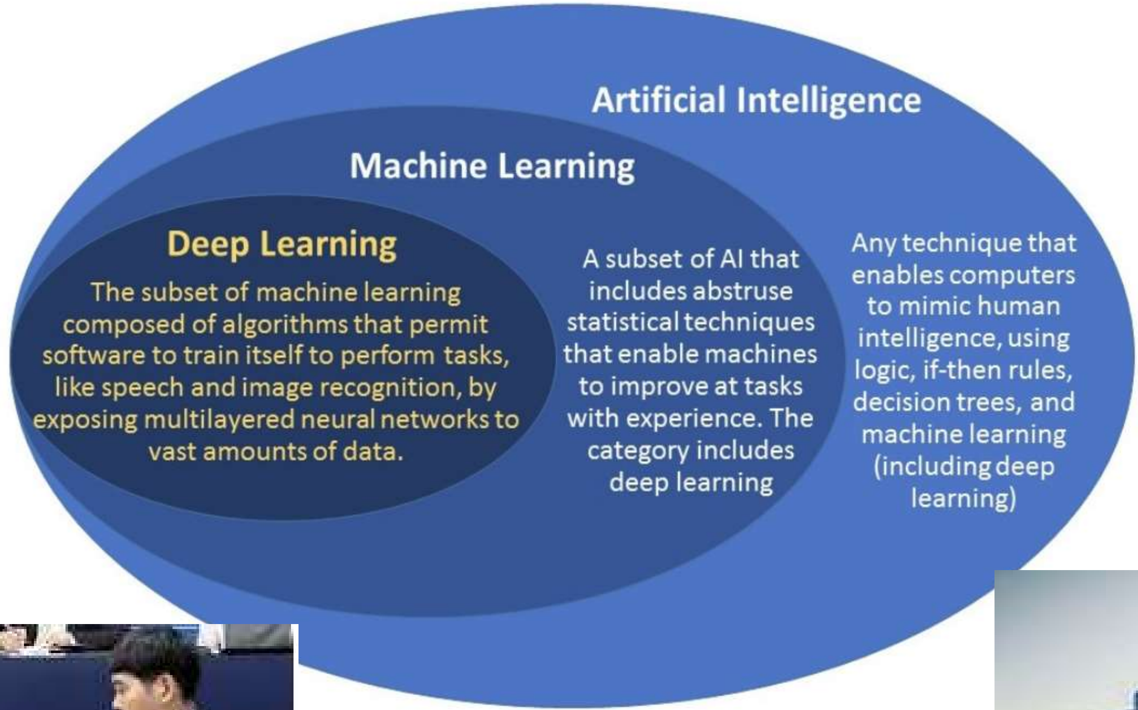
Deep Learning #1

29 Dec 2020

자율주행시스템 개발팀
신 주 석

Why we learn the Machine Learning?



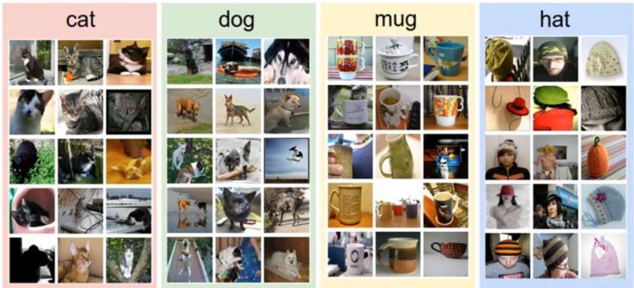
Goals

- **Understand basic ML algorithms**
  - **Linear regression, Logistic regression**
  - **Neural Network, CNN, RNN**
  - ~~**Boosting**~~
- **Solve our problems using ML (DNN) algorithm as black-box**
- ~~**Object detection and classification using Boosting algorithm library**~~

- **Several types of learning algorithm**
  - **Supervised Learning**
    - Training (Learning) with labeled data
  - **Unsupervised Learning**
  - **Reinforcement learning**
  - **Recommender systems**

- **Probably the most Common problem type in ML**
  - **Image labeling**: learning from tagged images
  - **E-mail spam filter**: learning from labeled (spam or ham) email
  - **Predicting exam score**: learning from previous exam score and time spent

- **Training Data Set**



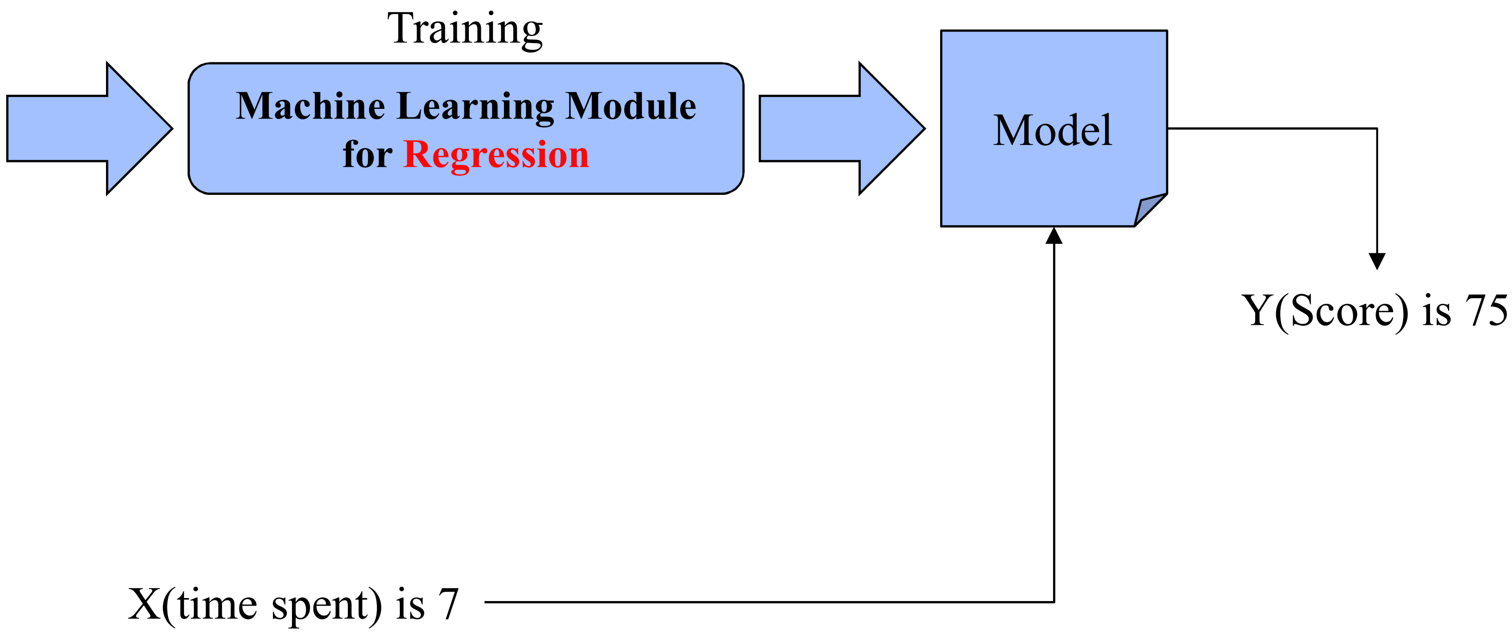
- **Type of Supervised learning**
  - **Regression**
    - » Predicts final exam score based on time spent
  - **Binary Classification**
    - » Pass/fail based on time spent
  - **Multi-label Classification**
    - » Grade (A, B, C, D and F) based on time spent

Regression

- Problem: Predicts final exam score based on time spent
  - Make (obtain) Training Data Set
  - Training using training data set
  - Estimates the final exam score

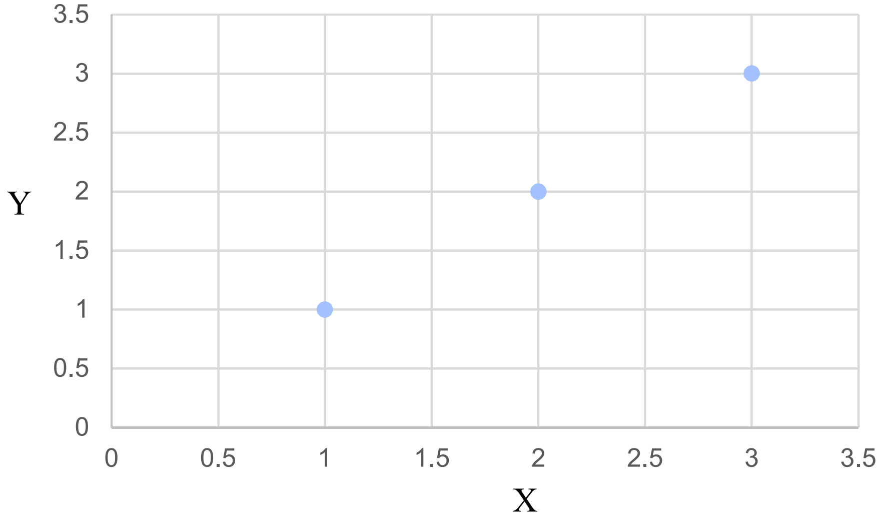
X(hours)	Y(score)
10	92
9	88
3	52
2	40
1	30

Training Data Set



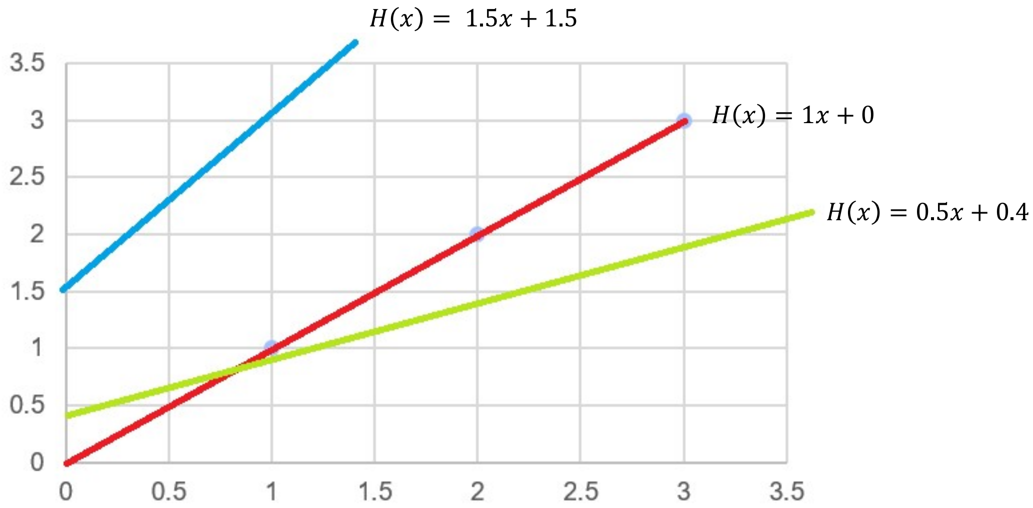
◆ Data 단순화

X	Y
1	1
2	2
3	3



◆ Hypothesis (Linear)

- $\boldsymbol{H(x) = Wx + b}$



Which H(x) is better?

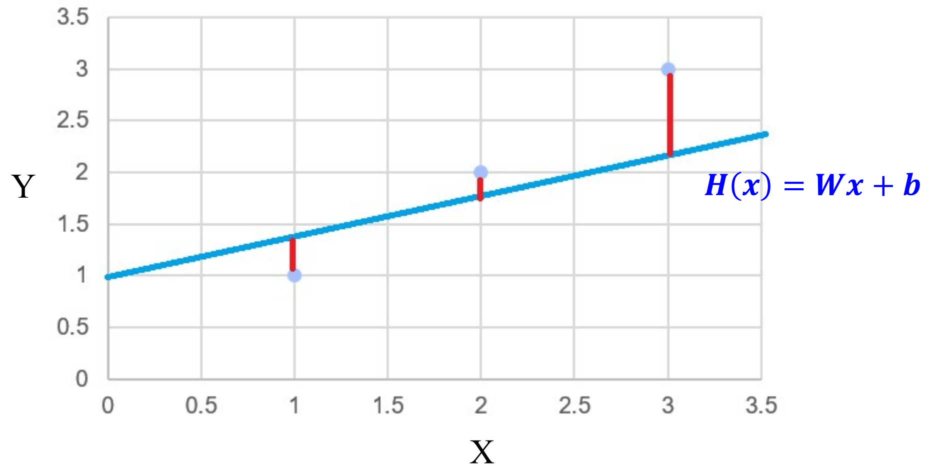
◆ Cost Function (Loss Function): 실제 데이터와 가설 함수가 얼마나 차이가 나는가?

