11. Tensorflow 그리고 Regression

- Tensorflow
- 회귀(Regression)

Tensorflow

(1) Tensorflow 2.0 개요

Tensorflow

- **Tensorflow**
  - Google에서 개발한 기계 학습(machine learning) 엔진
  - 2015년에 공개 소스 소프트웨어로 전환
  - C++ 언어로 작성되었고, Python 응용 프로그래밍 인터페이스(API)를 제공

- **Tensorflow의 특징**
  - 빠르고 유연함
  - 프로그램은 구조화된 graph(edge와 node)로 구성됨(버전 1)
  - 버전 2는 eager-execution으로 python의 함수 실행과 유사하게 바뀌었으며, Keras API를 지원하여, 보다 쉽고 강력한 머신러닝 구현 가능

Tensorflow 1.0과 2.0

- Tensorflow 1.0은 graph를 구성하고 이를 session을 만들어 실행시키는 구조를 가지고 있음

```
import tensorflow as tf

a = tf.constant(1)

with tf.Session() as sess:
    print(sess.run(a))
```

```
import tensorflow.compat.v1 as tf

tf.disable_v2_behavior()
```

```
1
```

- Tensorflow 2.0은 eager execution이 도입되었으며, numpy 객체와 완벽하게 호환됨

```
import tensorflow as tf

a = tf.constant(1)
print(a.numpy())    # convert to numpy element
```

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

x = tf.constant(35)
y = tf.Variable(x + 5)
model = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(model)
    print("x=", sess.run(x))
    print("y=", sess.run(y))
```

```
x = 35
y = 40
```

[Tensorflow 2.0]

```
import tensorflow as tf

x = tf.constant(35)
y = tf.Variable(x + 5)
print(f"x = {x.numpy()}")
print(f"y = {y.numpy()}")
```

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

sum = tf.Variable(0)
model = tf.global_variables_initializer()

sess = tf.Session()
sess.run(model)
for i in range(5):
    sum = sum + 1
    print(sess.run(sum))
sess.close()
```

```
1
2
3
4
5
```

[Tensorflow 2.0]

```
import tensorflow as tf

sum = tf.Variable(0)
for i in range(5):
    sum = sum + 1
    print(sum.numpy())
```

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

input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3)
mul = tf.multiply(input1, intermed)

with tf.Session() as sess:
    print(sess.run([mul, intermed]))
```

```
[21.0, 7.0]
```

[Tensorflow 2.0]

```
import tensorflow as tf

input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3)
mul = tf.multiply(input1, intermed)

print([mul.numpy(), intermed.numpy()])
```

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

input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1, input2)

with tf.Session() as sess:
    print(sess.run(output,\
                   feed_dict={input1:7.0, input2:2.0}))
```

```
14.0
```

[Tensorflow 2.0]

```
import tensorflow as tf

def run(input1, input2):
    return tf.multiply(input1, input2)

print(run(7.0, 2.0).numpy())
```

Tensorflow의 로그 필터링

- TF_CPP_MIN_LOG_LEVEL
  - Tensorflow는 TF_CPP_MIN_LOG_LEVEL 환경 변수를 통해 로깅을 제어
- 로그레벨

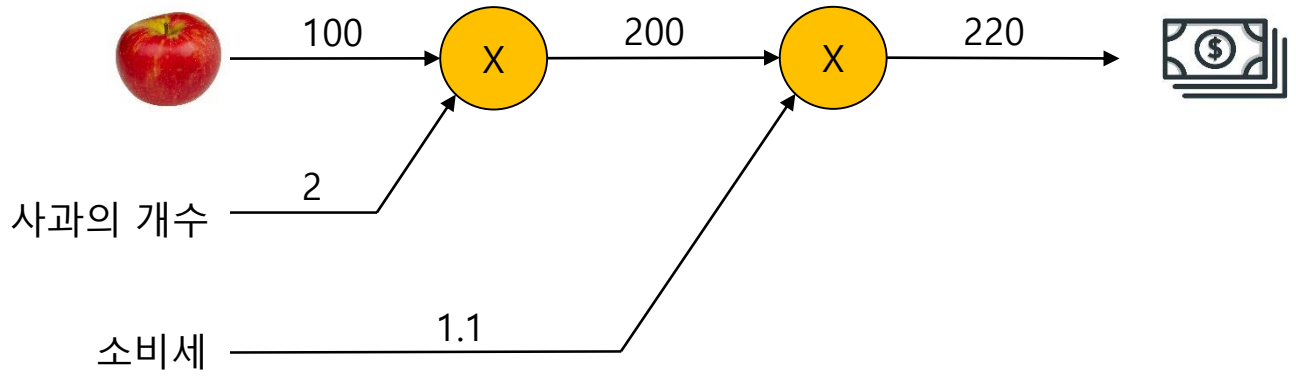
Level	Level for Humans	Description
0	DEBUG	Print all messages
1	INFO	Filter out INFO messages
2	WARNING	Filter out INFO & WARNING messages
3	ERROR	Filter out INFO & WARNING & ERROR messages

- 필터링 방법

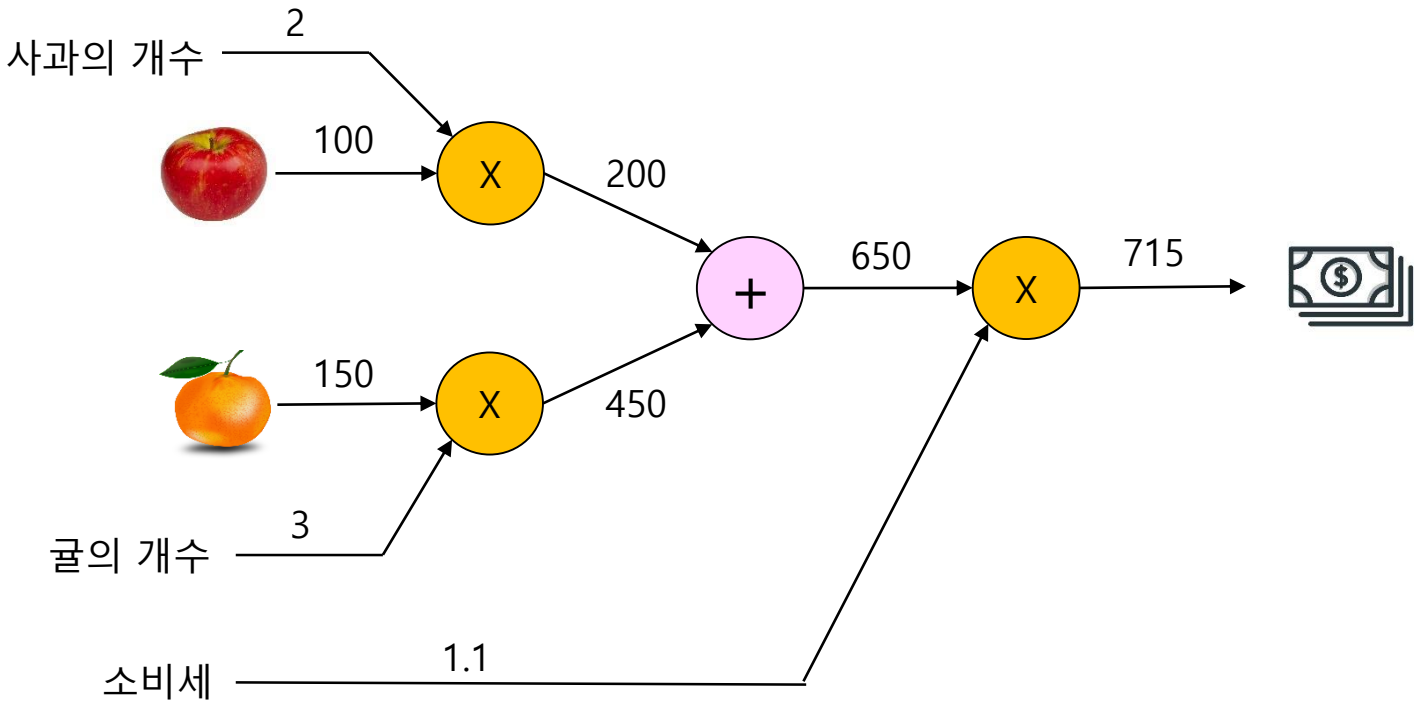
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

