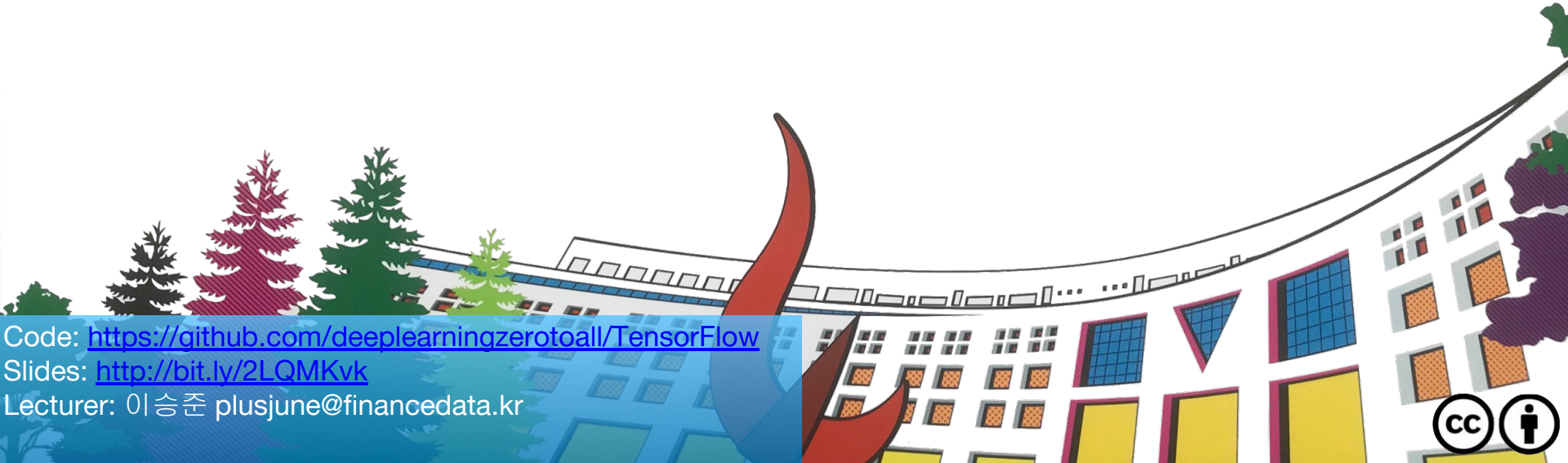
ML/DL for Everyone Season2

with TensorFlow

03 - How to minimize cost LAB



Code: https://github.com/deeplearningzerotoall/TensorFlow
Slides: http://bit.ly/2LQMKvk
Lecturer: 이승준 plusjune@financedata.kr

Simplified hypothesis

Hypothesis $H(x) = Wx$

Cost $cost(W) = \frac{1}{m} \sum_{i=1}^{m} (Wx_i - y_i)^2$

우리의 가설(예측) - 실제

Cost function in pure Python

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

```python
import numpy as np

X = np.array([1, 2, 3])
Y = np.array([1, 2, 3])

def cost_func(W, X, Y):
    c = 0
    for i in range(len(X)):
        c += (W * X[i] - Y[i]) ** 2
    return c / len(X)

for feed_W in np.linspace(-3, 5, num=15):
    curr_cost = cost_func(feed_W, X, Y)
    print("{:6.3f} | {:10.5f}".format(feed_W, curr_cost))
```

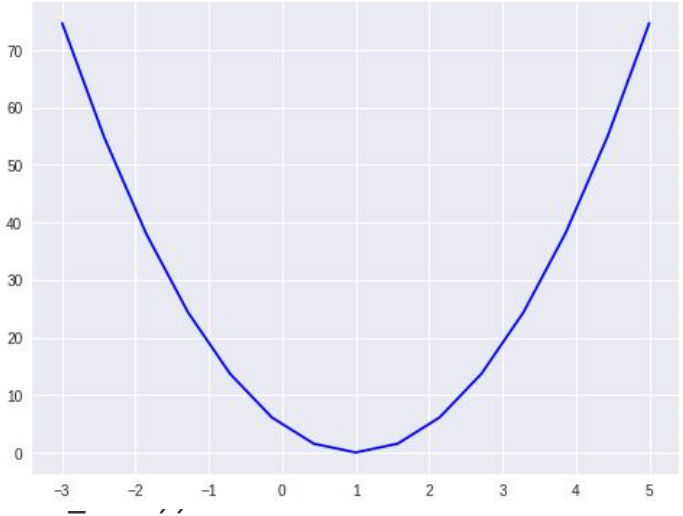
-3부터 5까지 15개의. 구간으로 나누기

```
     W          cost
-3.000 |   74.66667
-2.429 |   54.85714
-1.857 |   38.09524
-1.286 |   24.38095
-0.714 |   13.71429
-0.143 |    6.09524
 0.429 |    1.52381
 1.000 |    0.00000
 1.571 |    1.52381
 2.143 |    6.09524
 2.714 |   13.71429
 3.286 |   24.38095
 3.857 |   38.09524
 4.429 |   54.85714
 5.000 |   74.66667
```

