Minimizing Cost

Lecture 03

Simplified hypothesis

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

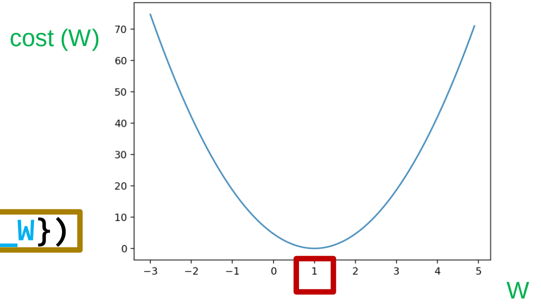
```python
import tensorflow as tf
import matplotlib.pyplot as plt
X = [1, 2, 3]
Y = [1, 2, 3]

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

# 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()
```

matplotlib

http://matplotlib.org/users/installing.html

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

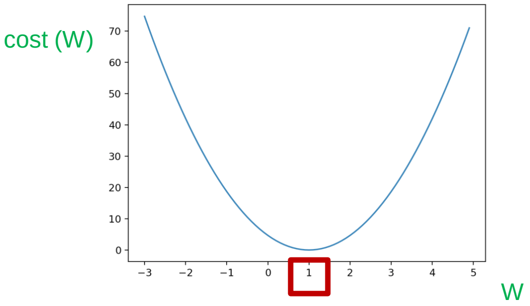


```python
import tensorflow as tf
import matplotlib.pyplot as plt
X = [1, 2, 3]
Y = [1, 2, 3]

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

# 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()
```

X	1	2	3
Y	1	2	3
W	-3		
hypothesis = X * W	-3	-6	-9
hypothesis - Y	-4	-8	-12
tf.square(hypothesis - Y)	16	64	144
tf.reduce_mean((tf.square(hypothesis - Y)	74.667		

