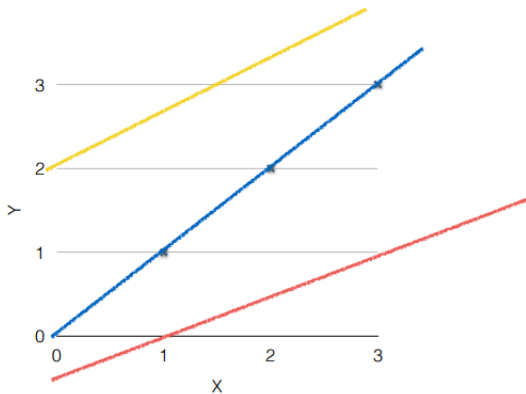


# [ TensorFlow로 선형 회귀 모델 구현 ]

## [ Linear regression(단일변수) ]

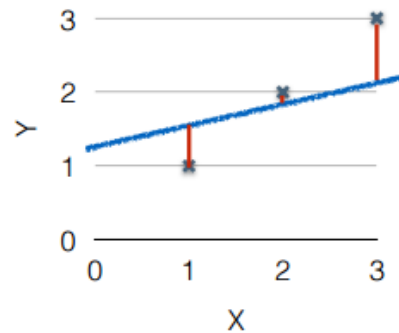
(Linear) Hypothesis :  $H(x) = Wx + b$

Which hypothesis is better?



minimize cost (W, b)

비용(Cost) 또는 손실(loss) : 예측 값 - 실제 값  
 $H(x) - y$



비용(Cost) 또는 손실(loss) 평균

$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2}{3}$$

$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$H(x) = Wx + b$  의 비용(Cost) 함수

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

1) 비용(Cost)/손실(loss) 함수

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

```
cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

2) 비용 최소화를 위한 기법 : 경사하강법

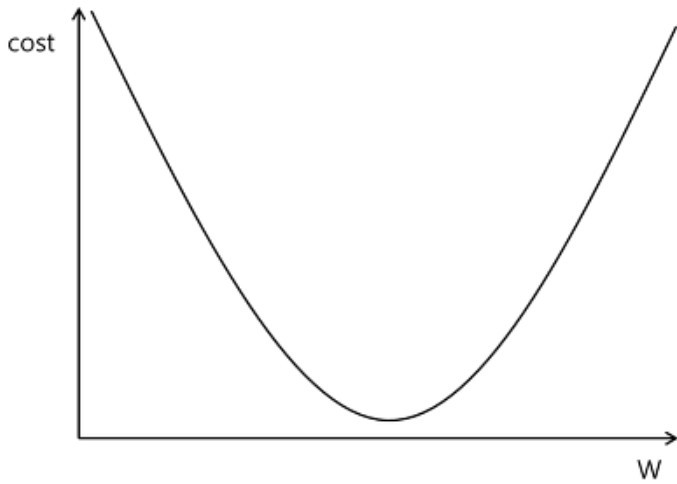
< Tensorflow 포함되어 있는 경사하강법 함수를 이용해 비용 최소화/최적화를 수행 >

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