계산그래프(computational graph)

- 계산과정을 그래프(graph)로 표현하여 해결하는 방법
- (예) 1개에 100원인 사과를 2개 샀다. 지불금액은 얼마인가? 단, 소비세가 10% 부과된다.

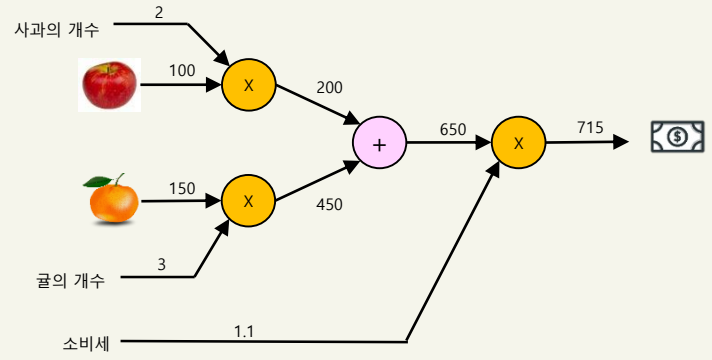


- (예) 사과를 2개, 귤은 3개 샀다. 사과는 1개에 100원, 귤은 1개에 150원이다. 지불금액은 얼마인가? 단, 소비세가 10% 부과된다.



Tensorflow 1.0 으로 구현한 예

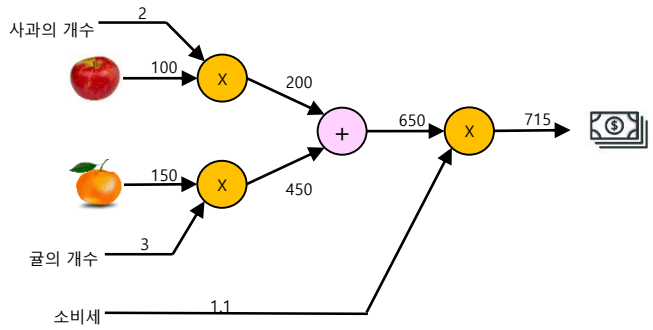
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

# placeholder data
apples = [2, 3, 4, 5]
oranges = [3, 5, 7, 9]

# constant
price_of_apple = tf.constant(100, dtype=tf.int16)
price_of_orange = tf.constant(150, dtype=tf.int16)
tax = tf.constant(1.1, dtype=tf.float32)

# placeholder
num_of_apple = tf.placeholder(tf.int16)
num_of_orange = tf.placeholder(tf.int16)

# nodes
price_of_apples = tf.multiply(price_of_apple, num_of_apple)
price_of_oranges = tf.multiply(price_of_orange, num_of_orange)

sum = tf.add(price_of_apples, price_of_oranges)

total_cost = tf.multiply(tf.cast(sum, tf.float32), tax)
```



```
# launch the graph in a session
with tf.Session() as sess:
    # initialize global variables
    sess.run(tf.global_variables_initializer())
    # computation of total_costs
    for i in range(4):
        _c = sess.run(total_cost, \
                      feed_dict={num_of_apple:apples[i], \
                                 num_of_orange:oranges[i]})
        print("apple =", apples[i], "orange =", oranges[i], \
              "then total_cost =", _c)
```

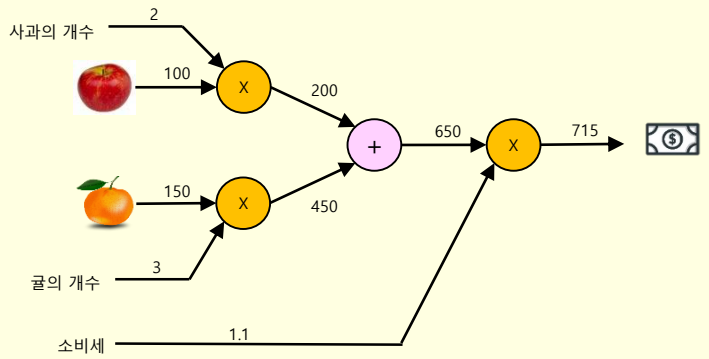
```
apple = 2 orange = 3 then total_cost = 715.0
apple = 3 orange = 5 then total_cost = 1155.0
apple = 4 orange = 7 then total_cost = 1595.0
apple = 5 orange = 9 then total_cost = 2035.0
```