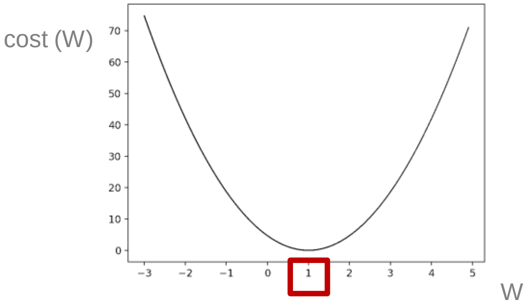
→ sum : 224

average : (sum/3) 74.667



```
feed_W : -3.0000000000000000     curr_cost : 74.666664     curr_W : -3.0
feed_W : -2.9000000000000004     curr_cost : 70.980003     curr_W : -2.9
feed_W : -2.8000000000000003     curr_cost : 67.386665     curr_W : -2.8
feed_W : -2.7000000000000002     curr_cost : 63.886669     curr_W : -2.7
feed_W : -2.6000000000000001     curr_cost : 60.479992     curr_W : -2.6
.
.
.
feed_W : 0.8000000000000000      curr_cost : 0.186667      curr_W : 0.8
feed_W : 0.9000000000000000      curr_cost : 0.046667      curr_W : 0.9
feed_W : 1.0000000000000000      curr_cost : 0.000000      curr_W : 1.0
feed_W : 1.1000000000000001      curr_cost : 0.046667      curr_W : 1.1
feed_W : 1.2000000000000002      curr_cost : 0.186667      curr_W : 1.2
feed_W : 1.3000000000000000      curr_cost : 0.420000      curr_W : 1.3
.
.
.
feed_W : 4.5000000000000000      curr_cost : 57.166668     curr_W : 4.5
feed_W : 4.6000000000000005      curr_cost : 60.479992     curr_W : 4.6
feed_W : 4.7000000000000002      curr_cost : 63.886658     curr_W : 4.7
feed_W : 4.8000000000000007      curr_cost : 67.386665     curr_W : 4.8
feed_W : 4.9000000000000004      curr_cost : 70.980003     curr_W : 4.9
```

```
import tensorflow as
import matplotlib.py
X = [1, 2, 3]
Y = [1, 2, 3]

W = tf.placeholder(t
# Our hypothesis for
hypothesis = X * W

# cost/loss function
cost = tf.reduce_mea
# Launch the graph i
sess = tf.Session()
# Initializes global
sess.run(tf.global_v
# Variables for plot
W_val = []
cost_val = []
for i in range(-30,
    feed_W = i * 0.1
    print("feed_W : %17.16f" %feed_W, end=" ")

    curr_cost, curr_W = sess.run([cost, W], feed_dict={W: feed_W})
    print("\t curr_cost : %8.6f" %curr_cost, "\t curr_W :", curr_W)

    W_val.append(curr_W)
    cost_val.append(curr_cost)

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

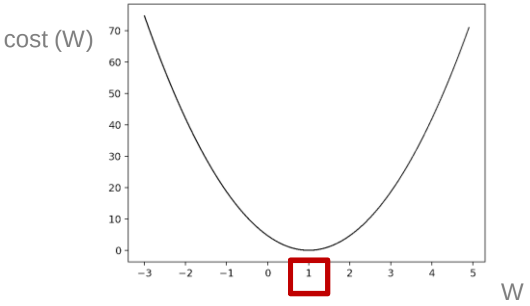


print("feed_W : %17.16f" %feed_W, end=" ")

print("₩t curr_cost : %8.6f" %curr_cost, "₩t curr_W :", curr_W)

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

W = tf.pl
# Our hyp
hypothesi

# cost/lo
cost = tf
# Launch
sess = tf
# Initial
sess.run(
# Variabl
W_val = [
cost_val
for i in
    feed_W
    print(

    curr_c
    print(

    W_val.append(curr_W)
    cost_val.append(curr_cost)

print ("\nW_val 출력 ")
print(type (W_val) , end=""    )
for i in range(len(W_val)):
    if (i%5==0):
        print("\n", end="")
    print ("\t[",i,"]:", W_val[i], end= "")


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

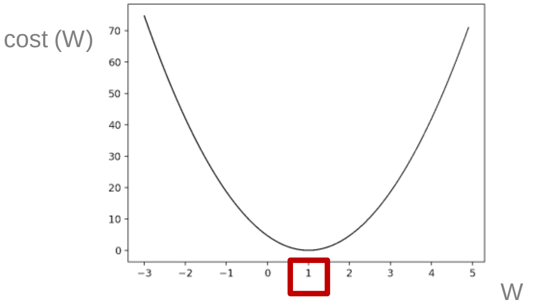
```
W_val 출력
<class 'list'>
        [ 0 ]: -3.0     [ 1 ]: -2.9     [ 2 ]: -2.8     [ 3 ]: -2.7     [ 4 ]: -2.6
        [ 5 ]: -2.5     [ 6 ]: -2.4     [ 7 ]: -2.3     [ 8 ]: -2.2     [ 9 ]: -2.1
        [ 10 ]: -2.0    [ 11 ]: -1.9    [ 12 ]: -1.8    [ 13 ]: -1.7    [ 14 ]: -1.6
        [ 15 ]: -1.5    [ 16 ]: -1.4    [ 17 ]: -1.3    [ 18 ]: -1.2    [ 19 ]: -1.1
        [ 20 ]: -1.0    [ 21 ]: -0.9    [ 22 ]: -0.8    [ 23 ]: -0.7    [ 24 ]: -0.6
        [ 25 ]: -0.5    [ 26 ]: -0.4    [ 27 ]: -0.3    [ 28 ]: -0.2    [ 29 ]: -0.1
        [ 30 ]: 0.0     [ 31 ]: 0.1     [ 32 ]: 0.2     [ 33 ]: 0.3     [ 34 ]: 0.4
        [ 35 ]: 0.5     [ 36 ]: 0.6     [ 37 ]: 0.7     [ 38 ]: 0.8     [ 39 ]: 0.9
        [ 40 ]: 1.0     [ 41 ]: 1.1     [ 42 ]: 1.2     [ 43 ]: 1.3     [ 44 ]: 1.4
        [ 45 ]: 1.5     [ 46 ]: 1.6     [ 47 ]: 1.7     [ 48 ]: 1.8     [ 49 ]: 1.9
        [ 50 ]: 2.0     [ 51 ]: 2.1     [ 52 ]: 2.2     [ 53 ]: 2.3     [ 54 ]: 2.4
        [ 55 ]: 2.5     [ 56 ]: 2.6     [ 57 ]: 2.7     [ 58 ]: 2.8     [ 59 ]: 2.9
        [ 60 ]: 3.0     [ 61 ]: 3.1     [ 62 ]: 3.2     [ 63 ]: 3.3     [ 64 ]: 3.4
        [ 65 ]: 3.5     [ 66 ]: 3.6     [ 67 ]: 3.7     [ 68 ]: 3.8     [ 69 ]: 3.9
        [ 70 ]: 4.0     [ 71 ]: 4.1     [ 72 ]: 4.2     [ 73 ]: 4.3     [ 74 ]: 4.4
        [ 75 ]: 4.5     [ 76 ]: 4.6     [ 77 ]: 4.7     [ 78 ]: 4.8     [ 79 ]: 4.9
```




```
print ("₩nW_val 출력 ")
print(type (W_val) , end=""    )
for i in range(len(W_val)):
    if (i%5==0):
        print("₩n", end="")
    print ("₩t[",i,"]:", W_val[i], end= "")
```

```
import tensorflow as tf
import matplotlib.pyplot as plt
X = [1, 2, 3]
Y = [1, 2, 3]
```

