

# Tensorflow

---

박경미

# 1. TensorFlow Basic

## ❖ TensorFlow

- An open source software library for numerical computation using data flow graphs
- **Python! 기반**

Deep learning libraries:  
Accumulated GitHub metrics

Aggregate popularity (30•contrib + 10•issues + 5•forks)•1e-3		
#1: 172.29		tensorflow/tensorflow
#2: 89.78		BVLC/caffe
#3: 69.70		fchollet/keras
#4: 53.09		dmlc/mxnet
#5: 38.23		Theano/Theano
#6: 29.86		deeplearning4j/deeplearning4j
#7: 27.99		Microsoft/CNTK
#8: 17.36		torch/torch7
#9: 14.43		baidu/paddle
#10: 13.10		pfnet/chainer
#11: 12.37		NVIDIA/DIGITS
#12: 10.42		tflearn/tflearn
#13: 9.20		pytorch/pytorch

Deep learning libraries: growth over past three months

new contributors from 2016-10-09 to 2017-02-10		new forks from 2016-10-09 to 2017-02-10	
#1: 192	tensorflow/tensorflow	#1: 6525	tensorflow/tensorflow
#2: 89	dmlc/mxnet	#2: 1822	BVLC/caffe
#3: 78	fchollet/keras	#3: 1316	fchollet/keras
#4: 42	baidu/paddle	#4: 999	dmlc/mxnet
#5: 29	Microsoft/CNTK	#5: 909	deeplearning4j/deeplearning4j
#6: 23	pfnet/chainer	#6: 887	Microsoft/CNTK
#7: 21	Theano/Theano	#7: 324	tflearn/tflearn
#8: 20	deeplearning4j/deeplearning4j	#8: 321	baidu/paddle
#9: 20	tflearn/tflearn	#9: 287	Theano/Theano
#10: 19	BVLC/caffe	#10: 257	torch/torch7
#11: 9	torch/torch7	#11: 175	NVIDIA/DIGITS
#12: 3	NVIDIA/DIGITS	#12: 142	pfnet/chainer

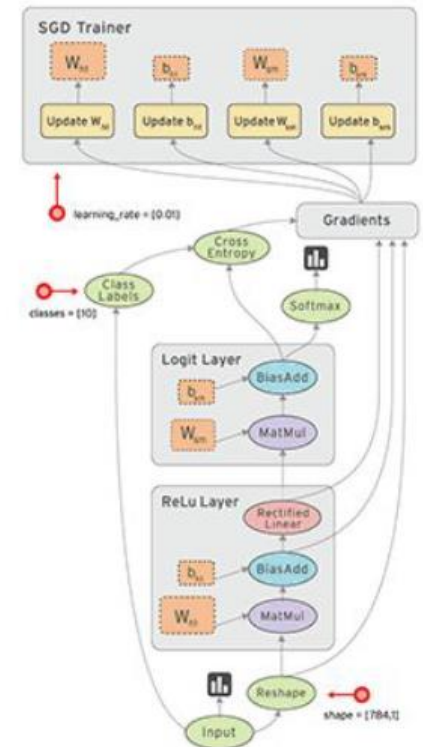
new issues from 2016-10-09 to 2017-02-10		aggregate metrics growth from 2016-10-09 to 2017-02-10	
#1: 1563	tensorflow/tensorflow	#1: 54.01	tensorflow/tensorflow
#2: 979	fchollet/keras	#2: 18.71	fchollet/keras
#3: 871	dmlc/mxnet	#3: 16.38	dmlc/mxnet
#4: 646	baidu/paddle	#4: 12.86	BVLC/caffe
#5: 486	Microsoft/CNTK	#5: 10.17	Microsoft/CNTK
#6: 361	deeplearning4j/deeplearning4j	#6: 9.32	baidu/paddle
#7: 318	BVLC/caffe	#7: 8.75	deeplearning4j/deeplearning4j
#8: 217	NVIDIA/DIGITS	#8: 4.21	Theano/Theano
#9: 214	Theano/Theano	#9: 3.89	tflearn/tflearn
#10: 167	tflearn/tflearn	#10: 3.14	NVIDIA/DIGITS
#11: 150	pfnet/chainer	#11: 2.90	pfnet/chainer
#12: 90	torch/torch7	#12: 2.46	torch/torch7

# TensorFlow Basic

## ❖ Data Flow Graph Computation

### ▪ Data Flow Graph

- 노드를 연결하는 엣지가 데이터를 노드는 데이터를 통해 수행하는 연산 역할을 하는 그래프 구조를 의미
- 데이터가 edge 역할을 하여 node로 흘러가는 그래프 구조를 갖으며 node에 지정된 연산을 하는 연산방법
- 텐서플로우의 경우 이름에서 알 수 있듯 텐서(Tensors)가 기본 자료구조이기 때문에 텐서플로우의
  - 엣지(Edge)는 텐서를 의미하며 엣지의 방향은 텐서의 흐름을 의미하고
  - 노드(Node)는 곱하고, 나누는 등 텐서를 처리하는 연산을 의미한다 .



