Multi-variable linear regression

Lecture 04

Hypothesis using matrix

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

Scores for exam

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

Hypothesis using matrix

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

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 Scores for General Psychology

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

```
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.]

# placeholders for a tensor that will be always fed.
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 = x1 * w1 + x2 * w2 + x3 * w3 + b
```

```
import tensorflow as tf
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.]

# placeholders for a tensor that will be always fed.
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 = x1 * w1 + x2 * w2 + x3 * w3 + b

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize. Need a very small learning rate for this data set
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(201):
    cost_val, hy_val, _ = sess.run([cost, hypothesis, train],
                                   feed_dict={x1: x1_data, x2: x2_data, x3: x3_data, Y: y_data})
    if step % 10 == 0:
        print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

0 Cost: 183483.22
Prediction:
[-224.89159 -274.88824 -268.49365 -291.39618 -211.79367]
10 Cost: 13.319193
Prediction:
[154.009 180.54523 180.24066 197.2651 135.59247]
20 Cost: 11.583258
Prediction:
[155.14319 181.93164 181.5947 198.74045 136.65503]
30 Cost: 11.527212
Prediction:
[155.13443 181.94423 181.59512 198.74173 136.6697]
...

180 Cost: 10.721645
Prediction:
[154.95505 182.068 181.54114 198.69507 136.83853]
190 Cost: 10.67021
Prediction:
[154.94334 182.07607 181.53764 198.69203 136.84956]
200 Cost: 10.619032
Prediction:
[154.93169 182.08412 181.5341 198.68898 136.86055]

```
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.]
```

```
import tensorflow as tf
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.]

# placeholders for a tensor that will be always fed.
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 = x1 * w1 + x2 * w2 + x3 * w3 + b

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize. Need a very small learning rate for this data set
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())
```

Print

Next slide

true value : [152.0, 185.0, 180.0, 196.0, 142.0]

```
0       Cost:  68719.125     Prediction: [ -80.06516   -95.63857   -94.42629  -103.596535  -72.279   ]
10      Cost:  1.3717095     Prediction: [151.827   183.08395 180.20096 195.46593 140.31721]
20      Cost:  0.7396033     Prediction: [152.52573 183.92926 181.03099 196.37012 140.96307]
30      Cost:  0.7366053     Prediction: [152.52498 183.93379 181.03262 196.37221 140.96764]
40      Cost:  0.7336332     Prediction: [152.52211 183.93575 181.03174 196.37155 140.97025]
50      Cost:  0.73065394    Prediction: [152.51924 183.93773 181.03087 196.37088 140.97287]
60      Cost:  0.7277094     Prediction: [152.51639 183.9397  181.03    196.37024 140.97546]
70      Cost:  0.72477674    Prediction: [152.51353 183.94168 181.02916 196.36958 140.97806]
80      Cost:  0.721859      Prediction: [152.51068 183.9436  181.02826 196.36893 140.98064]
90      Cost:  0.7189571     Prediction: [152.50786 183.94556 181.02739 196.36829 140.98322]
100     Cost:  0.71607697    Prediction: [152.50502 183.94748 181.02652 196.36763 140.98578]
110     Cost:  0.7132072     Prediction: [152.5022  183.94943 181.02568 196.36697 140.98834]
120     Cost:  0.7103611     Prediction: [152.49939 183.95135 181.02483 196.36632 140.99089]
130     Cost:  0.7075125     Prediction: [152.49658 183.95328 181.02396 196.36568 140.99345]
140     Cost:  0.7046921     Prediction: [152.4938  183.95522 181.02312 196.36505 140.996  ]
.
.
.
410     Cost:  0.63397604    Prediction: [152.42107 184.00511 181.0009  196.34828 141.06209]
420     Cost:  0.63154304    Prediction: [152.41849 184.00691 181.00012 196.3477  141.06447]
430     Cost:  0.6291391     Prediction: [152.4159  184.00867 180.99933 196.34709 141.0668 ]
440     Cost:  0.6267311     Prediction: [152.41333 184.01045 180.99854 196.3465  141.06915]
450     Cost:  0.6243508     Prediction: [152.41075 184.0122  180.99776 196.3459  141.07149]
460     Cost:  0.6219795     Prediction: [152.40819 184.01396 180.99698 196.3453  141.07382]
470     Cost:  0.6196192     Prediction: [152.40564 184.01572 180.9962  196.34473 141.07616]
480     Cost:  0.6172741     Prediction: [152.40308 184.01747 180.99542 196.34415 141.07846]
490     Cost:  0.6149209     Prediction: [152.40054 184.01924 180.99464 196.34355 141.0808 ]
500     Cost:  0.6126008     Prediction: [152.398   184.02097 180.99387 196.34296 141.0831 ]
```

Matrix

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

$$H(X) = XW$$

Matrix

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

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

```
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.]
```

```
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
```

```
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(201):
    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)
```

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

