

## ML\_Lec 04.

### Multi-Variable Linear Regression.

multi-variable/feature			
$x_1$ (quiz 1)	$x_2$ (quiz 2)	$x_3$ (midterm 1)	Y (final)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

$$H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$H(x_1, x_2, x_3, \dots, x_n) = w_1x_1 + w_2x_2 + w_3x_3 + w_nx_n + b$$

### Cost Function. //손실함수

$$Cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}) - y^{(i)})^2 \quad (m: \text{예}/\text{오}/\text{예} \text{ 수})$$

### Matrix. //행렬

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n \quad (b \text{는 편의상 생략})$$

## Matrix Multiplication.

$$H(X) = XW$$

1. 
$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1 w_1 + x_2 w_2 + x_3 w_3)$$

2. 
$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

$[5, 3] \quad [3, 1] \quad [5, 1]$

♠  $[n, 3] \quad [3, m] \quad [n, m]$

**Wx & XW.**

- Lecture (theory):

$$H(x) = Wx + b$$

- Implementation (TensorFlow)

$$H(X) = XW$$

## ML\_Lab 04-1.

### Test.

```
1 x1_data = [73, 93, 89, 96, 73]
2 x2_data = [80, 88, 91, 98, 66]
3 x3_data = [75, 93, 90, 100, 70]
4 y_data = [152, 185, 180, 196, 142]
5
6 x1 = tf.placeholder(tf.float32)
7 x2 = tf.placeholder(tf.float32)
8 x3 = tf.placeholder(tf.float32)
9 y = tf.placeholder(tf.float32)
10
11 w1 = tf.Variable(tf.random_normal([1]), name='weight1')
12 w2 = tf.Variable(tf.random_normal([1]), name='weight2')
13 w3 = tf.Variable(tf.random_normal([1]), name='weight3')
14 b = tf.Variable(tf.random_normal([1]), name='bias')
15
16 hypothesis = x1*w1 + x2*w2 + x3*w3 + b
17
18 cost = tf.reduce_mean(tf.square(hypothesis - y))
19
20 optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
21 train = optimizer.minimize(cost)
22
23 sess = tf.Session()
24 sess.run(tf.global_variables_initializer())
25 for step in range(2001):
26     cost_val, hy_val, _ = sess.run([cost, hypothesis, train],
27                                     feed_dict={x1: x1_data, x2: x2_data, x3: x3_data, y: y_data,})
28     if step % 10 == 0:
29         print("Step: ", step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

```
1 x_data = [[73, 80, 75], [93, 88, 93], [89, 91, 90], [96, 98, 100], [73, 66, 70]]
2 y_data = [[152], [185], [180], [196], [142]]
3
4 x = tf.placeholder(tf.float32, shape=[None, 3])
5 y = tf.placeholder(tf.float32, shape=[None, 1])
6 w = tf.Variable(tf.random_normal([3, 1]), name='weight')
7 b = tf.Variable(tf.random_normal([1]), name='bias')
8
9 hypothesis = tf.matmul(x, w)+b #.matmul()
10
11 cost = tf.reduce_mean(tf.square(hypothesis - y))
12
13 optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
14 train = optimizer.minimize(cost)
15
16 sess = tf.Session()
17 sess.run(tf.global_variables_initializer())
18 for step in range(2001):
19     cost_val, hy_val, _ = sess.run([cost, hypothesis, train], feed_dict={x: x_data, y: y_data})
20     if step % 10 == 0:
21         print("Step: ", step, "Cost: ", cost_val, "\nPrediction\n", hy_val)
```

```
[141.8001 ]]
Step: 1930 Cost: 0.17629588
Prediction
[[151.54753]
 [184.5915 ]
 [180.60968]
 [196.31358]
 [141.8004 ]]
```

## ML\_Lab 04-2.

### Loading Data form File.

data-01-test-score.csv

```
# EXAM1,EXAM2,EXAM3,FINAL
73,80,75,152
93,88,93,185
89,91,90,180
96,98,100,196
73,66,70,142
53,46,55,101
```

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

xy = np.loadtxt('data-01-test-score.csv', delimiter=',',
dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# Make sure the shape and data are OK
print(x_data.shape, x_data, len(x_data))
print(y_data.shape, y_data)

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([3, 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
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)
```

```
# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())
# Set up feed_dict variables inside the loop.
for step in range(2001):
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train],
        feed_dict={X: x_data, Y: y_data})
    if step % 10 == 0:
        print(step, "Cost: ", cost_val,
              "\nPrediction:\n", hy_val)

# Ask my score
print("Your score will be ", sess.run(hypothesis,
    feed_dict={X: [[100, 70, 101]]}))

print("Other scores will be ", sess.run(hypothesis,
    feed_dict={X: [[60, 70, 110], [90, 100, 80]]}))
```

## Slicing.

```
nums = range(5)      # range is a built-in function that creates a list of integers
print nums           # Prints "[0, 1, 2, 3, 4]"
print nums[2:4]       # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print nums[2:]        # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print nums[:2]        # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print nums[:]         # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print nums[:-1]       # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]    # Assign a new sublist to a slice
print nums            # Prints "[0, 1, 8, 9, 4]"
```

## Indexing, Slicing, Iterating.

```
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
# array([[ 1,  2,  3,  4],
#        [ 5,  6,  7,  8],
#        [ 9, 10, 11, 12]])

b[:, 1]
# array([ 2,  6, 10])

b[-1]
# array([ 9, 10, 11, 12])

b[-1, :]
# array([ 9, 10, 11, 12])

b[-1, ...]
# array([ 9, 10, 11, 12])

b[0:2, :]
# array([[1, 2, 3, 4],
#        [5, 6, 7, 8]])
```

```
a = np.array([1, 2, 3, 4, 5])
# array([1, 2, 3, 4, 5])

a[1:3]
# array([2, 3])

a[-1]
# 5

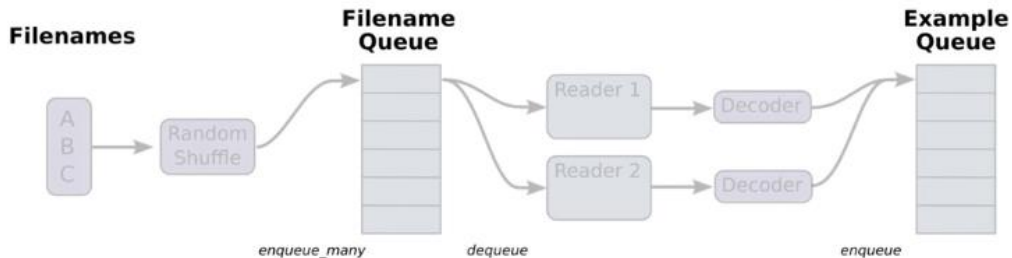
a[0:2] = 9

a
# array([9, 9, 3, 4, 5])
```

1. Arrays can be indexed, sliced, iterated much like lists and other sequence types in Python
2. As with Python lists, slicing in NumPy can be accomplished with the colon (:) syntax
3. Colon instances (:) can be replaced with dots (...)

