Machine Learning with TensorFlow

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy

TensorFlow Project에서 만나는 문제

- Version
  - https://github.com/tensorflow/tensorflow/releases
- Optimization
- SI적 관점으로 Deep Learning Project에 접근하는 개발 문화
  - SI → Solution
  - We → Optimization

TensorFlow Project에서 만나는 문제 (Cont.)

- Deprecated
  - Tensorflow1.0.0
    - tf.multiply, tf.subtract, tf.negative 가 추가되고 tf.mul, tf.sub, tf.neg 는 deprecated
    - tf.scalar_summary, tf.histogram_summary 같은 summary 연산자가 삭제되고 tf.summary.scalar, tf.summary.histogram 이 추가
  - TensorFlow 1.2.0-rc1
- https://github.com/tensorflow/tensorflow/releases
- 적용 스크립트
  - http://github.com/Finfra/TensorflowInstallMultiVersionWithJupyter

TensorFlow Execution Environment

- Jupyter Notebook

```
1  import tensorflow as tf
2
3  tf.__version__
```

```
'1.14.0'
```

- Google Colaboratory

```
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU 장치를 찾지 못했습니다.')
print('GPU 장치를 찾았습니다. 장치 : {}'.format(device_name))
```

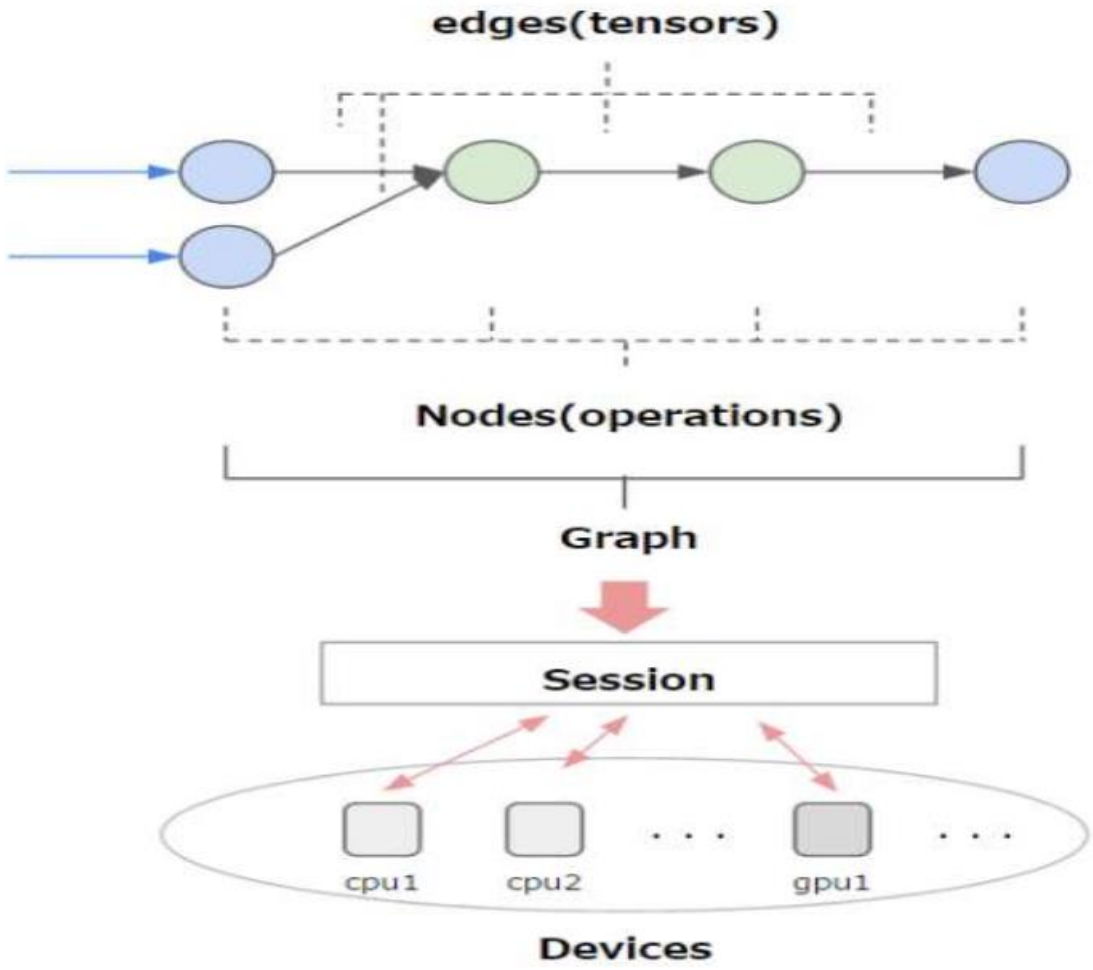
```
GPU 장치를 찾았습니다. 장치 : /device:GPU:0
```

TensorFlow

- Data Flow Graph를 사용하여 수치 연산을 하는 Open Source Library
- Graph의 Node는 수치 연산을 나타내고 Edge는 Node 사이를 이동하는 다차원 Data 배열(=Tensor)을 나타낸다.
- 유연한 Architecture로 한 번 작성하면 코드 수정 없이 Desktop, Server, 혹은 Mobile device에서 CPU나 GPU를 사용하여 연산 구동.
- TernsorFlow는 Machine Learning과 Deep Learning을 위해 Google에서 만든 Open Source Library(2005. 11월 Open)
- Cross Platform

TensorFlow (Cont.)

- Edges와 Nodes로 구조화된 Graph로 프로그램이 구성됨.




```
import tensorflow as tf

a = tf.constant(1)
print(a)
with tf.Session() as sess:
    print(a.eval())
```

```
Tensor("Const_170:0", shape=(), dtype=int32)
1
```

TensorFlow (Cont.)

- All TensorFlow codes contain two important parts:
  - Part 1: building the ***GRAPH***, it represents the data flow of the computations
  - Part 2: running a ***SESSION***, it executes the operations in the graph

TensorFlow Code의 시작

- 필요한 Library를 import 하자.
  - 가장 많이 쓰는 library 3개
  - **tensorflow**, **numpy**, **matplotlib**

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

%matplotlib inline
```

Computational Graph 정의와 실행의 분리 이해

- a = a + 1인 Graph 정의하여, 1,2,3 값을 출력하기
- **Lazy Computing**

```
# computational graph 정의와 실행(session run)의 차이
# computational graph 정의:  a = a +1
a = tf.Variable(0)                  # a = 0
a = tf.add(a, tf.constant(1)) # a = a+1

# 정의된 computational graph 실행
sess = tf.Session()      # sess란 이름의 세션 선언
for _ in range(3):       # for문은 쓰는 건 일단 이렇게 보고 넘어가시죠
    print(sess.run(a)) # sess.run을 통해 수행되며 해당값을 찍음
sess.close()             # 꼭 써야함!!!!! 안 그럼 메모리 차요
```

a=0
a=a+1

a=a+1로
업데이트하는
그래프를
3번 반복하기

Computational Graph 정의와 실행의 분리 이해 (Cont.)

- a = a + 1인 Graph 정의하여, 1,2,3 값을 출력하기

```
# computational graph 정의와 실행(session run)의 차이
# computational graph 정의:  a = a +1
a = tf.Variable(0)                  # a = 0
a = tf.add(a, tf.constant(1)) # a = a+1

# 정의된 computational graph 실행
sess = tf.Session()     # sess란 이름의 세션 선언
for _ in range(3):      # for문은 쓰는 건 일단 이렇게 보고 넘어가시죠
    print(sess.run(a)) # sess.run을 통해 수행되며 해당값을 찍음
sess.close()            # 꼭 써야함!!!!! 안 그럼 메모리 차요
```

a=0
a=a+1

a=a+1로 업데이트하는 그래프를 3번 반복하기

```
---------------------------------------------------------------------------
FailedPreconditionError                   Traceback (most recent call last)
<ipython-input-5-210999db8592> in <module>()
      7 sess = tf.Session()     # sess란 이름의 세션 선언
      8 for _ in range(3):      # for문은 쓰는 건 일단 이렇게 보고 넘어가시죠
----> 9     print(sess.run(a)) # sess.run을 통해 수행되며 해당값을 찍음

/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/session.pyc in run(self, fetc
hes, feed_dict, options, run_metadata)
    764     try:
    765       result = self._run(None, fetches, feed_dict, options_ptr,
--> 766                          run_metadata_ptr)
    767       if run_metadata:
    768         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/session.pyc in _run(self, han
dle, fetches, feed_dict, options, run_metadata)
    962     if final_fetches or final_targets:
    963       results = self._do_run(handle, final_targets, final_fetches,
--> 964                              feed_dict_string, options, run_metadata)
```

Variable 사용시 반드시 초기화(Initialization)

- 다른 Operation이 수행되기 전에 명시적으로 초기화해야 함.
- 모든 Variable들을 초기화시키는 명령어 활용
- sess 선언 바로 뒤에
  - **sess.run(tf.global_variables_initializer())**

```
# computational graph 정의와 실행(session run)의 차이
# computational graph 정의:  a = a +1
a = tf.Variable(0)
a = tf.add(a, tf.constant(1))

# 정의된 computational graph 실행
sess = tf.Session ()
sess.run(tf.global_variables_initializer()) # 초기화!!!! 꼭 해야해요!
for _ in range(3):
    print(sess.run(a))
sess.close()
```

Variable 사용시 반드시 초기화(Initialization) (Cont.)

- 다른 Operation이 수행되기 전에 명시적으로 초기화해야 함.
-
-

```
# computational graph 정의와 실행(session run)의 차이
# computational graph 정의:  a = a +1
a = tf.Variable(0)
a = tf.add(a, tf.constant(1))

# 정의된 computational graph 실행
sess = tf.Session ()
sess.run(tf.global_variables_initializer()) # 초기화!!!! 꼭 해야해요!
for _ in range(3):
    print(sess.run(a))
sess.close()
```

```
1
1
1
```

Variable 사용시 반드시 초기화(Initialization) (Cont.)

- 다른 Operation이 수행되기 전에 명시적으로 초기화해야 함.
- 모든 Variable들을 초기화시키는 명령어 활용
- ses
  - s

```
# computational graph 정의와 실행(session run)의 차이
# computational graph 정의:  a = a +1
a = tf.Variable(0)                    # a = 0
temp = tf.add(a, tf.constant(1)) # assign을 위해 temp 정의: temp = a+1
update = tf.assign(a, temp)          # a <- temp; a = a+1;

# 정의된 computational graph 실행
sess = tf.Session ()
sess.run(tf.global_variables_initializer()) # 초기화
for _ in range(3):
    sess.run(update)    # update를 실행
    print(sess.run(a)) # a값 출력을 실행
sess.close()
```

```
1
2
3
```

Terminology

- Tensor
  - 원래 의미는 2차원 이상, 임의의 차원을 가진 배열을 뜻한다.
  - TensorFlow는 방향성이 있는 Graph 구조로 model을 구성
  - Graph는 0개 이상의 입출력을 갖는 Node들의 연결체
  - Node는 Operation의 Instance라고 할 수 있음.
- Operation
  - TensorFlow에서 계산이 일어나는 단계를 의미
  - Tensor를 만들고 흐름을 구성하면서 흐름과 흐름 사이에서 자료의 곱셈이나 더하기, 뺄셈 등의 계산을 하는 단계
  - 다양한 속성 값(attribute)을 가질 수 있다.

Terminology (Cont.)

- Variable
  - 학습을 통해 변화하는 배열 값을 저장하기 위한 프로그램적인 구조물
  - TensorFlow가 학습할 때 다양한 Device에 설치 및 실행되며, 분산하여 처리되어 명시적 type(대부분은 실수형)을 지정해 준다.
- Session
  - TensorFlow에서 Graph를 구성한 후, 실제 graph를 수행을 할 수 있게 만들어 주는 프로그램을 의미
  - TensorFlow는 다양한 실행 환경(CPU, GPU, TPU, 원격 분산처리)에서 graph를 생성하고 이를 실행하기 위해 Client에서 Session을 만들어 전달한다.

Terminology (Cont.)

- GPU 가속 Computing
  - https://kr.nvidia.com/object/what-is-gpu-computing-kr.html
  - GPU와 CPU를 함께 이용하여 과학, 분석,공학, 소비자 및 기업 애플리케이션의 처리속도를 높이는 것
  - NVIDIA에 의해 2007년 개척
  - Application의 연산 집약적인 부분을 GPU로 넘기고 나머지 코드만을 CPU에서 처리하는 GPU 가속 컴퓨팅은 강력한 성능을 제공
  - CPU와의 차이는 그 작업 처리 방식을 비교해 보면, 하나의 CPU는 직렬 처리에 최적화된 몇 개의 Core로 구성된 반면, GPU는 병렬 처리용으로 설계된 수 천 개의 보다 소형이고 효율적인 Core로 구성되어 있다.

Terminology (Cont.)

- GPU 가속 Computing
  - TensorFlow Python API는 Python 2.7과 Python 3.3+를 지원
  - GPU 버전은 Linux만 지원하다가 현재는 MacOS, Windows, Linux 모두 지원
  - TensorFlow에서 학습 속도를 획기적으로 늘릴 수 있다.
  - 예를 들어 GPU없이 CPU만으로 며칠을 걸릴 기계학습도 GPU로는 단 몇 십분 만에 끝낼 수 있다.

What is a Tensor?

- TensorFlow programs use a data structure called ***tensor*** to represent all the data.
- TensorFlow에서 다양한 수학식을 계산하기 위한 가장 기본적이고 중요한 자료형.
- Simply, a ***Tensor*** is a multi-dimensional array.
  - 0-D tensor: scalar, rank 0
  - 1-D tensor: vector, rank 1
  - 2-D tensor: matrix, rank 2
  - 3이상이면 n-Tensor or n차원 Tensor

What is a Tensor? (Cont.)

- Hence, TensorFlow is simply referring to the flow of the ***Tensor***s in the computational graph.
- Rank와 Shape 개념

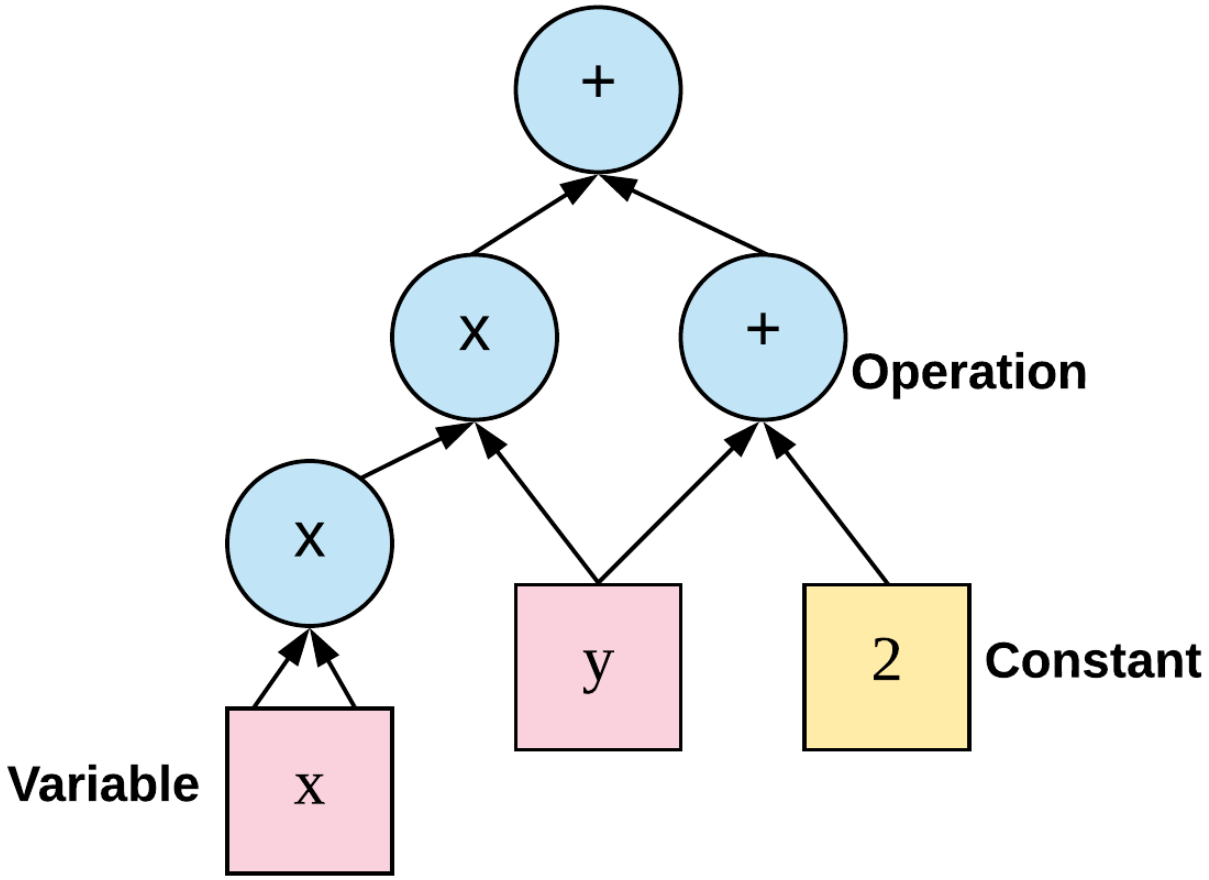
```
3                                #rank가 0인 tensor, shape []
[1., 2., 3. ]                    #rank가 1인 tensor, shape는 [3]
[[1., 2., 3.], [4., 5., 6.]]
                                 #rank가 2인 tensor,  shape는 [2,3]
[[[1., 2., 3.]], [[7., 8., 9.]]]
                                 #rank가 3인 tensor, shape는 [2,1,3]
```

GRAPH

- The biggest ideas are expressed as a computational ***graph***.
- In other words, the backbone of any TensorFlow program is a ***Graph***.
- A computational ***graph*** is a series of TensorFlow operations arranged into a graph of nodes
- A ***graph*** is just an arrangement of nodes that represent the operations in your model.

GRAPH (Cont.)

- Suppose, $f(x,y)=x^2y+y+2$.



GRAPH (Cont.)

- The graph is composed of a series of ***nodes*** connected to each other by ***edges***.
- Each ***node*** in the graph is called ***op***(***Operation***).
- So one node for each operation.
  - Either for operations on tensors (like math operations)
  - or
  - Generating tensors (like variables and constants).
- Each ***node*** takes zero or more tensors as inputs and produces a tensor as an output.

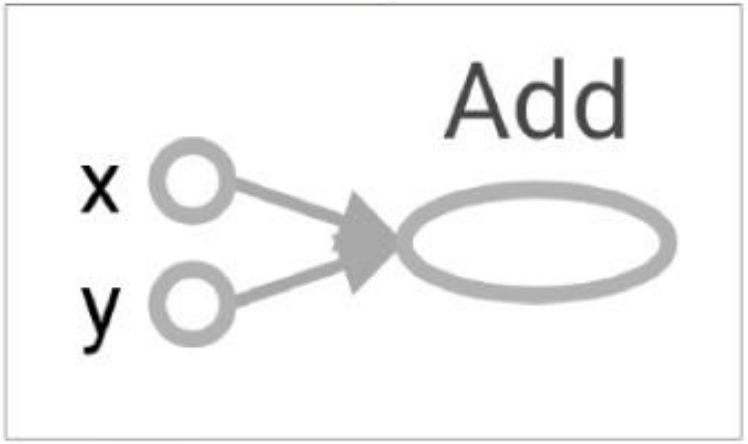
GRAPH (Cont.)

