Softmax Classifier

Lecture 06

Softmax function

Logistic Classifier

$$WX = \hat{Y}$$

$$y \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

SOFTMAX

$$S(yi) = \frac{e^{yi}}{\sum_i e^{yi}}$$

$p = 0.7$ A

$p = 0.2$ B

$p = 0.1$ C

1.0

PROBABILITIES

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

$$WX = \hat{Y}$$

$$y \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

$\hat{Y}$

logits

SOFTMAX

$$S(yi) = \frac{e^{yi}}{\sum_i e^{yi}}$$

$p = 0.7$

$p = 0.2$

$p = 0.1$

PROBABILITIES

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

Cost function: cross entropy

Cost (Loss)

$$L = \frac{1}{N} \sum_i D(S(WX_i + b), L_i)$$

TRAINING SET

$$D(S, L) = -\sum_i Li \; log(Si)$$

hypothesis

```
# Cross entropy cost/loss
# reduce_sum( , axis=1 ) axis=1 -> row에서 합계를 계산, axis=0 -> column에서 합계를 계산
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

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

ex06_1.ipynb (1/3)

```
import tensorflow as tf
x_data = [[1, 2, 1, 1], [2, 1, 3, 2], [3, 1, 3, 4], [4, 1, 5, 5], [1, 7, 5, 5], [1, 2, 5, 6], [1, 6, 6, 6], [1, 7, 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]]  # 0,1,2
                2         2         2         1         1         1         0         0

X = tf.placeholder("float", [None, 4])  # row : N , column : 4
Y = tf.placeholder("float", [None, 3])  # row : N , column : 3
nb_classes = 3 # column of Y

W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight') # row : 4 , column : 3
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss  (reduce_sum( , axis=1 ) axis=1 -> row에서 합계를 계산, axis=0 -> column에서 합계를 계산)
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

# optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
# train = optimizer.minimize(cost)  -> 실행 : sess.run(train)으로 실행해야 함
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

nb_classes

'ONE-HOT' ENCODING

$$\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$$D(S, L) = -\sum_i Li \; log(Si)$$

```
# Launch graph
with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())
   for i in range (len(x_data)):
        print (i+1,"번째 x_data", x_data[i], "==> y_data", y_data[i], end="")
        if (y_data[i] == [1, 0, 0]):
            print(" : 0")
        elif (y_data[i] == [0, 1, 0]):
            print(" : 1")
        elif (y_data[i] == [0, 0, 1]):
            print(" : 2")
```

```
1 번째 x_data [1, 2, 1, 1] ==> y_data [0, 0, 1] : 2
2 번째 x_data [2, 1, 3, 2] ==> y_data [0, 0, 1] : 2
3 번째 x_data [3, 1, 3, 4] ==> y_data [0, 0, 1] : 2
4 번째 x_data [4, 1, 5, 5] ==> y_data [0, 1, 0] : 1
5 번째 x_data [1, 7, 5, 5] ==> y_data [0, 1, 0] : 1
6 번째 x_data [1, 2, 5, 6] ==> y_data [0, 1, 0] : 1
7 번째 x_data [1, 6, 6, 6] ==> y_data [1, 0, 0] : 0
8 번째 x_data [1, 7, 7, 7] ==> y_data [1, 0, 0] : 0
```

```
for step in range(2001):
    sess.run(optimizer, feed_dict={X: x_data, Y: y_data}) # 실행 : sess.run(optimizer) 으로 실행해야 함
    if step % 200 == 0:
        cost_v, W_v = sess.run([cost,W],feed_dict={X: x_data, Y: y_data})
        print('='*20, step, "번째 Learning", '='*20)
        print("cost : ", cost_v, "\nW : ", W_v)
```

