

Deep Learning with Python

Chapter 1 딥러닝 기초 01

■ 목차

1. Perceptron의 구조

- a. Perceptron의 구조
- b. Perceptron의 가중치(Weight)
- c. Perceptron의 활성화 함수(Activation Function)

2. Linear Regression

- a. Regression이란?
- b. Linear Regression 설계 요소
- c. Tensorflow를 이용한 Linear Regression 구현

3. Multi-Variable Linear Regression

- a. Multi-Variable이란?
- b. Multi-Variable Linear Regression 설계 요소
- c. Tensorflow를 이용한 Multi-Variable Linear Regression 구현

■ 학습목표

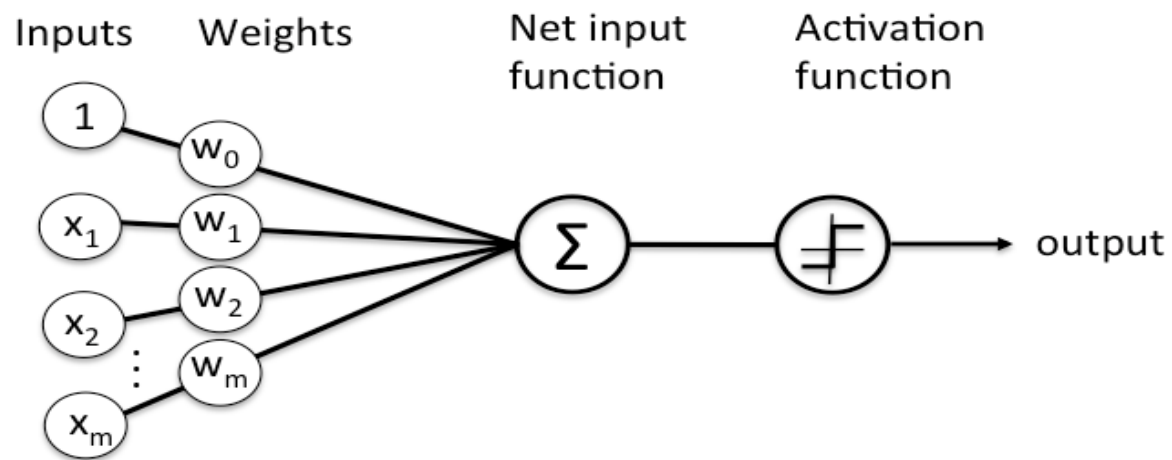
1. Perceptron의 구조를 배우고 활성화 함수의 역할을 이해한다.
2. Regression의 개념을 이해하고 Tensorflow를 이용하여 Linear Regression을 구현한다.
3. Multi-Variable을 이해하고 Tensorflow를 이용하여 Multi-Variable Linear Regression을 구현한다.

1. Perceptron의 구조

1. Perceptron의 구조

Perceptron이란?

- Frank Rosenblatt이 1957년에 고안한 딥러닝의 기원이 되는 알고리즘
- 다수의 신호를 입력으로 받아 하나의 신호를 출력한다.
- 신호는 '흐른다/안 흐른다(1 또는 0)'의 두가지 값을 갖는다.

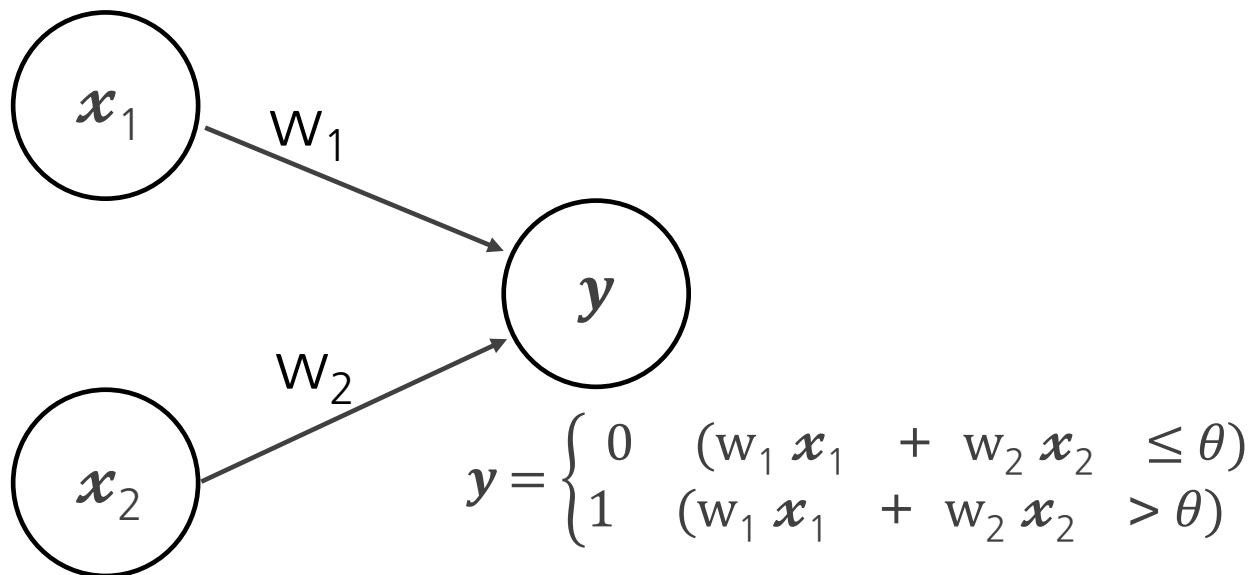


Schematic of Rosenblatt's perceptron.

1. Perceptron의 구조

+ 입력이 두 개인 Perceptron

- 입력 신호가 뉴런에 보내질 때 각각 고유한 가중치(Weight)가 곱해진다.
- 뉴런에서 보내온 신호의 총합이 임계값(θ)을 넘을 때 1을 출력하며 이를 뉴런의 활성화(Activation)라고 한다.

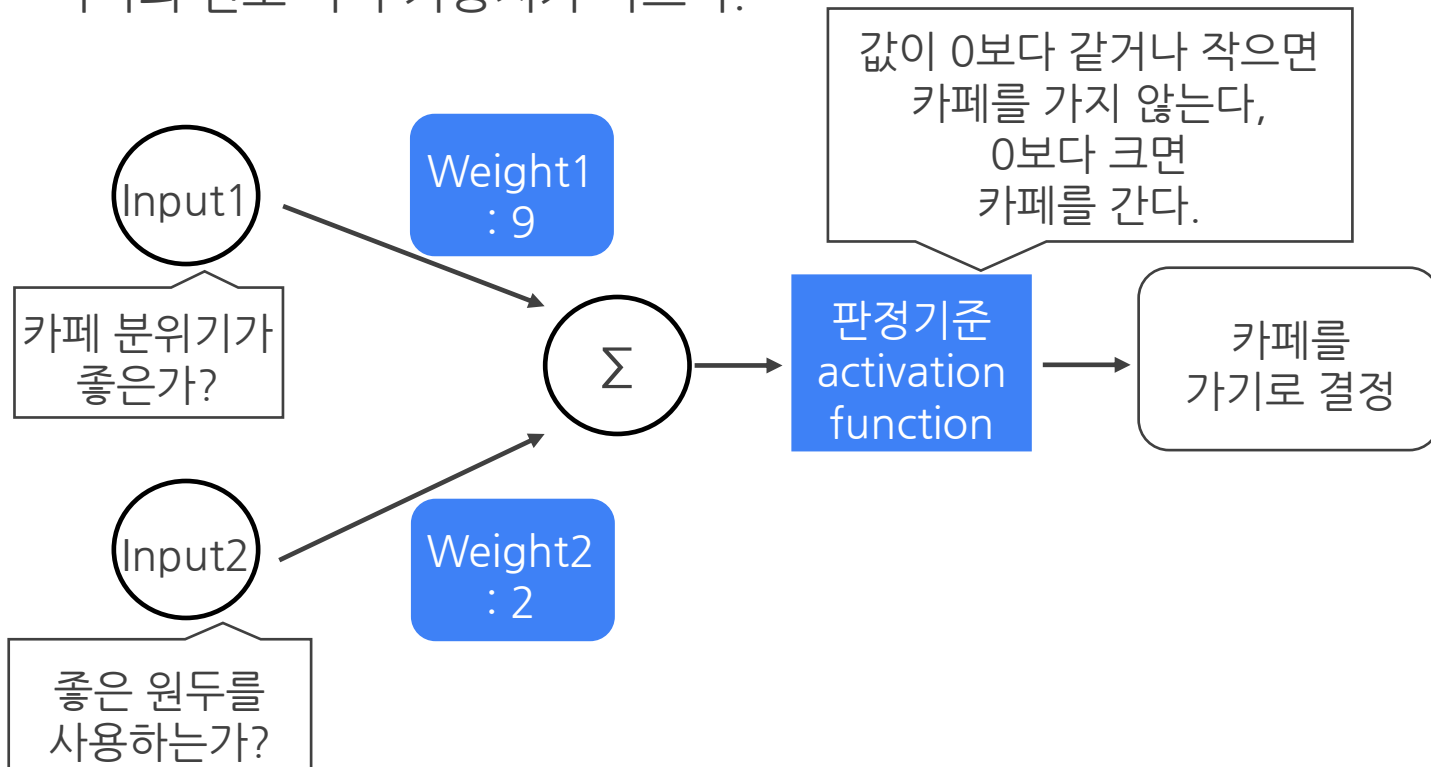


✓ x : 입력 신호, y : 출력 신호, w : 가중치

1. Perceptron의 구조

Perceptron 도식화 예

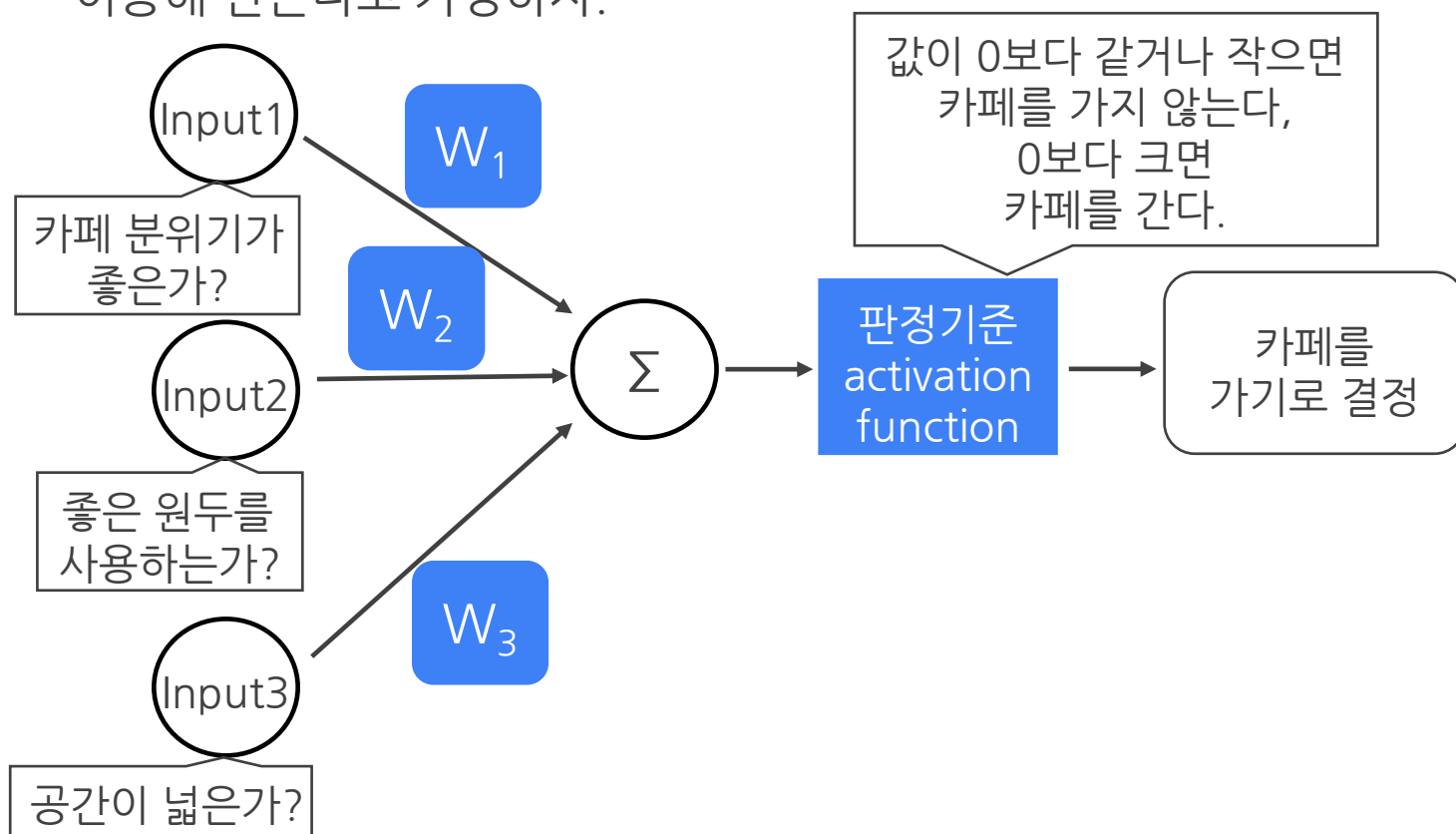
- 퍼셉트론은 복수의 입력신호 각각에 가중치를 부여한다.
- 가중치는 각 신호가 결과에 주는 영향력을 조절하는 요소이다.
- 각각의 신호마다 가중치가 다르다.



