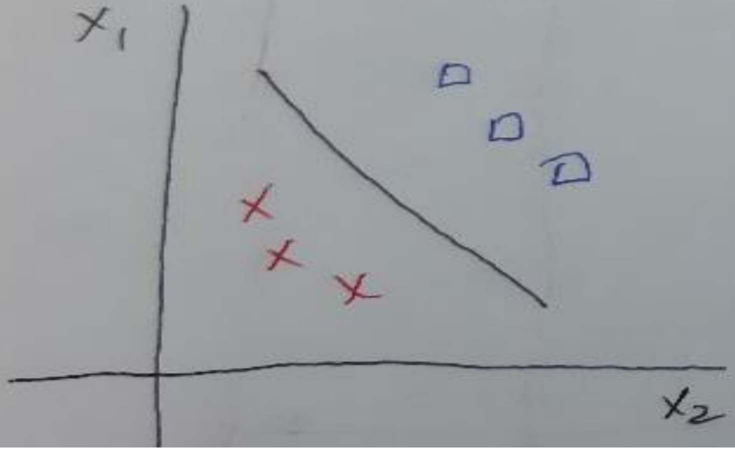
Softmax Regression
Multinomial Logistic Classification

(binary) Logistic Classification

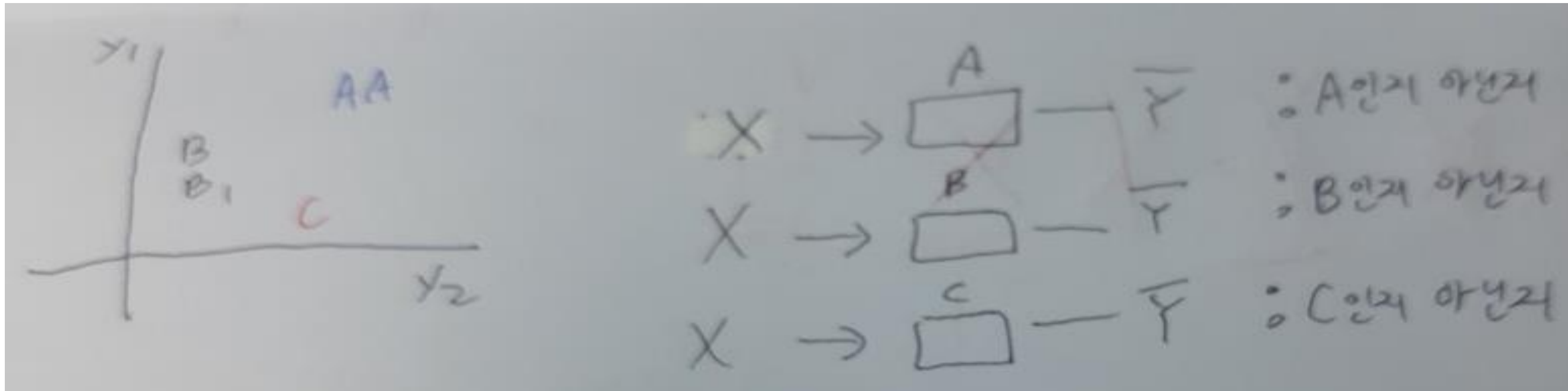
$$\begin{tikzcd} X \arrow[r] & \underset{W}{\boxed{\quad}} \arrow[r, "z"] & \underset{\text{sigmoid}}{\boxed{\quad}} \arrow[r] & \bar{Y}\begin{cases}0\\1\end{cases}\end{tikzcd}$$

$H_L(x)=WX$ $\qquad Z=H_L(x),\quad g(z)=\frac{1}{1+e^{-z}}$ $\qquad H_R(x)=g(H_L(x))$



“X와 □을 구별”