Weight → row : 4 , column : 3

```
========== 0 번째 Learning ==========
cost :  3.6451883
W :  [[ 0.7168987  -0.23286213  0.28869355]
 [ 1.4674631   1.4823829   1.052482  ]
 [-1.6186061   0.01691242 -0.8614841 ]
 [-1.0679966  -0.4912663   0.04156154]]
========== 200 번째 Learning ==========
cost :  0.596512
W :  [[-1.0872966   0.28811795  1.5719084 ]
 [ 1.3888952   1.0571648   1.5562694 ]
 [-0.49595264 -0.51861405 -1.4486113 ]
 [-0.44033748 -0.11886172 -0.95850164]]
========== 400 번째 Learning ==========
cost :  0.48940927
W :  [[-1.6811438   0.43864465  2.0152295 ]
 [ 1.2551967   1.0893495   1.6577837 ]
 [ 0.06578756 -0.7553408  -1.7736238 ]
 [-0.6009978   0.14388406 -1.0605853 ]]
.
.
.
```

```
.
.
.
========== 1800 번째 Learning ==========
cost :  0.1746932
W :  [[-3.7331817   0.9934499   3.5124629 ]
 [ 0.81080514  1.1602981   2.0312307 ]
 [ 2.6173372  -1.5376157  -3.542897  ]
 [-1.6726953   0.9069222  -0.7519227 ]]
========== 2000 번째 Learning ==========
cost :  0.16201103
W :  [[-3.9105399  1.0405984  3.6426697]
 [ 0.7697886  1.1618105  2.070735 ]
 [ 2.857559  -1.6031579 -3.7175748]
 [-1.7902993  0.9852507 -0.7126456]]
```

ex06_1.ipynb (3/3)

```
# Testing & One-hot encoding
# argmax (a, 0)에서 0의 의미는 같은 열에서 max value 인덱스를 출력 (인덱스는 0 부터 시작)
# argmax (a, 1)에서 1의 의미는 같은 행에서 max value 인덱스를 출력 (인덱스는 0 부터 시작)
print("\n● Testing & One-hot encoding & argmax")

a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
print("Test data X : [1, 11, 7, 9] --> ", end = "")
print("hypothesis :",a, "-->", sess.run(tf.argmax(a, 1)))
print('='*100)

b = sess.run(hypothesis, feed_dict={X: [[1, 3, 4, 3]]})
print("Test data X : 1, 3, 4, 3] --> ", end = "")
print("hypothesis :",b, "-->", sess.run(tf.argmax(b, 1)))
print('='*100)

c = sess.run(hypothesis, feed_dict={X: [[1, 1, 0, 1]]})
print("Test data X : [1, 1, 0, 1] --> ", end = "")
print("hypothesis :",c, "-->", sess.run(tf.argmax(c, 1)))
print('='*100)

all = sess.run(hypothesis, feed_dict={ X: [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]]})
print("Test data X : [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]] ")
print("hypothesis \n",all, "-->", sess.run(tf.argmax(all, 1)))
print('='*100)
```

```
● Testing & One-hot encoding & argmax
Test data X : [1, 11, 7, 9] --> hypothesis : [[8.581670e-03 9.914078e-01 1.050219e-05]] --> [1]
====================================================================================================
Test data X : 1, 3, 4, 3] --> hypothesis : [[0.8252531  0.15725392 0.01749295]] --> [0]
====================================================================================================
Test data X : [1, 1, 0, 1] --> hypothesis : [[2.0768354e-08 3.9369191e-04 9.9960631e-01]] --> [2]
====================================================================================================
Test data X : [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]]
hypothesis
 [[8.5816700e-03 9.9140781e-01 1.0502190e-05]
 [8.2525313e-01 1.5725392e-01 1.7492952e-02]
 [2.0768352e-08 3.9369191e-04 9.9960631e-01]] --> [1 0 2]
====================================================================================================
```

ex06_1.ipynb Notes

Test & one-hot encoding

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

```
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
print("Test data X : [1, 11, 7, 9] --> ", end = "")
print("hypothesis :",a, "-->", sess.run(tf.argmax(a, 1)))
print('='*100)
```

```
Test data X : [1, 11, 7, 9] --> hypothesis : [[8.581670e-03 9.914078e-01 1.050219e-05]] --> [1]
====================================================================================================
```