```
cost_val 출력
<class 'list'>
        [ 0 ]: 74.666664        [ 1 ]: 70.980003        [ 2 ]: 67.386665        [ 3 ]: 63.886669        [ 4 ]: 60.479992
        [ 5 ]: 57.166668        [ 6 ]: 53.946674        [ 7 ]: 50.819996        [ 8 ]: 47.786671        [ 9 ]: 44.846661
        [ 10 ]: 42.000000       [ 11 ]: 39.246666       [ 12 ]: 36.586662       [ 13 ]: 34.020004       [ 14 ]: 31.546667
        [ 15 ]: 29.166666       [ 16 ]: 26.879999       [ 17 ]: 24.686666       [ 18 ]: 22.586670       [ 19 ]: 20.580000
        [ 20 ]: 18.666666       [ 21 ]: 16.846666       [ 22 ]: 15.120000       [ 23 ]: 13.486667       [ 24 ]: 11.946668
        [ 25 ]: 10.500000       [ 26 ]: 9.146666        [ 27 ]: 7.886667        [ 28 ]: 6.720000        [ 29 ]: 5.646666
        [ 30 ]: 4.666667        [ 31 ]: 3.780000        [ 32 ]: 2.986667        [ 33 ]: 2.286666        [ 34 ]: 1.680000
        [ 35 ]: 1.166667        [ 36 ]: 0.746667        [ 37 ]: 0.420000        [ 38 ]: 0.186667        [ 39 ]: 0.046667
        [ 40 ]: 0.000000        [ 41 ]: 0.046667        [ 42 ]: 0.186667        [ 43 ]: 0.420000        [ 44 ]: 0.746666
        [ 45 ]: 1.166667        [ 46 ]: 1.680000        [ 47 ]: 2.286667        [ 48 ]: 2.986666        [ 49 ]: 3.779999
        [ 50 ]: 4.666667        [ 51 ]: 5.646666        [ 52 ]: 6.720001        [ 53 ]: 7.886665        [ 54 ]: 9.146668
        [ 55 ]: 10.500000       [ 56 ]: 11.946666       [ 57 ]: 13.486669       [ 58 ]: 15.119998       [ 59 ]: 16.846670
        [ 60 ]: 18.666666       [ 61 ]: 20.579996       [ 62 ]: 22.586670       [ 63 ]: 24.686666       [ 64 ]: 26.880005
        [ 65 ]: 29.166666       [ 66 ]: 31.546661       [ 67 ]: 34.020004       [ 68 ]: 36.586662       [ 69 ]: 39.246670
        [ 70 ]: 42.000000       [ 71 ]: 44.846661       [ 72 ]: 47.786663       [ 73 ]: 50.820007       [ 74 ]: 53.946674
        [ 75 ]: 57.166668       [ 76 ]: 60.479992       [ 77 ]: 63.886658       [ 78 ]: 67.386665       [ 79 ]: 70.980003
```




```
print ("\nW_val 출력 ")
print(type (W_val) , end=""    )
for i in range(len(W_val)):
    if (i%5==0):
        print("\n", end="")
    print ("\t[",i,"]:", W_val[i], end= "")

print ("\ncost_val 출력 ")
print(type (cost_val) , end=""    )
for i in range(len(cost_val)):
    if (i%5==0):
        print("\n", end="")
    print ("\t[",i,"]: %8.6f" %cost_val[i], end= "")


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

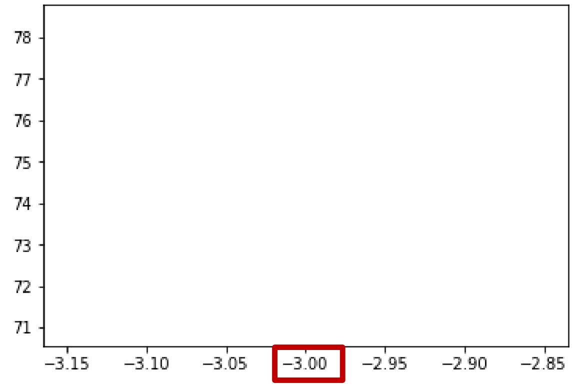
```
print ("₩ncost_val 출력 ")
print(type (cost_val) , end=""    )
for i in range(len(cost_val)):
    if (i%5==0):
        print("₩n", end="")
    print ("₩t[",i,"]: %8.6f" %cost_val[i], end= "")
```

```
import tensorflow as tf
import matplotlib.pyplot as plt
X = [1, 2, 3]
Y = [1, 2, 3]

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

# 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 = []
```

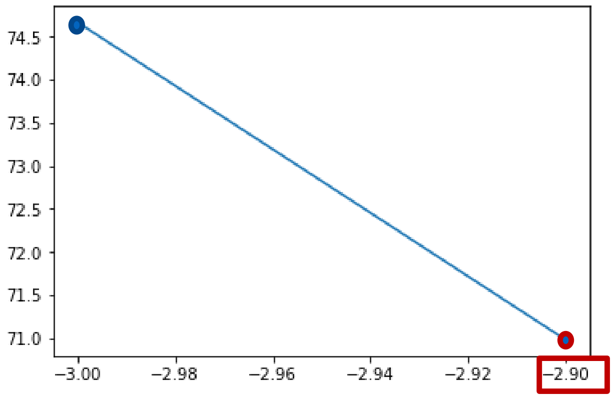
```
print("i:%3d" %i, " -> ", "curr_cost:%10.6f" %curr_cost, "₩t curr_W:%5.1f" %curr_W)
```

```
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)

    print("i:%3d" %i, " -> ", "curr_cost:%10.6f" %curr_cost, "\t curr_W:%5.1f" %curr_W)

plt.plot(W_val, cost_val)
plt.show()
```