## Queue Runners.

```
1 filename_queue = tf.train.string_input_producer(  
    ['data-01-test-score.csv', 'data-02-test-score.csv', ... ],  
    shuffle=False, name='filename_queue')  
  
3 record_defaults = [[0.], [0.], [0.], [0.]]  
xy = tf.decode_csv(value, record_defaults=record_defaults)
```



```
2 reader = tf.TextLineReader()  
key, value = reader.read(filename_queue)
```

1. 여러 개의 파일을 읽을 때, Filename Queue 에 쌓는다. (+Random Shuffle)
2. Reader 로 연결한다.
3. Decoder 를 거쳐 Example Queue 에 쌓는다.
4. 일정 배치만큼 읽어와 학습시킨다.

```
# collect batches of csv in  
train_x_batch, train_y_batch = \  
    tf.train.batch([xy[0:-1], xy[-1:]], batch_size=10)  
  
sess = tf.Session()  
...  
  
# Start populating the filename queue.  
coord = tf.train.Coordinator()  
threads = tf.train.start_queue_runners(sess=sess, coord=coord)  
  
for step in range(2001):  
    x_batch, y_batch = sess.run([train_x_batch, train_y_batch])  
    ...  
  
coord.request_stop()  
coord.join(threads)
```

```
import tensorflow as tf
filename_queue = tf.train.string_input_producer(
    ['data-01-test-score.csv'], shuffle=False, name='filename_queue')
```

```
reader = tf.TextLineReader()
key, value = reader.read(filename_queue)
```

```
# Default values, in case of empty columns. Also specifies the type of the
# decoded result.
```

```
record_defaults = [[0.], [0.], [0.], [0.]]
xy = tf.decode_csv(value, record_defaults=record_defaults)
```

```
# collect batches of csv in
train_x_batch, train_y_batch = \
    tf.train.batch([xy[0:-1], xy[-1:]], batch_size=10)
```

```
# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])
```

```
W = tf.Variable(tf.random_normal([3, 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
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)
```

```
# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

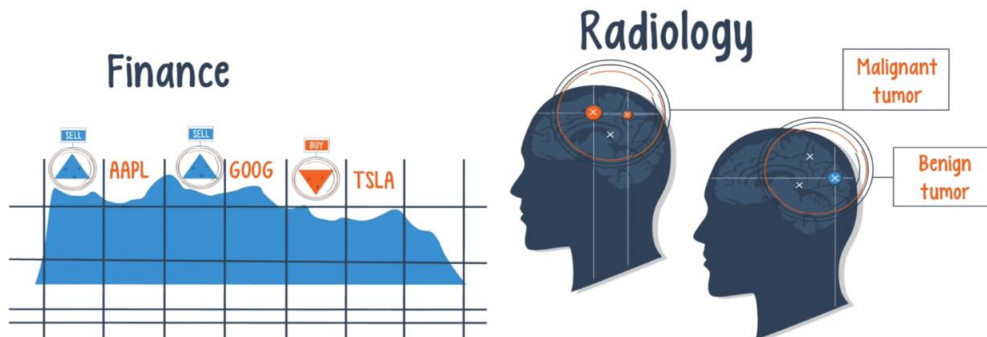
# Start populating the filename queue.
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(sess=sess, coord=coord)

for step in range(2001):
    x_batch, y_batch = sess.run([train_x_batch, train_y_batch])
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train],
        feed_dict={X: x_batch, Y: y_batch})
    if step % 10 == 0:
        print(step, "Cost: ", cost_val,
              "\nPrediction:\n", hy_val)

coord.request_stop()
coord.join(threads)
```

## ML\_Lec 05-1.

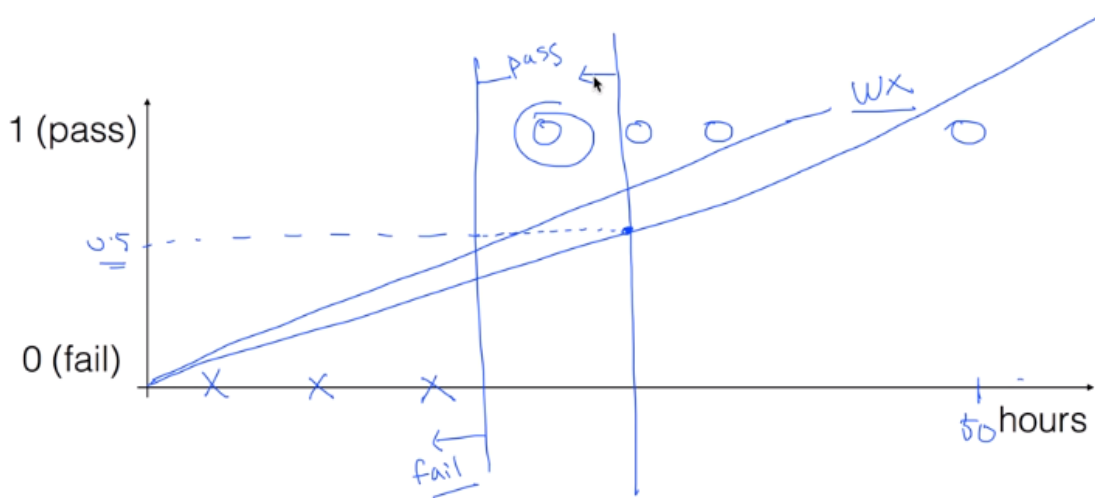
### Binary Classification.