– **How Fit the line to our training data**

» $H(x) - Y$

» $(H(x) - Y)^2$

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



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

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

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

Goal: Minimize Cost

◆ Minimize Cost

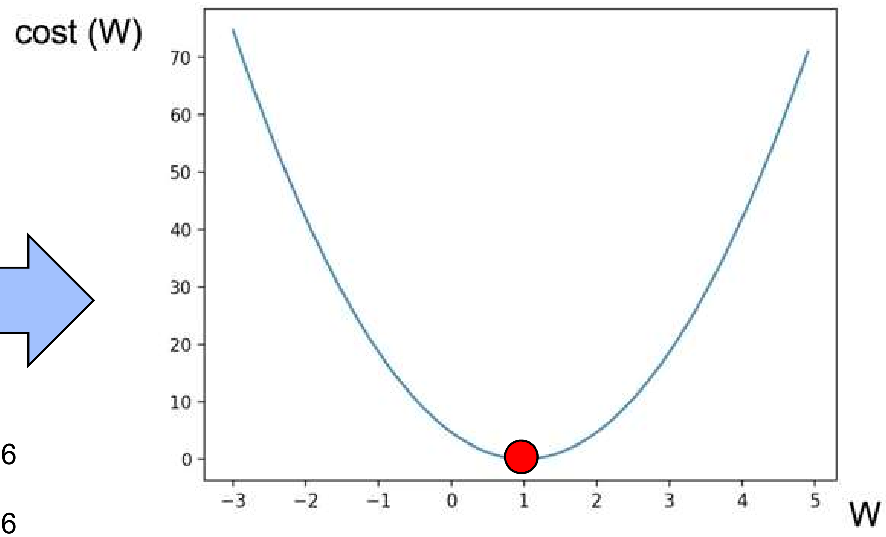
– Simplified Hypothesis

$$H(x) = Wx$$

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

X	Y
1	1
2	2
3	3

W=0, cost = 1/3((0-1)^2+(0-2)^2+(0-3)^2)=4.666
W=1, cost = 0
W=2, cost = 1/3((2-1)^2+(4-2)^2+(6-3)^2)=4.666



e.g.) W가 5일때 시작->w값 조정->cost 계산, 0으로 수렴 할때까지 반복,

◆ How would you find the lowest point(minimized cost)?

– Gradient Descent Algorithm

» Minimize cost function

» Be used many minimization problems

» For a given cost function, cost(W, b), it will find W, b to minimize cost

» It can be applied to more general function: cost(w_1, w_2, w_3, .. w_n, b)

Gradient Descent Algorithm

– Formal Definition

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

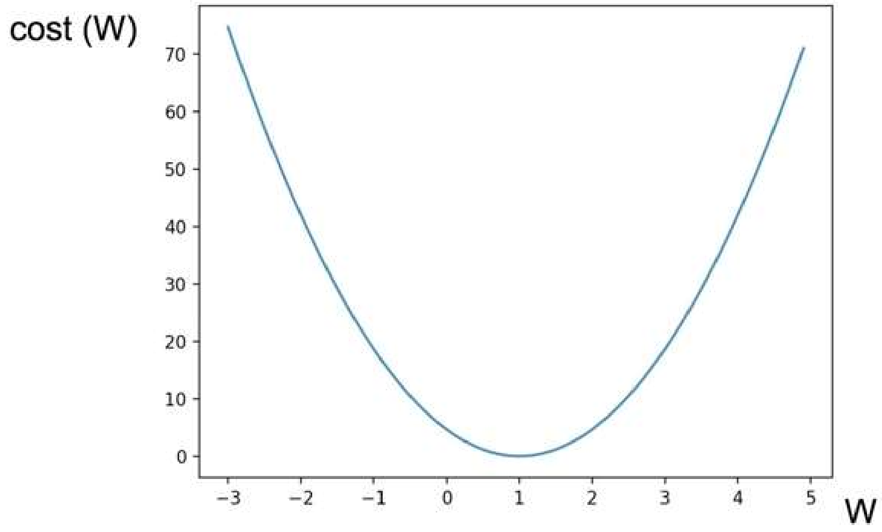
$$cost(W) = \frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

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

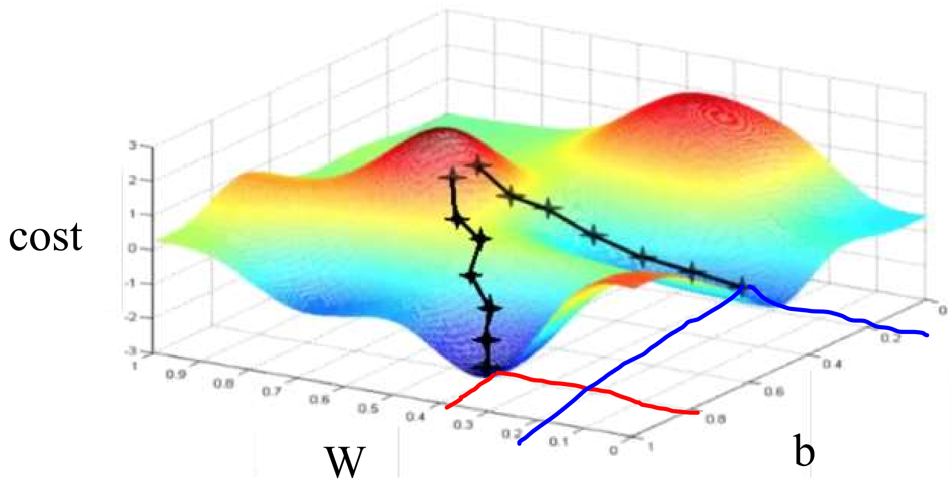
$$W := W - \alpha\frac{\partial}{\partial W}\frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha\frac{1}{2m}\sum_{i=1}^{m}2(Wx^{(i)} - y^{(i)})x^{(i)}$$

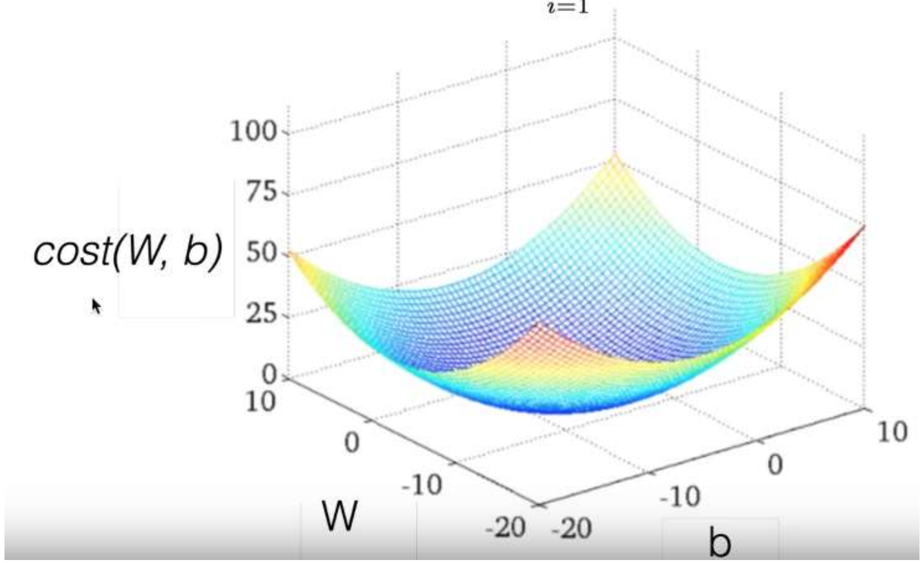
$$\boxed{W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)}}$$



- **Convex Function**



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



Hypothesis => Cost function => gradient Descent algorithm

One-variable vs Multi-variable

Regression using 1 input

X(hours)	Y(score)
10	92
9	88
3	52
2	40
1	30

One-variable

Regression using 3 inputs

X_1(term1)	X_2(term2)	X_3(term3)	Y(final score)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Multi-variable

Hypothesis

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

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

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

- **Hypothesis of Multi-variable using Matrix**

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

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

https://www.mathsisfun.com/algebra/matrix-multiplying.html

- **Hypothesis of Multi-variable using Matrix**

X_1(term1)	X_2(term2)	X_3(term3)	Y(final score)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

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

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

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

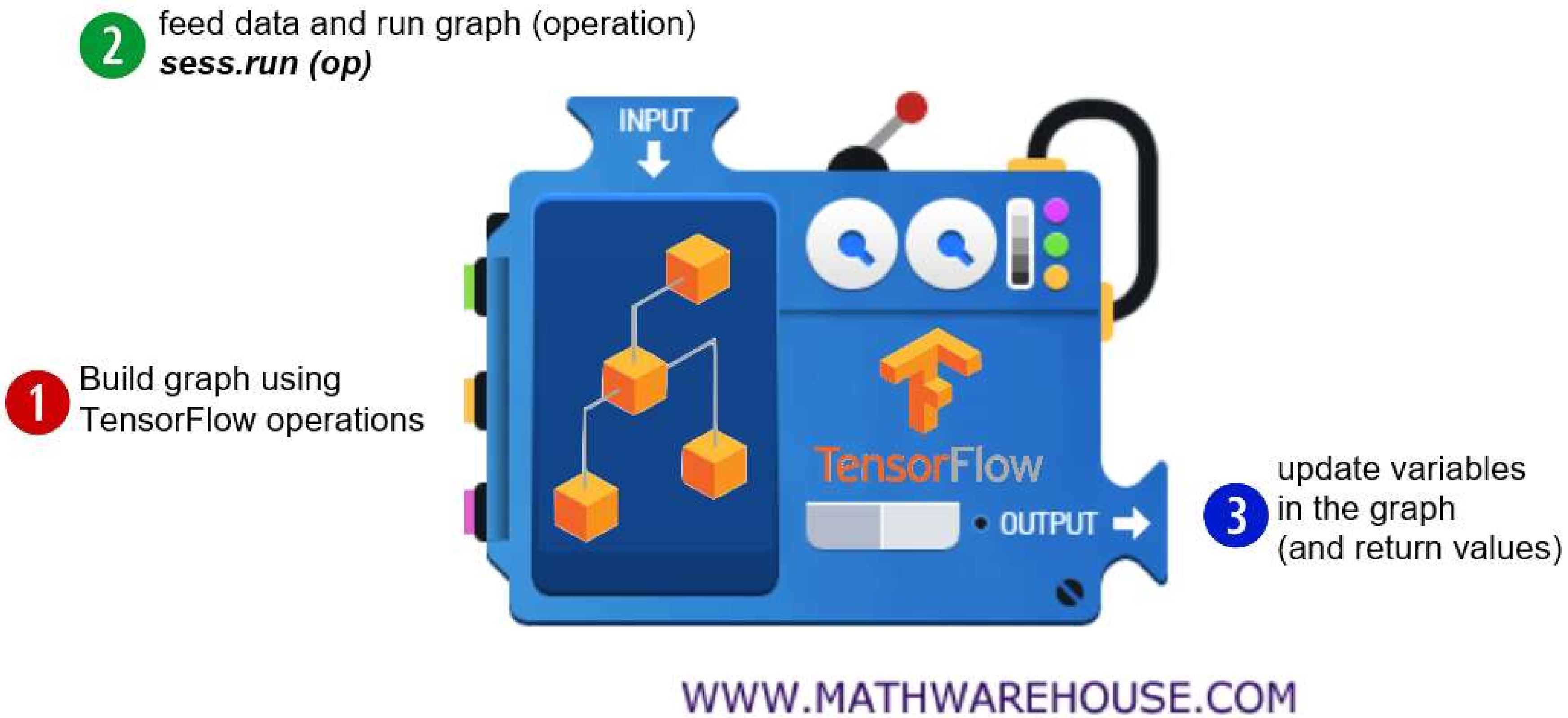
[5 x 3] [3 x 1] [5 x 1]

[n x 3] [3 x 1] [n x 1]

Machine Learning / Deep Learning #1 (실습)

Linear Regression

TensorFlow Mechanics



pip3 install pylint
pip3 install jupyter (optional)
pip3 install tensorflow
pip3 install matplotlib

◆ H

–

◆ Si

–

In [1]:
```python
import tensorflow as tf
import matplotlib.pyplot as plt
```

In [2]:
```python
X = [1, 2, 3]
Y = [1, 2, 3]

W = tf.placeholder(tf.float32)
```

