

# < TensorFlow >

91914145 장혜선  
07.06





# 목차

1

What is ???

- TensorFlow
- Jupyter

2

TensorFlow 문법 1.x vs 2.x

- Tensor      • placeholder
- Session

3

행렬 곱과 선형회귀

- 행렬 곱
- 선형회귀

## 1-1. What is TensorFlow?



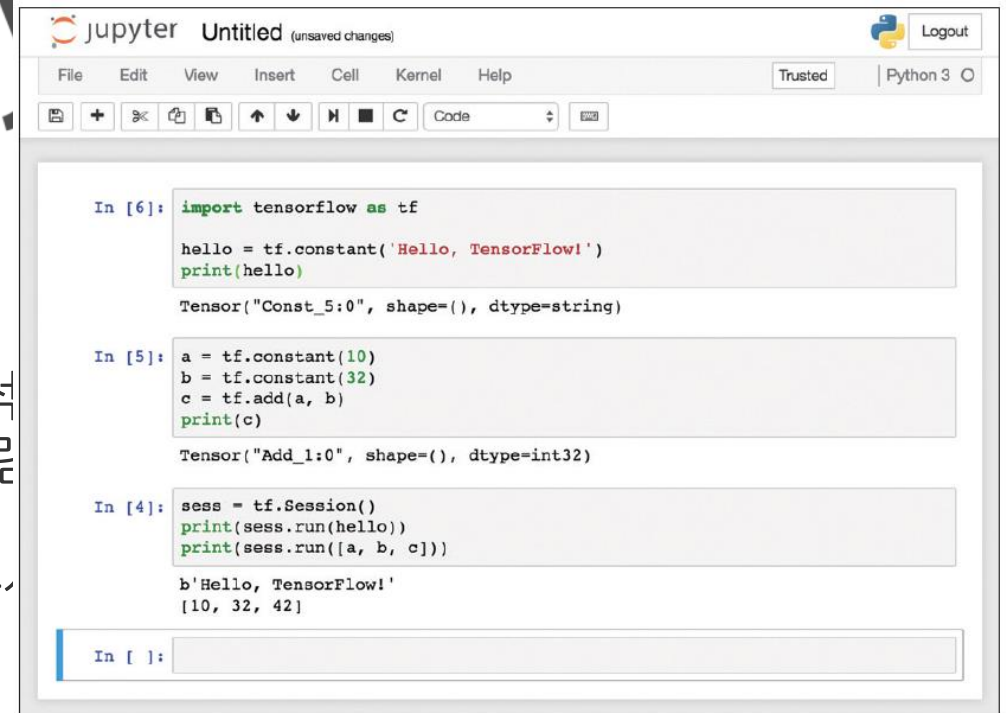
# TensorFlow

- ✓ 머신 러닝 프로그램 중 특히 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공하는 머신 러닝 라이브러리
- ✓ 그래프 형태의 수학적 계산을 수행하는 핵심 라이브러리를 구현한 후, 그 위에 딥러닝을 포함한 여러 머신 러닝을 쉽게 할 수 있는 다양한 라이브러리를 올린 형태

## 1-2. What is Jupyter?



- ✓ 웹 브라우저상에서 파이썬 코드를 단계적  
빠르게 확인해볼 수 있도록 하는 프로그램
- ✓ 코드 한 줄씩 실행 결과를 보여주는 대화-



```

jupyter Untitled (unsaved changes)
File Edit View Insert Cell Kernel Help Trusted Python 3

In [6]: import tensorflow as tf
        hello = tf.constant('Hello, TensorFlow!')
        print(hello)
        Tensor("Const_5:0", shape=(), dtype=string)

In [5]: a = tf.constant(10)
        b = tf.constant(32)
        c = tf.add(a, b)
        print(c)
        Tensor("Add_1:0", shape=(), dtype=int32)

In [4]: sess = tf.Session()
        print(sess.run(hello))
        print(sess.run([a, b, c]))
        b'Hello, TensorFlow!'
        [10, 32, 42]

In [ ]:
  
```



## 2-1. TensorFlow [ tensor ]

1.x

```
In [85]: import tensorflow as tf

hello = tf.constant('Hello, TensorFlow!')
print(hello)

a = tf.constant(10)
b = tf.constant(32)
c = tf.add(a,b)
print(c)
```

```
Tensor("Const:0", shape=(), dtype=string)
Tensor("Add_83:0", shape=(), dtype=int32)
```

