300 steps, you should see an accuracy of 100%
```

```
            step_loss, acc = sess.run([loss, acct_res], feed_dict={X: x_data, y: y_data})
```

```
            print("Step: {:5}\t Loss: {:.105f}\t Acc: {:.2%}" .format(step, step_loss, acc))
```

Step:	0	Loss:	560.45282	Acc:	36.63%
Step:	10	Loss:	120.88565	Acc:	81.19%
Step:	20	Loss:	86.91508	Acc:	83.17%
Step:	30	Loss:	71.21052	Acc:	89.11%
Step:	40	Loss:	61.65193	Acc:	89.11%
Step:	50	Loss:	54.69436	Acc:	93.07%
			.		
Step:	450	Loss:	12.54190	Acc:	100.00%
Step:	460	Loss:	12.33679	Acc:	100.00%
Step:	470	Loss:	12.13882	Acc:	100.00%
Step:	480	Loss:	11.94763	Acc:	100.00%
Step:	490	Loss:	11.76283	Acc:	100.00%

```
target1 = tf.one_hot(y, nb_classes) # ex) [[1. 0. 0. 0. 0. 0. 0.]] 이렇게 one hot 형태로 101개 출력
target2 = tf.reshape(target1, [-1, nb_classes])
target3 = tf.cast(target2, tf.float32)
```

ex09_1 (propagation).ipynb

(Notes)

```
# Forward propagation
layer_1 = tf.matmul(X, W) + b
y_pred = sigma(layer_1) # layer_1 각 행에서 표준편차

# Loss Function (end of forward propagation)
loss_i = - target3 * tf.log(y_pred) - (1. - target3) * tf.log(1. - y_pred)
loss = tf.reduce_sum(loss_i)

# Dimension Check
assert y_pred.shape.as_list() == target3.shape.as_list()
```

```
# Prediction and Accuracy
prediction = tf.argmax(y_pred, 1)
```

```
acct_mat = tf.equal(tf.argmax(y_pred, 1), tf.argmax(target3, 1))
acct_res = tf.reduce_mean(tf.cast(acct_mat, tf.float32))
```

```
# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(500):
        sess.run(train_step, feed_dict={X: x_data, y: y_data})

        if step % 10 == 0:
            # Within 300 steps, you should see an accuracy of 100%
            step_loss, acc = sess.run([loss, acct_res], feed_dict={X: x_data, y: y_data})
            print("Step: {:5}\t Loss: {:.105f}\t Acc: {:.2%}" .format(step, step_loss, acc))
```

Step:	0	Loss:	560.45282	Acc:	36.63%
Step:	10	Loss:	120.88565	Acc:	81.19%
Step:	20	Loss:	86.91508	Acc:	83.17%
Step:	30	Loss:	71.21052	Acc:	89.11%
Step:	40	Loss:	61.65193	Acc:	89.11%
Step:	50	Loss:	54.69436	Acc:	93.07%
.					
.					
Step:	450	Loss:	12.54190	Acc:	100.00%
Step:	460	Loss:	12.33679	Acc:	100.00%
Step:	470	Loss:	12.13882	Acc:	100.00%
Step:	480	Loss:	11.94763	Acc:	100.00%
Step:	490	Loss:	11.76283	Acc:	100.00%

Prediction and Accuracy

```
prediction = tf.argmax(y_pred, 1) # 예측한 y_pred에서 제일 큰 값이 있는 열 위치
```

Launch graph

```
with tf.Session() as sess:
```

```
    sess.run(tf.global_variables_initializer())
```

```
    for step in range(500):
```

```
        sess.run(train_step, feed_dict={X: x_data, y: y_data})
```

```
        if step % 10 == 0:
```

```
            # Within 300 steps, you should see an accuracy of 100%
```

```
            step_loss, acc = sess.run([loss, acct_res], feed_dict={X: x_data, y: y_data})
```

```
            print("Step: {:5}\t Loss: {:.105f}\t Acc: {:.2%}" .format(step, step_loss, acc))
```

Let's see if we can predict

```
    # 예측한 y_pred에서 제일 큰 값이 있는 열 위치
```

```
    pred = sess.run(prediction, feed_dict={X: x_data})
```

```
    line = 0
```

```
    for p, y in zip(pred, y_data):
```

```
        line = line + 1
```

```
        msg = "[{}]\t Prediction: {:d}\t True y: {:d} "
```

```
        print("Line : " , line, "-->", msg.format(p == int(y[0]), p, int(y[0])))
```

```
        # p == int(y[0]) : 예측한 y_pred 에서 제일 큰 값이 있는 열 위치와 True value (y[0]) 이 같은 값인지 비교하여 같으면 True 출력
```

```
        # p : 예측한 y_pred 에서 제일 큰 값이 있는 열 위치 출력
```

```
        # int(y[0]) : True value (y[0]) 출력
```

ex09_1 (propagation).ipynb

(Notes)

```
Line : 1 --> [True] Prediction: 0 True y: 0
Line : 2 --> [True] Prediction: 0 True y: 0
Line : 3 --> [True] Prediction: 3 True y: 3
Line : 4 --> [True] Prediction: 0 True y: 0
Line : 5 --> [True] Prediction: 0 True y: 0
Line : 6 --> [True] Prediction: 0 True y: 0
Line : 7 --> [True] Prediction: 0 True y: 0
.
.
.
Line : 97 --> [True] Prediction: 0 True y: 0
Line : 98 --> [True] Prediction: 5 True y: 5
Line : 99 --> [True] Prediction: 0 True y: 0
Line : 100 --> [True] Prediction: 6 True y: 6
Line : 101 --> [True] Prediction: 1 True y: 1
```