```
# placeholder data
apples  = [2, 3, 4, 5]
oranges = [3, 5, 7, 9]
```



Tensorflow 2.0 으로 구현한 예

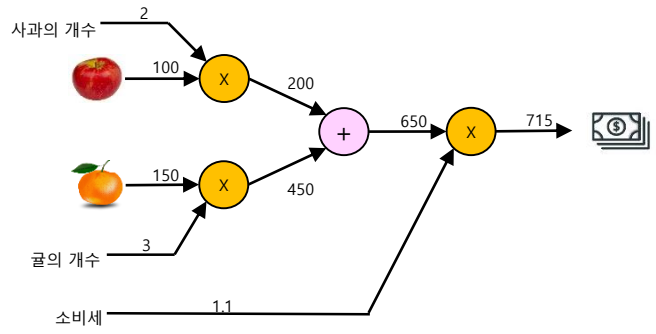
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

# data
apples = [2, 3, 4, 5]
oranges = [3, 5, 7, 9]

# constant
price_of_apple = tf.constant(100, dtype=tf.int16)
price_of_orange = tf.constant(150, dtype=tf.int16)
tax = tf.constant(1.1, dtype=tf.float32)

def run(num_of_apple, num_of_orange):
    price_of_apples = tf.multiply(price_of_apple, num_of_apple)
    price_of_oranges = tf.multiply(price_of_orange, num_of_orange)
    sum = tf.add(price_of_apples, price_of_oranges)
    total_cost = tf.multiply(tf.cast(sum, tf.float32), tax)
    return total_cost.numpy()
```



```
# computation of total_costs
for i in range(4):
    c = run(apples[i], oranges[i])
    print("apple =", apples[i], "orange =", oranges[i], \
          "then total_cost =", c)
```

```
apple = 2 orange = 3 then total_cost = 715.0
apple = 3 orange = 5 then total_cost = 1155.0
apple = 4 orange = 7 then total_cost = 1595.0
apple = 5 orange = 9 then total_cost = 2035.0
```

```
# placeholder data
apples  = [2, 3, 4, 5]
oranges = [3, 5, 7, 9]
```



(2) Tensor와 자료형

Tensor

- **Tensor는 0 차원 부터 n차원까지를 가지는 대표적인 데이터 클래스**
  - **0차 텐서 : Scalar**
    - 벡터가 없기 때문에 스칼라
  - **1차 텐서 : Vector**
    - 벡터가 하나 있기 때문에 벡터
  - **2차 텐서 : Matrix**
    - 벡터가 두 개 일렬로 있기 때문에 행렬
  - **3차 텐서 : Cube**
    - 2차 텐서가 일렬로 있기 때문에 큐브
  - **N차 텐서 : n-1차 텐서가 일렬로 구성됨**

```python
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

scalar = tf.constant(100)
vector = tf.constant([1,2,3,4,5])
matrix = tf.constant([[1,2,3],
                      [4,5,6]])
cube = tf.constant([[[1,2],[3,4]],
                    [[5,6],[7,8]],
                    [[9,10],[11,12]]])
print(scalar.get_shape())
print(vector.get_shape())
print(matrix.get_shape())
print(cube.get_shape())
print()
print(f"scalar = {scalar}")
print(f"vector = {vector}")
print(f"matrix =\n{matrix}")
print(f"cube =\n{cube}")
```

```
()
(5,) # 1개일때 ,를 붙임
(2, 3)
(3, 2, 2)

scalar = 100
vector = [1 2 3 4 5]
matrix =
[[1 2 3]
 [4 5 6]]
cube =
[[[ 1  2]
  [ 3  4]]

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
  [11 12]]]
```

Tensor의 자료형

data type	type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating po ints: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating po ints: real and imaginary parts.

tf.Variable()

- **tf.Variable(<initial-value>, name=<optional-name>)**

```
import tensorflow as tf
import numpy as np

y = tf.Variable([1,2,3,4,5])
print(y.name)
print(y.value)
print(y.dtype, "\n")
print(y)
print("rank =", y.numpy().ndim)
print("shape =", y.numpy().shape)
print("size =", y.numpy().size)
print("dtype =", y.numpy().dtype)
print("itemsize =", y.numpy().itemsize)
```

```
Variable:0
<bound method BaseResourceVariable.value of <tf.Variable 'Variable:0' shape=(5,) dtype=int32, numpy=array([1, 2, 3,
4, 5])>>
<dtype: 'int32'>

<tf.Variable 'Variable:0' shape=(5,) dtype=int32, numpy=array([1, 2, 3, 4, 5])>
rank = 1
shape = (5,)
size = 5
dtype = int32
itemsize = 4
```

tf.constant()

- c = tf.constant(<value>)
  - 상수정의
  - 상수로 정의된 c도 Tensor

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

a = tf.constant(1)
b = tf.constant(2)
c = tf.add(a, b)

print(f"c = {c.numpy()}")
```

```
c = 3
```

일정한 값으로 상수생성

- tf.zeros(shape, <dtype>)
- tf.ones(shape, <dtype>)
- tf.fill(shape, value, name=None)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

x1 = tf.zeros([3,4], tf.int32)
x2 = tf.ones([3,4], tf.int32)
x3 = tf.fill([3,4], 3)

print(f"zeros =\n {x1.numpy()}\n")
print(f"ones =\n {x2.numpy()}\n")
print(f"fill =\n {x3.numpy()}")
```

```
zeros =
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]

ones =
 [[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]

fill =
 [[3 3 3 3]
 [3 3 3 3]
 [3 3 3 3]]
```

Sequence값으로 상수생성

- tf.linspace(start, end, num, name=None)
- tf.range(start, limit, delta, dtype=None, name=None)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

y1 = tf.linspace(10.0, 20.0, 3, name='y1')
y2 = tf.range(3, 18, 3, tf.int32, name='y2')
y3 = tf.range(3, 1, -0.5, tf.float32, name='y3')

print(f"linspace\n = {y1.numpy()}\n")
print(f"range =\n {y2.numpy()}\n")
print(f"range =\n {y3.numpy()}")
```

```
linspace
 = [10. 15. 20.]

range =
 [ 3  6  9 12 15]

range =
 [3.  2.5 2.  1.5]
```

Random값으로 상수생성

- tf.random.normal(shape, mean, stddev, dtype, seed, name)
- tf.random.uniform(shape, min, max, dtype, seed, name)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

z1 = tf.random.normal([2,3], -1, 4)       # mean=-1, stddev=4
z2 = tf.random.uniform([2,3], 1, 9)       # min=1, max=9

print(f"random.normal =\n{z1.numpy()}")
print(f"random.uniform =\n{z2.numpy()}")
```

```
random.normal =
[[ 4.9526978   0.28776753 -4.853117  ]
 [-2.9761887  -3.9380176   2.0844114 ]]
random.uniform =
[[8.270946  8.629451  2.8683615]
 [1.3540659 2.0960474 5.3683586]]
```

(3) Operation

Element-wise mathematical operations

- 주요함수
  - 사칙연산
    - add, subtract, multiply, divide(div), mod
  - 부호관련
    - abs, negative, sign
  - 기타 수학관련연산
    - square, round, pow, sqrt, exp, log
    - maximum, minimum
    - sin, cos, tan
    - ceil, floor

Element-wise mathematical operations

- Scalar 값에 대한 사칙연산
  - add(), subtract(), multiply(), divide()