```
train_op = optimizer.minimize(cost)
```

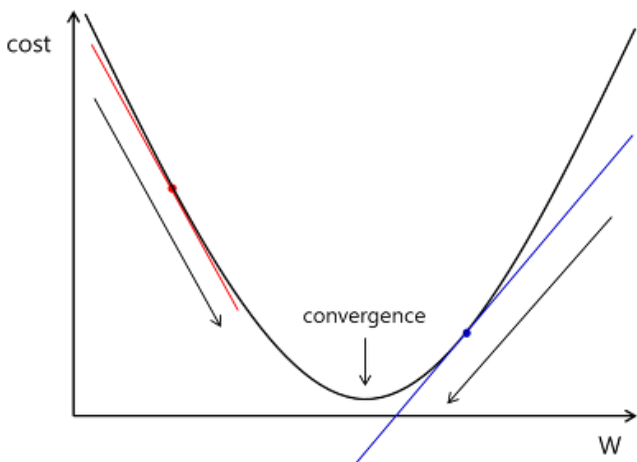
## [ 경사하강법(Gradient descent algorithm) 이란? ]

What  $\text{cost}(W)$  looks like?



$\text{cost}$ 를 줄이기 위해 변경되는  $W$ 의 파라미터의 상관관계를 그래프로 나타낸다면,  $\text{cost}$ 의 값이 최소가 되는 것은  $W$ 의 값이 가운데로 수렴하게 된다는 것.  
(편의상 추가적으로 더하는 항인 바이어스의 값은 제외)

기울기가 0인 곳을 찾는 것이 경사하강법



## 예) What cost(W) looks like?

ts\_09\_1\_cost.py

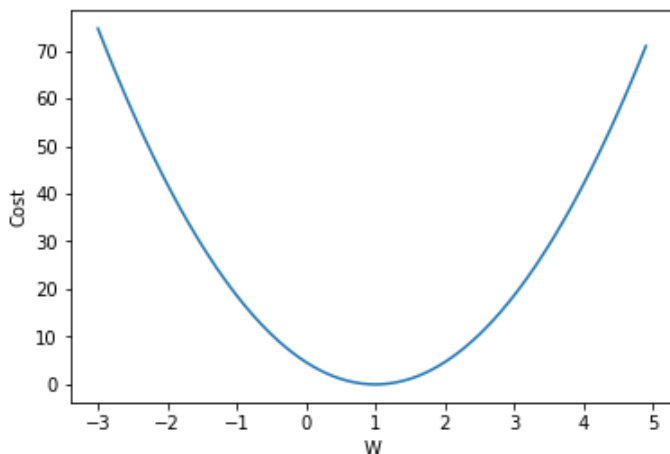
W값 변화에 따른 Cost 값을 시각화해서 보기

```
import tensorflow as tf
import matplotlib.pyplot as plt
X = [1, 2, 3]
Y = [1, 2, 3]
W = tf.placeholder(tf.float32)
# Our hypothesis for linear model X * W
hypothesis = X * W
# cost/Loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())
# Variables for plotting cost function
W_val = []
cost_val = []

for i in range(-30, 50):
    feed_W = i * 0.1
    curr_cost, curr_W = sess.run([cost, W], feed_dict={W: feed_W})
    W_val.append(curr_W)
    cost_val.append(curr_cost)

# Show the cost function
plt.plot(W_val, cost_val)
plt.ylabel("Cost")
plt.xlabel("W")
plt.show()
sess.close()
```

출력 결과



$X = [1, 2, 3]$ ,  $Y = [1, 2, 3]$  이므로  $W$ 가 1일 때 비용이 0으로 수렴함. ( $Y = WX$ ,  $W=1$ )

경사하강법으로 Cost 기울기가 0으로 수렴하는 W값 찾기

훈련 데이터  $X = [1, 2, 3]$ ,  $Y = [1, 2, 3]$ 이므로 비용이 0으로 수렴하는  $W=1$ 을 찾아야 함. ( $Y = WX$ ,  $W=1$ )

ts\_09\_2\_cost.py

```
import tensorflow as tf
# tf Graph Input
X = [1, 2, 3]
Y = [1, 2, 3]
# Set wrong model weights
W = tf.Variable(5.0)
# Linear model
hypothesis = X * W
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize: Gradient Descent Magic
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
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())

for step in range(100):
    print(step, sess.run(W))
    sess.run(train) # cost의 기울기가 0이 되는 지점을 경사하강법으로 찾음.

sess.close()
```

(출력 결과)

```
0 5.0
1 1.2666664
2 1.0177778
3 1.0011852
4 1.000079
5 1.0000052
6 1.0000004
7 1.0
8 1.0
9 1.0
.. 중략
98 1.0
99 1.0
```

W의 초기값을 랜덤한 값으로 바꾸어서 실행해도 최종 1.0으로 잘 찾아내는지 실습해봅니다.

ts\_09\_Linear Regression.py

```
# X 와 Y 의 상관관계를 분석하는 기초적인 선형 회귀 모델을 만들고 실행해봅니다.
import tensorflow as tf

x_data = [1, 2, 3]
y_data = [1, 2, 3]

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

# name: 나중에 텐서보드등으로 값의 변화를 추적하거나 살펴보기 쉽게 하기 위해
# 이름을 붙여줍니다.
X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")
print(X)
print(Y)