```
import tensorflow as tf
a = 2
b = 3
c = tf.add(a, b, name='Add')
print(c)
```

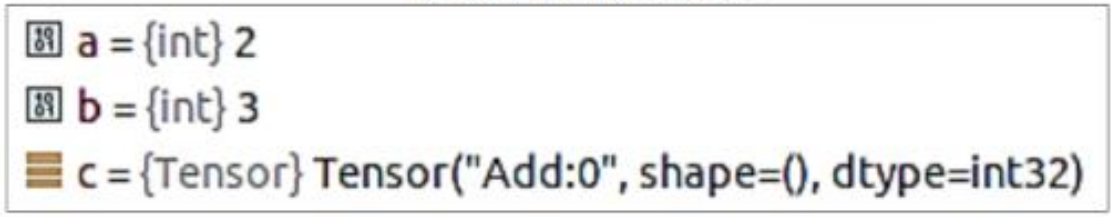
Scala

```
Tensor("Add:0", shape=(), dtype=int32)
```

Graph



Variables



GRAPH (Cont.)

- Therefore,
  - A TensorFlow ***Graph*** is something like a function definition in Python.
  - It ***WILL NOT*** do any computation.
  - It ***ONLY*** defines computation operations.

SESSION

- To compute anything, a graph must be launched in a ***session***.
- Technically, ***session*** places the graph ops on hardware such as CPUs or GPUs.
- ***Session*** provides methods to execute them.

```
sess = tf.Session()
print(sess.run(c))
sess.close()
```

```
5
```

SESSION (Cont.)

- with을 사용해서 **`close()`**를 별도로 하지 않기

파이썬 기본

```
x= 0
for i in range(5) :
    x = x+ 1
    print(x)
```

```
1
2
3
4
5
```

```
import tensorflow as tf

x = tf.Variable(0, name='x')

model = tf.global_variables_initializer()

with tf.Session() as session:
    for i in range(5):
        session.run(model)
        x = x + 1
        print(session.run(x))
```

```
1
2
3
4
5
```

SESSION (Cont.)

- Remember to close the session at the end of the session.

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

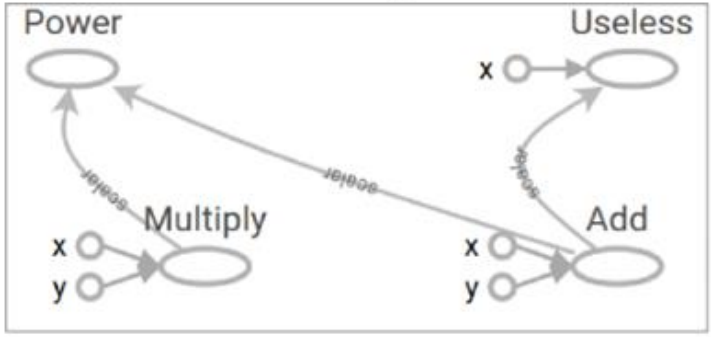
```
5
```

Example

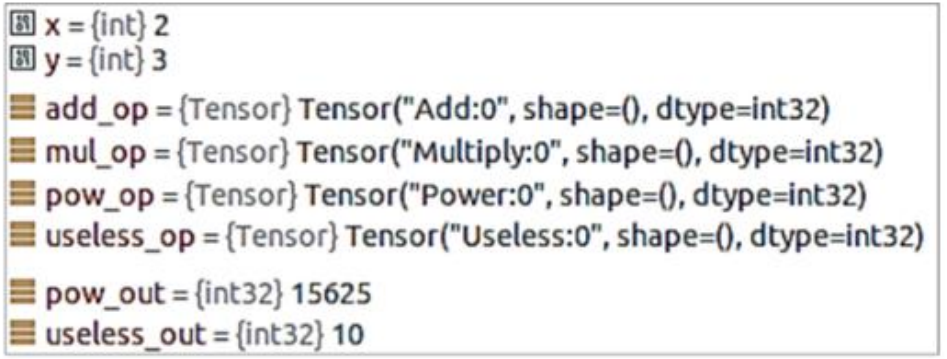
```
1  import tensorflow as tf
2  x = 2
3  y = 3
4  add_op = tf.add(x, y, name = 'Add')
5  mul_op = tf.multiply(x, y, name = 'Multiply')
6  pow_op = tf.pow(add_op, mul_op, name = 'Power')
7  useless_op = tf.multiply(x, add_op, name = 'Useless')
8
9  with tf.Session() as sess:
10     pow_out, useless_out = sess.run([pow_op, useless_op])
11
12 print(pow_out, useless_out)
```

```
15625  10
```

Graph

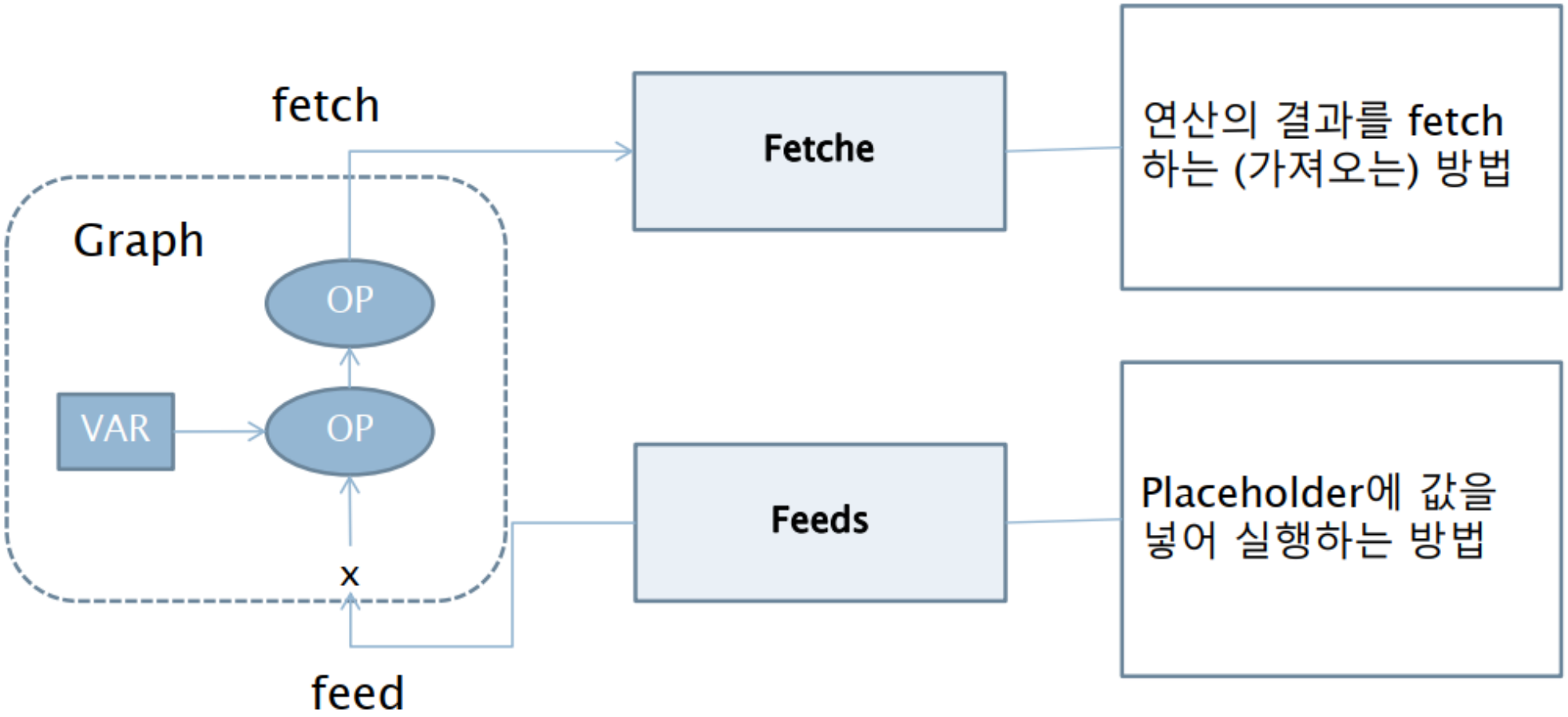


Variables



TenforFlow 실행 구조

- Session은 ***fetch***와 ***feed*** 2가지 방법으로 처리



fetch : 한 개 실행예시

- Tensor에 할당되어야 실제 Session에서 실행됨.

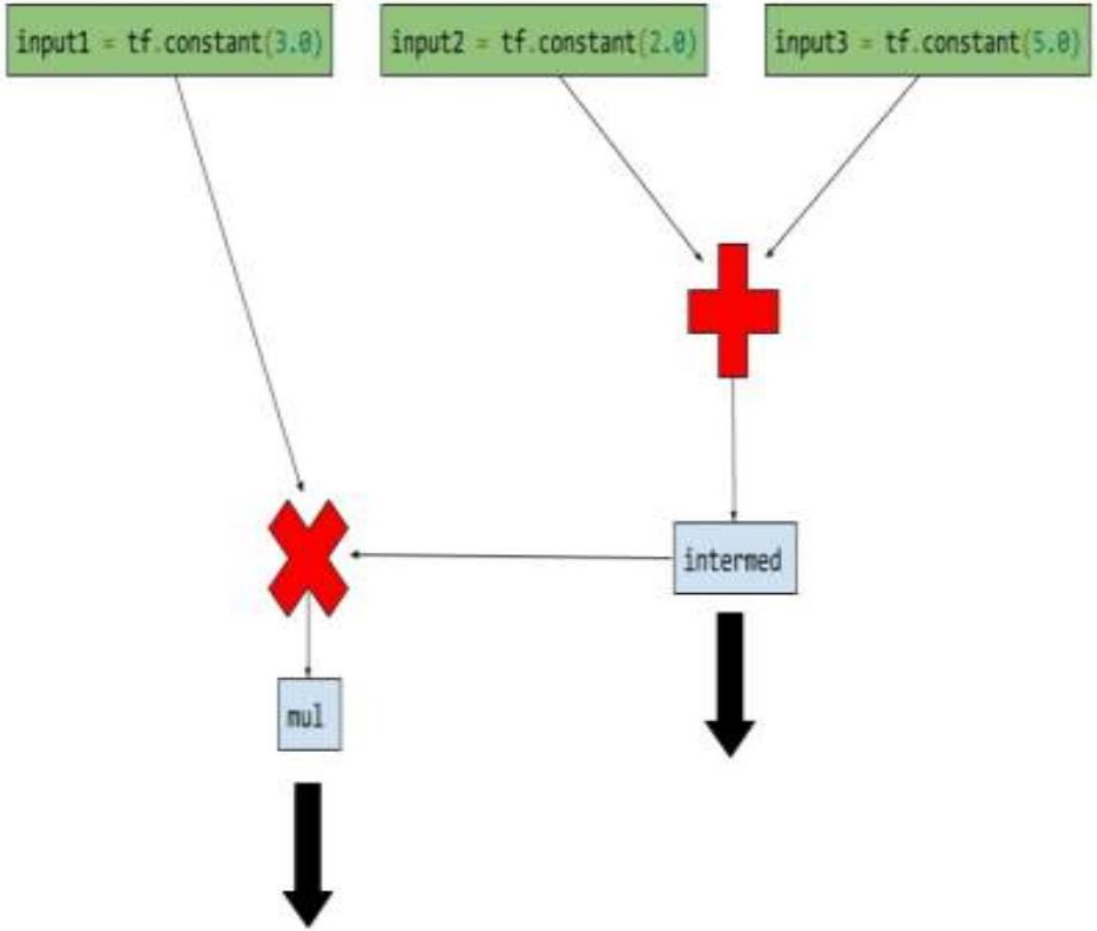

```
import tensorflow as tf

with tf.Session() as sess:
    print(tf.add(1,1).eval())
```

```
2
```

Tensor를 기준으로 실행하면 operation 영역을 실행해서 결과를 보여줌

fetch : 여러 개 실행 예시

- **`Session.run()`**에 list로 여러 개 실행되는 Tensor 처리




```
1  import tensorflow as tf
2
3  input1 = tf.constant(3.0)
4  input2 = tf.constant(2.0)
5  input3= tf.constant(5.0)
6  intermed = tf.add(input2, input3)
7  mul = tf.multiply(input1, intermed)
8  with tf.Session() as sess:
9      result = sess.run([mul, intermed])
10     print(result)
```

```
[21.0, 7.0]
```

TensorFlow feed 실행 예시

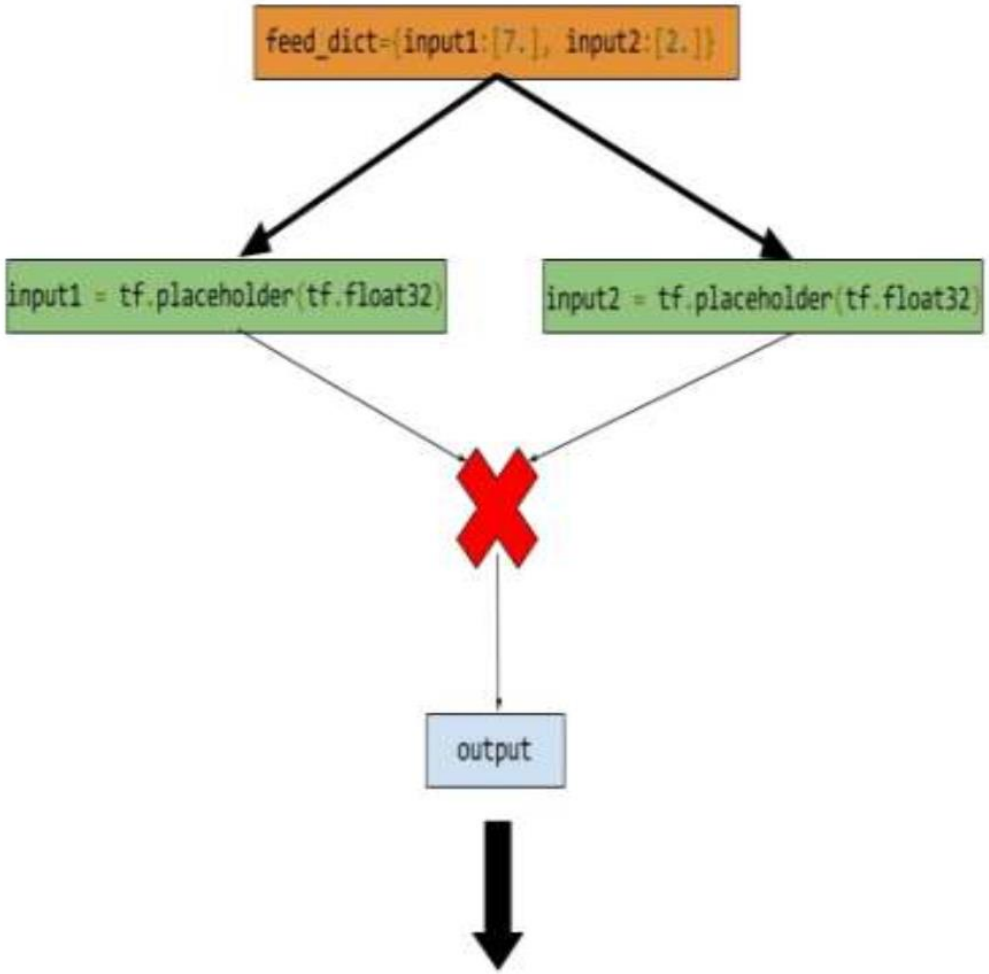
- Session은 ***feed***일 경우는 반드시 **feed_dict**로 처리 값을 할당해야 함.