```
0 Cost:  10480.361
Prediction:
 [[65.108696]
 [72.39651 ]
 [74.32275 ]
 [81.11035 ]
 [53.65494 ]]
10 Cost:  12.366552
Prediction:
 [[155.6036 ]
 [181.18958]
 [181.50548]
 [197.8308 ]
 [136.6419 ]]
20 Cost:  12.207312
Prediction:
 [[155.86429]
 [181.52776]
 [181.82576]
 [198.18076]
 [136.90508]]
.
.
.
.
180 Cost:  11.237604
Prediction:
 [[155.65942]
 [181.67027]
 [181.76436]
 [198.13177]
 [137.0957 ]]
190 Cost:  11.179713
Prediction:
 [[155.64687]
 [181.67892]
 [181.76056]
 [198.12872]
 [137.10732]]
200 Cost:  11.122178
Prediction:
 [[155.63435]
 [181.68753]
 [181.75677]
 [198.12567]
 [137.1189 ]]
```

```
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.]
```

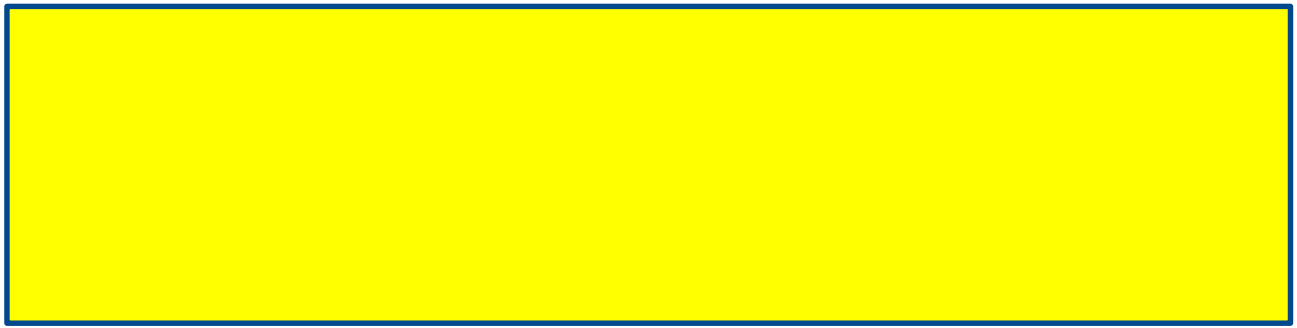
reshape

```
import numpy as np
x = np.arange(12).reshape(3, 4)
print("x.shape :",x.shape)
print("x : \n", x)
print("x.reshape \n", x.reshape(-1,2))
print("x.reshape \n", x.reshape(2,-1))
```

```
x.shape : (3, 4)
x :
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
x.reshape
 [[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]]
x.reshape
 [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

```
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())
```

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



Print

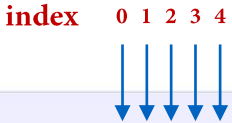
Next slide

```
true value :  [[152.0], [185.0], [180.0], [196.0], [142.0]]
0 Cost:  11856.827
hypothesis shape : (5, 1)
hypothesis prediction
[[56.847008]
 [67.03378 ]
 [66.55113 ]
 [73.52583 ]
 [50.111755]]
hypothesis (Convert to row vector) ->  [[56.847008 67.03378  66.55113  73.52583  50.111755]]
================================================================================================
true value :  [[152.0], [185.0], [180.0], [196.0], [142.0]]
10 Cost:  4.4845324
hypothesis shape : (5, 1)
hypothesis prediction
[[153.14717]
 [182.79272]
 [180.6035 ]
 [197.72633]
 [138.40971]]
hypothesis (Convert to row vector) ->  [[153.14717 182.79272 180.6035  197.72633 138.40971]]
================================================================================================
true value :  [[152.0], [185.0], [180.0], [196.0], [142.0]]
20 Cost:  4.3570404
hypothesis shape : (5, 1)
hypothesis prediction
[[153.43153]
 [183.1478 ]
 [180.94647]
 [198.10019]
 [138.68358]]
hypothesis (Convert to row vector) ->  [[153.43153 183.1478  180.94647 198.10019 138.68358]]
================================================================================================
.
.
```

```
true value :  [[152.0], [185.0], [180.0], [196.0], [142.0]]
180 Cost:  4.0664644
hypothesis shape : (5, 1)
hypothesis prediction
[[153.32271]
 [183.22476]
 [180.91478]
 [198.07086]
 [138.7897 ]]
hypothesis (Convert to row vector) ->  [[153.32271 183.22476 180.91478 198.07086 138.7897 ]]
================================================================================================
true value :  [[152.0], [185.0], [180.0], [196.0], [142.0]]
190 Cost:  4.049096
hypothesis shape : (5, 1)
hypothesis prediction
[[153.31602]
 [183.2294 ]
 [180.91278]
 [198.069  ]
 [138.79616]]
hypothesis (Convert to row vector) ->  [[153.31602 183.2294  180.91278 198.069   138.79616]]
================================================================================================
true value :  [[152.0], [185.0], [180.0], [196.0], [142.0]]
200 Cost:  4.03183
hypothesis shape : (5, 1)
hypothesis prediction
[[153.30934]
 [183.23402]
 [180.91078]
 [198.06712]
 [138.80257]]
hypothesis (Convert to row vector) ->  [[153.30934 183.23402 180.91078 198.06712 138.80257]]
================================================================================================
```