2.x

```
In [1]: import tensorflow as tf

hello = tf.constant('Hello, TensorFlow!')
print(hello)

a = tf.constant(10)
b = tf.constant(32)
c = tf.add(a,b)
print(c)
```

```
tf.Tensor(b'Hello, TensorFlow!', shape=(), dtype=string)
tf.Tensor(42, shape=(), dtype=int32)
```



## 2-1. TensorFlow [ tensor ]

```
Tensor("Const:0", shape=(), dtype=string)  
Tensor("Add_83:0", shape=(), dtype=int32)
```

Rank

차원의 수

3 -> 랭크가 0인 텐서, 세이프 [ ]  
[1,2] -> 랭크가 1인 텐서, 세이프 [2]  
[[1,2],[3,4]] -> 랭크가 2인 텐서, 세이프는 [2,2]  
[[[1,2,3]],[[4,5,6]]] -> 랭크가 3인 텐서, 세이프는 [2,1,3]

Shape

각 차원의  
요소 개수

```
[[[1. 2.]  
 [3. 4.]  
 [5. 6.]], shape=(3, 2), dtype=float32)
```

## 2-2. TensorFlow [ Session ]

1.x

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

hello = tf.constant('Hello, TensorFlow!')
print(hello)
```

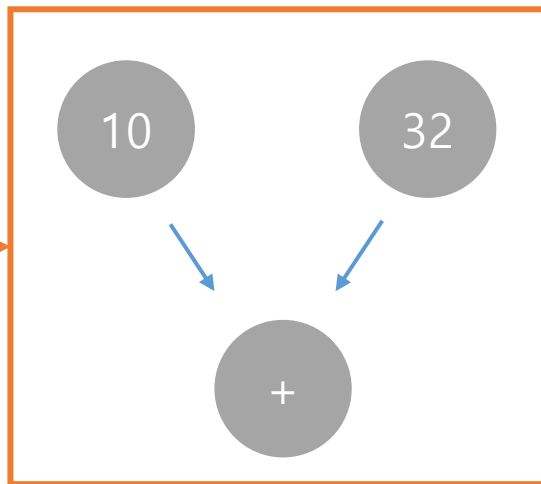
```
a = tf.constant(10)
b = tf.constant(32)
c = tf.add(a,b)
print(c)
```

```
sess = tf.Session()
print(sess.run(hello))
print(sess.run([a, b, c]))
```

```
sess.close()
```

```
Tensor("Const_23:0", shape=(), dtype=string)
Tensor("Add_88:0", shape=(), dtype=int32)
b'Hello, TensorFlow!'
[10, 32, 42]
```

생성



실행

10 + 32



## 2-3. TensorFlow [ placeholder ]

- ✓ 사용할 입력 값을 나중에 받기 위해 사용하는 매개변수

1.x

```
a = tf.placeholder(tf.float32,[None]) #변수값 지정 X
b = tf.placeholder(tf.float32,[None])
result = tf.add(a,b)

print("== 변수 설정 전 ==")
print(a)

print("== 값 지정 후 결과 값 ==")
sess = tf.Session()
sess.run(tf.global_variables_initializer())

print(sess.run(result, feed_dict = {a: [1, 3], b: [2, 5]})) #입력값 지정
```

```
== 변수 설정 전 ==
Tensor("Placeholder_42:0", shape=(?,), dtype=float32)
값 지정 후 결과 값
[3. 8.]
```



## 3-1. 행렬 곱

$$\begin{pmatrix} \overset{1}{a} & \overset{2}{b} \\ \underset{2}{c} & \underset{4}{d} \end{pmatrix} \begin{pmatrix} \overset{1}{1} & \overset{2}{3} \\ \underset{2}{2} & \underset{4}{4} \end{pmatrix} = \begin{pmatrix} \overset{11}{ax1+bx2} & \overset{12}{ax3+bx4} \\ \underset{21}{cx1+dx2} & \underset{22}{cx3+dx4} \end{pmatrix}$$

$$\begin{pmatrix} \overset{1}{a} & \overset{2}{b} & \overset{3}{c} \\ \underset{2}{d} & \underset{5}{e} & \underset{6}{f} \end{pmatrix} \begin{pmatrix} \overset{1}{1} & \overset{2}{4} \\ \underset{2}{2} & \underset{5}{5} \\ \underset{3}{3} & \underset{6}{6} \end{pmatrix} = \begin{pmatrix} \overset{11}{ax1+bx2+cx3} & \overset{12}{ax4+bx5+cx6} \\ \underset{21}{dx1+ex2+fx3} & \underset{22}{dx4+ex5+fx6} \end{pmatrix}$$

$$\begin{pmatrix} \overset{1}{a} & \overset{2}{b} \\ \underset{2}{c} & \underset{4}{d} \\ \underset{3}{e} & \underset{4}{f} \end{pmatrix} \begin{pmatrix} \overset{1}{1} & \overset{2}{3} \\ \underset{2}{2} & \underset{4}{4} \end{pmatrix} = \begin{pmatrix} \overset{11}{ax1+bx2} & \overset{12}{ax3+bx4} \\ \underset{21}{cx1+dx2} & \underset{22}{cx3+dx4} \\ \underset{31}{ex1+fx2} & \underset{32}{ex3+fx4} \end{pmatrix}$$

- ✓ 행렬 곱  $A \times B$ 에 대하여, 행렬 A의 열 수와 행렬 B의 행수는 같아야 한다.
- ✓ 행렬 곱  $A \times B$ 를 계산한 행렬 AB의 크기는 A의 행 개수와 B의 열 개수가 된다.

## 3-1. 행렬 곱

1.x

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



