Deep Learning (2)

윤명현

2020. 7.

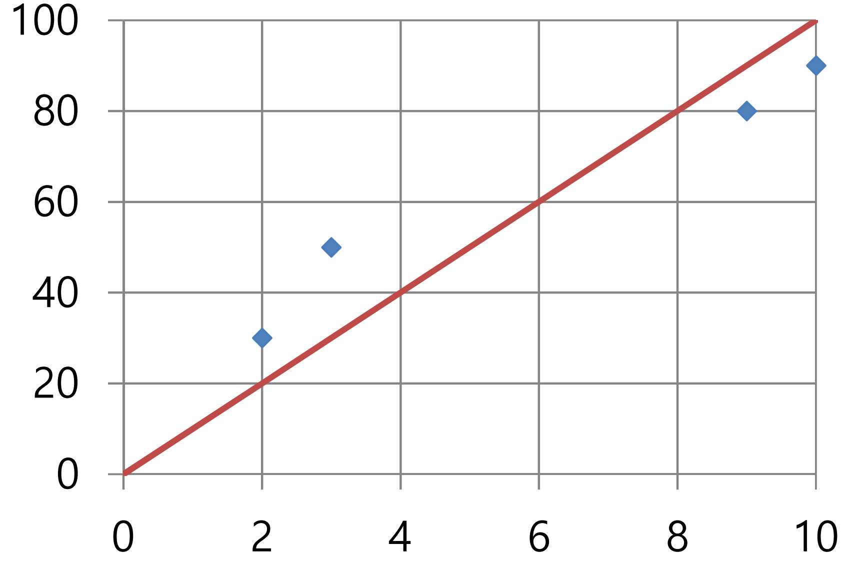
목 차

Linear Regression

Linear Regression

- Predicting exam score

x(hour)	y(score)
10	90
9	80
3	50
2	30



- Hypothesis

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

weight bias

- Error/cost/loss/objective function

$$C(w, b) = \frac{1}{N}\sum_{i=1}^{N}\{H(x_i) - y_i\}^2$$

Building & Launching Graph

```
x_train = [2, 3, 9, 10]
y_train = [30, 50, 80, 90]

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

```
hypothesis = x_train * w + b
cost = tf.reduce_mean(tf.square(hypothesis - y_train))
```

* Gradient descent

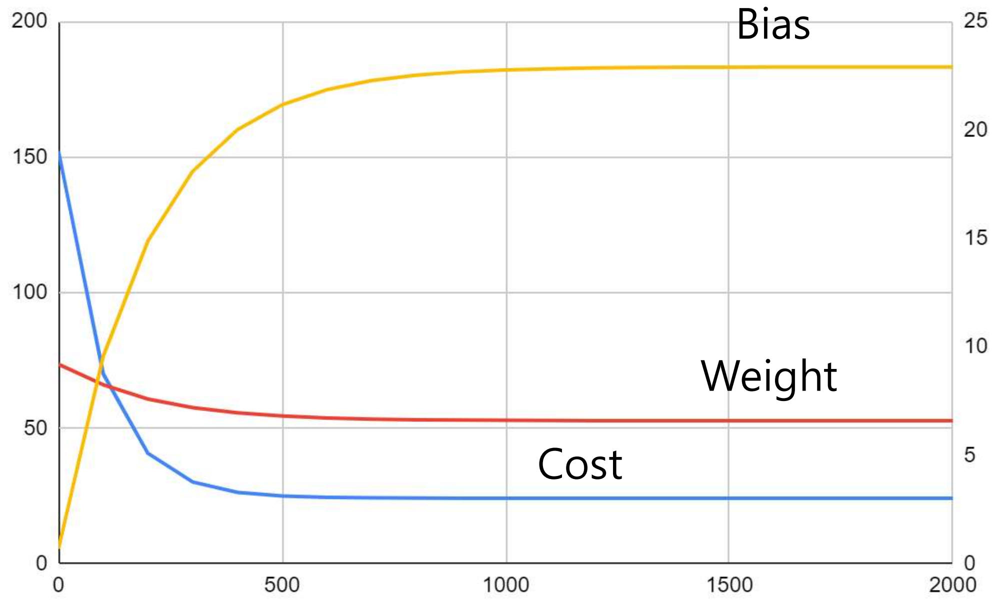
```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(2001):
   sess.run(train)
   if step % 100 == 0:
      print(step, sess.run(cost), sess.run(w), sess.run(b))
```

Training Output

Cost Weight Bias

```
0 152.24857 [9.202369] [0.702132]
100 70.06434 [8.258013] [9.567877]
200 40.80526 [7.5966783] [14.88568]
300 30.232336 [7.1991315] [18.082363]
400 26.411758 [6.960155] [20.003979]
500 25.031162 [6.8164997] [21.159119]
600 24.532278 [6.730144] [21.85351]
700 24.351997 [6.678232] [22.270931]
800 24.286858 [6.647028] [22.521847]
900 24.263315 [6.62827] [22.672676]
1000 24.25481 [6.6169934] [22.763357]
1100 24.251734 [6.6102147] [22.817865]
1200 24.250633 [6.6061397] [22.85063]
1300 24.250229 [6.603691] [22.87032]
1400 24.250078 [6.602219] [22.882156]
1500 24.250027 [6.601334] [22.889273]
1600 24.25001 [6.600802] [22.893549]
1700 24.250008 [6.600482] [22.896122]
1800 24.25 [6.60029] [22.897667]
1900 24.249996 [6.6001744] [22.898596]
2000 24.249994 [6.600106] [22.89915]
```



Placeholder

