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

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

#입력과 출력 동일하게 1,2,3의 값으로 준비

def cost\_func(W, X, Y):

hypothesis = X \* W

return tf.reduce\_mean(tf.square(hypothesis - Y))

#hypothsis에서 y를 뺀후 제곱 후 평균을 낸값(cost function 구현)

W\_values = np.linspace(-3, 5, num=15)

-3에서 5까지 15개의 구간으로 나눔

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\_cost))

#리스트값을 뽑아서 cost가 w에 따라서 어떻게 변하는지 기록 후 출력

alpha = 0.01

gradient = tf.reduce\_mean(tf.multiply(tf.multiply(W, X) - Y, X))

#기울기

descent = W - tf.multiply(alpha, gradient)

#알파값을 구한 후 w로 빼줌 =새로운 w값

W.assign(descent)

#세로운w값 업데이트

tf.set\_random\_seed(0) # for reproducibility

#다음에 다시 실행했을 때도 똑같이 재현될 수 있도록 random seed를 특정한 값으로 초기화

x\_data = [1., 2., 3., 4.]

y\_data = [1., 3., 5., 7.]

#x\_data, y\_data준비

W = tf.Variable(tf.random\_normal([1], -100., 100.))

#w정의 정규분포를 따르는 랜덤 넘버를 1개로 변수를 만들어 w에 할당

for step in range(300):

hypothesis = W \* X

cost = tf.reduce\_mean(tf.square(hypothesis - Y))

#gradient descent 300회 진행

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]))

#10번에 한번씩 cost값과 w값 출력

#cost는 0으로 수렴,w는 특정값으로 수렴하는 것을 알 수 있음