```
X = tf.placeholder(tf.float32, [None, 3])
print(X)
```



```
x_data = [[1, 2, 3], [4, 5, 6]]
```

```
W = tf.Variable(tf.random_normal([3, 2]))
```

```
expr = tf.matmul(X, W)
```

```
sess = tf.Session()
```

```
sess.run(tf.global_variables_initializer())
```

```
print("=== x_data ===")
```

```
print(x_data)
```

```
print("=== W ===")
```

```
print(sess.run(W))
```

```
print("=== expr ===")
```

```
print(sess.run(expr, feed_dict={X: x_data}))
```

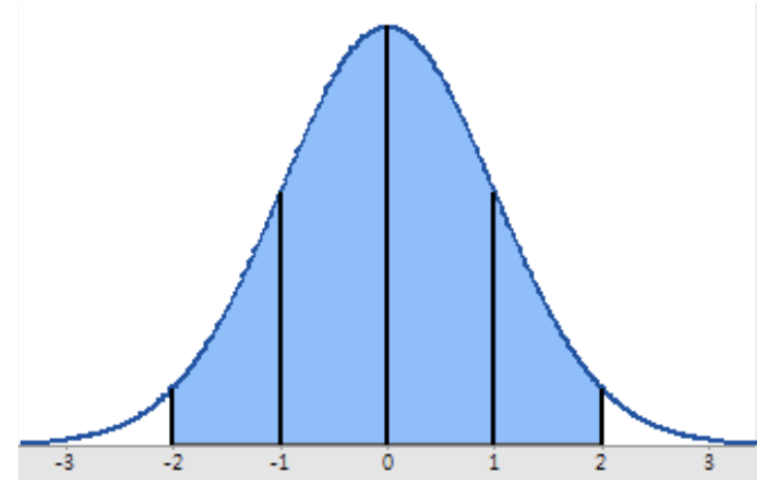
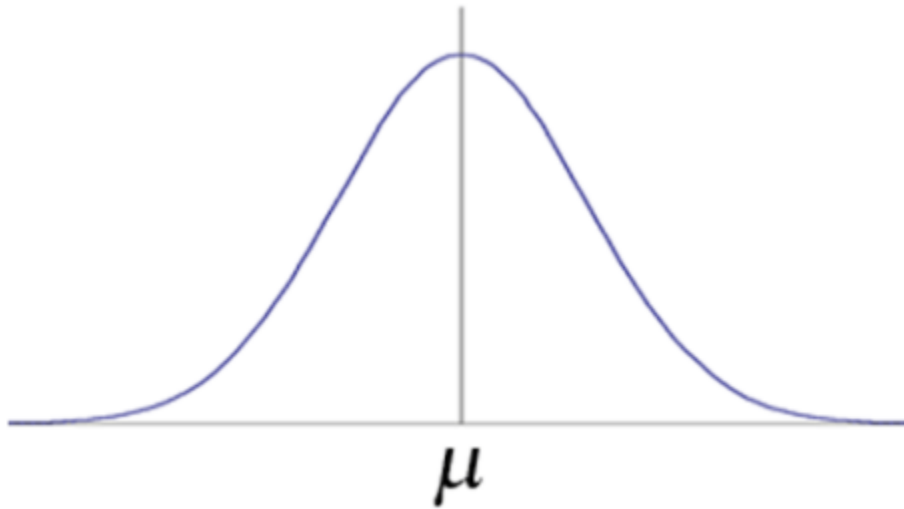
```
sess.close()
```

출력