```
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 = x * w + b
cost = tf.reduce_mean(tf.square(hypothesis - y))
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(2001):
   cost_val, w_val, b_val, _ = sess.run([cost, w, b, train],
                 feed_dict={x: [2, 3, 9, 10], y: [30, 50, 80, 90]})
   if step % 100 == 0:
      print(step, cost_val, w_val, b_val)
```

Parameter Optimization

- Gradient (steepest) descent

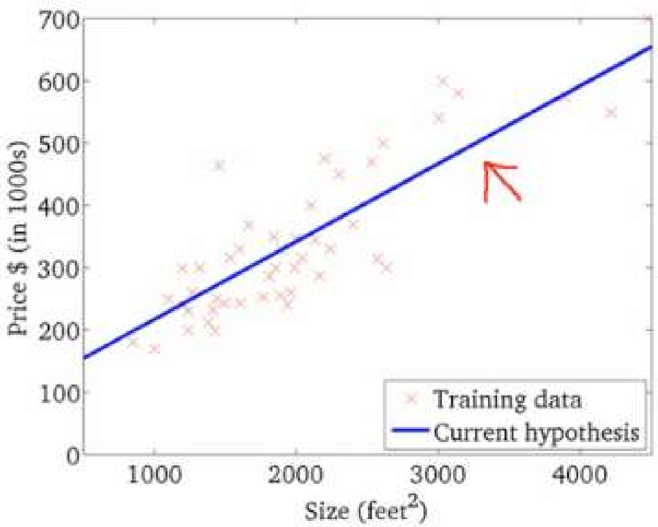
$$\theta^* = \arg\min_{\theta} J(\theta) \quad \Longrightarrow \quad \nabla J(\theta) = 0$$

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \alpha \nabla J\left(\theta^{(\tau)}\right)$$

learning rate

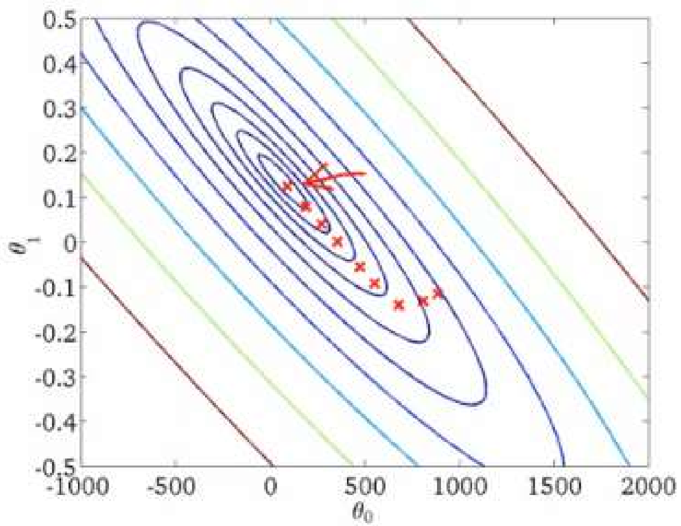
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

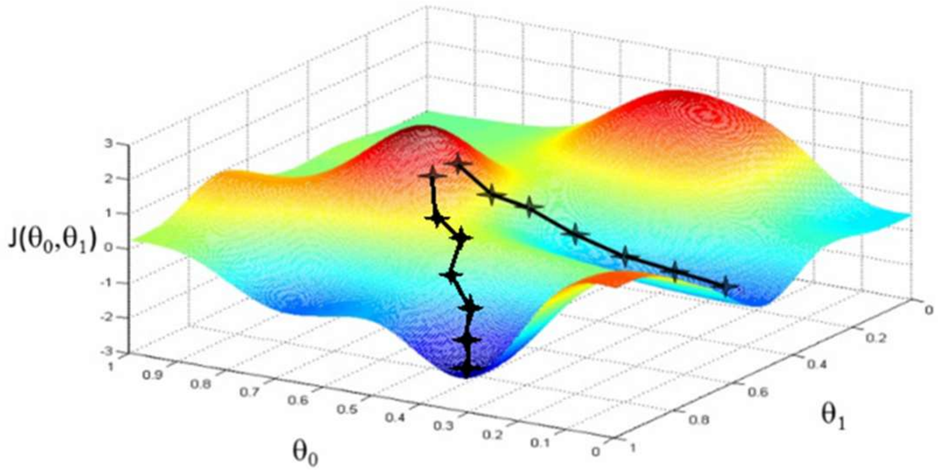
(function of the parameters θ_0, θ_1)



<http://www.holehouse.org/mlclass/>

Gradient Descent Algorithm

- Start with initial guesses
- Keep changing *w* and *b* a bit to try and reduce *cost(w,b)*
- Each time you change the parameters, you select the gradient which reduces *cost(w,b)* the most possible
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
  - Where you start can determine which minimum you end up

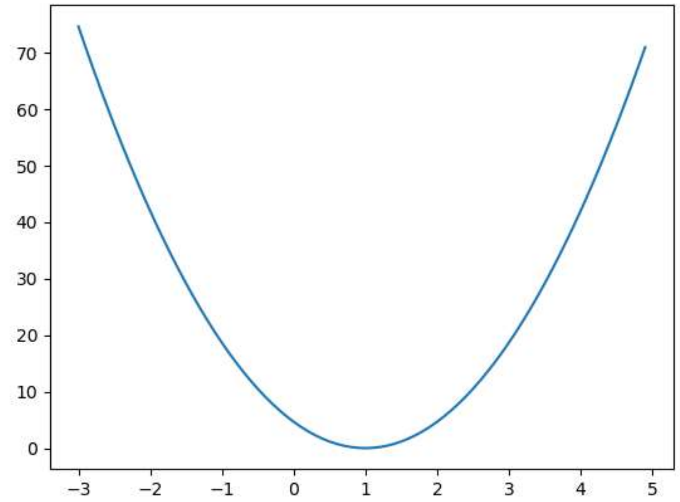


Gradient Descent

```
x = [1, 2, 3]
y = [1, 2, 3]
hypothesis = x * w
```

$$C(w,b) = \frac{1}{N}\sum_{i=1}^{N}\{H(x_i) - y_i\}^2$$

$$w = w - \alpha\frac{1}{N}\sum_{i=1}^{N}(wx_i - y_i)x_i$$