## ❖ TensorFlow 설치

```
pip install --upgrade tensorflow
```

관리자 모드로 console 열기

```
conda install tensorflow # pip install tensorflow
```

## ❖ 버전확인

```
import tensorflow as tf  
tf.__version__
```

## ❖ example

```
import tensorflow.compat.v1 as tf  
tf.disable_v2_behavior()  
  
hello=tf.constant("Hello, TensorFlow")  
sess=tf.Session()  
print(sess.run(hello))
```

Tensorflow2버전 설치 후 버전1 사용

## ❖ Computational Graph

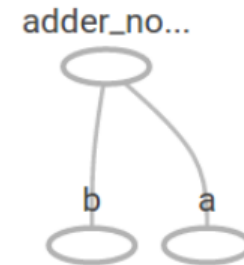
```
node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
node3 = tf.add(node1, node2)
```

```
print("node1:", node1, "node2:", node2)
print("node3: ", node3)
```

```
node1: Tensor("Const_1:0", shape=(), dtype=float32)
node2: Tensor("Const_2:0", shape=(), dtype=float32)
node3: Tensor("Add:0", shape=(), dtype=float32)
```

```
sess = tf.Session()
print("sess.run(node1, node2): ", sess.run([node1, node2]))
print("sess.run(node3): ", sess.run(node3))
```

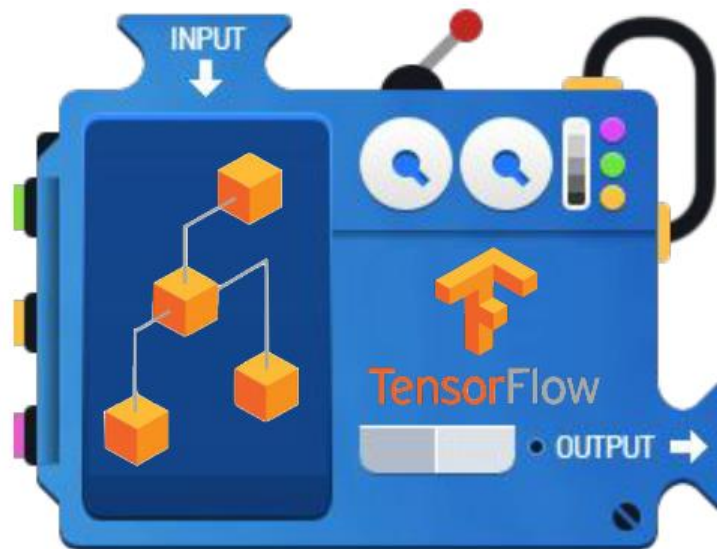
```
sess.run(node1, node2): [3.0, 4.0]
sess.run(node3): 7.0
```



## ❖ TensorFlow Mechanics

2 feed data and run graph (operation)  
`sess.run (op)`

1 Build graph using  
TensorFlow operations



3 update variables  
in the graph  
(and return values)

## ❖ Computational Graph

(1) Build graph (tensors) using TensorFlow operations

```
In [4]: node1 = tf.constant(3.0, tf.float32)
        node2 = tf.constant(4.0) # also tf.float32 implicitly
        node3 = tf.add(node1, node2)
```

(2) feed data and run graph (operation)  
***sess.run (op)***

(3) update variables in the graph  
(and return values)

```
In [6]: sess = tf.Session()
        print("sess.run(node1, node2): ", sess.run([node1, node2]))
        print("sess.run(node3): ", sess.run(node3))
```

```
sess.run(node1, node2): [3.0, 4.0]
sess.run(node3): 7.0
```

---

## ❖ Placeholder

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b # + provides a shortcut for tf.add(a, b)

print(sess.run(adder_node, feed_dict={a: 3, b: 4.5}))
print(sess.run(adder_node, feed_dict={a: [1,3], b: [2, 4]}))
```

7.5 [ 3. 7.]

```
add_and_triple = adder_node * 3.
print(sess.run(add_and_triple, feed_dict={a: 3, b:4.5}))
```

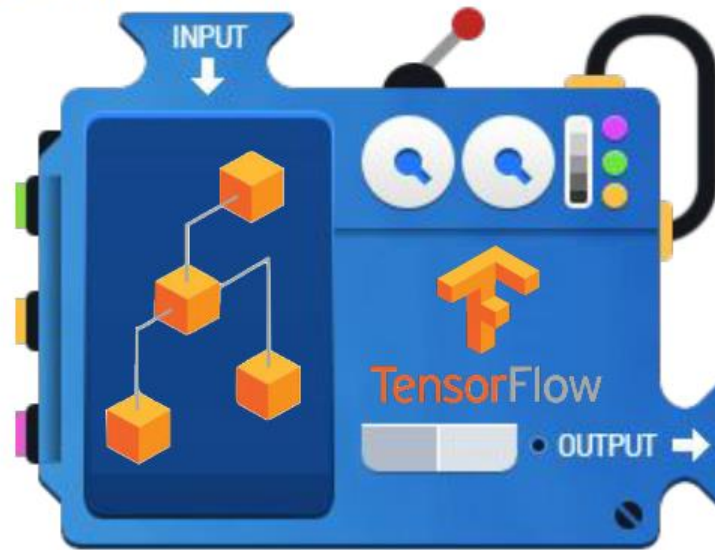
22.5



## ❖ TensorFlow Mechanics

2 feed data and run graph (operation)  
`sess.run (op, feed_dict={x: x_data})`

1 Build graph using  
TensorFlow operations



3 update variables  
in the graph  
(and return values)

---

## ❖ Tensors : Everything is Tensor

```
3 # a rank 0 tensor; this is a scalar with shape []  
[1., 2., 3.] # a rank 1 tensor; this is a vector with shape [3]  
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]  
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

```
[[[1.0, 2.0, 3.0]], [[7.0, 8.0, 9.0]]]
```

```
t = tf.Constant([1., 2., 3.])
```

## ❖ Tensor Ranks, Shapes, and Types : Ranks

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Rank	Math entity	Python example
0	Scalar (magnitude only)	<b>s = 483</b>
1	Vector (magnitude and direction)	<b>v = [1.1, 2.2, 3.3]</b>
2	Matrix (table of numbers)	<b>m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]</b>
3	3-Tensor (cube of numbers)	<b>t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]</b>
n	n-Tensor (you get the idea)	<b>....</b>