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

x = tf.constant(20)
y = tf.constant(7)

add = tf.add(x, y)
sub = tf.subtract(x, y)
mul = tf.multiply(x, y)
div = tf.divide(x, y)

print(add.numpy())
print(sub.numpy())
print(mul.numpy())
print(div.numpy())
```

```
27
13
140
2.857142857142857
```

절대값, 부호

- abs(), negative(), sign()

$$\begin{cases} -1 : if\ x < 0 \\ 0 : if\ x = 0 \\ 1 : if\ x > 0 \end{cases}$$

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

x = tf.Variable([-5,3])
y = tf.abs(x)
z = tf.negative(y)
p = tf.sign(z)

print(x.numpy())
print(y.numpy())
print(z.numpy())
print(p.numpy())
```

```
[-5  3]
[5 3]
[-5 -3]
[-1 -1]
```

- 벡터에 대한 나누기 및 나머지 연산
  - divide(), mod()
- 천정값 / 마루값
  - ceil(), floor()

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

x = tf.constant([1,2,3])
y = tf.constant([4,5,6])
z = tf.constant(3.75)

print(tf.divide(x,y).numpy())
print(tf.math.mod(x, y).numpy())
print(tf.math.ceil(z).numpy())
print(tf.math.floor(z).numpy())
```

```
[ 0.25  0.4   0.5 ]
[1 2 3]
4.0
3.0
```

미분계수 구하기

```
import tensorflow as tf

def myGradient(x):
    with tf.GradientTape() as tape:
        tape.watch(x) # 상수형 텐서인 경우에는
                      # 반드시 변수형 텐서처럼 바꿔야 함
        y = tf.multiply(2.0, tf.pow(x, 2.0))  # y=2*x**2
    return tape.gradient(y, x).numpy()

a = myGradient(tf.constant(1.0))
b = myGradient(tf.constant(3.0))
print(a)
print(b)
```

$$y = 2x^2$$

$$\frac{\partial y}{\partial x} = 4x$$

```
4.0
12.0
```

복소수 처리

- tf.dtypes.complex()
- tf.math.conj(), tf.math.real(), tf.math.imag()

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

real = tf.constant([2.25, 3.50])
imag = tf.constant([4.75, 2.75])
c = tf.dtypes.complex(real, imag)

print(c.numpy())
print(tf.math.conj(c).numpy())
print(tf.math.real(c).numpy())
print(tf.math.imag(c).numpy())
```

```
[ 2.25+4.75j  3.50+2.75j]
[ 2.25-4.75j  3.50-2.75j]
[ 2.25  3.5 ]
[ 4.75  2.75]
```

Matrix operations

- tf.reduce_sum(),tf.reduce_mean()
- tf.reduce_min(),tf.reduce_max()

```
import tensorflow as tf

mat1 = tf.constant([[1,2,3],[4,5,6]], tf.float32)

colsum = tf.reduce_sum(mat1, axis=0)    # row축을 압축
rowsum = tf.reduce_sum(mat1, axis=1)    # column축을 압축
sum = tf.reduce_sum(mat1)
mean = tf.reduce_mean(mat1)
min = tf.reduce_min(mat1)
max = tf.reduce_max(mat1)

print('mat1=\n', mat1.numpy())
print('colsum=', colsum.numpy())
print('rowsum=', rowsum.numpy())
print('sum=', sum.numpy())
print('mean=', mean.numpy())
print('min=', min.numpy())
print('max=', max.numpy())
```

```
mat1=
 [[ 1.  2.  3.]
 [ 4.  5.  6.]]
colsum= [ 5.  7.  9.]
rowsum= [  6.  15.]
sum= 21.0
mean= 3.5
min= 1.0
max= 6.0
```

행렬의 곱셈

- **tf.matmul()**

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

mat1 = tf.constant([[1,2,3],[4,5,6]], tf.float32)
mat2 = tf.constant([[9,8],[7,6],[5,4]], tf.float32)

mat3 = tf.matmul(mat1, mat2)

print('mat1(2,3)=\n', mat1.numpy())
print('mat2(3,2)=\n', mat2.numpy())
print('mat3(2,2)=\n', mat3.numpy())
```

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \times \begin{bmatrix}9 & 8\\ 7 & 6\\ 5 & 4\end{bmatrix} = \begin{bmatrix}38 & 32\\ 101 & 86\end{bmatrix}$$

```
mat1(2,3)=
 [[ 1.  2.  3.]
 [ 4.  5.  6.]]
mat2(3,2)=
 [[ 9.  8.]
 [ 7.  6.]
 [ 5.  4.]]
mat3(2,2)=
 [[  38.   32.]
 [ 101.   86.]]
```

tf.argmax() / tf.argmin()

- argmax(input, axis=None, name=None)
- argmin(input, axis=None, name=None)
  - 반환: 축(axis)방향으로 최대/최소값을 갖는 index

```
import tensorflow as tf

list = tf.constant([1,2,3,4,5])
mat = tf.Variable(tf.random.uniform([3,3]))
max1 = tf.argmax(list, axis=0)
max2 = tf.argmax(mat, axis=0)   # 0(row direction): column max
max3 = tf.argmax(mat, axis=1)   # 1(col direction): row max

print('mat=\n', mat.numpy())
print('argmax of list =\n', max1.numpy())
print('argmax of mat(0 axis) =\n', max2.numpy())
print('argmax of mat(1 axis) =\n', max3.numpy())
```

```
mat=
 [[0.65266836 0.75492203 0.08796299]
 [0.47562075 0.5926542  0.90533984]
 [0.5489801  0.9219023  0.6279311 ]]
argmax of list =
 4
argmax of mat(0 axis) =
 [0 2 1]
argmax of mat(1 axis) =
 [1 2 1]
```

shape 변경

- **tf.reshape(tensor, shape, name=None)**

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

x = tf.constant([1,2,3,4,5,6,7,8,9])
mat1 = tf.reshape(x, [3,3])
y = tf.constant([1,2,3,4,5,6,7,8])
mat2 = tf.reshape(y, [2,-1]) # -1 is inferred

print('x=', x.numpy())
print('mat1=\n', mat1.numpy())
print('\ny=', y.numpy())
print('mat2=\n', mat2.numpy())
```

```
x= [1 2 3 4 5 6 7 8 9]
mat1=
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

y= [1 2 3 4 5 6 7 8]
mat2=
 [[1 2 3 4]
 [5 6 7 8]]
```

역행렬, 행렬식

- **tf.linalg.inv(), tf.linalg.det()**

```
import tensorflow as tf

mat = tf.Variable(tf.random.uniform([3,3]))
invmat = tf.linalg.inv(mat)
det = tf.linalg.det(mat)

print('mat=\n', mat.numpy())
print('inverse of mat=\n', invmat.numpy())
print('determinent of mat =', det.numpy())
```

```
mat=
 [[0.6985841  0.39672828 0.41619956]
 [0.51830316 0.49340367 0.8835522 ]
 [0.9252442  0.5337249  0.9392698 ]]
inverse of mat=
 [[-0.16064888 -2.9721544   2.8670304 ]
 [ 6.5303993   5.3533297  -7.9294524 ]
 [-3.552544   -0.11417056  2.7462215 ]]
determinent of mat = 0.050636277
```