1. Perceptron의 구조

Perceptron의 가중치(Weight) (1/2)

- 데이트 코스로 카페를 갈 것인지 의사결정하는 모델을 퍼셉트론을 이용해 만든다고 가정하자.



1. Perceptron의 구조



Perceptron의 가중치(Weight) (2/2)

- 의사결정 시 고려사항을 앞의 예와 같이 3가지라고 가정하자.

1. 카페 분위기가 좋은가? (W_1)

2. 좋은 원두를 사용하는가? (W_2)

3. 공간이 넓은가? (W_3)

- 가중치는 Input의 중요도(영향력)를 나타낸다.

- 분위기를 가장 고려한다면 $W_1 = 5$, $W_2 = 2.5$, $W_3 = 2.5$ 로 설정할 수 있다.

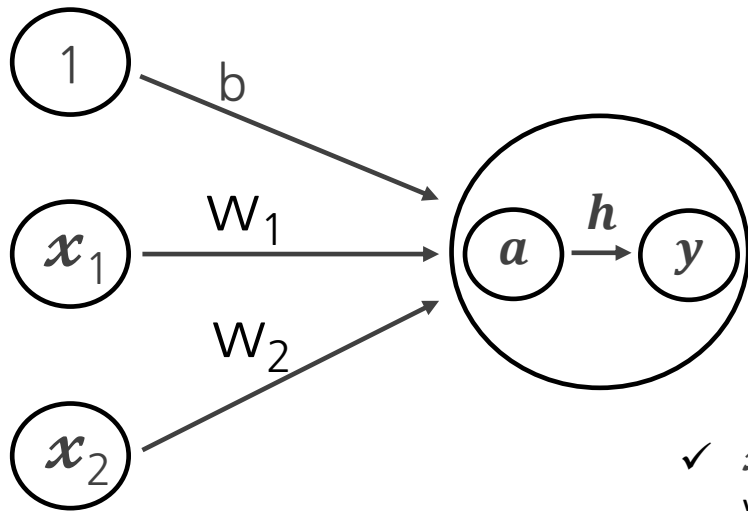
- 원두를 가장 고려한다면 $W_1 = 2$, $W_2 = 6$, $W_3 = 2$ 로 설정할 수 있다.

- 공간이 넓은지를 가장 고려한다면 $W_1 = 2$, $W_2 = 1$, $W_3 = 7$ 로 설정할 수 있다.

1. Perceptron의 구조

Perceptron의 활성화 함수(activation function) (1/2)

- 활성화 함수는 개별 뉴런에 들어오는 입력 신호의 총합을 출력 신호로 변환하는 역할을 한다.
- 활성화 함수를 구현하기 위해 편향(bias)이 적용된 퍼셉트론을 사용한다.
- 편향은 뉴런이 얼마나 쉽게 활성화하는지를 조정하는 역할을 한다.
- 입력 신호가 두 개인 경우 활성화 함수 처리 과정은 다음과 같다.



$$a = b + w_1 x_1 + w_2 x_2$$

$$h(a) = y$$

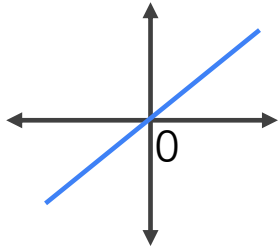
✓ x : 입력 신호, y : 출력 신호, a : 총합,
 w : 가중치, b : 편향, $h(a)$: 활성화 함수

1. Perceptron의 구조

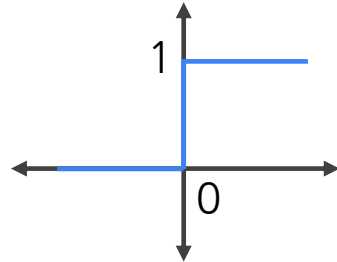
Perceptron의 활성화 함수(activation function) (2/2)

- 활성화 함수에는 여러 종류가 있다.

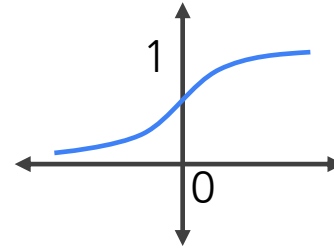
1. Linear function



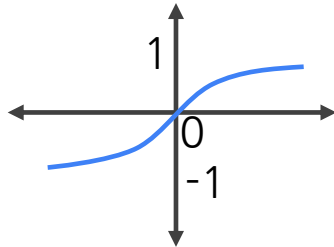
2. Step function



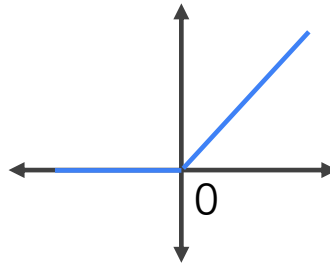
3. Sigmoid function



4. Hyperbolic tangent function



5. ReLU function

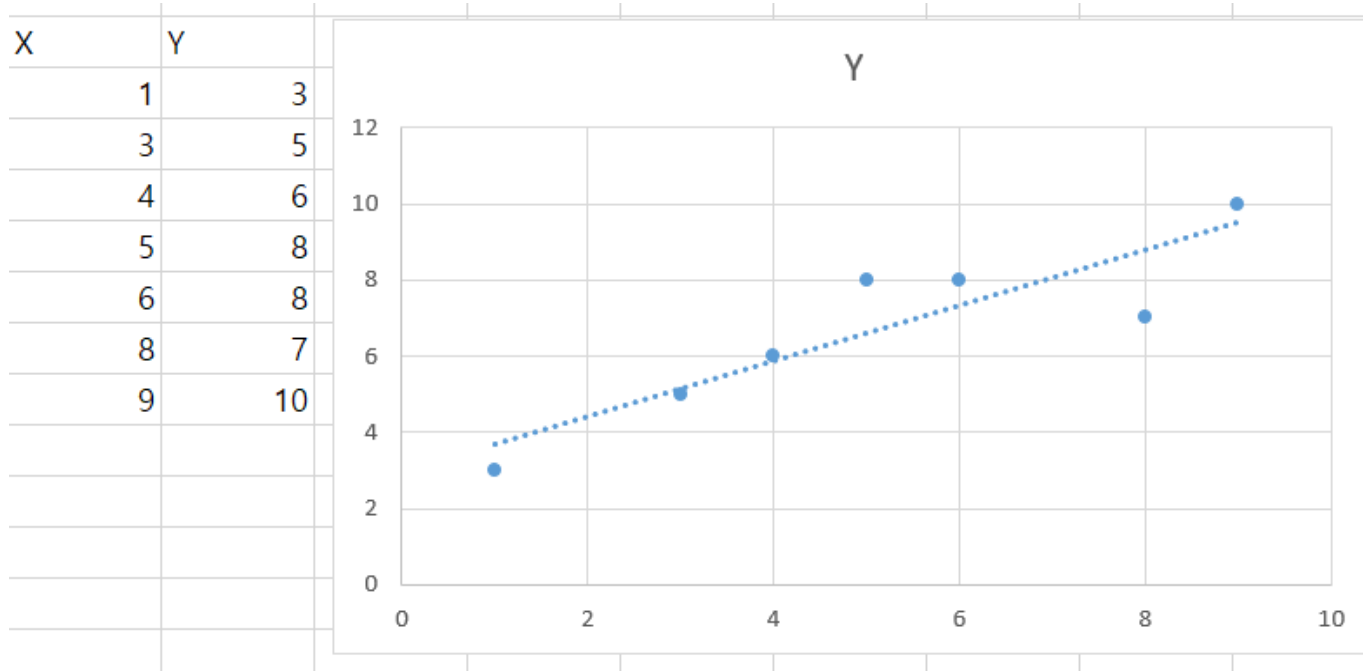


2. Linear Regression

2. Linear Regression

Regression이란? (1/2)

- 그래프 상에 무작위로 흩어져있는 데이터의 경향을 찾아 선으로 나타내는 것, 선이 직선 형태일 경우 Linear Regression이라 하며 Prediction(예측)의 한 방법이다.



예) X: 시간(s), Y: 평균 속도(m/s)

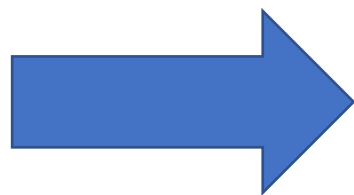
2. Linear Regression



Regression이란? (2/2)

- 데이터의 경향을 찾는다는 것은 새로운 데이터가 들어왔을 때 해당하는 값을 예측하는 것을 의미한다.
- 데이터의 경향을 찾기 위해서 퍼셉트론 모델을 사용한다.
- 여기서 X변수는 Y변수의 변화를 예측할 수 있다고 여겨지는 변수로 특징(Feature)이라 한다.
- Y변수는 결과로 간주되는 변수로 반응(Response)이라 한다.

X 변수



Y 변수

2. Linear Regression

Linear Regression 설계 요소

1. 모델(출력 계산 방법)

- : 주어진 데이터를 예측하기 위해 설계하는 함수로 퍼셉트론 모델을 사용하여 가중치(Weight)와 편향(bias)을 학습한다.
- 가중치(Weight)는 뉴런(X)이 결과(Y)에 주는 영향력(중요도)을 조절하는 매개 변수이다.
- 편향(bias)은 뉴런이 얼마나 쉽게 활성화하는지를 조정한다.

2. 비용 함수(Cost Function)

- : 예측값과 실제값의 차이인 오차(Cost)를 어떻게 계산할 것인지 설정한다.

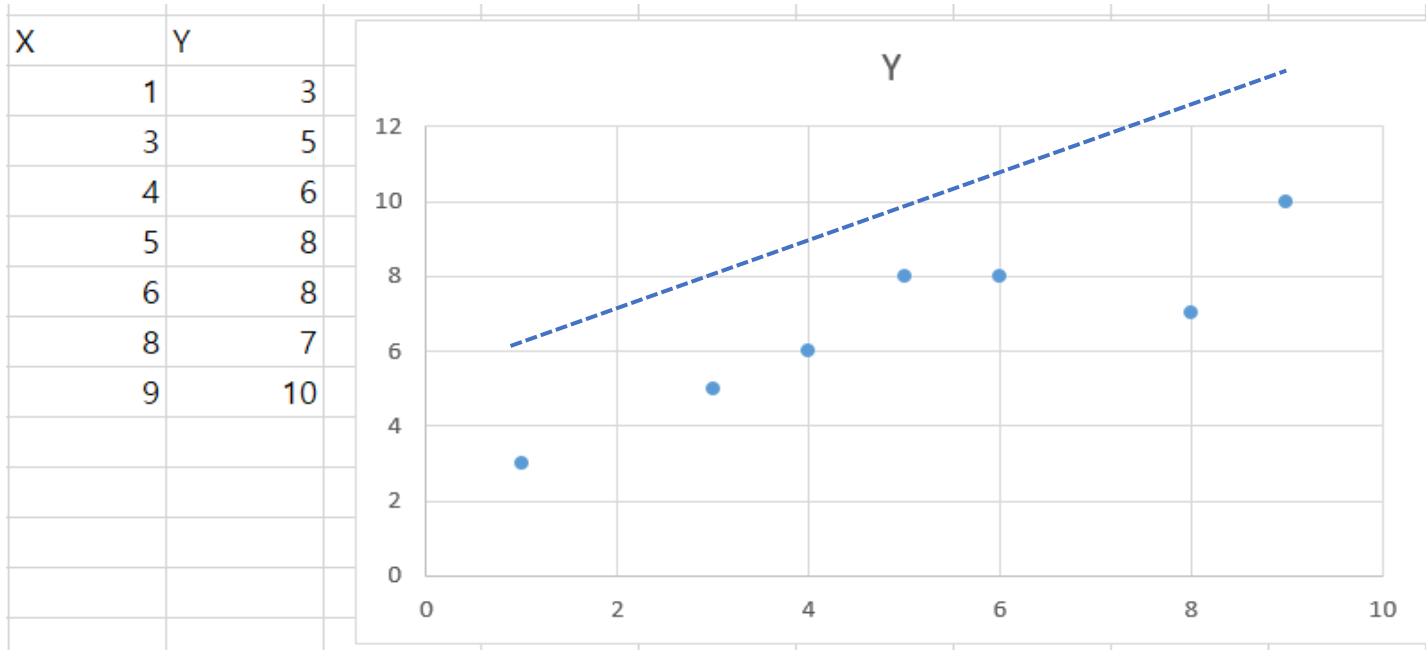
3. 최적화(Optimization)

- : Cost(오차)를 최소화하기 위해 경사하강법(Gradient Descent) 알고리즘을 사용한다.