Hypothesis is $H(x) = Wx$

In [3]:
```python
hypothesis = W * X
```

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

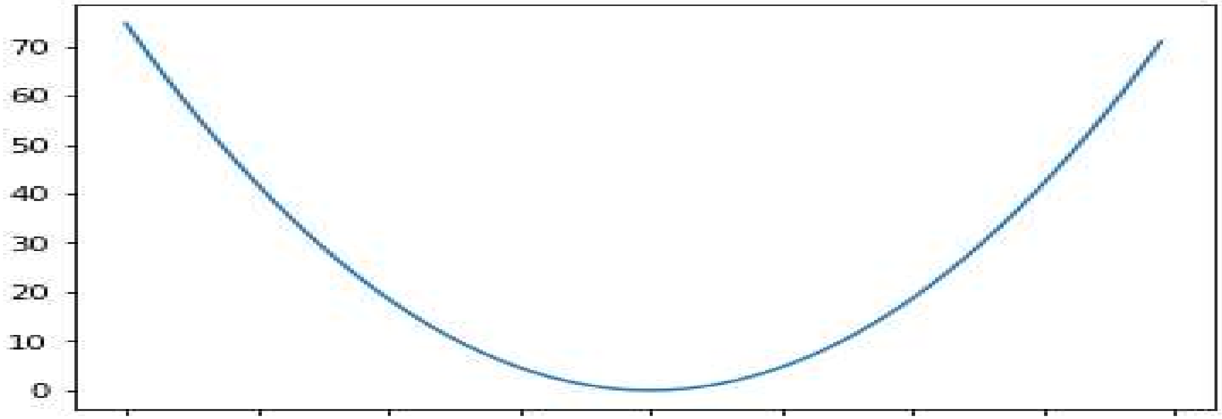
In [4]:
```python
cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

In [5]:
```python
sess = tf.Session()
sess.run(tf.global_variables_initializer())

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)

plt.plot(W_val, cost_val)
plt.show()
```

◆ C

imp

tf.d



1. Op
2. Sel

◆ Gradient Descent

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

In [1]:
```
import tensorflow as tf
```

In [2]:
```
x_data = [1, 2, 3]
y_data = [1, 2, 3]

W = tf.Variable(tf.random_normal([1]), name='weight')
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
```

Hypothesis is $H(x) = Wx$

In [3]:
```
hypothesis = W * X
```

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

In [4]:
```
cost = tf.reduce_mean(tf.square(hypothesis-Y))
```

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

In [5]:
```
learning_rate = 0.1
gradient = tf.reduce_mean((W*X-Y) * X)
gradient_descent = W - learning_rate * gradient
train = W.assign(gradient_descent)
```

In [6]:
```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for step in range(50):
    sess.run(train, feed_dict={X: x_data, Y: y_data})
    print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W))
```

```
0 15.8009 [-0.84008455]
1 4.49448 [ 0.01862156]
2 1.27843 [ 0.47659814]
3 0.363643 [ 0.72085232]
4 0.103436 [ 0.85112125]
5 0.0294219 [ 0.92059797]
6 0.00836888 [ 0.95765227]
7 0.00238048 [ 0.97741455]
8 0.000677111 [ 0.98795444]
9 0.0001926 [ 0.99357569]
10 5.47849e-05 [ 0.99657369]
11 1.55836e-05 [ 0.99817264]
12 4.43245e-06 [ 0.9990254]
13 1.26102e-06 [ 0.99948019]
14 3.58608e-07 [ 0.99972278]
15 1.02069e-07 [ 0.99985212]
16 2.90099e-08 [ 0.99992114]
17 8.27377e-09 [ 0.99995792]
18 2.34392e-09 [ 0.99997759]
19 6.71251e-10 [ 0.99998802]
20 1.89058e-10 [ 0.99999362]
21 5.42724e-11 [ 0.9999966]
22 1.44998e-11 [ 0.99999821]
23 4.24431e-12 [ 0.99999905]
24 1.40806e-12 [ 0.99999946]
25 4.51195e-13 [ 0.9999997]
26 1.29082e-13 [ 0.99999982]
27 9.9476e-14 [ 0.99999988]
28 2.4869e-14 [ 0.99999994]
29 0.0 [ 1.]
30 0.0 [ 1.]
31 0.0 [ 1.]
32 0.0 [ 1.]
33 0.0 [ 1.]
34 0.0 [ 1.]
35 0.0 [ 1.]
36 0.0 [ 1.]
37 0.0 [ 1.]
38 0.0 [ 1.]
39 0.0 [ 1.]
40 0.0 [ 1.]
41 0.0 [ 1.]
42 0.0 [ 1.]
43 0.0 [ 1.]
44 0.0 [ 1.]
45 0.0 [ 1.]
46 0.0 [ 1.]
47 0.0 [ 1.]
48 0.0 [ 1.]
49 0.0 [ 1.]
```

```
In [1]: import tensorflow as tf

        W = tf.Variable(tf.random_normal([1]), name='weight')
        b = tf.Variable(tf.random_normal([1]), name='bias')
        X = tf.placeholder(tf.float32, shape=[None])
        Y = tf.placeholder(tf.float32, shape=[None])
```

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

```
In [2]: Hypothesis = W * X + b
```

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

```
In [3]: cost = tf.reduce_mean(tf.square(Hypothesis - Y))
```

Minimize cost using GradientDescentOptimizer library

```
In [7]: opt = tf.train.GradientDescentOptimizer(learning_rate=0.01)
        train = opt.minimize(cost)
```

```
In [11]: sess = tf.Session()
         sess.run(tf.global_variables_initializer())

         for i in range(2001):
             cost_val, weight_val, bias_val, _ = sess.run([cost, W, b, train],
                                                          feed_dict={X: [1, 2, 3, 4, 5],
                                                                     Y: [2.5, 4.5, 6.5, 8.5, 10.5]})

             if i % 100 == 0:
                 print(i, cost_val, weight_val, bias_val)
```

```
0 132.87 [-0.43006706] [-0.3104496]
100 0.00151496 [ 2.02518439] [ 0.40907657]
200 0.000769555 [ 2.01794934] [ 0.43519729]
300 0.000390905 [ 2.01279259] [ 0.45381415]
400 0.000198568 [ 2.0091176] [ 0.46708274]
500 0.000100866 [ 2.00649834] [ 0.47653922]
600 5.1235e-05 [ 2.00463152] [ 0.48327905]
700 2.60266e-05 [ 2.00330114] [ 0.48808247]
800 1.32214e-05 [ 2.00235271] [ 0.49150586]
900 6.71871e-06 [ 2.00167727] [ 0.49394539]
1000 3.41341e-06 [ 2.00119567] [ 0.49568424]
1100 1.73471e-06 [ 2.00085258] [ 0.49692339]
1200 8.81556e-07 [ 2.00060749] [ 0.49780649]
1300 4.48418e-07 [ 2.00043344] [ 0.49843609]
1400 2.27995e-07 [ 2.00030899] [ 0.4988848]
1500 1.15932e-07 [ 2.00022054] [ 0.49920467]
1600 5.89449e-08 [ 2.00015736] [ 0.49943283]
1700 3.00276e-08 [ 2.00011182] [ 0.49959561]
1800 1.5286e-08 [ 2.00008035] [ 0.49971116]
1900 7.80088e-09 [ 2.00005674] [ 0.49979386]
2000 3.89874e-09 [ 2.00004053] [ 0.49985388]
```

```
In [13]: print (sess.run(Hypothesis, feed_dict={X: [5, 6, 7]}))

[ 10.50005627  12.50009727  14.50013828]
```

Multi-Variable Linear Regression

◆ Using Matrix multiplication & GradientDescentOptimizer

X_1(term1)	X_2(term2)	X_3(term3)	Y(final score)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

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

In [21]:
```
import tensorflow as tf

x1 = [73, 93, 89, 96, 73]
x2 = [80, 88, 91, 98, 66]
x3 = [75, 93, 90, 100, 70]
y = [152, 185, 180, 196, 142]

X1 = tf.placeholder(tf.float32)
X2 = tf.placeholder(tf.float32)
X3 = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

w1 = tf.Variable(tf.random_normal([1]), name='weight1')
w2 = tf.Variable(tf.random_normal([1]), name='weight2')
w3 = tf.Variable(tf.random_normal([1]), name='weight3')
b = tf.Variable(tf.random_normal([1]), name='bias')
```

Hypothesis is $H(x_1, x_2, x_3) = x_1w_1 + x_2w_2 + x_3w_3 + b$

In [22]:
```
hypothesis = X1 * w1 + X2 *w2 + X3 * w3 + b
```

cost function is $cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}) - y^{(i)})^2$

In [23]:
```
cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

Using GradientDescentOptimizer

In [30]:
```
opt = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = opt.minimize(cost)
```

In [31]:
```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

In [32]:
```
for i in range(3001):
    cost_val, hypothesis_val, _ = sess.run([cost, hypothesis, train], feed_dict={X1: x1, X2: x2, X3: x3, Y: y})

    if i % 100 == 0:
        print(i, "Cost: ", cost_val, "\nHypothesis: \n", hypothesis_val)
```

```
2500 Cost:  1.45464
Hypothesis:
 [ 151.61050415  184.3903656   180.60047913  197.66853333  140.10124207]
2600 Cost:  1.42781
Hypothesis:
 [ 151.58720398  184.4067688   180.59388733  197.65943909  140.12641907]
2700 Cost:  1.40222
Hypothesis:
 [ 151.5645752   184.42272949  180.58752441  197.65054321  140.15100098]
2800 Cost:  1.37778
Hypothesis:
 [ 151.54258728  184.43823242  180.58132935  197.64175415  140.17495728]
2900 Cost:  1.35445
Hypothesis:
 [ 151.52122498  184.45327759  180.57531738  197.63314819  140.198349  ]
3000 Cost:  1.33215
Hypothesis:
 [ 151.50050354  184.46794128  180.5695343   197.62467957  140.22117615]
```

◆ Using Matrix multiplication & GradientDescentOptimizer

X_1(term1)	X_2(term2)	X_3(term3)	Y(final score)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

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

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

In [2]:
```
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]]

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 is $H(X) = XW + b$

In [3]:
```
hypothesis = tf.matmul(X, W) + b
```

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

In [4]:
```
cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

In [6]:
```
opt = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = opt.minimize(cost)
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(3001):
    cost_val, hypothesis_val, _ = sess.run([cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})

    if i % 100 == 0:
        print(i, "Cost: ", cost_val, "\nHypothesis: \n", hypothesis_val)
```

```
2900 Cost:  1.36345
Hypothesis:
 [[ 153.43000793]
 [ 183.3924408 ]
 [ 181.1622467 ]
 [ 195.97227478]
 [ 141.08538818]]
3000 Cost:  1.30317
Hypothesis:
 [[ 153.38841248]
 [ 183.42080688]
 [ 181.14932251]
 [ 195.96443176]
 [ 141.12130737]]
```

In [8]:
```
print("Predicts final score: \n", sess.run(hypothesis, feed_dict={X: [[100, 70, 80], [60, 70, 80], [90, 100, 100]]}))
```

```
Predicts final score:
 [[ 172.3429718 ]
 [ 134.88110352]
 [ 191.89489746]]
```

Multi-Variable Linear Regression

Predict Final Score (read file)

– **Score_mlr03.txt** (http://college.cengage.com/mathematics/brase/understandable_statistics/7e/students/datasets/mlr/frames/frame.html)

Score_mlr03.txt

```
#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
69  74  77  149
47  56  60  115
87  79  90  175
79  70  88  164
69  70  73  141
70  65  74  141
93  95  91  184
79  80  73  152
70  73  78  148
93  89  96  192
78  75  68  147
81  90  93  183
88  92  86  177
78  83  77  159
82  86  90  177
86  82  89  175
78  83  85  175
76  83  71  149
96  93  95  192
```

```python
import tensorflow as tf
import numpy as np
```

```python
data = np.loadtxt('score_mlr03.txt', unpack=False, dtype='float32')

train_data_x = data[0:-5, 0:-1]
train_data_y = data[0:-5, [-1]]

test_data_x = data[-5:, 0:-1]
test_data_y = data[-5:, [-1]]
#print(test_data_y)
```

```python
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 = tf.matmul(X, W) + b
cost = tf.reduce_mean(tf.square(hypothesis - Y))

opt = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = opt.minimize(cost)
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(10000):
    cost_val, hypothesis_val, _ = sess.run([cost, hypothesis, train], feed_dict={X: train_data_x, Y: train_data

    if i % 1000 == 0:
        print(i, "Cost: ", cost_val)

print("Predicts final score: \n", sess.run(hypothesis, feed_dict={X: test_data_x}))
```