## ❖ Tensor Ranks, Shapes, and Types : shapes

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

## ❖ Tensor Ranks, Shapes, and Types : type

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

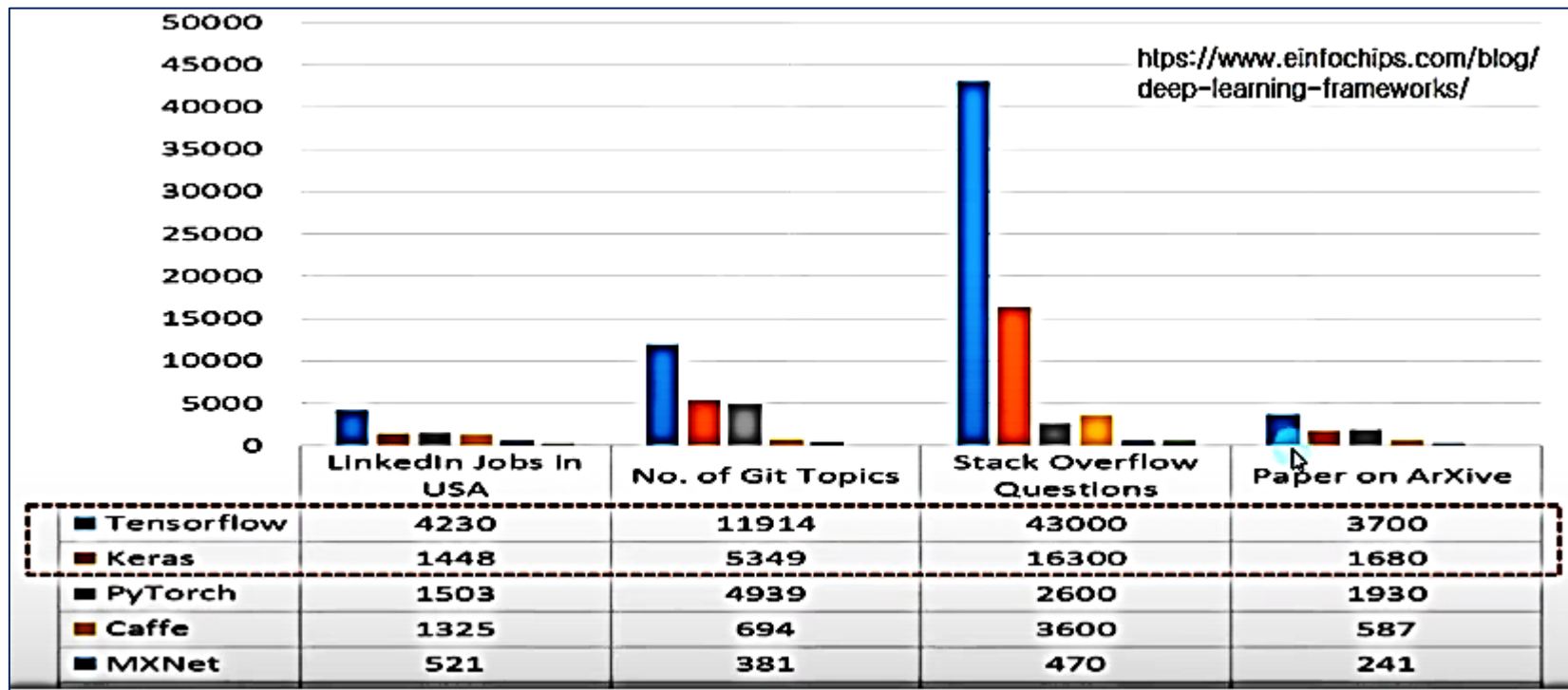
Data type	Python type	Description
DT_FLOAT	<code>tf.float32</code>	32 bits floating point.
DT_DOUBLE	<code>tf.float64</code>	64 bits floating point.
DT_INT8	<code>tf.int8</code>	8 bits signed integer.
DT_INT16	<code>tf.int16</code>	16 bits signed integer.
DT_INT32	<code>tf.int32</code>	32 bits signed integer.
DT_INT64	<code>tf.int64</code>	64 bits signed integer.

...

## 2. TensorFlow 2 -Deep Learning Library

### ❖ TensorFlow2

- 텐서(Tensor)를 흘러 보내면서(Flow) 딥러닝 알고리즘을 수행하는 라이브러리
- 2020년 현재 전 세계에서 가장 많이 사용
- TensorFlow 2.0부터 직관적이고 쉽게 배울 수 있는 Keras를 High-Level API로 공식 지원함으로써, 이러한 영향력은 더욱 커질 것으로 예상됨



# 1. TensorFlow-Deep Learning Library

## ❖ TensorFlow 2.0

- 2019년 9월 30일에 TensorFlow 2.0 정식 Release 됨
- 즉시 실행 모드로 불리는 Eager Execution 적용되어 코드의 직관성이 높아졌으며
- 사용자 친화적이어서 쉽게 배울 수 있는 Keras만을 High-Level API로 공식 지원함

"If you have started with TensorFlow 2.0 and have never seen TensorFlow 1.x, then you are lucky."  
Deep Learning with TensorFlow 2 and Keras, 2nd Edition, Packt, 2020.04

### Eager Execution

```
W = tf.Variable(tf.random.normal([1])) # 가우시안 분포  
print('initial W = ', W.numpy())  
print('=====')
```

```
# session 생성 없이 즉시 실행 (Eager Execution)  
# numpy() 메서드 사용하면 numpy 값을 리턴해줌
```