2. Linear Regression

퍼셉트론 정의하기

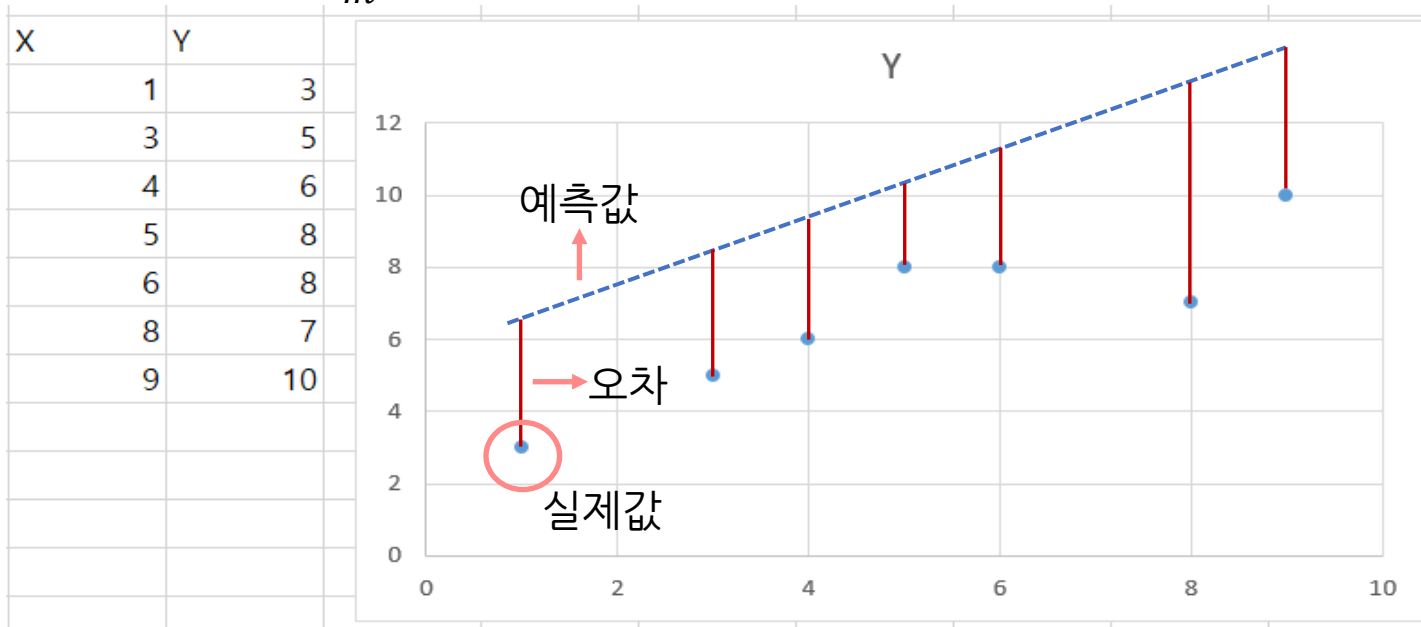
- 주어진 데이터를 가장 잘 나타내는 직선을 구하기 위해 가중치와 편향을 사용하여 모델을 정의한다.
- $H(x) = Wx + b$, W :가중치, b : 편향
- 위의 모델로 구한 값이 예측값이 된다.



2. Linear Regression

비용 함수(Cost Function) 정의하기

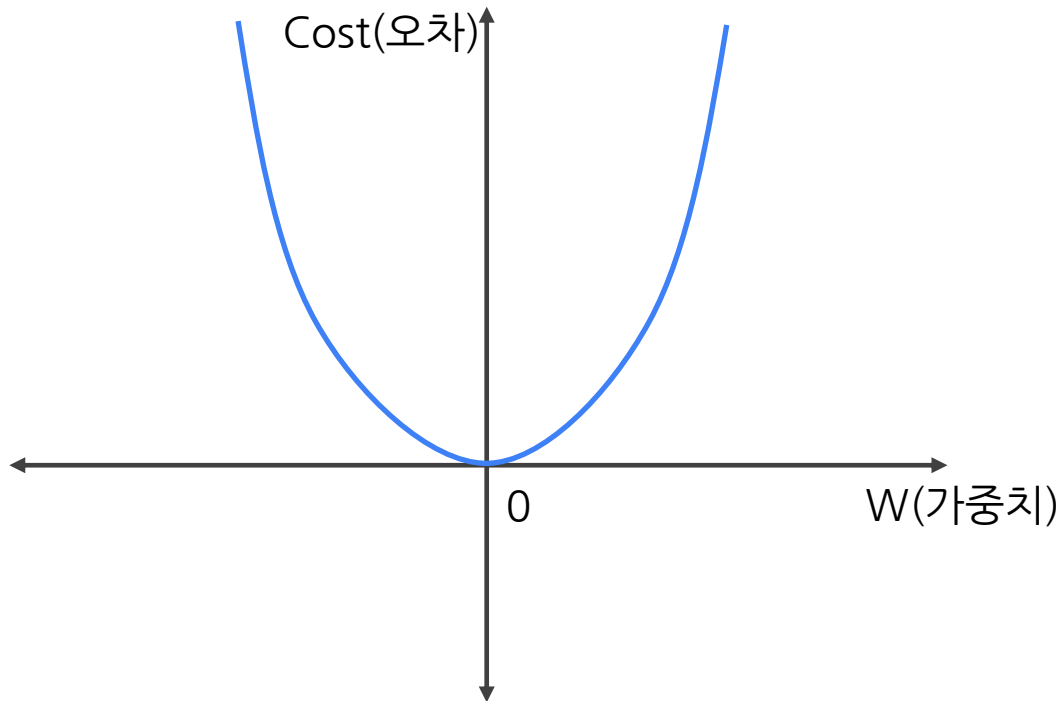
- 모델로 구한 예측값과 실제값(Y)의 차이인 오차를 어떻게 계산 할 것인지 설정한다.
- 비용 함수는 모델로 구한 예측값과 실제값(Y)의 차이를 제공하여 평균한다(Mean Squared Error, MSE).
- $cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$, $H(x) = Wx + b$



2. Linear Regression

최적화(Optimization) 정의하기 (1/3)

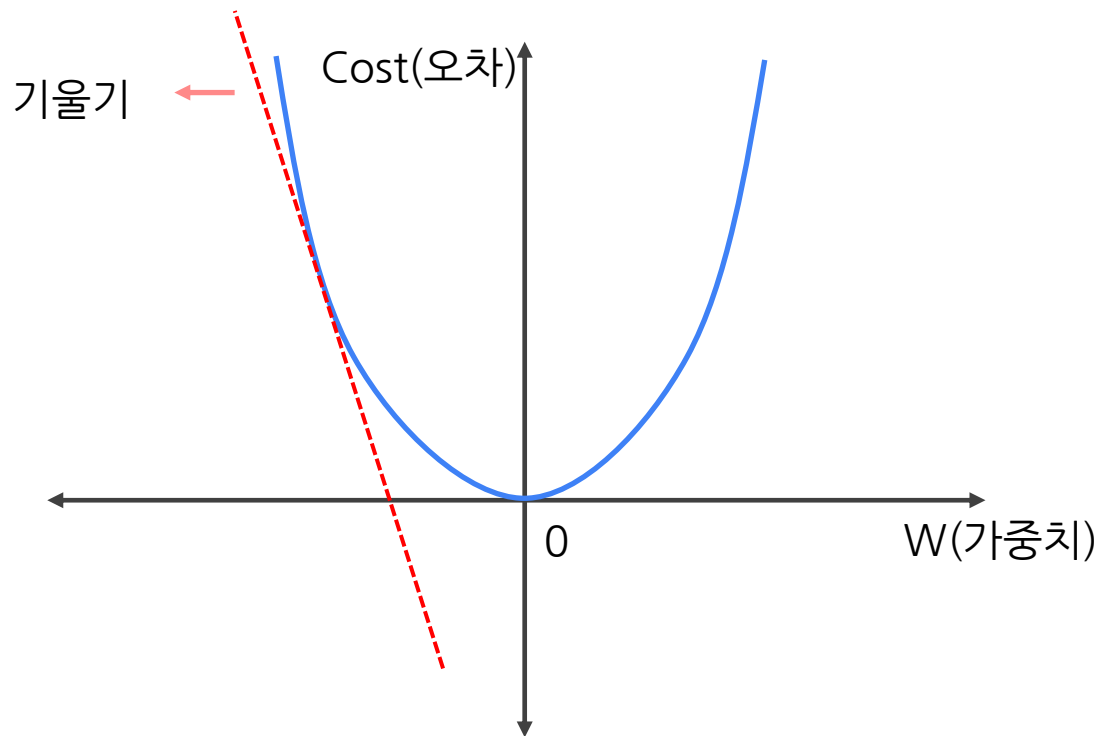
- 최적화한다는 것은 Cost(오차)를 최소화하는 것이며 이를 위해 W (가중치)와 b (편향)을 업데이트한다.
- 가중치(W)에 따른 Cost(오차) 그래프를 나타내면 아래와 같다.



2. Linear Regression

최적화(Optimization) 정의하기 (2/3)

- Cost(오차)를 최소화하는 방법으로 경사하강법(Gradient Descent) 알고리즘을 사용한다.

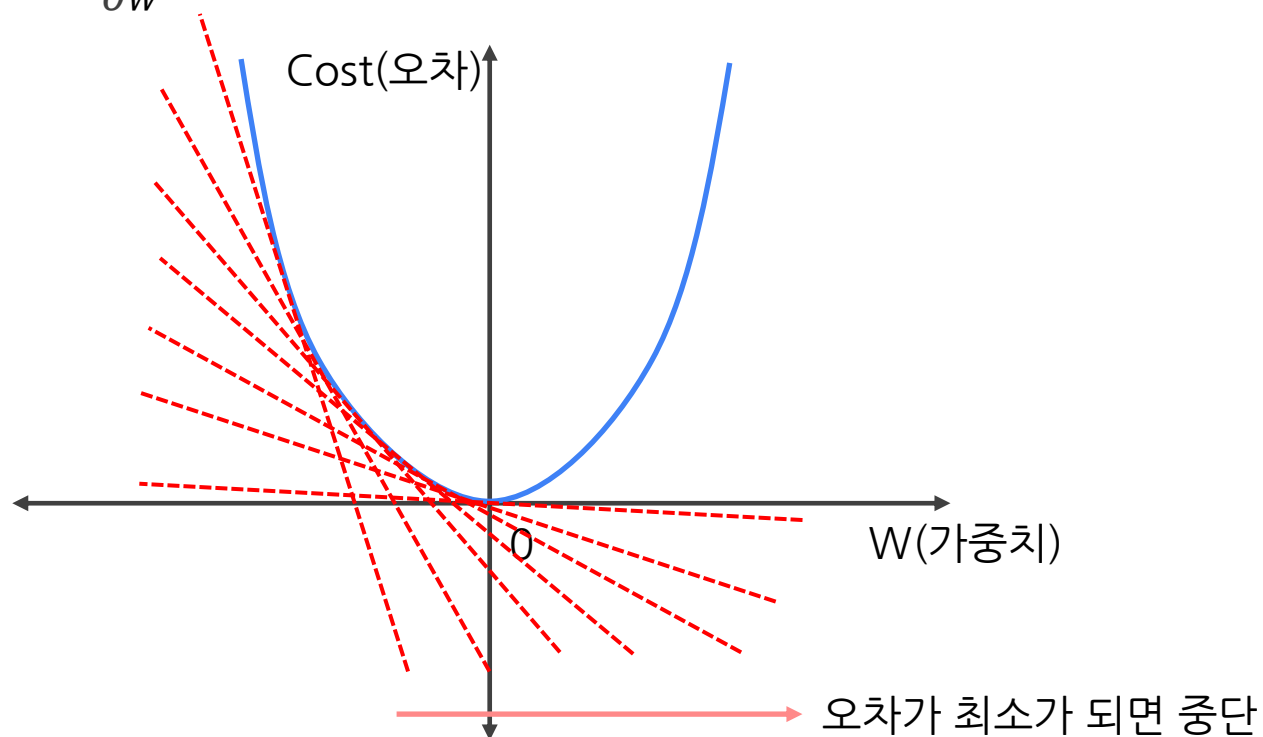


2. Linear Regression

최적화(Optimization) 정의하기 (3/3)

- 경사하강법(Gradient Descent) 알고리즘의 원리는 W (가중치)에 대한 미분값을 통해 W 값을 업데이트하여 오차의 최저점을 찾는 것이다.