```
0 Cost:  1378.34
1000 Cost:  16.6156
2000 Cost:  10.7944
3000 Cost:  7.76518
4000 Cost:  6.15109
5000 Cost:  5.27372
6000 Cost:  4.78909
7000 Cost:  4.51798
8000 Cost:  4.36484
9000 Cost:  4.27768
Predicts final score:
 [[ 175.58564758]
 [ 173.81864929]
 [ 167.02781677]
 [ 151.144104  ]
 [ 190.10917664]]
```

Predict Selling price of houses

– **Sell_house.txt** (http://people.sc.fsu.edu/~jburkardt/datasets/regression/x26.txt)

sell_house.txt

index	X1	X2	X3	X4								
1	4.9176	1.0	3.4720	0.998	1.0	7	4	42	3	1	0	25.9
2	5.0208	1.0	3.5310	1.500	2.0	7	4	62	1	1	0	29.5
3	4.5429	1.0	2.2750	1.175	1.0	6	3	40	2	1	0	27.9
4	4.5573	1.0	4.0500	1.232	1.0	6	3	54	4	1	0	25.9
5	5.0597	1.0	4.4550	1.121	1.0	6	3	42	3	1	0	29.9
6	3.8910	1.0	4.4550	0.988	1.0	6	3	56	2	1	0	29.9
7	5.8980	1.0	5.8500	1.240	1.0	7	3	51	2	1	1	30.9
8	5.6039	1.0	9.5200	1.501	0.0	6	3	32	1	1	0	28.9
9	16.4202	2.5	9.8000	3.420	2.0	10	5	42	2	1	1	84.9
10	14.4598	2.5	12.8000	3.000	2.0	9	5	14	4	1	1	82.9
11	5.8282	1.0	6.					32	1	1	0	35.9
12	5.3003	1.0	4.					30	1	2	0	31.5
13	6.2712	1.0	5.					30	1	2	0	31.0
14	5.9592	1.0	6.6660	1.121	2.0	6	3	32	2	1	0	30.9
15	5.0500	1.0	5.0000	1.020	0.0	5	2	46	4	1	1	30.0
16	5.6039	1.0	9.5200	1.501	0.0	6	3	32	1	1	0	28.9
17	8.2464	1.5	5.1500	1.664	2.0	8	4	50	4	1	0	36.9
18	6.6969	1.5	6.9020	1.488	1.5	7	3	22	1	1	1	41.9
19	7.7841	1.5	7.1020	1.376	1.0	6	3	17	2	1	0	40.5
20	9.0384	1.0	7.8000	1.500	1.5	7	3	23	3	3	0	43.9
21	5.9894	1.0	5.5200	1.256	2.0	6	3	40	4	1	1	37.5
22	7.5422	1.5	4.0000	1.690	1.0	6	3	22	1	1	0	37.9
23	8.7951	1.5	9.8900	1.820	2.0	8	4	50	1	1	1	44.5
24	6.0931	1.5	6.7265	1.652	1.0	6	3	44	4	1	0	37.9
25	8.3607	1.5	9.1					48	1	1	1	38.9
26	8.1400	1.0	8.0					3	1	3	0	36.9
27	9.1416	1.5	7.3					31	4	1	0	45.8
28	12.0000	1.5	5.0000	1.200	2.0	6	3	30	3	1	1	41.0

Training Data (rows 1–23)

Testing Data (rows 24–28)