```
Tensor("Placeholder_45:0", shape=(?, 3), dtype=float32)
=== x_data ===
[[1, 2, 3], [4, 5, 6]]
=== W ===
[[ 0.29691058 -0.30573136]
 [ 0.6454115   0.92965597]
 [-0.39686683 -0.7264136 ]]
=== expr ===
[[ 0.397133   -0.6256602 ]
 [ 2.0334985  -0.93312716]]
```

## 3-1. 행렬 곱

### 정규분포



```
W = tf.Variable(tf.random_normal([3, 2]))
```

- ✓ **random\_normal** : 정규확률분포 값을 생성해주는 함수
- ✓ 기본값은 평균값 0 표준편차 1

## 3-1. 행렬 곱

1.x

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

X = tf.placeholder(tf.float32, [None, 3])
print(X)

x_data = [[1, 2, 3], [4, 5, 6]]

W = tf.Variable(tf.random_normal([3, 2]))

✓ expr = tf.matmul(X, W)
sess = tf.Session()
sess.run(tf.global_variables_initializer())

print("=== x_data ===")
print(x_data)
print("=== W ===")
print(sess.run(W))
print("=== expr ===")

✓ print(sess.run(expr, feed_dict={X: x_data}))

sess.close()
```

출력

```
Tensor("Placeholder_45:0", shape=(?, 3), dtype=float32)
=== x_data ===
[[1, 2, 3], [4, 5, 6]]
=== W ===
[[ 0.29691058 -0.30573136]
 [ 0.6454115   0.92965597]
 [-0.39686683 -0.7264136 ]]
=== expr ===
[[ 0.397133   -0.6256602 ]
 [ 2.0334985  -0.93312716]]
```

$$\begin{pmatrix} \overset{1}{a} & \overset{2}{b} & \overset{3}{c} \\ \underset{1}{d} & \underset{2}{e} & \underset{3}{f} \end{pmatrix} \begin{pmatrix} \overset{1}{1} & \overset{2}{4} \\ \underset{1}{2} & \underset{2}{5} \\ \underset{1}{3} & \underset{2}{6} \end{pmatrix} = \begin{pmatrix} \overset{11}{ax1+bx2+cx3} & \overset{12}{ax4+bx5+cx6} \\ \underset{21}{dx1+ex2+fx3} & \underset{22}{dx4+ex5+fx6} \end{pmatrix}$$

## 3-1. 행렬 곱

2.x

```
import tensorflow as tf

@tf.function
def expr(x):
    re = tf.matmul(x,W)
    return re

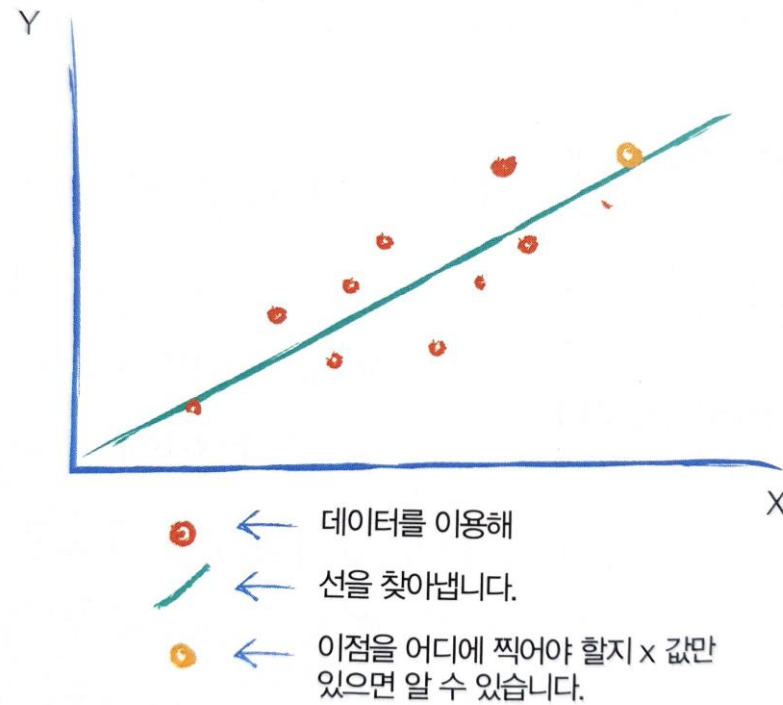
W = tf.Variable(tf.random.normal([3, 2]))
X = tf.constant([[1, 2, 3], [4, 5, 6]],dtype="float")

print("=== x ===")
print(X)
print("=== W ===")
print(W)
print("=== expr ===")
print(expr(X))
```