Loading Data from File

Slicing




```
nums = [ 0,1,2,3,4]
nums
nums[2:4]
nums[2:]
nums[:2]
nums[:]
nums[:-1]
nums[2:4]=[8,9]
nums
```

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

Slicing

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

비교

```
c=[[1,2,3,4], [5,6,7,8], [9,10,11,12]]
print (c)
type(c)
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
list
```

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

1	2	3	4
5	6	7	8
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]])
```

tf.random_

균등한 확률로 랜덤값이 나오게 하려면 → uniform
범위내에서 종모양의 정규분포에 따라 값이 랜덤값이 나오게 하려면 → normal

```
import tensorflow as tf
a=tf.random_uniform([1])
b=tf.random_normal([1])
print("Session 1")
with tf.Session() as sess1:
    print(sess1.run(a))
    print(sess1.run(a))
    print(sess1.run(b))
    print(sess1.run(b))

print("Session 2")
with tf.Session() as sess2:
    print(sess2.run(a))
    print(sess2.run(a))
    print(sess2.run(b))
    print(sess2.run(b))
```

```
Session 1
[0.11832285]
[0.905094]
[0.3956182]
[0.4454496]
Session 2
[0.05520594]
[0.1945405]
[0.12527454]
[0.12360418]
```

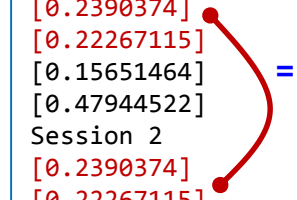
tf.random_

```
import tensorflow as tf
a=tf.random_uniform([1])
b=tf.random_normal([1])
print("Session 1")
with tf.Session() as sess1:
    print(sess1.run(a))
    print(sess1.run(a))
    print(sess1.run(b))
    print(sess1.run(b))

print("Session 2")
with tf.Session() as sess2:
    print(sess2.run(a))
    print(sess2.run(a))
    print(sess2.run(b))
    print(sess2.run(b))
```

```
Session 1
[0.11832285]
[0.905094]
[0.3956182]
[0.4454496]
Session 2
[0.05520594]
[0.1945405]
[0.12527454]
[0.12360418]
```

```
import tensorflow as tf
a=tf.random_uniform([1], seed=1)
b=tf.random_uniform([1])
print("Session 1")
with tf.Session() as sess1:
    print(sess1.run(a))
    print(sess1.run(a))
    print(sess1.run(b))
    print(sess1.run(b))

print("Session 2")
with tf.Session() as sess2:
    print(sess2.run(a))
    print(sess2.run(a))
    print(sess2.run(b))
    print(sess2.run(b))
```

```
Session 1
[0.2390374]
[0.22267115]
[0.15651464]
[0.47944522]
Session 2
[0.2390374]
[0.22267115]
[0.6993104]
[0.39458954]
```



tf.random_

```
import tensorflow as tf
a=tf.random_uniform([2,2],-1,4) #shape, mean, standard deviation
b=tf.random_normal([1,2],-1,4) #shape, mean, standard deviation
print("Session 1","="*20)
with tf.Session() as sess1:
    print("a : ",sess1.run(a))
    print("a : ",sess1.run(a))
    print("b : ",sess1.run(b))
    print("b : ",sess1.run(b))

print("\nSession 2","="*20)
with tf.Session() as sess2:
    print("a : ",sess2.run(a))
    print("a : ",sess2.run(a))
    print("b : ",sess2.run(b))
    print("b : ",sess2.run(b))
```

```
Session 1 ====================
a :  [[ 0.46835685 -0.9140465 ]
 [-0.43962872  2.0696054 ]]
a :  [[-0.24747825  0.44044995]
 [ 3.5657067   2.804232  ]]
b :  [[-4.631923  -3.6227627]]
b :  [[-1.7505416  1.4698434]]

Session 2 ====================
a :  [[-0.59933937 -0.01183367]
 [ 1.99195     2.9339738 ]]
a :  [[ 3.449287    2.597684  ]
 [ 0.67940617 -0.03768122]]
b :  [[-1.7212455  0.7322532]]
b :  [[1.9086502 1.4359376]]
```

tf.random_

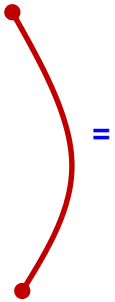
```
import tensorflow as tf
a=tf.random_uniform([2,2],-1,4,seed=7) #shape, mean, standard deviation
b=tf.random_normal([1,2],-1,4) #shape, mean, standard deviation
print("Session 1","="*20)
with tf.Session() as sess1:
    print("a : ",sess1.run(a))
    print("a : ",sess1.run(a))
    print("b : ",sess1.run(b))
    print("b : ",sess1.run(b))

print("\nSession 2","="*20)
with tf.Session() as sess2:
    print("a : ",sess2.run(a))
    print("a : ",sess2.run(a))
    print("b : ",sess2.run(b))
    print("b : ",sess2.run(b))
```