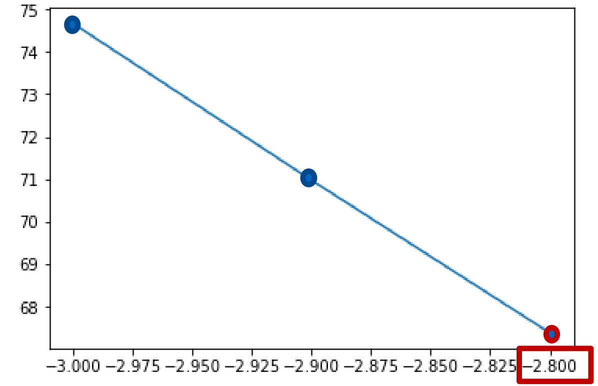
i:-30 -> curr_cost: 74.666664 curr_W: -3.0



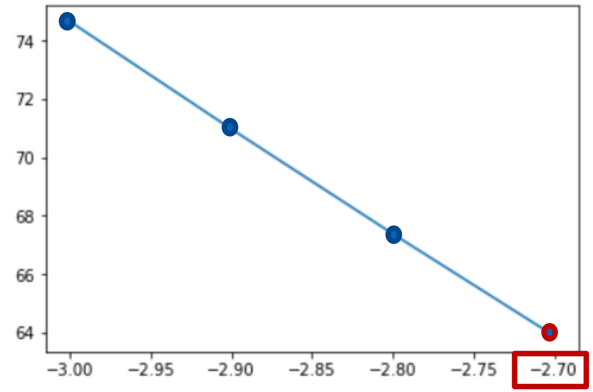
i:-29 -> curr_cost: 70.980003 curr_W: -2.9



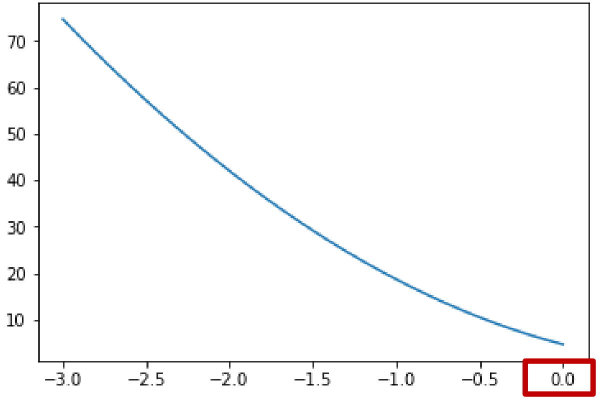
i:-28 -> curr_cost: 67.386665 curr_W: -2.8



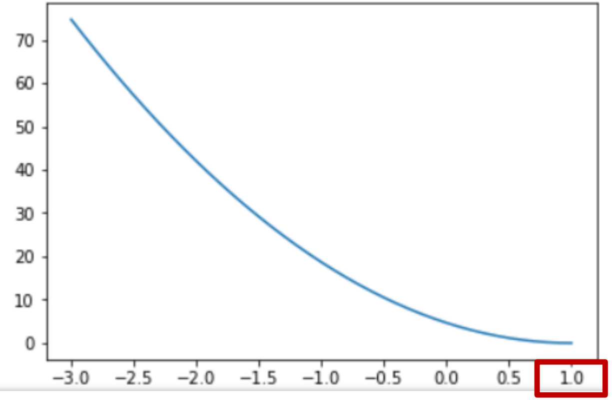
i:-27 -> curr_cost: 63.886669 curr_W: -2.7