```
print("Predicts: \n", sess.run(hypothesis, feed_dict={X: x_test_data}))
```

```
Predicts:
 [[ 38.3451767 ]
 [ 47.06274033]
 [ 41.91234207]
 [ 45.6203537 ]
 [ 55.87483215]]
```

I, the index;
A1, the local selling prices, in hundreds of dollars;
A2, the number of bathrooms;
A3, the area of the site in thousands of square feet;
A4, the size of the living space in thousands of square feet;
A5, the number of garages;
A6, the number of rooms;
A7, the number of bedrooms;
A8, the age in years;
A9, 1 = brick, 2 = brick/wood, 3 = aluminum/wood, 4 = wood.
A10, 1 = two story, 2 = split level, 3 = ranch
A11, number of fire places.
B, the selling price.

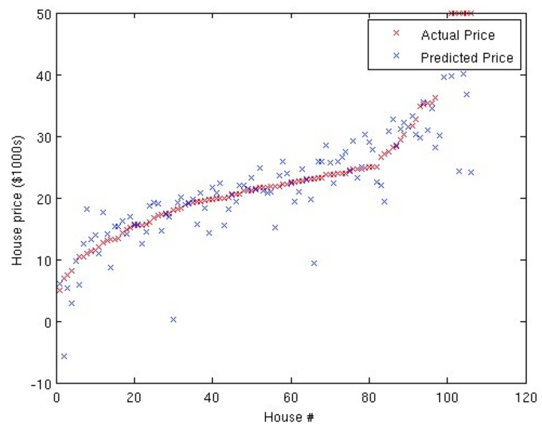
I, 색인
A1, 수백 달러의 현지 판매 가격
A2, 욕실 수
A3, 수천 평방 피트의 부지
A4, 수천 평방 피트의 생활 공간의 크기
A5, 차고 수
A6, 객실 수
A7, 침실 수
A8, 건물 년식;
A9, 1 = 벽돌, 2 = 벽돌 / 목재, 3 = 알루미늄 / 목재, 4 = 목재.
A10, 1 = 2층, 2 = 스플릿 레벨, 3 = 목장
A11, 화재 대피 장소.
B, 판매 가격

basketball_mlr09.txt

X1 = height in feet
X2 = weight in pounds
X3 = percent of successful field goals (out of 100 attempted)
X4 = percent of successful free throws (out of 100 attempted)
X5 = average points scored per game

Linear to Logistic Regression

- **Regression**
  - **Help predict values on a continuous spectrum**
    - Predicting what the price of a house will be
    - Predicts the final exam score
    - Predicts the basketball game score



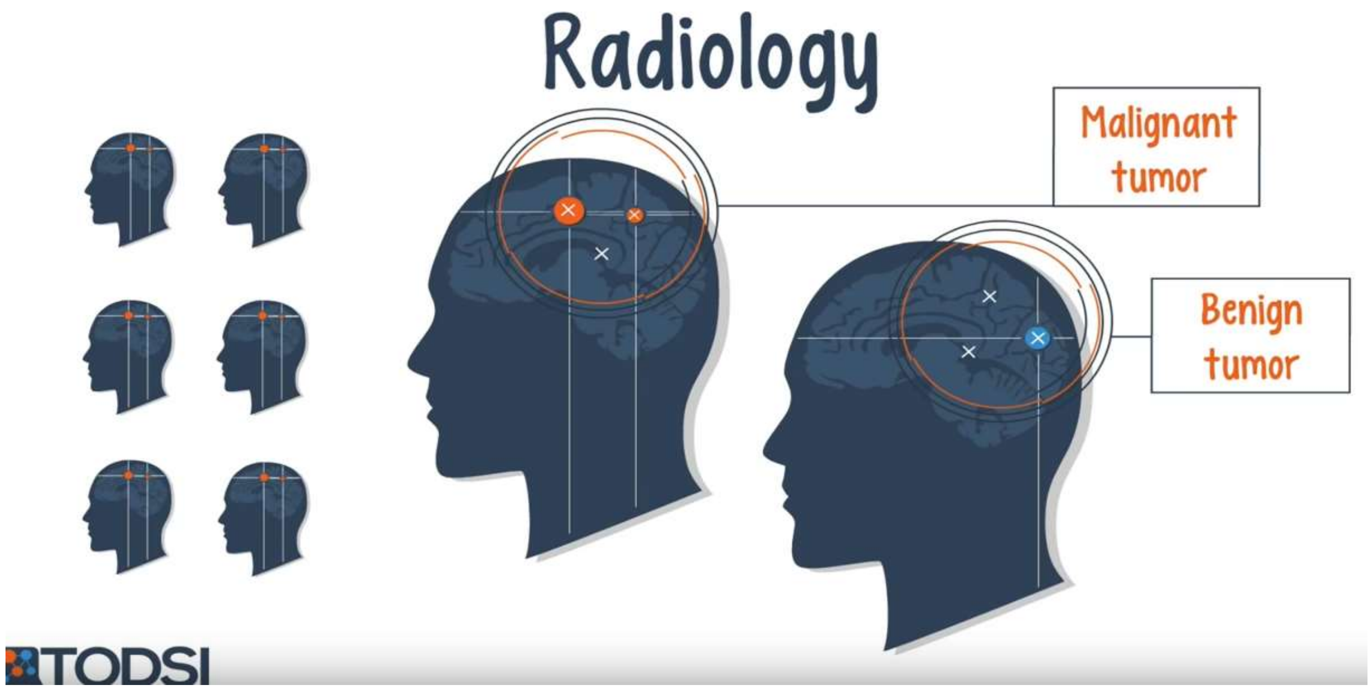
- **Classifying data among discrete classes**
  - **Determining whether a patient has cancer**
  - **Spam or not**
  - **Facebook feed: show or hide**
  - **Identifying the species of fish**

Regression: Continuous value
Classification: Discrete Value

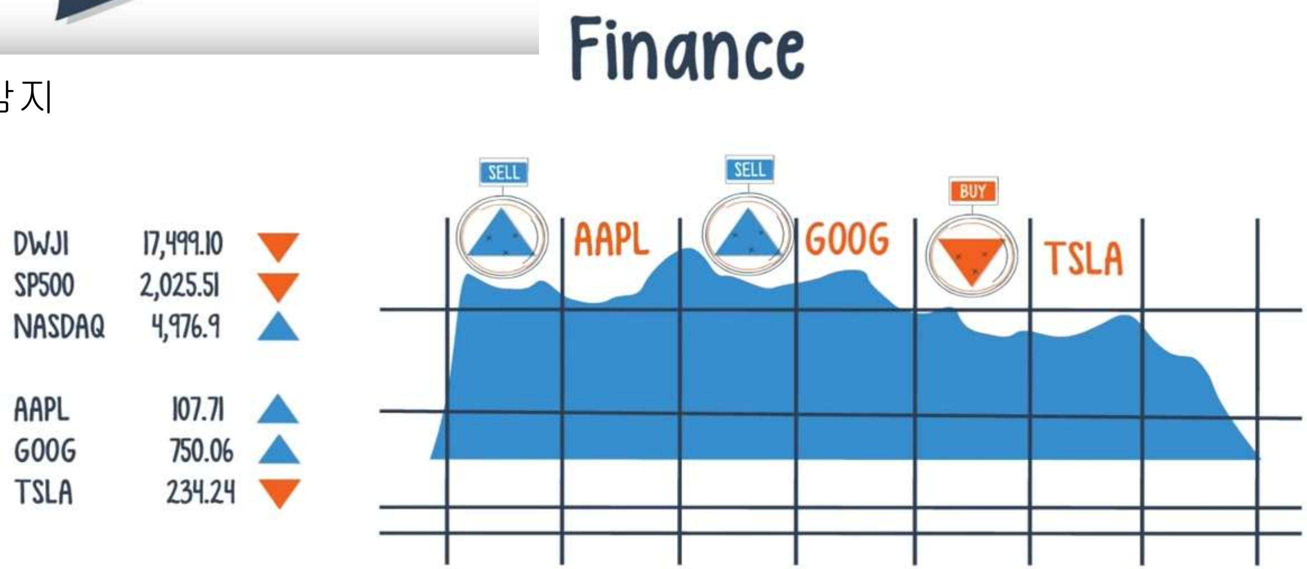
Solves it using Classification algorithm

Called
"Logistic Regression"

Classification 사용 예



CNN을 통하여 종양이나 암의 변이를 감지

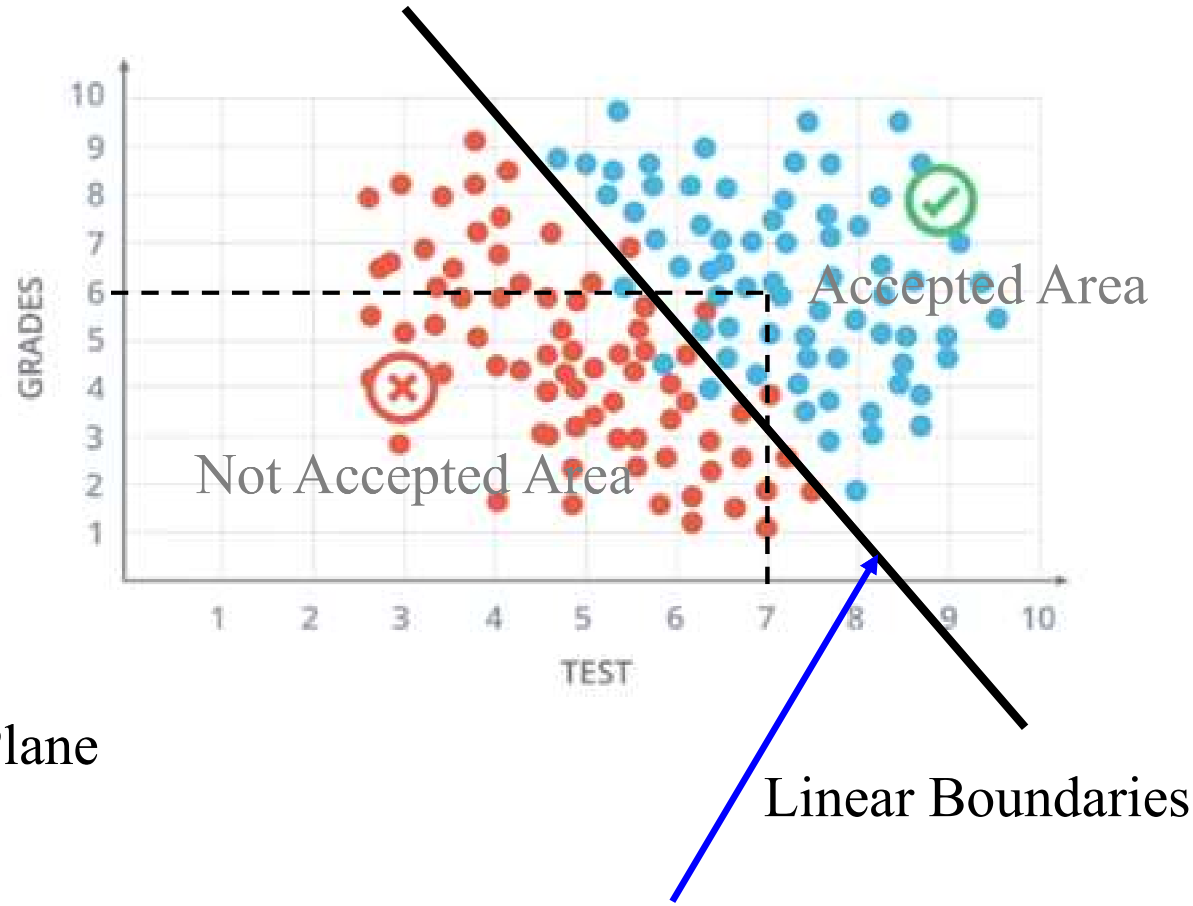


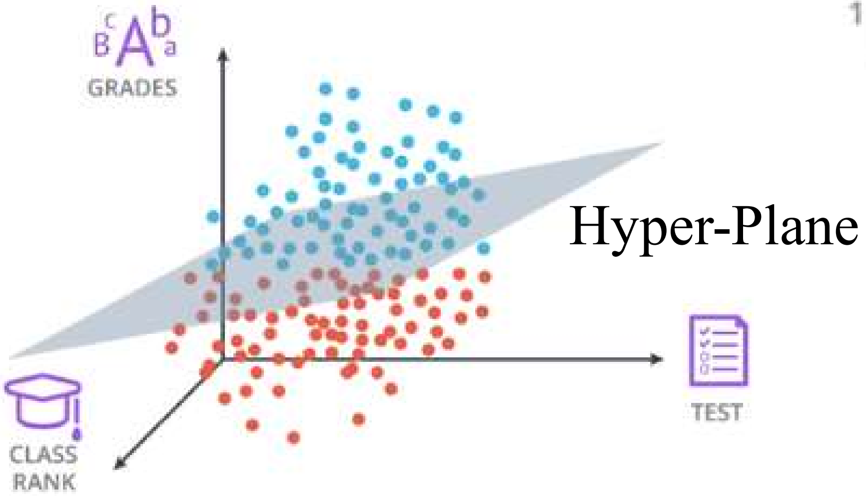
https://www.youtube.com/watch?v=BmkA1ZsG2P4

Deep Network를 통하여 매도, 매수 결정

Classification Problem

- **Example: Acceptance at a University**
  - **합격여부 기준: Test, Grades**
  - **How Do We know that student 3 will be accepted at University?**

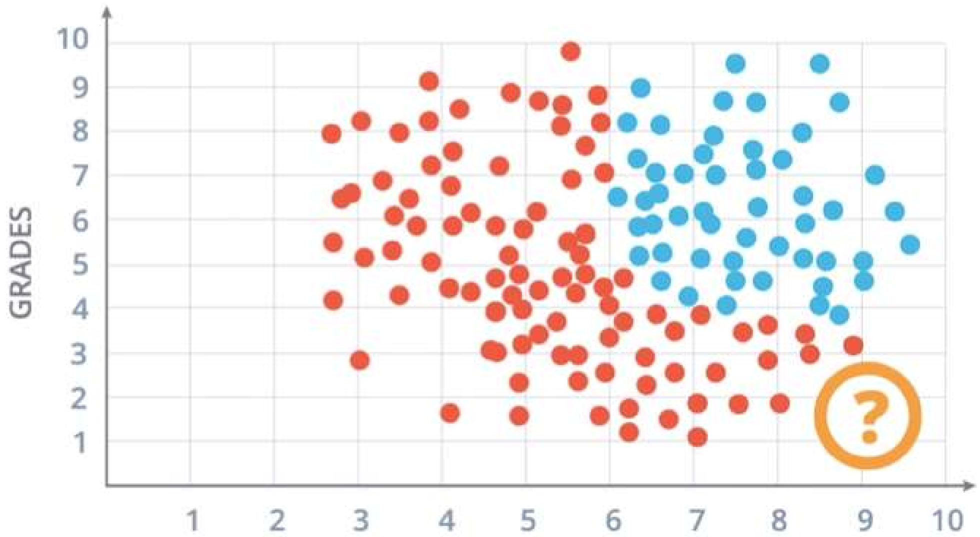