$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$



2. Linear Regression

Tensorflow를 이용한 Linear Regression 설계하기

1. 그래프를 만든다. (Build graph)
2. session을 열고 sess.run()으로 그래프를 실행시킨다. (Run)
3. 반환되는 결과 값을 가지고 반복해서 변수를 업데이트하여 학습시킨다.
(Update)

2. Linear Regression

전체 코드 (1/2)

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

learning_rate = 0.01
training_cnt = 10

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

X = tf.placeholder("float")
Y = tf.placeholder("float")

W = tf.Variable([.0], tf.float32, name = "weight")
b = tf.Variable([.0], tf.float32, name = "bias")

pred = tf.add(tf.multiply(X, W), b)

cost = tf.reduce_mean(tf.pow(pred-Y, 2))
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)
```

2. Linear Regression

전체 코드 (2/2)

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_cnt):
    r_cost, r_W, r_b, _ = sess.run([cost, W, b, op_train],
    feed_dict = {X: train_X, Y: train_Y})
    print("Running count : " '%04d' % (epoch+1), "Training
cost =", r_cost, "W =", r_W, "b =", r_b)

print("Optimization Finished!")
print("Running count : " '%04d' % (epoch+1), "Training cost =",
r_cost, "W =", r_W, "b =", r_b)
```

2. Linear Regression

모델 구축(Build graph) 전체 코드 (1/8)

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

learning_rate = 0.01
training_cnt = 10

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

X = tf.placeholder("float")
Y = tf.placeholder("float")

W = tf.Variable([.0], tf.float32, name = "weight")
b = tf.Variable([.0], tf.float32, name = "bias")

pred = tf.add(tf.multiply(X, W), b)

cost = tf.reduce_mean(tf.pow(pred-Y, 2))
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)
```


2. Linear Regression

모델 구축(Build graph) (2/8)

라이브러리 import

```
# 필요한 library import
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

* 라이브러리 설치

```
(tf-py36) :~$ pip install tensorflow
(tf-py36) :~$ pip install numpy
(tf-py36) :~$ pip install matplotlib
```

2. Linear Regression

🔍 모델 구축(Build graph) (3/8)

✓ 파라미터 값 설정

- 학습을 위한 기초 파라미터
- learning_rate : 학습 정도를 조절하는 파라미터. 값이 너무 작으면 Train 되지 않을 수 있고 너무 크면 overshooting이 발생할 수 있다.
- training_cnt : data set에 대한 training 반복 횟수

파라미터값 설정

learning_rate = 0.01

training_cnt = 10

✓ 트레이닝 데이터 변수 선언

- 입력으로 들어가는 x data, y data 설정
- numpy array를 사용

Training Data

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

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

2. Linear Regression

모델 구축(Build graph) (4/8)

tf graph input

- placeholder : 학습용 데이터를 담는 그릇
- 값은 미정이나 어떤 타입, 데이터 구조를 갖는 데이터를 정의함, 즉 입력값을 받을 수 있음
- 아래는 어떤 데이터인지는 모르겠지만, float 타입 데이터를 정의 하겠다라는 것
- * **tf.placeholder() << 괄호 사이에 shift+tab을 누르면 사용 가능한 변수들이 보임**
- dtype : 저장되는 데이터 형 (실수, 정수 등 정의)
- shape : 행렬의 차원 정의, shape=[3,3]으로 정의하면, 이 플레이스홀더는 3x3행렬을 저장
- name : 플레이스홀더의 이름 정의

```
X = tf.placeholder("float")  
Y = tf.placeholder("float")
```

2. Linear Regression

🔍 모델 구축(Build graph) (5/8)

✓ set model weight

- variable : 변수를 의미하며 임의의 값을 지정
- 그래프를 계산하면서 최적화 할 변수들(신경망을 좌우하는 값) 입력 시 사용

```
# Set model weights
```

```
W = tf.Variable([.0], tf.float32, name = "weight")  
b = tf.Variable([.0], tf.float32, name = "bias")
```

↑ ↑ ↑
초기값, data type, 이름 정의

✓ linear model (예측 값 구현)

$$H(x) = Wx + b$$

```
# Construct a linear model (Wx+b)
```

```
pred = tf.add(tf.multiply(X, W), b)
```

or

```
pred = X * W + b
```

2. Linear Regression

🔍 모델 구축(Build graph) (6/8)

☑ cost/loss function 구현

$$\text{cost}(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

cost/loss function

```
cost = tf.reduce_mean(tf.pow(pred - Y, 2))
```

or

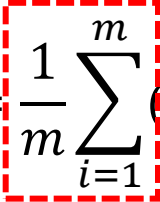
```
cost = tf.reduce_mean(tf.square(pred - Y))
```

2. Linear Regression

🔍 모델 구축(Build graph) (7/8)

* reduce_mean?

- 들어오는 입력의 평균값을 구하는 함수


$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

```
t = [ 1., 2., 3., 4.]  
r_m = tf.reduce_mean(t)
```

```
sess = tf.Session()  
sess.run(tf.global_variables_initializer())  
sess.run(r_m)
```

2.5

2. Linear Regression

모델 구축(Build graph) (8/8)

- ✓ 학습 방법 → cost를 최소화
 - GradientDescent 함수 사용 (경사하강법)

```
cost = tf.reduce_mean(tf.pow(pred-Y, 2))  
optimizer = tf.train.GradientDescentOptimizer(learning_rate)  
op_train = optimizer.minimize(cost)
```

2. Linear Regression

모델 실행(run/update) 전체 코드 (1/3)

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_cnt):
    r_cost, r_W, r_b, _ = sess.run([cost, W, b, op_train],
    feed_dict = {X: train_X, Y: train_Y})
    print("Running count : " '%04d' % (epoch+1), "Training
cost =", r_cost, "W =", r_W, "b =", r_b)

print("Optimization Finished!")
print("Running count : " '%04d' % (epoch+1), "Training cost
=", r_cost, "W =", r_W, "b =", r_b)
```


2. Linear Regression

🔍 모델 실행(run/update) (2/3)

Tensorflow 세션 생성

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

변수들에 초기값 할당

```
init = tf.global_variables_initializer()
```

```
sess.run(init)
```

for epoch **in** range(training_cnt): *# training_cnt 횟수만큼 반복 실행*

```
    r_cost, r_W, r_b, _ = sess.run([cost, W, b, op_train],
```

```
    feed_dict = {X: train_X, Y: train_Y})
```

#sess.run을 통해 학습한(변경된) cost, W, b 값을 r_cost, r_W, r_b

에 update

```
    print("Running count : " '%04d' % (epoch+1), "Training cost
```

```
=", r_cost, "W =", r_W, "b =", r_b)
```

sess.run을 통해 학습된 값 출력

```
print("Optimization Finished!")
```

```
print("Running count : " '%04d' % (epoch+1), "Training cost =",
```

```
r_cost, "W =", r_W, "b =", r_b)
```

2. Linear Regression

모델 실행(run/update) (3/3)

```
Running count : 0001 Training cost = 4.6666665 W = [0.09333334] b = [0.04]
Running count : 0002 Training cost = 3.6927407 W = [0.17635557] b = [0.07546666]
Running count : 0003 Training cost = 2.9228852 W = [0.2502104] b = [0.10690311]
Running count : 0004 Training cost = 2.314336 W = [0.31591463] b = [0.13475662]
Running count : 0005 Training cost = 1.8332924 W = [0.37437233] b = [0.1594249]
Running count : 0006 Training cost = 1.4530338 W = [0.42638725] b = [0.18126151]
Running count : 0007 Training cost = 1.1524409 W = [0.47267398] b = [0.20058079]
Running count : 0008 Training cost = 0.9148195 W = [0.51386786] b = [0.21766222]
Running count : 0009 Training cost = 0.72697395 W = [0.5505337] b = [0.23275426]
Running count : 0010 Training cost = 0.5784736 W = [0.58317375] b = [0.24607782]
Optimization Finished!
Running count : 0010 Training cost = 0.5784736 W = [0.58317375] b = [0.24607782]
```

2. Linear Regression

def(함수) 사용하여 코딩하기 (1/4)

```
def make_graph() :  
    global X, Y, W, b, pred ,cost,optimizer, op_train  
    # 전역변수 선언  
  
    X = tf.placeholder("float")  
    Y = tf.placeholder("float")  
  
    W = tf.Variable([.0], tf.float32, name = "weight")  
    b = tf.Variable([.0], tf.float32, name = "bias")  
  
    pred = tf.add(tf.multiply(X, W), b)  
  
    cost = tf.reduce_mean(tf.pow(pred-Y, 2))  
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)  
    op_train = optimizer.minimize(cost)
```

2. Linear Regression

def(함수) 사용하여 코딩하기 (2/4)

```
def run_graph() :  
  
    sess = tf.Session()  
    init = tf.global_variables_initializer()  
    sess.run(init)  
  
    for epoch in range(training_cnt):  
        r_cost, r_W, r_b, _ = sess.run([cost, W, b, op_train],  
                                         feed_dict = {X: train_X, Y: train_Y})  
        print("Running count : " '%04d' % (epoch+1), "Training  
cost =", r_cost, "W =", r_W, "b =", r_b)  
  
    print("Optimization Finished!")  
    print("Running count : " '%04d' % (epoch+1), "Training cost  
=", r_cost, "W =", r_W, "b =", r_b)
```

2. Linear Regression

def(함수) 사용하여 코딩하기 (3/4)

- 함수 실행하기

```
make_graph()
```

```
run_graph()
```

```
Running count : 0001 Training cost = 4.6666665 W = [0.09333334] b = [0.04]
Running count : 0002 Training cost = 3.6927407 W = [0.17635557] b = [0.07546666]
Running count : 0003 Training cost = 2.9228852 W = [0.2502104] b = [0.10690311]
Running count : 0004 Training cost = 2.314336 W = [0.31591463] b = [0.13475662]
Running count : 0005 Training cost = 1.8332924 W = [0.37437233] b = [0.1594249]
Running count : 0006 Training cost = 1.4530338 W = [0.42638725] b = [0.18126151]
Running count : 0007 Training cost = 1.1524409 W = [0.47267398] b = [0.20058079]
Running count : 0008 Training cost = 0.9148195 W = [0.51386786] b = [0.21766222]
Running count : 0009 Training cost = 0.72697395 W = [0.5505337] b = [0.23275426]
Running count : 0010 Training cost = 0.5784736 W = [0.58317375] b = [0.24607782]
Optimization Finished!
Running count : 0010 Training cost = 0.5784736 W = [0.58317375] b = [0.24607782]
```

2. Linear Regression

def(함수) 사용하여 코딩하기 (4/4)

- 학습 횟수 -> 100

```
learning_rate = 0.01  
training_cnt = 100
```

```
Running count : 0010 Training cost = 0.5784736 W = [0.58317375] b = [0.24607782]  
Running count : 0020 Training cost = 0.07108992 W = [0.76570153] b = [0.31615967]  
Running count : 0030 Training cost = 0.02202488 W = [0.82453835] b = [0.33207244]  
Running count : 0040 Training cost = 0.016636007 W = [0.84514725] b = [0.33140582]  
Running count : 0050 Training cost = 0.015439604 W = [0.85390174] b = [0.32575577]  
Running count : 0060 Training cost = 0.014674502 W = [0.85894114] b = [0.31869748]  
Running count : 0070 Training cost = 0.013981109 W = [0.8627786] b = [0.311331]  
Running count : 0080 Training cost = 0.013323699 W = [0.8661908] b = [0.30399284]  
Running count : 0090 Training cost = 0.012697489 W = [0.86941886] b = [0.29678383]  
Running count : 0100 Training cost = 0.012100749 W = [0.8725385] b = [0.2897323]  
Optimization Finished!  
Running count : 0100 Training cost = 0.012100749 W = [0.8725385] b = [0.2897323]
```

2. Linear Regression

🔍 시각화 - 10번 학습 (1/4)

```
# Graphic display
plt.figure(figsize=(10,8))
plt.plot(train_X, train_Y,
label='y_data',marker="o",color='r',linewidth=1.5)

pagination('....')

# 학습( training_cnt 횟수만큼)
for epoch in range(training_cnt):      # training_cnt = 10
    r_cost, r_W, r_b, _ = sess.run([cost, W, b, op_train],
    feed_dict = {X: train_X, Y: train_Y})

    if (epoch+1) % display_step == 0:  # 나머지가 0이될 때 결과 출력
        print("Running count : " '%04d' % (epoch+1), "Training
cost =", r_cost, "W =", r_W, "b =", r_b)
```

2. Linear Regression

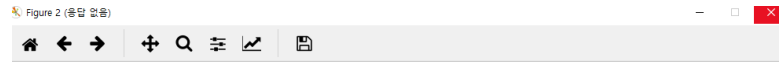
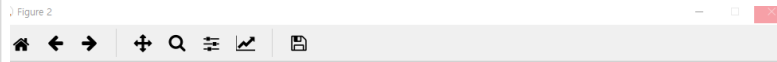
시각화 - 10번 학습 (2/4)

```
# Graphic display
plt.plot(train_X, sess.run(W) * train_X + sess.run(b),
marker = '^')
plt.title('running_count : [%04d], traing cost = [%0.6f],
weight = [%0.6f], bais = [%0.6f]' % (epoch+1, r_cost, r_W, r_b))
plt.pause(0.1)

pagenation('.....')
print("Optimization Finished!")
print("Running count : " '%04d' % (epoch+1), "Training cost =",
r_cost, "W =", r_W, "b =", r_b)
```


2. Linear Regression

시각화 - 10번 학습 (3/4)

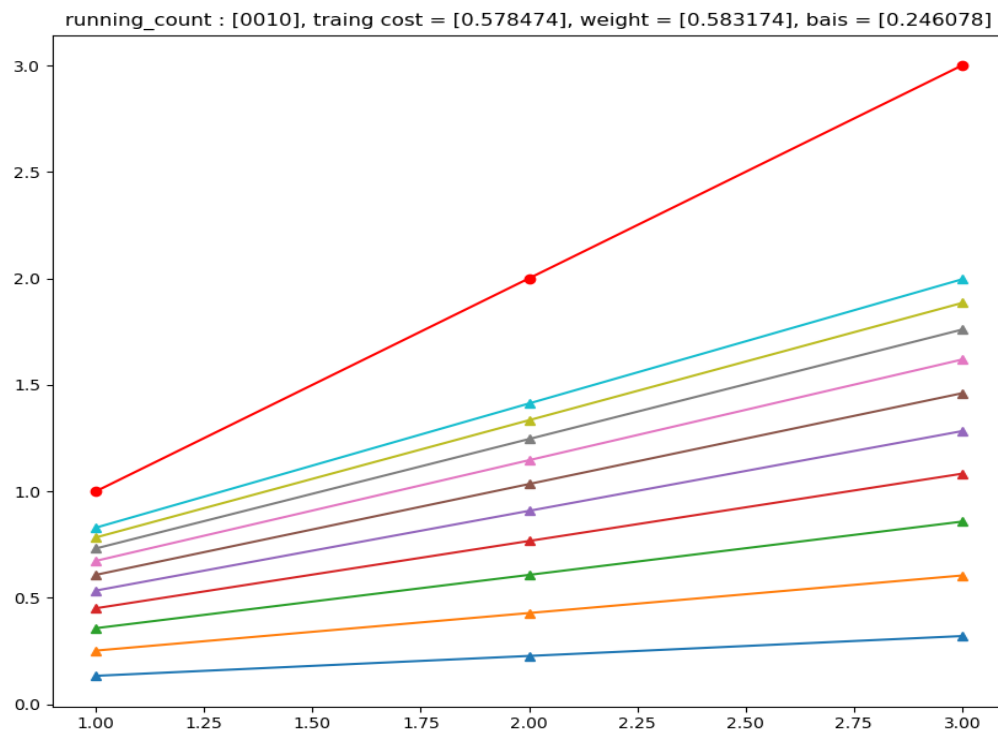


r2:14335 y=0.358571

2. Linear Regression

시각화 - 10번 학습 (4/4)

Figure 2



2. Linear Regression

시각화 - 100번 학습 (1/4)

```
# Graphic display
plt.figure(figsize=(10,8))
plt.plot(train_X, train_Y, label='y_data', marker="o",
color='r',linewidth=1.5)

pagenation('.....')