```
Session 1 ====================
a :  [[ 1.4197049   1.5310154 ]
 [ 0.2757188  -0.37628865]]
a :  [[ 0.52801156  3.8846369 ]
 [-0.3537575  -0.52644885]]
b :  [[-3.5266707  7.6168528]]
b :  [[-3.800884  -3.8644233]]

Session 2 ====================
a :  [[ 1.4197049   1.5310154 ]
 [ 0.2757188  -0.37628865]]
a :  [[ 0.52801156  3.8846369 ]
 [-0.3537575  -0.52644885]]
b :  [[-2.7445269 -0.8103109]]
b :  [[-2.4123158  5.2338667]]
```

tf.random_

```
import tensorflow as tf
tf.set_random_seed(777)   #shape, start value, end value

a=tf.random_uniform([1])   #shape, start value, end value
b=tf.random_normal([1,2])   #shape, start value, end value
print("Session 1","="*20)

with tf.Session() as sess1:
    print("a : ",sess1.run(a))
    print("a : ",sess1.run(a))
    print("b : ",sess1.run(b))
    print("b : ",sess1.run(b))

print("\nSession 2","="*20)
with tf.Session() as sess2:
    print("a : ",sess2.run(a))
    print("a : ",sess2.run(a))
    print("b : ",sess2.run(b))
    print("b : ",sess2.run(b))
```

```
Session 1 ====================
a :  [0.1003145]
a :  [0.4195832]
b :  [[-1.6822096   0.05430611]]
b :  [[-0.28773162 -0.46478754]]

Session 2 ====================
a :  [0.1003145]
a :  [0.4195832]
b :  [[-1.6822096   0.05430611]]
b :  [[-0.28773162 -0.46478754]]
```



Loading data from file

data-01-test-score.csv

x_data | y_data

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

comma-separated values 파일

```
import numpy as np
xy = np.loadtxt('g:\\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.shape, ", len(x_data) : ", len(x_data))
print( x_data)
print("y_data.shape :", y_data.shape, ", len(y_data) : ", len(y_data))
print(y_data)
```

x_data.shape : (25, 3) , len(x_data) : 25

y_data.shape : (25, 1) , len(y_data) : 25

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility 난수발생의 초기값을 주면 실행할 때마다 같은 결과가 나온다.

xy = np.loadtxt('g:\\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.shape, ", len(x_data) : ", len(x_data))
print( x_data)
print("y_data.shape :", y_data.shape, ", len(y_data) : ", len(y_data))
print(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(101):
   cost_val, w_val, hy_val, _  = sess.run([cost, W, hypothesis, train],feed_dict={X: x_data, Y: y_data})
   if step % 10 == 0:
       print("-"*5, "step :",step, "-"*5, "> cost : (", cost_val ,")")
       print("x data : \n", x_data)
       print("W : \n", w_val)
       print("sess.run(W) : \n", sess.run(W))
       print("sess.run(b) : \n", sess.run(b))
       print("hypothesis : \n", hy_val)



print("="*30)
print("Final cost : ", cost_val) # final cost value 확인
print("Final W : \n", w_val)   # final W value 확인
print("Final sess.run(W) : \n", sess.run(W)) # W 확인
Print("Final sess.run(b) : \n", sess.run(b)) # b 확인
print("Final hypothesis : \n", hy_val) # final hypothesis value 확인

# Ask my score
```

~~~~~~

If the input data value then [100, 70, 101]
score will be
[[216.60124]]

If the input data value then [60, 70, 110], [90, 100, 80]
scores will be
[[171.78261]
[166.04927]]
~~~~~~