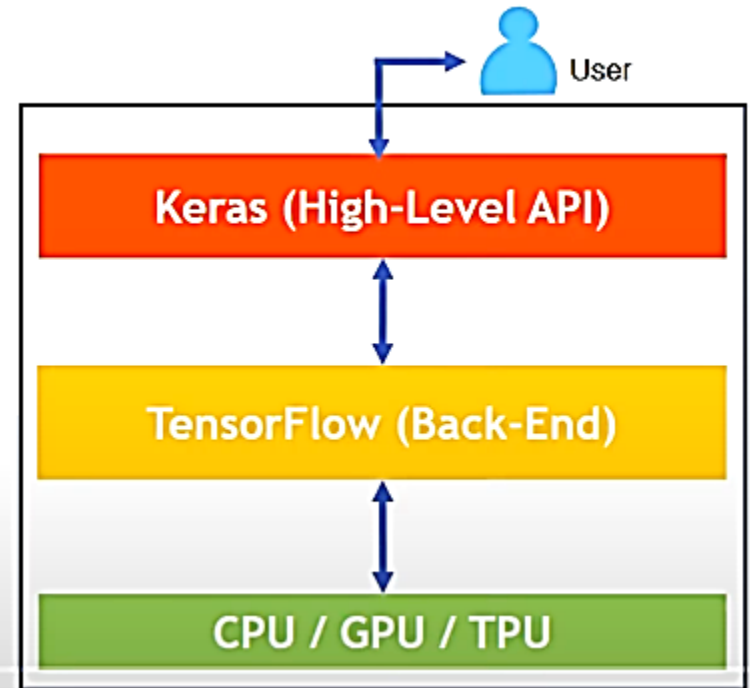
```
for step in range(2):  
    W = W + 1.0  
    print('step = ', step, ', W = ', W.numpy())
```

```
initial W = [0.588869]
```

```
=====
```

```
step = 0 , W = [1.588869]
```

```
step = 1 , W = [2.588869]
```



## 2. Eager Execution

### ❖ Eager Execution(즉시 실행모드)

- 계산 그래프와 세션을 생성하지 않고 즉시 실행 가능한 **Eager Execution** 적용

```
import tensorflow as tf
import numpy as np
```

```
tf.__version__  
  
'2.2.0'
```

```
a = tf.constant(10)  
b = tf.constant(20)
```

```
c = a + b  
d = (a+b).numpy()
```

```
print(type(c))  
print(c)  
print(type(d), d)
```

```
d_numpy_to_tensor = tf.convert_to_tensor(d)
```

```
print(type(d_numpy_to_tensor))  
print(d_numpy_to_tensor)
```

```
<class 'tensorflow.python.framework.ops.EagerTensor'>  
tf.Tensor(30, shape=(), dtype=int32)  
<class 'numpy.int32'> 30  
<class 'tensorflow.python.framework.ops.EagerTensor'>  
tf.Tensor(30, shape=(), dtype=int32)
```

- TensorFlow 1.x에서는 계산 그래프를 선언하고, 세션을 통해 텐서(Tensor)를 주고받으며 계산하는 구조
- TensorFlow 2.X에서는 자동으로 Eager Execution(즉시 실행 모드) 적용되기 때문에 그래프와 세션을 만들지 않아도 텐서 값을 계산하고 numpy() 함수를 이용하면 파이썬의 넘파이 타입으로 변환할 수 있음



Eager Execution 기능을 통해서 텐서플로를 파이썬처럼 사용할 수 있음



## 2. Eager Execution

### ❖ Eager Execution(즉시 실행모드)

- TensorFlow 2.0에서는 오퍼레이션을 실행하는 순간 연산이 즉시 수행(**Eager Execution**)되기 때문에 오퍼레이션 실행결과를 `numpy()` 메서드를 통하여 바로 알 수 있다.

#### TensorFlow 1.5

```
%tensorflow_version 1.x

import tensorflow as tf

print('tensorflow version = ', tf.__version__)
print('=====')

a = tf.constant(1.0)
b = tf.constant(2.0)

c = a + b

print('c = ', c)

with tf.Session() as sess:    # session 만든 후에 연산 실행
    print(sess.run(c))

tensorflow version = 1.15.2
=====
c = Tensor("add_7:0", shape=(), dtype=float32)
3.0
```

#### TensorFlow 2.X

```
import tensorflow as tf

tf.__version__

'2.2.0'

a = tf.constant(1.0)
b = tf.constant(2.0)

c = a + b

print(c.numpy())    # Eager Execution

3.0
```

## 2. Eager Execution

### ❖ tf.Variable(...)

- TensorFlow에서 tf.Variable()값을 토기화하기 위해 세션내에서 tf.global\_variables\_initializer() 과정이 필요 없으며, 변수를 정의함과 동시에 초기 값이 할당됨(Eager Execution)

```
%tensorflow_version 1.x

import tensorflow as tf

print('tensorflow version = ', tf.__version__)
print('=====')

W = tf.Variable(tf.random_normal([1])) # 가우시안 분포
print(W)
print('=====')

# session 생성 하고,
# tf.Variable(...) 초기화 해주는 코드 실행 후 연산 실행

with tf.Session() as sess:

    # 변수 노드 값 초기화
    sess.run(tf.global_variables_initializer())

    for step in range(2):
        W = W + 1.0
        print('step = ', step, ', W = ', sess.run(W))
```

```
tensorflow version = 1.15.2
```

```
=====
<tf.Variable 'Variable_1:0' shape=(1,) dtype=float32_ref>
=====
```

```
step = 0 , W = [0.8110943]
```

```
step = 1 , W = [1.8110943]
```

```
W = tf.Variable(tf.random_normal([1])) # 가우시안 분포
print('initial W = ', W.numpy())
print('=====')
```

```
# session 생성 없이 즉시 실행 (Eager Execution)
# numpy() 메서드 사용하면 numpy 값을 리턴해줄
```

```
for step in range(2):
    W = W + 1.0
    print('step = ', step, ', W = ', W.numpy())
```

```
initial W = [0.588869]
```

```
=====
```

```
step = 0 , W = [1.588869]
```

```
step = 1 , W = [2.588869]
```

## 2. Eager Execution

### ❖ tf.placeholder(...) 삭제

- TensorFlow 1.x 버전에서 함수를 실행하여 결과를 얻기 위해서는 `tf.placeholder()`에 입력 값을 주고, 그 값을 이용하여 함수에서 정의된 연산을 실행하였으니, TF 2.0에서는 일반적인 python 코드와 마찬가지로 함수에 값을 직접 넘겨주고 즉시 결과를 얻을 수 있음(Eager Execution)