Cost function in pure Python

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

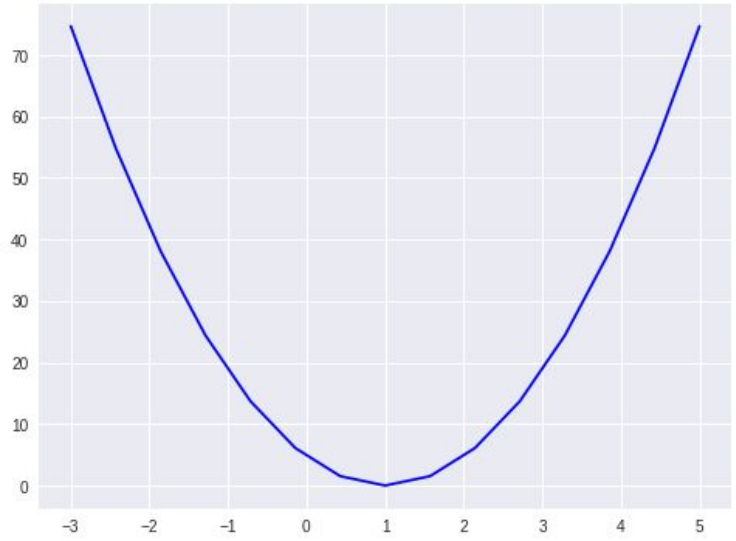
```
import numpy as np

X = np.array([1, 2, 3])
Y = np.array([1, 2, 3])

def cost_func(W, X, Y):
    c = 0
    for i in range(len(X)):
        c += (W * X[i] - Y[i]) ** 2
    return c / len(X)

for feed_W in np.linspace(-3, 5, num=15):
    curr_cost = cost_func(feed_W, X, Y)
    print("{:6.3f} | {:10.5f}".format(feed_W,
```



Cost function in TensorFlow

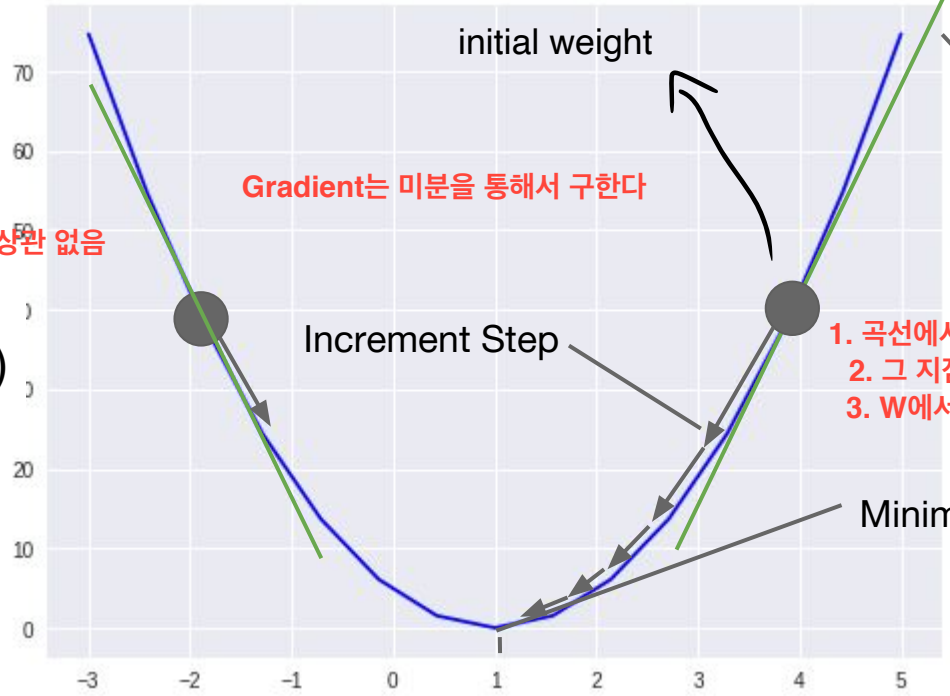
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx_i - y_i)^2$$

```
X = np.array([1, 2, 3])
Y = np.array([1, 2, 3])

def cost_func(W, X, Y):
  hypothesis = X * W
  return tf.reduce_mean(tf.square(hypothesis - Y))

W_values = np.linspace(-3, 5, num=15)
cost_values = []

for feed_W in W_values:
    curr_cost = cost_func(feed_W, X, Y)
    cost_values.append(curr_cost)
    print("{:6.3f} | {:10.5f}".format(feed_W, curr_
```