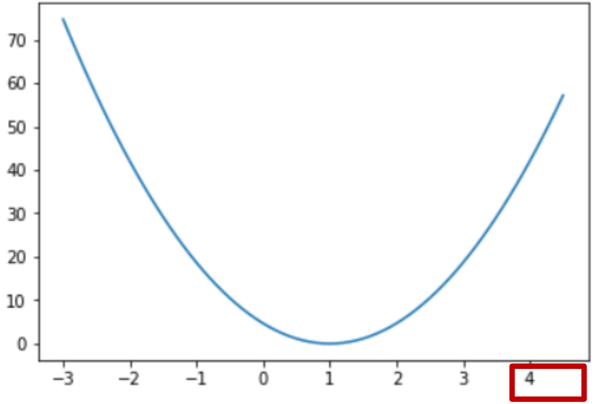
i: 0 -> curr_cost: 4.666667 curr_W: 0.0



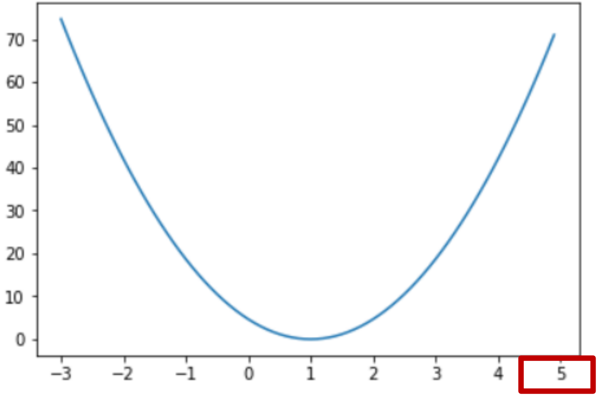
i: 10 -> curr_cost: 0.000000 curr_W: 1.0



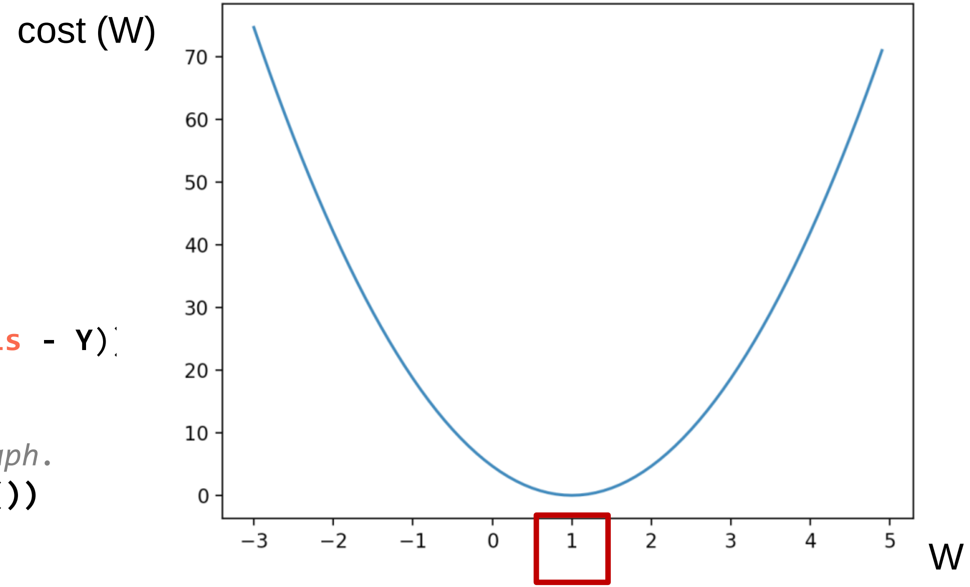
i: 45 -> curr_cost: 57.166668 curr_W: 4.5



i: 49 -> curr_cost: 70.980003 curr_W: 4.9

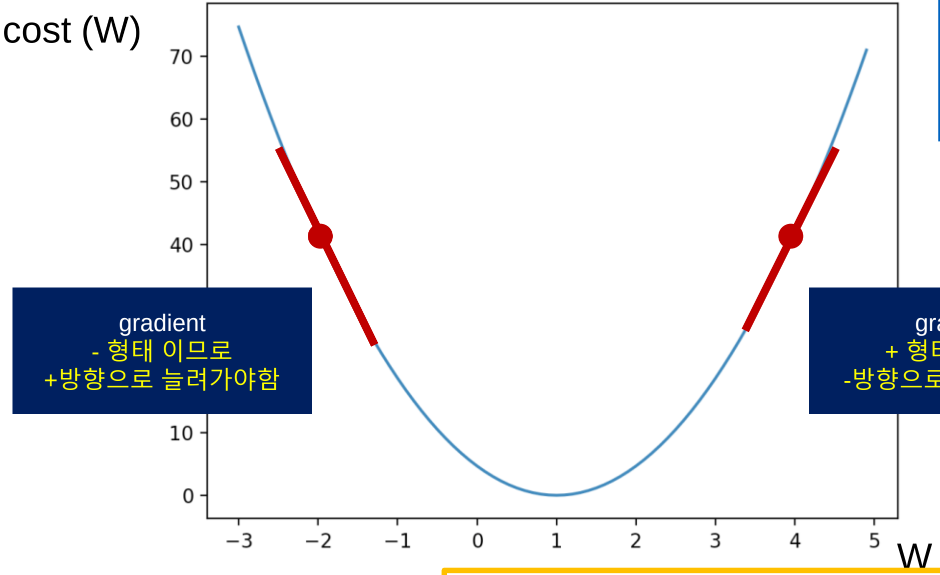


```python
import tensorflow as tf
import matplotlib.pyplot as plt
X = [1, 2, 3]
Y = [1, 2, 3]

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

# 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()
```