참고:

- Inverse Matrix(역행렬)
  - 행렬 A와 곱하면 단위행렬I가 나오는 행렬 A^{-1}를 역행렬이라고 함(즉, $A \cdot A^{-1} = A^{-1} \cdot A = I$)
  - 선형방정식 풀이에 주로 사용됨
  - 정방행렬에 대해서만 정의됨

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

- Determinant(행렬식)
  - 어떤 행렬의 역행렬 존재여부에 대한 판별값
    - det의 값이 0이면 역행렬 없음
  - 정방행렬에 대해서만 정의됨

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \det(A) = ad - bc$$

전치행렬

- **tf.linalg.matrix_transpose()**

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

mat1 = tf.Variable(tf.random.uniform([2,3], \
                maxval=10, dtype=tf.int32))
tr1 = tf.linalg.matrix_transpose(mat1)

mat2 = tf.Variable(tf.random.uniform([2,2,3], \
                maxval=10, dtype=tf.int32))
tr2 = tf.linalg.matrix_transpose(mat2)

print('mat1=\n', mat1.numpy())
print('tran mat1=\n', tr1.numpy())
print('\nmat2=\n', mat2.numpy())
print('tran mat2=\n', tr2.numpy())
```

```
mat1=
 [[1 2 3]
 [3 2 2]]
tran mat1=
 [[1 3]
 [2 2]
 [3 2]]

mat2=
 [[[6 3 0]
  [7 0 3]]

 [[9 7 1]
  [1 2 1]]]
tran mat2=
 [[[6 7]
  [3 0]
  [0 3]]

 [[9 1]
  [7 2]
  [1 1]]]
```

대각행렬, 단위행렬

- tf.linalg.trace() : 대각원소의 합 구하기
- tf.linalg.diag() : 대각행렬 만들기
- tf.linalg.eye() : 단위행렬 만들기

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

x = tf.constant([1,2,3,4])
mat1 = tf.constant([[1,2,3],\
                    [4,5,6],\
                    [7,8,9]])
mat2 = tf.linalg.diag(x)
mat3 = tf.linalg.eye(3)    # 정방형 단위행렬
mat4 = tf.linalg.eye(2, 3) # 비정방형 단위행렬
dsum = tf.linalg.trace(mat1)    # 주 대각선요소의 합

print('mat2=\n', mat2.numpy())
print('mat3=\n', mat3.numpy())
print('mat4=\n', mat4.numpy())
print('dsum=', dsum.numpy()) # 1+5+9=15
```

```
mat2=
 [[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
mat3=
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
mat4=
 [[1. 0. 0.]
 [0. 1. 0.]]
dsum= 15
```

norm

- norm
  - 벡터의 크기
    - (예) $a = \begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix}$ $\|a\| = \sqrt{3^2 + (-2)^2 + 1^2} = 3.742$

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

v = tf.constant([3,-2, 1], dtype=tf.float32)

print('v=', v.numpy())
print('norm=', tf.norm(v).numpy())
```

```
v= [ 3. -2.  1.]
norm= 3.7416575
```

(4) Tensorboard

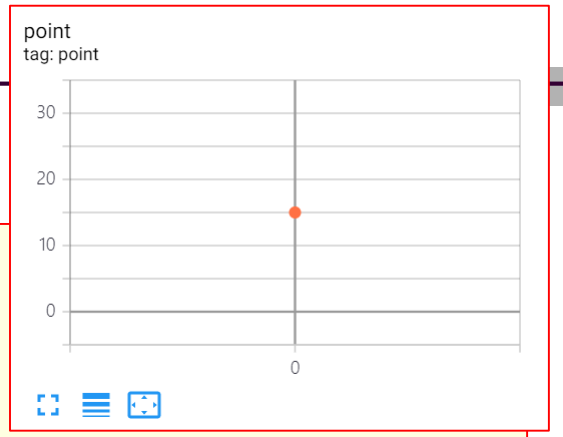
Tensorboard

- Tensorflow의 내용을 visual하게 확인할 수 있는 도구

- 일반적인 사용방법
  1. FileWriter(저장할 log경로 지정)생성
     - writer = tf.summary.create_file_writer("./log")
  2. 필요한 summary를 추가
     - tf.summary.scalar('point', c, step)
  3. Terminal에서 Tensorboard 실행하고,
     표시된 URL을 브라우저(Chrome)로 확인
     - tensorboard --logdir=./log

Tensorboard 예제1

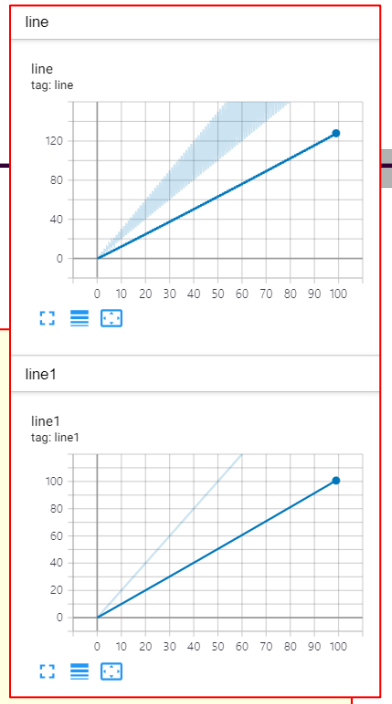
- 점 찍기




```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import matplotlib.pyplot as plt

a = tf.constant(3.0)
b = tf.constant(5.0)
c = a * b

writer = tf.summary.create_file_writer(r"c:\temp\log_ex1")

with writer.as_default():
    tf.summary.scalar('point', c.numpy(), 0)
    writer.flush()
```

Tensorboard 예제2

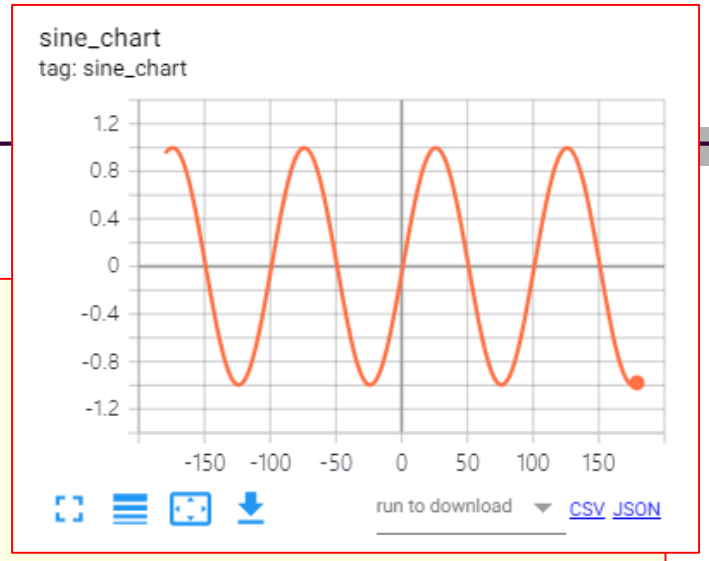
- 선 그리기