#### ♠ 0,1 Encoding

- 1) Spam Detection: Spam(1) or Ham(0)
- 2) Facebook Feed: Show(1) or Hide(0)
- 3) Credit Card Fraudulent Transaction: Legitimate(0) or Fraud(1)

### Pass(1)/Fail(0) Based on Study Hours. (If Linear Regression.)



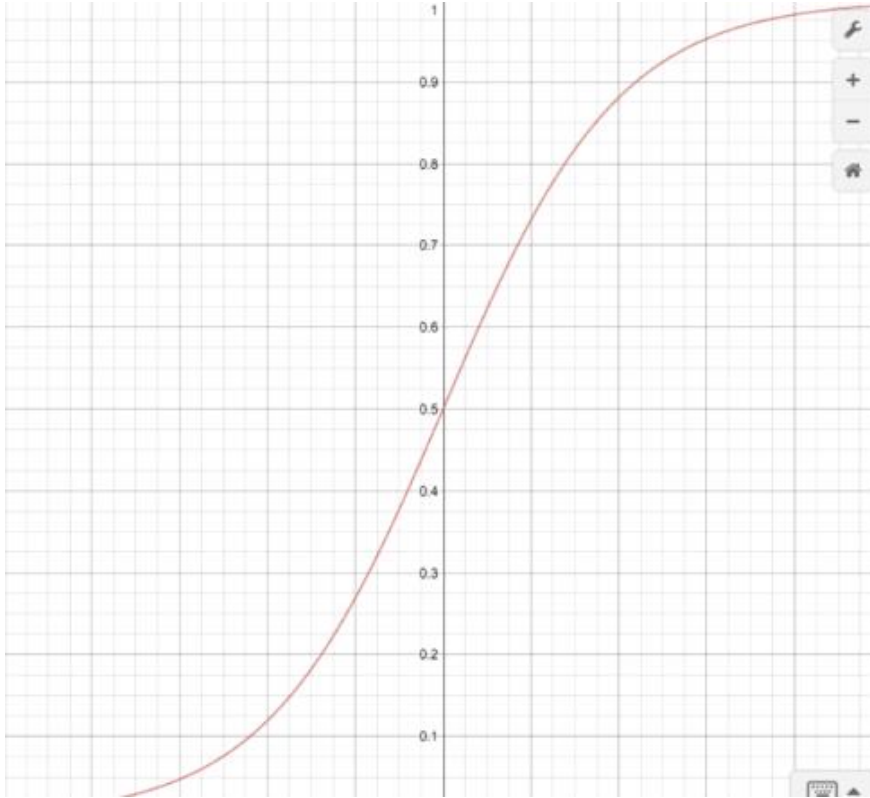
#### ♠ 문제점

- 1) 수치가 커지면(50hours) 선이 기울어진다.
- 2) 선이 기울어지면 기준값(0.5)의 위치가 바뀐다.
- 3) 기준값의 위치가 바뀌면 기존 결과의 판단이 달라진다.
- 4) 1보다 큰 값, 0보다 작은 값이 나올 수 있다.



## Logistic Regression. //로지스틱 회귀

$$g(z) = \frac{1}{1 + e^{-z}} \rightarrow H(x) = \frac{1}{1 + e^{-w^T X (-wT)}}$$

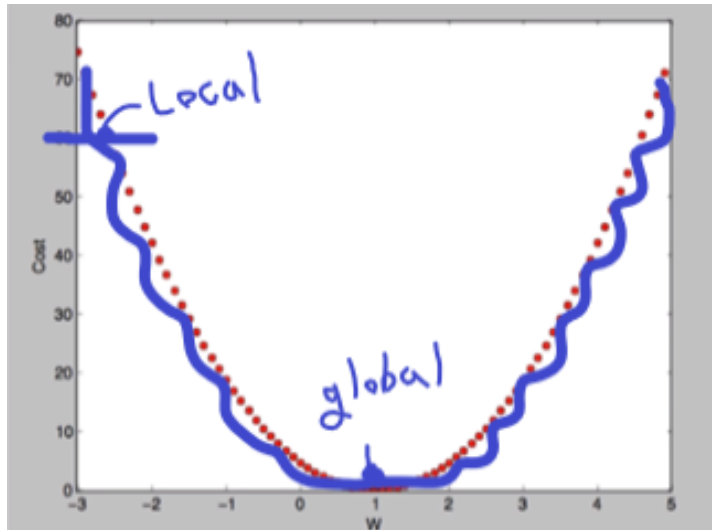


### ♠ Logistic Function, Sigmoid Function

Sigmoid: Curved in two directions, like the letter "S", or the Greek  $\varsigma$ (Sigma) //들여쓰기 예외

## ML\_Lec 05-2.

### Cost Function.



#### 1. 문제점

Local Cost에서 끝나 Global Cost로 갈 수 없다.

$$2. \text{Cost}(W) = \frac{1}{m} \sum c(H(x), y)$$

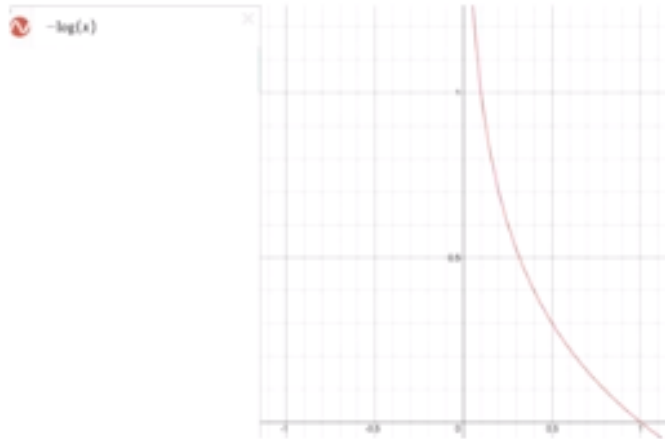
$$c(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases} \quad // \left(\frac{1}{1+e^{-z}}\right) \text{ 때문에 } \log(H(x)) \text{ 사용}$$

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

3.  $-\log(H(x)) : y = 1$

$H(x) = 1 \rightarrow \text{cost} = 0$

$H(x) = 0 \rightarrow \text{cost} = \infty$



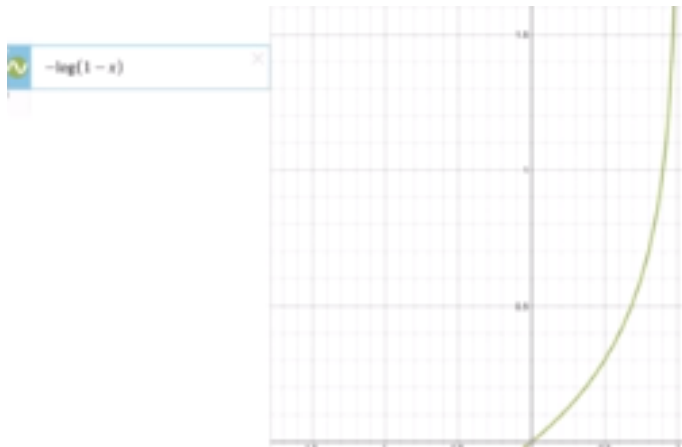
cost

H(x), W

4.  $-\log(1 - H(x)) : y = 0$

$H(x) = 0 \rightarrow \text{cost} = 0$

$H(x) = 1 \rightarrow \text{cost} = \infty$



cost

H(x), W

## Gradient Descent Algorithm.

$$\text{Cost}(W) = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) \log(1 - H(x))$$

$$W := W - \alpha \frac{\partial}{\partial W} \text{Cost}(W) \quad (\alpha = \text{Learning Rate})$$