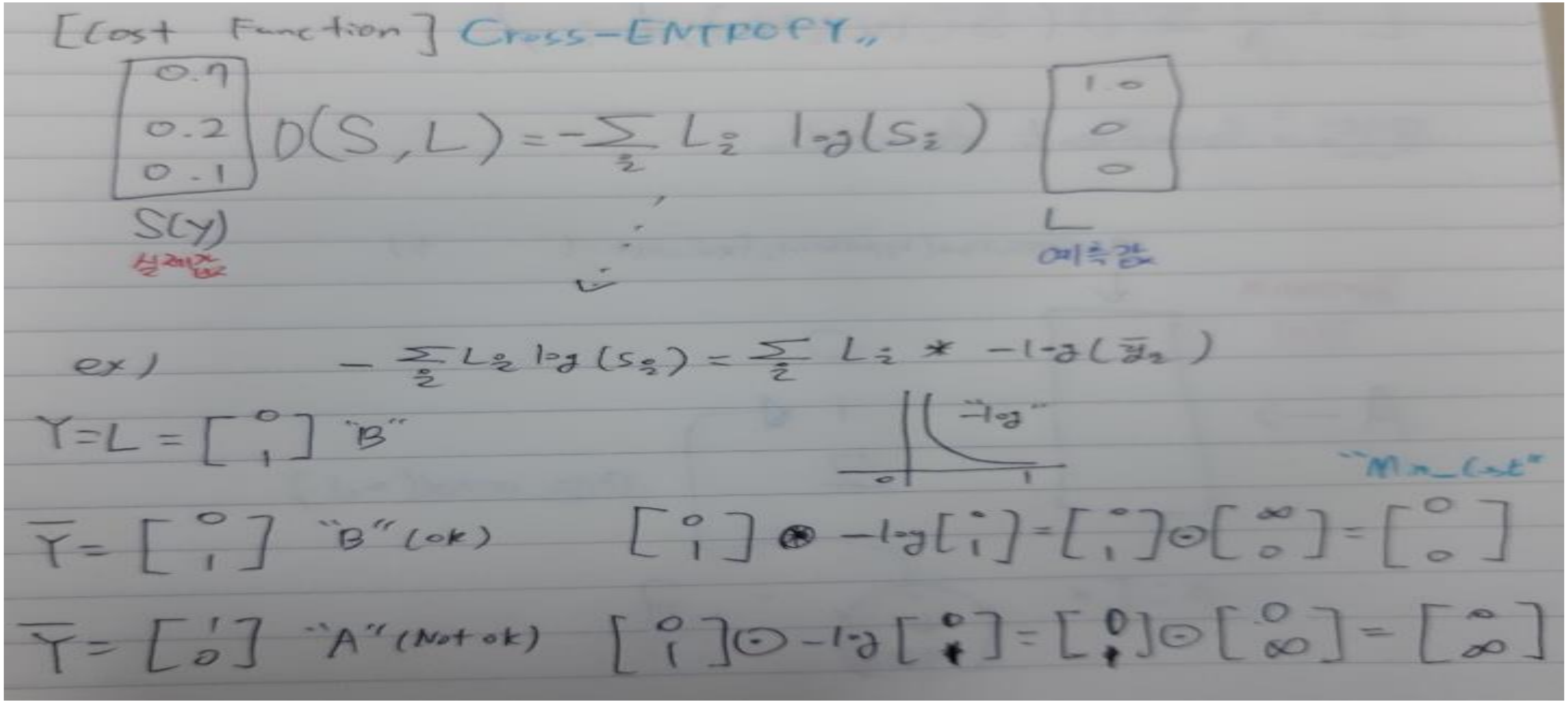
Multinomial Logistic Classification



⇓ 합체

$$\begin{bmatrix} W_{A1} & W_{A2} & W_{A3} \\ W_{B1} & W_{B2} & W_{B3} \\ W_{C1} & W_{C2} & W_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} W_{A1}X_1 + W_{A2}X_2 + W_{A3}X_3 \\ W_{B1}X_1 + W_{B2}X_2 + W_{B3}X_3 \\ W_{C1}X_1 + W_{C2}X_2 + W_{C3}X_3 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix}$$

Multinomial Logistic Classification



[Cost Function] Cross-ENTROPY,,

$$\begin{bmatrix}0.7\\0.2\\0.1\end{bmatrix} D(S,L) = -\sum_i L_i \log(S_i) \quad \begin{bmatrix}1.0\\0\\0\end{bmatrix}$$

S(y) 실제값 L 예측값

ex) $-\sum_i L_i \log(S_i) = \sum_i L_i * -\log(S_i)$

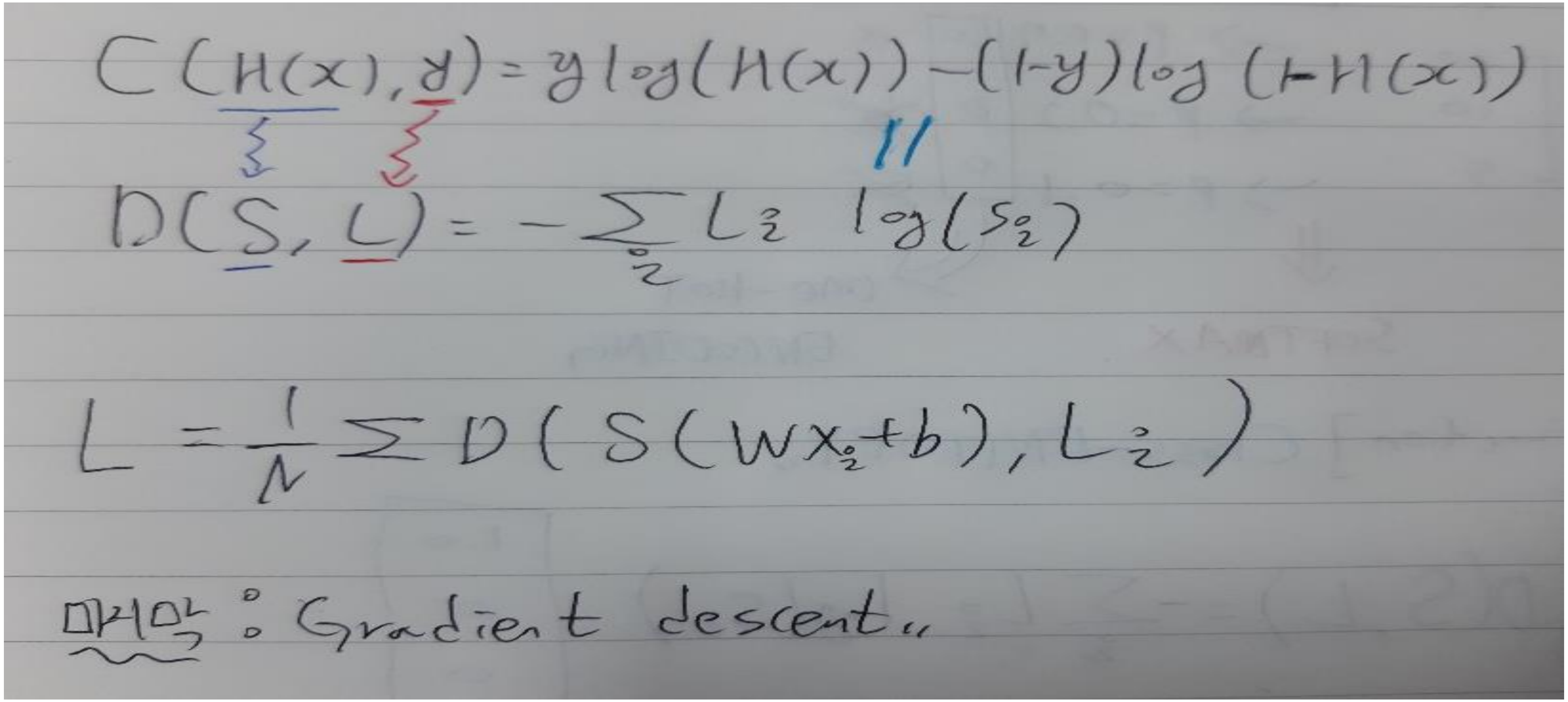
$Y = L = \begin{bmatrix}0\\1\end{bmatrix}$ "B"

"Min_Cost"