```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import matplotlib.pyplot as plt

def mul(X,Y):
    return tf.multiply(X, Y)

writer = tf.summary.create_file_writer(r"c:\temp\log_ex2\mul1")
writer = tf.summary.create_file_writer(r"c:\temp\log_ex2\mul2")

for step in range(100):
    with writer.as_default():
        tf.summary.scalar('line1', mul(step*1.0, 2.0), step=step)
        tf.summary.scalar('line2', mul(step*1.0, 3.0), step=step)
        writer.flush()
```

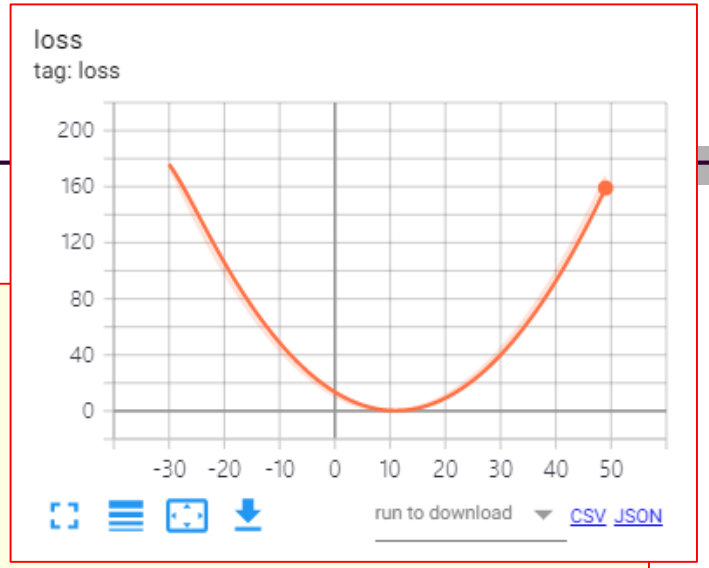
Tensorboard 예제3

- sine곡선 그리기




```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import matplotlib.pyplot as plt
import math

def sine(X):
    return tf.sin((X/100.0)*2.0*math.pi)

writer = tf.summary.create_file_writer(r"c:\temp\log_ex3")

for step in range(-180, 180):
    with writer.as_default():
        tf.summary.scalar('sine_chart', sine(step*1.0), step=step)
        writer.flush()
```

Tensorboard 예제4



- Cost함수 그리기

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import matplotlib.pyplot as plt

X = tf.constant([1,2,3,4,5], dtype=tf.float32)
Y = tf.constant([1,2,3,4,5], dtype=tf.float32)

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

writer = tf.summary.create_file_writer(r"c:\temp\log_ex4")

for i in range(-30, 50):
    _c = loss(i*0.1)
    with writer.as_default():
        tf.summary.scalar('loss', _c.numpy(), step=i)
        writer.flush()
```

회귀(Regression)

(1) Linear Regression이란?

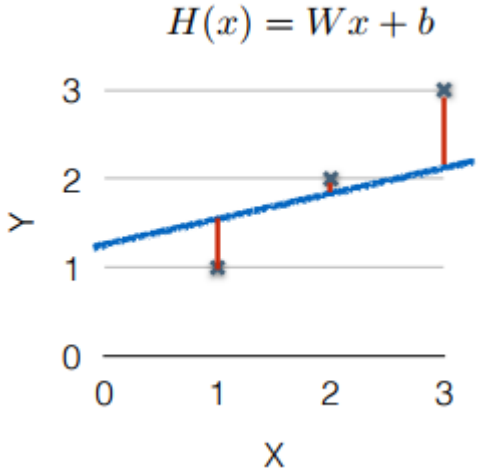
Regression이란?

- 회귀분석(Regression Analysis)
  - 2개 또는 그 이상 변수들의 의존관계를 파악함으로써 특정 변수(종속변수)의 값을 예측하는 통계학의 한 분야

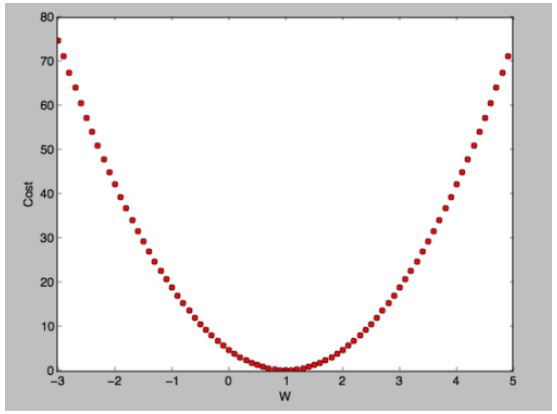
- Linear Regression Analysis(선형 회귀분석)
  - 두 변수 x, y에 대한 n개의 측정값 (x1, y1), (x2, y2), ···, (xn, yn)이 있을 때
  - 주어진 가설(hypothesis)에 대한 비용(cost)이 최소화 되도록 하는 직선을 찾는 문제

Linear Regression

- **Linear Hypothesis**
  - $H(x) = Wx + b$



- **Cost Function**
  - $cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right) - y^i\right)^2$
  - cost(W)함수의 모양
  - Tensorboard 예제 참고



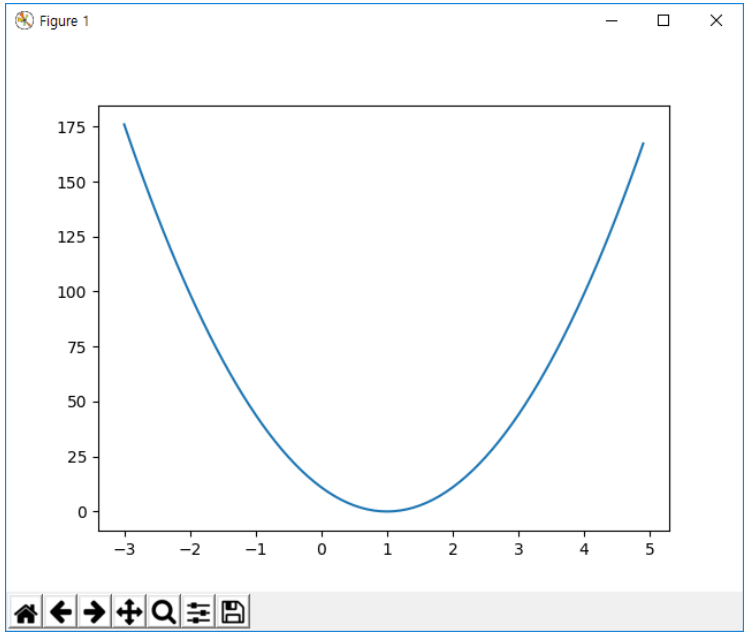
- **Linear Regression이란?**
  - cost함수 $cost(W, b)$ 를 최소화하는 W와 b를 찾는 문제