```python
import tensorflow as tf

a = tf.placeholder('float')
b = tf.placeholder('float')

y = tf.multiply(a, b)
z = tf.add(y, y)

elems = tf.Variable([1.0, 2.0, 2.0, 2.0])

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # feed로 호출하기
    print(sess.run(y, feed_dict={a:3, b:3}))
    #placeholder를 다시 사용할 때는 재할당 필요
    print(sess.run(z, feed_dict={a:4, b:4}))
    # fetch로 호출하기
    print(sess.run(elems))
```

```
9.0
32.0
[1. 2. 2. 2.]
```

feed

- **`Session.run()`**에 list로 실행되는 Tensor 처리




```python
import tensorflow as tf

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.], input2:[2.]}))
```

```
[array([14.], dtype=float32)]
```

Tensor Types in TensorFlow

- TensorFlow does have its own data structure.
- TensorFlow programs use a tensor data structure to represent all data
- Only tensors are passed between operations in the computation graph.
- Commonly used in creating neural network models are namely **`Constant`**, **`Variable`**, and **`Placeholder`**.

Tensor Types in TensorFlow - Constant

- Is used as constants.
- Create a node
  - Takes value
  - Does not change.
- Can simply create a constant tensor using **`tf.constant`**.

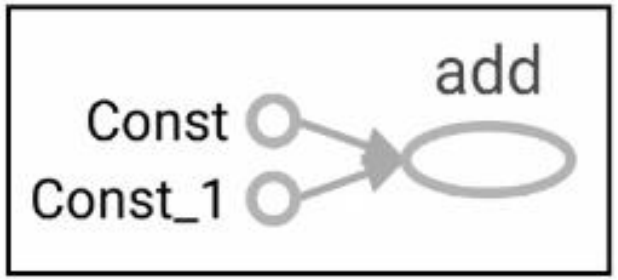
```
tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False)
```

Tensor Types in TensorFlow – Constant (Cont.)

```
# create graph
a = tf.constant(2)
b = tf.constant(3)
c = a + b
# launch the graph in a session
with tf.Session() as sess:
    print(sess.run(c))
```

```
5
```

Graph



Variables

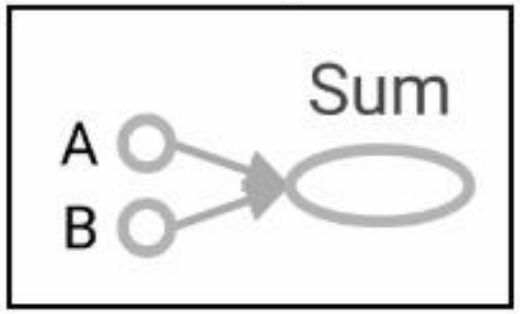
```
a = {Tensor} Tensor("Const:0", shape=(), dtype=int32)
b = {Tensor} Tensor("Const_1:0", shape=(), dtype=int32)
c = {Tensor} Tensor("add:0", shape=(), dtype=int32)
```

Tensor Types in TensorFlow – Constant (Cont.)

```
# create graph
a = tf.constant(2, name='A')
b = tf.constant(3, name='B')
c = tf.add(a, b, name='Sum')
# launch the graph in a session
with tf.Session() as sess:
    print(sess.run(c))
```

```
5
```

Graph



Variables

```
a = {Tensor} Tensor("A:0", shape=(), dtype=int32)
b = {Tensor} Tensor("B:0", shape=(), dtype=int32)
c = {Tensor} Tensor("Sum:0", shape=(), dtype=int32)
```

Tensor Types in TensorFlow – Constant (Cont.)

```
s = tf.constant(2.3, name='scalar', dtype=tf.float32)
m = tf.constant([[1, 2], [3, 4]], name='matrix')
# Launch the graph in a session
with tf.Session() as sess:
    print(sess.run(s))
    print(sess.run(m))
```

```
2.3
[[1 2]
 [3 4]]
```

Tensor Types in TensorFlow – Variable

- Is stateful nodes which output their current value.
- Graph를 최적화하는 용도로 TensorFlow가(좀 더 정확히는 학습 함수들이) 학습한 결과를 갱신하기 위해 사용하는 변수
- 이 변수의 값들이 신경망의 성능을 좌우하게 됨.
- By default, gradient updates (used in all neural networks) will apply to all variables in graph.
- In fact, variables are the things that want to tune in order to minimize the loss.
- Creating a variables is an operation.

Tensor Types in TensorFlow – Variable

Create Variables

- To create a variable, we should use **`tf.Variable`** as:

```
# Create a variable.
w = tf.Variable(<initial-value>, name=<optional-name>)
```

- Some examples of creating scalar and matrix variables are as follows:

```
s = tf.Variable(2, name="scalar")
m = tf.Variable([[1, 2], [3, 4]], name="matrix")
W = tf.Variable(tf.zeros([784,10]))
```

Tensor Types in TensorFlow – Variable

Create Variables

- Calling **`tf.Variable`** to create a variable is the old way of creating a variable.
- TensorFlow recommends to use the **`tf.get_variable`**.

```
tf.get_variable(name,
                shape=None,
                dtype=None,
                initializer=None,
                regularizer=None,
                trainable=True,
                collections=None,
                caching_device=None,
                partitioner=None,
                validate_shape=True,
                use_resource=None,
                custom_getter=None,
                constraint=None)
```

Tensor Types in TensorFlow – Variable

Create Variables

- Some examples are as follows:

```
s = tf.get_variable("scalar", initializer=tf.constant(2))
m = tf.get_variable("matrix", initializer=tf.constant([[0, 1], [2, 3]]))
W = tf.get_variable("weight_matrix", shape=(784, 10), initializer=tf.zeros_initial
izer())
```

Tensor Types in TensorFlow – Variable

Initialize Variables

- Just like most programming languages, Variables need to be initialized before being used.
- To initialize variables, have to :
  - Invoke ***a variable initializer*** operation
  - Run the operation on the session.

Tensor Types in TensorFlow – Variable

Initialize Variables

- Create two variables and add them together.

```
a = tf.get_variable(name="var_1", initializer=tf.constant(2))
b = tf.get_variable(name="var_2", initializer=tf.constant(3))
c = tf.add(a, b, name="Add1")

# launch the graph in a session
with tf.Session() as sess:
    # now let's evaluate their value
    print(sess.run(a))
    print(sess.run(b))
    print(sess.run(c))
```

- **`FailedPreconditionError`**: Attempting to use uninitialized value

Tensor Types in TensorFlow – Variable

Initialize Variables

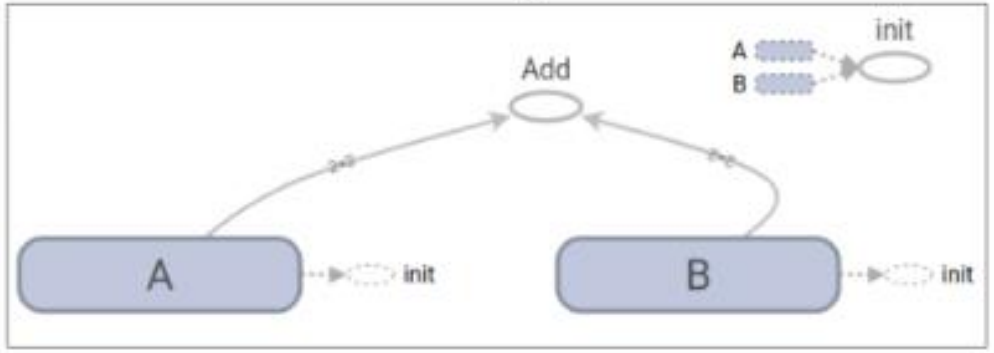
```
# create graph
a = tf.get_variable(name="A", initializer=tf.constant(2))
b = tf.get_variable(name="B", initializer=tf.constant(3))
c = tf.add(a, b, name="Add")
# add an Op to initialize global variables
init_op = tf.global_variables_initializer()

# launch the graph in a session
with tf.Session() as sess:
    # run the variable initializer operation
    sess.run(init_op)
    # now let's evaluate their value
    print(sess.run(a))
    print(sess.run(b))
    print(sess.run(c))
```

```
2
3
5
```

Tensor Types in TensorFlow – Variable

Initialize Variables

Graph



Variables

a = {Variable} <tf.Variable 'A:0' shape=(2, 2) dtype=int32_ref>
b = {Variable} <tf.Variable 'B:0' shape=(2, 2) dtype=int32_ref>
c = {Tensor} Tensor("Add:0", shape=(2, 2), dtype=int32)
init_op = {Operation} name: "init"\nop: "NoOp"\ninput: "^A/A

Tensor Types in TensorFlow – Variable

```
# create graph
weights = tf.get_variable(name="W", shape=[2,3], initializer=tf.truncated_normal_i
nitializer(stddev=0.01))
biases = tf.get_variable(name="b", shape=[3], initializer=tf.zeros_initializer())

# add an Op to initialize global variables
init_op = tf.global_variables_initializer()

# launch the graph in a session
with tf.Session() as sess:
    # run the variable initializer
    sess.run(init_op)
    # now we can run our operations
    W, b = sess.run([weights, biases])
    print('weights = {}'.format(W))
    print('biases = {}'.format(b))
```

```
weights = [[-0.00376599 -0.00506956  0.00082394]
 [ 0.0016487   0.00981423 -0.00226094]]
biases = [0. 0. 0.]
```

Tensor Types in TensorFlow – Placeholder

- Is simply a variable to assign data in a future time.
- Graph에 사용할 입력값을 나중에 받기 위해 사용하는 매개변수
- Is nodes whose value is fed in at execution time.

```
a = tf.placeholder(tf.float32, shape=[5])
b = tf.placeholder(dtype=tf.float32, shape=None, name=None)
X = tf.placeholder(tf.float32, shape=[None, 784], name='input')
Y = tf.placeholder(tf.float32, shape=[None, 10], name='label')
```

Tensor Types in TensorFlow – Placeholder (Cont.)

- Create a constant vector and a placeholder and add them together.

```
a = tf.constant([5, 5, 5], tf.float32, name='A')
b = tf.placeholder(tf.float32, shape=[3], name='B')
c = tf.add(a, b, name="Add")

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

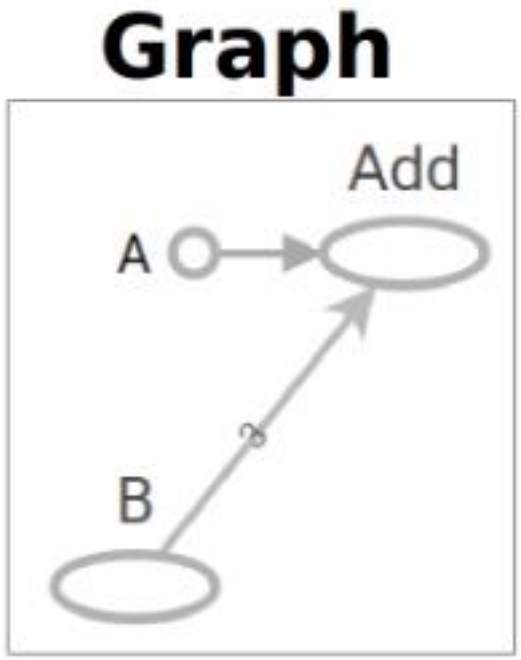
- **InvalidArgumentError**: You must feed a value for placeholder tensor 'B' with dtype float and shape [3] error

Tensor Types in TensorFlow – Placeholder (Cont.)

```
a = tf.constant([5, 5, 5], tf.float32, name='A')
b = tf.placeholder(tf.float32, shape=[3], name='B')
c = tf.add(a, b, name="Add")

with tf.Session() as sess:
    # create a dictionary:
    d = {b: [1, 2, 3]}
    # feed it to the placeholder
    print(sess.run(c, feed_dict=d))
```

```
[6. 7. 8.]
```

Tensor Types in TensorFlow – Placeholder (Cont.)



Variables

a	{Tensor} Tensor("A:0", shape=(3,), dtype=float32)
b	{Tensor} Tensor("B:0", shape=(3,), dtype=float32)
c	{Tensor} Tensor("Add:0", shape=(3,), dtype=float32)
d	{dict} {<tf.Tensor 'B:0' shape=(3,) dtype=float32>: [1, 2, 3]}

TensorFlow Strings

- Python3에서는 문자열(str) unicode가 기본이므로 str에서 encoding 처리를 통해 bytes → Unicode 변환

```
import tensorflow as tf
sess = tf.Session()
hello = tf.constant('Hello, TensorFlow')
print(sess.run(hello))
print(str(sess.run(hello),encoding="utf-8"))
```

```
b'Hello, TensorFlow'
Hello, TensorFlow
```

TensorFlow 빌딩 / 실행 구조

- TensorFlow는 빌딩구조와 실행구조에 대한 처리 순서

1. tensorflow 모듈을 가져 와서 tf 호출
2. x라는 상수 값을 만들고 숫자 값 35를 지정.
3. y라는 변수를 만들고 방정식 x + 5로 정의
4. global_variables_initializer로 변수를 초기화
5. 값을 계산하기 위한 세션 만들기
6. 4에서 만든 모델 실행
7. 변수 y 만 실행하고 현재 값을 출력

```
import tensorflow as tf

x = tf.constant(35, name='x')
y = tf.Variable(x + 5, name='y')

model = tf.global_variables_initializer()

with tf.Session() as session:
    session.run(model)
    print(session.run(y))
```

```
40
```

TensorFlow 빌딩 / 실행 구조

- TensorFlow는 빌딩구조와 실행구조가 분리
- Lazy Evaluation 방식
  - tensor와 tensor들이 연산들을 먼저 정의하여 Graph를 만들고, 그 후 필요할 때 연산을 실행하는 코드를 넣어 '원하는 시점'에 실제 연산을 수행하도록 한다.
  - 함수형 언어에서 많이 사용
  - 실제 계산은 C++로 구현한 Core Library에서 수행하므로 비록 Python code이지만, 매우 뛰어난 성능을 얻을 수 있다.
  - Model 구성과 실행을 분리하여 프로그램을 좀더 깔끔하게 작성.