# X 와 Y 의 상관 관계를 분석하기 위한 가설 수식을 작성합니다.
# y = W * x + b
# W 와 X 가 행렬이 아니므로 tf.matmul 이 아니라 기본 곱셈 기호를 사용했습니다.
hypothesis = W * X + b
```

ts\_09\_Linear Regression.py(계속)    경사 하강법을 이용하여 비용을 최소화하는 train\_op 생성

```
# 손실 함수를 작성합니다.
# mean(h - Y)^2 : 예측값과 실제값의 거리를 비용(손실) 함수로 정합니다.
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# 텐서플로우에 기본적으로 포함되어 있는 함수를 이용해 경사 하강법 최적화를 수행합니다.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
# 비용을 최소화 하는 것이 최종 목표
train_op = optimizer.minimize(cost)
```

ts\_09\_Linear Regression.py(계속)    train\_op 100번 실행

```
# 세션을 생성하고 초기화합니다.
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # 최적화를 100번 수행합니다.
    for step in range(100):
        # sess.run 을 통해 train_op 와 cost 그래프를 계산합니다.
        # 이 때, 가설 수식에 넣어야 할 실제값을 feed_dict 을 통해 전달합니다.
        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})

        print(step, cost_val, sess.run(W), sess.run(b))

    # 최적화가 완료된 모델에 테스트 값을 넣고 결과가 잘 나오는지 확인해봅니다.
    print("\n=== Test ===")
    print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
    print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X: 2.5}))
```

(출력결과)

```

Tensor("X_4:0", dtype=float32)
Tensor("Y_4:0", dtype=float32)
0 1.9936131 [0.8426495] [0.51892483]
1 0.05821353 [0.78194] [0.47808003]
2 0.033460755 [0.79423064] [0.46968803]
3 0.031608704 [0.79840684] [0.45805818]
.... 중략
99 0.00029564253 [0.98051] [0.04430538]
=== Test ===
X: 5, Y: [4.946855]
X: 2.5, Y: [2.4955802]

```

---

상관관계를 알아내고자 학습한 데이터가  $x\_data = [1, 2, 3]$ ,  $y\_data = [1, 2, 3]$  이므로,  
 $hypothesis(X) = W * X + b$  식에서  $W=1$ ,  $b=0$  인 것을 알 수 있습니다.

ts\_09\_Linear Regression.py 프로그램에서 학습한 데이터로 입력하지 않았던,  
 $X : 5$  의 결과로 4.946855,  $X: 2.5$  2.4955802 의 결과로 비교적 정확히 예측한 것을 확인할 수 있습니다.

## [ (Multi-variable) linear regression ]

$$H(x_1, x_2, x_3) = (x_1 w_1 + b_1) + (x_2 w_2 + b_2) + (x_3 w_3 + b_3)$$

$$H(x_1, x_2, x_3) = (x_1 w_1 \quad ) + (x_2 w_2 \quad ) + (x_3 w_3 \quad )$$

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1 w_1 + x_2 w_2 + x_3 w_3) \quad H(X) = XW$$

행렬곱 예)

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$

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

[5, 3]

[3, 1]

[5, 1]

ts\_10\_1\_multiVariable\_Linear\_Regression.py

```

import tensorflow as tf

x_data = [[73., 80., 75.],
           [93., 88., 93.],
           [89., 91., 90.],
           [96., 98., 100.],
           [73., 66., 70.]]
y_data = [[152.],
           [185.],
           [180.],
           [196.],
           [142.]]