$\bar{Y} = \begin{bmatrix}0\\1\end{bmatrix}$ "B" (ok) $\begin{bmatrix}0\\1\end{bmatrix} \circledast -\log\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} \odot \begin{bmatrix}\infty\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$

$\bar{Y} = \begin{bmatrix}1\\0\end{bmatrix}$ "A" (Not ok) $\begin{bmatrix}0\\1\end{bmatrix} \odot -\log\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} \odot \begin{bmatrix}0\\\infty\end{bmatrix} = \begin{bmatrix}0\\\infty\end{bmatrix}$

Multinomial Logistic Classification



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

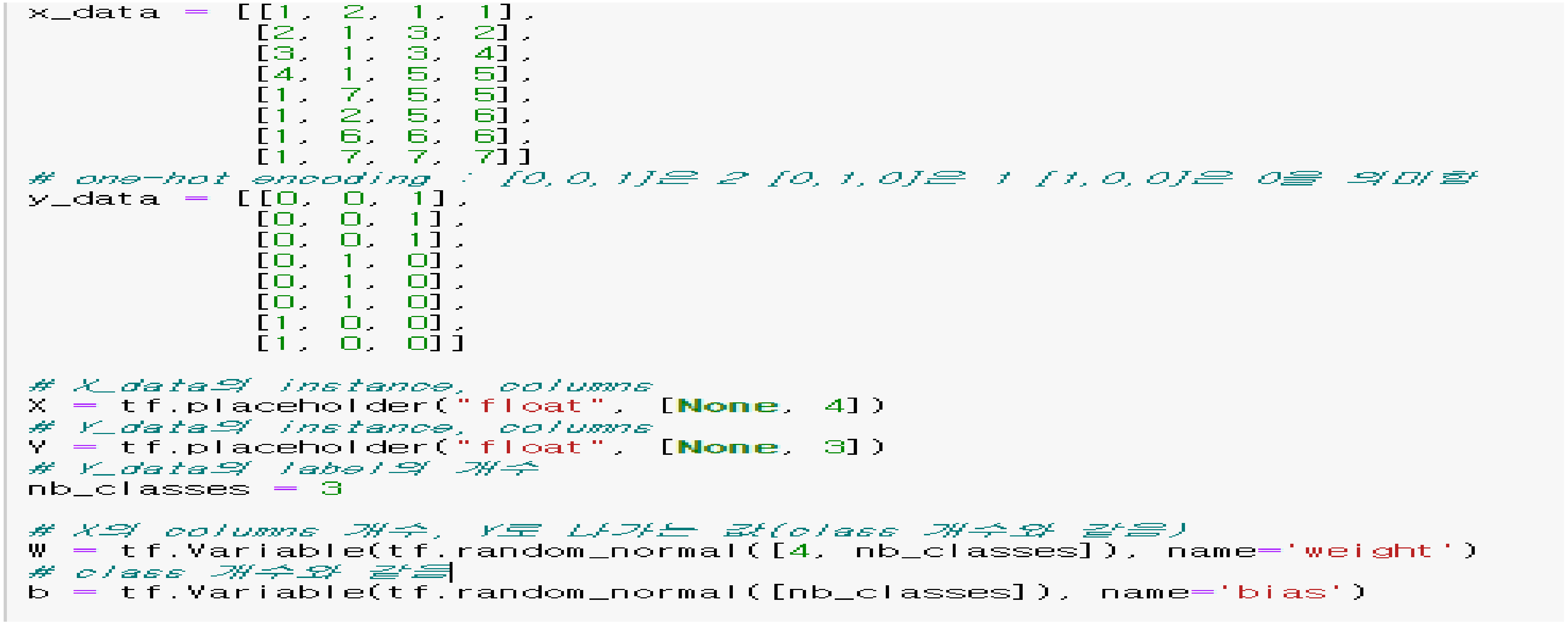
$$//$$

$$D(\underline{S}, \underline{L}) = -\sum_i L_i \log(S_i)$$

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

마지막 : Gradient descent..