## ML\_Lab 05.

### Test.

```
1 x_data = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
2 y_data = [[0], [0], [0], [1], [1], [1]] #0,1 Encoding
3
4 x = tf.placeholder(tf.float32, shape=[None, 2])
5 y = tf.placeholder(tf.float32, shape=[None, 1])
6 w = tf.Variable(tf.random_normal([2, 1]), name='weight') #.random_normal([n: 들어오는 개수, m: 나가는 개수])
7 b = tf.Variable(tf.random_normal([1]), name='bias') #.random_normal([m: 나가는 개수])
8 hypothesis = tf.sigmoid((tf.matmul(x, w) + b) * 0.1) #.sigmoid()
9
10 cost = tf.reduce_mean(y * tf.log(hypothesis) + (1-y) * tf.log(1-hypothesis)) #.log()
11 train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
12
13 predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32) #.cast()
14 accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, y), dtype=tf.float32)) #.equal()
15
16 with tf.Session() as sess:
17     sess.run(tf.global_variables_initializer())
18
19     for step in range(10001):
20         cost_val, _ = sess.run([cost, train], feed_dict={x: x_data, y: y_data})
21         if step % 200 == 0:
22             print(step, cost_val)
23
24     h, c, a = sess.run([hypothesis, predicted, accuracy],
25                         feed_dict={x: x_data, y: y_data})
26     print("\nHypothesis: ", h, "\nCorrect(Y): ", c, "\nAccuracy: ", a)
27
28 #그냥 실행을 돌릴 경우 nan이 출력되어 0.1을 hypothesis에 곱해주었다.
29 #이후 실행은 되지만, 제대로 된 결과가 나오지 않음.
```