# 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())

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)

```

(출력결과) ... 중략

2000 Cost: 0.98337793

Prediction:

[[151.94438]

[184.51367]

[181.09804]

[194.60779]

[143.23828]

\*\*\*\*\*

y\_data 와 비슷하게 예측



## [ TensorFlow로 분류 모델 구현 ]

Spam Detection: Spam or Ham

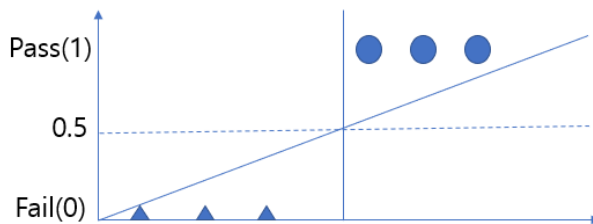
Facebook feed: show or hide

Credit Card Fraudulent Transaction detection: legitimate/fraud

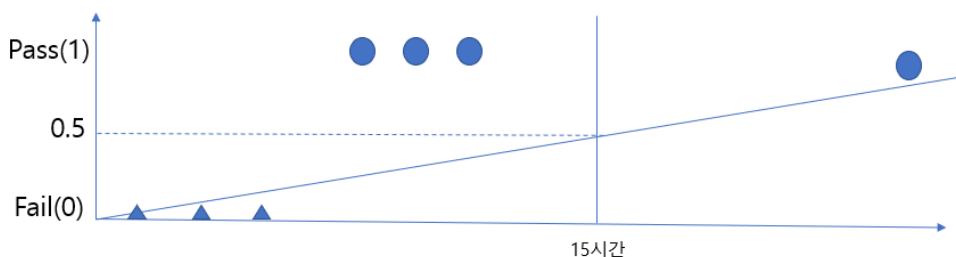
### Logistic Classification - sigmoid 함수, 새로운 cost 함수

< 선형 회귀로 분류 문제를 해결할 때의 문제점 >

예) 5시간 공부한 학생 - Pass (1)  
 7시간 공부한 학생 - Pass (1)  
 8시간 공부한 학생 - Pass (1)  
 1시간 공부한 학생 - Fail (0)  
 2시간 공부한 학생 - Fail (0)  
 4시간 공부한 학생 - Fail (0)

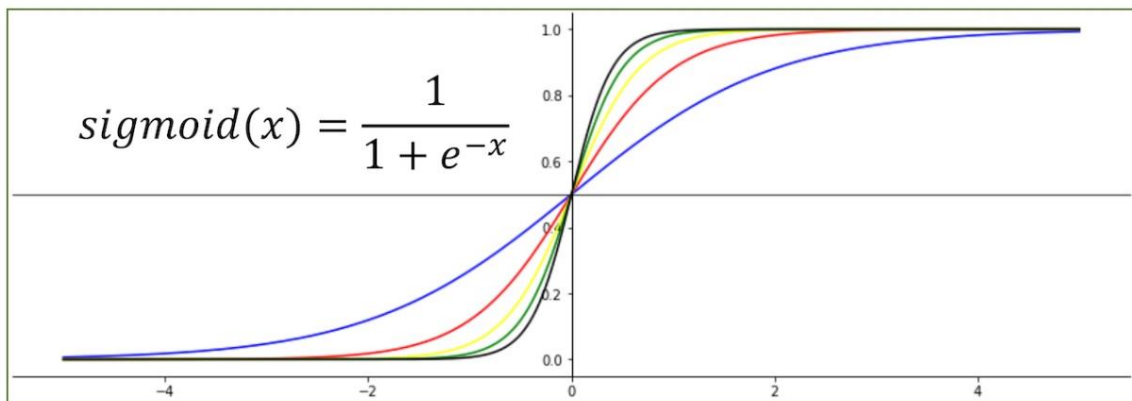


만약, 50 시간 공부한 학생이 있으면 ?



1) binary classification 의 결과는 0과 1이 필요한데,

$H(x) = Wx + b$  는 1보다 훨씬 큰 수가 나타날 수 있음.  
 $H(x)$  를 0~1 사이의 값으로 변경해주는 함수.  $\rightarrow$  sigmoid

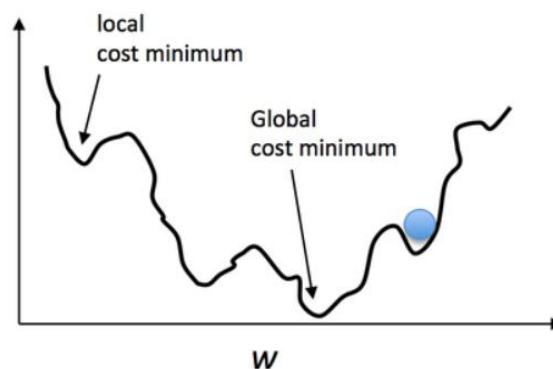


그래서,  $H(x)$ 를 sigmoid 함수로 한번 더 계산. (  $H(x)$ 를 0~1 사이의 값으로 변경 )

$$\frac{1}{1 + e^{-H(x)}}$$

sigmoid 함수의 특징은,  $x$ 값이 아무리 크거나 작더라도 그 결과값은 최대가 1, 최소가 0이 됩니다.

2) sigmoid 함수를 통과한 값의 Cost 는 다음과 같이 구불구불한 모습으로 경사 하강법으로 최소 Cost 값을 찾을 때 문제가 발생할 수 있습니다. gradient descent algorithm 을 적용해도 우리가 필요로 하는 최종 값인 Global cost minimum 을 구하지 않고, local cost minimum 까지만 구할 수 있습니다.



이것을 해결하기 위해서 cost 함수를 아래와 같이 바꾸어야 합니다. 새로운 cost 함수

$$c(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

Tensorflow에서 sigmoid 함수와 새로운 cost 함수 적용 예)