```
learning_rate = 0.1
gradient = tf.reduce_mean((w*x-y)*x)
descent = w-learning_rate*gradient
update = w.assign(descent)
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)
```

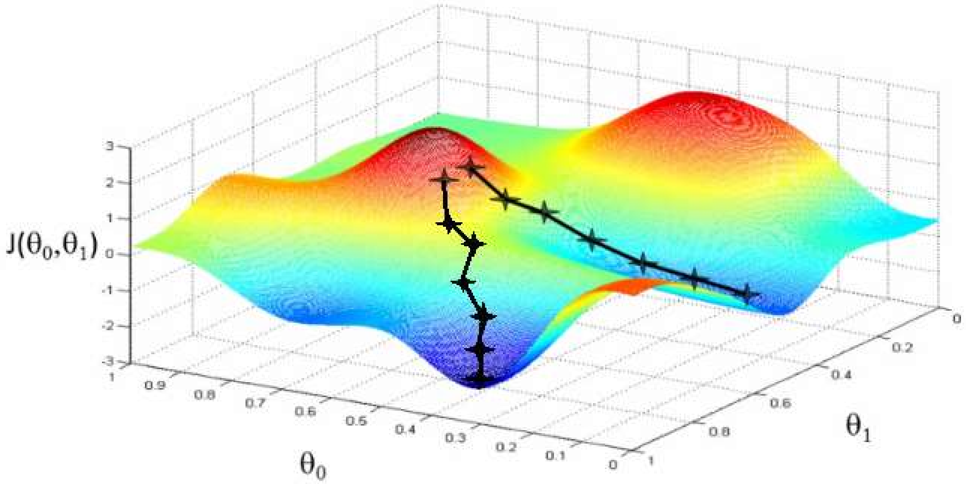
compute_gradient, apply_gradient

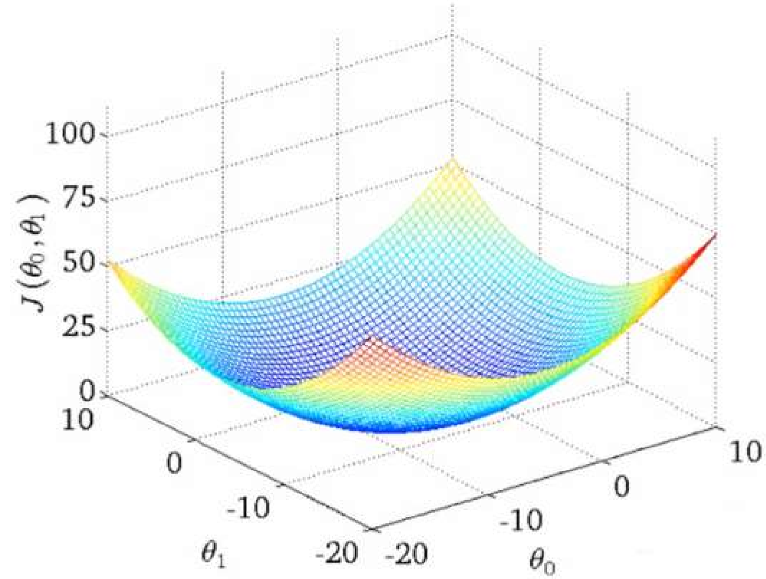
Gradient에 변화를 주고자 할 때

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)

gvs = optimizer.compute_gradients(cost)
apply_gradients = optimizer.apply_gradients(gvs)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

for step in range(100):
   print(step, sess.run([gradient, W, gvs]))
   sess.run(apply_gradients)
```

Convex Function

- Convex objective function





<http://www.holehouse.org/mlclass/>

Multi-variable Linear Regression

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test score for general psychology

Hypothesis using matrix

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

```
x1_data = [73., 93., 89., 96., 73.]
x2_data = [80., 88., 91., 98., 66.]
x3_data = [75., 93., 90., 100., 70.]
y_data = [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]))
w2 = tf.Variable(tf.random_normal([1]))
w3 = tf.Variable(tf.random_normal([1]))
b = tf.Variable(tf.random_normal([1]))

hypothesis = x1*w1 + x2*w2 + x3*w3 + b
```

Matrix Form

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

```
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 = tf.matmul(x,w) + b
```

(N개의 데이터)

Hypothesis Using Matrix

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

`[n, 3]` **`[3, 2]`** `[n, 2]`

$$H(X) = XW \qquad H(X) = W^T X$$

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

Indexing, Slicing, Iterating

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

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

a[1:3]      array([2,3])
a[-1]       5
a[0:2]=9    array([9,9,3,4,5]
```

```
b = np.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]])
```

Loading Data from File

Data-01-test-score.csv

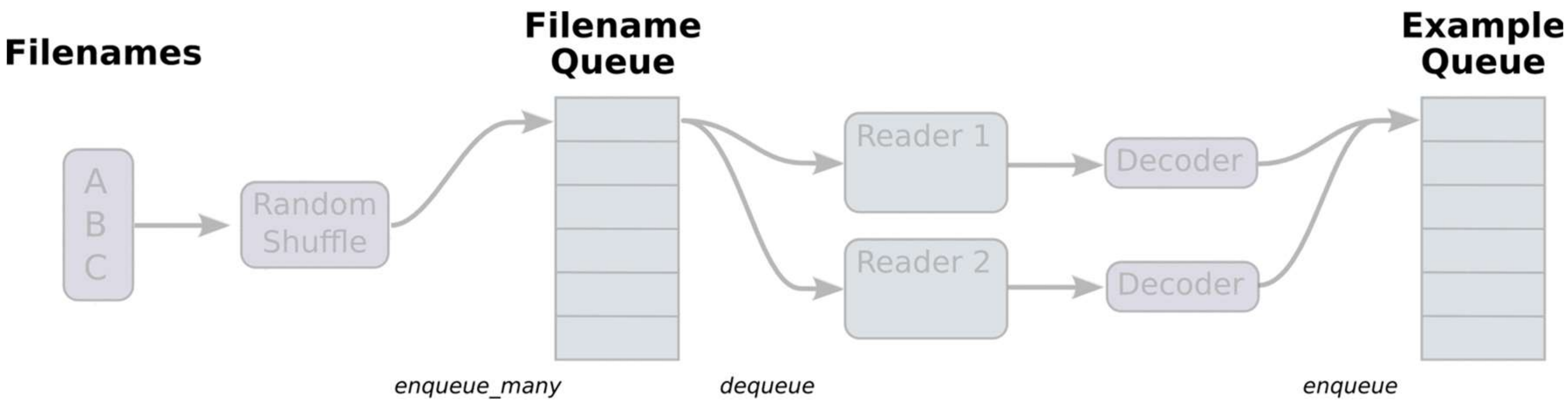
```
# EXAM1,EXAM2,EXAM3,FINAL
73,80,75,152
93,88,93,185
89,91,90,180
96,98,100,196
73,66,70,142
53,46,55,101
```