$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

Gradient descent



current

미분

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

gradinet

$$cost(W) = \boxed{\frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)})^2}$$

gradinet

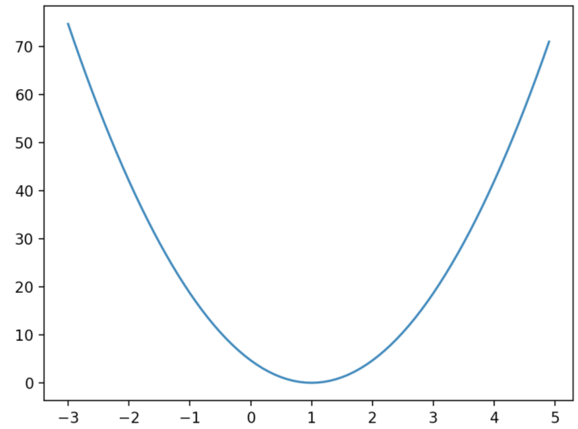
```
import tensorflow as tf
x_data = [1, 2, 3]
y_data = [1, 2, 3]

# Create a tensor of shape [1] consisting of random normal values
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 = X * W

# 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())
print("step  \t cost  \t\t     W ")
for step in range(41):
    sess.run(update, feed_dict={X: x_data, Y: y_data})
    print(step, "\t", sess.run(cost, feed_dict={X: x_data, Y: y_data}), "\t", sess.run(W))
```

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

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

Gradient descent

cost (W)



W

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

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

```
W = tf.Variable(tf.random_normal([1]), name='weight')
learning_rate = 0.1
gradient = tf.reduce_mean((W * X - Y) * X)
descent = W - learning_rate * gradient
update = W.assign(descent)
```

텐서플로에서는 =로 할당하지 않고 assign으로 할당할 수 있다.
즉, W의 값을 assign 메소드를 사용하여 새롭게 할당한다.
이러게 할당하는 부분을 update라는 연산자로 사용하겠다는 뜻

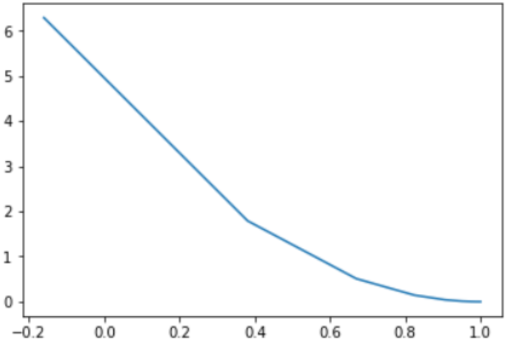
step	cost	W
0	0.65983266	[0.6239778]
1	0.18768568	[0.7994548]
2	0.053386167	[0.89304256]
3	0.015185393	[0.94295603]
4	0.0043194103	[0.96957654]
5	0.0012286264	[0.9837742]
6	0.00034947737	[0.99134624]
7	9.940494e-05	[0.9953847]
8	2.8275383e-05	[0.9975385]
9	8.042801e-06	[0.9986872]
10	2.2877316e-06	[0.9992998]
11	6.5078694e-07	[0.9996266]
12	1.8508689e-07	[0.99980086]
13	5.262274e-08	[0.9998938]
14	1.4987608e-08	[0.9999433]
15	4.258099e-09	[0.9999698]
16	1.204801e-09	[0.9999839]
17	3.43789e-10	[0.9999914]
18	9.884715e-11	[0.9999954]
19	2.8162361e-11	[0.99999756]
20	7.716494e-12	[0.9999987]

step	cost	W
21	2.3874236e-12	[0.9999993]
22	5.163277e-13	[0.99999964]
23	1.2908193e-13	[0.9999998]
24	9.947598e-14	[0.9999999]
25	2.4868996e-14	[0.99999994]
26	0.0	[1.]
27	0.0	[1.]
28	0.0	[1.]
29	0.0	[1.]
30	0.0	[1.]
31	0.0	[1.]
32	0.0	[1.]
33	0.0	[1.]
34	0.0	[1.]
35	0.0	[1.]
36	0.0	[1.]
37	0.0	[1.]
38	0.0	[1.]
39	0.0	[1.]
40	0.0	[1.]

```
import tensorflow as tf
import matplotlib.pyplot as plt

x_data = [1, 2, 3]
y_data = [1, 2, 3]

# Create a tensor of shape [1] consisting of random normal values
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 = X * W

# 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())
```