```
hypothesis = tf.sigmoid(tf.matmul(X, W) + b) # sigmoid
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis)) # cost function
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost) # 경사하강법으로 비용 최소화
```

## [이진 분류 예]

data-02-diabetes.csv 파일에 다음과 같은 건강에 대한 특성 정보가 있습니다.

마지막 컬럼이 당뇨병이 존재하는 지 안하는 지 분류 정보가 있습니다.

이 데이터로 학습한 후, 새로운 데이터로 예측을 해봅니다.

이진 분류 지도 학습 알고리즘으로 sigmoid 함수와 새로운 cost 함수를 활용합니다.

-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

ts\_10\_2\_BiClassification\_logistic\_Regressin.py

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

xy = np.loadtxt('data-02-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 8])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([8, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(-tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
tf.log(1 - hypothesis))

train = tf.train.GradientDescentOptimizer(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))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0:
            print(step, cost_val)

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

```

(출력 결과) (759, 8) (759, 1)  
 0 0.9333563  
 200 0.83229107  
 400 0.78616315   중략....  
 [0.]   중략....  
 [1.]  
 [1.]  
 Accuracy: 0.77602106

## [다중 분류 예] softmax 함수 , softmax\_cross\_entropy\_with\_logits

softmax 함수를 이용하여 0~1사이의 값으로 p(확률)값으로 나오게 바꿉니다.

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

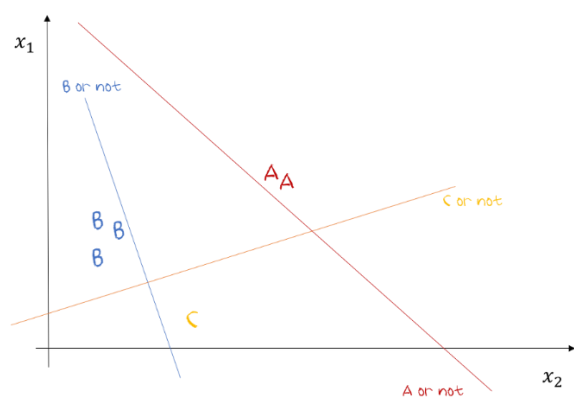
```
#Test & one-hot encoding
```

```
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
```

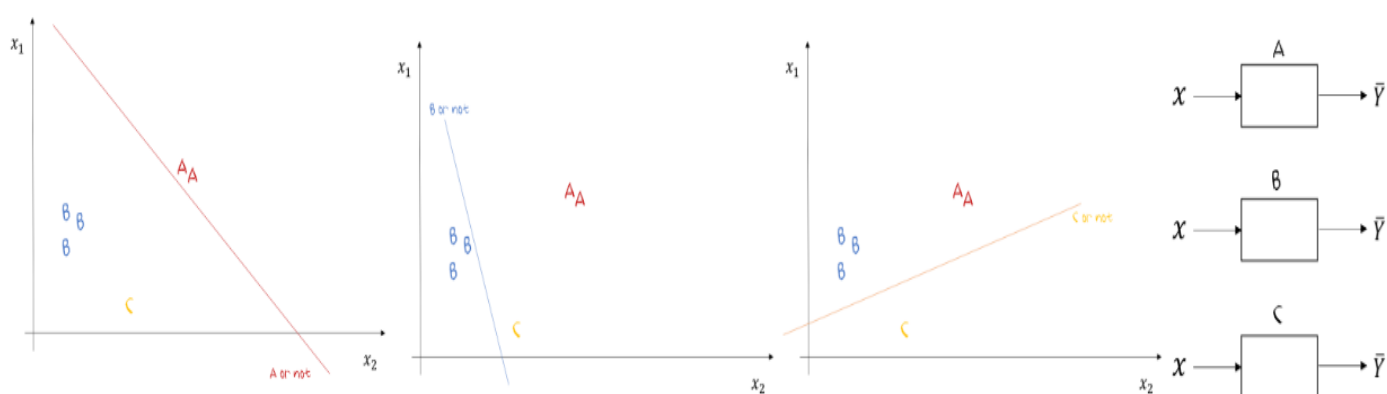
```
print(a, sess.run(tf.argmax(a, 1))) # 가장 큰 값을 가진 것을 반환
```