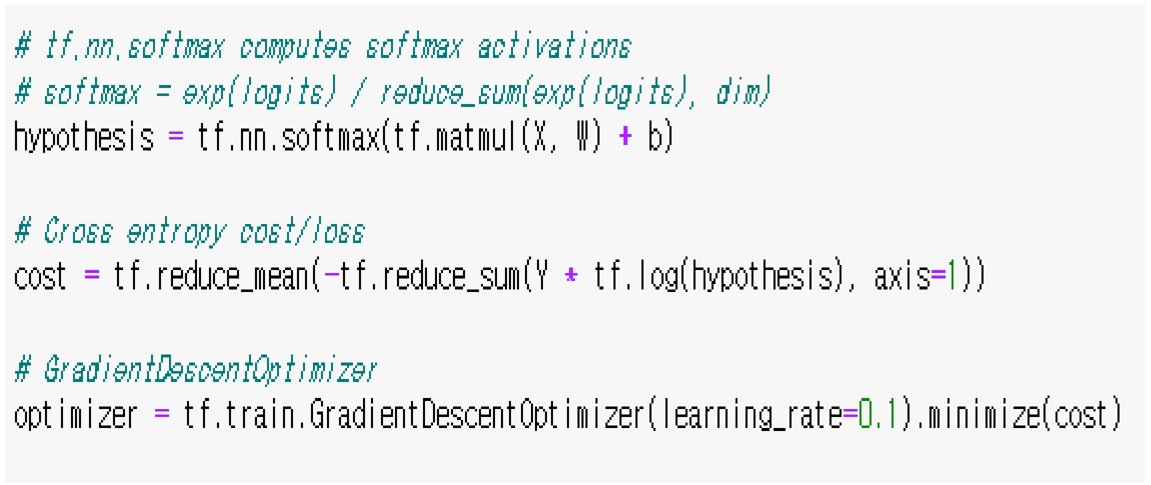
Softmax Classification 구현 TF



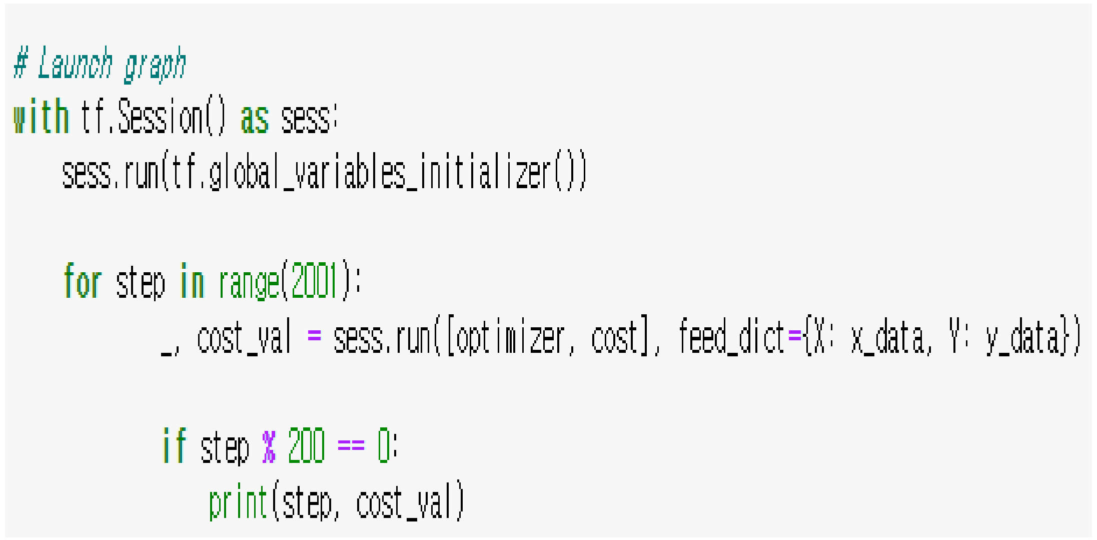
```
x_data = [[1, 2, 1, 1],
          [2, 1, 3, 2],
          [3, 1, 3, 4],
          [4, 1, 5, 5],
          [1, 7, 5, 5],
          [1, 2, 5, 6],
          [1, 6, 6, 6],
          [1, 7, 7, 7]]
# one-hot encoding : [0,0,1]은 2 [0,1,0]은 1 [1,0,0]은 0을 의미함
y_data = [[0, 0, 1],
          [0, 0, 1],
          [0, 0, 1],
          [0, 1, 0],
          [0, 1, 0],
          [0, 1, 0],
          [1, 0, 0],
          [1, 0, 0]]

# X_data의 instance, columns
X = tf.placeholder("float", [None, 4])
# Y_data의 instance, columns
Y = tf.placeholder("float", [None, 3])
# Y_data의 label의 개수
nb_classes = 3

# X의 columns 개수, Y로 나가는 값(class 개수와 같음)
W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
# class 개수와 같음
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')
```

1. 데이터 설정

Softmax Classification 구현 TF



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

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

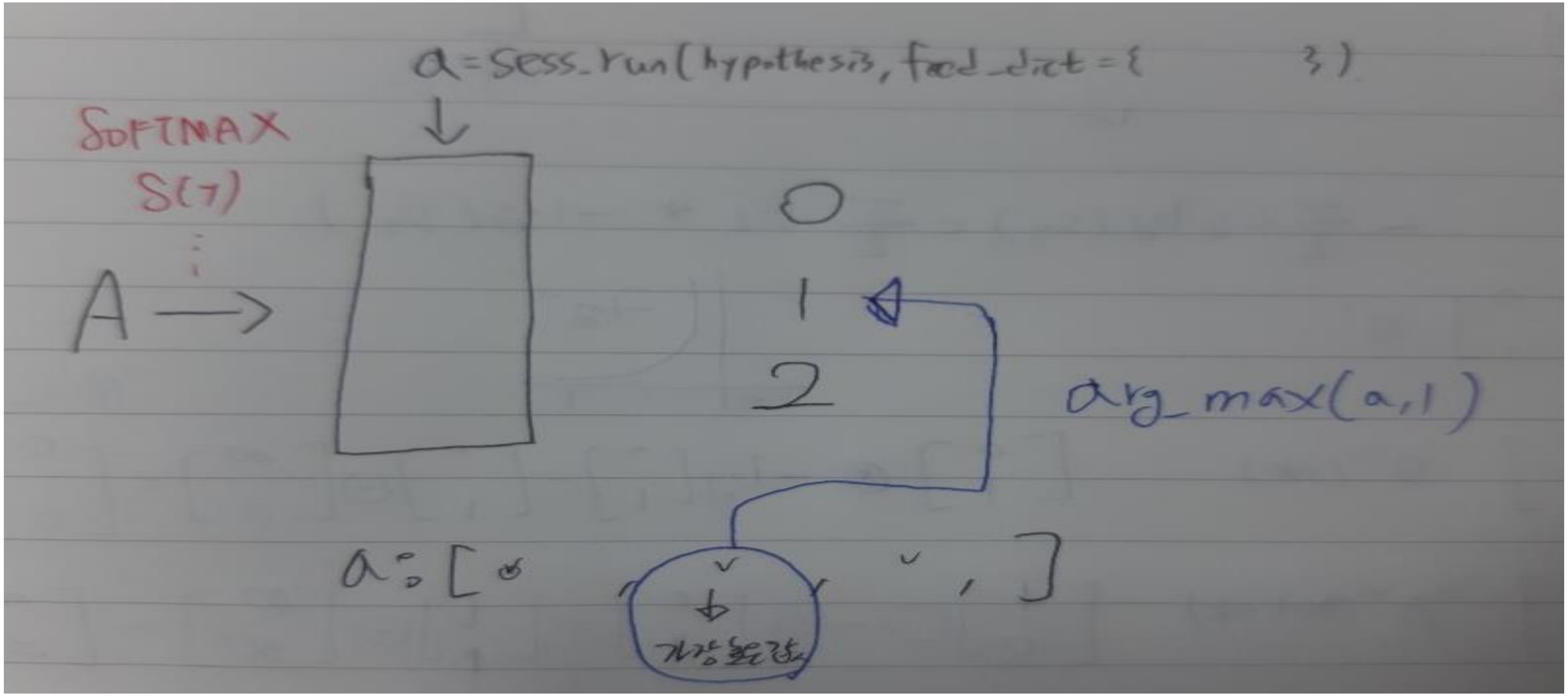


```
# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(2001):
        _, cost_val = sess.run([optimizer, cost], feed_dict={X: x_data, Y: y_data})

        if step % 200 == 0:
            print(step, cost_val)
```