$S(y)$ → softmax

8.581670e-03

9.914078e-01

1.050219e-05

→ ~~a~~ b ~~c~~

x_data
row : N
column : 4

1	2	1	1
2	1	3	2
3	1	3	4
4	1	5	5

x

W
row : 4
column : 3

=

y_data
row : N
column : 3

0	0	1
0	0	1
0	0	1
0	1	0

Test & one-hot encoding

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

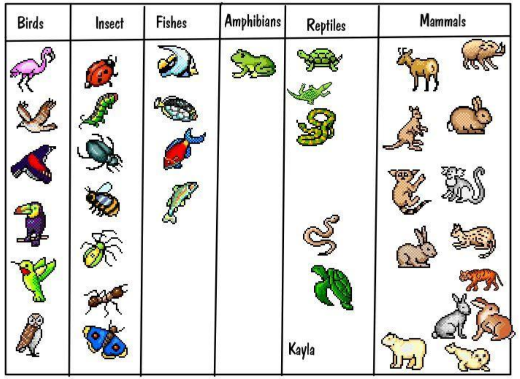
```
all = sess.run(hypothesis, feed_dict={ X: [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]]})
print("Test data X : [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]] ")
print("hypothesis \n",all, "-->", sess.run(tf.argmax(all, 1)))
print('='*100)
```

```
Test data X : [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]]
hypothesis
 [[8.5816700e-03 9.9140781e-01 1.0502190e-05]
 [8.2525313e-01 1.5725392e-01 1.7492952e-02]
 [2.0768352e-08 3.9369191e-04 9.9960631e-01]] --> [1 0 2]
====================================================================================================
```

Fancy softmax Classifier

- one_hot
- reshape
- cross_entropy

Animal classification



x_data

y_data
0~6

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

```python
# 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]]
```

tf.one_hot and reshape

y_data

0~6

x_data

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

tf.one_hot :

If the input indices is rank **N**,

the output will have rank **N+1**.

ex) input : [[0] [3]]

→ output : [[[1000000]] [[0001000]]]

```
nb_classes = 7
Y = tf.placeholder(tf.int32, [None, 1])  #  None row, 1 column (0 ~ 6) -> shape=(?, 1)
Y_one_hot = tf.one_hot(Y, nb_classes)  # Y는 0부터 시작함, one hot shape=(?, 1, 7)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes] )  # shape=(?, 7)
```

tf.one_hot and reshape

x_data

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1

y_data

0~6

0
3
0
0
0
0
3
3
0
0
1
3
6
6
6
1
0

```
nb_classes = 7
Y = tf.placeholder(tf.int32, [None, 1])  # 0~6 사이의 값을 넣기 위해 shape=(?, 1)설정
Y_one_hot = tf.one_hot(Y, nb_classes)  # one hot shape=(?, 1, 7)
                      0~6     7
```

Y_one_hot	Y
1000000	0
0100000	1
0010000	2
0001000	3
0000100	4
0000010	5
0000001	6

0 3 Y_one_hot

[[[1000000] [0001000] ….]]

```
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes]) # shape=(?, 7)
                                       7
```

[[1000000] [0001000] ….]

tf.one_hot and reshape

```
import numpy as np
x = np.arange(12)
print(x)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
x = x.reshape(3, 4)
x.shape
```

```
(3, 4)
```

```
print(x)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
x.reshape(-1, 1)
```

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

```
x.reshape(-1, 2)
```

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

```
x.reshape(-1, 3)
```

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

softmax_cross_entropy_with_logits

SCORE

```
logit = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logit)
```