### < softmax 함수가 필요한 이유 설명 >

아래와 같은 ABC를 분류한다고 생각해봅시다.



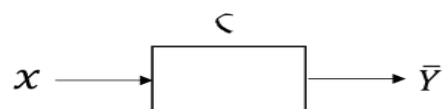
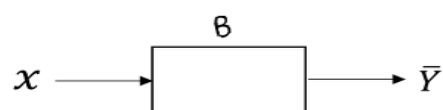
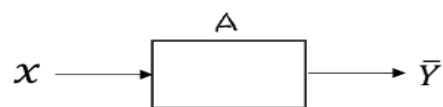
3개의 Binary Classification을 가지고 구현이 가능



$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1x_1 + w_2x_2 + w_3x_3]$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1x_1 + w_2x_2 + w_3x_3]$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1x_1 + w_2x_2 + w_3x_3]$$

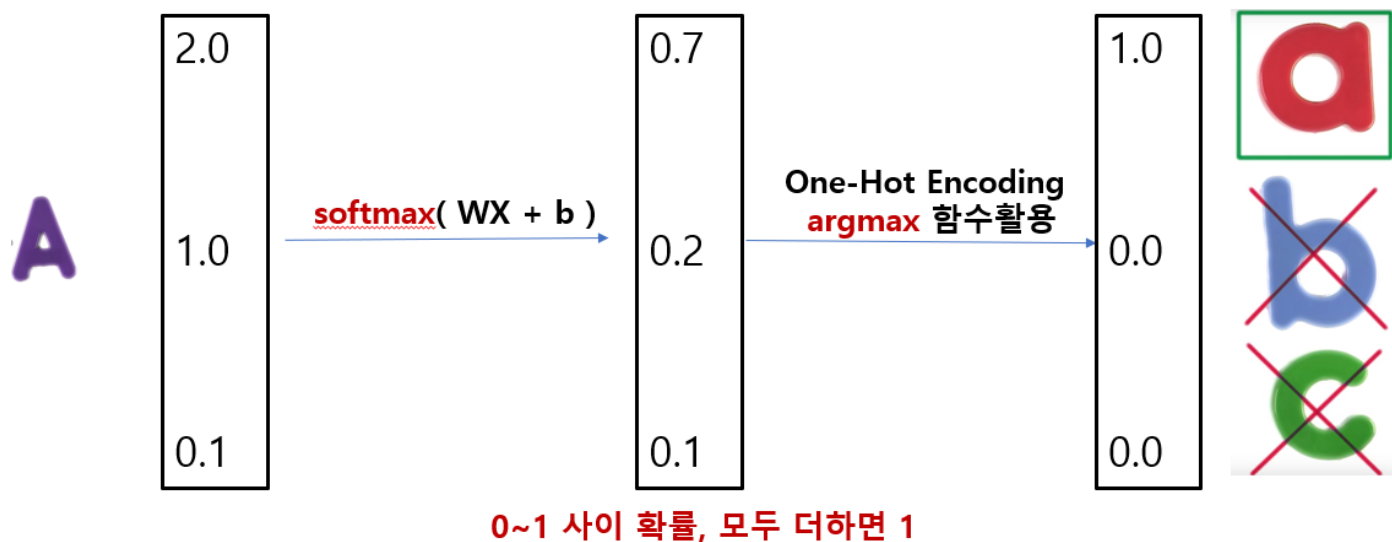


$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix}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

하나의 값이 아니라 A,B,C의 벡터 값으로 나오게 됩니다.

softmax 함수를 이용하여 0~1사이의 값으로 p(확률)값으로 나오게 바꿀 수 있습니다.

