

# ML/DL for Everyone Season2

with  TensorFlow

## 04 - Multi-variable linear regression LAB

Code: <https://github.com/deeplearningzerotoall/TensorFlow>

Slides: <http://bit.ly/2LQMKvk>

Lecturer: 이승준 plusjune@financedata.kr



# Hypothesis using matrix

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

$x_1$	$x_2$	$x_3$	$y$
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

# Hypothesis using matrix

입력 데이터

출력 데이터

$x_1$	$x_2$	$x_3$	$y$
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

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

*# data and Label*

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

*# weights*

```
w1 = tf.Variable(10.)  
w2 = tf.Variable(10.)  
w3 = tf.Variable(10.)  
b  = tf.Variable(10.)
```

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

```
# data and Label
```

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

```
# random weights
```

초기값은 1을 줬는데, 아무 값이나 줘도 괜찮고 보통 랜덤값을 줌

```
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]))
```

```
learning_rate = 0.000001
```

```
for i in range(1000+1):
```

```
    # tf.GradientTape() to record the gradient of the cost function
```

```
    with tf.GradientTape() as tape: 아래에 있는 변수의 정보 tape에 기록
```

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

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

```
    # calculates the gradients of the cost
```

```
    w1_grad, w2_grad, w3_grad, b_grad = tape.gradient(cost, [w1, w2, w3, b])
```

weight를 지속적으로 업데이트

```
    # update w1, w2, w3 and b
```

```
    w1.assign_sub(learning_rate * w1_grad)
```

```
    w2.assign_sub(learning_rate * w2_grad)
```

```
    w3.assign_sub(learning_rate * w3_grad)
```

```
    b.assign_sub(learning_rate * b_grad)
```

```
if i % 50 == 0:
```

```
    print("{:5} | {:.12.4f}".format(i, cost.numpy()))
```

0	11325.9121
50	135.3618
100	11.1817
150	9.7940
200	9.7687
250	9.7587
300	9.7489
350	9.7389
400	9.7292
450	9.7194
500	9.7096
550	9.6999
600	9.6903
650	9.6806
700	9.6709
750	9.6612
800	9.6517
850	9.6421
900	9.6325
950	9.6229
1000	9.6134

tape에 gradient를 호출해서 cost(함수)에 대한 w1, w2, w3, b 변수의 기울기 (gradient) 값을 구한다

Matrix를 이용하면 더 간결하게 표현할 수 있어

# Matrix

x값의 column이 3개 이므로 row가 3개 필요

$$H(X) = XW$$

출력값은 1개

```
data = np.array([
    # X1,    X2,    X3,    y
    [ 73.,   80.,   75.,  152. ],
    [ 93.,   88.,   93.,  185. ],
    [ 89.,   91.,   90.,  180. ],
    [ 96.,   98.,  100.,  196. ],
    [ 73.,   66.,   70.,  142. ]
], dtype=np.float32)
```

```
W = tf.Variable(tf.random_normal([3, 1]))
```

```
b = tf.Variable(tf.random_normal([1]))
```

bias는 1개

```
# hypothesis, prediction function
```

```
def predict(X):
    return tf.matmul(X, W) + b
```

b는 설명을 위해서 추가한 것이고 이후에 생략할 수 있다

```
# slice data
```

```
X = data[:, :-1]
```

행 부분은 :이므로 전체, 열은

```
y = data[:, [-1]]
```

column에서 마지막 '-1'은 마지막 column을 제외한 모든 column

```
data = np.array([
    # X1,    X2,    X3,    y
    [ 73.,   80.,   75.,  152. ],
    [ 93.,   88.,   93.,  185. ],
    [ 89.,   91.,   90.,  180. ],
    [ 96.,   98.,  100.,  196. ],
    [ 73.,   66.,   70.,  142. ]
], dtype=np.float32)
```

```
# slice data
X = data[:, :-1]
y = data[:, [-1]]

W = tf.Variable(tf.random_normal([3, 1]))
b = tf.Variable(tf.random_normal([1]))

learning_rate = 0.000001
```

*# hypothesis, prediction function*

```
def predict(X):
    return tf.matmul(X, W) + b
```

*.matmul 행렬 간 곱*

한 번 도는 것을 n\_epoch라고 한다

```
n_epochs = 2000
for i in range(n_epochs+1):
    # record the gradient of the cost function
    with tf.GradientTape() as tape:
        cost = tf.reduce_mean((tf.square(predict(X) - y)))

    # calculates the gradients of the loss
    W_grad, b_grad = tape.gradient(cost, [W, b])

    # updates parameters (W and b)
    W.assign_sub(learning_rate * W_grad)
    b.assign_sub(learning_rate * b_grad)

    if i % 100 == 0:
        print("{:5} | {:.10.4f}".format(i, cost.numpy()))
```

5행 1열 = 5행 3열 \* W // 따라서 W는 3행 1열

epoch	cost
0	112662.8359
100	17.9033
200	4.0140
300	3.9923
400	3.9724
500	3.9527
600	3.9330
700	3.9134
800	3.8939
900	3.8746
1000	3.8553
1100	3.8362
1200	3.8171
1300	3.7981
1400	3.7793
1500	3.7606
1600	3.7419
1700	3.7234
1800	3.7049
1900	3.6866
2000	3.6684

# With Matrix

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

$$H(X) = XW$$

```
# initialize W
```

```
w1 = tf.Variable(tf.random_normal([1]))  
w2 = tf.Variable(tf.random_normal([1]))  
w3 = tf.Variable(tf.random_normal([1]))
```

```
# hypothesis, prediction function
```

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

```
# update w1, w2, w3
```

```
w1.assign_sub(learning_rate * w1_grad)  
w2.assign_sub(learning_rate * w2_grad)  
w3.assign_sub(learning_rate * w3_grad)
```

```
# initialize W
```

```
W = tf.Variable(tf.random_normal([3, 1]))
```

tensorflow 2.x 에서는 random.normal

```
# hypothesis, prediction function
```

```
tf.matmul(X, W) + b
```

```
# updates parameters (W and b)
```

```
W.assign_sub(learning_rate * W_grad)
```

# What's Next?

- Logistic (Regression) Classification