#### TensorFlow 1.15

```
%tensorflow_version 1.x
import tensorflow as tf

print('tensorflow version = ', tf.__version__)
print('=====')

a = tf.placeholder(tf.float32) # 입력 값 저장할 노드 정의
b = tf.placeholder(tf.float32) # 입력 값 저장할 노드 정의

# 함수 정의
def tensor_sum(x, y):
    return x + y

result = tensor_sum(a, b) # 함수 결과 값 저장 할 노드 정의

# session 생성 하고,
# feed_dict 통해서 placeholder 노드에 값 대입

with tf.Session() as sess:
    print(sess.run(result, feed_dict={a: [1.0], b: [3.0]}))
```

실제 데이터는 세션 내에서 입력받기 받는 용도로 사용됨 (Lazy Evaluation)

플레이스홀더 노드에 대입되는 값

#### TensorFlow 2.x

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

# 함수 정의
def tensor_sum(x, y):
    return x + y

result = tensor_sum(a, b)

print(type(result))
print(result.numpy())
```

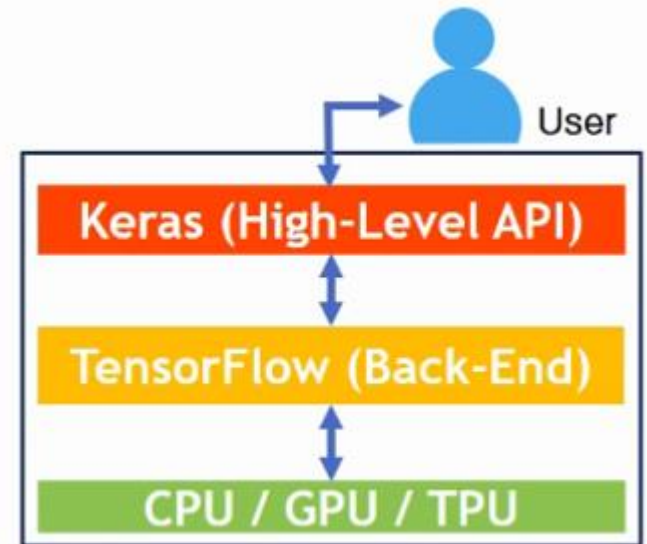
Eager Execution