Gradient descent

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (W(x_i) - y_i) x_i$$



최저점에 다가갈 수록 기울기 작아진다

Gradient descent

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

$$W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(W(x_i) - y_i)x_i$$

함수의 기울기

새로운 W 값

```
alpha = 0.01
gradient = tf.reduce_mean(tf.multiply(tf.multiply(W, X) - Y, X))
descent = W - tf.multiply(alpha, gradient)
W.assign(descent)
```

Gradient descent

랜덤 시드를 특정한 값으로 초기화 (다음 다시 이 코드를 실행해도 동일한 결과를 위해서)

```
tf.set_random_seed(0)  # for reproducibility

x_data = [1., 2., 3., 4.]
y_data = [1., 3., 5., 7.]

W = tf.Variable(tf.random_normal([1], -100., 100.))

for step in range(300):
    hypothesis = W * X
    cost = tf.reduce_mean(tf.square(hypothesis - Y))

    alpha = 0.01
    gradient = tf.reduce_mean(tf.multiply(tf.multiply(W, X) - Y, X))
    descent = W - tf.multiply(alpha, gradient)
    W.assign(descent)

    if step % 10 == 0:
        print('{:5} | {:10.4f} | {:10.6f}'.format(
            step, cost.numpy(), W.numpy()[0]))
```

정규 분포를 따르는 행렬 1개짜리
.random_normal([행렬 수], 평균, 편차

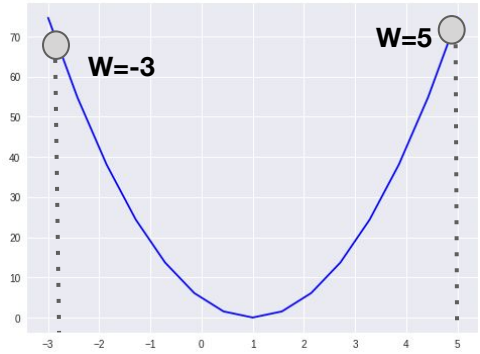
$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (W(x_i) - y_i) x_i$$

Gradient descent

```
tf.set_random_seed(0)  # for reproducibility

x_data = [1., 2., 3., 4.]
y_data = [1., 3., 5., 7.]

W = tf.Variable(tf.random_normal([1], -100., 100.))

for step in range(300):
    hypothesis = W * X
    cost = tf.reduce_mean(tf.square(hypothesis - Y))

    alpha = 0.01
    gradient = tf.reduce_mean(tf.multiply(tf.multiply(W, X) - Y, X)
    descent = W - tf.multiply(alpha, gradient)
    W.assign(descent)

    if step % 10 == 0:
        print('{:5} | {:10.4f} | {:10.6f}'.format(
            step, cost.numpy(), W.numpy()[0]))
```

```
step        cost             W

  0 | 11716.3086 |   48.767971
 10 |  4504.9126 |   30.619968
 20 |  1732.1364 |   19.366755
 30 |   666.0052 |   12.388859
 40 |   256.0785 |    8.062004
 50 |    98.4620 |    5.379007
 60 |    37.8586 |    3.715335
 70 |    14.5566 |    2.683725
 80 |     5.5970 |    2.044044
 ......
240 |     0.0000 |    1.000499
250 |     0.0000 |    1.000309
260 |     0.0000 |    1.000192
270 |     0.0000 |    1.000119
280 |     0.0000 |    1.000074
290 |     0.0000 |    1.000046
```

Output when W=5




```
tf.set_random_seed(0)  # for reproducibility

x_data = [1., 2., 3., 4.]
y_data = [1., 3., 5., 7.]

W = tf.Variable([5.0])
```

W를 값을 앞에서는 랜덤 값을 주었는데, 특정 값을 줘도 같은 결과

```
for step in range(300):
    hypothesis = W * X
    cost = tf.reduce_mean(tf.square(hypothesis - Y))

    alpha = 0.01
    gradient = tf.reduce_mean(tf.multiply(tf.multiply(W, X) - Y, X)
    descent = W - tf.multiply(alpha, gradient)
    W.assign(descent)

    if step % 10 == 0:
        print('{:5} | {:10.4f} | {:10.6f}'.format(
            step, cost.numpy(), W.numpy()[0]))
```

step	cost	W
0	74.6667	4.813334
10	28.7093	3.364572
20	11.0387	2.466224
30	4.2444	1.909177
40	1.6320	1.563762
50	0.6275	1.349578
60	0.2413	1.216766
70	0.0928	1.134412
80	0.0357	1.083346
....	....	
270	0.0000	1.000009
280	0.0000	1.000006
290	0.0000	1.000004

What’s Next?

- Multi-Variable Linear regression