```
import numpy as np

xy = np.loadtxt('data-01-test-score.csv',delimiter=',',dtype=np.floaf32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, x_data, len(x_data))
print(y_data,shape, y_data)
```

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)
```

Batch Training

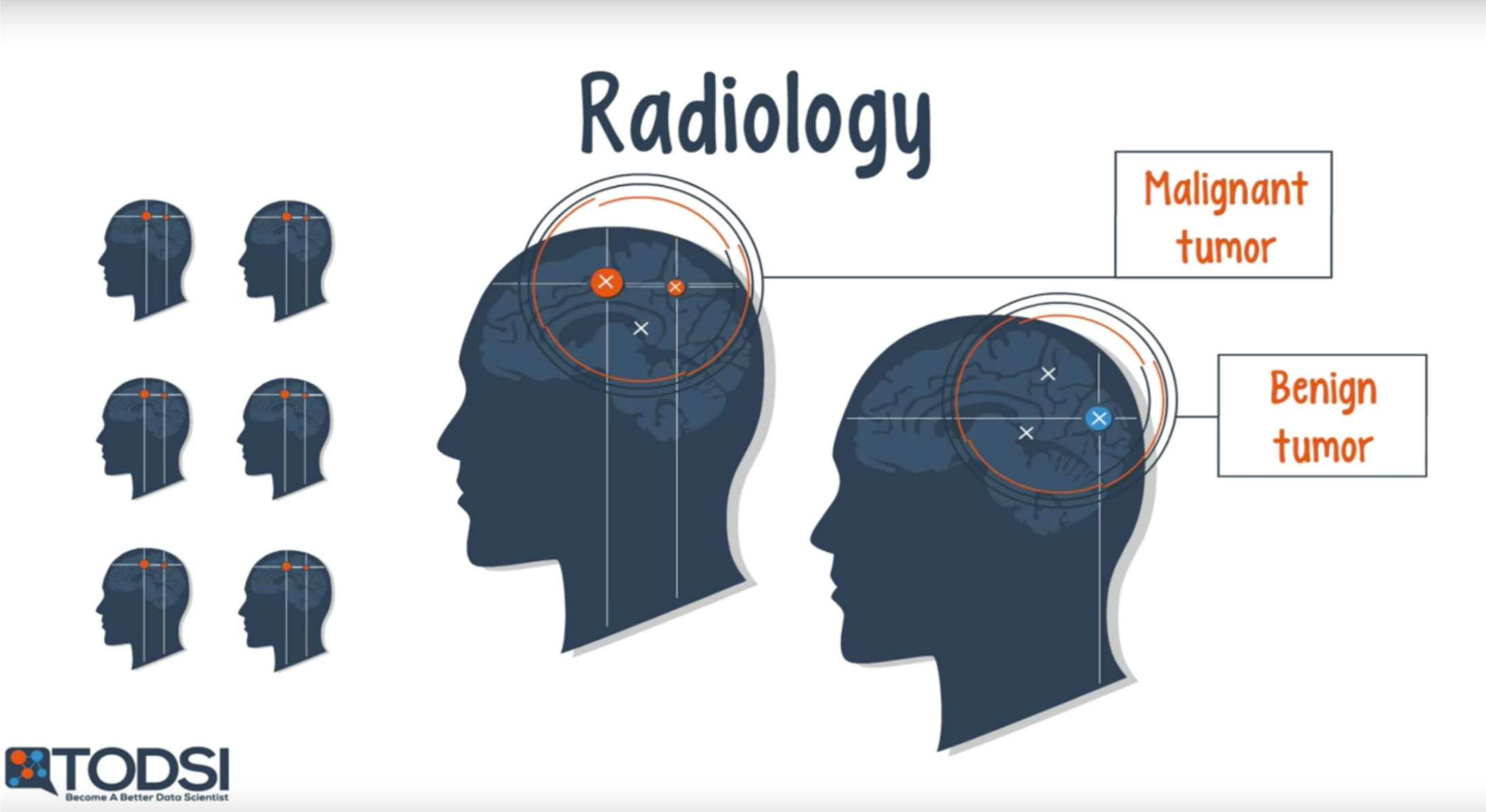
```
train_x_batch, train_y_batch = \
   tf.train.batch([xy[0:-1], xy[-1:]], batch_size=10)

sess = tf.Session()

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)
```

shuffle_batch

```
min_after_dequeue = 10000
capacity = min_after_dequeue + 3 * batch_size
example_batch, label_batch = tf.train.shuffle_batch(
   [example, label], batch_size=batch_size, capacity=capacity,
   min_after_dequeue=min_after_dequeue)
```

Logistic Regression

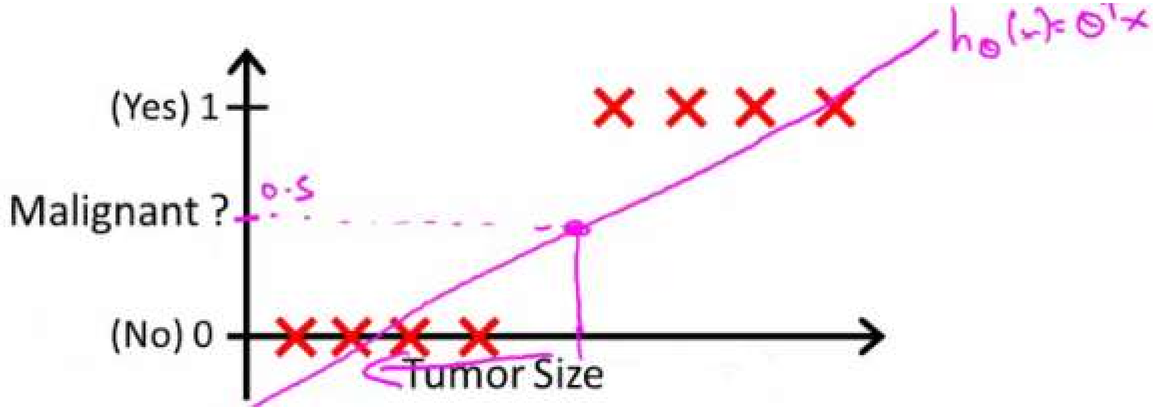
Tumor Malignancy



<“Deep Learning Use Cases” (https://www.youtube.com/watch?v=BmkA1ZsG2P4)>

Binary Classification (Logistic Regression)