# 학습( training_cnt 횟수만큼)
for epoch in range(training_cnt):      # training_cnt = 100
    r_cost, r_W, r_b, _ = sess.run([cost, W, b, op_train],
    feed_dict = {X: train_X, Y: train_Y})

    if (epoch+1) % display_step == 0: # 나머지가 0이될 때 결과 출
력, display_step= 10 (10번 마다 출력)
        print("Running count : " '%04d' % (epoch+1), "Training
cost =", r_cost, "W =", r_W, "b =", r_b)
```

2. Linear Regression

시각화 - 100번 학습 (2/4)

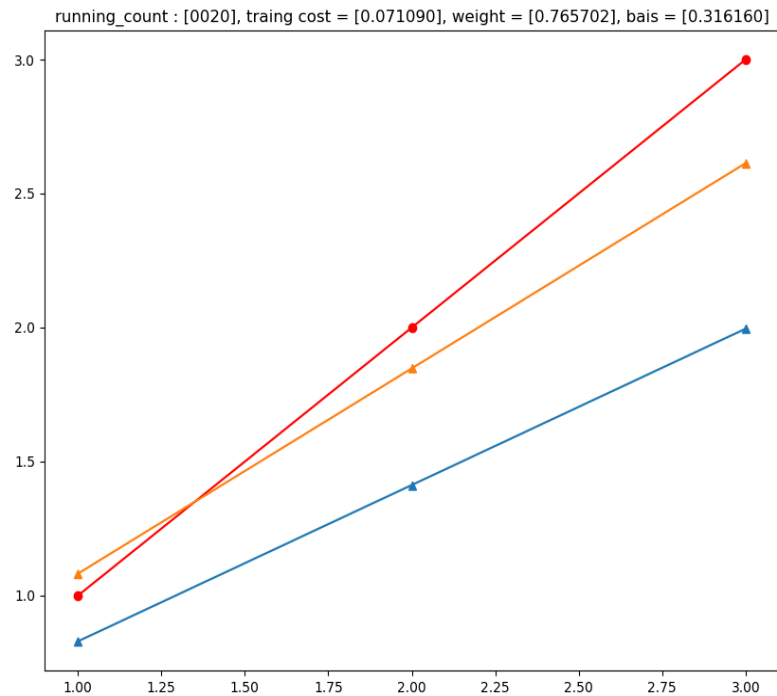
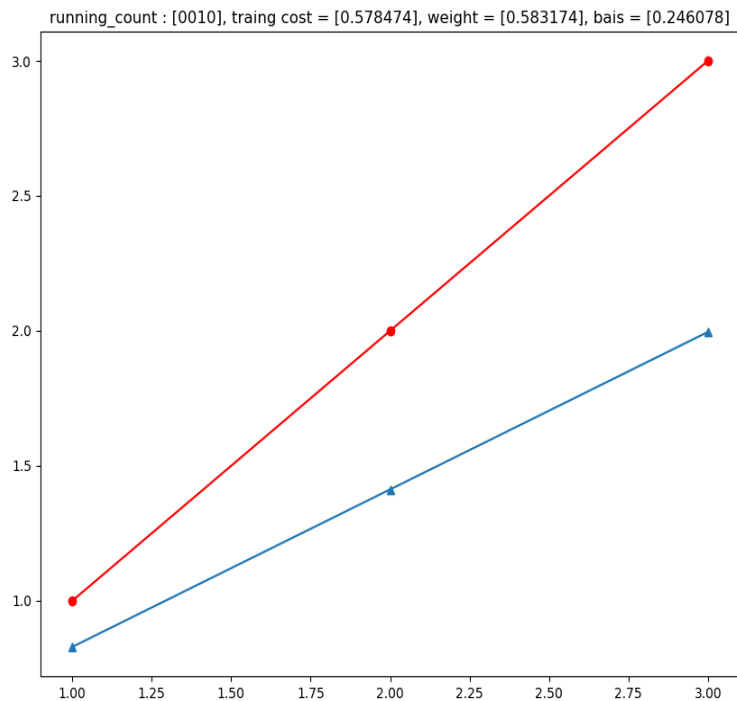
```
# Graphic display
plt.plot(train_X, sess.run(W) * train_X + sess.run(b),
marker = '^')
plt.title('running_count : [%04d], traing cost = [%.6f],
weight = [%.6f], bais = [%.6f]' % (epoch+1, r_cost, r_W, r_b))
plt.pause(0.1)