TensorFlow 빌딩 / 실행 구조

- Graph의 실행은 Session안에서 이루어져야 하며, Session 객체와 **`run()`**을 사용

```
import tensorflow as tf
x2 = tf.linspace(-1.0, 1.0, 10)
print(x2)

g = tf.get_default_graph()
print([op.name for op in g.get_operations()])


sess = tf.Session()
print(sess.run(x2))
sess.close()
```

```
Tensor("LinSpace:0", shape=(10,), dtype=float32)
['x', 'LinSpace/start', 'LinSpace/stop', 'LinSpace/num', 'LinSpace']
[-1.         -0.77777779 -0.55555558 -0.33333331 -0.1111111   0.11111116
  0.33333337  0.55555558  0.77777779  1.        ]
```

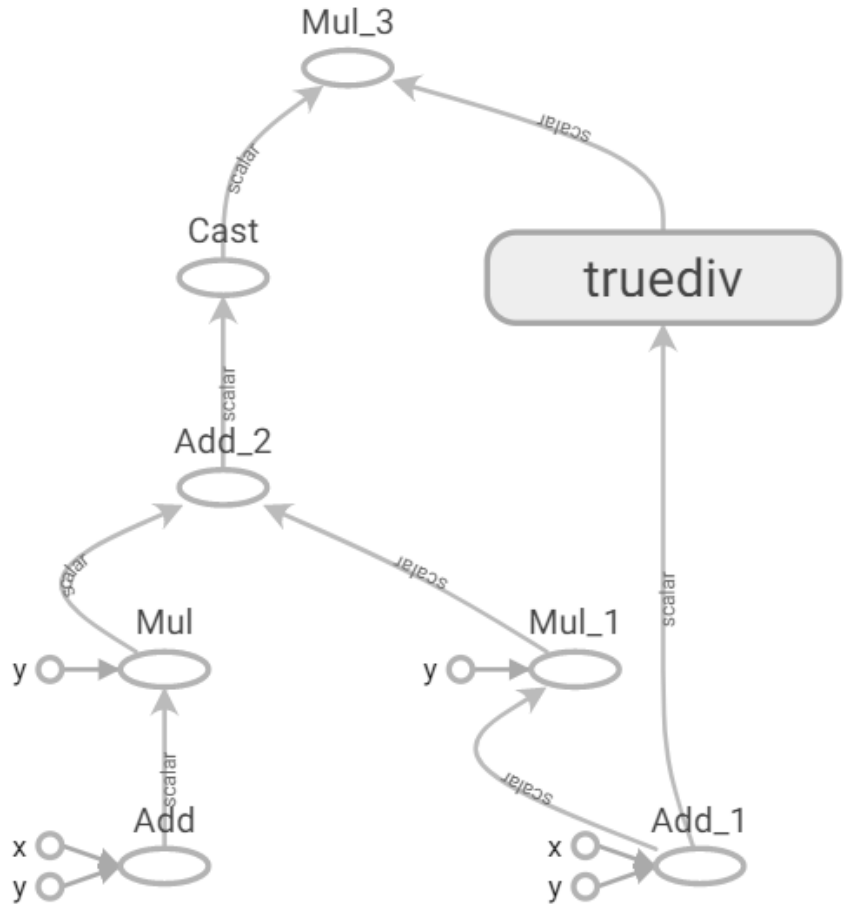
TensorBoard 실행하기

```
1  import tensorflow as tf
2  a = tf.add(1,2)
3  b = tf.multiply(a, 3)
4  c = tf.add(4, 5)
5  d = tf.multiply(c, 6)
6  e = tf.multiply(4, 5)
7  f = tf.math.divide(c, 6)
8  g = tf.add(b, d)
9  g = tf.cast(g, dtype=tf.float64)
10 h = tf.multiply(g, f)
11 with tf.Session() as sess:
12     writer = tf.summary.FileWriter(r'./graphs/sample', sess.graph)
13     print(sess.run(h))
14     writer.close()
```

94.5

TensorBoard 실행하기 (Cont.)

```
C:₩PythonHome>tensorboard --logdir="./graphs/sample" --port 6007
```



assert 처리하기

- **assert()**는 error가 발생할 때만 출력

```
1  assert 2 + 2 == 4, "Houston we've got a problem"
2  assert 2 + 2 == 3, "Houston we've got a problem"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-34-38818947d2dc> in <module>
      1 assert 2 + 2 == 4, "Houston we've got a problem"
----> 2 assert 2 + 2 == 3, "Houston we've got a problem"

AssertionError: Houston we've got a problem
```

assert 처리하기 (Cont.)

- **assert()**는 operation 처리이므로 조건에 맞아 error 없이 처리됨.

```python
import tensorflow as tf
#tf.assert_negative(x, data=None, summarize=None, message=None, name=None)

x = tf.constant([-2.25, -3.25])
y = tf.assert_negative(x)

#tf.assert_positive(x, data=None, summarize=None, message=None, name=None)
xp = tf.constant([2.25, 3.25])
yp = tf.assert_positive(xp)

with tf.Session() as sess:
    print('y ', type(y))
    print(y.run())
    print('yp ', type(yp))
    print(yp.run())
```

```
y  <class 'tensorflow.python.framework.ops.Operation'>
None
yp  <class 'tensorflow.python.framework.ops.Operation'>
None
```

assert 처리하기 (Cont.)

- **assert()**는 operation 처리이므로 조건에 맞지 않으면 error 처리됨.

```python
import tensorflow as tf
#tf.assert_negative(x, data=None, summarize=None, message=None, name=None)

x = tf.constant([-2.25, -3.25])
y = tf.assert_negative(x)

#tf.assert_positive(x, data=None, summarize=None, message=None, name=None)
xp = tf.constant([-2.25, 3.25])
yp = tf.assert_positive(xp)

with tf.Session() as sess:
    print('y ', type(y))
    print(y.run())
    print('yp ', type(yp))
    print(yp.run())
```

```
y  <class 'tensorflow.python.framework.ops.Operation'>
None
yp  <class 'tensorflow.python.framework.ops.Operation'>

---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩tensorflow₩python₩client₩session.py in _do_call(self, fn, *args)
   1355     try:
-> 1356       return fn(*args)
   1357     except errors.OpError as e:
```

TensorFlow vs NumPy

- TensorFlow의 연산에서 사실상 많은 함수들이 numpy와 거의 비슷한 이름을 사용하고 있다.
- 따라서 numpy를 잘 안다는 것은 TensorFlow 연산을 잘 안다는 뜻이 된다.
- 또한, 실제 TensorFlow 연산은 많은 부분이 numpy와 같이 이루어지고 있어서 TensorFlow를 잘 쓰기 위해서는 numpy 공부도 할 필요가 있다.
- 다음 표는 TensorFlow의 형식과 연산에 대한 유사성을 표로 표시한 것이다.
- 또한, 이들 연산이 모두 행렬연산이라는 데 중요한 공통성이 있다.

TensorFlow vs NumPy (Cont.)

NumPy	TensorFlow
ndarray	tensor
a = np.zeros((2,2)); b = np.ones((2,2))	a = tf.zeros((2,2)); b = tf.ones((2,2))
a.shape	a.get_shape()
np.reshap(a, (1,4))	tf.reshape(a, (1, 4))
numpy.sum(b, axis=1)	tf.reduce_sum(a, reduction_indices[1])
numpy.mean()	tf.reduce_mean()
numpy.subtract()	tf.subtract()
numpy.sqrt()	tf.sqrt()
numpy.argmax()	tf.math.argmax()
numpy.argmin()	tf.math.argmin()
np.dot(a, b)	tf.matmul(a, b)

TensorFlow vs NumPy (Cont.)

- 행렬에 대한 열축(reduction_indices = 1) 합산

```
import numpy as np

a = np.zeros((2,2))
print(a)
b = np.ones((2,2))
print(b)
# 열단위로 합산
print( np.sum(b, axis=1))

print(a.shape)
print(np.reshape(a, (1,4)))
```

```
[[ 0.  0.]
 [ 0.  0.]]
[[ 1.  1.]
 [ 1.  1.]]
[ 2.  2.]
(2, 2)
[[ 0.  0.  0.  0.]]
```

```
import tensorflow as tf

sess = tf.InteractiveSession()
a = tf.zeros((2,2))
b = tf.ones((2,2))
print(tf.reduce_sum(b, reduction_indices=1).eval())

print(a.get_shape())
print(tf.reshape(a, (1, 4)).eval())

sess.close()
```

```
[ 2.  2.]
(2, 2)
[[ 0.  0.  0.  0.]]
```

TensorFlow vs NumPy (Cont.)

- TensorFlow는 작성과 실행영역이 분리 되어 있음.

Python

```
c = 1
v = c + 10
print(v)
```

```
11
```

TensorFlow

```
import tensorflow as tf

c = tf.constant(1)
c10 = tf.constant(10)
v = tf.add(c,c10)

sess = tf.Session()

print(sess.run(v))

sess.close()
```

```
11
```

```
import tensorflow as tf

c = tf.constant(1)
c10 = tf.constant(10)
v = tf.Variable(0)
ca = tf.add(c,c10)
va = tf.add(v,ca)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
print(sess.run(va))

sess.close()
```

```
11
```

출력 지정하기

- Session에서 실행된 결과를 출력 format에 맞춰서 출력하기

```python
import tensorflow as tf

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
bias = tf.Variable(1.0)
y_pred = x ** 2 + bias      # x -> x^2 + bias
loss = (y - y_pred)**2      # l2 loss?

with tf.Session() as session:
    #변수 초기화
    session.run(tf.global_variables_initializer())

    # OK, print 1.000 = (3**2 + 1 - 9)**2
    print('Loss(x,y) = %.3f' % session.run(loss, {x: 3.0, y: 9.0}))
    # OK, print 10.000; for evaluating y_pred only, input to y is not required
    print('pred_y(x) = %.3f' % session.run(y_pred, {x: 3.0}))
    # OK, print 1.000 bias evaluates to 1.0
    print('bias      = %.3f' % session.run(bias))
```

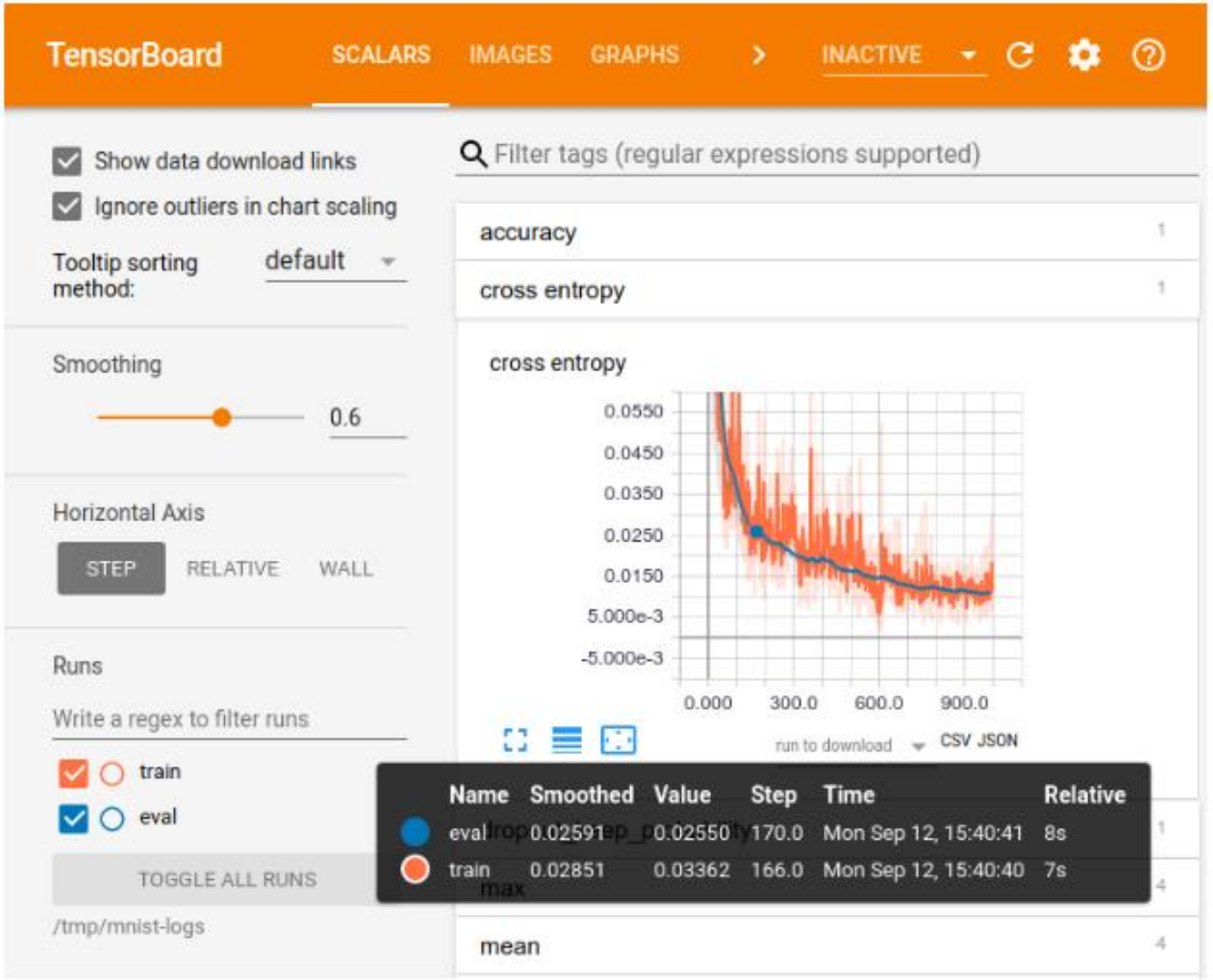
```
Loss(x,y) = 1.000
pred_y(x) = 10.000
bias      = 1.000
```

Introduction to TensorBoard

- Is a visualization software
- Comes with any standard TensorFlow installation.
- In Google's words:

"*The computations you'll use TensorFlow for (like training a massive deep neural network) can be complex and confusing. To make it easier to understand, debug, and optimize TensorFlow programs, we've included a suite of visualization tools called TensorBoard.*"

Introduction to TensorBoard (Cont.)



Refer to https://www.easy-tensorflow.com/tf-tutorials/basics/introduction-to-tensorboard

Introduction to TensorBoard (Cont.)

- Was created as a way to help understand the flow of tensors in model → Can debug and optimize it.
- It is generally used for two main purposes:
  - Visualizing the Graph
  - Writing Summaries to Visualize Learning

Introduction to TensorBoard (Cont.)

Visualizing the Graph

- Need to add a very few lines of code to it.
- This will export the TensorFlow operations into a file, called ***event file*** (or event log file).
- TensorBoard is able to read this file and give insight into the model graph and its performance.

Introduction to TensorBoard (Cont.)

Visualizing the Graph

```
import tensorflow as tf

# create graph
a = tf.constant(2)
b = tf.constant(3)
c = tf.add(a, b)
# launch the graph in a session
with tf.Session() as sess:
    print(sess.run(c))
```

```
5
```

Introduction to TensorBoard (Cont.)

Visualizing the Graph

- To visualize the program with TensorBoard, need to write log files of the program.
- To write event files, we first need to create a ***writer*** for those logs, using this code:

```
writer = tf.summary.FileWriter([logdir], [graph])
```

Introduction to TensorBoard (Cont.)

Visualizing the Graph

- `[logdir]`
  - Is the folder where want to store those log files.
  - Can choose `[logdir]` to be something meaningful such as './graphs'.
- `[graph]`
  - Is the graph of the program we're working on.

Introduction to TensorBoard (Cont.)

Visualizing the Graph

- There are two ways to get the graph:
- Call the graph using **`tf.get_default_graph()`**
  - Returns the default graph of the program
- Set it as **`sess.graph`**
  - Returns the session's graph (note that this requires us to already have created a session).

Introduction to TensorBoard (Cont.)

Visualizing the Graph

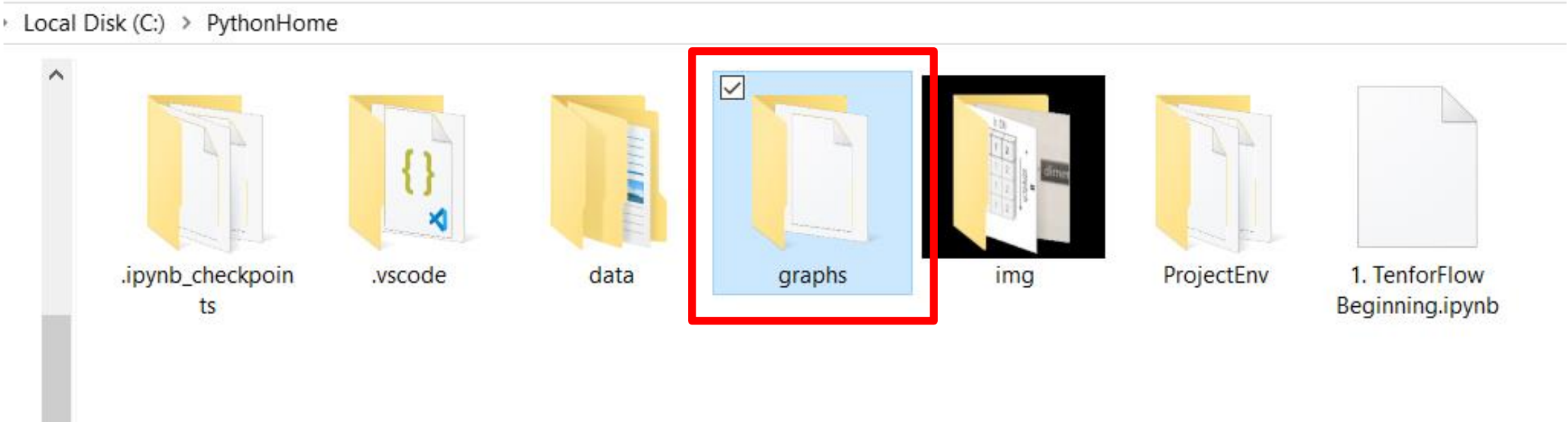
```
import tensorflow as tf
tf.reset_default_graph()   # To clear the defined variables and operations of the previous cell

# create graph
a = tf.constant(2)
b = tf.constant(3)
c = tf.add(a, b)

# creating the writer out of the session
# writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())

# launch the graph in a session
with tf.Session() as sess:
    # or creating the writer inside the session
    writer = tf.summary.FileWriter('./graphs', sess.graph)
    print(sess.run(c))
```

```
5
```

Introduction to TensorBoard (Cont.)

