“Multi-Variable Linear Regression”

ex)

	Quiz 1(X_1)	Quiz2(X_2)	Quiz3(X_3)	Final(Y)
Instance 1	70	80	70	150
Instance 2	90	85	90	185
Instance 3	90	90	90	180
Instance 4	95	98	100	195
Instance 5	70	65	75	140

<The code that we used until now>

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

x1_data = [70., 90., 90., 95., 70.]
x2_data = [80., 85., 90., 98., 65.]
x3_data = [70., 90., 90., 100., 75.]
y_data = [150., 185., 180., 195., 140.]

# 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(2001):
   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)
```

<The code using "Matrix">

→We use this because if there are a number of instances, then the code will be super long

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

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

Why using dot?
→ It minimizes errors + (tf.float32)
= The result of cost function is approaching to zero

```
# 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])
```

shape= [X,Y] →In this X case, it can be [None,3] or [5,3]
X : How many pairs are there in big square bracket
Y: How many numbers are there in small square bracket

```
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
                matmul → Matrix Multiply
Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()

# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

for step in range(2001):
   cost_val, hy_val, _ = sess.run(
       [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
   if step % 10 == 0:
       print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

<Materials by>

-Sung Kim (Youtuber)

Code: https://github.com/hunkim/DeepLearningZeroToAll/