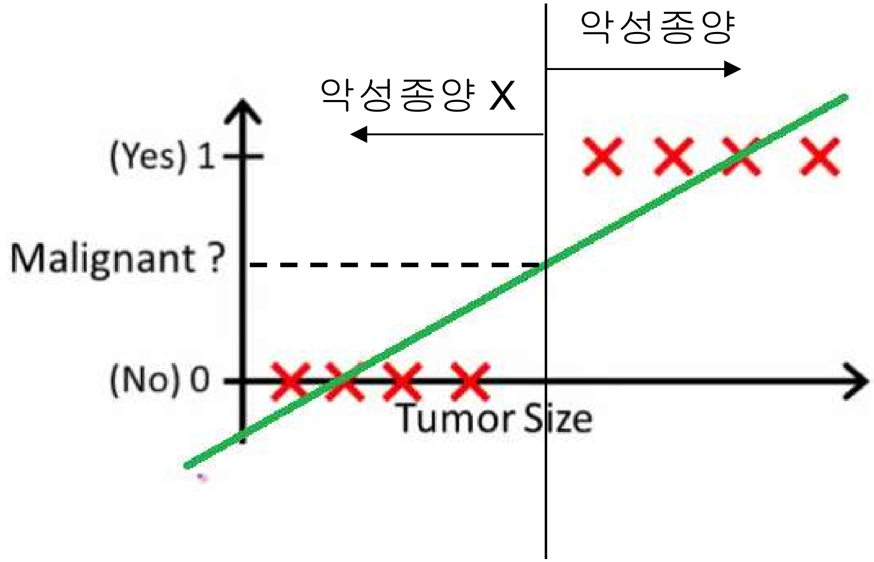
How Do We find this line?

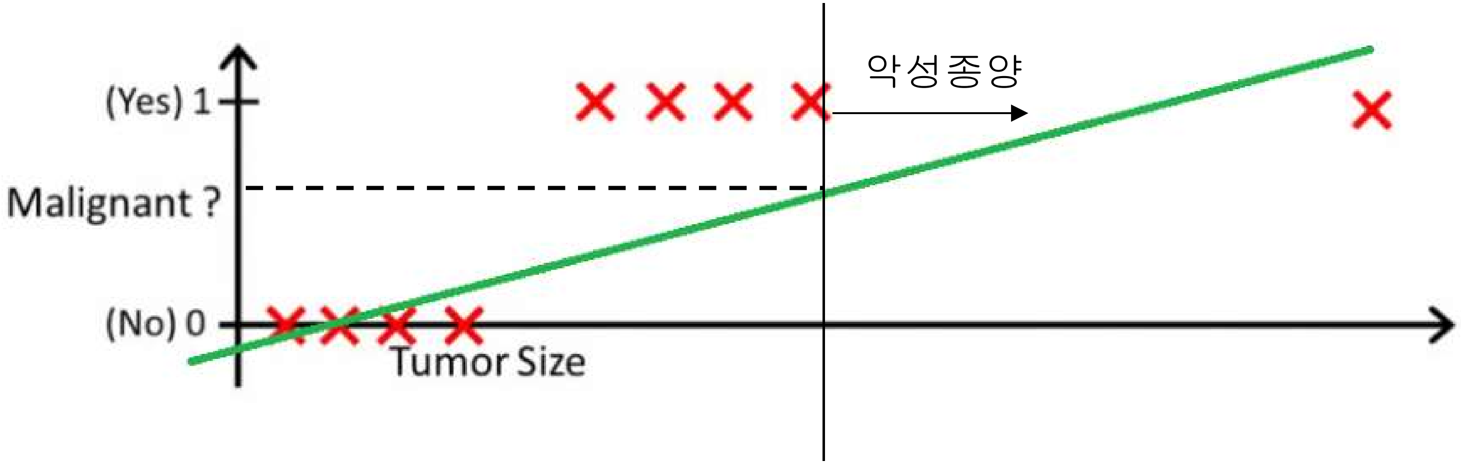
◆ **Classification 문제에 Linear Regression을 적용할 경우의 문제점**

Real Data



◆ Classification 문제에 Linear Regression을 적용할 경우의 문제점





(http://stats.stackexchange.com)

Logistic Hypothesis

- **0 <= h(x) <= 1 을 가지는 함수를 찾음**
  - $H(x) = Wx + b$

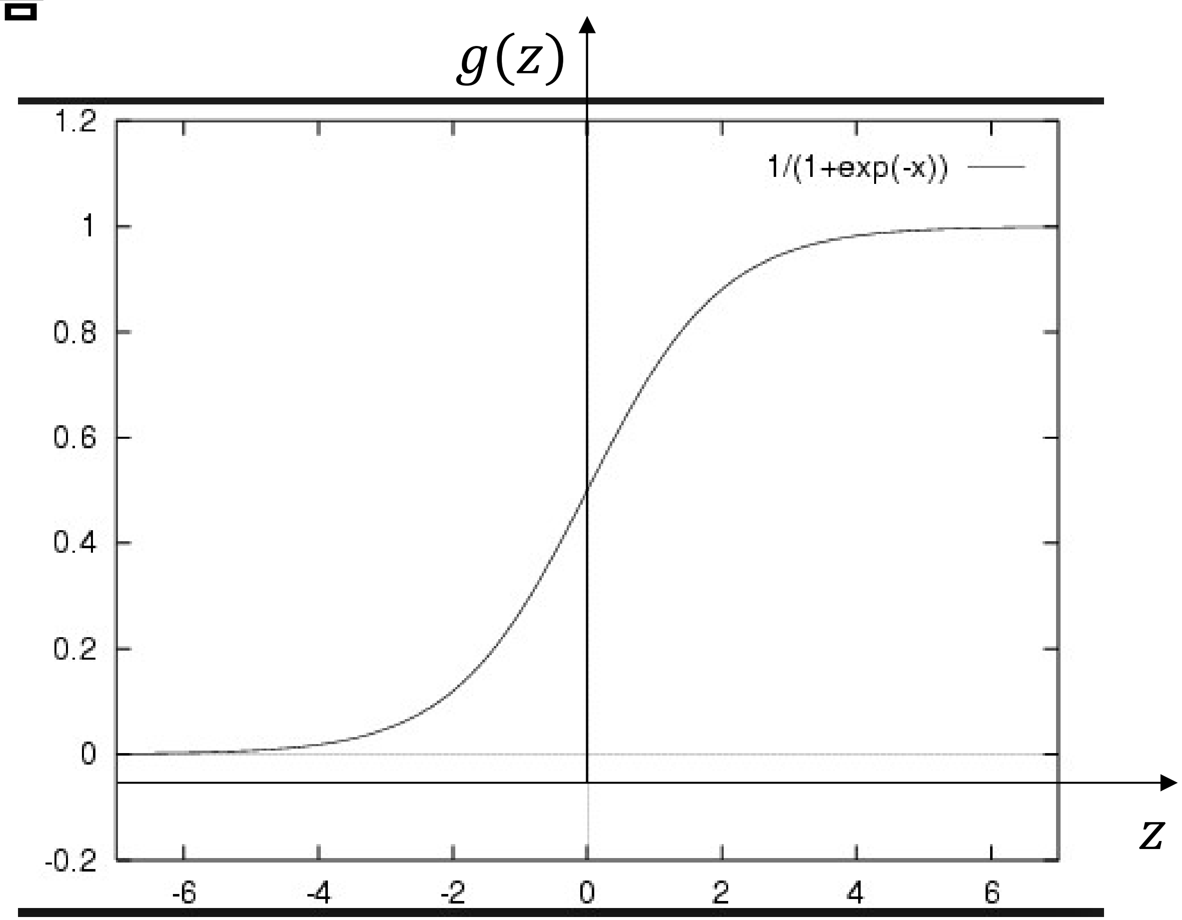
$z = Wx + b$

⇩

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

⇩

0~1 사이 값



$$H(X) = \frac{1}{1+e^{-\boxed{W^T X}}}$$

Linear
Regression Hypothesis

Logistic Function
Sigmoid Function

Cost Function of the Logistic Regression

◆ Cost Function

Linear Regression Cost Function

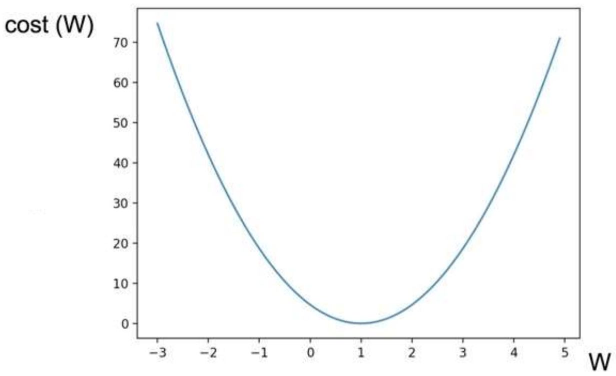
$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

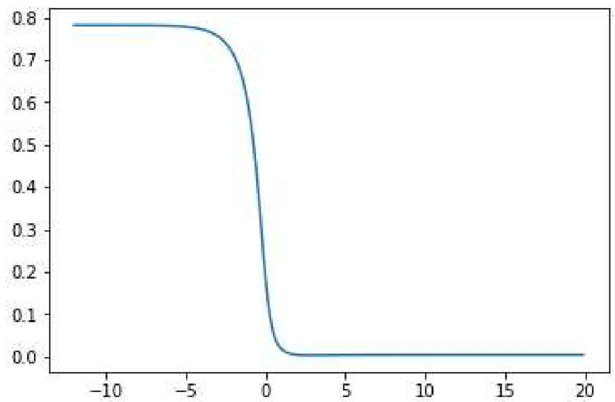
Linear Regression Hypothesis

$$\boxed{H(x) = Wx + b}$$

Logistic Regression Hypothesis

$$\boxed{H(X) = \frac{1}{1+e^{-W^T X}}}$$





Can we apply the Gradient Descent Algorithm this graph?

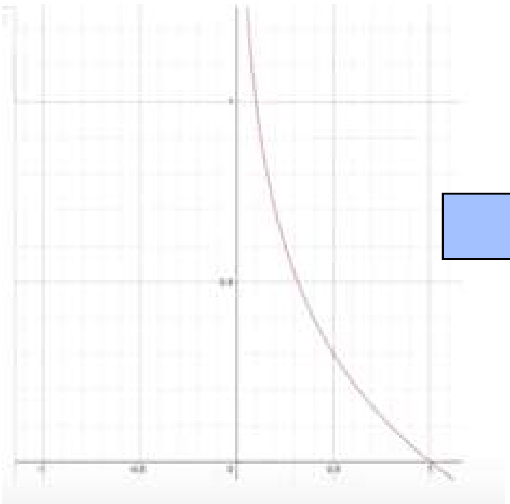
Cost Function

$$cost(W) = \frac{1}{m} \sum \quad c(H(x), y)$$

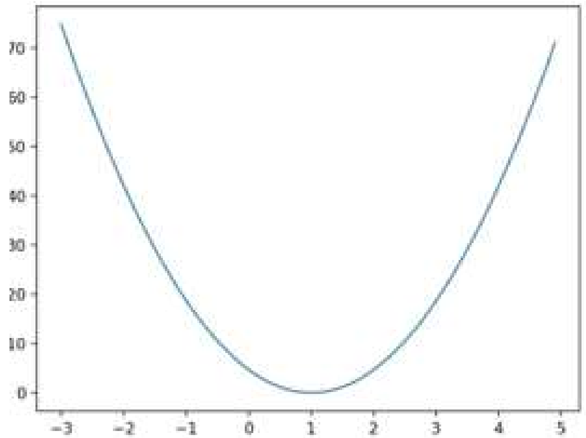
$$c(H(x), y) = \begin{cases} -log(H(x)) & : y = 1 \\ -log(1 - H(x)) & : y = 0 \end{cases}$$

$$\boldsymbol{c(H(x), y) = -ylog(Hx)) - (1 - y)\log(1 - H(x))}$$

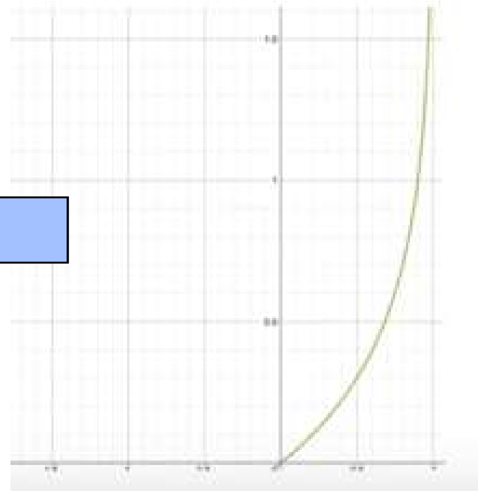
$$-log(H(x))$$



$$y = 1, c = -log(H(x))$$
$$y = 0, c = -log(1 - H(x))$$



$$-log(1 - H(x))$$



Cost Function of the Logistic Regression

◆ **Minimize Cost: Gradient Descent Algorithm**

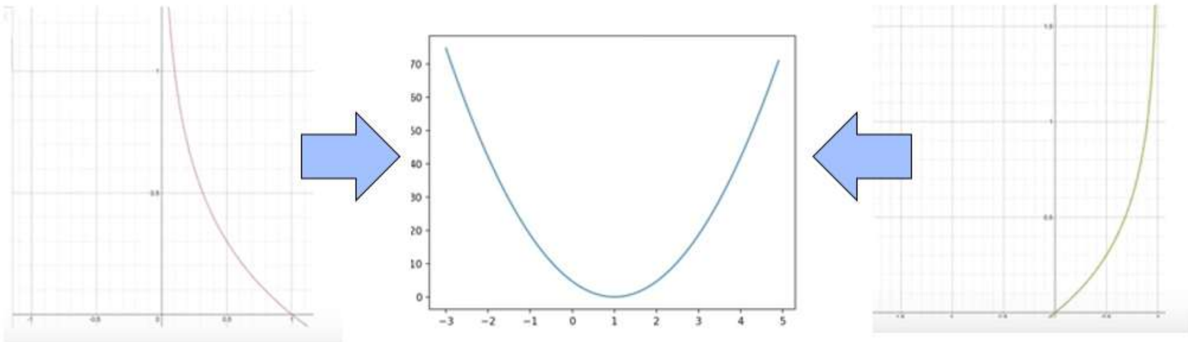
$$cost(W) = \frac{1}{m} \sum c(H(x), y)$$

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

$$\boldsymbol{c(H(x), y) = -ylog(Hx)) - (1 - y)\log(1 - H(x))}$$

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

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



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

◆ Classifying acceptance at a University (read file)

– **Data_logistic.txt**

» Test Dataset: Last 10 rows

» Train Dataset: n-10 rows

» Design Nodes

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

Hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

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