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

```
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
```

```
x_data.shape : (25, 3) , len(x_data) :  25
[[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]
 [ 69.  74.  77.]
 [ 47.  56.  60.]
 [ 87.  79.  90.]
 [ 79.  70.  88.]
 [ 69.  70.  73.]
 [ 70.  65.  74.]
 [ 93.  95.  91.]
 [ 79.  80.  73.]
 [ 70.  73.  78.]
 [ 93.  89.  96.]
 [ 78.  75.  68.]
 [ 81.  90.  93.]
 [ 88.  92.  86.]
 [ 78.  83.  77.]
 [ 82.  86.  90.]
 [ 86.  82.  89.]
 [ 78.  83.  85.]
 [ 76.  83.  71.]
 [ 96.  93.  95.]]
```

```
y_data.shape : (25, 1) , len(y_data) :  25
[[152.]
 [185.]
 [180.]
 [196.]
 [142.]
 [101.]
 [149.]
 [115.]
 [175.]
 [164.]
 [141.]
 [141.]
 [184.]
 [152.]
 [148.]
 [192.]
 [147.]
 [183.]
 [177.]
 [159.]
 [177.]
 [175.]
 [175.]
 [149.]
 [192.]]
```

```
print("-"*5, "step :",step, "-"*5, "> cost : (", cost_val ,")")
print("x data : \n", x_data)
print("W : \n", w_val)
print("sess.run(W) : \n", sess.run(W))
print("sess.run(b) : \n", sess.run(b))
print("hypothesis : \n", hy_val)
```

```
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
```

```
----- step : 0 ----- > cost : ( 6124.322 )
x data :
 [[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]
 [ 69.  74.  77.]
 [ 47.  56.  60.]
 [ 87.  79.  90.]
 [ 79.  70.  88.]
 [ 69.  70.  73.]
 [ 70.  65.  74.]
 [ 93.  95.  91.]
 [ 79.  80.  73.]
 [ 70.  73.  78.]
 [ 93.  89.  96.]
 [ 78.  75.  68.]
 [ 81.  90.  93.]
 [ 88.  92.  86.]
 [ 78.  83.  77.]
 [ 82.  86.  90.]
 [ 86.  82.  89.]
 [ 78.  83.  85.]
 [ 76.  83.  71.]
 [ 96.  93.  95.]]
```

```
W :
 [[ 1.5859851 ]
 [-0.29999283]
 [ 1.3056532 ]]
sess.run(W) :
 [[ 1.5859851 ]
 [-0.29999283]
 [ 1.3056532 ]]
sess.run(b) :
 [1.2055854]
```

```
hypothesis :
 [[74.180595]
 [80.63322 ]
 [83.980225]
 [91.3227  ]
 [59.210712]
 [41.890907]
 [70.11859 ]
 [54.823997]
 [72.922676]
 [66.100105]
 [65.148254]
 [60.243664]
 [86.905014]
 [71.78623 ]
 [69.10599 ]
 [82.41498 ]
 [65.882515]
 [86.28957 ]
 [84.18406 ]
 [76.09278 ]
 [81.279625]
 [75.88126 ]
 [78.24949 ]
 [75.09364 ]
 [85.10978 ]]
```

```
print("-"*5, "step :",step, "-"*5, "> cost : (", cost_val ,")")
print("x data : \n", x_data)
print("W : \n", w_val)
print("sess.run(W) : \n", sess.run(W))
print("sess.run(b) : \n", sess.run(b))
print("hypothesis : \n", hy_val)
```

```
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
```

```
----- step : 100 ----- > cost : ( 35.186806 )
x data :
 [[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]
 [ 69.  74.  77.]
 [ 47.  56.  60.]
 [ 87.  79.  90.]
 [ 79.  70.  88.]
 [ 69.  70.  73.]
 [ 70.  65.  74.]
 [ 93.  95.  91.]
 [ 79.  80.  73.]
 [ 70.  73.  78.]
 [ 93.  89.  96.]
 [ 78.  75.  68.]
 [ 81.  90.  93.]
 [ 88.  92.  86.]
 [ 78.  83.  77.]
 [ 82.  86.  90.]
 [ 86.  82.  89.]
 [ 78.  83.  85.]
 [ 76.  83.  71.]
 [ 96.  93.  95.]]
```

```
W :
 [[ 1.3675066]
 [-0.4606597]
 [ 1.0979582]]
sess.run(W) :
 [[ 1.3675066]
 [-0.4606597]
 [ 1.0979582]]
sess.run(b) :
 [1.202987]
```

```
hypothesis :
 [[146.52287]
 [189.95477]
 [179.80666]
 [197.13441]
 [147.48653]
 [112.88066]
 [146.01387]
 [105.55443]
 [182.60304]
 [173.61385]
 [143.46568]
 [148.23643]
 [184.53175]
 [152.53331]
 [148.94067]
 [192.78792]
 [147.9803 ]
 [172.61974]
 [173.58574]
 [154.17473]
 [172.53734]
 [178.75427]
 [162.95908]
 [144.85098]
 [193.94914]]
```

```
print("="*30)
print("Final cost : ", cost_val)
print("Final W : \n", w_val)
print("Final sess.run(W) : \n", sess.run(W))
print("Final sess.run(b) : \n", sess.run(b))
print("Final hypothesis : \n", hy_val)
```

```
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
```

```
==============================
Final cost :  35.186806
Final W :
 [[ 1.3675066]
 [-0.4606597]
 [ 1.0979582]]
Final sess.run(W) :
 [[ 1.3675066]
 [-0.4606597]
 [ 1.0979582]]
Final sess.run(b) :
 [1.202987]
```

```
Final hypothesis :
 [[146.52287]
 [189.95477]
 [179.80666]
 [197.13441]
 [147.48653]
 [112.88066]
 [146.01387]
 [105.55443]
 [182.60304]
 [173.61385]
 [143.46568]
 [148.23643]
 [184.53175]
 [152.53331]
 [148.94067]
 [192.78792]
 [147.9803 ]
 [172.61974]
 [173.58574]
 [154.17473]
 [172.53734]
 [178.75427]
 [162.95908]
 [144.85098]
 [193.94914]]
```

```
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
```

```
========================
Final cost :  35.186806
Final W :
 [[ 1.3675066]
 [-0.4606597]
 [ 1.0979582]]
Final sess.run(W) :
 [[ 1.3675066]
 [-0.4606597]
 [ 1.0979582]]
Final sess.run(b) :
 [1.202987]
```

```
Final hypothesis :
[[146.52287]
 [189.95477]
 [179.80666]
 [197.13441]
 [147.48653]
 [112.88066]
 [146.01387]
 [105.55443]
 [182.60304]
 [173.61385]
 [143.46568]
 [148.23643]
 [184.53175]
 [152.53331]
 [148.94067]
 [192.78792]
 [147.9803 ]
 [172.61974]
 [173.58574]
 [154.17473]
 [172.53734]
 [178.75427]
 [162.95908]
 [144.85098]
 [193.94914]]
```

~~~~~~

If the input data value then [100, 70, 101]
score will be
[[216.60124]]

If the input data value then [60, 70, 110], [90, 100, 80]
scores will be
[[171.78261]
[166.04927]]
~~~~~~