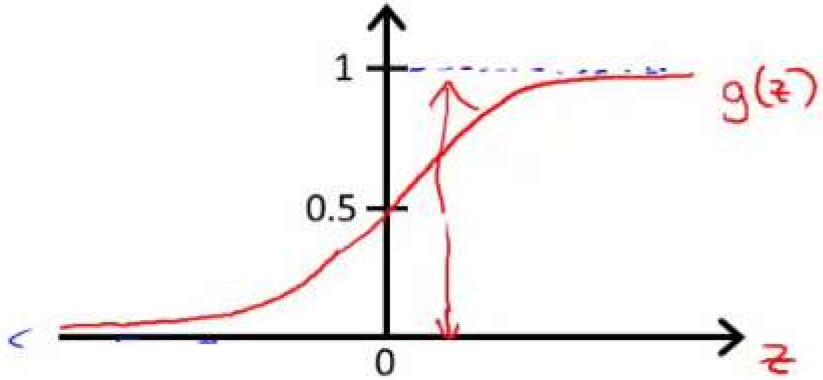
- Tumor size vs. malignancy (0, 1)



Linear regression work?

- Logistic hypothesis

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

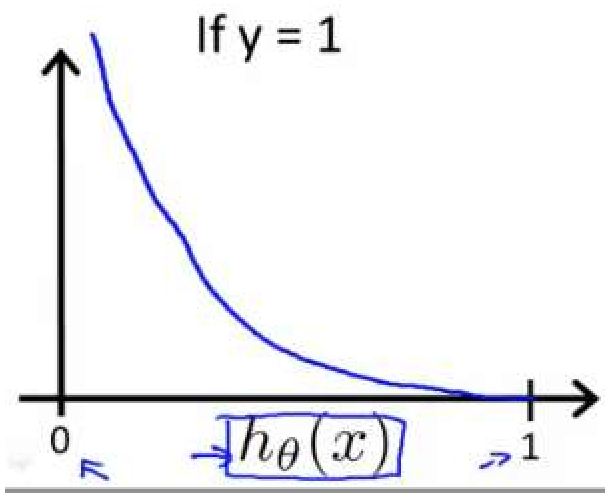


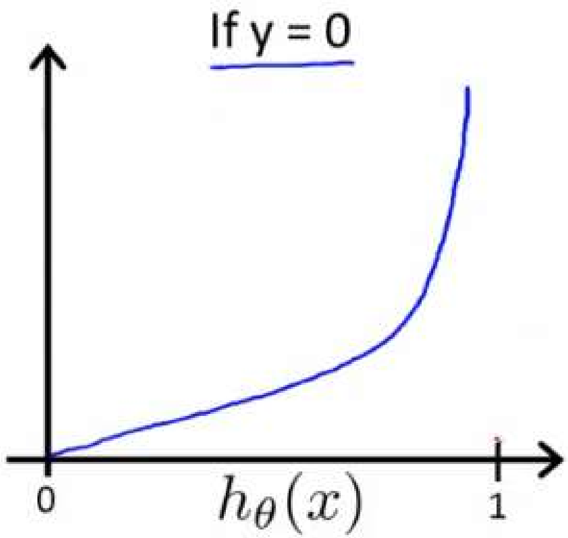
<http://www.holehouse.org/mlclass/>

Cost Function

- Convex logistic regression cost function

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$





$$C(h_\theta(x), y) = -y\log\big(h_\theta(x)\big) - (1-y)\log(1 - h_\theta(x))$$

Building Graph

```
x_data = [[1, 2],[2, 3],[3, 1],[4, 3],[5, 3],[6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]

x = tf.placeholder(tf.float32, shape=[None, 2])
y = tf.placeholder(tf.float32, shape=[None, 1])
w = tf.Variable(tf.random_normal([2, 1]), name="weight")
b = tf.Variable(tf.random_normal([1]), name="bias")
```

* Cost function

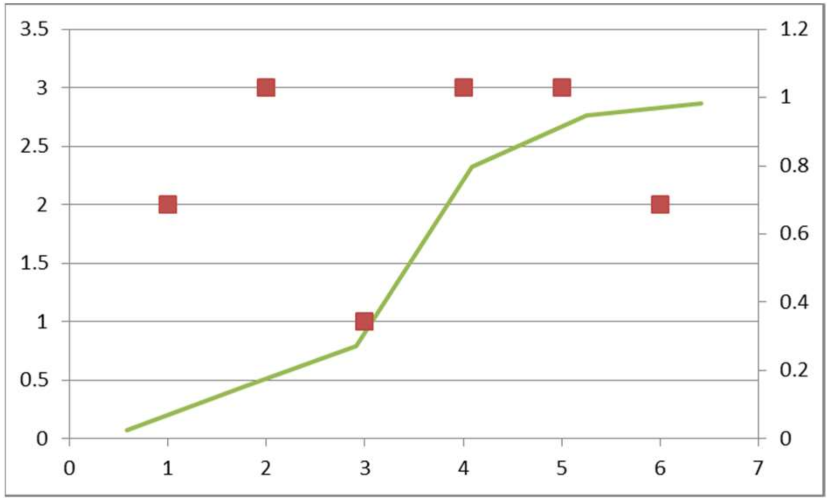
```
hypothesis = tf.sigmoid(tf.matmul(x, w) + b)
cost = -tf.reduce_mean(y*tf.log(hypothesis) + (1-y)*
                       tf.log(1-hypothesis))
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)
```

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

Training Output

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

   h, c, a = sess.run([hypothesis, predicted, accuracy],
                        feed_dict={x: x_data, y: y_data})
   print("\nHypothesis: ",h,"\nCorrect (y): ",c,"\nAccuracy: ", a)
```