```
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

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

    for i in range(20001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: train_data_x, Y: train_data_y})
        if i % 2000 == 0:
            print(i, cost_val)

    acc = sess.run(accuracy, feed_dict={X: test_data_x, Y: test_data_y})
```

```
0 0.806795
2000 0.238209
4000 0.182931
6000 0.162146
8000 0.151287
10000 0.144676
12000 0.140276
14000 0.137172
16000 0.134888
18000 0.133155
20000 0.131809
```

```
print("Accuracy: ", acc)
```

◆ Classifying diabetes

– **Data-03-diabetes.csv (Test Dataset: Last 20 rows)**

```
data = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
```

```
print("Accuracy: ", acc)
```

```
Accuracy:  0.85
```

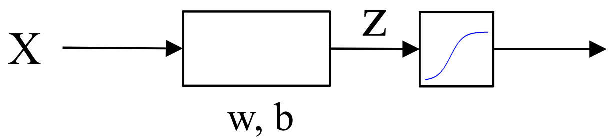
- **Logistic regression**

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

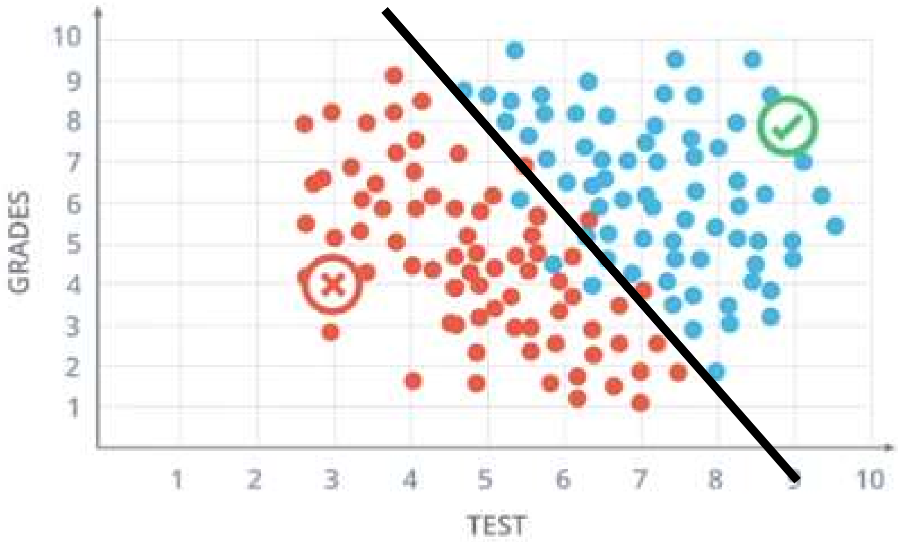
$$Z = H_L(x)$$

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

$$H_{LR} = g(H_L(x))$$

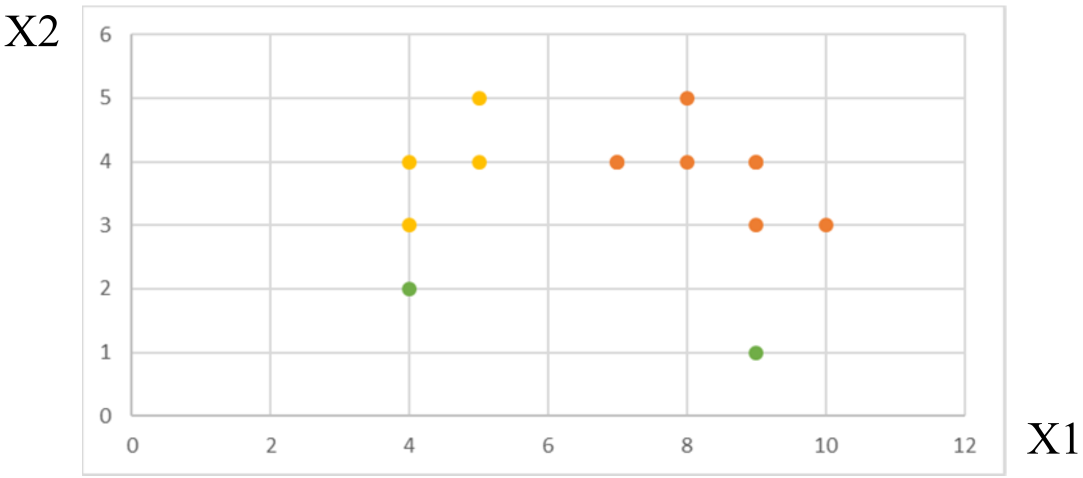


$$\bar{Y} \begin{cases} 0 \ (hypothesis < 0.5) \\ 1 \ (hypothesis \geq 0.5) \end{cases}$$

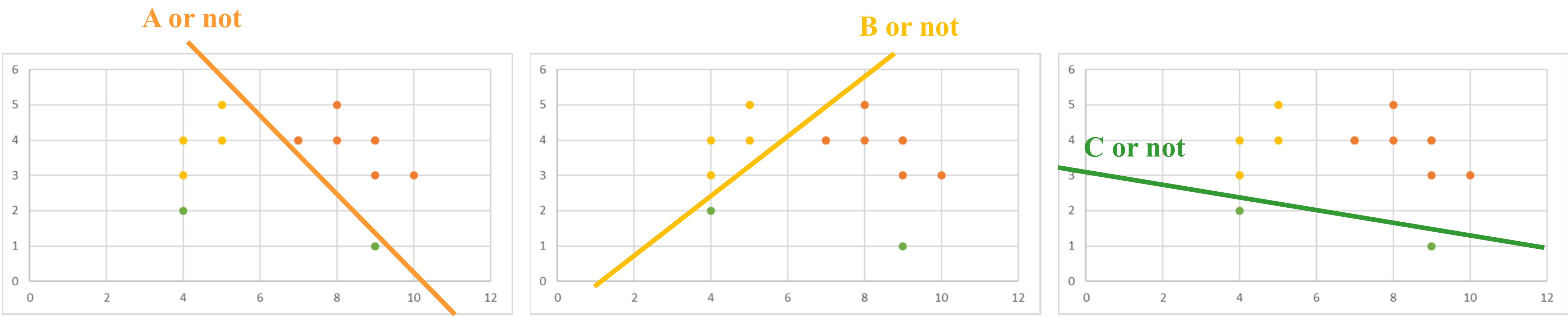


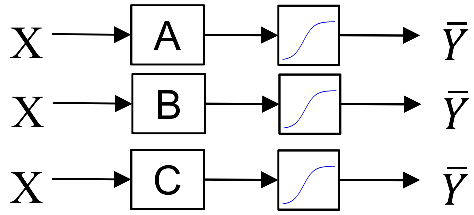
◆ Example of the Multinomial Classification

x1 (test)	x2 (attend)	y (grade)
10	3	A
9	4	A
9	4	A
9	3	A
8	4	A
8	5	A
7	4	A
7	4	A
5	5	B
5	4	B
4	4	B
4	3	B
4	2	C
9	1	C



◆ Combines the binary Classification



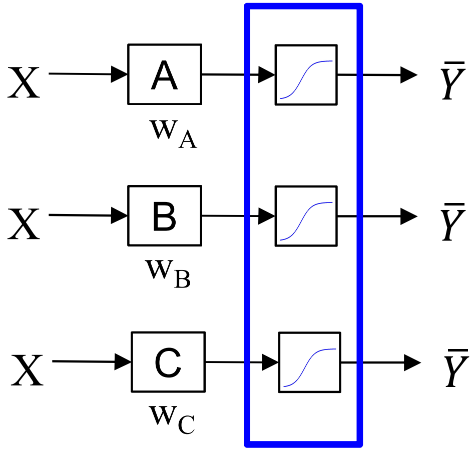


- **Multinomial Classification**

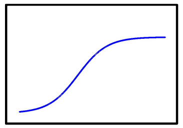
$$[W_{A1} \quad WA_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [W_{A1}\, x_1 + WA_2 x_2\,]$$

$$[W_{B1} \quad W_{B2}]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [W_{B1}\, x_1 + W_{B2} x_2\,]$$

$$[W_{C1} \quad W_{C2}]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [W_{C1}\, x_1 + W_{C2} x_2\,]$$



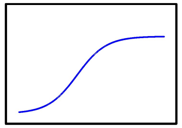
$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + WA_2 x_2 \\ W_{B1}x_1 + WB_2 x_2 \\ W_{C1}x_1 + WC_2 x_2 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases} \Rightarrow$$



$$\Rightarrow \begin{cases} 0.88 \\ 0.73 \\ 0.52 \end{cases}$$

- **Softmax**

$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + WA_2x_2 \\ W_{B1}x_1 + WB_2x_2 \\ W_{C1}x_1 + WC_2x_2 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases} \Rightarrow$$



$$\Rightarrow \begin{cases} 0.88 \\ 0.73 \\ 0.52 \end{cases}$$

$$\begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases} \Rightarrow \text{Softmax} \Rightarrow \begin{cases} 0.41 \\ 0.34 \\ 0.24 \end{cases}$$

	Grade
0.41	$\boxed{A}$
0.34	B
0.24	C

Score $\longrightarrow$ Probability

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

Why we use exponential term?

Multinomial Classification

- **Cross-Entropy (Cost Function)**

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

실제 값(Label data) L

	Grade	
0.65	1.0	A
0.24	0.0	B
0.11	0.0	C

$$D(S,L) = \boxed{-\sum_i L_i * log(Si)}$$

element wise Multiplication

$$\boxed{Cost = \frac{1}{N}\sum_i D\,(S_i, Li)}$$

$$-\sum_i Y_i * log(\bar{Y}_i) = \sum_i Y_i * -log(\bar{Y}_i)$$

$$Y = \begin{bmatrix}0\\1\end{bmatrix}$$

$$\bar{Y} = \begin{bmatrix}0\\1\end{bmatrix} \Rightarrow \begin{bmatrix}0\\1\end{bmatrix} * \begin{bmatrix}\infty\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix} = 0$$

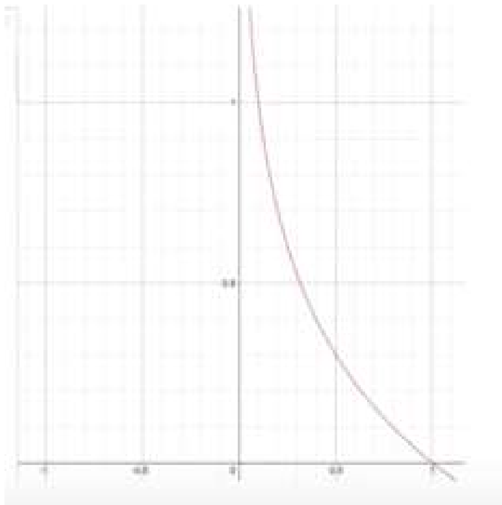
$$\bar{Y} = \begin{bmatrix}1\\0\end{bmatrix} \Rightarrow \begin{bmatrix}0\\1\end{bmatrix} * \begin{bmatrix}0\\\infty\end{bmatrix} = \begin{bmatrix}0\\\infty\end{bmatrix} = \infty$$

$$Y = \begin{bmatrix}1\\0\end{bmatrix}$$

$$\bar{Y} = \begin{bmatrix}1\\0\end{bmatrix} \Rightarrow \begin{bmatrix}1\\0\end{bmatrix} * \begin{bmatrix}0\\\infty\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix} = 0$$

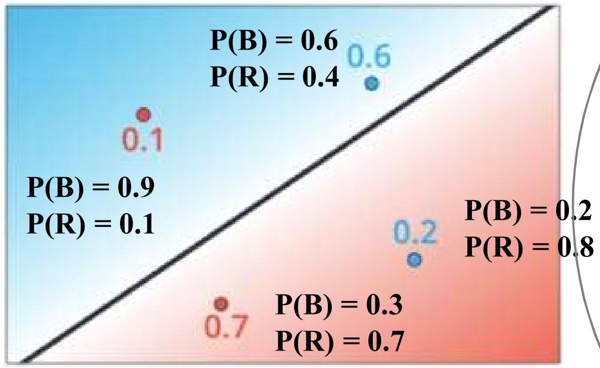
$$\bar{Y} = \begin{bmatrix}0\\1\end{bmatrix} \Rightarrow \begin{bmatrix}1\\0\end{bmatrix} * \begin{bmatrix}\infty\\0\end{bmatrix} = \begin{bmatrix}\infty\\0\end{bmatrix} = \infty$$

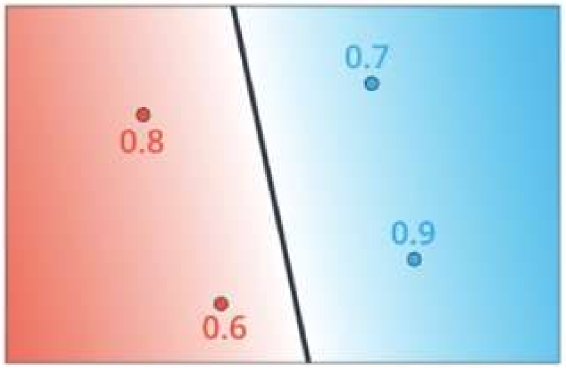
$-log(x)$



◆ Cross-Entropy

(어느 모델이 좋은 모델인가?)





0.6*0.2*0.1*0.7=0.0084

0.7*0.9*0.8*0.6=$\boxed{0.3024}$

Products are bad, but sums are good!!

Cross Entropy가 낮을수록 좋은 모델

$$\log(0.6) + \log(0.2) + \log(0.1) + \log(0.7)$$
$$= -0.22 + -0.69 + -1 + -0.15$$

$$-\log(0.6) - \log(0.2) - \log(0.1) - \log(0.7)$$
$$= 0.22 + 0.69 + 1 + 0.15 = \mathbf{2.0757}$$

$$-\log(0.7) - \log(0.9) - \log(0.8) - \log(0.6)$$
$$= 0.15 + 0.04 + 0.09 + 0.22 = \mathbf{0.5194}$$

Multinomial Classification

- **Classifying type of the animal**

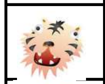
$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 \\ W_{B1}x_1 + W_{B2}x_2 \\ W_{C1}x_1 + W_{C2}x_2 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases}$$

$$\begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases} \Rightarrow \text{Softmax} \Rightarrow \begin{cases} 0.65 \\ 0.24 \\ 0.11 \end{cases}$$

Score → Probability

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

$$D(S, L) = -\sum_i L_i * log(Si) \qquad Cost = \frac{1}{N} \sum_i D\,(S_i, Li)$$

0		1	0	0	0	0	0	0
1		0	1	0	0	0	0	0
2		0	0	1	0	0	0	0
3		0	0	0	1	0	0	0
4		0	0	0	0	1	0	0
5		0	0	0	0	0	1	0
6		0	0	0	0	0	0	1

"One-hot encoding"

```
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)

classes = 7  # 0 ~ 6

X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1])
Y_one_hot = tf.one_hot(Y, classes)  # N x 1 x 7
print("one_hot_encoding", Y_one_hot)
Y_one_hot = tf.reshape(Y_one_hot, [-1, classes])  # N x 7
print("reshape: one_hot_encoding", Y_one_hot)

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

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

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

```
for i in range(2000):
    sess.run(opt, feed_dict={X: x_data, Y: y_data})
    if i % 100 == 0:
        loss, acc = sess.run([cost, accuracy], feed_dict={
                             X: x_data, Y: y_data})
        print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(
            i, loss, acc))

print("Training Done!!!")
pred = sess.run(prediction, feed_dict={X: x_data})
for p, y in zip(pred, y_data.flatten()):
    print("[{}] Prediction: {} True Y: {}".format(p == int(y), p, int(y)))
```

Multinomial Classification

- **Classifying digit using MNIST Dataset**

$$\begin{bmatrix} W_{A1} & W_{A2} \\ W_{B1} & W_{B2} \\ W_{C1} & W_{C2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 \\ W_{B1}x_1 + W_{B2}x_2 \\ W_{C1}x_1 + W_{C2}x_2 \end{bmatrix} = \begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases}$$

$$\begin{bmatrix} \bar{Y}_A \\ \bar{Y}_B \\ \bar{Y}_C \end{bmatrix} = \begin{cases} 2.0 \\ 1.0 \\ 0.1 \end{cases} \Rightarrow \text{Softmax} \Rightarrow \begin{cases} 0.65 \\ \vdots \\ 0.24 \\ \vdots \\ 0.11 \end{cases}$$