```
<class 'tensorflow.python.framework.ops.EagerTensor'>
4.0
```

### 3. Keras as High Level API

#### ❖ Keras in TensorFlow 2.0

- Keras 창시자 프랑소와 솔레(Francois Chollet)가 TF 2.0 개발에 참여하였고, TF 2.0에서 공식적이고 유일한 High-Level API로서 keras를 채택함
- 프랑소와 솔레는 앞으로 native Keras 보다 tf.keras 처럼 TF에서 케라스를 사용할 것을 권장함



#### ❖ Keras 특징

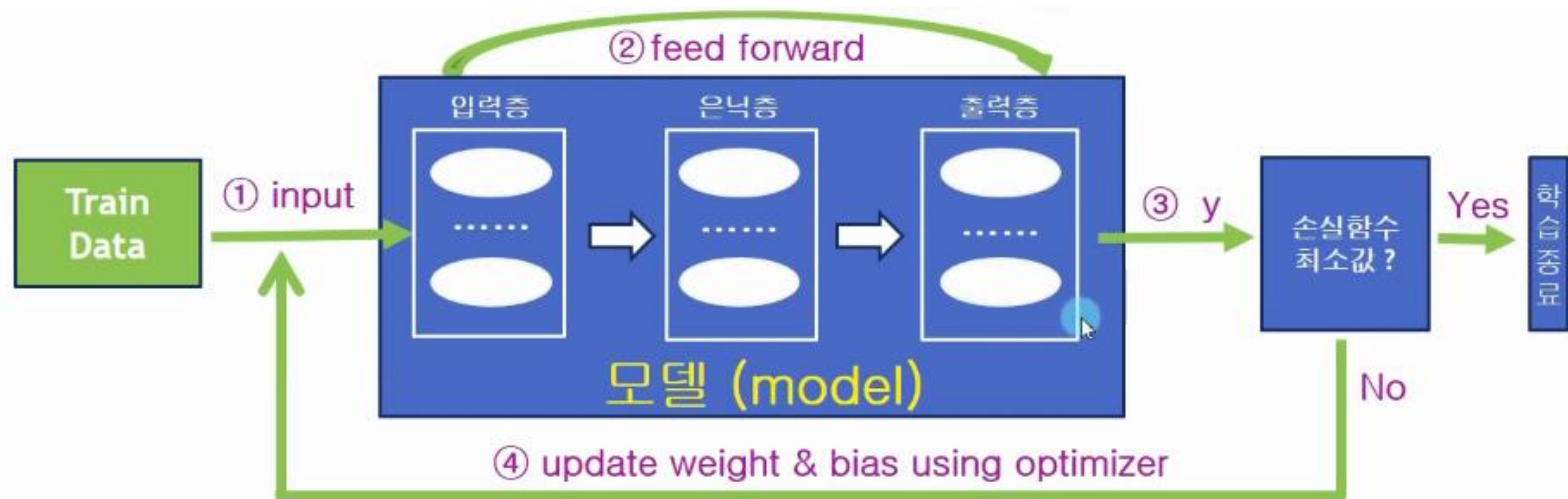
- **User Friendliness** : Keras의 직관적인 API를 이용하면 일반 신경망(ANN), CNN, RNN 모델 또는 이를 조합한 다양한 딥러닝 모델을 (몇 줄의 코드만으로) 쉽게 구축할 수 있음
- **Modularity** : Keras에서 제공하는 모듈은 독립적으로 설정 가능함, 즉 신경망 층, 손실함수, 활성화 함수, 최적화 알고리즘 등은 모두 독립적인 모듈이기 때문에 이러한 모듈을 서로 조합하기만 하면 새로운 딥러닝 모델을 쉽고 빠르게 만들어서 학습시킬 수 있음



Keras 가장 핵심적인 데이터 구조 모델

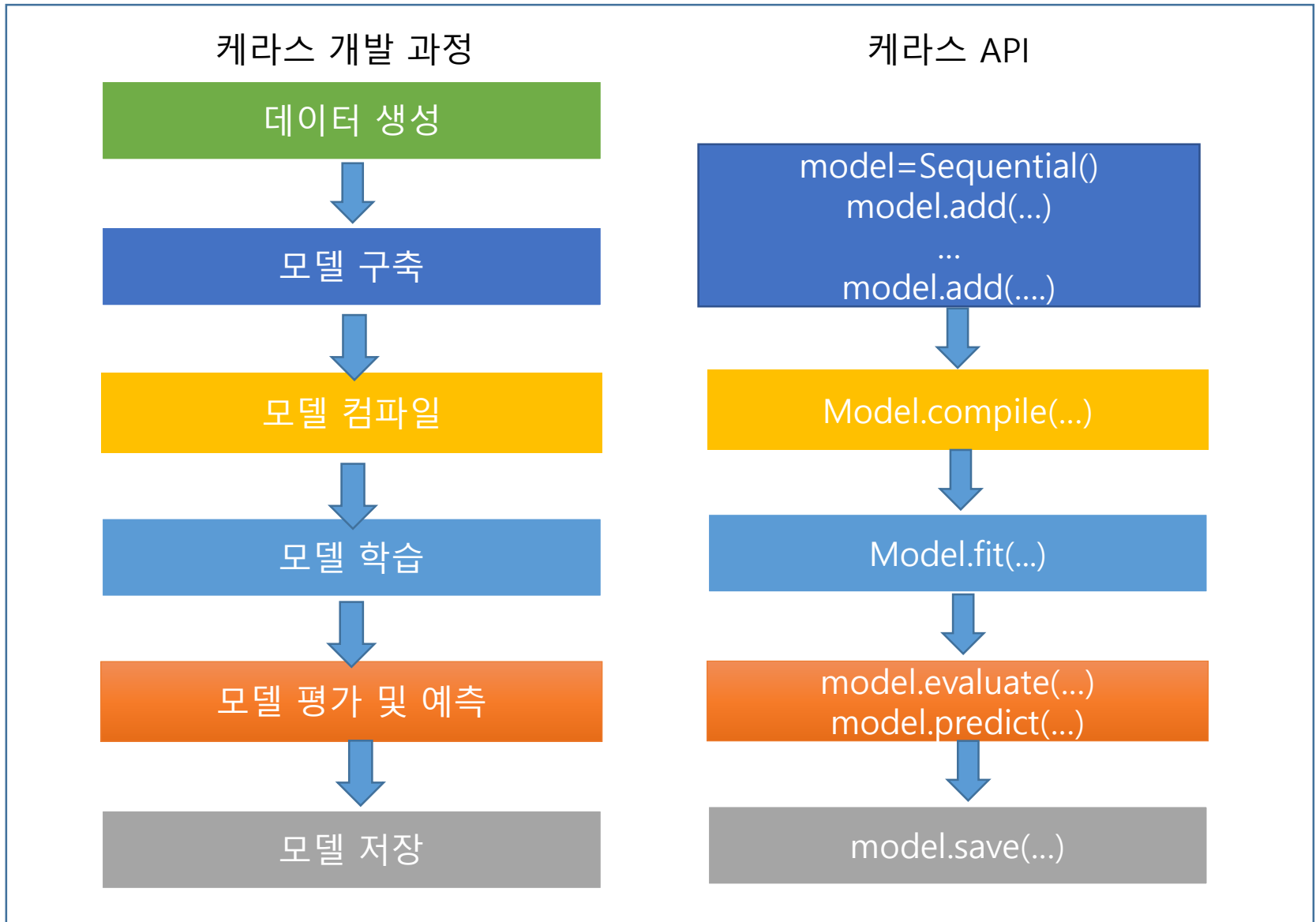