```
0.02434957   -> 0
0.1491625    -> 0
0.27264518   -> 0
0.7965147    -> 1
0.94870824   -> 1
0.98327523   -> 1
```

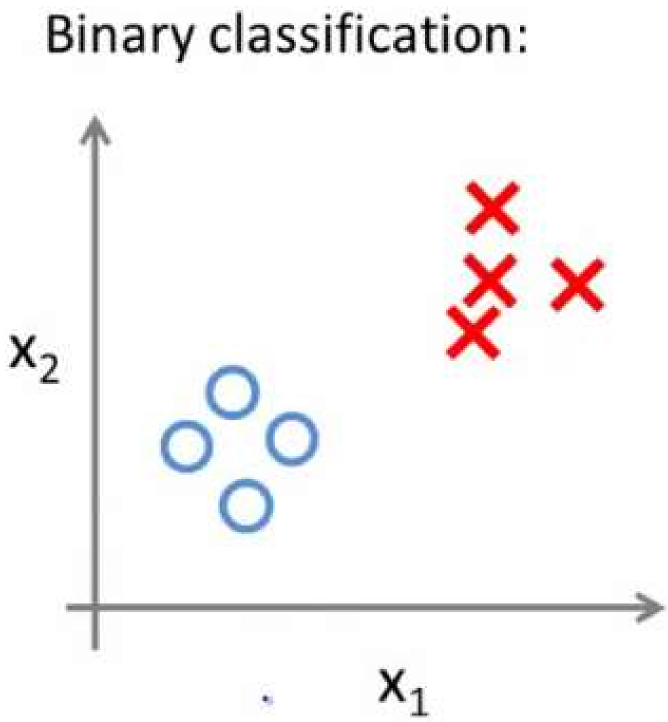
Classifying Diabetes

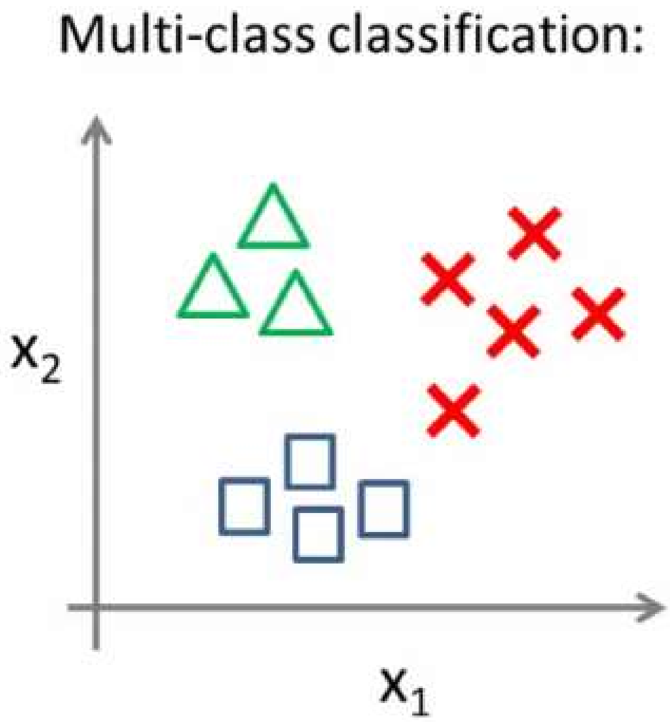
data-03-diabetes.csv

-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

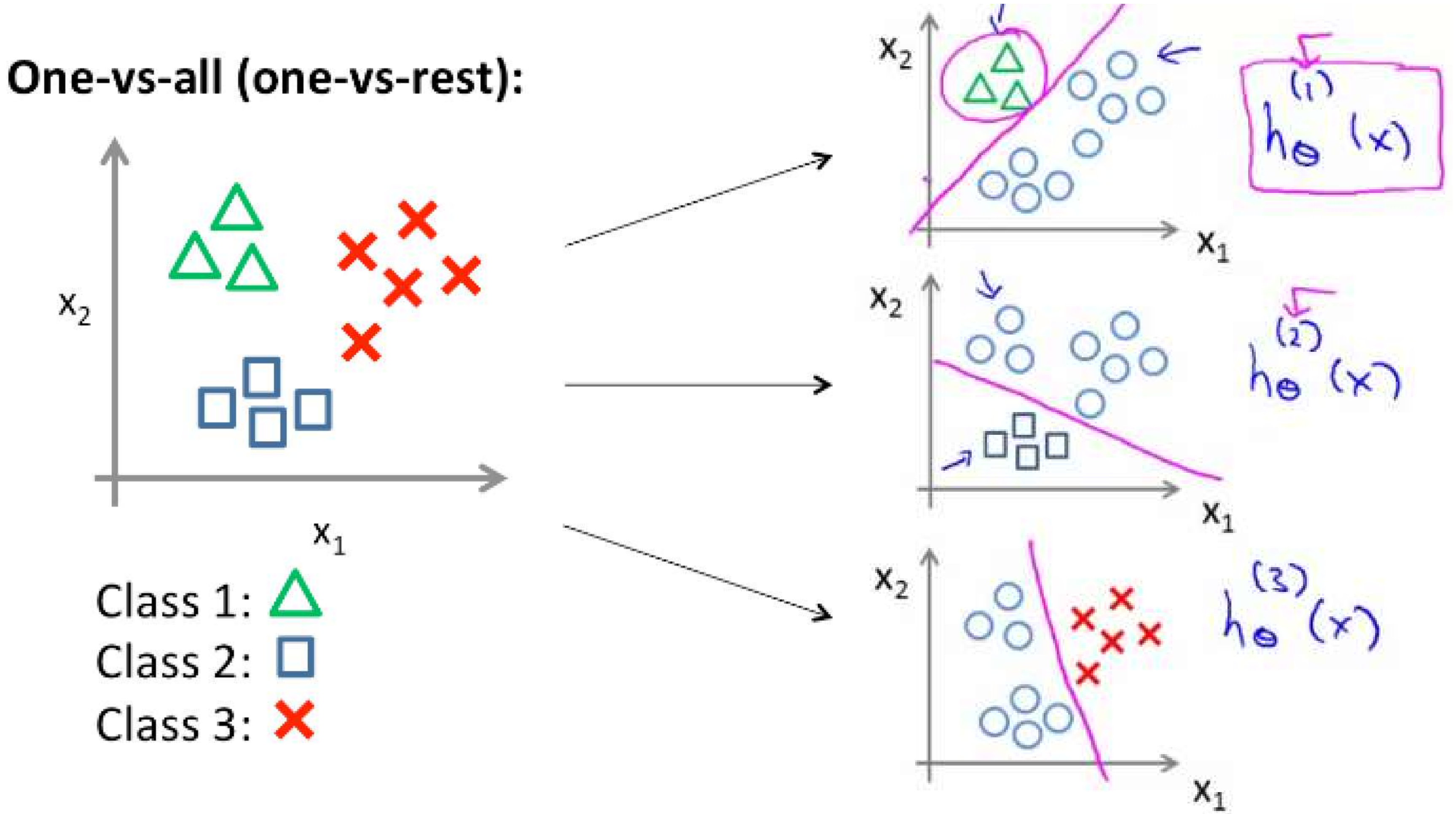
```
xy = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xt[:, [-1]]
```

Multi-class Classification





One-vs-all Classification