**softmax** 함수를 통과시킨 결과 값이 가장 큰 값이 결과 .



새로운 Cost 함수 필요. - **Cross-entropy** cost function

다음과 같이 1번 식 또는 2번식으로 사용 가능.

```
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)
```

ts\_10\_3\_MultiClassification.py

```
import tensorflow as tf

x_data = [[1, 2, 1, 1], # 특성 데이터
          [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, 3종류의 결과분류
y_data = [[0, 0, 1], # beginner 회원
          [0, 0, 1], # beginner 회원
          [0, 0, 1], # beginner 회원
          [0, 1, 0], # VIP 회원
          [0, 1, 0], # VIP 회원
          [0, 1, 0], # VIP 회원
          [1, 0, 0], # VVIP 회원
          [1, 0, 0]] # VVIP 회원

X = tf.placeholder("float", [None, 4])
Y = tf.placeholder("float", [None, 3])
nb_classes = 3
```

ts\_10\_3\_MultiClassification.py (계속)

```
W = tf.Variable(tf.random_normal([4, 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)
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))

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):
        sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}))

    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)))
```

(출력 결과)

0 3.009141

200 0.5865452 ... 중략

1800 0.17551824

2000 0.16281393

-----

[[1.7313376e-02 9.8267800e-01 8.7036642e-06]] [1]

-----

[[0.75537467 0.22770871 0.0169166 ]] [0]

-----



[[1.5998591e-08 3.8091152e-04 9.9961901e-01]] [2]



























































-----  
[[1.7313376e-02 9.8267800e-01 8.7036642e-06]

[7.5537467e-01 2.2770871e-01 1.6916601e-02]

[1.5998589e-08 3.8091152e-04 9.9961901e-01]] [1 0 2]

(동물의 16가지 특성 정보를 이용하여 동물 중 분류하기 예제)

다음과 같이 6 종류의 동물 종류가 있다.

Birds	Insect	Fishes	Amphibians	Reptiles	Mammals
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					
					

data-04-zoo.csv 파일에 각 동물의 16가지 특성 정보가( feathers ,hair, milk 등 ) 다음과 같이 있고, 마지막 컬럼은 해당 동물이 7가지 종류(6가지와 기타 중) 중 몇 번에 속하는 동물 인지의 정보가 있다. (0~5번 중 하나) → One Hot 인코딩이 필요하다.

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

ts\_10\_4\_MultiClassification\_animal.py

```

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('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

nb_classes = 7 # 0 ~ 6

X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1]) # 0 ~ 6 shape[?,1]
Y_one_hot = tf.one_hot(Y, nb_classes) # one hot. shape[?, 1, 7]
print("one_hot", Y_one_hot)
# One-Hot 이후에는 한차원이 더 늘어나므로 원래의 차원대로, ReShape 필요
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes]) #ReShap 후 shape[?, 7]

print("reshape", Y_one_hot)

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)

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)))

```

(출력 결과)

(101, 16) (101, 1)

one\_hot Tensor("one\_hot\_3:0", shape=(?, 1, 7), dtype=float32)

reshape Tensor("Reshape\_3:0", shape=(?, 7), dtype=float32)

Step:	0	Loss: 5.525	Acc: 4.95%
Step:	100	Loss: 0.747	Acc: 74.26%
Step:	200	Loss: 0.432	Acc: 82.18%
Step:	300	Loss: 0.301	Acc: 92.08%
Step:	400	Loss: 0.231	Acc: 94.06%
Step:	500	Loss: 0.188	Acc: 97.03%
Step:	600	Loss: 0.159	Acc: 99.01%
Step:	700	Loss: 0.138	Acc: 99.01% .. 중략
Step:	1900	Loss: 0.055	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 .. 중략

[True] Prediction: 0 True Y: 0

[True] Prediction: 1 True Y: 1

[True] Prediction: 1 True Y: 1