### 3. Keras as High Level API

#### ❖ Keras-모델(Model)

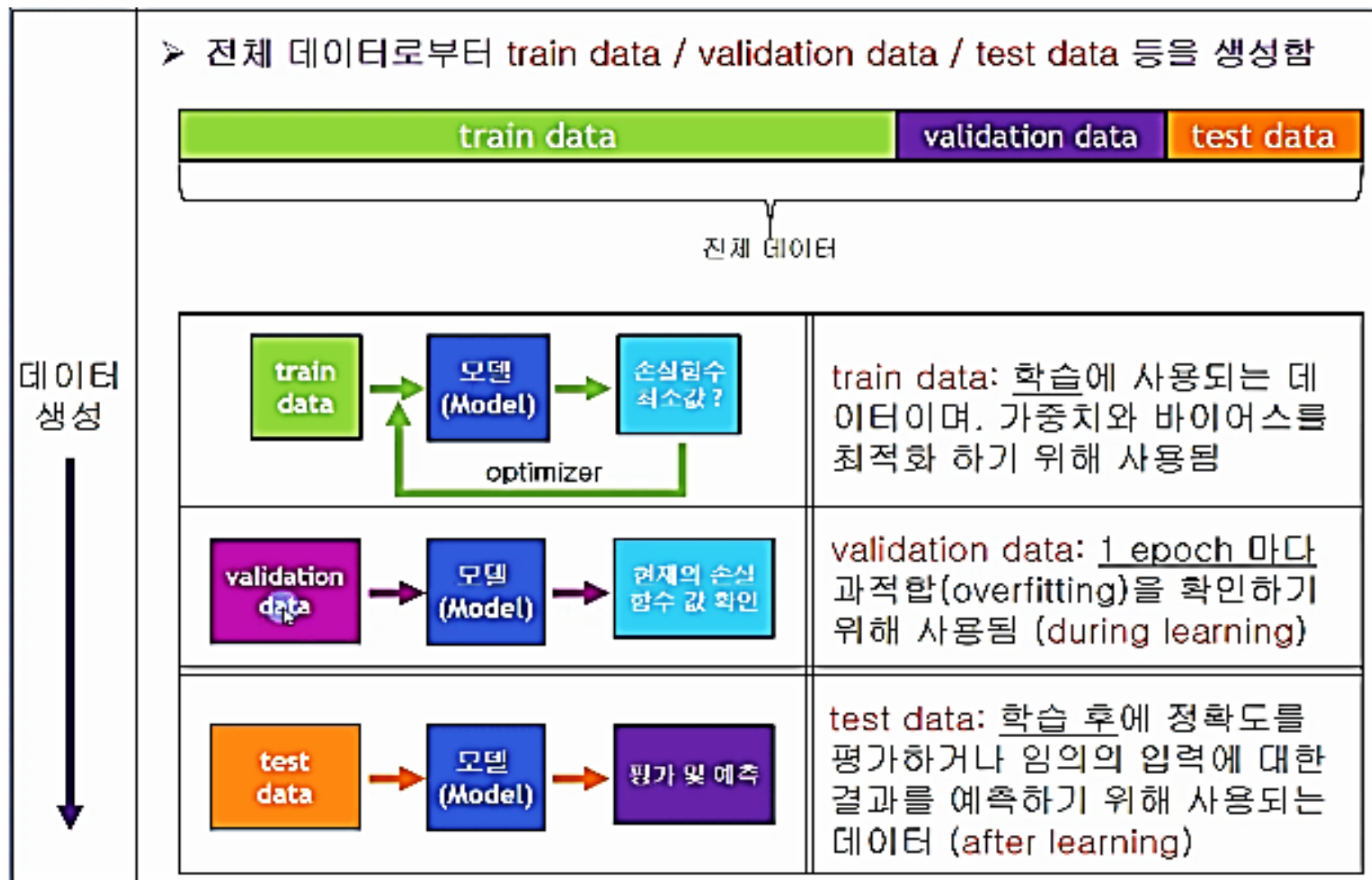


- 모델(model)은 각 층(layer)을 포함하고 있는 인공 신경망 자체를 나타냄	→	<code>model = Sequential()</code> # 모델 생성
- 모델의 기본 단위는 층(layer)이며, 이러한 층을 레고 처럼 순차적으로 쌓기만 하면 일반 신경망(ANN), CNN, RNN 또는 이들을 조합한 다양한 모델을 구축 할 수 있음	→	<code>model.add(...)</code> # 층 추가 <code>model.add(...)</code> # 층 추가 <code>model.add(...)</code> # 층 추가
- 모델이 구축되면 손실 함수 (loss function) 값이 최소가 될 때 까지 ① ~ ④ 과정을 반복하며 최적의 가중치(weight)와 바이어스(bias) 값을 찾는 학습(learning)이 진행됨	→	<code>model.compile(...)</code> # 손실함수 지정 # 옵티마이저 지정 <code>model.fit(...)</code> # 학습진행, ① ~ ④

## 4. Keras-개발과정(데이터 생성)



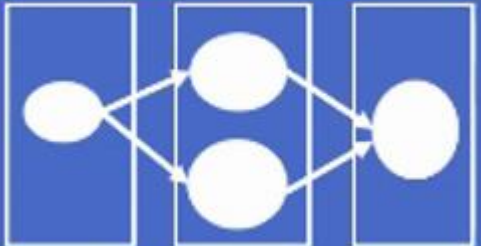
## 4. Keras-개발과정(데이터 생성)



## 4. Keras 개발과정(모델 구축)

모델  
구축

입력층   은닉층   출력층



모델 (model)