## Classifying diabetes



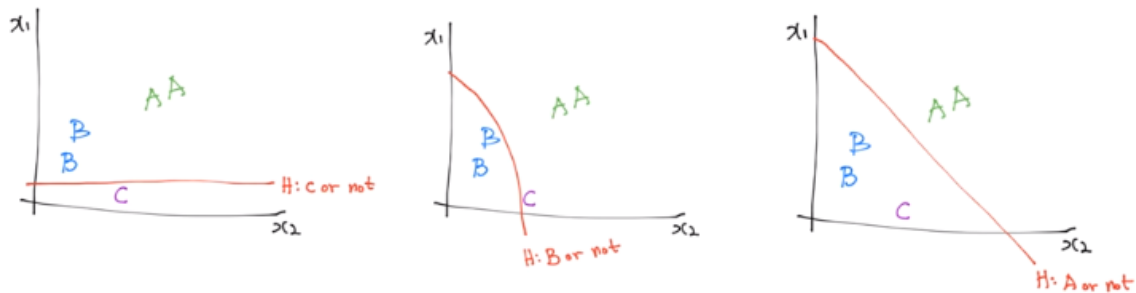
-0.411765	0.165829	0.213115	0	0	-0.23696	-0.894962	-0.7	1
-0.647059	-0.21608	-0.180328	-0.353535	-0.791962	-0.0760059	-0.854825	-0.833333	0
0.176471	0.155779	0	0	0	0.052161	-0.952178	-0.733333	1
-0.764706	0.979899	0.147541	-0.0909091	0.283688	-0.0909091	-0.931682	0.0666667	0
-0.0588235	0.256281	0.57377	0	0	0	-0.868488	0.1	0
-0.529412	0.105528	0.508197	0	0	0.120715	-0.903501	-0.7	1
0.176471	0.688442	0.213115	0	0	0.132638	-0.608027	-0.566667	0
0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1

```
xy = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

## ML\_Lec 06-1.

### Multinomial Classification.

Hyper Plane으로 각각의 경우를 구분하고, Matrix Multiplication을 이용한다. //n차원-1의 선형함수



$$1. X \rightarrow \text{「C」} \rightarrow Y$$

$$2. X \rightarrow \text{「B」} \rightarrow Y$$

$$3. X \rightarrow \text{「A」} \rightarrow Y$$

+++++

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

|||||

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix}$$

The diagram shows the matrix multiplication result being compared to the target values  $\bar{y}_A, \bar{y}_B, \bar{y}_C$ . The result is then passed through a sigmoid function  $\sigma$  to produce the final output  $\hat{Y}$ . The diagram is crossed out with a large 'X'.

## ML\_Lec 06-2.


### Softmax Function.

Softmax Function을 이용하여 값을 확률과 같이 바꾸고, One Hot Encoding을 이용한다.

1.

$$\begin{bmatrix} w_{A1}x_1 + v_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + v_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + v_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix} \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$


$0 \sim 1$   
↓





2.

LOGISTIC CLASSIFIER

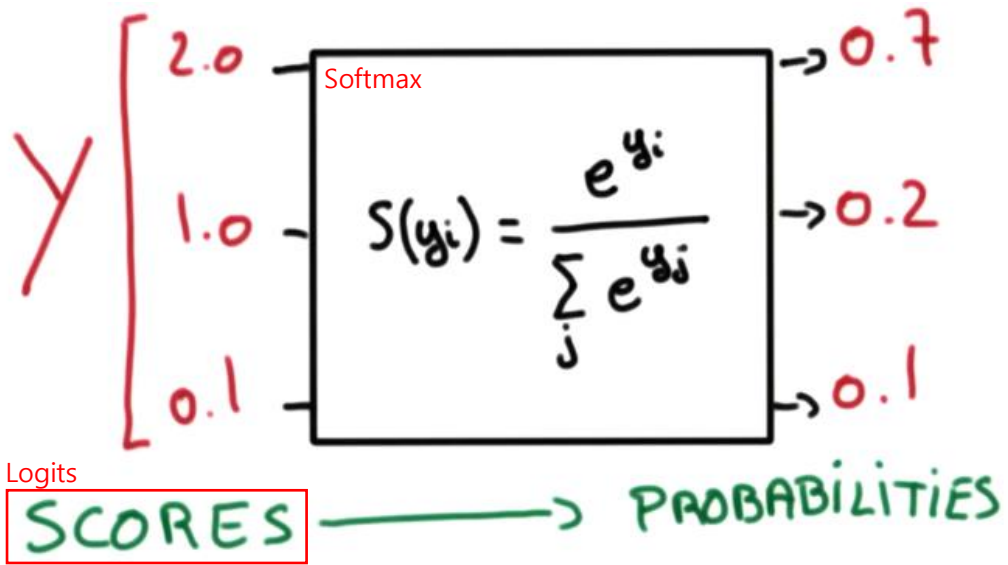
$$WX = Y \begin{bmatrix} A \\ B \\ C \end{bmatrix} \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

→  $p = 0.7$  

→  $p = 0.2$  

→  $p = 0.1$  

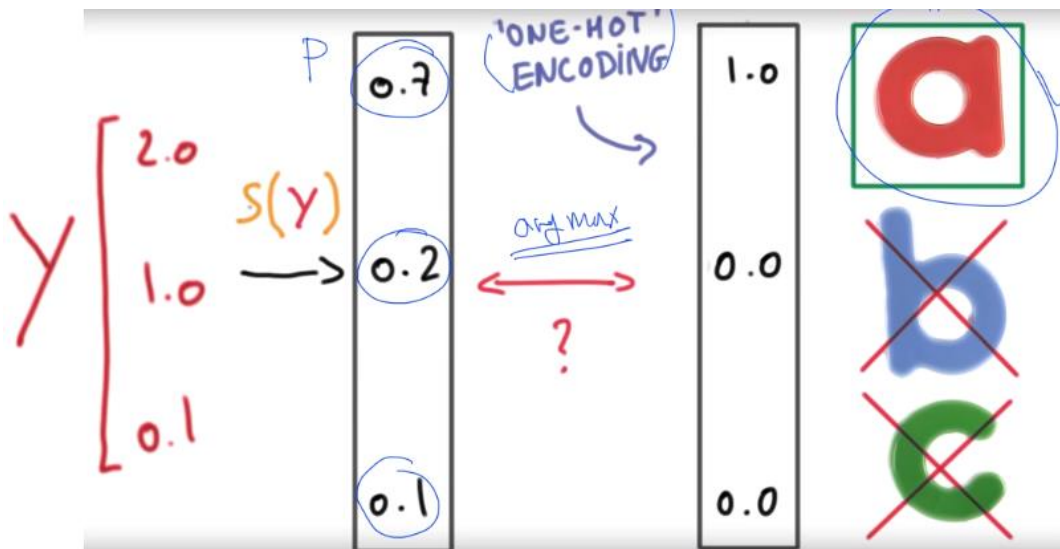
$\frac{2.0}{3.1} \approx 0.645$   
 $\frac{1.0}{3.1} \approx 0.323$   
 $\frac{0.1}{3.1} \approx 0.032$



3.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

- 1) 0~1의 값
- 2)  $\sum = 1$
- 3) 확률(p)

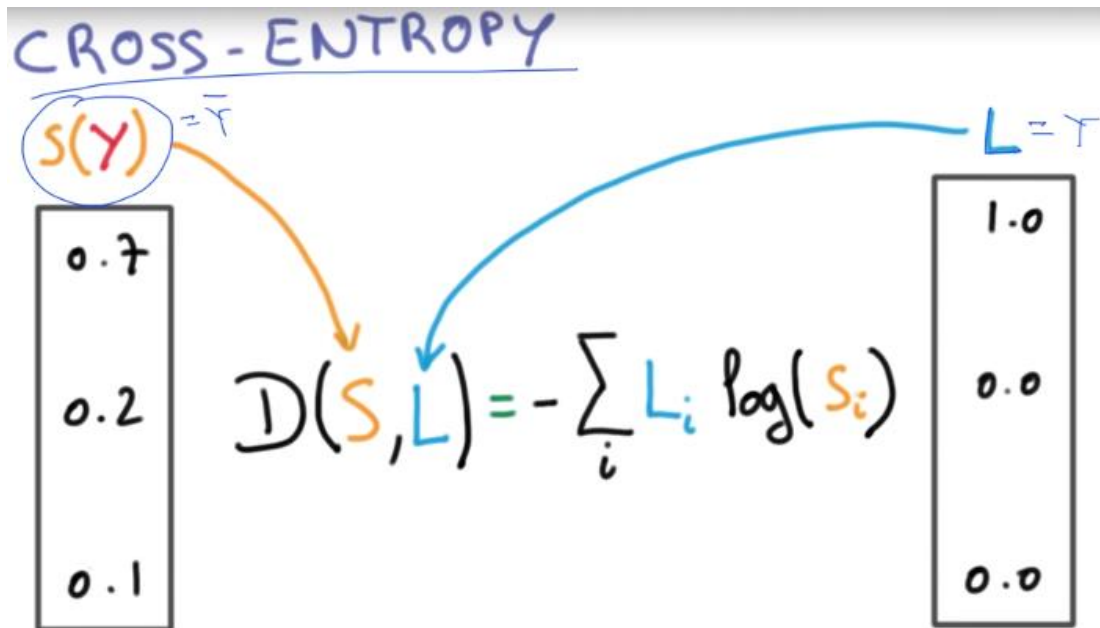


4.

#### ♠ One Hot Encoding

제일 큰 값(확률)을 1로 하고 나머지는 0으로 한다. //들여쓰기 예외

## Cross-Entropy Cost Function.



$$-\sum_j L_j \log(S_j) = \sum_j L_j \times (-\log(S_j)) \quad // -\log(S_j): 0 \sim 1$$

$$1. \quad L = \begin{bmatrix} A \\ B \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B \quad (\text{Real})$$

$$2. \quad S = \begin{bmatrix} A \\ B \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B \quad (\text{T}), \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot -\log \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \text{Cost} = 0$$

$$3. \quad S = \begin{bmatrix} A \\ B \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = A \quad (\text{F}), \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot -\log \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ \infty \end{bmatrix} \rightarrow \text{Cost} = \infty$$

## Logistic Cost & Cross Entropy.

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

$// ?$

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

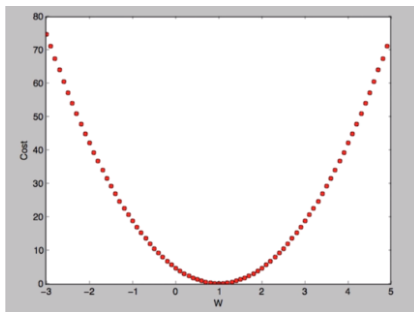


## Cost Function.

$$\mathcal{L}_{oss} = \frac{1}{N} \sum_i D(S(WX_i + b), L_i) \quad (i = \text{Training Set})$$

## Gradient Descent Algorithm.

$-\alpha \Delta \mathcal{L}(w_1, w_2)$  //자세한 미분은 다루지 않음.



## ML\_Lab 06-1.

### Test.