data-01-test-score.csv 파일을 이용한 Self Test

Print
Next slide

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

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

```
print(step, "Cost: ", cost_val,
                "\nPrediction:\n", hy_val)
```

(25, 3) [[73. 80. 75.]
 [93. 88. 93.]
 [89. 91. 90.]
 [96. 98. 100.]
 [73. 66. 70.]
 [53. 46. 55.]
 [69. 74. 77.]
 [47. 56. 60.]
 [87. 79. 90.]
 [79. 70. 88.]
 [69. 70. 73.]
 [70. 65. 74.]
 [93. 95. 91.]
 [79. 80. 73.]
 [70. 73. 78.]
 [93. 89. 96.]
 [78. 75. 68.]
 [81. 90. 93.]
 [88. 92. 86.]
 [78. 83. 77.]
 [82. 86. 90.]
 [86. 82. 89.]
 [78. 83. 85.]
 [76. 83. 71.]
 [96. 93. 95.]] 25

(25, 1) [[152.]
 [185.]
 [180.]
 [196.]
 [142.]
 [101.]
 [149.]
 [115.]
 [175.]
 [164.]
 [141.]
 [141.]
 [184.]
 [152.]
 [148.]
 [192.]
 [147.]
 [183.]
 [177.]
 [159.]
 [177.]
 [175.]
 [175.]
 [149.]
 [192.]]

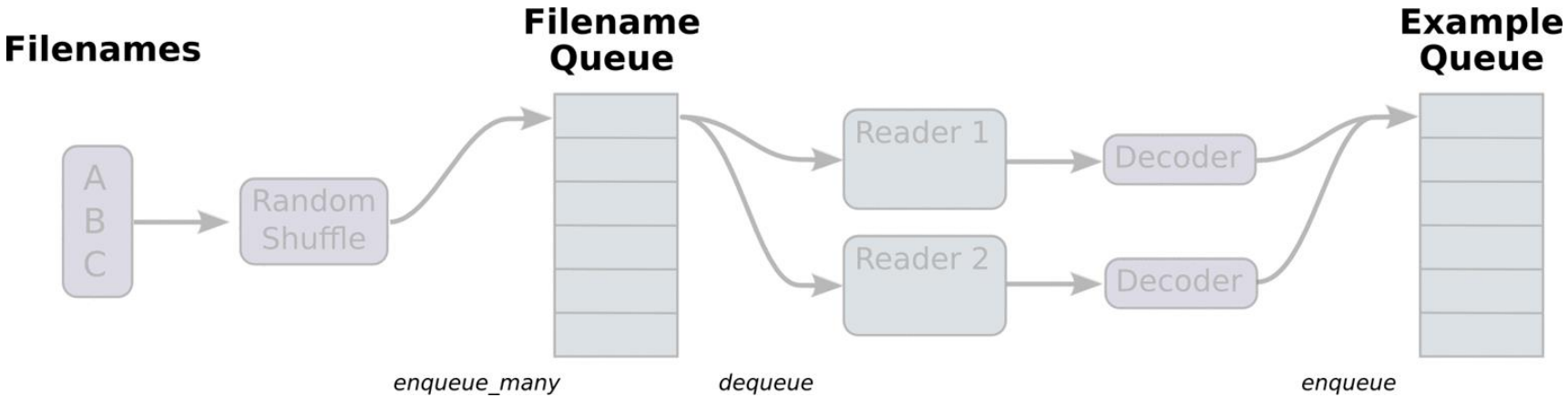
0 Cost: 6794.88 Prediction:	10 Cost: 30.941683 Prediction:	100 Cost: 29.279634 Prediction:
[[75.121445]	[[151.91267]	[[152.4106]
[94.47313]	[186.7695]	[187.36853]
[91.04281]	[181.983]	[182.56651]
[97.97375]	[197.0233]	[197.73706]
[74.02787]	[144.41174]	[144.80415]
[53.175064]	[105.07982]	[105.49004]
[70.35249]	[144.5075]	[145.15811]
[47.876087]	[102.86822]	[103.51313]
[87.78398]	[174.05234]	[174.708]
[79.027596]	[158.95253]	[159.73297]
[70.23902]	[141.67464]	[142.2212]
[70.62783]	[141.06839]	[141.63762]
[95.368126]	[189.31291]	[189.83539]
[81.245766]	[159.32883]	[159.65097]
[71.16429]	[145.6612]	[146.32355]
[94.34674]	[188.01419]	[188.6967]
[80.10899]	[154.4623]	[154.67412]
[82.745514]	[171.74423]	[172.57439]
[90.46315]	[180.02461]	[180.51016]
[80.25856]	[160.39627]	[160.8449]
[83.54958]	[170.50504]	[171.24017]
[87.18368]	[173.77861]	[174.41705]
[79.63953]	[162.54216]	[163.22066]
[78.73298]	[156.136]	[156.45164]
[97.8319]]	[193.4822]]	[194.06593]]

```
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]]}))
```