Visualizing the Graph

- Now if run this code, it creates a directory inside your current directory (beside your Python code) which contains the ***event file***.



Introduction to TensorBoard (Cont.)

Visualizing the Graph

- Next, go to Terminal and make sure that the present working directory.

```
$ cd ~/PythonHome/tensorboard
```

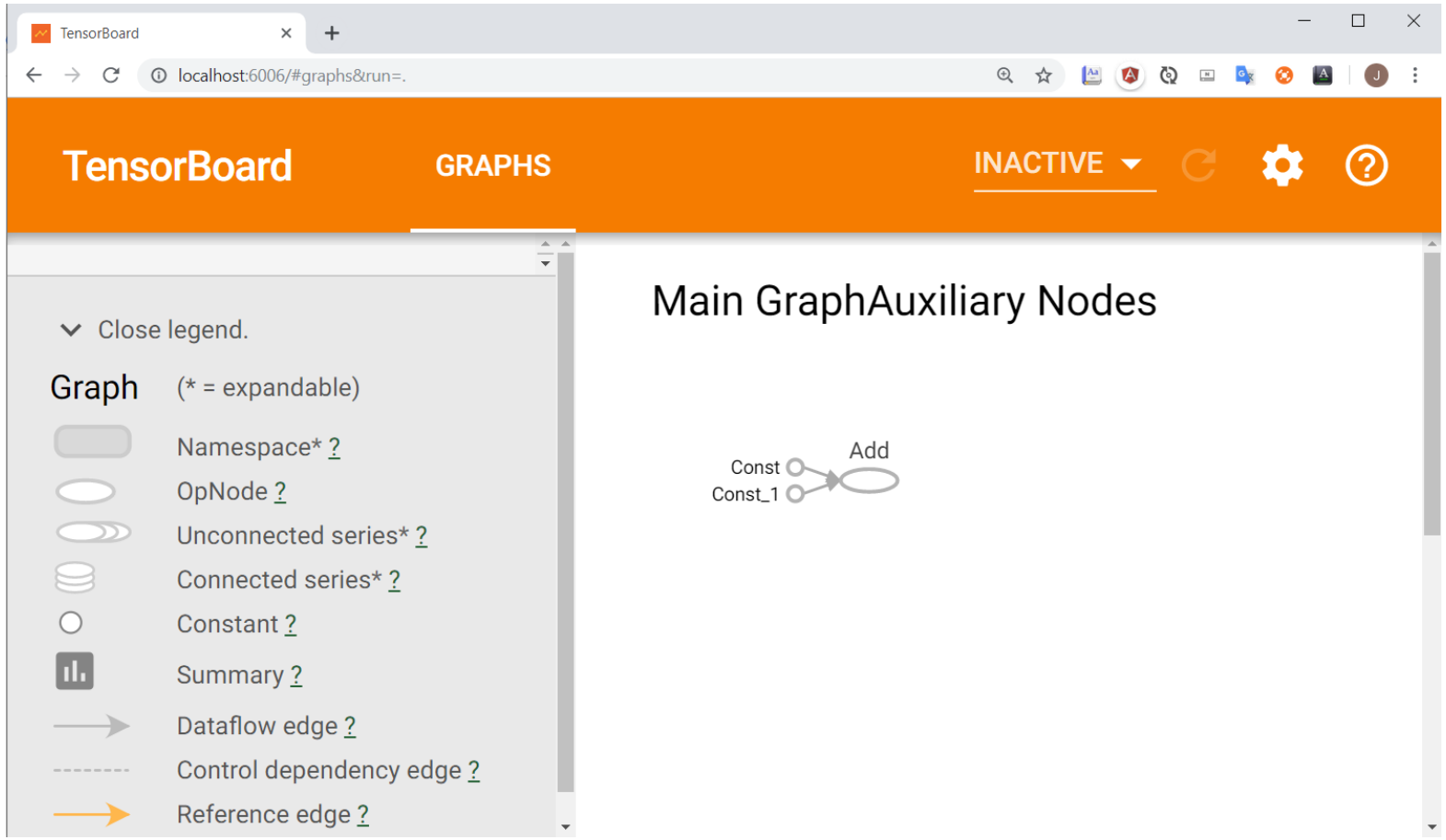
- Then run:

```
$ tensorboard --logdir="./graphs" --port
6006
```

Introduction to TensorBoard (Cont.)

Visualizing the Graph

- http://localhost:6006/



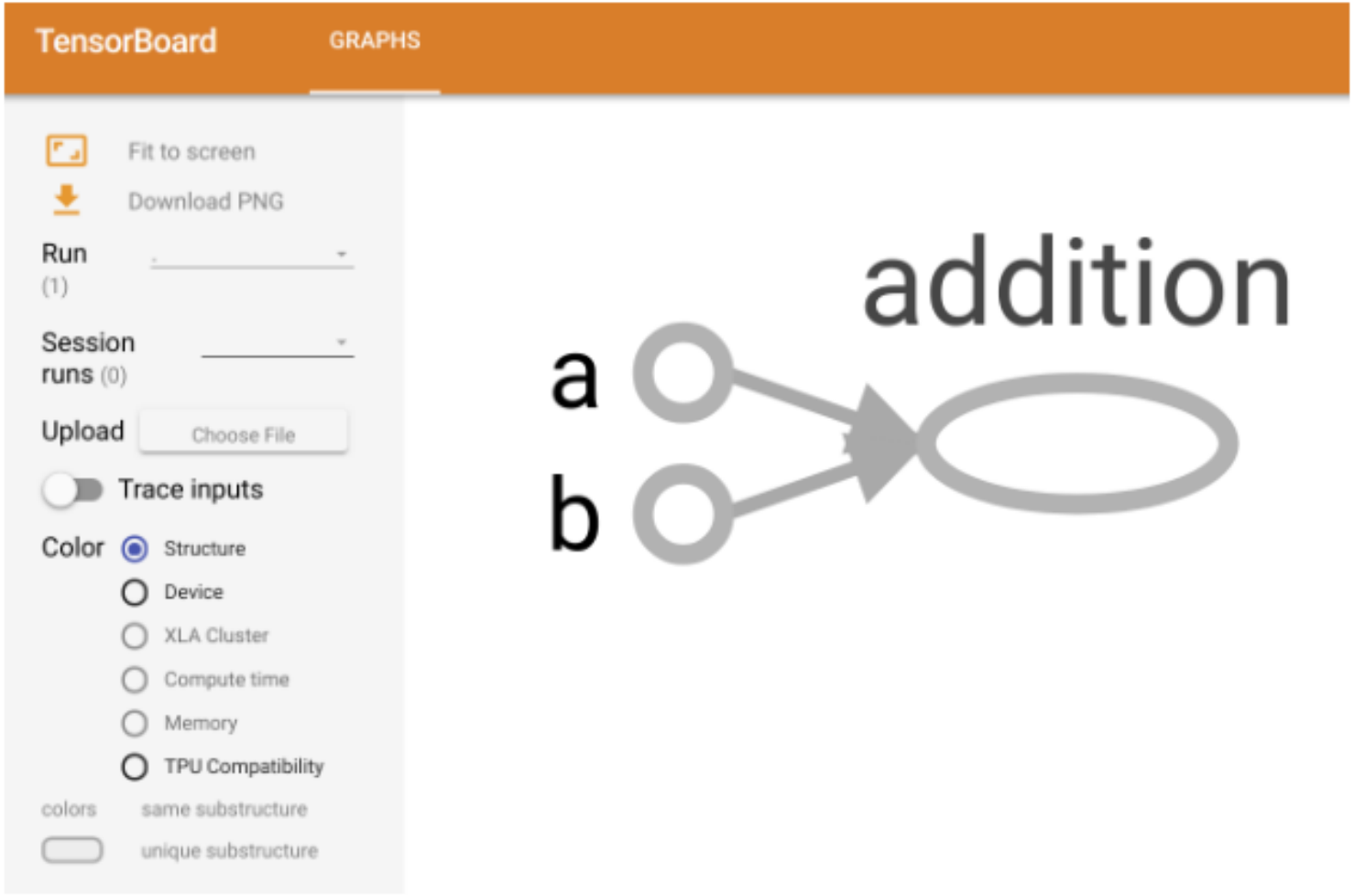
Introduction to TensorBoard (Cont.)

Visualizing the Graph

```
import tensorflow as tf
tf.reset_default_graph()   # To clear the defined variables and operations of the
previous cell

# create graph
a = tf.constant(2, name="a")
b = tf.constant(3, name="b")
c = tf.add(a, b, name="addition")

# creating the writer out of the session
# writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())

# launch the graph in a session
with tf.Session() as sess:
    # or creating the writer inside the session
    writer = tf.summary.FileWriter('./graphs', sess.graph)
    print(sess.run(c))
```

Introduction to TensorBoard (Cont.)

Visualizing the Graph



Introduction to TensorBoard (Cont.)

Writing Summaries to Visualize Learning

- A special operation called ***summary*** to visualize the model parameters (like weights and biases of a neural network), metrics (like loss or accuracy value), and images (like input images to a network).
- ***Summary*** is a special TensorBoard operation
- Takes in a regular tenor
- Outputs the summarized data to your disk (i.e. in the ***event file***).

Introduction to TensorBoard (Cont.)

Writing Summaries to Visualize Learning

- There are three main types of summaries:
- **`tf.summary.scalar`**
  - Used to write a single scalar-valued tensor (like classificaion loss or accuracy value)
- **`tf.summary.histogram`**
  - Used to plot histogram of all the values of a non-scalar tensor (like weight or bias matrices of a neural network)
- **`tf.summary.image`**
  - Used to plot images (like input images of a network, or generated output images of an autoencoder or a GAN)

Introduction to TensorBoard (Cont.)

Writing Summaries to Visualize Learning : tf.summary.scalar

- Writing the values of a scalar tensor that changes over time or iterations.
- In the case of neural networks (say a simple network for classification task), it's usually used to monitor the changes of loss function or classification accuracy.

Linear Regression

선형 회귀(Linear Regression)

- 시험 공부하는 시간을 늘리면 늘릴 수록 성적이 잘 나온다. 하루에 걷는 횟수를 늘릴 수록 몸무게는 줄어든다. 집의 평수가 클수록 집의 매매 가격은 비싼 경향이 있다.
- 위의 경우는 수학적으로 생각해보면 **어떤 요인의 수치에 따라서 특정 요인의 수치가 영향을 받고 있다**고 말할 수 있다.
- 즉, **어떤 변수의 값에 따라서 특정 변수의 값이 영향을 받고 있다**고 볼 수 있다.
- 다른 변수의 값을 변하게 하는 변수를 x, 변수 x에 의해서 값이 종속적으로 변하는 변수 y라고 해보자.

선형 회귀(Linear Regression) (Cont.)

- 이때 변수 x의 값은 독립적으로 변할 수 있는 것에 반해, y값은 계속해서 x의 값에 의해서, 종속적으로 결정되므로 x를 독립 변수, y를 종속 변수라고 한다.
- 앞의 예에서 시험 공부 시간은 독립 변수 x, 성적은 종속 변수 y라고 할 수 있다.
- 선형 회귀는 종속 변수 y와 한 개 이상의 독립 변수 x와의 선형 관계를 모델링하는 분석 기법이다.

단순 선형 회귀 분석(Simple Linear Regression Analysis)

- 아래의 수식은 단순 선형 회귀의 수식이다.

$$y = Wx + b$$

- 여기서 독립 변수 x와 곱해지는 값 W를 Machine Learning에서는 가중치(Weight), 별도로 더해지는 값 b를 편향(Bias)이라고 한다.
- 직선 방정식에서는 각각 직선의 기울기와 절편을 의미. W와 b가 왜 필요할까요? W와 b가 없이 y와 x란 수식은 y는 x와 같다는 하나의 식밖에 표현하지 못합니다.

단순 선형 회귀 분석(Simple Linear Regression Analysis) (Cont.)

- W와 b가 왜 필요할까?
  - W와 b가 없이 y와 x란 수식은 y는 x와 같다는 하나의 식밖에 표현하지 못하기 때문이다.

$$y = x$$

  - 여러 다양한 직선을 표현하기 위해서는 W와 b가 필요하다.
  - W와 b가 x와 y의 관계를 모델링하는 값이다.
  - W와 b의 값을 엉망으로 찾으면 x와 y의 관계를 찾아내지 못한다.
  - 반대로 W와 b의 값을 제대로 찾으면 x와 y의 관계를 제대로 찾아내게 된다.

다중 선형 회귀 분석(Multiple Linear Regression Analysis)

- 가설 :
  - 집의 매매 가격은 단순히 집의 평수가 크다고 결정되는 게 아니다.
  - 집의 층의 수, 방의 개수, 지하철 역과의 거리와도 영향이 있는 것 같다.
  - 즉, 다수의 요소를 가지고 집의 매매 가격을 예측해본다면 보고 싶다면.
- y는 여전히 1개이지만 이제 x는 1개가 아니라 여러 개가 된다.
- 이것을 다중 선형 회귀 분석이라고 한다.

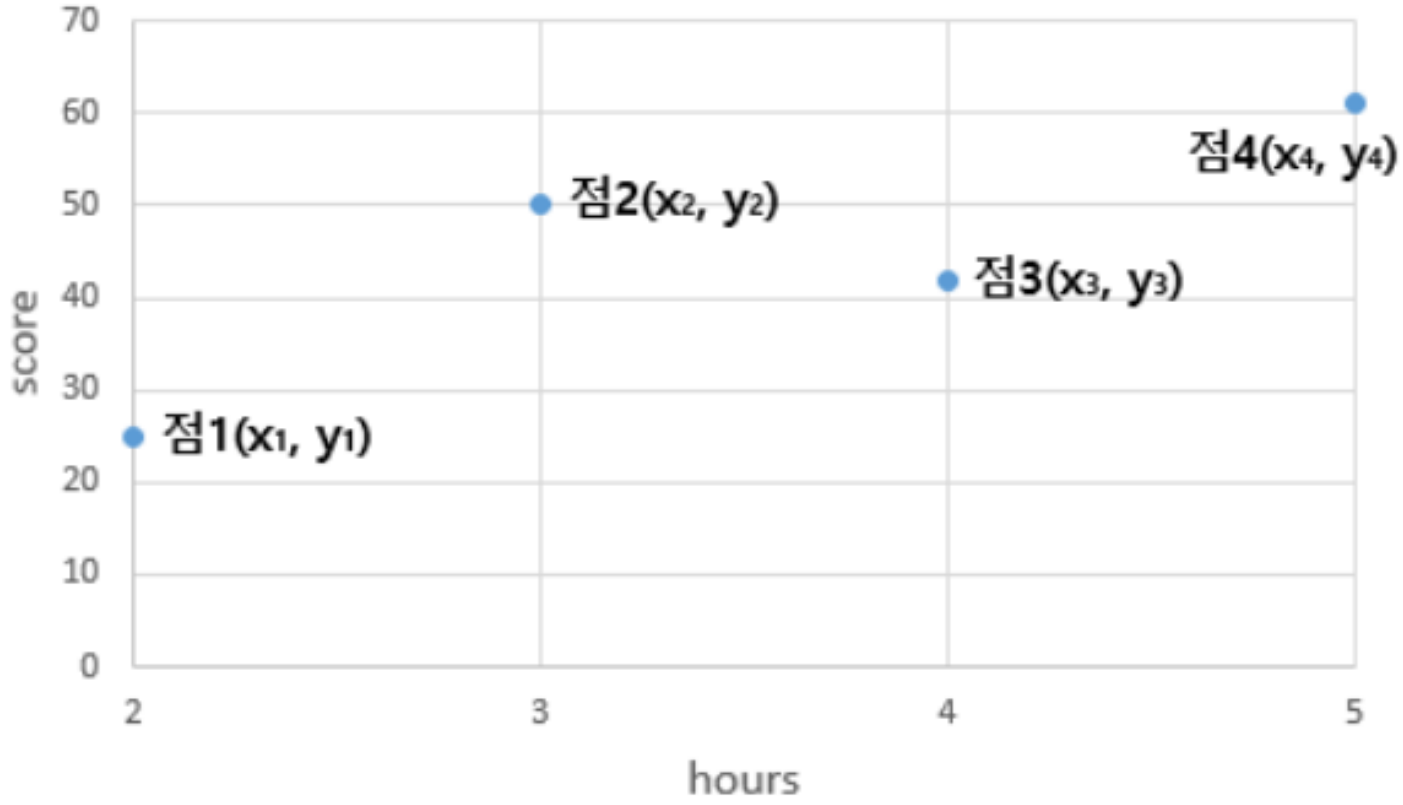
가설(Hypothesis) 세우기

- 어떤 학생이 공부 시간에 따라 다음과 같은 점수를 얻었다는 데이터가 있다.
- 공부시간이 x라면, 점수는 y이다.

hours(x)	score(y)
2	25
3	50
4	42
5	61

가설(Hypothesis) 세우기 (Cont.)