```
1 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]]
2 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]]
3 #One Hot Encoding [0,0,1]=2, [0,1,0]=1, [1,0,0]=0
4
5 x = tf.placeholder(tf.float32, shape=[None, 4])
6 y = tf.placeholder(tf.float32, shape=[None, 3])
7 nb_classes = 3
8 w = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
9 b = tf.Variable(tf.random_normal([nb_classes]), name='bias')
10 hypothesis = tf.nn.softmax(tf.matmul(x, w) + b) #.softmax()
11
12 cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(hypothesis), axis=1))
13 optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
14
15 sess = tf.Session()
16 sess.run(tf.global_variables_initializer())
17
18 for step in range(2001):
19     sess.run(optimizer, feed_dict={x: x_data, y: y_data})
20     if step % 200 == 0:
21         print("Step:", step, "Cost:", sess.run(cost, feed_dict={x: x_data, y: y_data}))
```

```
Step: 0 Cost: 7.807994
Step: 200 Cost: 0.58248025
Step: 400 Cost: 0.48184776
Step: 600 Cost: 0.4060764
Step: 800 Cost: 0.33714813
Step: 1000 Cost: 0.2666617
Step: 1200 Cost: 0.22295333
Step: 1400 Cost: 0.20297591
```

```
1 a = sess.run(hypothesis, feed_dict={x: [[1,11,7,9]]})
2 print(a, sess.run(tf.argmax(a, 1))) #.arg_max()
3
4 print('-----')
5
6 b = sess.run(hypothesis, feed_dict={x: [[1,3,4,3]]})
7 print(b, sess.run(tf.argmax(b, 1)))
8
9 print('-----')
10
11 c = sess.run(hypothesis, feed_dict={x: [[1,1,0,1]]})
12 print(c, sess.run(tf.argmax(c, 1)))
13
14 print('-----')
15
16 all = sess.run(hypothesis, feed_dict={x: [[1,11,7,9], [1,3,4,3], [1,1,0,1]]})
17 print(all, sess.run(tf.argmax(all, 1)))
```

```
[[3.2509041e-01 6.7477059e-01 1.3895857e-04]] [1]
-----
[[0.47370106 0.43137628 0.09492271]] [0]
-----
[[5.8806567e-05 1.0025946e-02 9.8991525e-01]] [2]
-----
[[3.2509065e-01 6.7477041e-01 1.3895867e-04]
 [4.7370121e-01 4.3137610e-01 9.4922714e-02]
 [5.8806570e-05 1.0025946e-02 9.8991525e-01]] [1 0 2]
```

## ML\_Lab 06-2.

### Softmax\_cross\_entropy\_with\_logits.

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

1

```
# 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=logits,
                                                  labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
```

1. ML\_Lab 06-1.에서의 코스트
2. .softmax\_cross\_entropy\_with\_logits()을 사용한 코스트

### Test.

```
1 #원래 강의와 다르게 직접 데이터를 입력함.
2
3 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]]
4 y_data = [[2], [2], [2], [1], [1], [1], [0], [0]]
5
6 x = tf.placeholder(tf.float32, shape=[None, 4])
7 y = tf.placeholder(tf.int32, shape=[None, 1])
8 y_one_hot = tf.one_hot(y, nb_classes) #.one_hot()
9 y_one_hot = tf.reshape(y_one_hot, [-1, nb_classes]) #.reshape()
10 #rank N0 rank N+10이 되기 때문에 .one_hot() 이후 .reshape()을 사용한다.
11 nb_classes = 3
12 w = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
13 b = tf.Variable(tf.random_normal([nb_classes]), name='bias')
14 logits = tf.matmul(x, w) + b
15 hypothesis = tf.nn.softmax(logits)
16
17
18 cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y_one_hot)
19 cost = tf.reduce_mean(cost_i) #.softmax_cross_entropy_with_logits()
20 optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
21
22 sess = tf.Session()
23 sess.run(tf.global_variables_initializer())
24
25 for step in range(2001):
26     sess.run(optimizer, feed_dict={x: x_data, y: y_data})
27     if step % 200 == 0:
28         print("Step:", step, "Cost:", sess.run(cost, feed_dict={x: x_data, y: y_data}))
```

## Animal Classification.

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

nb_classes = 7 # 0 ~ 6

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

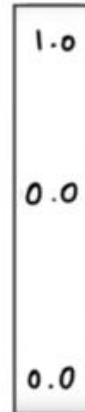
Y_one_hot = tf.one_hot(Y, nb_classes) # one hot
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])

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

# 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)

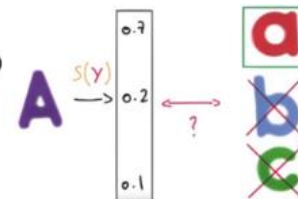
# Cross entropy cost/Loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                  labels=Y_one_hot)
```

'ONE-HOT'  
ENCODING



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

```
prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
```



```
for step in range(2000):
    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}\tLoss: {:.3f}\tAcc: {:.2%}".format(
            step, loss, acc))
```

```
# 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)))
```

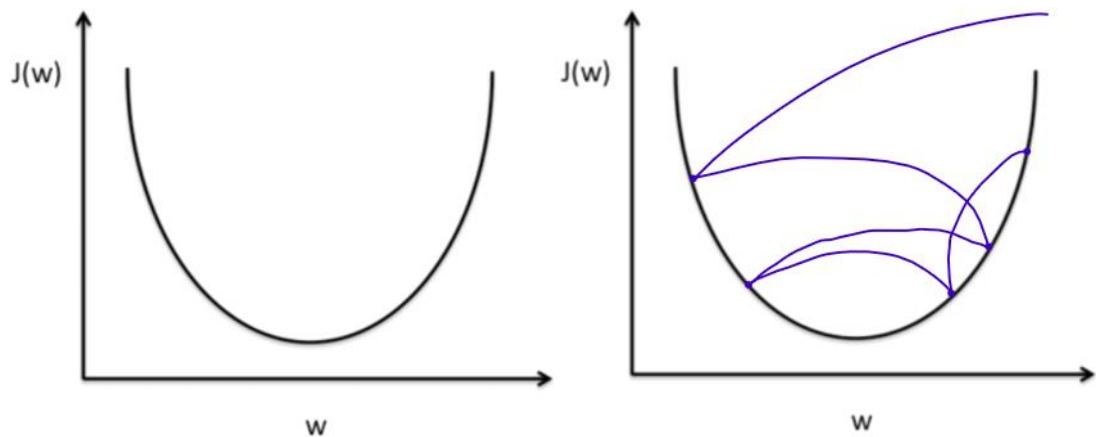
```
Step: 1100 Loss: 0.101 Acc: 99.01%
Step: 1200 Loss: 0.092 Acc: 100.00%
Step: 1300 Loss: 0.084 Acc: 100.00%
...
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
```

## ML\_Lec 07-1.

About Machine Learning Tips.