Your score will be
[[191.01485]]
Other scores will be
[[141.471]
 [184.83368]]

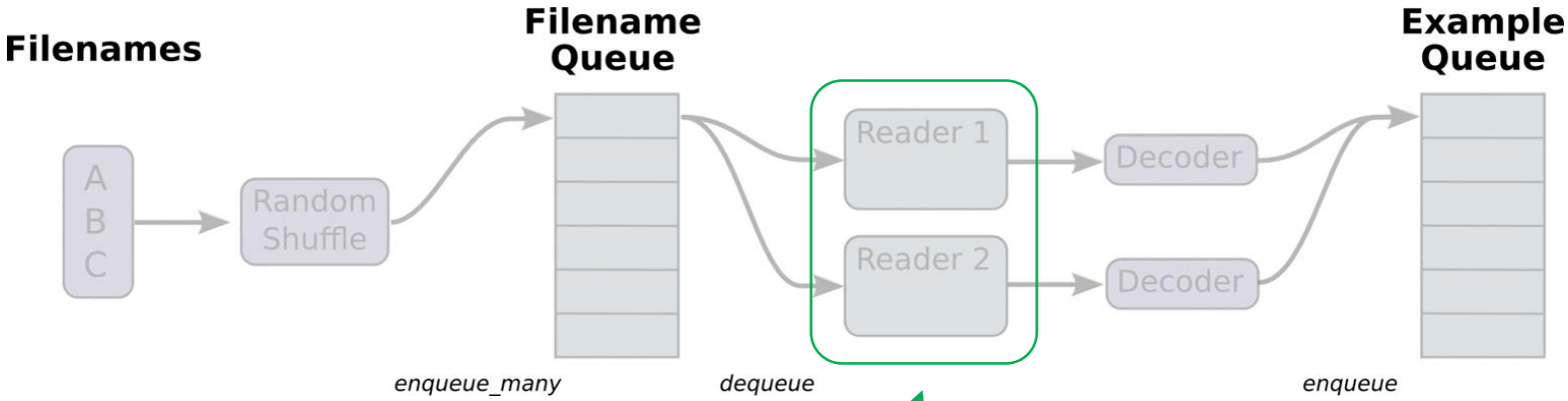
파일이 너무 크면
텐서플로에서는
Queue Runners(큐러너)를
사용한다.

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)
```

tf.train.batch

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

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(101):
   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(
    ['g:\\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(101):
    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)
# 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]]}))
```

```
0 Cost:  16549.555
Prediction:
 [[17.438633]
 [38.414707]
 [28.744263]
 [32.80502 ]
 [32.37227 ]
 [28.1679  ]
 [21.845985]
 [11.90316 ]
 [41.75515 ]
 [43.515163]]
10 Cost:  71.919495
Prediction:
 [[144.80006]
 [191.46127]
 [179.55899]
 [197.05446]
 [149.08711]
 [114.21464]
 [144.80202]
 [103.06957]
 [184.78198]
 [175.99176]]
```

```
.
.
.

100 Cost:  73.38932
Prediction:
 [[145.90152]
 [192.30104]
 [180.63896]
 [198.19327]
 [149.63803]
 [114.50667]
 [145.73486]
 [103.88781]
 [185.40894]
 [176.4467 ]]
```

```
Your score will be  [[222.69714]]
Other scores will be  [[169.29909]
 [163.73312]]
```

```
import tensorflow as tf
filename_queue = tf.train.string_input_producer(
    ['g:\\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)
```

Print

Next slide

```
coord.request_stop()
coord.join(threads)
# Ask my score
print("X: [[100, 70, 101] -> Your score will be ",
      sess.run(hypothesis, feed_dict={X: [[100, 70, 101]]}))

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