1

one-hot

```
# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

=

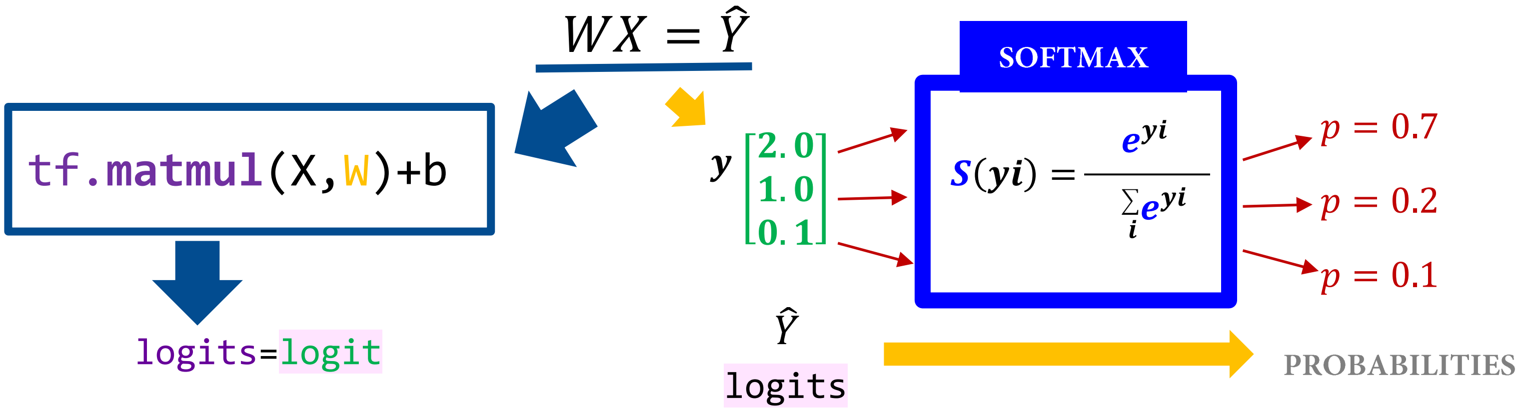
2

```
# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logit,
                                                 labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
```

Y의 label

softmax_cross_entropy_with_logits

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```



SCORE

```
logit = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logit)
```

ex06_2.ipynb (1/2)

```
import tensorflow as tf
import numpy as np
# Predicting animal type based on various features
xy = np.loadtxt('g:\\data-04-zoo.csv', delimiter=',', dtype=np.float32)  # column : 17
x_data = xy[:, 0:-1] # column : 16
y_data = xy[:, [-1]] # column : 1

nb_classes = 7  # Y의 종류가 총 7개로 구성되어 있음

X = tf.placeholder(tf.float32, [None, 16]) #  row : None , column : 16
Y = tf.placeholder(tf.int32, [None, 1])  # row : None , column : 1 ( 0, 1, 2, 3, 4, 5, 6 (총 7개 종류로 구성))

Y_one_hot = tf.one_hot(Y, nb_classes)  # one hot
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes]) # ex) 1000000… structure

W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight') #  row : 16 , column : 7
b = tf.Variable(tf.random_normal([nb_classes]), name='bias') #  column : 7

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
logit = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logit)

# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logit, labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)

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

# tf.argmax(A , flag) : matrix A에서 flag가 0이면 column 기준, 1이면 row 기준으로 큰 값의 index를 반환함 (index는 0부터 시작..)
prediction = tf.argmax(hypothesis, 1) # flag가 1이므로 row 기준으로 큰 값의 index를 반환함
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1)) # flag가 1이므로 row 기준으로 큰 값의 index를 반환함
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

tf.argmax
큰 값이 있는 곳의 index를 반환함

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

    # Learning
    for step in range(2001):
        sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            loss, acc = sess.run([cost, accuracy], feed_dict={X: x_data, Y: y_data})
            print("Step: {:5}\tCost: {:.3f}\tAccuracy: {:.2%}".format(step, loss, acc) )

    # Let's see if we can predict
    pred = sess.run(prediction, feed_dict={X: x_data})
    line = len(x_data)
    i = 0
    print ("\n","="*30, line, "개  Data analysis ", "="*20)
    for p, y in zip(pred, y_data.flatten()): # y_data.flatten() 다차원 배열을 1차원 배열로 변형시킴
        i = i+1
        print(i, "번째 : Is the prediction equal to the true value? => [{}] : Prediction: {} True Y: {}".format(p == int(y), p, int(y)))

