# “Minimizing Cost!”

```
# basic
import tensorflow.compat.v1 as tf
import matplotlib.pyplot as plt
tf.disable_v2_behavior()

X = [1, 2, 3]
Y = [1, 2, 3]

W = tf.placeholder(tf.float32)
# Our hypothesis for linear model X * W
hypothesis = X * W
```

For simple calculation: H(x) = Wx

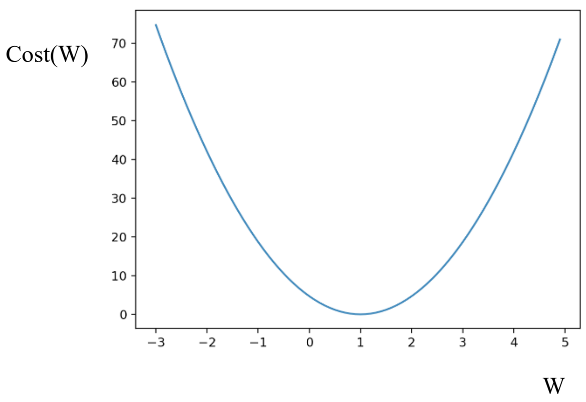
```
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

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

# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())
# Variables for plotting cost function
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)

# Show the cost function
plt.plot(W_val, cost_val)
plt.show()
```




➔Cost function

(Convex function)

# <Then, how to find the minimum “W” in the graph?>

```
# Minimize: Gradient Descent using derivative:
   W -= learning_rate * derivative

learning_rate = 0.1
gradient = tf.reduce_mean((W * X - Y) * X)
descent = W - learning_rate * gradient
update = W.assign(descent)
```

Same definition:

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

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

→gradient =$\frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)}) x^{(i)}$

→update
= update a new W in W so that you can change the value of W in the right way

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

This means “descent = new W”

## So, if you put this in the code….

```
# basic
import tensorflow.compat.v1 as tf
import matplotlib.pyplot as plt
tf.disable_v2_behavior()

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)

# Our hypothesis for linear model X * W
hypothesis = W * X

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize: Gradient Descent using derivative: W -= learning_rate * derivative
learning_rate = 0.1
gradient = tf.reduce_mean((W * X - Y) * X)
descent = W - learning_rate * gradient
update = W.assign(descent)

# 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(21):
   sess.run(update, feed_dict={X: x_data, Y: y_data})
   print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W))
```

## <But this is when the “Hypothesis” function is simple!!>

So….this will help you minimize the cost no matter how complex the function will be…

```
# Minimize: Gradient Descent Magic (Better way)
optimizer =
  tf.train.GradientDescentOptimizer(learning_rate=0.1)
train = optimizer.minimize(cost)
```

This is same as:

```
learning_rate = 0.1
gradient = tf.reduce_mean((W * X - Y) * X)
descent = W - learning_rate * gradient
update = W.assign(descent)
```

## <What if you want to change the “gradient”?>

(Not using the gradient that tensorflow provides)

```
import tensorflow as tf
X = [1, 2, 3]
Y = [1, 2, 3]

W = tf.Variable(5.)

hypothesis = X * W

gradient = tf.reduce_mean((W * X - Y) * X) * 2

cost = tf.reduce_mean(tf.square(hypothesis - Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)

gvs = optimizer.compute_gradients(cost, [W])

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)
```

Change 1.
: Compute gradients for the cost as you want

Change 2.
: Apply it!

# <Materials by>

-Sung Kim (Youtuber)

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