ML lab 02
Simple Linear Regression LAB

2021년 3월 12일 금요일 오후 10:55

Lec-02 정리

W는 weight(가중치), b는 bias(편향). 우리가 정한 가설을 가설함수를 우리의 모델이라고 칭하기도 한다.

Hypothesis and cost function

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

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

Build hypothesis and cost

In [10]:
```
import tensorflow as tf

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

W = tf.Variable(2.9)
b = tf.Variable(0.5)

# hypothesis = W * x + b
hypothesis = W * x_data + b

# cost Function
cost = tf.reduce_mean(tf.square(hypothesis - y_data))
print(cost)


v = [1.,2.,3.,4.]
# 줄어들면서 mean을 구함
print(tf.reduce_mean(v))

# square: 제곱
print(tf.square(3))
```

```
tf.Tensor(45.660004, shape=(), dtype=float32)
tf.Tensor(2.5, shape=(), dtype=float32)
tf.Tensor(9, shape=(), dtype=int32)
```

Gradient descent(경사 하강 알고리즘, 경사하강법)

함수의 기울기(경사)를 구하고 경사의 절댓값이 낮은 쪽으로 계속 이동시켜서 극값에 이를 때까지 반복시키는 것을 경사하강법이라고 한다.

즉, cost(W,b) 를 minimize하게 하는 W와 b를 찾는 알고리즘이다.

실습 결과

In [12]:
```
import tensorflow as tf

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

W = tf.Variable(2.9)
b = tf.Variable(0.5)

# hypothesis = W * x + b
hypothesis = W * x_data + b

# cost Function
cost = tf.reduce_mean(tf.square(hypothesis - y_data))
print(cost)


v = [1.,2.,3.,4.]
# 줄어들면서 mean을 구함
print(tf.reduce_mean(v))

# square: 제곱
print(tf.square(3))

learning_rate = 0.01

# Gradient Descent
# 블럭 안 변수들의 변화 정보를 tape에 기록함

with tf.GradientTape() as tape:
    hypothesis = W * x_data + b
    cost = tf.reduce_mean(tf.square(hypothesis - y_data))

# 이후 tape의 gradient값. 즉, 미분값을 구한다.
# 변수들에 대한 개별 미분값. 즉, 기울기값을 튜플로 반환한다.
# cost function에 대해서 W, b에 대한 미분값(gradient)를 각각 반환한다.
W_grad, b_grad = tape.gradient(cost, [W, b])

# W와 b값 업데이트
# assign_sub : -= 과 같은 역할
# A.assign_sub(B) == ( A = A - b )
W.assign_sub(learning_rate * W_grad)
b.assign_sub(learning_rate * b_grad)
```

```
tf.Tensor(45.660004, shape=(), dtype=float32)
tf.Tensor(2.5, shape=(), dtype=float32)
tf.Tensor(9, shape=(), dtype=int32)
```

Out[12]:
```
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=0.376>
```

In [16]:
```
import tensorflow as tf

# Data
x_data = [1,2,3,4,5]
y_data = [1,2,3,4,5]

# W, b initalize
W = tf.Variable(2.9)
b = tf.Variable(0.5)

hypothesis = W * x_data + b
cost = tf.reduce_mean(tf.square(hypothesis - y_data))

# W, b update
for i in range(100):
    # Gradient Descent
    with tf.GradientTape() as tape:
        hypothesis = W * x_data + b
        cost = tf.reduce_mean(tf.square(hypothesis - y_data))
    W_grad, b_grad = tape.gradient(cost, [W, b])

    W.assign_sub(learning_rate * W_grad)
    b.assign_sub(learning_rate * b_grad)

    if i % 10 == 0:
        print("{:5}|{:10.4}|{:10.4}|{:10.6f}".format(i, W.numpy(), b.numpy(), cost))
```

```
    i|         W|         b|      cost
    0|     2.452|     0.376| 45.660004
   10|     1.104|  0.003398|  0.206336
   20|     1.013|  -0.02091|  0.001026
   30|     1.007|  -0.02184|  0.000093
   40|     1.006|  -0.02123|  0.000083
   50|     1.006|  -0.02053|  0.000077
   60|     1.005|  -0.01984|  0.000072
   70|     1.005|  -0.01918|  0.000067
   80|     1.005|  -0.01854|  0.000063
   90|     1.005|  -0.01793|  0.000059
```