출력

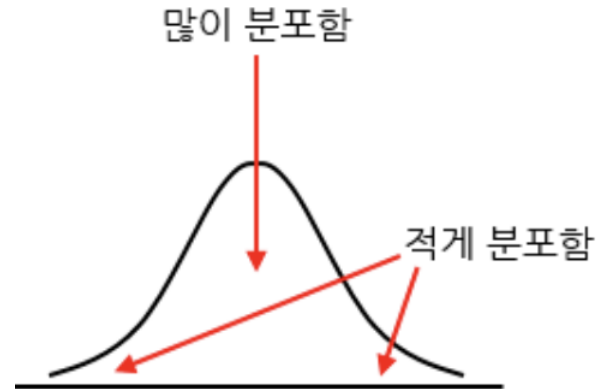
```
=== x ===
tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
=== W ===
<tf.Variable 'Variable:0' shape=(3, 2) dtype=float32, numpy=
array([[ 0.8868826 , -0.9164401 ],
        [ 1.323665  , -0.03584254],
        [ 0.7964435 ,  1.0625445 ]], dtype=float32)>
=== expr ===
tf.Tensor(
[[ 5.923543  2.1995082]
 [14.944517  2.530294 ]], shape=(2, 2), dtype=float32)
```

## 3-2. 선형회귀

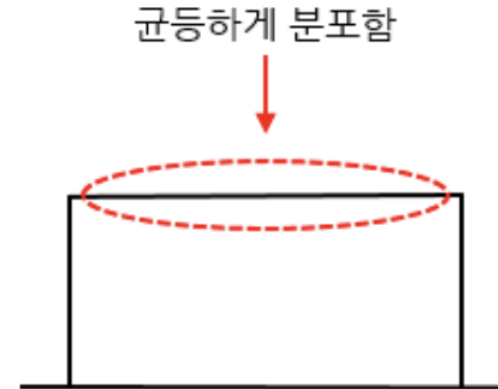


✓ X와 Y의 상관관계 분석

## 3-2. 선형회귀



< 정규분포 >



< 균등분포 >

- ✓ `random_uniform`: 지정한 값 사이의 균등분포를 가진 무작위 값으로 초기화
- ✓ 하한 구간을 포함되지만, 상한은 포함되지 않음
- ✓ 실수형 기본 구간은 0~1

## 3-2. 선형회귀



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

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

W = tf.Variable(tf.random_uniform([1], -1.0,1.0))
b = tf.Variable(tf.random_uniform([1], -1.0,1.0))

X = tf.placeholder(tf.float32, name = "X")
Y = tf.placeholder(tf.float32, name = "Y")

result = W * X + b

cost = tf.reduce_mean(tf.square(result - Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = optimizer.minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(100):
        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y:y_data})

        print(step, cost_val, sess.run(W), sess.run(b))

    print("\n=== Test ===")
    print("X : 5, Y: ", sess.run(result, feed_dict={X: 5}))
    print("X : 2.5, Y : ", sess.run(result, feed_dict={X: 2.5}))
```



## 3-2. 선형회귀

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

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

W = tf.Variable(tf.random_uniform([1], -1.0,1.0))
b = tf.Variable(tf.random_uniform([1], -1.0,1.0))

✓ X = tf.placeholder(tf.float32, name ="X")
  Y = tf.placeholder(tf.float32, name = "Y")

✓ result = W * X + b

cost = tf.reduce_mean(tf.square(result - Y))
✓ optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
  train_op = optimizer.minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(100):
        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y:y_data})

        print(step, cost_val, sess.run(W), sess.run(b))

    print("\n=== Test ===")
    print("X : 5, Y: ", sess.run(result, feed_dict={X: 5}))
    print("X : 2.5, Y : ", sess.run(result, feed_dict={X: 2.5}))
```