### Learning Rate. //학습률

1. Large Learning Rate: Overshooting.

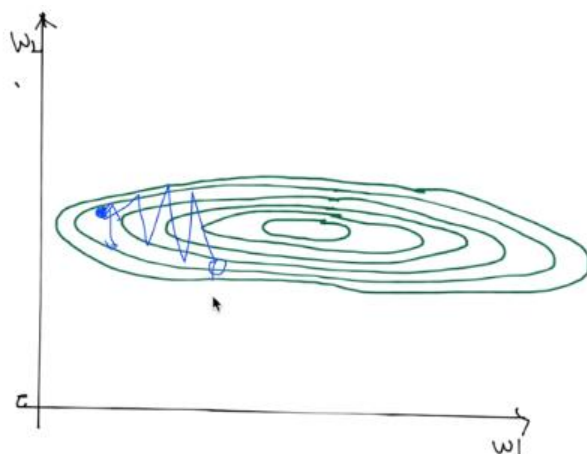


2. Small Learning Rate: Takes too long & Stops at local minimum. //강조 예외
3. Try several Learning Rates
  - 1) Observe the Cost Function.
  - 2) Check it goes down in a reasonable rate.

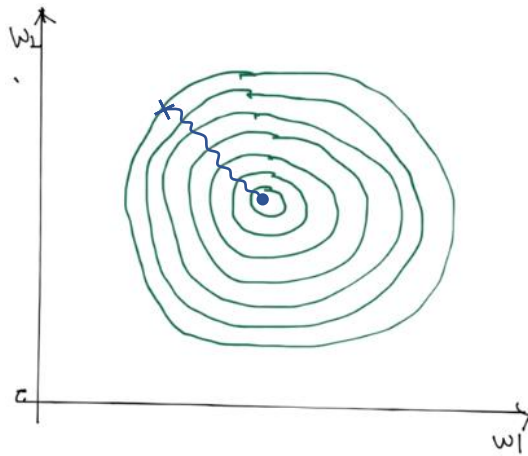
### Data(X) Preprocessing for Gradient Descent. //데이터 전처리.

1. 값의 차이가 심하게 날 경우

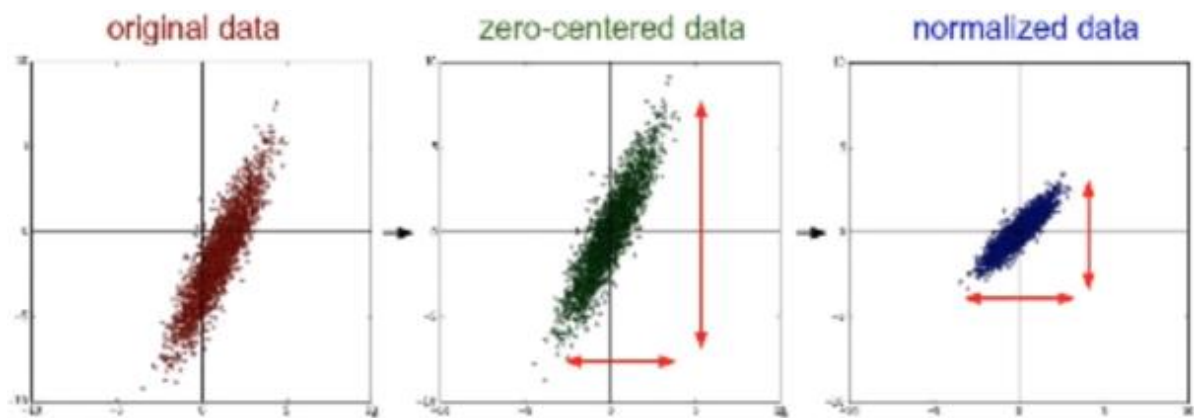
x1	x2	y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C



## 2. 정상적인 경우



## 3. Data(X) Preprocessing

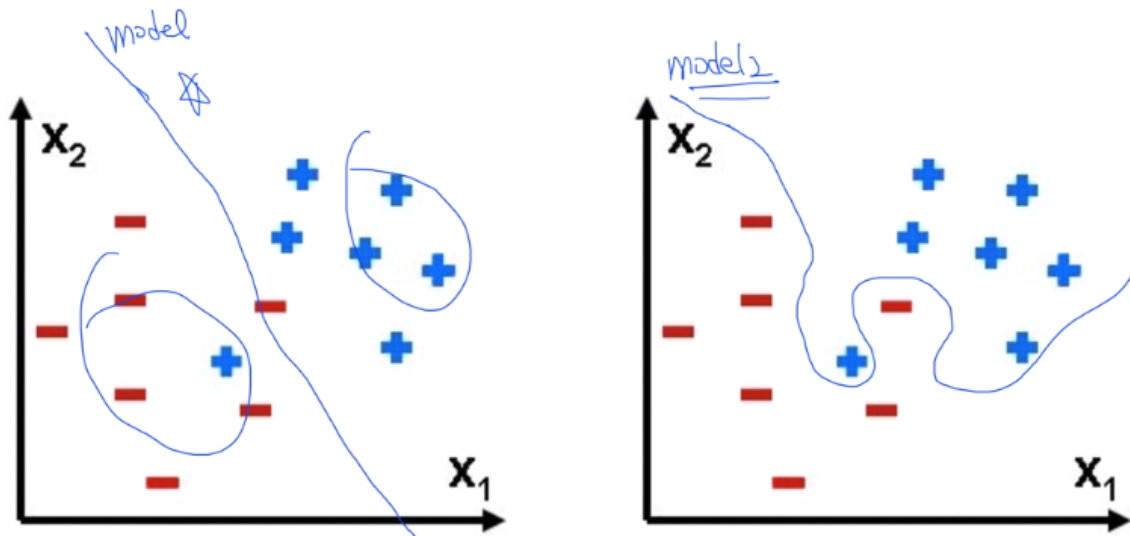


♠ **Standardization** //표준화

$$x'_j = \frac{x_j - \mu_j}{\sigma_j} \text{ //들여쓰기 예외}$$

Code: `X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()` //들여쓰기 예외

## Overfitting. //과적합



1. Our model is very good with Training Data Set. (with memorization)
2. Not good at test dataset or in real use.
3. Solutions for Overfitting
  - 1) More Training Data
  - 2) Reduce the number of features
  - 3) **Regularization**

## Regularization. //일반화, 정규화

Let's not have too big numbers in the weight.