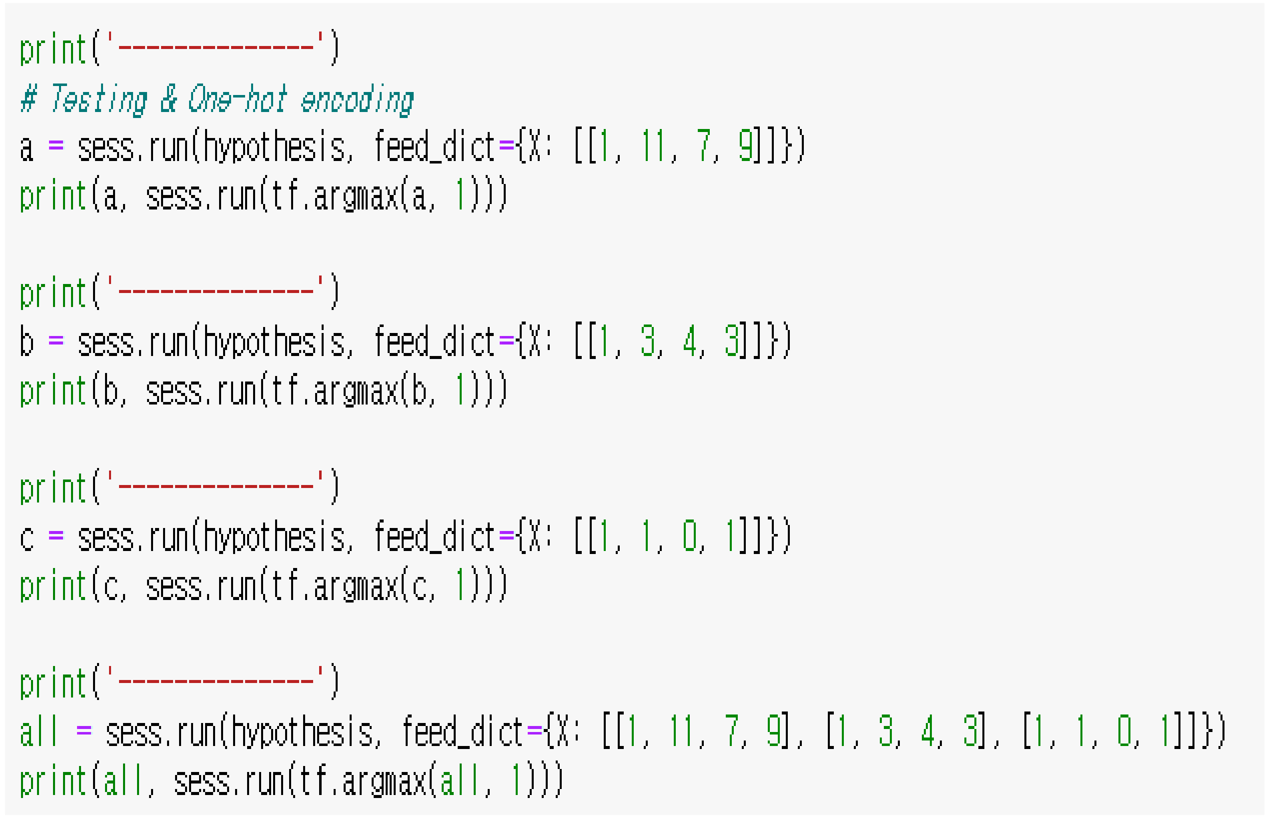
2. hypothesis, cost, optimizier 설정

3. 학습 시작

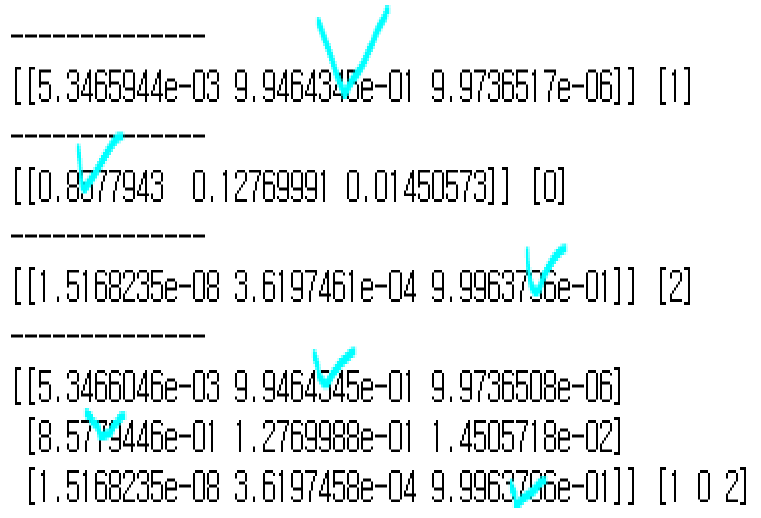
Softmax Classification 구현 TF



Softmax Classification 구현 TF



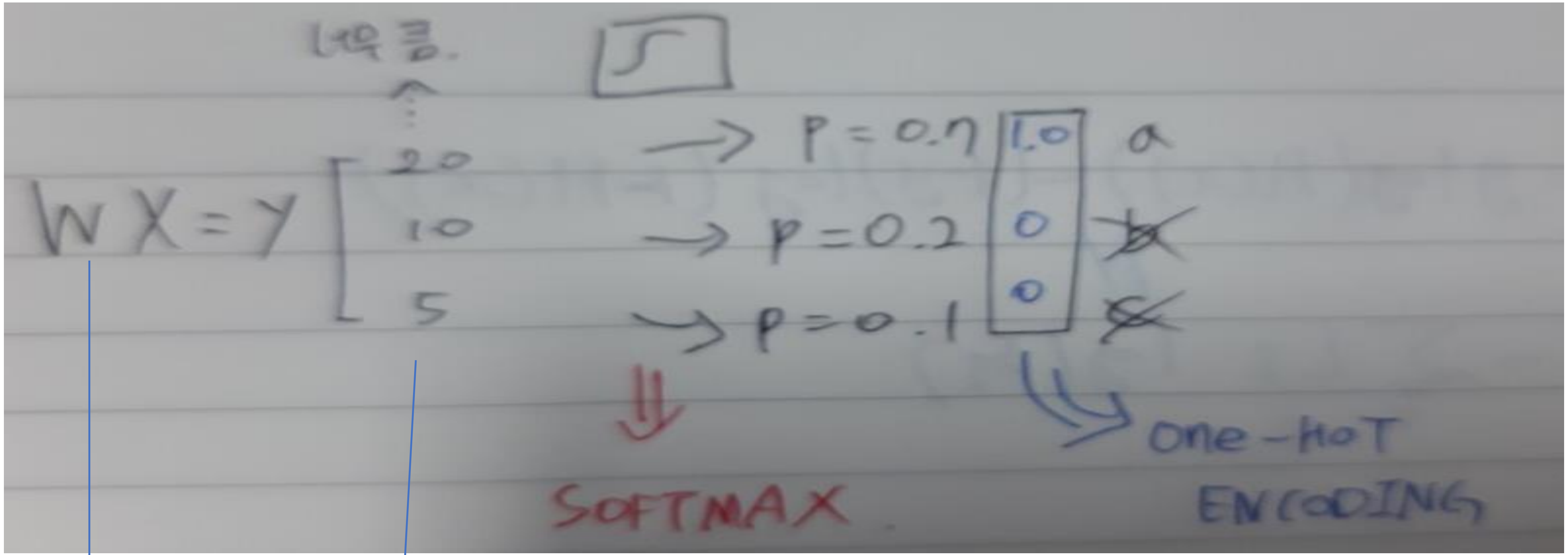
```
print('--------------')
# Testing & One-hot encoding
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
print(a, sess.run(tf.argmax(a, 1)))

print('--------------')
b = sess.run(hypothesis, feed_dict={X: [[1, 3, 4, 3]]})
print(b, sess.run(tf.argmax(b, 1)))

print('--------------')
c = sess.run(hypothesis, feed_dict={X: [[1, 1, 0, 1]]})
print(c, sess.run(tf.argmax(c, 1)))

print('--------------')
all = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]]})
print(all, sess.run(tf.argmax(all, 1)))
```



```
--------------
[[5.3465944e-03 9.9464345e-01 9.9736517e-06]] [1]
--------------
[[0.8577943  0.12769991 0.01450573]] [0]
--------------
[[1.5168235e-08 3.6197461e-04 9.9963706e-01]] [2]
--------------
[[5.3466046e-03 9.9464345e-01 9.9736508e-06]
 [8.5779446e-01 1.2769988e-01 1.4505718e-02]
 [1.5168235e-08 3.6197458e-04 9.9963706e-01]] [1 0 2]
```

4. 결과해석
첫번째 : [[1, 11, 7, 9]] 결과 : 1 class
두번째 : [[1, 3, 4, 3]] 결과 : 0 class
세번째 : [[1, 1, 0, 1]] 결과 : 2 class