```
model = Sequential()
model.add(Flatten(input_shape=(1,)))
model.add(Dense(2, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))




model.add(Dense(2, activation='sigmoid', input_shape=(1,)))
```

①  
②



- **Flatten**는 입력으로 들어오는 다차원 데이터를 1차원으로 정렬하기 위해 사용되는 레이어이며, 입력 데이터(차원)의 수를 `input_shape=(1,)` 과 같이 기술함
- **Dense**는 각 층의 입력과 출력 사이에 있는 모든 노드가 서로 연결되어 있는 완전 연결 층(FC)을 나타내며, **Dense** 첫번째 인자인 2, 1 등은 **출력 노드수를 나타냄**
  - 노드의 활성화 함수는 `activation='...'` 형태로 나타내며, 대표적인 활성화 함수로는 선형회귀 문제에서는 'linear', 일반적인 classification 경우에는 'sigmoid', 'softmax', 'relu', 'tanh' 등이 데이터에 따라 다양하게 사용됨
  - 또한 첫번째 계층에 바로 Dense 계층을 사용하는 경우라면, 코드 ① 같은 Flatten(), Dense() 두개의 코드를 ② 에서 처럼 Dense에서 `input_shape=(1,)` 을 이용하여 한 줄의 코드로 나타낼 수 있음 (일반적으로 ② 처럼 한 줄의 코드로 나타냄)



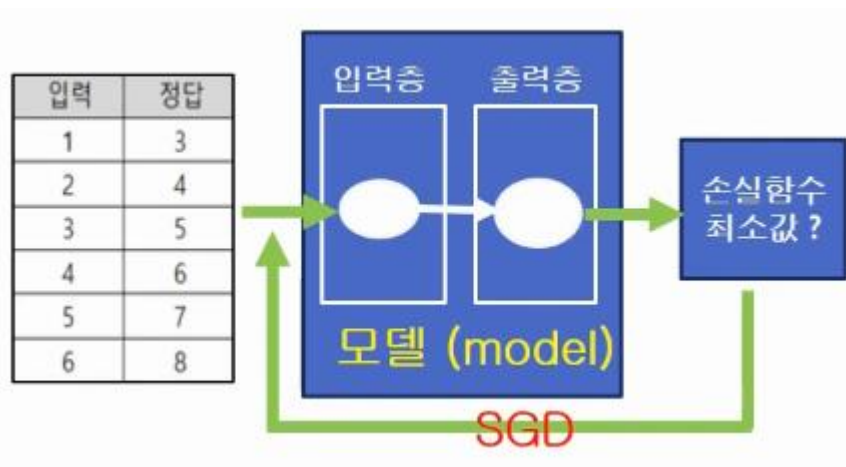
# 4. Keras-개발과정(컴파일링, 모델학습)

 모델 컴파일 	<p>구축된 모델을 기계가 이해할 수 있도록 컴파일(compile)을 해야하며, 최적화 방법(알고리즘), 오차함수, 학습 과정 중에 모니터링 할 지표(metric)를 나타낼 수 있음</p> <p>일반적으로 <code>model.compile()</code> 후에는 <code>model.summary()</code>를 통해서 구축 모델을 확인 함</p> <p>[예1] <code>model.compile(optimizer=SGD(learning_rate=0.1), loss='mse', metrics=['accuracy'])</code></p> <ul style="list-style-type: none"><li>- 최적화 알고리즘 SGD, 학습율 0.1, 오차함수 mse, 메트릭은 accuracy(loss 는 기본)</li></ul> <p>[예2] <code>model.compile(optimizer=Adam(learning_rate=1e-4),</code></p> <div><ul style="list-style-type: none"><li>✓ 손실함수 종류로는 평균제곱 오차인 'mse', 이진 분류인 'binary_crossentropy' 다중 클래스 위해 'categorical_crossentropy' 또는 'sparse_categorical_crossentropy'</li><li>✓ 메트릭은 기본적으로 'loss'인 측정임, 즉 <code>metrics=['loss']</code>임</li></ul></div>
모델 학습 	<p>손실 함수 값이 최소가 될 때까지 각 층의 가중치와 바이어스를 업데이트하는 과정</p> <p>[예] <code>model.fit(x_train, t_train, epochs=10, batch_size=100, verbose=0,</code></p> <div><ul style="list-style-type: none"><li>✓ <code>Verbose=</code> 학습 중 손실 값, 정확도, 진행 상태 등의 출력 형태를 설정함(0, 1, 2)</li><li>✓ 검증데이터가 별도로 있다면, <code>validation_split</code> 대신 <code>validation_data</code> 이용하여 지정 가능</li></ul></div> <p><code>(epochs=10), batch_size 100(생략가능), 학습데이터의 20%를 검증데이터로 자름</code></p>

## 4. Keras 개발과정(모델 평가, 모델저장)

 모델 평가	<p>학습을 마친 후, test data를 통해서 모델을 평가 하고 임의의 데이터에 대해 예측함</p> <p>[예1] <code>model.evaluate(x_test, t_test, epochs=10, batch_size=100)</code> 1<sup>st</sup> 인자는 테스트 데이터, 2<sup>nd</sup> 인자는 정답데이터, 배치사이즈 100 (생략가능)</p> <p>[예2] <code>model.predict(x_input_data, batch_size=100)</code> 1<sup>st</sup> 인자는 예측하고자 하는 데이터, batch_size 생략가능</p>
 모델 저장	<p>학습이 끝난 (가중치와 바이어스가 최적화된) 신경망 구조를 저장해 놓는다면, 다양한 테스트 데이터에 대해 재 학습 없이 지속적으로 사용할 수 있음</p> <p>[예1] <code>model.save("model_name.h5")</code> 학습이 끝난 모델을 hdf5 파일에 저장함</p> <p>[예2] <code>model = tensorflow.keras.models.load_model("model_name.h5")</code> 저장되어 있는 모델(model_name.hdf5)을 불러옴</p>

# 5. Keras-Simple LinearRegression Exercise



```
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.optimizers import SGD

import numpy as np

print(tf.__version__)
```

2.2.0

# [1] 데이터셋 생성

```
x_data = np.array([1, 2, 3, 4, 5, 6])
t_data = np.array([3, 4, 5, 6, 7, 8])
```

# [2] 모델 (model) 구축

```
model = Sequential() # 모델

model.add(Flatten(input_shape=(1,))) # 입력층

model.add(Dense(1, activation='linear')) # 출력층

# model.add(Dense(1, input_shape=(1,), activation='linear'))
```

# [3] 모델 (model) 컴파일 및 summary

```
model.compile(optimizer=SGD(learning_rate=1e-2), loss='mse')

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 1)	0
dense (Dense)	(None, 1)	2

Total params: 2  
Trainable params: 2  
Non-trainable params: 0

## 5. Keras-Simple LinearRegression Exercise-계속

```
# [4] 모델 학습
```

```
hist = model.fit(x_data, t_data, epochs=1000)
```

```
Epoch 1/1000
```

```
1/1 [=====] - 0s 1ms/step - loss: 27.0592
```

```
Epoch 2/1000
```

```
1/1 [=====] - 0s 2ms/step - loss: 12.8209
```

```
Epoch 3/1000
```

```
1/1 [=====] - 0s 2ms/step - loss: 6.8888
```

```
Epoch 998/1000
```

```
1/1 [=====] - 0s 2ms/step - loss: 3.8253e-04
```

```
Epoch 999/1000
```

```
1/1 [=====] - 0s 2ms/step - loss: 3.7974e-04
```

```
Epoch 1000/1000
```

```
1/1 [=====] - 0s 2ms/step - loss: 3.7697e-04
```

```
# [5] 모델 (model) 사용
```

```
result = model.predict(np.array([-3.1, 3.0, 3.5, 15.0, 20.1]))
```

```
print(result)
```

```
[[-1.1760317]
```

```
 [ 4.986803 ]
```

```
 [ 5.4919534]
```

```
 [17.110413 ]
```

```
 [22.262945 ]]
```