1.  $Loss = \frac{1}{N} \sum_j D(S(WX_i + b), L_i) + \lambda \sum W^2$  ( $\lambda = Regularization\ Strength$ )
  - 1)  $\lambda = 0$  No Regularization
  - 2)  $\lambda = 1$  High Regularization
  - 3)  $\lambda = 0.001$  Little Regularization
2. Code: `l2reg = 0.001 * tf.reduce_sum(tf.square(W))`

## ML\_Lec 07-2.

### Evaluation using Training Set.

모든 데이터를 학습시키는 것은 좋은 방법이 아니다.

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

1. 100% Correct (Accuracy)
2. Can memorize

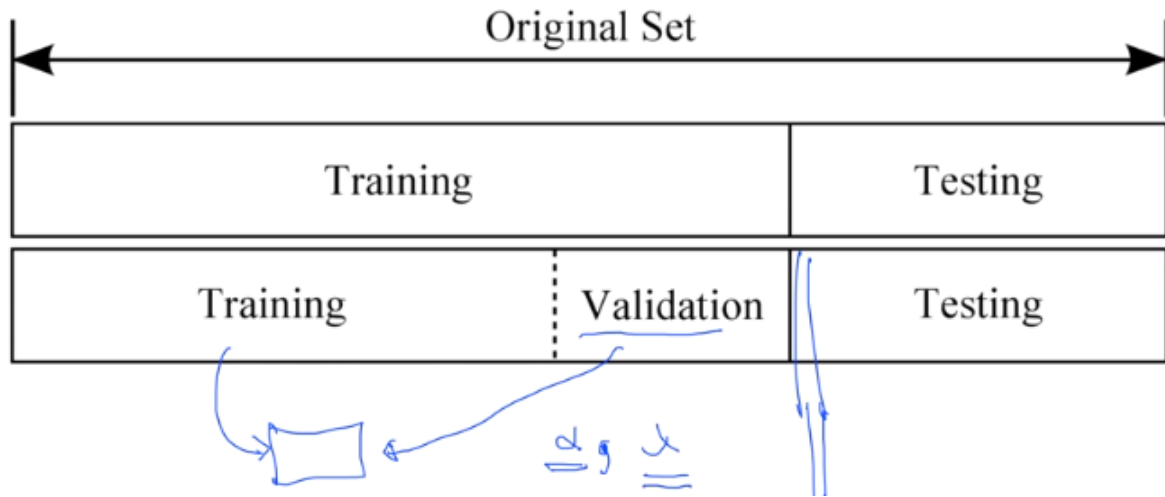
### Training Sets & Test Sets.

	Size	Price
training	2104	400
	1600	330
	2400	369
	1416	232
	3000	540
	1985	300
	1534	315
test set	1427	199
	1380	212
	1494	243



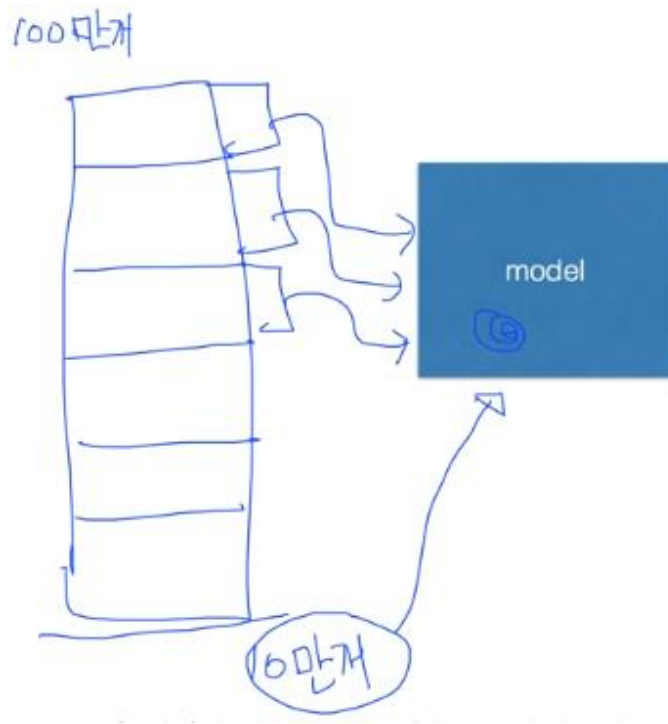
## Training, Validation and Test Sets.

$\alpha$ (Learning Rate) &  $\lambda$ (Regularization Strength)의 값을 조정한다.



## Online Learning.

이미 100만개가 입력되어 있을 때, 추가로 10만개의 데이터만 추가하면 된다.



## MNIST Dataset.



[train-images-idx3-ubyte.gz](#): training set images (9912422 bytes)  
[train-labels-idx1-ubyte.gz](#): training set labels (28881 bytes)  
[t10k-images-idx3-ubyte.gz](#): test set images (1648877 bytes)  
[t10k-labels-idx1-ubyte.gz](#): test set labels (4542 bytes)

## Memo

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

<code>.decode_csv()</code>	<code>//Queue, text decode</code>
<code>.batch()</code>	<code>//Queue, text batch</code>
<code>.Coordinator()</code>	<code>//Queue, Coordinator 생성.</code>
<code>.start_queue_runners()</code>	<code>//Queue, Queue 를 Thread 로 시작.</code>
<code>.request_stop()</code>	<code>//Queue, 중지</code>
<code>.join()</code>	<code>//Queue, 대기</code>
<code>.sigmoid()</code>	<code>//S 자 곡선</code>
<code>.log()</code>	<code>//로그</code>
<code>.cast()</code>	<code>//새로운 자료형</code>
<code>.equal()</code>	<code>//값이 같은지</code>
<code>.softmax()</code>	<code>//softmax</code>
<code>.arg_max()</code>	<code>//arg_max</code>
<code>.one_hot()</code>	<code>//one_hot</code>
<code>.reshape()</code>	<code>//reshape</code>
<code>.softmax_cross_entropy_with_logits()</code>	<code>//softmax_cross_entropy_with_logits</code>
<code>.format()</code>	<code>//format</code>
<code>.flatten()</code>	<code>//flatten</code>

`//Lab으로 연결 https://colab.research.google.com/drive/1gaTpEufmhoK2CsEsNyfDDtyynQ\_HRpSu`

`//14폰트, 12폰트, 10폰트`

`//1.        1)        a.        *♠`

`//0.71    1.34`