- 좌표 평면에 그려보자.
- 단, 아래의 그래프에서 x와 y에 붙은 숫자는 서로 다른 독립 변수와 종속 변수를 의미하는 것이 아니라, 단순 선형 회귀에서의 서로 다른 값을 의미한다.

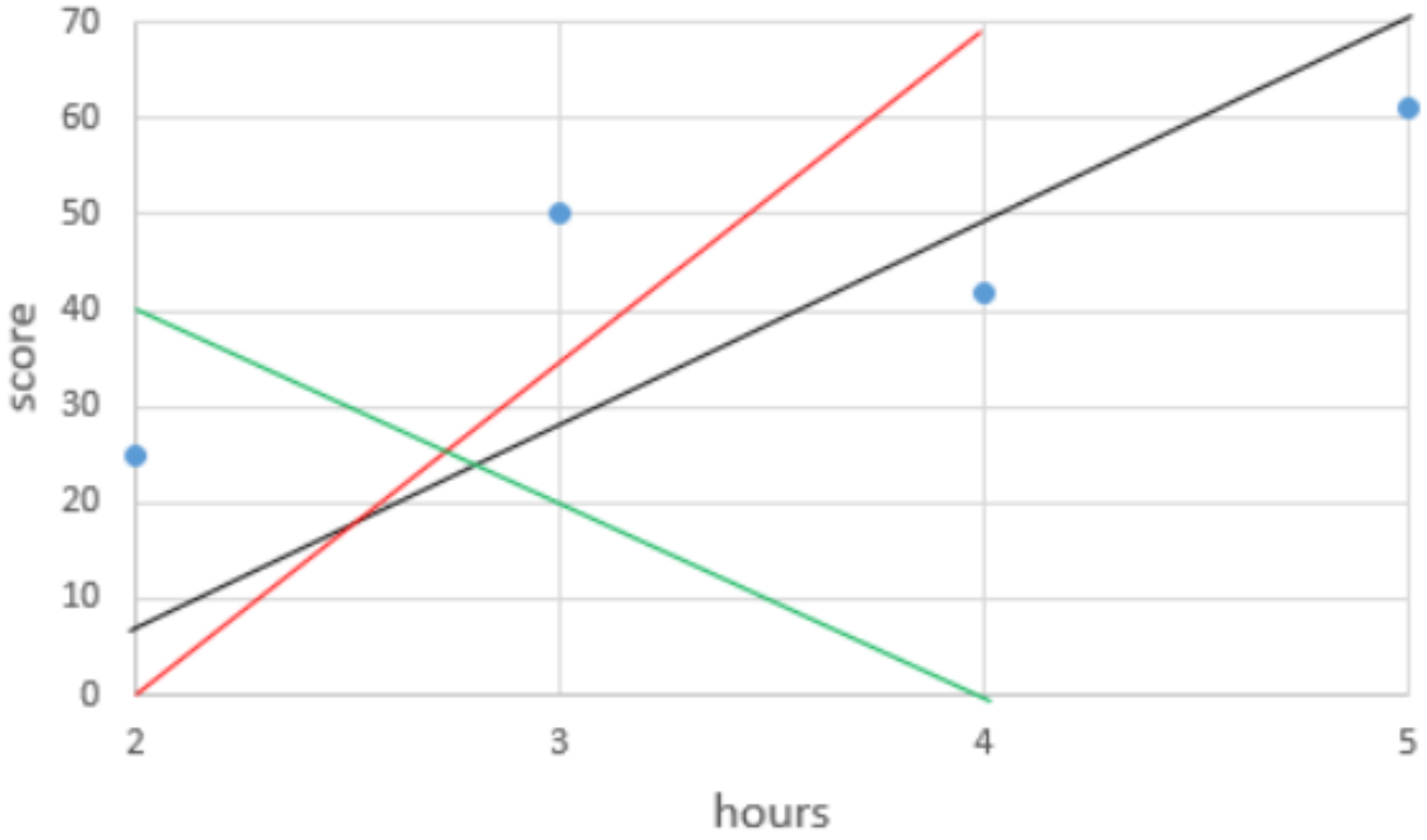


가설(Hypothesis) 세우기 (Cont.)

- 이미 제시된 데이터로부터 x와 y의 관계를 유추하고, 이 학생이 6시간을 공부하였을 때의 성적, 그리고 7시간, 8시간을 공부하였을 때의 성적을 예측해보고 싶다.
- x와 y의 관계를 유추하기 위해서 수학적으로 식을 세워보게 되는데 이 때 머신 러닝에서는 y와 x간의 관계를 유추한 식을 가설(Hypothesis)이라고 한다.
- 아래의 H(x)에서 H는 Hypothesis를 의미한다.

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

가설(Hypothesis) 세우기 (Cont.)



가설(Hypothesis) 세우기 (Cont.)

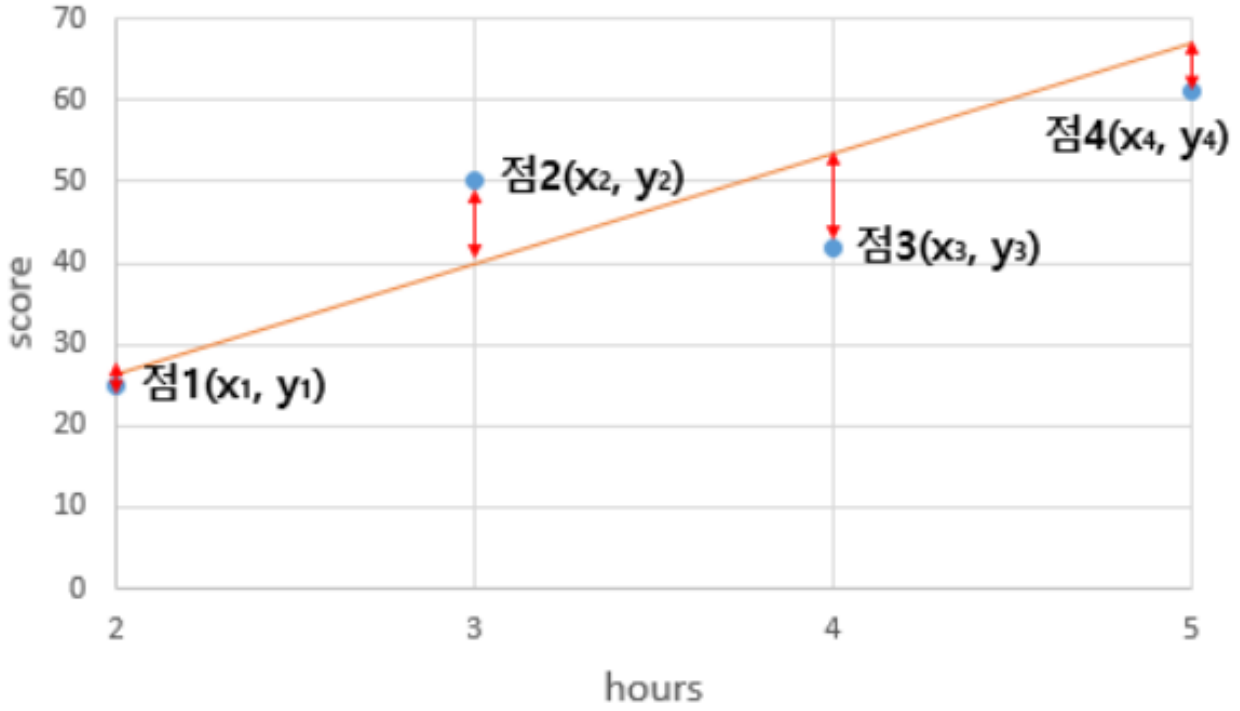
- 앞의 그림은 W와 b의 값에 따라서 천차만별로 그려지는 직선의 모습을 보여준다.
- 결국 선형 회귀는 주어진 데이터로부터 y와 x의 관계를 가장 잘 나타내는 직선을 그리는 일을 말한다.
- 그리고 어떤 직선인지 결정하는 것은 W와 b의 값이므로 선형 회귀에서 해야 할 일은 결국 적절한 W와 b를 찾아내는 일이다.
- 그렇다면, 어떻게 적절한 W와 b를 찾을 수 있을까?

비용 함수(Cost function) : 평균 제곱 오차(MSE)

- Machine Learning은 W와 b를 찾기 위해서 **실제값과 가설로부터 얻은 예측값의 오차를 계산하는 식을 세우고, 이 식의 값을 최소화하는 최적의 W와 b를 찾아낸다**.
- 이 때 실제값과 예측값에 대한 오차에 대한 식을 목적 함수(Objective function) 또는 비용 함수(Cost function) 또는 손실 함수(Loss function)라고 한다.
- 목적 함수(Objective Function)
  - 함수의 값을 최소화하거나, 최대화하거나 하는 목적을 가진 함수
- 비용함수(Cost Function) or 손실함수(Loss Function)
  - 그 값을 최소화하려는 함수.

비용 함수(Cost function) : 평균 제곱 오차(MSE) (Cont.)

- 비용 함수는 단순히 실제값과 예측값에 대한 오차를 표현하면 되는 것이 아니라, 예측값의 오차를 줄이는 일에 최적화 된 식이어야 한다.
- 주로 평균 제곱 오차(Mean Squared Error, MSE)가 사용



비용 함수(Cost function) : 평균 제곱 오차(MSE) (Cont.)

- 처음에는 랜덤으로 선을 그린다.
- 이제 이 직선으로부터 서서히 W와 b의 값을 바꾸면서 정답인 직선을 찾아나간다.
- 사실 y와 x의 관계를 가장 잘 나타내는 직선을 그린다는 것은 앞의 그림에서 모든 점들과 위치적으로 가장 가까운 직선을 그린다는 것과 같은 의미이다.
- 오차는 주어진 데이터에서 각 x에서의 실제값 y와 앞의 직선에서 예측하고 있는 H(x)값의 차이를 말한다.
- 앞의 그림에서 ↕는 각 점에서의 오차의 크기를 보여준다.
- 이 오차를 줄여가면서 W와 b의 값을 찾아내기 위해서는 오차의 크기를 측정할 방법이 필요하게 된다.

비용 함수(Cost function) : 평균 제곱 오차(MSE) (Cont.)

- 오차의 크기를 측정하기 위한 가장 기본적인 방법은 각 오차를 모두 더하는 방법이 있다.
- 위의 y = 13x + 1 직선이 예측한 예측값을 각각 실제값으로부터 오차를 계산하여 표를 만들어보면 아래와 같다.

hours(x)	**2**	**3**	**4**	**5**
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

비용 함수(Cost function) : 평균 제곱 오차(MSE) (Cont.)

- 그런데, 수식적으로 단순히 실제값 - 예측값을 수행하면 오차값이 음수가 나오는 경우가 생길 수 있다.
- 이럴 경우, 오차를 모두 더하면 제대로 된 오차의 크기를 측정할 수 없기 때문에, 보통 오차의 크기를 측정하기 위해서 모든 오차를 제곱하여 더하는 방법을 사용한다.
- 앞의 그림에서의 모든 점과 직선 사이의 ↕ 거리를 제곱하고 모두 더한다.
- 이를 수식으로 표현하면 아래와 같습니다.
- 단, 여기서 n은 갖고 있는 데이터의 개수를 의미합니다.

$$\sum_{i}^{n} [y_i - H(x_i)]^2$$

비용 함수(Cost function) : 평균 제곱 오차(MSE) (Cont.)

- 이 수식을 실제로 계산하면 각 오차를 제곱하여 더하면 되므로 4 + 100 + 49 + 25 = 178이 된다.
- 이때 데이터의 개수인 n으로 나누면, 오차의 제곱합에 대한 평균을 구할 수 있는데 이를 평균 제곱 오차(Mean Squared Error, MSE)라고 한다.

$$\frac{1}{n}\sum_{i}^{n}\left[y_i - H(x_i)\right]^2$$

비용 함수(Cost function) : 평균 제곱 오차(MSE) (Cont.)

- 이를 실제로 계산하면 178을 4로 나눈 값인 44.5가 된다.
- 이는 y = 13x + 1의 예측값과 실제값의 평균 제곱 오차의 값이 44.5임을 의미하는 것이다.
- 평균 제곱 오차는 적절한 W와 b를 찾기 위한 최적화된 식이다.
- 평균 제곱 오차를 W와 b에 의한 비용 함수(Cost function)로 재정의 해보면 다음과 같다.

$$cost(W, b) = \frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$

비용 함수(Cost function) : 평균 제곱 오차(MSE) (Cont.)

- 모든 점들과의 오차가 클 수록 평균 제곱 오차는 커지며, 오차가 작아질 수록 평균 제곱 오차는 작아진다.
- 그러므로 이 평균 최곱 오차. 즉, Cost(W,b)를 최소가 되게 만드는 W와 b를 구하면 결과적으로 y와 x의 관계를 가장 잘 나타내는 직선을 그릴 수 있게 된다.

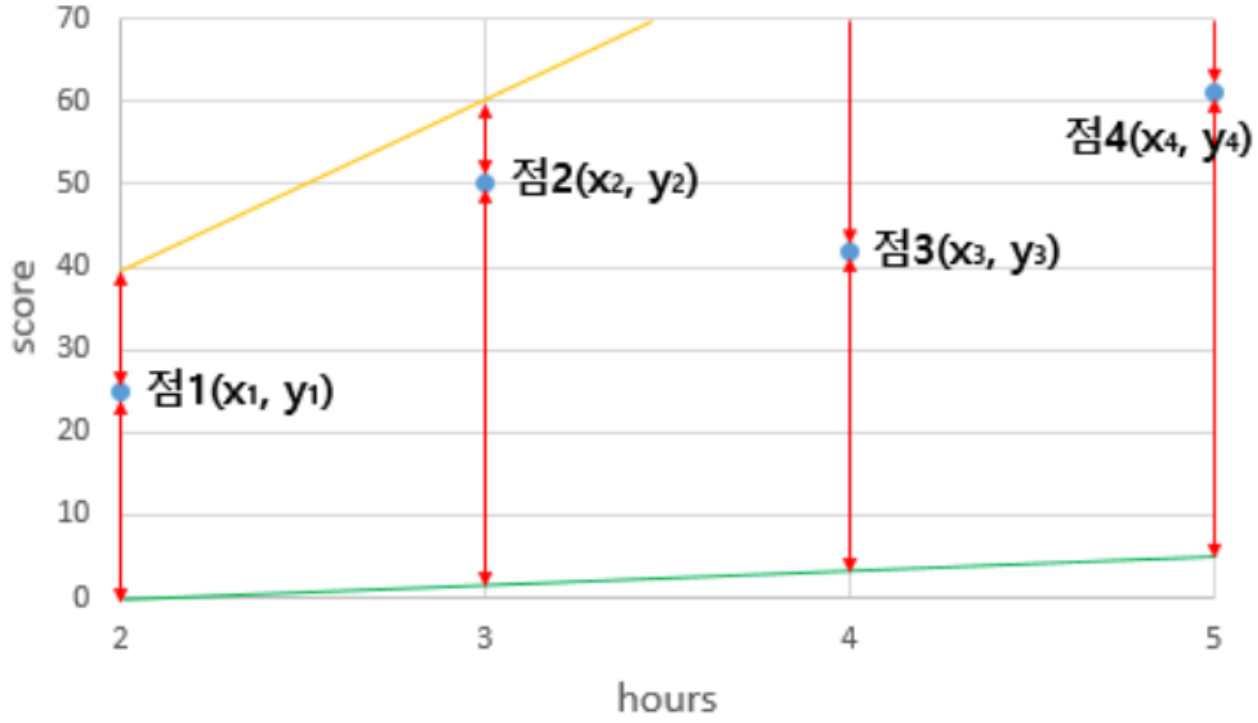
$$W, b \rightarrow minimize\ cost(W, b)$$

옵티마이저(Optimizer) : 경사하강법(Gradient Descent)

- 선형 회귀를 포함한 수많은 ML, DL의 학습은 결국 비용 함수를 최소화하는 매개 변수인 W와 b을 찾기 위한 작업을 수행한다.
- 이때 사용되는 것이 옵티마이저(Optimizer) Algorithm이다.
- 최적화 알고리즘이라고도 부른다.
- 그리고 이 Optimizer Algorithm을 통해 적절한 W와 b를 찾아내는 과정을 ML에서는 학습(Taining)이라고 부른다.
- 여기서는 가장 기본적인 Optimizer Algorithm인 경사 하강법(Gradient Descent)에 대해서 다뤄보자.

옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 경사 하강법을 이해하기 위해서 cost와 기울기 W와의 관계를 이해해야 한다.
- W는 ML 용어로는 가중치라고 불리지만, 직선의 방정식 관점에서 보면 직선의 기울기를 의미한다.

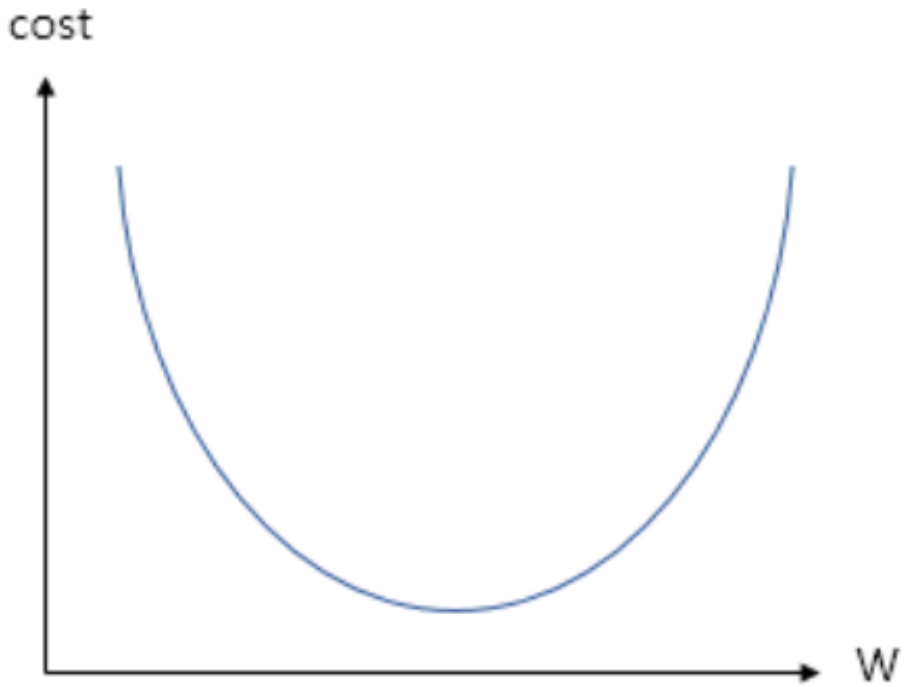


옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 앞의 그림에서 보면, 노란색선은 기울기 W가 20일 때, 초록색선은 기울기 W가 1일 때를 보여주고 있다.
- 이를 수식으로 바꾸면, 각각 y = 20x, y = x에 해당되는 직선이다.
- ↕는 각 점에서의 실제값과 두 직선의 예측값과의 오차를 보여준다.
- 기울기가 지나치게 크면 실제값과 예측값의 오차가 커지고, 기울기가 지나치게 작아도 실제값과 예측값의 오차가 커지는 것을 알 수 있다.
- 사실 b 또한 마찬가지인데 b가 지나치게 크거나 작으면 오차가 커지게 된다.

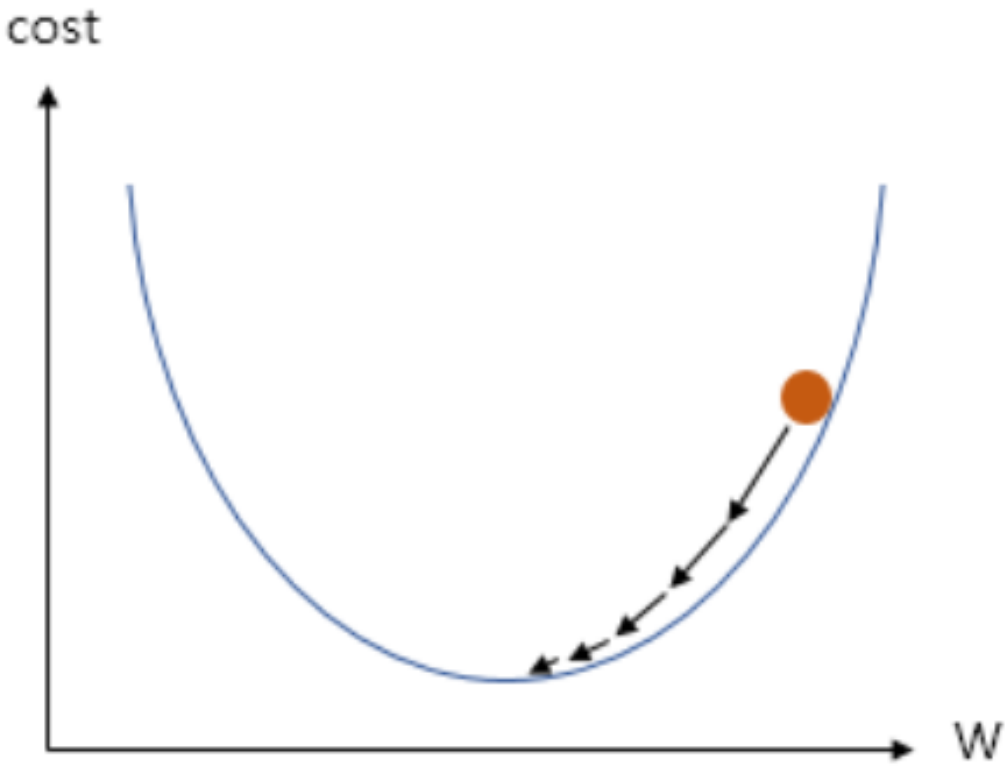
옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 간단하게 설명하기 위해, 편향 b가 없이 단순히 가중치 W만을 사용하면 y = Wx라는 가설 H(x)를 가지고, 경사하강법을 수행해보자.
- 비용 함수의 값 cost(W)는 cost라고 줄여서 표현해서 W와 cost의 관계를 그래프로 표현하면 오른쪽 그림과 같다.



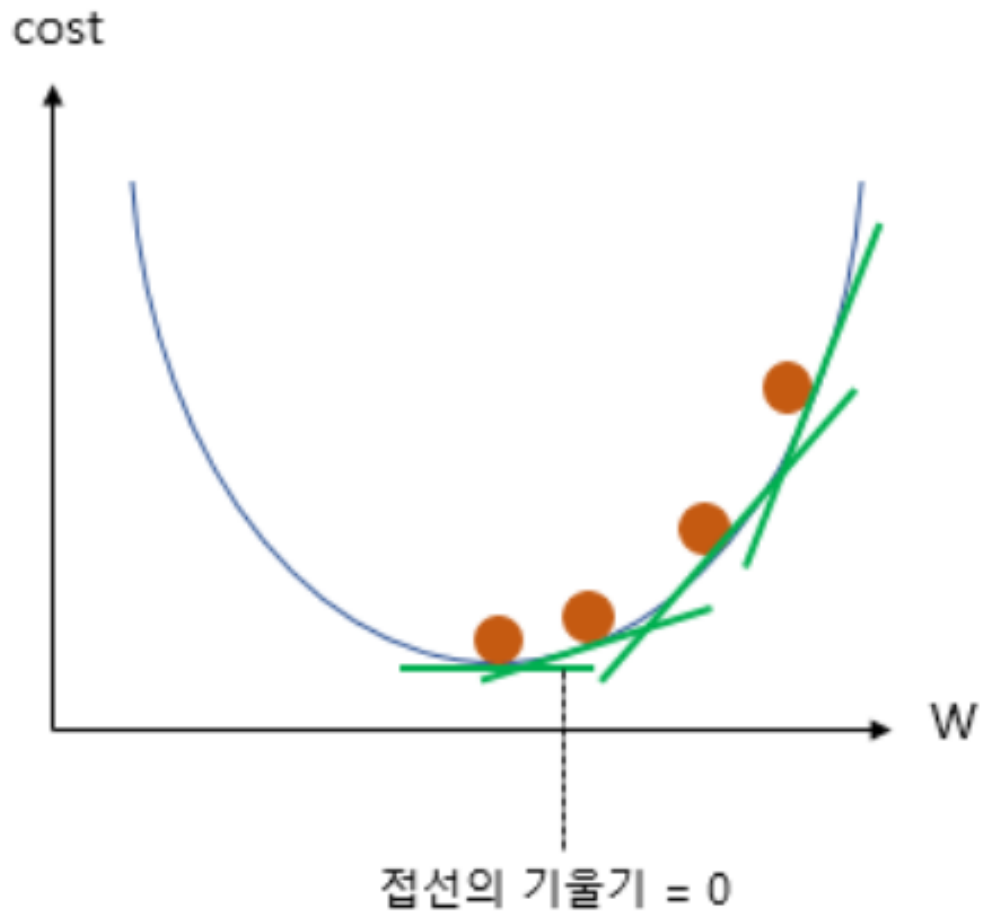
옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 기울기 W가 무한대로 커지면 커질수록 cost의 값 또한 무한대로 커지고, 반대로 기울기 W가 무한대로 작아져도 cost의 값은 무한대로 커지게 된다.
- 아래의 그래프에서 cost가 가장 작을 때는 맨 아래의 볼록한 부분이다.
- 기계가 해야할 일은 cost가 가장 최소값을 가지게 하는 W를 찾는 일이므로, 맨 아래의 볼록한 부분의 W의 값을 찾아야 한다.



옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 기계는 임의의 랜덤값 W값을 정한 뒤에, 맨 아래의 볼록한 부분을 향해 점차 W의 값을 수정해나간다.
- 옆의 그림은 W값이 점차 수정되는 과정을 보여준다.
- 이를 가능하게 하는 것이 경사 하강법(Gradient Descent)이라고 한다.
- 경사 하강법은 한 점에서의 순간 변화율 또는 다른 표현으로는 접선에서의 기울기의 개념을 사용한다.



옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 주목할 것은 맨 아래의 볼록한 부분으로 갈수록 접선의 기울기가 점차 작아진다는 점이다.
- 결국, 맨 아래의 볼록한 부분에서는 결국 접선의 기울기가 0이 된다.
- 그래프 상으로는 초록색 화살표가 수평이 되는 지점을 말한다.
- 즉, cost가 최소화가 되는 지점은 접선의 기울기가 0이 되는 지점이며, 또한 미분값이 0이 되는 지점이다.
- 경사 하강법의 아이디어는 비용 함수(Cost function)를 미분하여 현재 W에서의 접선의 기울기를 구하고, 접선의 기울기가 낮은 방향으로 W의 값을 변경하고 다시 미분하고 이 과정을 접선의 기울기가 0인 곳을 향해 W의 값을 변경하는 작업을 반복하는 것에 있다.

옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 기존의 비용 함수(Cost function)는 다음과 같았다.

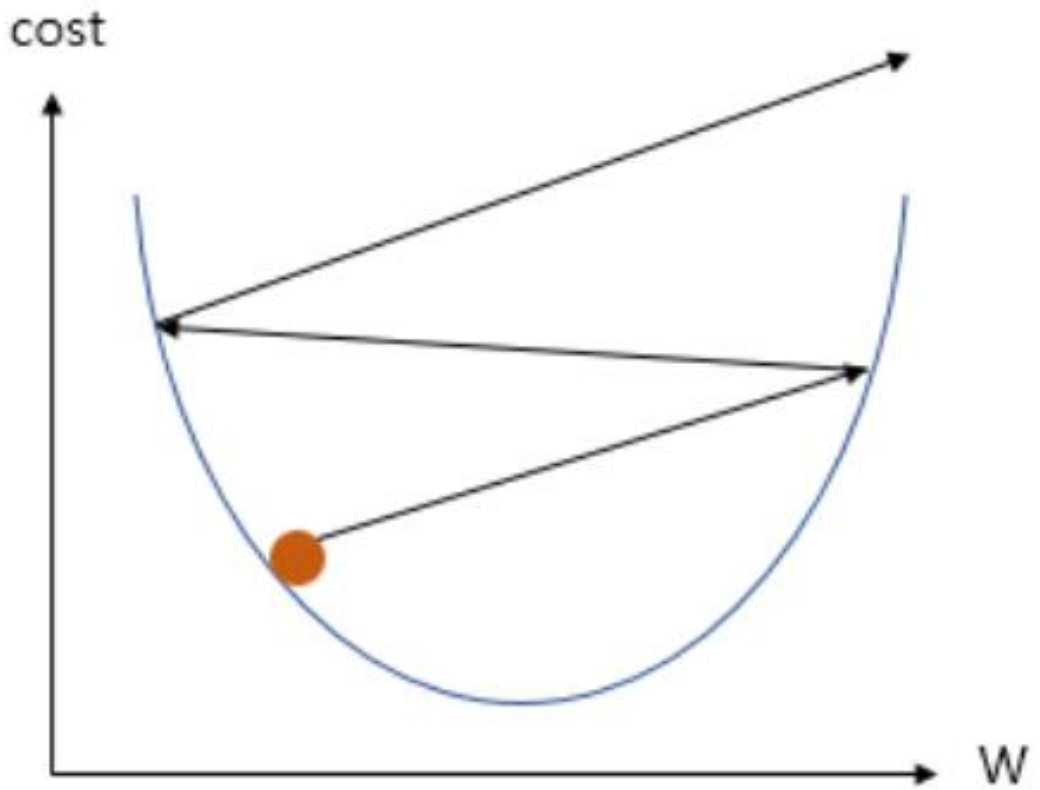
$$cost(W) = \frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$

- 이제 cost를 최소화하는 W를 구하기 위한 식은 다음과 같다.
- 해당 식은 접선의 기울기가 0이 될 때까지 반복한다.

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

옵티마이저(Optimizer) : 경사하강법(Gradient Descent) (Cont.)

- 앞의 식은 현재 W에서의 접선의 기울기와 α와 곱한 값을 현재 W에서 빼서 새로운 W의 값으로 한다는 것을 의미한다.
- α는 여기서 학습률(learning rate)이라고 한다.
- 학습률 α은 W의 값을 변경할 때, 얼마나 크게 변경할지를 결정하는 역할을 한다.



케라스로 구현하는 선형 회귀

- 앞의 식은 현재 W에서의 접선의 기울기와 α와 곱한 값을 현재 W에서 빼서 새로운 W의 값으로 한다는 것을 의미한다.
- α는 여기서 학습률(learning rate)이라고 한다.
- 학습률 α은 W의 값을 변경할 때, 얼마나 크게 변경할지를 결정하는 역할을 한다.

Logistic Regression

로지스틱 회귀(Logistic Regression) - 이진 분류

- 개요
  - 일상 속 풀고자 하는 많은 문제 중에서는 두 개의 선택지 중에서 정답을 고르는 문제가 많다.
  - 예를 들어 시험을 봤는데 이 시험 점수가 합격인지 불합격인지가 궁금할 수도 있고, 어떤 메일을 받았을 때 이게 정상 메일인지 스팸 메일인지를 분류하는 문제도 그렇다.
  - 이렇게 둘 중 하나를 결정하는 문제를 이진 분류(Binary Classification)이라고 하고, 이런 문제를 풀기 위한 대표적인 알고리즘으로 로지스틱 회귀(Logistic Regression)가 있다.