    # Testing & One-hot encoding
    test_x = [[0,0,1,0,0,1,0,1,1,0,0,1,0,1,0,0]]
    print("\n• Testing & One-hot encoding & argmax")
    print("Test data X :", test_x )
    print("prediction :", sess.run(prediction, feed_dict={X: test_x}))
```

```python
# Learning
    for step in range(2001):
        sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            loss, acc = sess.run([cost, accuracy], feed_dict={X: x_data, Y: y_data})
            print("Step: {:5}\tCost: {:.3f}\tAccuracy: {:.2%}".format(step, loss, acc) )
```

```
Step:     0	Cost: 4.624	Accuracy: 9.90%
Step:   100	Cost: 0.680	Accuracy: 78.22%
Step:   200	Cost: 0.433	Accuracy: 85.15%
Step:   300	Cost: 0.313	Accuracy: 87.13%
Step:   400	Cost: 0.237	Accuracy: 91.09%
Step:   500	Cost: 0.187	Accuracy: 97.03%
Step:   600	Cost: 0.153	Accuracy: 99.01%
Step:   700	Cost: 0.129	Accuracy: 100.00%
Step:   800	Cost: 0.112	Accuracy: 100.00%
Step:   900	Cost: 0.099	Accuracy: 100.00%
Step:  1000	Cost: 0.089	Accuracy: 100.00%
Step:  1100	Cost: 0.080	Accuracy: 100.00%
Step:  1200	Cost: 0.074	Accuracy: 100.00%
Step:  1300	Cost: 0.068	Accuracy: 100.00%
Step:  1400	Cost: 0.063	Accuracy: 100.00%
Step:  1500	Cost: 0.059	Accuracy: 100.00%
Step:  1600	Cost: 0.055	Accuracy: 100.00%
Step:  1700	Cost: 0.052	Accuracy: 100.00%
Step:  1800	Cost: 0.050	Accuracy: 100.00%
Step:  1900	Cost: 0.047	Accuracy: 100.00%
Step:  2000	Cost: 0.045	Accuracy: 100.00%
```

```python
# Let's see if we can predict
    pred = sess.run(prediction, feed_dict={X: x_data})
    line = len(x_data)
    i = 0
    print ("\n","="*30, line, "개  Data analysis ", "="*20)
    for p, y in zip(pred, y_data.flatten()): # y_data.flatten() 다차원 배열을 1차원 배열로 변형시킴
        i = i+1
        print(i, "번째 : Is the prediction equal to the true value? => [{}] : Prediction: {} True Y: {}".format(p == int(y), p, int(y)))
```

```
 ============================== 101 개  Data analysis  ====================
1 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
2 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
3 번째 : Is the prediction equal to the true value? => [True] : Prediction: 3 True Y: 3
4 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
5 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
6 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
7 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
8 번째 : Is the prediction equal to the true value? => [True] : Prediction: 3 True Y: 3
9 번째 : Is the prediction equal to the true value? => [True] : Prediction: 3 True Y: 3
.
.
.
97 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
98 번째 : Is the prediction equal to the true value? => [True] : Prediction: 5 True Y: 5
99 번째 : Is the prediction equal to the true value? => [True] : Prediction: 0 True Y: 0
100 번째 : Is the prediction equal to the true value? => [True] : Prediction: 6 True Y: 6
101 번째 : Is the prediction equal to the true value? => [True] : Prediction: 1 True Y: 1
```

```
test_x = [[0,0,1,0,0,1,0,1,1,0,0,1,0,1,0,0]]
    print("\n● Testing & One-hot encoding & argmax")
    print("Test data X :", test_x )
    print("prediction :", sess.run(prediction, feed_dict={X: test_x}))
```

```
● Testing & One-hot encoding & argmax
Test data X : [[0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0]]
prediction : [3]
```