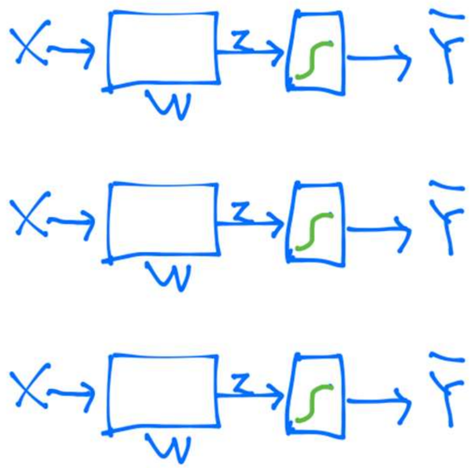
<http://www.holehouse.org/mlclass/>

Multidimensional Classification

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

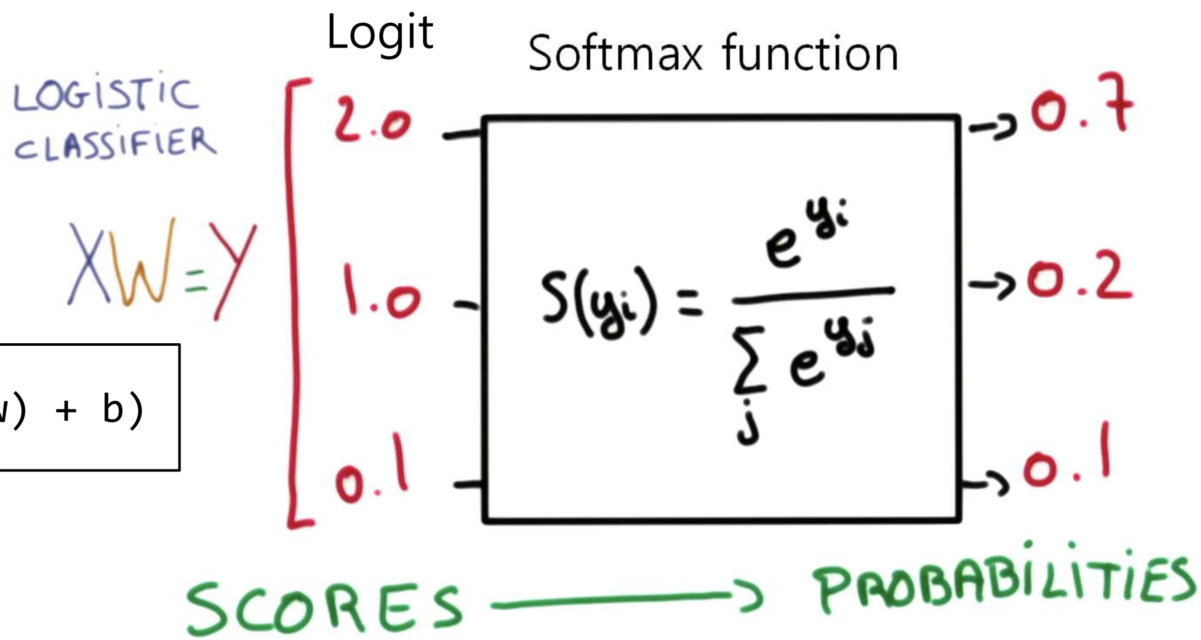
$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 \end{bmatrix}$$

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



$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} =$$

Softmax Regression



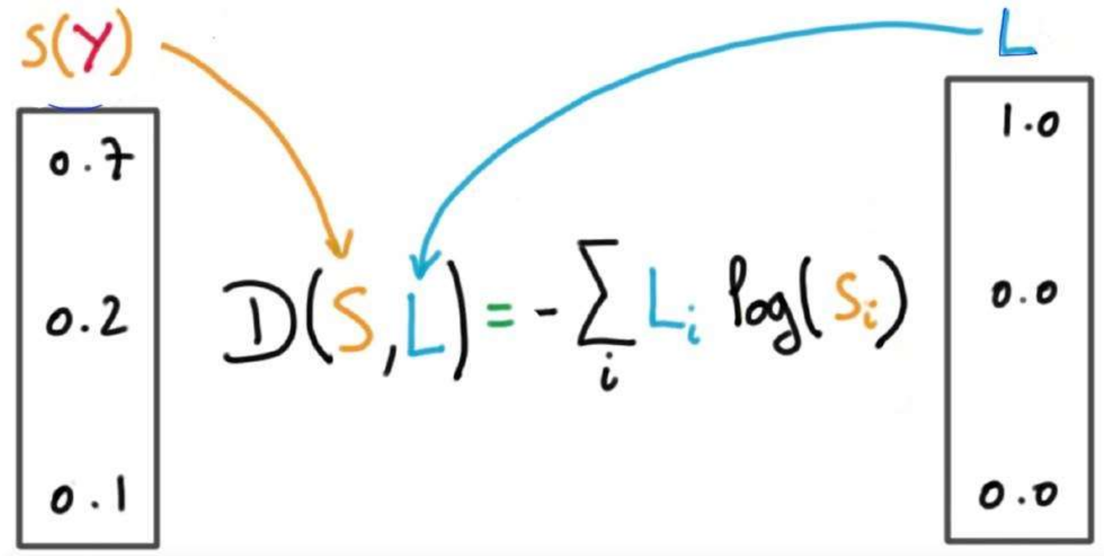