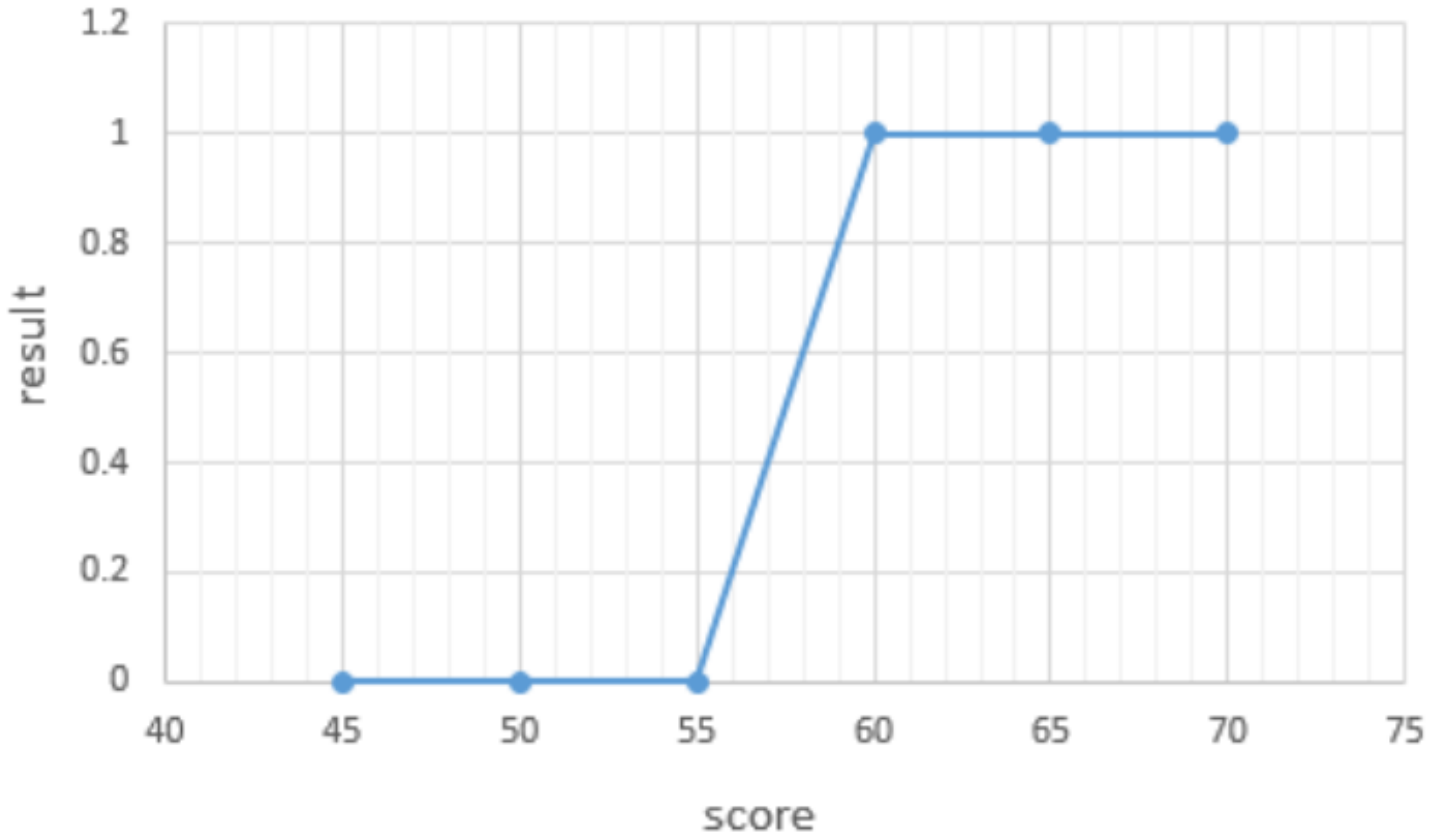
이진 분류(Binary Classification)

- 다음의 예를 보자.
  - 학생들이 시험 성적에 따라서 다음과 같은 결과를 얻었다는 데이터가 있다고 가정해보자.
  - 시험 성적이 x라면, 합격과 불합격 결과는 y가 된다.
  - 이 시험의 커트라인은 공개되지 않았는데 이 데이터로부터 특정 점수를 얻었을 때의 합격, 불합격 여부를 판정하는 모델을 만들려고 한다.

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격

이진 분류(Binary Classification) (Cont.)

- 앞의 데이터에서 합격을 1, 불합격을 0이라고 하였을 때 그래프를 그려보면 아래와 같다.



이진 분류(Binary Classification) (Cont.)

- 이러한 점들을 표현하는 그래프는 알파벳의 S자 형태로 표현된다.
- 또한 이번 예제의 경우 실제값 y가 0 또는 1이라는 두 가지 값밖에 가지지 않으므로, 이 문제를 풀기 위해서는 예측값이 0과 1사이의 값을 가지도록 하는 것이 보편적이다.
- 즉, 0과 1사이의 값을 확률로 해석하면 문제를 풀기가 훨씬 용이해진다.
- 0과 1사이의 값을 가지면서, S자 형태로 그려지는 이러한 조건을 충족하는 유명한 함수가 존재하는데, 바로 시그모이드 함수(Sigmoid function)이다.

시그모이드 함수(Sigmoid function)

- 이 문제에서 사용하게 될 Sigmoid 함수의 방정식은 아래와 같다.
- 종종 σ로 축약해서 표현하기도 한다.
- 이것은 앞의 문제를 풀기 위한 가설(Hypothesis)식이기도 하다.

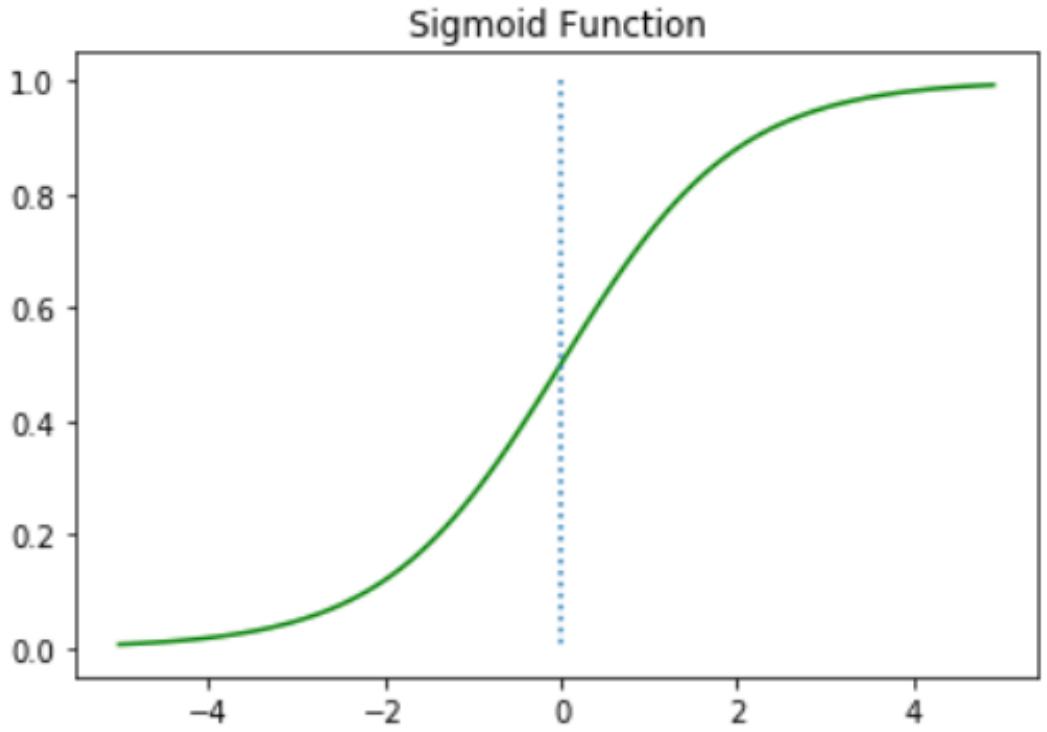
$$H(X) = \frac{1}{1 + e^{-(Wx+b)}} = sigmoid(Wx + b) = \sigma(Wx + b)$$

- 여기서 구해야 할 것은 여전히 주어진 데이터에 가장 적합한 가중치 W(weight)와 편향 b(bias)이다.

시그모이드 함수(Sigmoid function) (Cont.)

- Sigmoid 함수를 Python의 Matplotlib을 통해서 Graph로 표현하면 다음과 같다.
- 아래의 그래프는 W는 1, b는 0임을 가정한 그래프입니다.

```python
import numpy as np # 넘파이 사용
import matplotlib.pyplot as plt # 맷플롯립 사용
%matplotlib inline

def sigmoid(x):
    return 1/(1+np.exp(-x))
x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid(x)

plt.plot(x, y, 'g')
plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
plt.title('Sigmoid Function')
plt.show()
```



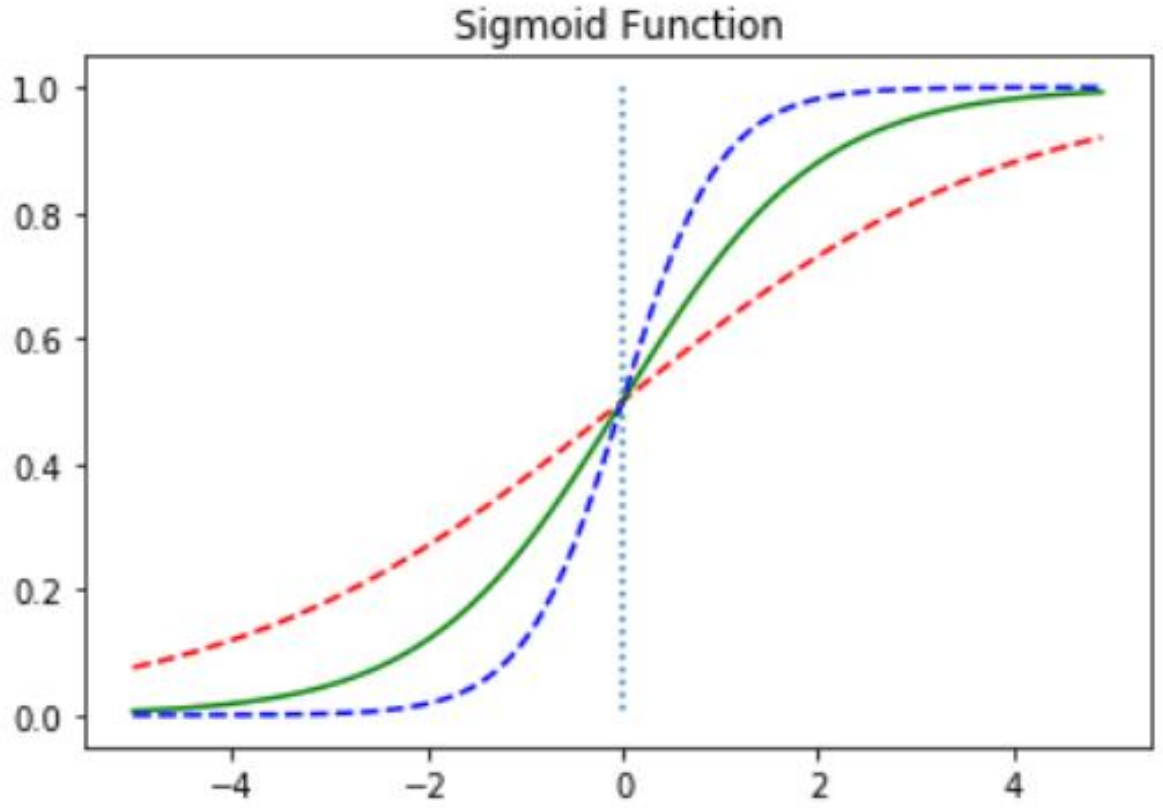
시그모이드 함수(Sigmoid function) (Cont.)

- 앞의 그래프를 통해 알 수 있는 것은 Sigmoid 함수는 출력값을 0과 1 사이의 값으로 조정하여 반환한다는 것이다.
- 마치 S자의 모양을 연상시킨다.
- x가 0일 때 0.5의 값을 가진다.
- x가 증가하면 1에 수렴한다.

시그모이드 함수(Sigmoid function) (Cont.)

- W 의 값을 변화시키고 이에 따른 그래프를 확인해보자.

```python
import numpy as np # 넘파이 사용
import matplotlib.pyplot as plt # 맷플롯립 사용

def sigmoid(x):
    return 1/(1+np.exp(-x))

x = np.arange(-5.0, 5.0, 0.1)
y1 = sigmoid(0.5*x)
y2 = sigmoid(x)
y3 = sigmoid(2*x)

plt.plot(x, y1, 'r', linestyle='--') # W의 값이 0.5일때
plt.plot(x, y2, 'g') # W의 값이 1일때
plt.plot(x, y3, 'b', linestyle='--') # W의 값이 2일때
plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
plt.title('Sigmoid Function')
plt.show()
```



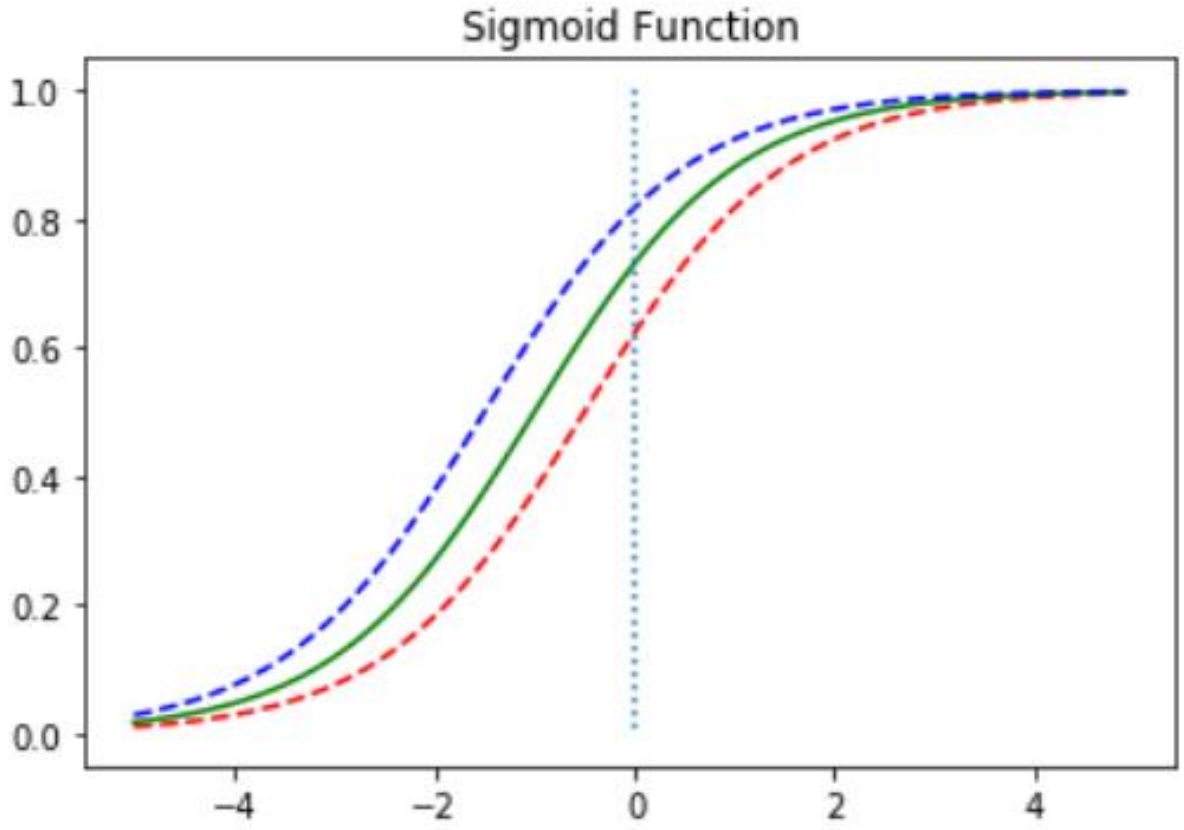
시그모이드 함수(Sigmoid function) (Cont.)

- 앞의 그래프는
  - W의 값이 0.5일때 빨간색선,
  - W의 값이 1일때는 초록색선,
  - W의 값이 2일때 파란색선
- 이 나오도록 하였다.
- **W의 값이 커지면 경사가 커지고 W의 값이 작아지면 경사가 작아진다**.

시그모이드 함수(Sigmoid function) (Cont.)

- b의 값에 따라서 그래프가 어떻게 변하는지 확인해보자.

```python
import numpy as np # 넘파이 사용
import matplotlib.pyplot as plt # 맷플롯립 사용

def sigmoid(x):
    return 1/(1+np.exp(-x))
x = np.arange(-5.0, 5.0, 0.1)
y1 = sigmoid(x+0.5)
y2 = sigmoid(x+1)
y3 = sigmoid(x+1.5)

plt.plot(x, y1, 'r', linestyle='--') # x + 0.5
plt.plot(x, y2, 'g') # x + 1
plt.plot(x, y3, 'b', linestyle='--') # x + 1.5
plt.plot([0,0],[1.0,0.0], ':') # 가운데 점선 추가
plt.title('Sigmoid Function')
plt.show()
```



시그모이드 함수(Sigmoid function) (Cont.)

- 앞의 그래프는 b의 값에 따라서 그래프가 이동하는 것을 보여준다.
- 정리
  - Sigmoid 함수는 입력값이 커지면 1에 수렴하고, 입력값이 작아지면 0에 수렴한다.
  - 0부터의 1까지의 값을 가진다.
  - 출력값이 0.5 이상이면 1(True), 0.5이하면 0(False)로 만들면 이진 분류 문제로 사용할 수 있다.