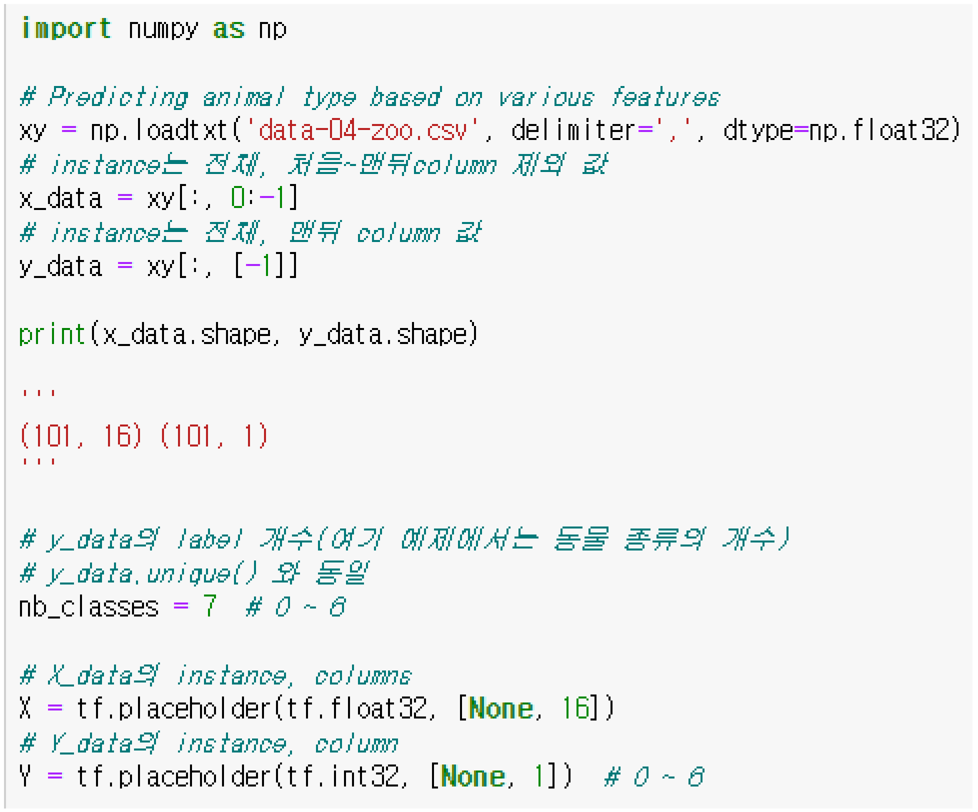
Fancy Softmax Classification TF



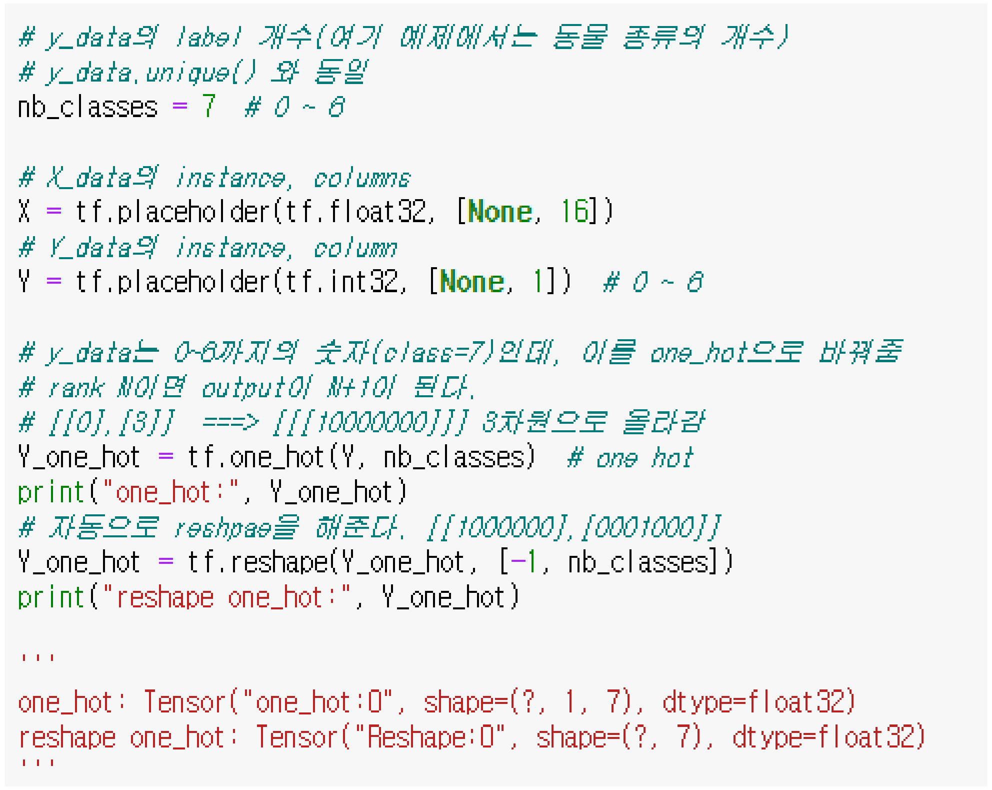
Logistic classifier

scores

Fancy Softmax Classification TF



```
import numpy as np

# Predicting animal type based on various features
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
# instance는 전체, 처음~맨뒤column 제외 값
x_data = xy[:, 0:-1]
# instance는 전체, 맨뒤 column 값
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

'''
(101, 16) (101, 1)
'''

# y_data의 label 개수(여기 예제에서는 동물 종류의 개수)
# y_data.unique() 와 동일
nb_classes = 7  # 0 ~ 6

# X_data의 instance, columns
X = tf.placeholder(tf.float32, [None, 16])
# Y_data의 instance, column
Y = tf.placeholder(tf.int32, [None, 1])  # 0 ~ 6
```



```
# y_data의 label 개수(여기 예제에서는 동물 종류의 개수)
# y_data.unique() 와 동일
nb_classes = 7  # 0 ~ 6

# X_data의 instance, columns
X = tf.placeholder(tf.float32, [None, 16])
# Y_data의 instance, column
Y = tf.placeholder(tf.int32, [None, 1])  # 0 ~ 6

# y_data는 0-6까지의 숫자(class=7)인데, 이를 one_hot으로 바꿔줌
# rank N이면 output이 N+1이 된다.
# [[0],[3]]  ===> [[[1000000]]] 3차원으로 올라감
Y_one_hot = tf.one_hot(Y, nb_classes)  # one hot
print("one_hot:", Y_one_hot)
# 자동으로 reshpae를 해준다. [[1000000],[0001000]]
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
print("reshape one_hot:", Y_one_hot)

'''
one_hot: Tensor("one_hot:0", shape=(?, 1, 7), dtype=float32)
reshape one_hot: Tensor("Reshape:0", shape=(?, 7), dtype=float32)
'''
```