## 3-2. 선형회귀

손실함수

$result = W * X + b$

```
cost = tf.reduce_mean(tf.square(result - Y))
```

- ✓ 한 쌍(  $x, y$  )의 데이터에 대한 손실값을 계산하는 부분
- ✓ 손실값 : 실제값과 예측한 값이 얼마나 차이나는가를 나타낸 값

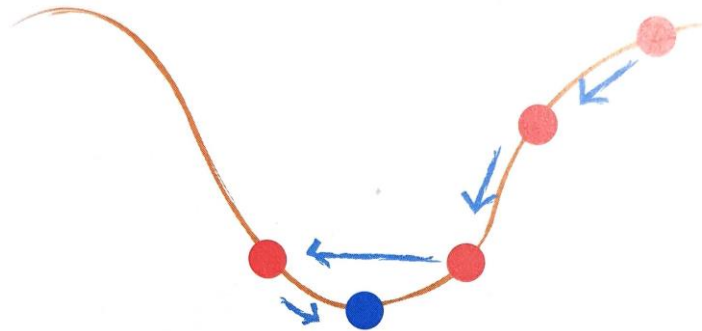
=> 손실값을 최소화하기 위해 다양한 변수들을 넣어 계산해보며 학습하는 것

## 3-2. 선형회귀

연산 그래프

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)  
train_op = optimizer.minimize(cost)
```

- ✓ 최적화 함수 : 가중치와 편향 값을 변경해가면서 손실값을 최소화하는 값을 찾아주는 함수
- ✓ 경사하강법 사용 (기본적인 알고리즘)



## 3-2. 선형회귀

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

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

W = tf.Variable(tf.random_uniform([1], -1.0,1.0))
b = tf.Variable(tf.random_uniform([1], -1.0,1.0))

X = tf.placeholder(tf.float32, name = "X")
Y = tf.placeholder(tf.float32, name = "Y")

result = W * X + b

cost = tf.reduce_mean(tf.square(result - Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = optimizer.minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(100):
        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y:y_data})

        print(step, cost_val, sess.run(W), sess.run(b))

    print("\n=== Test ===")
    print("X : 5, Y : ", sess.run(result, feed_dict={X: 5}))
    print("X : 2.5, Y : ", sess.run(result, feed_dict={X: 2.5}))
```

### 출력

```
0 0.9778037 [1.2726939] [-0.5083914]
1 0.050943363 [1.2215362] [-0.51579064]
2 0.038006797 [1.2210853] [-0.501247]
3 0.03607583 [1.2152379] [-0.4894317]
4 0.034360718 [1.2101219] [-0.4776405]
5 0.032728534 [1.2050643] [-0.46616116]
6 0.03117388 [1.2001355] [-0.45495465]
7 0.02969311 [1.1953242] [-0.44401792]
8 0.028282663 [1.1906288] [-0.433344]
9 0.026939208 [1.1860462] [-0.4229267]
10 0.025659597 [1.1815737] [-0.41275987]
11 0.024440741 [1.1772089] [-0.4028374]
12 0.023279795 [1.1729488] [-0.3931535]
13 0.02217398 [1.1687913] [-0.38370234]
14 0.021120703 [1.1647336] [-0.3744784]
15 0.020117452 [1.1607736] [-0.3654762]
16 0.019161852 [1.1569088] [-0.3566904]
17 0.018251656 [1.1531367] [-0.34811583]
18 0.017384678 [1.1494554] [-0.33974737]
19 0.016558893 [1.1458627] [-0.33158004]
20 0.015772345 [1.1423562] [-0.3236091]
```

## 3-2. 선형회귀

=== Test ===

X: 5, Y: [4.9823065]  
X: 2.5, Y: [2.4985285]

학습 횟수 / 손실값

➤ 100 / 0.0176935

=== Test ===

X: 5, Y: [4.9994907]  
X: 2.5, Y: [2.4999576]

➤ 200 / 0.0005093

=== Test ===

X: 5, Y: [4.999844]  
X: 2.5, Y: [2.4999871]

➤ 300 / 0.000156

=== Test ===

X: 5, Y: [5.000028]  
X: 2.5, Y: [2.5000026]

➤ 400 / 0.000028

=== Test ===

X: 5, Y: [4.999992]  
X: 2.5, Y: [2.4999993]

➤ 500 / 0.000008

=== Test ===

X: 5, Y: [4.9999995]  
X: 2.5, Y: [2.4999998]

➤ 600 / 0.0000005

=== Test ===

X: 5, Y: [5.]  
X: 2.5, Y: [2.5]

➤ 700 / 0



< 감사합니다. >

91914145 장혜선  
07.06 발표 끝.