pagenation('.....')
print("Optimization Finished!")
print("Running count : " '%04d' % (epoch+1), "Training cost =",
r_cost, "W =", r_W, "b =", r_b)
```

2. Linear Regression

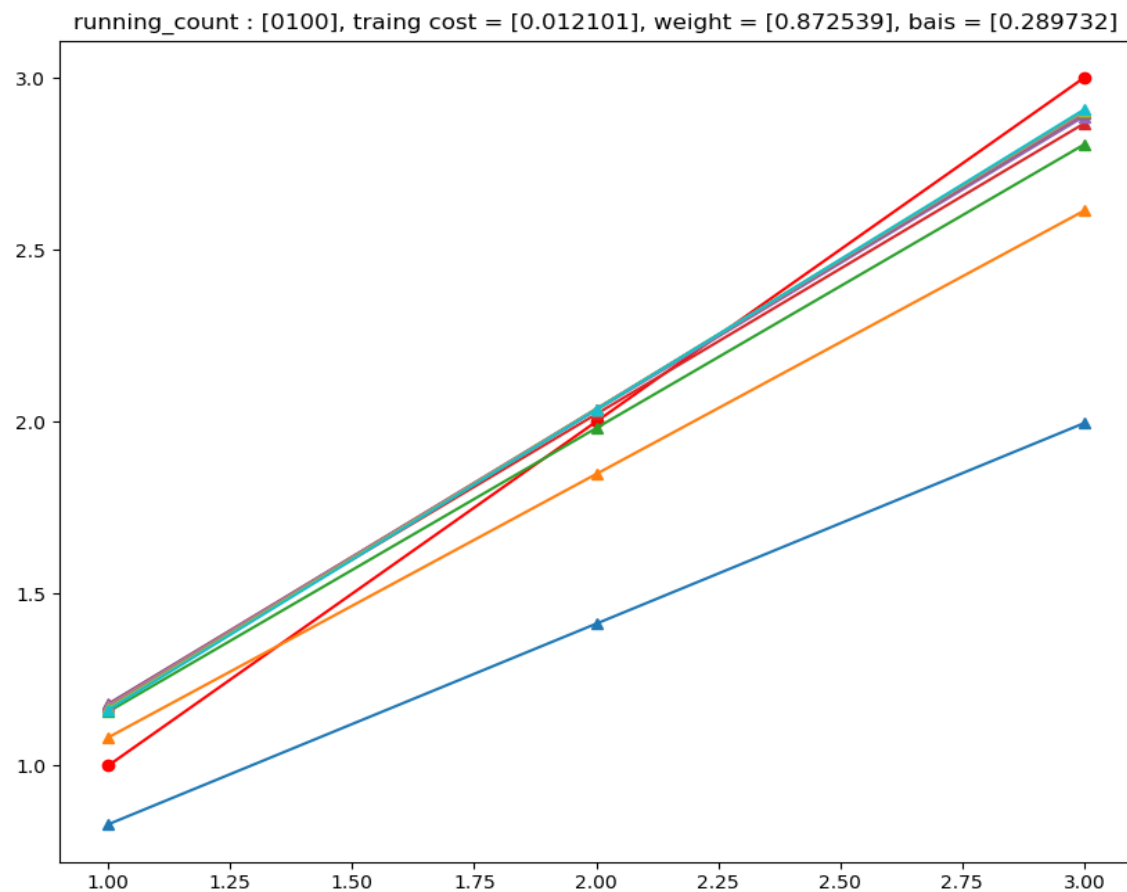


시각화 - 100번 학습 (3/4)



2. Linear Regression

시각화 - 100번 학습 (4/4)

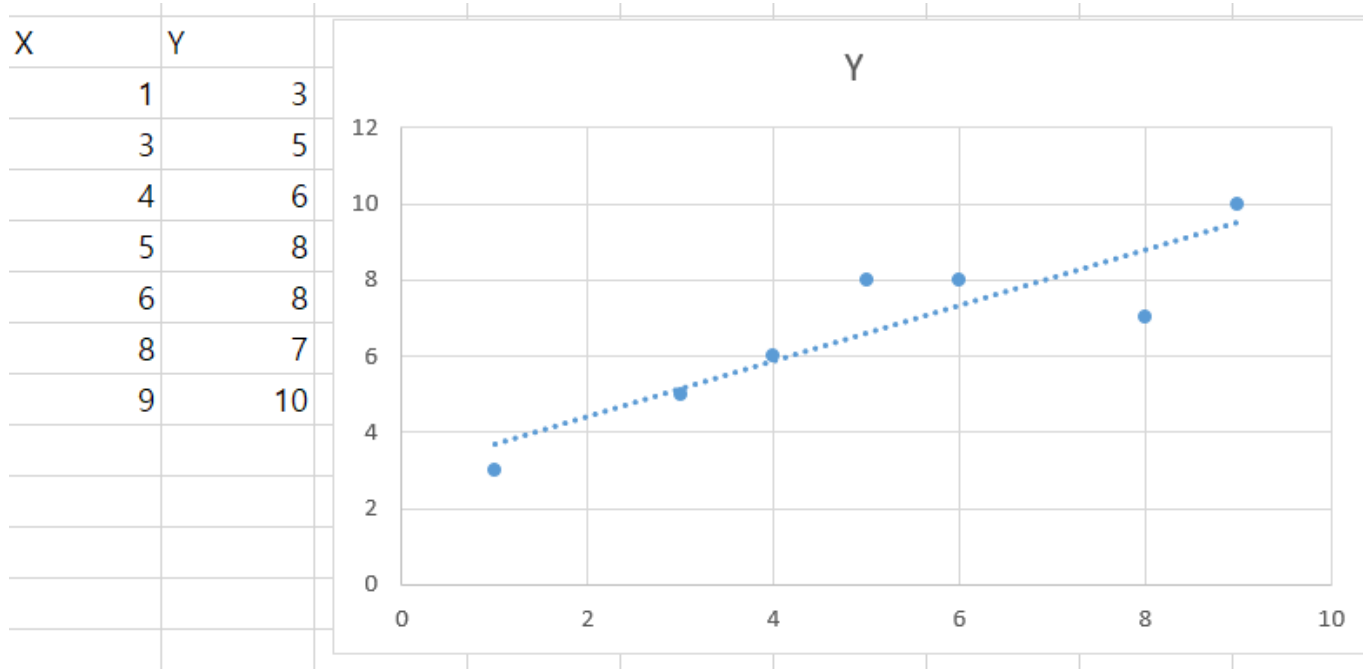


3. Multi-Variable Linear Regression

3. Multi-Variable Linear Regression

Regression이란? (1/2)

- 그래프 상에 무작위로 흩어져있는 데이터의 경향을 찾아 선으로 나타내는 것, 선이 직선 형태일 경우 Linear Regression이라 하며 Prediction(예측)의 한 방법이다.

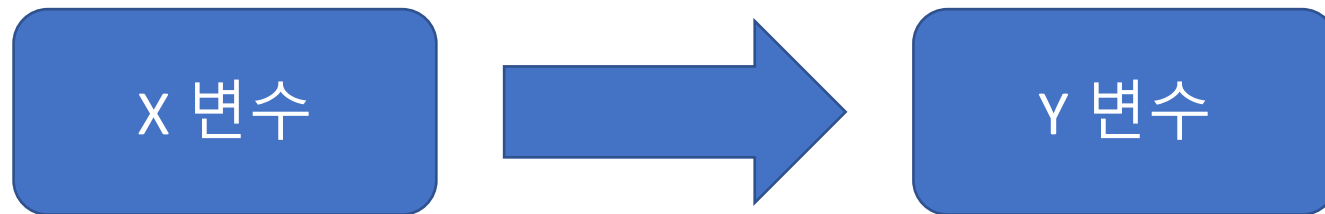


예) X: 시간(s), Y: 평균 속도(m/s)

3. Multi-Variable Linear Regression

Regression이란? (2/2)

- 데이터의 경향을 찾는다는 것은 새로운 데이터가 들어왔을 때 해당하는 값을 예측하는 것을 의미한다.
- 데이터의 경향을 찾기 위해서 퍼셉트론 모델을 사용한다.
- 여기서 X변수는 Y변수의 변화를 예측할 수 있다고 여겨지는 변수로 특징(Feature)이라 한다.
- Y변수는 결과로 간주되는 변수로 반응(Response)이라 한다.

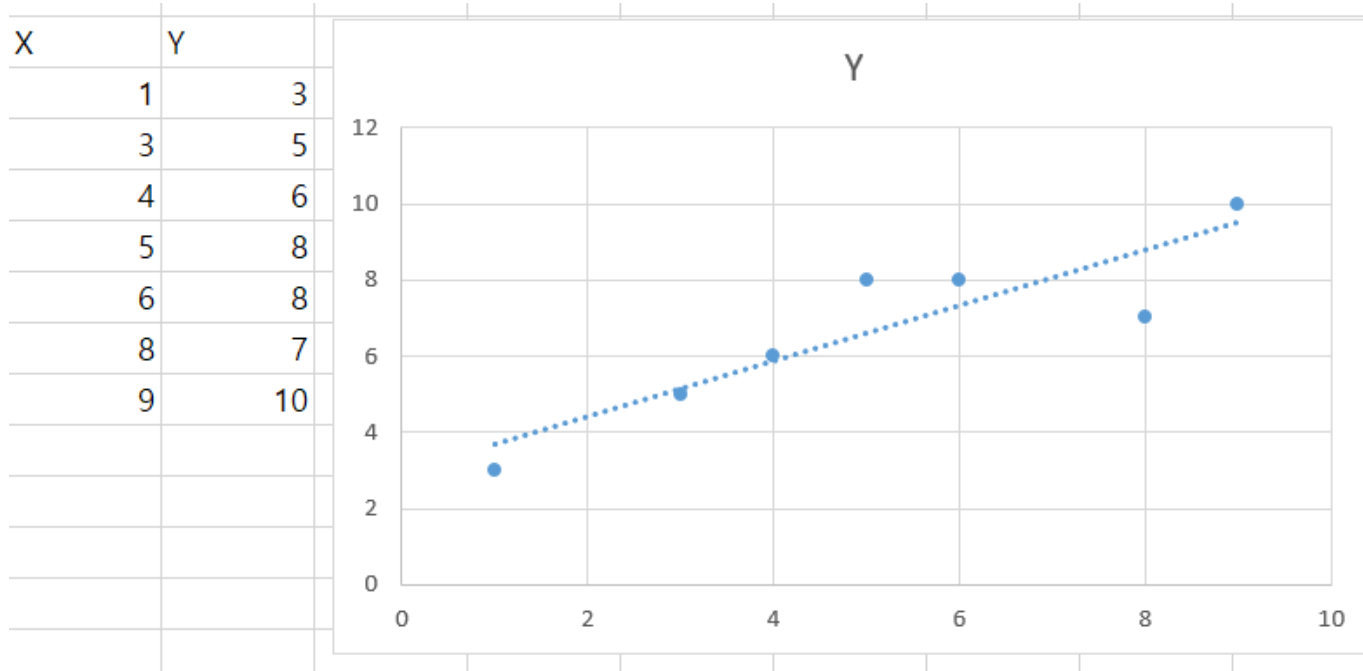


3. Multi-Variable Linear Regression



Multi-Variable이란? (1/2)

- 이전까지 배웠던 Linear Regression은 X변수가 1개, Y변수가 1개인 Simple Linear Regression이다.



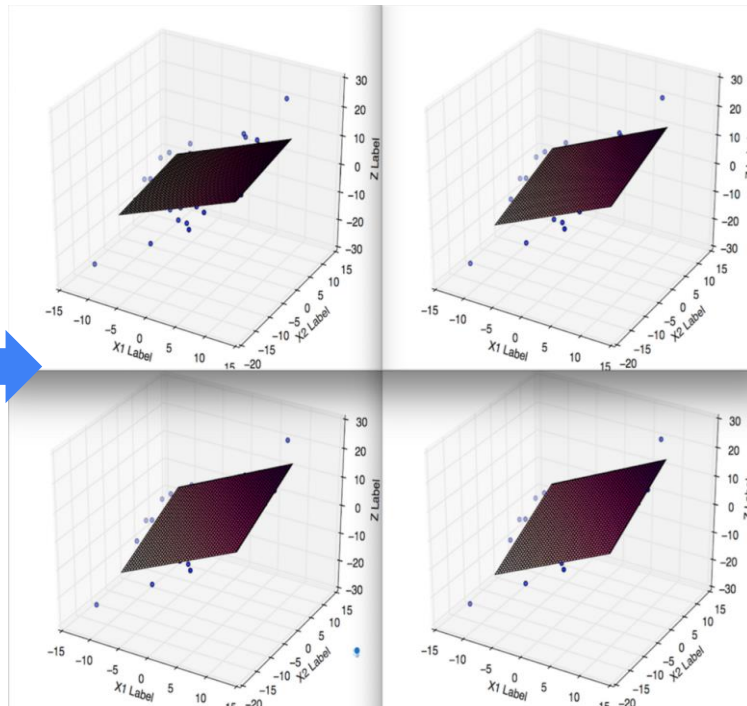
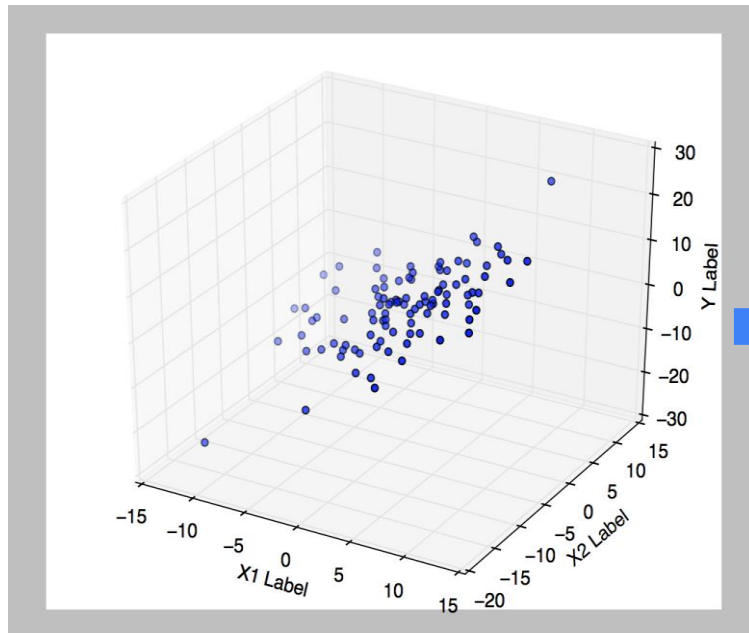
예) X: 시간(s), Y: 평균 속도(m/s)

3. Multi-Variable Linear Regression



Multi-Variable이란? (2/2)

- 지금부터는 X변수가 2개 이상, Y변수가 1개인 Multi-Variable Linear Regression을 다룰 것이다.
- 그래프 상에 무작위로 흩어져있는 데이터의 경향을 평면으로 나타낸다.



3. Multi-Variable Linear Regression

Multi-Variable Linear Regression 설계 요소

1. 모델(출력 계산 방법)

- : 주어진 데이터를 예측하기 위해 설계하는 모델로 퍼셉트론 모델을 사용하여 가중치(Weight)와 편향(bias)을 학습한다.
- 가중치(Weight)는 뉴런(X)이 결과(Y)에 주는 영향력(중요도)을 조절하는 매개 변수이다.
- 편향(bias)은 뉴런이 얼마나 쉽게 활성화하는지를 조정한다.

2. 비용 함수(Cost Function)

- : 예측값과 실제값의 차이인 오차(Cost)를 어떻게 계산할 것인지 설정한다.

3. 최적화(Optimization)

- : Cost(오차)를 최소화하기 위해 경사하강법(Gradient Descent) 알고리즘을 사용한다.

3. Multi-Variable Linear Regression

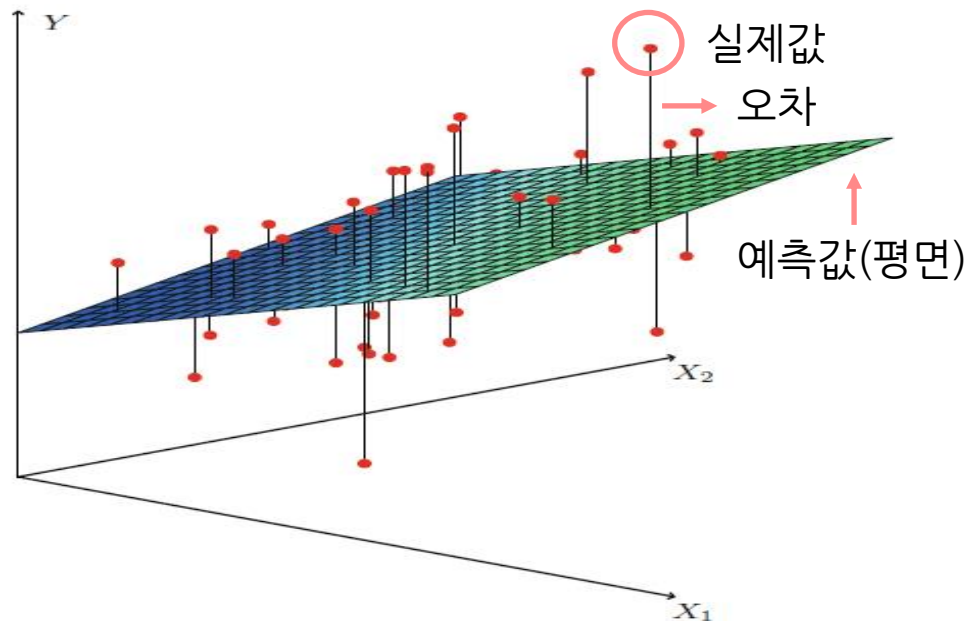
모델(출력 계산 방법) 정의하기

- 주어진 데이터를 가장 잘 나타내는 평면을 구하기 위해
가중치(Weight)와 편향(bias)을 사용하여 모델을 정의한다.
- X변수 즉, 특징(Feature)이 n 개인 모델은 다음과 같다.
- $H(x_1, x_2, \dots, x_n) = W_1x_1 + W_2x_2 + \dots + W_nx_n + b$
- $H(x)$: 예측값, W : 가중치, b : 편향
- 위의 모델로 구한 값이 예측값이 된다.

3. Multi-Variable Linear Regression

비용 함수(Cost Function) 정의

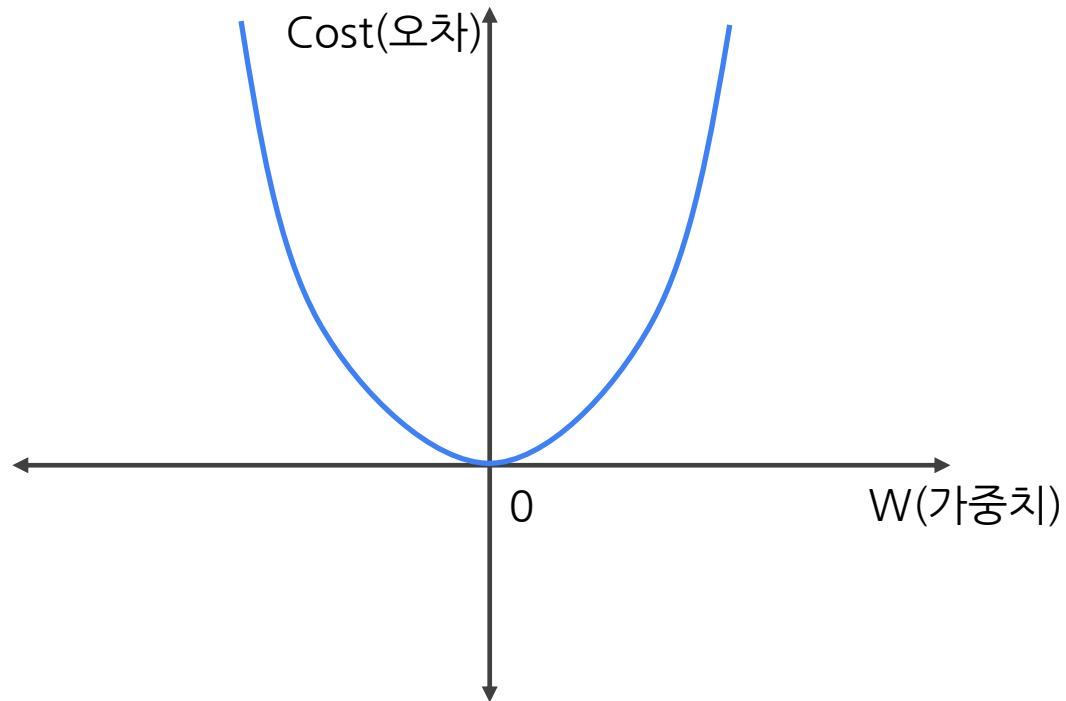
- 모델로 구한 예측값과 실제값(Y)의 차이인 오차를 어떻게 계산할 것인지 설정한다.
- 비용 함수는 모델로 구한 예측값과 실제값(Y)의 차이를 제공하여 평균한다.
- $cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}) - y^{(i)})^2$



3. Multi-Variable Linear Regression

최적화(Optimization) 정의 (1/3)

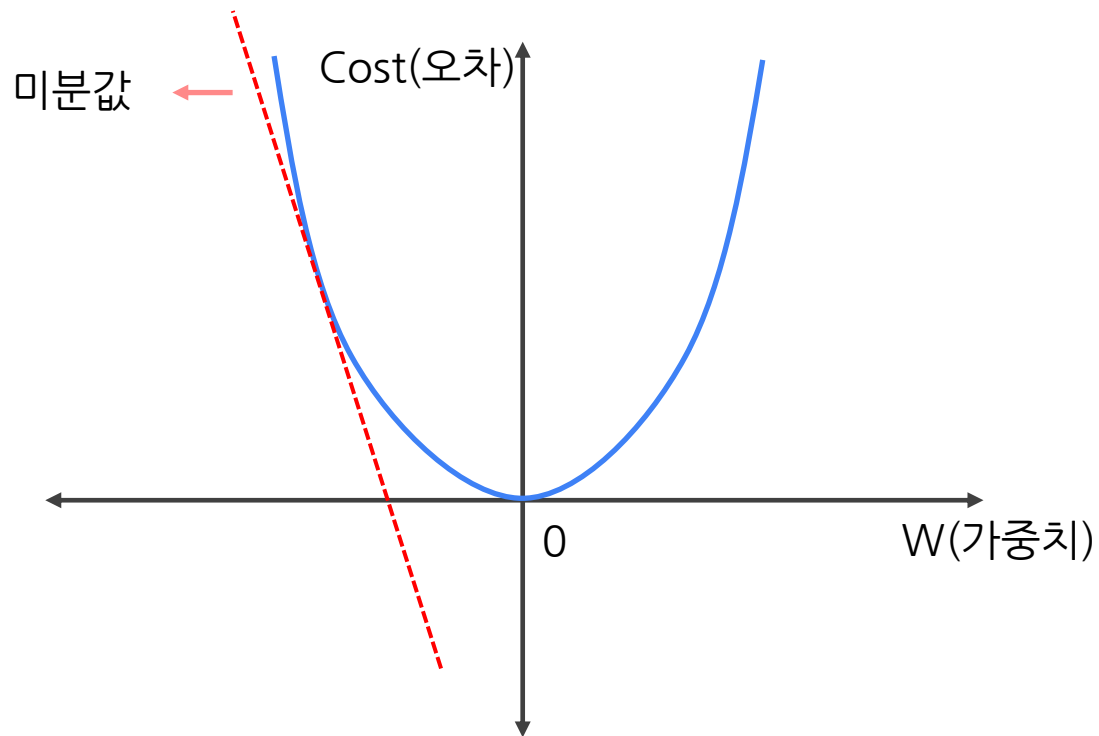
- 최적화한다는 것은 Cost(오차)를 최소화하는 것이며 이를 위해 W (가중치)와 b (편향)을 업데이트한다.
- W (가중치)에 따른 Cost(오차) 그래프를 나타내면 아래와 같다.



3. Multi-Variable Linear Regression

최적화(Optimization) 설계하기 (2/3)

- Cost(오차)를 최소화하는 방법으로 경사하강법(Gradient Descent) 알고리즘을 사용한다.

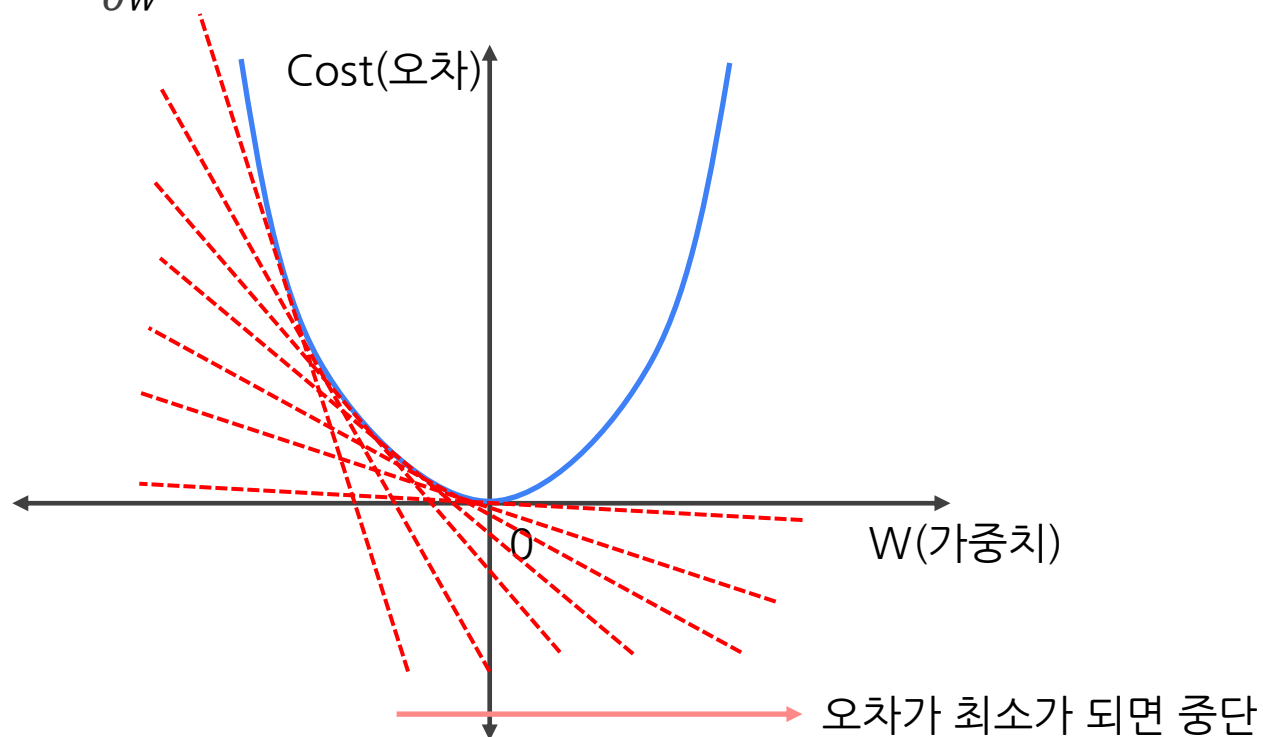


3. Multi-Variable Linear Regression

🔍 최적화(Optimization) 설계하기 (3/3)

- 경사하강법(Gradient Descent) 알고리즘의 원리는 W (가중치)에 대한 미분값을 통해 W 값을 업데이트하여 오차의 최저점을 찾는 것이다.

$$- W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$



3. Multi-Variable Linear Regression

Tensorflow를 이용한 Linear Regression 설계하기

1. 그래프를 만든다. (Build graph)
2. session을 열고 sess.run()으로 그래프를 실행시킨다. (Run)
3. 반환되는 결과 값을 가지고 반복해서 변수를 업데이트하여 학습시킨다.
(Update)

3. Multi-Variable Linear Regression

전체 코드 (1/3)