(2) 가설, 손실함수, 경사하강법

cost함수의 모양

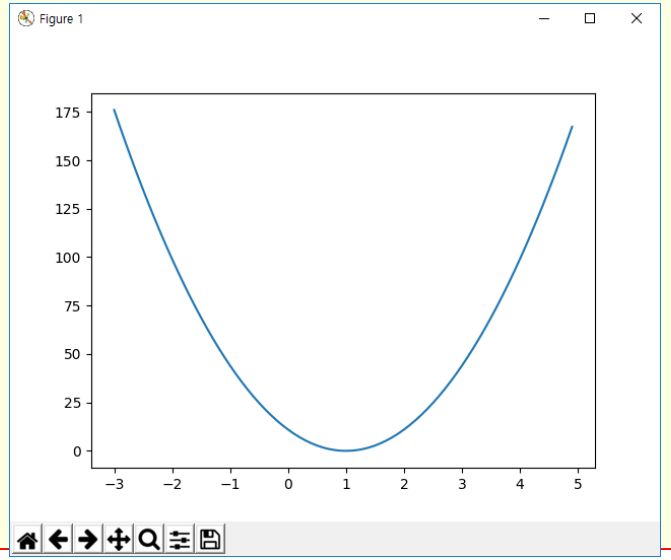
- cost함수를 단순화시켜서 생각해 보자!
  - $H(x) = Wx$
  - $cost(W) = \frac{1}{m}\sum_{i=1}^{m}\left(Wx^i - y^i\right)^2$
- 계산결과
  - x = {1, 2, 3, 4, 5}
  - y = {1, 2, 3, 4, 5}

```
w = -3.0        c = 176.0
w = -2.0        c =  99.0
w = -1.0        c =  44.0
w =  0.0        c =  11.0
w =  1.0        c =   0.0
w =  2.0        c =  11.0
w =  3.0        c =  44.0
w =  4.0        c =  99.0
```

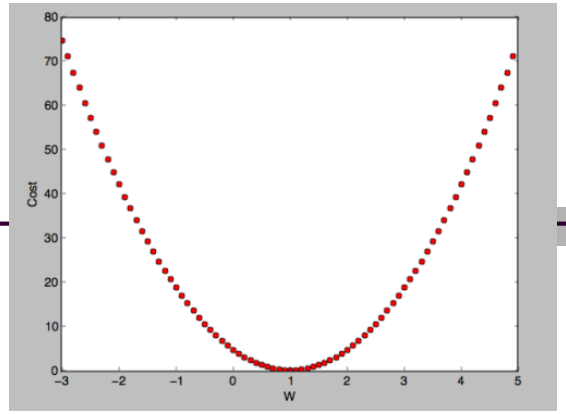


```python
import tensorflow as tf
import matplotlib.pyplot as plt

X = tf.constant([1,2,3,4,5], dtype=tf.float32)
Y = tf.constant([1,2,3,4,5], dtype=tf.float32)

def run(W):
    hypothesis = W * X
    cost = tf.reduce_mean(tf.square(hypothesis - Y))
    return cost.numpy()

W_val = []
cost_val = []
for i in range(-30, 50):
    w = i * 0.1
    c = run(w)

    if i%10 == 0:
        print('w = ', w, end='\t')
        print('c = ', c)
    W_val.append(w)
    cost_val.append(c)

plt.plot(W_val, cost_val)
plt.show()
```

```
w =  -3.0  c =  176.0
w =  -2.0  c =  99.0
w =  -1.0  c =  44.0
w =  0.0  c =  11.0
w =  1.0  c =  0.0
w =  2.0  c =  11.0
w =  3.0  c =  44.0
w =  4.0  c =  99.0
```



How to minimize cost?