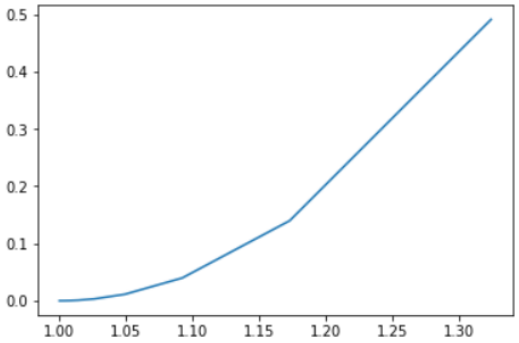
step	cost	W
0	6.2836437	[-0.16038573]
1	1.7873474	[0.3811276]
2	0.508401	[0.66993475]
3	0.1446118	[0.8239652]
4	0.04113406	[0.90611476]
5	0.011700347	[0.94992787]
6	0.0033280961	[0.97329485]
7	0.0009466668	[0.98575723]
8	0.0002692744	[0.99240386]
9	7.6593424e-05	[0.99594873]
10	2.1786791e-05	[0.99783933]
11	6.196909e-06	[0.99884766]
12	1.7625867e-06	[0.9993854]
13	5.013033e-07	[0.99967223]
14	1.4264435e-07	[0.9998252]
15	4.0554635e-08	[0.9999068]
16	1.1543709e-08	[0.9999503]
17	3.286285e-09	[0.9999735]
18	9.329296e-10	[0.9999859]
19	2.6142763e-10	[0.9999925]
20	7.3949735e-11	[0.999996]
21	2.1486812e-11	[0.99999785]
22	5.8513194e-12	[0.99999887]
23	1.8047785e-12	[0.9999994]
24	4.5119464e-13	[0.9999997]
25	1.2908193e-13	[0.9999998]

step	cost	W
26	9.947598e-14	[0.9999999]
27	2.4868996e-14	[0.99999994]
28	0.0	[1.]
29	0.0	[1.]
30	0.0	[1.]
31	0.0	[1.]
32	0.0	[1.]
33	0.0	[1.]
34	0.0	[1.]
35	0.0	[1.]
36	0.0	[1.]
37	0.0	[1.]
38	0.0	[1.]
39	0.0	[1.]
40	0.0	[1.]
41	0.0	[1.]
42	0.0	[1.]
43	0.0	[1.]
44	0.0	[1.]
45	0.0	[1.]
46	0.0	[1.]
47	0.0	[1.]
48	0.0	[1.]
49	0.0	[1.]
50	0.0	[1.]



step	cost	W
0	0.49110034	[1.3244007]
1	0.13969077	[1.1730137]
2	0.039734248	[1.092274]
3	0.011302193	[1.0492128]
4	0.0032148378	[1.0262468]
5	0.00091444066	[1.0139983]
6	0.0002601041	[1.0074657]
7	7.3986324e-05	[1.0039817]
8	2.1045045e-05	[1.0021236]
9	5.9866684e-06	[1.0011326]
10	1.7025194e-06	[1.000604]
11	4.841698e-07	[1.0003221]
12	1.3774762e-07	[1.0001718]
13	3.9115548e-08	[1.0000916]
14	1.1147942e-08	[1.0000489]
15	3.1744254e-09	[1.0000261]
16	9.111479e-10	[1.000014]
17	2.5492378e-10	[1.0000074]
18	7.316222e-11	[1.0000039]
19	2.1486812e-11	[1.0000021]
20	6.631732e-12	[1.0000012]
21	1.8047785e-12	[1.0000006]
22	5.163277e-13	[1.0000004]
23	2.6526928e-13	[1.0000002]
24	9.947598e-14	[1.0000001]
25	0.0	[1.]

step	cost	W
26	0.0	[1.]
27	0.0	[1.]
28	0.0	[1.]
29	0.0	[1.]
30	0.0	[1.]
31	0.0	[1.]
32	0.0	[1.]
33	0.0	[1.]
34	0.0	[1.]
35	0.0	[1.]
36	0.0	[1.]
37	0.0	[1.]
38	0.0	[1.]
39	0.0	[1.]
40	0.0	[1.]
41	0.0	[1.]
42	0.0	[1.]
43	0.0	[1.]
44	0.0	[1.]
45	0.0	[1.]
46	0.0	[1.]
47	0.0	[1.]
48	0.0	[1.]
49	0.0	[1.]
50	0.0	[1.]



```python
import tensorflow as tf
x_data = [1, 2, 3]
y_data = [1, 2, 3]
# Create a tensor of shape [1] consisting of random normal values
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 = X * W

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

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

=

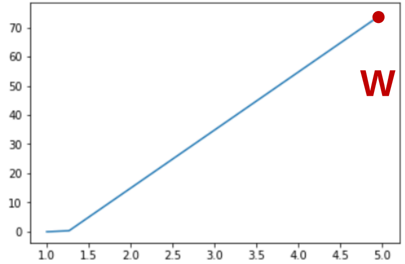
```python
# Minimize: Gradient Descent using derivative: W -= learning_rate * derivative
learning_rate = 0.1
gradient = tf.reduce_mean((W * X - Y) * X)
descent = W - learning_rate * gradient
update = W.assign(descent)
```