```
import tensorflow as tf
import numpy as np

learning_rate = 0.01
training_cnt = 10
display_step = 1

train_X1 = np.array([1., 0., 3., 0., 5.])
train_X2 = np.array([0., 2., 0., 4., 0.])

train_Y = np.array([1., 2., 3., 4., 5.])

X1 = tf.placeholder("float")
X2 = tf.placeholder("float")
Y = tf.placeholder("float")
```

3. Multi-Variable Linear Regression

전체 코드 (2/3)

```
W1 = tf.Variable([.0], tf.float32, name = "weight1")
W2 = tf.Variable([.0], tf.float32, name = "weight2")
b = tf.Variable([.0], tf.float32, name = "bias")

pred = X1 * W1 + X2 * W2 + b

cost = tf.reduce_mean(tf.pow(pred-Y, 2))
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
```

3. Multi-Variable Linear Regression

전체 코드 (3/3)

```
for epoch in range(training_cnt):
    r_cost, r_W1, r_W2, r_b, r_pred, _ = sess.run([cost, W1, W2,
    b, pred, op_train],
    feed_dict = {X1: train_X1, X2: train_X2, Y: train_Y})

    if (epoch+1) % display_step == 0:
        print("Run_count : [%04d], Train_cost =[%.4f], W1
        =[%.4f], W2 =[%.4f], b =[%.4f],
        pred =[%.4f %.4f %.4f %.4f %.4f]"
              % (epoch+1, r_cost, r_W1, r_W2,
        r_b, r_pred[0], r_pred[1], r_pred[2], r_pred[3], r_pred[4] ))

    print("Optimization Finished!")
    print("Run_count : [%04d], Train_cost =[%.4f], W1 =[%.4f], W2
    =[%.4f], b =[%.4f], pred =[%.4f %.4f %.4f %.4f %.4f]"
          % (epoch+1, r_cost, r_W1, r_W2,
    r_b, r_pred[0], r_pred[1], r_pred[2], r_pred[3], r_pred[4] ))
```

3. Multi-Variable Linear Regression

모델 구축(Build graph) (1/6)

파라미터 값 설정

- 학습을 위한 기초 파라미터
- learning_rate : 값이 너무 적으면 Train 되지 않을 수 있고 값이 너무 크면 overshooting이 발생할 수 있다.
- training_cnt : data set에 대한 training 반복 횟수

```
# 파라미터값 설정
learning_rate = 0.01
training_cnt = 10
display_step = 1 # 원하는 출력 위치 조정
```

3. Multi-Variable Linear Regression

모델 구축(Build graph) (2/6)

트레이닝 데이터 변수 선언

- 입력으로 들어가는 x data(input 2개), y data(output 1개) 설정
- numpy array를 사용

```
# Training Data
```

```
train_X1 = np.array([1., 0., 3., 0., 5.])
```

```
train_X2 = np.array([0., 2., 0., 4., 0.])
```

```
train_Y = np.array([1., 2., 3., 4., 5.])
```

3. Multi-Variable Linear Regression

모델 구축(Build graph) (3/6)

tf graph input

- placeholder : 학습용 데이터를 담는 그릇
- 값은 미정이나 어떤 타입, 데이터 구조를 갖는 데이터를 정의함, 즉 입력값을 받을 수 있음
- 아래는 어떤 데이터인지는 모르겠지만, float 타입 데이터를 정의 하겠다라는 것
- * **tf.placeholder() << 괄호 사이에 shift+tab을 누르면 사용 가능한 변수들이 보임**
- dtype : 저장되는 데이터 형 (실수, 정수 등 정의)
- shape : 행렬의 차원 정의, shape=[3,3]으로 정의하면, 이 플레이스홀더는 3x3행렬을 저장
- name : 플레이스홀더의 이름 정의

```
X1 = tf.placeholder("float")  
X2 = tf.placeholder("float")  
Y = tf.placeholder("float")
```

입력 변수를 담는 placeholder 2개로 늘어남

3. Multi-Variable Linear Regression

모델 구축(Build graph) (4/6)

set model weight

- variable : 변수를 의미하며 임의의 값을 지정
- 그래프를 계산하면서 최적화 할 변수들(신경망을 좌우하는 값) 입력 시 사용
- 입력 변수가 2개이기 때문에 각각에 weight가 주어지게 된다.

```
# Set model weights
```

```
W1 = tf.Variable([.0], tf.float32, name = "weight1")
```

```
W2 = tf.Variable([.0], tf.float32, name = "weight2")
```

```
b = tf.Variable([.0], tf.float32, name = "bias")
```

  
초기값, data type, 이름 정의

3. Multi-Variable Linear Regression

🔍 모델 구축(Build graph) (5/6)

☑ linear model (예측 값 구현)

$$H(x_1, x_2) = x_1 w_1 + x_2 w_2 + b$$

x_1	x_2	Y
1	0	1
0	2	2
3	0	3
0	4	4
5	0	5

$$1 * 1 + 0 * 1 + 0 = 1$$

$$0 * 1 + 2 * 1 + 0 = 2$$

$$3 * 1 + 0 * 1 + 0 = 3$$

$$0 * 1 + 4 * 1 + 0 = 4$$

$$5 * 1 + 0 * 1 + 0 = 5$$

```
train_X1 = np.array([1., 0., 3., 0., 5.])
train_X2 = np.array([0., 2., 0., 4., 0.])
```

```
train_Y = np.array([1., 2., 3., 4., 5.])
```

```
X1 = tf.placeholder("float")
```

```
X2 = tf.placeholder("float")
```

```
Y = tf.placeholder("float")
```

```
W1 = tf.Variable([.0], tf.float32,
name = "weight1")
```

```
W2 = tf.Variable([.0], tf.float32,
name = "weight2")
```

```
b = tf.Variable([.0], tf.float32,
name = "bias")
```

```
pred = X1 * W1 + X2 * W2 + b
```

3. Multi-Variable Linear Regression

🔍 모델 구축(Build graph) (6/6)

✓ cost/loss function 구현

$$\text{cost}(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x_1^{(i)}, x_2^{(i)}) - y^{(i)})^2$$

cost/loss function

```
cost = tf.reduce_mean(tf.pow(pred-Y, 2))
```

or

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

✓ 학습 방법 → cost를 최소화

- GradientDescent 함수 사용 (경사하강법)

```
cost = tf.reduce_mean(tf.pow(pred-Y, 2))
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
```

```
op_train = optimizer.minimize(cost)
```

3. Multi-Variable Linear Regression

모델 실행(run/update) (1/3)

세션을 열고 그래프 실행

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

변수들에 초기값 할당

```
init = tf.global_variables_initializer()
```

```
sess.run(init)
```

for epoch **in** range(training_cnt): *# training_cnt 횟수만큼
반복 실행*

```
    r_cost, r_W1, r_W2, r_b, r_pred, _ = sess.run([cost,  
    W1, W2, b, pred, op_train],
```

```
    feed_dict = {X1: train_X1, X2: train_X2, Y: train_Y})
```

sess.run을 통해 학습한(변경된) 파라미터 값들을 할당

3. Multi-Variable Linear Regression

모델 실행(run/update) (2/3)