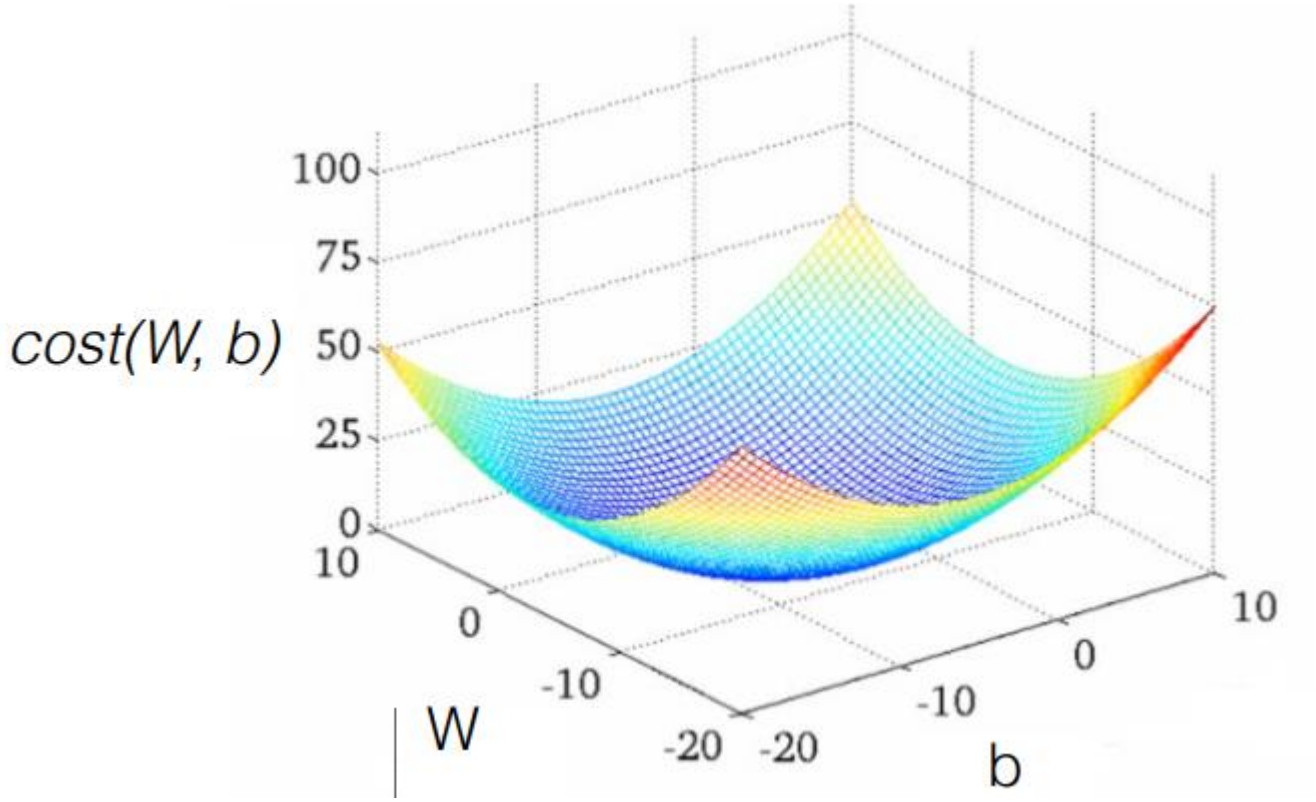


- 경사하강알고리즘을 이용
  - Gradient descent algorithm
    - 임의의 곳에서 시작하여 경사도(gradient)에 따라 W를 변경시켜가면서 cost함수의 값이 최소화되는 W를 구하는 알고리즘
  - 경사도(gradient)는 미분값
  - W값의 변화
    - $W = W - \alpha \frac{\partial}{\partial W} cost(W)$
    - α는 learning rate
- **Gradient descent algorithm**

$$W = W - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(W x^i - y^i \right) x^i$$

가중치와 편향이 고려된 경사하강법

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



(3) Linear Regression 구현

Linear Regression(Tensorflow로 구현)

(1) Graph 구성(가설, 비용함수, 학습함수 정의)

- Hypothesis(가설) 정의
  - 여기에 사용되는 변수(Variable)생성
- Cost/Loss함수 정의
- Train함수 정의

(2) Training…

- 충분한 만큼 반복하면서…
  - Graph상의 Tensor들을 실행
    - Train 함수 실행
    - Loss 함수 실행
  - 결과를 출력(또는, Tensorboard로 확인)

(3) Testing…

참고: @tf.function 데코레이터

- **@tf.function**
  - 파이썬 문법의 일부를 높은 성능의 텐서플로 그래프 코드로 변환시키기 위해 사용
  - 오토그래프 (AutoGraph)
    - Python 문법을 활용해서 그래프 코드를 작성

- **@tf.function을 함수에 붙여주는 경우**
  - 일반 함수들처럼 사용 가능
  - 그래프 내에서 컴파일 되었을 때는 더 빠르게 실행하고, GPU 또는 TPU를 사용해서 작동함
    - TPU : Tensorflow Processing Unit
  - 작은 연산을 많이 포함하는 경우 빠르게 동작함

Graph 구성

- Hypothesis(가설) 정의
  - $H(x) = Wx + b$

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

W = tf.Variable(tf.random.normal([1]), name='Weight')
B = tf.Variable(tf.random.normal([1]), name='Bias')

@tf.function
def Hypothesis(X):
    return W * X + B
```

- Cost/Loss함수 정의
  - $cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x^i\right) - y^i\right)^2$

```
@tf.function
def loss(H, Y):
    return tf.reduce_mean(tf.square(H - Y))
```

Train함수 정의

```
@tf.function
def train(X, Y, learning_rate=0.01):
    with tf.GradientTape() as tape:
        _loss = loss(Hypothesis(X), Y)
    _w, _b = tape.gradient(_loss, [W, B])
    W.assign_sub(learning_rate * _w)  # 텐서의 값 변경을 위한 메소드
    B.assign_sub(learning_rate * _b)  # assign(=), assign_add(+=),
                                        assign_sub(-=)
```

Training

- 충분한 만큼 반복하면서...
  - Graph상의 Tensor들을 실행
    - Train 함수 실행
    - Loss 함수 실행
  - 결과를 출력(또는, Tensorboard로 확인)

```
for step in range(2001):
    train(x_data, y_data, learning_rate=0.01)
    _c = loss(Hypothesis(x_data), y_data)
    if step % 20 == 0:
        print(f"{step}: {_c.numpy()} {W.numpy()} {B.numpy()}")
print('\nfinal W =', W.numpy(), 'b =', B.numpy())
```

```
0: 23.687816619873047 [-0.6621914] [0.72487795]
20: 0.1931050717830658 [0.7089279] [1.0270315]
40: 0.16822198033332825 [0.7336918] [0.9613495]
...
1000: 0.0002522937720641494 [0.98968774] [0.03723036]
1020: 0.00022033273125998676 [0.9903631] [0.03479213]
...
1980: 3.305472660031228e-07 [0.9996267] [0.00134762]
2000: 2.8870039159301086e-07 [0.99965113] [0.00125941]
```

Testing

```
for step in range(2001):
    train(x_data, y_data, learning_rate=0.01)
    _c = loss(Hypothesis(x_data), y_data)
    if step % 20 == 0:
        print(f"{step}: {_c.numpy()} {W.numpy()} {B.numpy()}")
print('\nfinal W =', W.numpy(), 'b =', B.numpy())
```

```
#test data
test_data = [2, 4, 1, 5, 3]
for data in test_data:
    y = data * W.numpy() + B.numpy()
    print("X =", data, "then Y =", y)
```

```
final W = [0.99965113] b = [0.00125941]
X = 2 then Y = [2.0005617]
X = 4 then Y = [3.9998639]
X = 1 then Y = [1.0009105]
X = 5 then Y = [4.999515]
X = 3 then Y = [3.0002127]
```

전체코드

```
import tensorflow as tf

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

W = tf.Variable(tf.random.normal([1]), name='Weight')
B = tf.Variable(tf.random.normal([1]), name='Bias')

@tf.function
def Hypothesis(X):
    return W * X + B

@tf.function
def loss(H, Y):
    return tf.reduce_mean(tf.square(H - Y))

@tf.function
def train(X, Y, learning_rate=0.01):
    with tf.GradientTape() as tape:
        _loss = loss(Hypothesis(X), Y)
    _w, _b = tape.gradient(_loss, [W, B])
    W.assign_sub(learning_rate * _w)  # 텐서의 값 변경을 위한 메소드
    B.assign_sub(learning_rate * _b)  # assign(=), assign_add(+=), assign_sub(-=)

for step in range(2001):
    train(x_data, y_data, learning_rate=0.01)
    _c = loss(Hypothesis(x_data), y_data)
    if step % 20 == 0:
        print(f"{step}: {_c.numpy()} {W.numpy()} {B.numpy()}")
print('\nfinal W =', W.numpy(), 'b =', B.numpy())

#test data
test_data = [2, 4, 1, 5, 3]
for data in test_data:
    y = data * W.numpy() + B.numpy()
    print("X =", data, "then Y =", y)
```