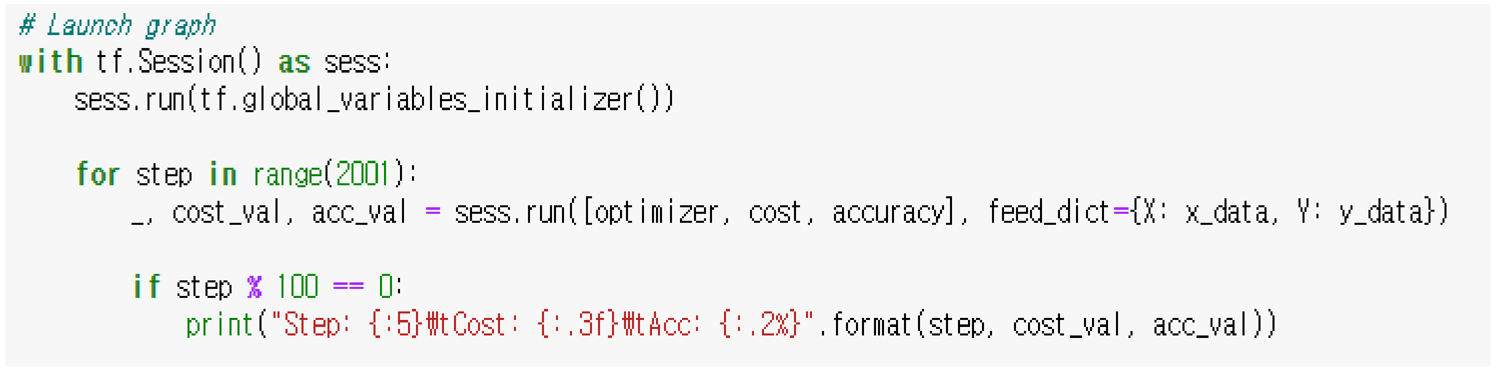
1. 데이터 설정

Fancy Softmax Classification TF

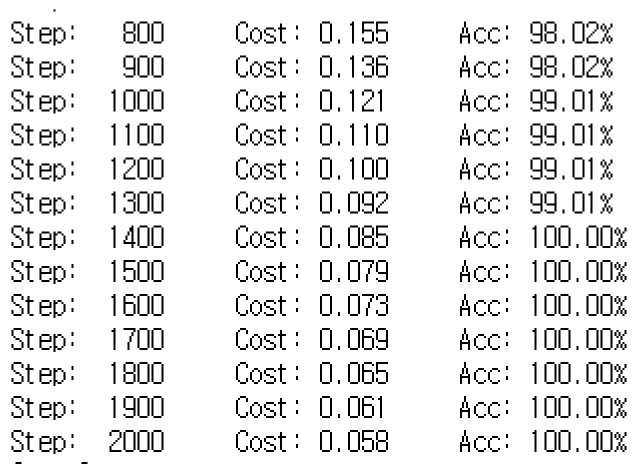
```
# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

# Cross entropy cost/loss
# cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis), axis=1)) 과 동일한 표현임
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,
                                                                  labels=tf.stop_gradient([Y_one_hot])))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# softmax에서 가장 큰 값을 찾음(예측)
prediction = tf.argmax(hypothesis, 1)
# 정답인지 판단
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
# 정답률 판단
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

2. hypothesis, cost, optimizier, prediction, 정확도 예측 설정

Fancy Softmax Classification TF



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

    for step in range(2001):
        _, cost_val, acc_val = sess.run([optimizer, cost, accuracy], feed_dict={X: x_data, Y: y_data})

        if step % 100 == 0:
            print("Step: {:5}₩tCost: {:.3f}₩tAcc: {:.2%}".format(step, cost_val, acc_val))
```

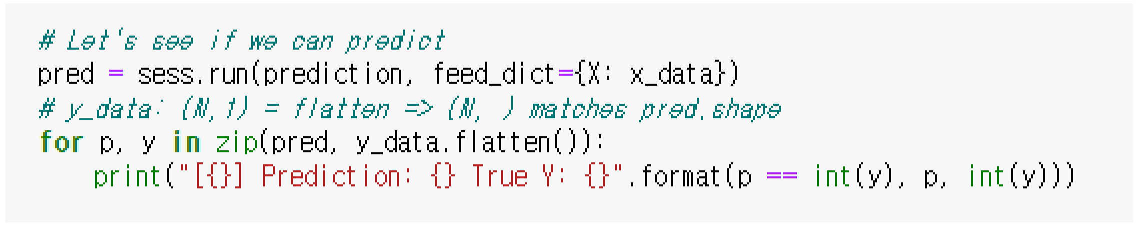


```
Step:   800     Cost: 0.155     Acc: 98.02%
Step:   900     Cost: 0.136     Acc: 98.02%
Step:  1000     Cost: 0.121     Acc: 99.01%
Step:  1100     Cost: 0.110     Acc: 99.01%
Step:  1200     Cost: 0.100     Acc: 99.01%
Step:  1300     Cost: 0.092     Acc: 99.01%
Step:  1400     Cost: 0.085     Acc: 100.00%
Step:  1500     Cost: 0.079     Acc: 100.00%
Step:  1600     Cost: 0.073     Acc: 100.00%
Step:  1700     Cost: 0.069     Acc: 100.00%
Step:  1800     Cost: 0.065     Acc: 100.00%
Step:  1900     Cost: 0.061     Acc: 100.00%
Step:  2000     Cost: 0.058     Acc: 100.00%
```

3. 데이터 학습

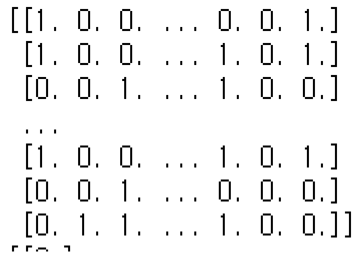
해석 : Loss(Cost)는 점점 최소화 되면서 Acc(정확도)는 점점 증가한다.

Fancy Softmax Classification TF

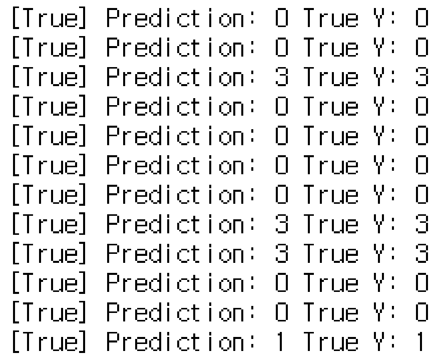


```
# Let's see if we can predict
pred = sess.run(prediction, feed_dict={X: x_data})
# y_data: (N,1) = flatten => (N, ) matches pred.shape
for p, y in zip(pred, y_data.flatten()):
    print("[{}] Prediction: {} True Y: {}".format(p == int(y), p, int(y)))
```

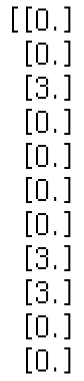
4. 원래 있던 x_data들을 넣고 y값을 예측해본다.



```
[[1. 0. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 1. 0. 1.]
 [0. 0. 1. ... 1. 0. 0.]
 ...
 [1. 0. 0. ... 1. 0. 1.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 1. 0. 0.]]
```



```
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 1 True Y: 1
```



```
[[0.]
 [0.]
 [3.]
 [0.]
 [0.]
 [0.]
 [0.]
 [3.]
 [3.]
 [0.]
 [0.]
```

해석 : 원래 x data를 instance로 예측했더니 위와 같이 나왔고 이는 원래 y값과 같다.