```python
# 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(41):
   sess.run(update, feed_dict={X: x_data, Y: y_data})
   print(step, "\t", sess.run(cost, feed_dict={X: x_data, Y: y_data}), "\t", sess.run(W))
```

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

Output when W=5

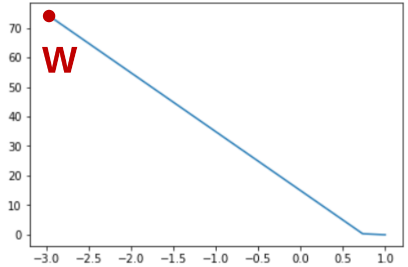
```
import tensorflow as tf
import matplotlib.pyplot as plt
# tf Graph Input
X = [1, 2, 3]
Y = [1, 2, 3]

# Set wrong model weights
W = tf.Variable(5.0)
# Linear model
hypothesis = X * W
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize: Gradient Descent Magic
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
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())

W_val = []
cost_val = []

print("step  \t  cost   \t  W")
for step in range(21):
   curr_W = sess.run(W)
   curr_cost = sess.run(cost)
   print(step, "\t",curr_cost, "\t", curr_W)
   W_val.append(curr_W)
   cost_val.append(curr_cost)
   sess.run(train)

plt.plot( W_val, cost_val)
plt.show()
```



step	cost	W
0	74.666664	5.0
1	0.3318512	1.2666664
2	0.0014748968	1.0177778
3	6.555027e-06	1.0011852
4	2.91322e-08	1.000079
5	1.2839034e-10	1.0000052
6	5.163277e-13	1.0000004
7	0.0	1.0
8	0.0	1.0
9	0.0	1.0
10	0.0	1.0
11	0.0	1.0
12	0.0	1.0
13	0.0	1.0
14	0.0	1.0
15	0.0	1.0
16	0.0	1.0
17	0.0	1.0
18	0.0	1.0
19	0.0	1.0
20	0.0	1.0

Output when W = -3

```
import tensorflow as tf
import matplotlib.pyplot as plt
# tf Graph Input
X = [1, 2, 3]
Y = [1, 2, 3]

# Set wrong model weights
W = tf.Variable(-3.0)
# Linear model
hypothesis = X * W
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize: Gradient Descent Magic
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
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())

W_val = []
cost_val = []

print("step  \t  cost   \t  W")
for step in range(21):
   curr_W = sess.run(W)
   curr_cost = sess.run(cost)
   print(step, "\t",curr_cost, "\t", curr_W)
   W_val.append(curr_W)
   cost_val.append(curr_cost)
   sess.run(train)

plt.plot( W_val, cost_val)
plt.show()
```



step	cost	W
0	74.666664	-3.0
1	0.3318512	0.7333336
2	0.0014748932	0.98222226
3	6.555027e-06	0.9988148
4	2.91322e-08	0.99992096
5	1.3195844e-10	0.9999947
6	5.163277e-13	0.99999964
7	2.4868996e-14	0.99999994
8	0.0	1.0
9	0.0	1.0
10	0.0	1.0
11	0.0	1.0
12	0.0	1.0
13	0.0	1.0
14	0.0	1.0
15	0.0	1.0
16	0.0	1.0
17	0.0	1.0
18	0.0	1.0
19	0.0	1.0
20	0.0	1.0

Optional: compute_gradient and apply_gradient

```
import tensorflow as tf
X = [1, 2, 3]
Y = [1, 2, 3]
# Set wrong model weights
W = tf.Variable(5.)
# Linear model
hypothesis = X * W

# Manual gradient
gradient = tf.reduce_mean((W * X - Y) * X) * 2
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize: Gradient Descent Magic
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)

# Get gradients
gvs = optimizer.compute_gradients(cost, W)
# Optional: modify gradient if necessary
# Apply gradients
apply_gradients = optimizer.apply_gradients(gvs)

sess = tf.Session()
sess.run(tf.global_variables_initializer())
print("step \t manul_gradient \tW    \t computer_gvs (gradient, W)")
print("="*80)
for step in range(100):
   print(step, "\t", sess.run(gradient), "\t %f" %sess.run(W), "\t" , sess.run(gvs)) # print(step, sess.run([gradient, W, gvs]))
   sess.run(apply_gradients)
```

Optional: compute_gradient and apply_gradient

step	manul_gradient	W	computer_gvs (gradient, W)
0	37.333332	5.000000	[(37.333336, 5.0)]
1	33.84889	4.626667	[(33.84889, 4.6266665)]
2	30.689657	4.288177	[(30.689657, 4.2881775)]
3	27.825287	3.981281	[(27.825287, 3.9812808)]
4	25.228262	3.703028	[(25.228264, 3.703028)]
5	22.873621	3.450745	[(22.873623, 3.4507453)]
.			
.			
.			
95	0.0033854644	1.000363	[(0.0033854644, 1.0003628)]
96	0.0030694802	1.000329	[(0.0030694804, 1.0003289)]
97	0.0027837753	1.000298	[(0.0027837753, 1.0002983)]
98	0.0025234222	1.000270	[(0.0025234222, 1.0002704)]
99	0.0022875469	1.000245	[(0.0022875469, 1.0002451)]