```
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
```

```
========== step : 0 ==========
x_batch
[[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]
 [ 69.  74.  77.]
 [ 47.  56.  60.]
 [ 87.  79.  90.]
 [ 79.  70.  88.]]
y_batch
[[152.]
 [185.]
 [180.]
 [196.]
 [142.]
 [101.]
 [149.]
 [115.]
 [175.]
 [164.]]
0 Cost:  15626.736
Prediction:
 [[33.822662]
 [33.227776]
 [36.44336 ]
 [43.341988]
 [20.089188]
 [18.80383 ]
 [39.361553]
 [39.253216]
 [33.390327]
 [37.8106  ]]
```

```
========== step : 1 ==========
x_batch
[[69. 70. 73.]
 [70. 65. 74.]
 [93. 95. 91.]
 [79. 80. 73.]
 [70. 73. 78.]
 [93. 89. 96.]
 [78. 75. 68.]
 [81. 90. 93.]
 [88. 92. 86.]
 [78. 83. 77.]]
y_batch
[[141.]
 [141.]
 [184.]
 [152.]
 [148.]
 [192.]
 [147.]
 [183.]
 [177.]
 [159.]]
1 Cost:  6966.9272
Prediction:
 [[ 73.37271]
 [ 70.24524]
 [ 88.44955]
 [ 67.67078]
 [ 81.97356]
 [ 92.13124]
 [ 57.6976 ]
 [103.21454]
 [ 84.87416]
 [ 77.22461]]
```

```
========== step : 2 ==========
x_batch
[[ 82.  86.  90.]
 [ 86.  82.  89.]
 [ 78.  83.  85.]
 [ 76.  83.  71.]
 [ 96.  93.  95.]
 [ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]]
y_batch
[[177.]
 [175.]
 [175.]
 [149.]
 [192.]
 [152.]
 [185.]
 [180.]
 [196.]
 [142.]]
2 Cost:  2882.1062
Prediction:
 [[129.36467]
 [119.89173]
 [122.91422]
 [101.43338]
 [127.88004]
 [109.11273]
 [123.7469 ]
 [125.61891]
 [140.46437]
 [ 89.12875]]
```

```
========== step : 3 ==========
x_batch
[[53. 46. 55.]
 [69. 74. 77.]
 [47. 56. 60.]
 [87. 79. 90.]
 [79. 70. 88.]
 [69. 70. 73.]
 [70. 65. 74.]
 [93. 95. 91.]
 [79. 80. 73.]
 [70. 73. 78.]]
y_batch
[[101.]
 [149.]
 [115.]
 [175.]
 [164.]
 [141.]
 [141.]
 [184.]
 [152.]
 [148.]]
3 Cost:  569.48157
Prediction:
 [[ 83.78645 ]
 [132.1648  ]
 [108.04447 ]
 [141.39131 ]
 [137.85143 ]
 [121.363716]
 [117.552574]
 [151.6006  ]
 [120.1795  ]
 [132.00679 ]]
```

```
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
```

```
========== step : 4 ==========
x_batch
[[93. 89. 96.]
 [78. 75. 68.]
 [81. 90. 93.]
 [88. 92. 86.]
 [78. 83. 77.]
 [82. 86. 90.]
 [86. 82. 89.]
 [78. 83. 85.]
 [76. 83. 71.]
 [96. 93. 95.]]
y_batch
[[192.]
 [147.]
 [183.]
 [177.]
 [159.]
 [177.]
 [175.]
 [175.]
 [149.]
 [192.]]
4 Cost:  515.2943
Prediction:
 [[164.57298]
 [115.26143]
 [172.02415]
 [154.17735]
 [139.23433]
 [161.77357]
 [152.17473]
 [153.813  ]
 [130.29564]
 [163.54369]]
```

```
========== step : 99 ==========
x_batch
[[93. 89. 96.]
 [78. 75. 68.]
 [81. 90. 93.]
 [88. 92. 86.]
 [78. 83. 77.]
 [82. 86. 90.]
 [86. 82. 89.]
 [78. 83. 85.]
 [76. 83. 71.]
 [96. 93. 95.]]
y_batch
[[192.]
 [147.]
 [183.]
 [177.]
 [159.]
 [177.]
 [175.]
 [175.]
 [149.]
 [192.]]
99 Cost:  40.23986
Prediction:
 [[186.25519]
 [132.75537]
 [192.32066]
 [174.96257]
 [157.80383]
 [181.72513]
 [172.2168 ]
 [172.84021]
 [148.3078 ]
 [185.77002]]
```

```
========== step : 100 ==========
x_batch
[[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]
 [ 69.  74.  77.]
 [ 47.  56.  60.]
 [ 87.  79.  90.]
 [ 79.  70.  88.]]
y_batch
[[152.]
 [185.]
 [180.]
 [196.]
 [142.]
 [101.]
 [149.]
 [115.]
 [175.]
 [164.]]
100 Cost:  37.926846
Prediction:
 [[156.2176 ]
 [180.52586]
 [181.48537]
 [201.24272]
 [132.52562]
 [101.62663]
 [157.41142]
 [126.56079]
 [171.0205 ]
 [165.17174]]
```

X: [[100, 70, 101] -> Your score will be [[183.92894]]
X: [[60, 70, 110], [90, 100, 80]] -> Other scores will be
[[192.63106]
[173.11311]]