Score → Probability

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

$$D(S, L) = -\sum_i L_i * log(Si) \qquad Cost = \frac{1}{N} \sum_i D\,(S_i, Li)$$

```python
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, classes])

W = tf.Variable(tf.random_normal([784, classes]))
b = tf.Variable(tf.random_normal([classes]))

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

# adjust learning_rate
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
opt = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

MINIST Dataset

55,000	train-images-idx3-ubyte.gz:	training set images (9912422 bytes)
	train-labels-idx1-ubyte.gz:	training set labels (28881 bytes)
10,000	t10k-images-idx3-ubyte.gz:	test set images (1648877 bytes)
	t10k-labels-idx1-ubyte.gz:	test set labels (4542 bytes)

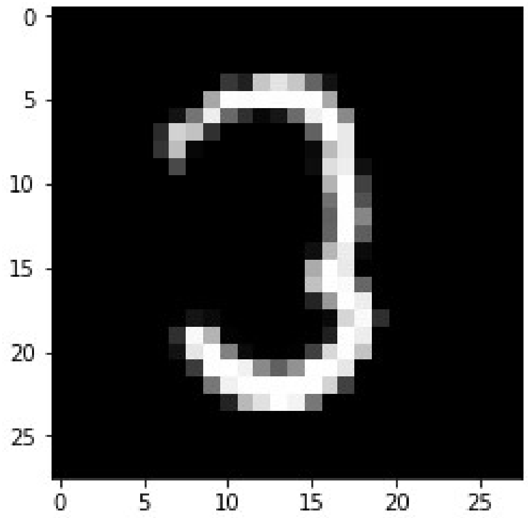
◆ Classifying digit using MNIST Dataset

In the neural network terminology:

- one **epoch** = one forward pass and one backward pass of *all* the training examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

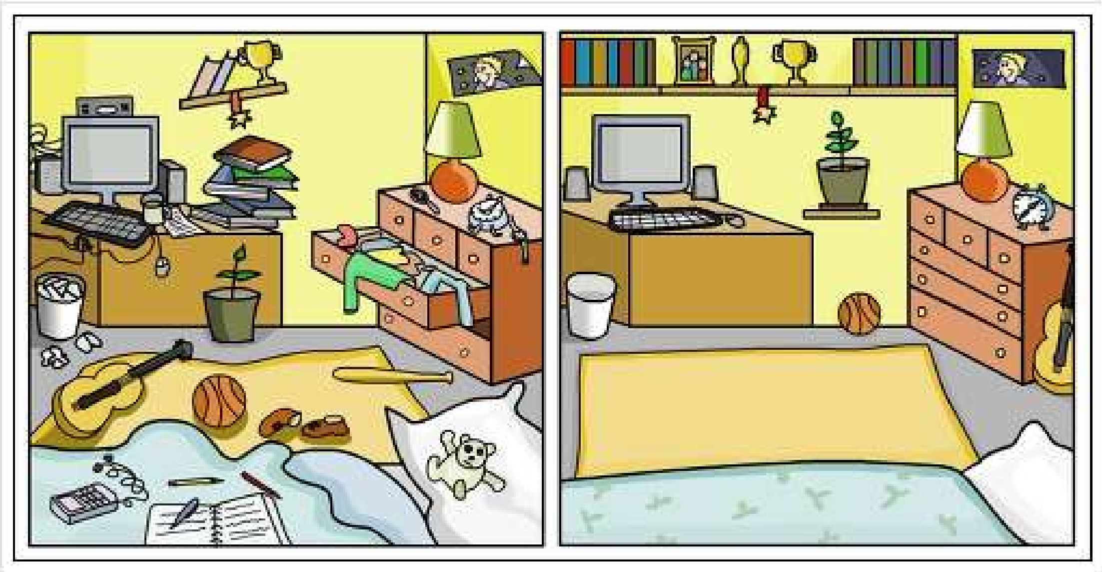
Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

```
Training Done!!!!
Accuracy:  0.8906
Label:  [3]
Prediction:  [3]
```



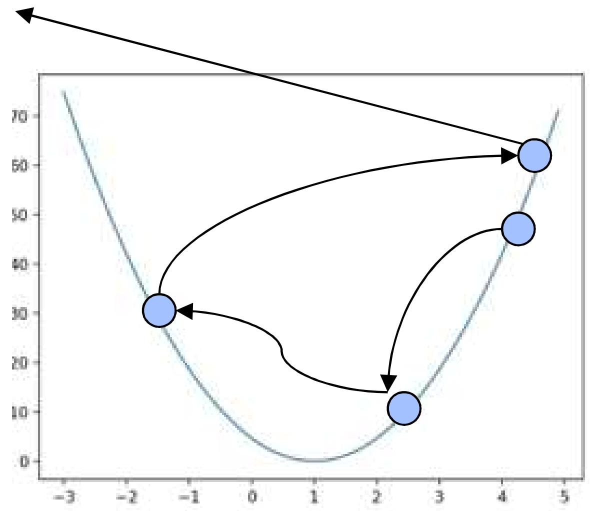

```python
# Test model
correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

# parameters (modify this)
training_epochs = 10
batch_size = 256

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0
        iteration = int(mnist.train.num_examples / batch_size)

        for i in range(iteration):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            c, _ = sess.run([cost, opt], feed_dict={
                            X: batch_xs, Y: batch_ys})
            avg_cost += c / iteration

        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

    print("Training Done!!!!")

    # Test the model using test sets
    print("Accuracy: ", sess.run(accuracy, feed_dict={
          X: mnist.test.images, Y: mnist.test.labels}))

    # Get one and predict
    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
    print("Prediction: ", sess.run(
        tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

    plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), 'gray')
    plt.show()
```

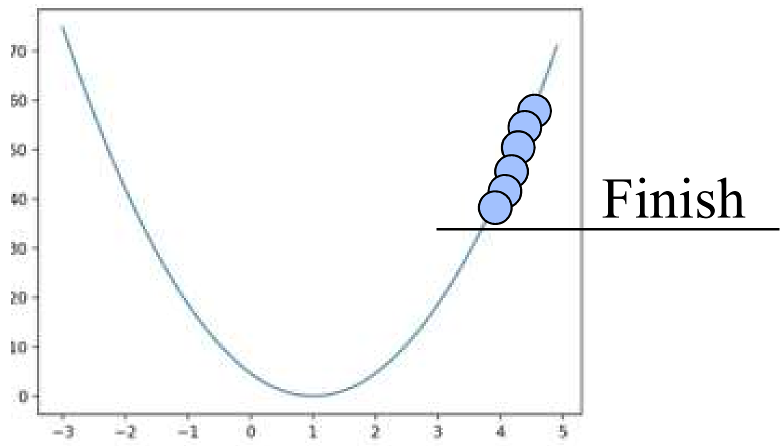
- **Cross-Entropy**



Learning rate

- Large Learning rate: Overshooting
- Small Learning rate: takes too long time, stops at local minimum



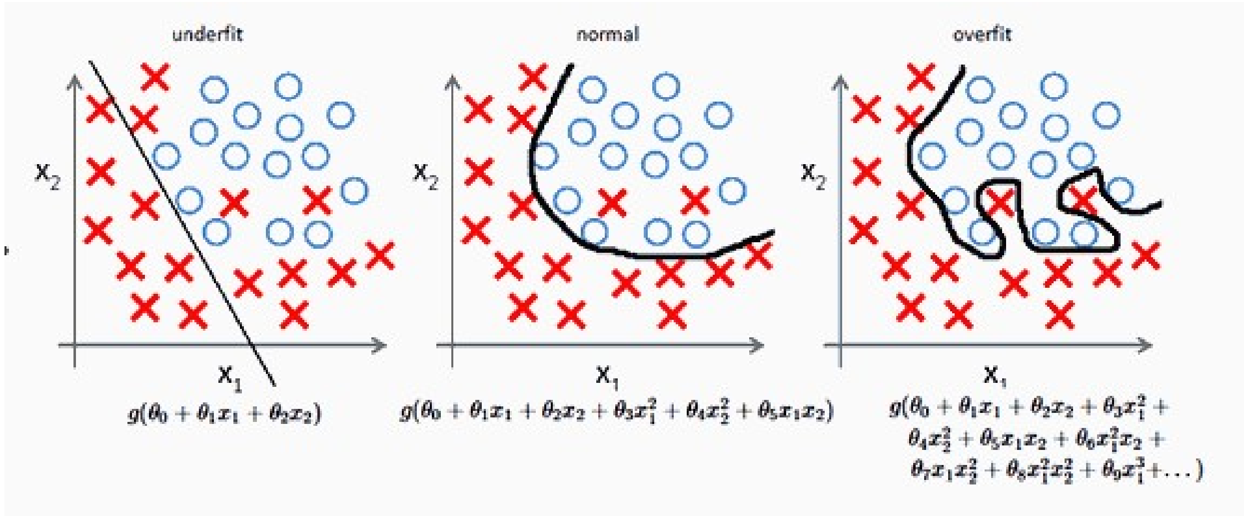


Solution:

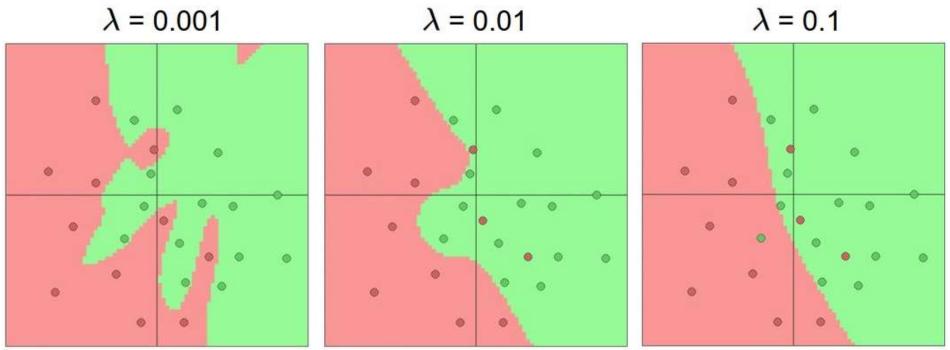
- Observe the cost function

◆ Overfitting

– Our model is very good with training data set
– Not good at test dataset or in real use



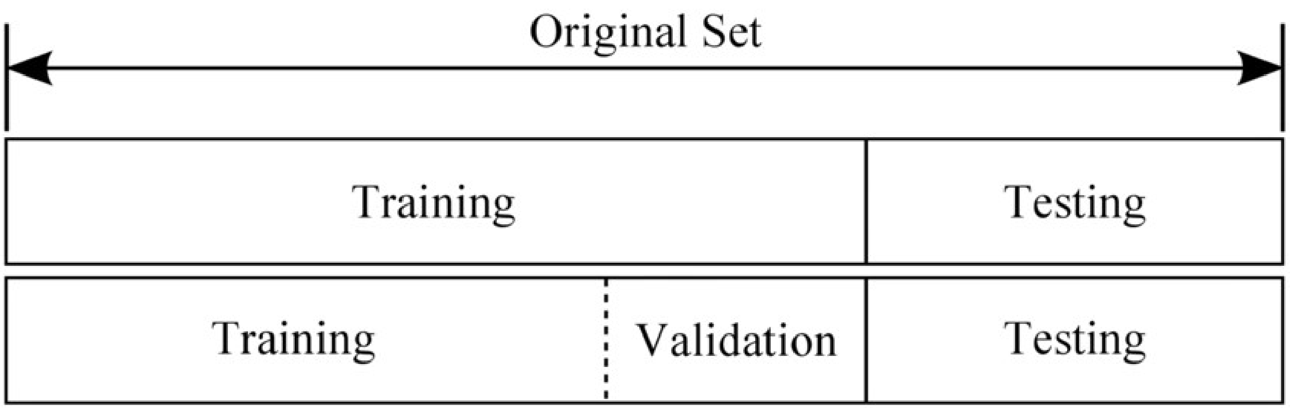
Solution:
- **More Training Data**
- **Reduce the number of features**
- **Regularization**



$$min_\theta \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

http://mlwiki.org/index.php/Overfitting

◆ Dataset 구성



MINIST Dataset

55,000	train-images-idx3-ubyte.gz:	training set images (9912422 bytes)
	train-labels-idx1-ubyte.gz:	training set labels (28881 bytes)
10,000	t10k-images-idx3-ubyte.gz:	test set images (1648877 bytes)
	t10k-labels-idx1-ubyte.gz:	test set labels (4542 bytes)

Thank you & Good luck !