```
tf.matmul(x, w) + b)
```

```
hypothesis = tf.nn.softmax(tf.matmul(x, w) + b)
```

Cost Function

- Cross-entropy function

one-hot encoding



$$C(\theta) = \frac{1}{N}\sum_i D(S(\theta^T x_i), L_i)$$

```
# Cross entropy cost function
cost = tf.reduce_mean(-tf.reduce_mean(y*tf.log(hypothesis, axis=1))
```

Building Graph

```
x_data=[[1,2,1,1],[2,1,3,2],[3,1,3,4],[4,1,5,5],[1,7,5,5],
          [1,2,5,6],[1,6,6,6,],[1,7,7,7]]
y_data=[[0,0,1],[0,0,1],[0,0,1],[0,1,0],[0,1,0],[0,1,0],[1,0,0],[1,0,0]]
x = tf.placeholder("float32", [None, 4])
y = tf.placeholder("float32", [None, 3])
nb_classes = 3
```

(Y의 갯수)

```
w = tf.Variable(tf.random_normal([4, nb_classes]), name="weight")
b = tf.Variable(tf.random_normal([nb_classes]), name="bias")
```

```
hypothesis = tf.nn.softmax(tf.matmul(x, w) + b)
cost = tf.reduce_mean(-tf.reduce_mean(y*tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).
                                         minimize(cost)
```

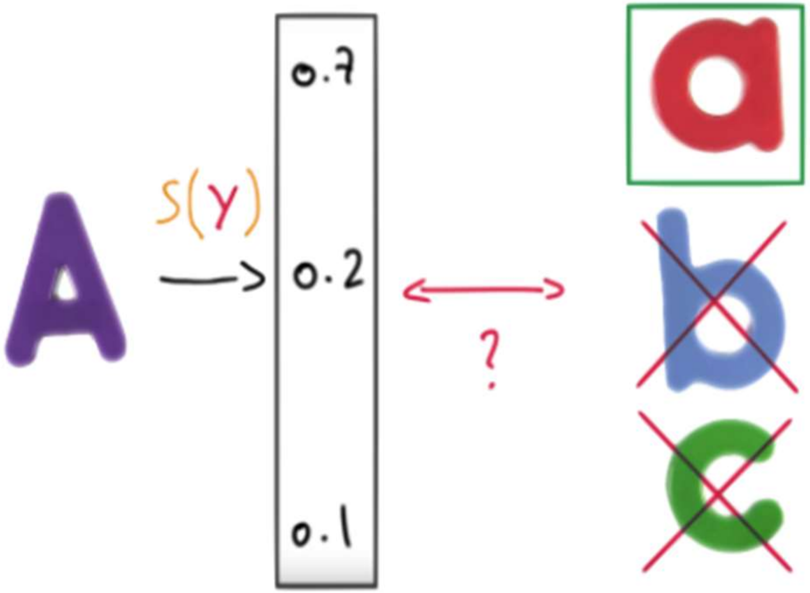
```
with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())
   for step in range(2001):
      sess.run(optimizer, feed_dict={x: x_data, y: y_data})
      if step % 200 == 0:
         print(step, sess.run(cost, feed_dict={x: x_data, y: y_data}))
```

Test Output

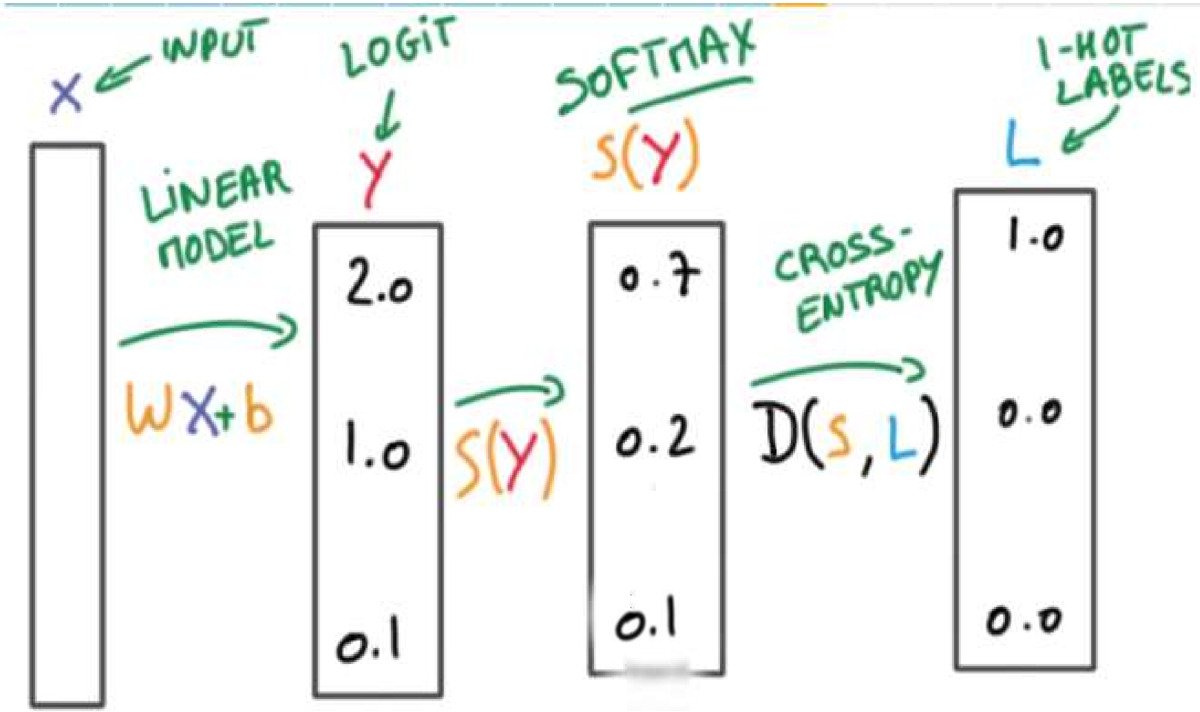
```
a = sess.run(hypothesis, feed_dict={x: [[1,11,7,9]]})
print(a, sess.run(tf.arg_max(a, 1)))
```

```
[0.95726204 0.04099095 0.00174696] -> [0]
```

첫번째 class



softmax_cross_entropy_with_logits




```
logits = tf.matmul(x, w) + b
```

1

```
cost = tf.reduce_mean(-tf.reduce_mean(y*tf.log(hypothesis), axis=1)
```

2

```
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                  labels=y_one_hot)
cost = tf.reduce_mean(cost_i)
```