```
if (epoch+1) % display_step == 0:
    print("Run_count : [%04d], Train_cost =[%.4f], W1
    =[%.4f], W2 =[%.4f], b =[%.4f],
    pred =[%.4f %.4f %.4f %.4f %.4f]"
          % (epoch+1, r_cost, r_W1, r_W2,
    r_b,r_pred[0],r_pred[1],r_pred[2],r_pred[3],r_pred[4] ))
    # sess.run을 통해 학습된 값 출력(pred의 경우 value가 5개가 나오기 때문에 그 값을
    # 소수점4번째까지만 나타내려 각각 print함 )

print("Optimization Finished!")
print("Run_count : [%04d], Train_cost =[%.4f], W1 =[%.4f], W2
    =[%.4f], b =[%.4f], pred =[%.4f %.4f %.4f %.4f %.4f]"
      % (epoch+1, r_cost, r_W1, r_W2,
    r_b,r_pred[0],r_pred[1],r_pred[2],r_pred[3],r_pred[4] ))
```

3. Multi-Variable Linear Regression

모델 실행(run/update) (3/3)

```
Run_count : [0001], Train_cost =[11.0000], W1 =[0.1400], W2 =[0.0800], b
=[0.0600], pred =[0.0000 0.0000 0.0000 0.0000 0.0000]
Run_count : [0002], Train_cost =[8.2482], W1 =[0.2582], W2 =[0.1522], b
=[0.1118], pred =[0.2000 0.2200 0.4800 0.3800 0.7600]
Run_count : [0003], Train_cost =[6.2131], W1 =[0.3581], W2 =[0.2173], b
=[0.1567], pred =[0.3701 0.4162 0.8866 0.7205 1.4030]
Run_count : [0004], Train_cost =[4.7033], W1 =[0.4423], W2 =[0.2762], b
=[0.1954], pred =[0.5147 0.5913 1.2308 1.0259 1.9470]
Run_count : [0005], Train_cost =[3.5794], W1 =[0.5133], W2 =[0.3294], b
=[0.2290], pred =[0.6377 0.7477 1.5223 1.3001 2.4069]
Run_count : [0006], Train_cost =[2.7396], W1 =[0.5732], W2 =[0.3775], b
=[0.2580], pred =[0.7423 0.8877 1.7690 1.5465 2.7956]
Run_count : [0007], Train_cost =[2.1096], W1 =[0.6237], W2 =[0.4211], b
=[0.2831], pred =[0.8312 1.0131 1.9777 1.7681 3.1241]
Run_count : [0008], Train_cost =[1.6348], W1 =[0.6662], W2 =[0.4607], b
=[0.3049], pred =[0.9068 1.1254 2.1542 1.9677 3.4016]
Run_count : [0009], Train_cost =[1.2755], W1 =[0.7019], W2 =[0.4965], b
=[0.3238], pred =[0.9711 1.2262 2.3034 2.1475 3.6358]
Run_count : [0010], Train_cost =[1.0022], W1 =[0.7320], W2 =[0.5290], b
=[0.3401], pred =[1.0257 1.3167 2.4296 2.3097 3.8335]
Optimization Finished!
Run_count : [0010], Train_cost =[1.0022], W1 =[0.7320], W2 =[0.5290], b
=[0.3401], pred =[1.0257 1.3167 2.4296 2.3097 3.8335]
```

3. Multi-Variable Linear Regression

def(함수) 사용하여 코딩하기 (1/7)

```
def make_graph() :  
    global X, Y, W, b, pred ,cost,optimizer, op_train  
    # 전역변수 선언  
  
    X1 = tf.placeholder("float")  
    X2= tf.placeholder("float")  
    Y = tf.placeholder("float")  
  
    W1 = tf.Variable([.0], tf.float32, name = "weight1")  
    W2 = tf.Variable([.0], tf.float32, name = "weight2")  
    b = tf.Variable([.0], tf.float32, name = "bias")  
  
    pred = X1 * W1 + X2 * W2 + b  
  
    cost = tf.reduce_mean(tf.pow(pred-Y, 2))  
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)  
    op_train = optimizer.minimize(cost)
```

3. Multi-Variable Linear Regression

def(함수) 사용하여 코딩하기 (2/7)

```
def run_graph() :  
  
    sess = tf.Session()  
    init = tf.global_variables_initializer()  
    sess.run(init)  
  
    for epoch in range(training_cnt):  
        r_cost, r_W1, r_W2, r_b, r_pred, _ = sess.run([cost, W1,  
W2, b, pred, op_train],  
        feed_dict = {X1: train_X1, X2: train_X2, Y: train_Y})
```


3. Multi-Variable Linear Regression

def(함수) 사용하여 코딩하기 (3/7)

```
    if (epoch+1) % display_step == 0:
        print("Run_count : [%04d], Train_cost =[%.4f], W1
=[%.4f], W2 =[%.4f], b =[%.4f], pred
=[%.4f %.4f %.4f %.4f %.4f]"
              % (epoch+1, r_cost, r_W1, r_W2,
r_b,r_pred[0],r_pred[1],r_pred[2],r_pred[3],r_pred[4] ))

        print("Optimization Finished!")
        print("Run_count : [%04d], Train_cost =[%.4f], W1 =[%.4f],
W2 =[%.4f], b =[%.4f], pred =[%.4f %.4f %.4f %.4f %.4f]"
              % (epoch+1, r_cost, r_W1, r_W2,
r_b,r_pred[0],r_pred[1],r_pred[2],r_pred[3],r_pred[4] ))
```

3. Multi-Variable Linear Regression

def(함수) 사용하여 코딩하기 (4/7)

- 함수 실행하기

```
make_graph()
```

```
run_graph()
```

3. Multi-Variable Linear Regression

def(함수) 사용하여 코딩하기 (5/7)

- 함수 실행하기

```
Run_count : [0001], Train_cost =[11.0000], W1 =[0.1400], W2 =[0.0800], b =[0.0600],  
pred =[0.0000 0.0000 0.0000 0.0000 0.0000]  
Run_count : [0002], Train_cost =[8.2482], W1 =[0.2582], W2 =[0.1522], b =[0.1118],  
pred =[0.2000 0.2200 0.4800 0.3800 0.7600]  
Run_count : [0003], Train_cost =[6.2131], W1 =[0.3581], W2 =[0.2173], b =[0.1567],  
pred =[0.3701 0.4162 0.8866 0.7205 1.4030]  
Run_count : [0004], Train_cost =[4.7033], W1 =[0.4423], W2 =[0.2762], b =[0.1954],  
pred =[0.5147 0.5913 1.2308 1.0259 1.9470]  
Run_count : [0005], Train_cost =[3.5794], W1 =[0.5133], W2 =[0.3294], b =[0.2290],  
pred =[0.6377 0.7477 1.5223 1.3001 2.4069]  
Run_count : [0006], Train_cost =[2.7396], W1 =[0.5732], W2 =[0.3775], b =[0.2580],  
pred =[0.7423 0.8877 1.7690 1.5465 2.7956]  
Run_count : [0007], Train_cost =[2.1096], W1 =[0.6237], W2 =[0.4211], b =[0.2831],  
pred =[0.8312 1.0131 1.9777 1.7681 3.1241]  
Run_count : [0008], Train_cost =[1.6348], W1 =[0.6662], W2 =[0.4607], b =[0.3049],  
pred =[0.9068 1.1254 2.1542 1.9677 3.4016]  
Run_count : [0009], Train_cost =[1.2755], W1 =[0.7019], W2 =[0.4965], b =[0.3238],  
pred =[0.9711 1.2262 2.3034 2.1475 3.6358]  
Run_count : [0010], Train_cost =[1.0022], W1 =[0.7320], W2 =[0.5290], b =[0.3401],  
pred =[1.0257 1.3167 2.4296 2.3097 3.8335]  
Optimization Finished!  
Run_count : [0010], Train_cost =[1.0022], W1 =[0.7320], W2 =[0.5290], b =[0.3401],  
pred =[1.0257 1.3167 2.4296 2.3097 3.8335]
```

3. Multi-Variable Linear Regression

def(함수) 사용하여 코딩하기 (6/7)

- 학습 횟수 -> 100

```
learning_rate = 0.01  
training_cnt = 100  
display_step = 10
```

3. Multi-Variable Linear Regression

def(함수) 사용하여 코딩하기 (7/7)

- 학습 횟수 -> 100

```
Run_count : [0010], Train_cost =[1.0022], W1 =[0.7320], W2 =[0.5290], b =[0.3401],  
pred =[1.0257 1.3167 2.4296 2.3097 3.8335]  
Run_count : [0020], Train_cost =[0.1409], W1 =[0.8616], W2 =[0.7291], b =[0.4198],  
pred =[1.2726 1.8489 2.9856 3.2816 4.6986]  
Run_count : [0030], Train_cost =[0.0509], W1 =[0.8831], W2 =[0.8094], b =[0.4342],  
pred =[1.3163 2.0422 3.0809 3.6504 4.8454]  
Run_count : [0040], Train_cost =[0.0364], W1 =[0.8875], W2 =[0.8437], b =[0.4303],  
pred =[1.3183 2.1138 3.0927 3.7966 4.8671]  
Run_count : [0050], Train_cost =[0.0324], W1 =[0.8900], W2 =[0.8599], b =[0.4207],  
pred =[1.3115 2.1392 3.0910 3.8567 4.8704]  
Run_count : [0060], Train_cost =[0.0302], W1 =[0.8927], W2 =[0.8687], b =[0.4093],  
pred =[1.3029 2.1466 3.0877 3.8826 4.8726]  
Run_count : [0070], Train_cost =[0.0284], W1 =[0.8956], W2 =[0.8746], b =[0.3975],  
pred =[1.2940 2.1468 3.0847 3.8949 4.8754]  
Run_count : [0080], Train_cost =[0.0267], W1 =[0.8987], W2 =[0.8791], b =[0.3858],  
pred =[1.2853 2.1443 3.0820 3.9016 4.8787]  
Run_count : [0090], Train_cost =[0.0251], W1 =[0.9016], W2 =[0.8830], b =[0.3742],  
pred =[1.2767 2.1407 3.0794 3.9060 4.8821]  
Run_count : [0100], Train_cost =[0.0236], W1 =[0.9046], W2 =[0.8867], b =[0.3630],  
pred =[1.2684 2.1368 3.0770 3.9094 4.8856]  
Optimization Finished!  
Run_count : [0100], Train_cost =[0.0236], W1 =[0.9046], W2 =[0.8867], b =[0.3630],  
pred =[1.2684 2.1368 3.0770 3.9094 4.8856]
```

3. Multi-Variable Linear Regression

Matrix를 사용한 구현

```
learning_rate = 0.01
training_cnt = 10
display_step = 1

train_MX = np.array([[1., 0.], [0., 2.], [3., 0.], [0., 4.], [5., 0.] ])
train_Y = np.array([[1.], [2.], [3.], [4.], [5.]])

MX = tf.placeholder("float", shape=[None, 2])
Y = tf.placeholder("float", shape=[None, 1])

W = tf.Variable(tf.zeros([2, 1]), name='weight')
b = tf.Variable(tf.zeros([1]), name='bias')

pred = tf.matmul(MX, W) + b

cost = tf.reduce_mean(tf.pow(pred - Y, 2))
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)
```

3. Multi-Variable Linear Regression

🔍 모델 구축(Build graph) - Matrix 사용

$$H(x_1, x_2) = w_1x_1 + w_2x_2 + b \quad (x_1 \ x_2) \cdot \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = (x_1w_1 + x_2w_2)$$

x_1	x_2	Y
1	0	1
0	2	2
3	0	3
0	4	4
5	0	5

$$1 \times 1 + 0 \times 1 + 0 = 1$$

$$0 \times 1 + 2 \times 1 + 0 = 2$$

$$3 \times 1 + 0 \times 1 + 0 = 3$$

$$0 \times 1 + 4 \times 1 + 0 = 4$$

$$5 \times 1 + 0 \times 1 + 0 = 5$$

```
train_MX = np.array([[1., 0.], [0., 2.],
[3., 0.], [0., 4.], [5., 0.] ])
train_Y = np.array([[1.], [2.], [3.],
[4.], [5.]])
```

```
MX = tf.placeholder("float", shape=[None,
2])
Y = tf.placeholder("float", shape=[None,
1])
```

```
W = tf.Variable(tf.zeros([2, 1]),
name='weight')
b = tf.Variable(tf.zeros([1]),
name='bias')
```

```
pred = tf.matmul(MX, W) + b
```

3. Multi-Variable Linear Regression



연습문제 - Input 3개

- 다음과 같은 점수를 가지고 Multi-Variable Linear Regression을 Tensorflow로 구현해 보자

X_1	X_2	X_3	Y
83	67	75	144
63	88	96	166
98	71	68	172
77	99	67	154
54	76	33	112

감사합니다.

참고자료(Reference)

<https://medium.com/qandastudy/mathpresso-%EB%A8%B8%EC%8B%A0-%EB%9F%AC%EB%8B%9D-%EC%8A%A4%ED%84%B0%EB%94%94-4-%ED%9A%8C%EA%B7%80-%EB%B6%84%EC%84%9D-regression-1-6d6cc0aaa483> (다중선형회귀1)

https://godongyoung.github.io/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D/2018/01/20/ISL-linear-regression_ch3.html (다중선형회귀2)

<https://blog.naver.com/choiym24/221183453720> (퍼셉트론)

https://blog.naver.com/na_young_1124/221347064835 (활성화함수)

<https://blog.naver.com/wpdus2694/221230981760> (경사하강법)