AiDA Lab.
Artificial Intelligence & Data Analysis Laboratory

2021 인공지능 소수전공

51~55차시: 텐서플로

2021.08.04 17:30~22:15

Seokhwan Yang

텐서플로 소개

- Google에서 개발, 공개한 딥러닝 프레임워크. 파이썬에 최적화됨
  - 엄밀하게는 다양한 작업에 대한 데이터 흐름 프로그래밍을 위한 오픈소스 심볼릭 수학 라이브러리로 개발되었지만 딥러닝 용으로도 많이 사용됨
  - 최근에는 딥러닝을 위한 대표적인 라이브러리로서 활용됨
  - 개발은 C++로 진행되었으나 파이썬, 자바, 고 등 다양한 언어를 지원하며 파이썬을 최우선으로 지원 중
  - 내부 실행은 고성능 C++ 바이너리로 제공되므로 빠른 속도와 높은 추상화 능력을 지원 가능

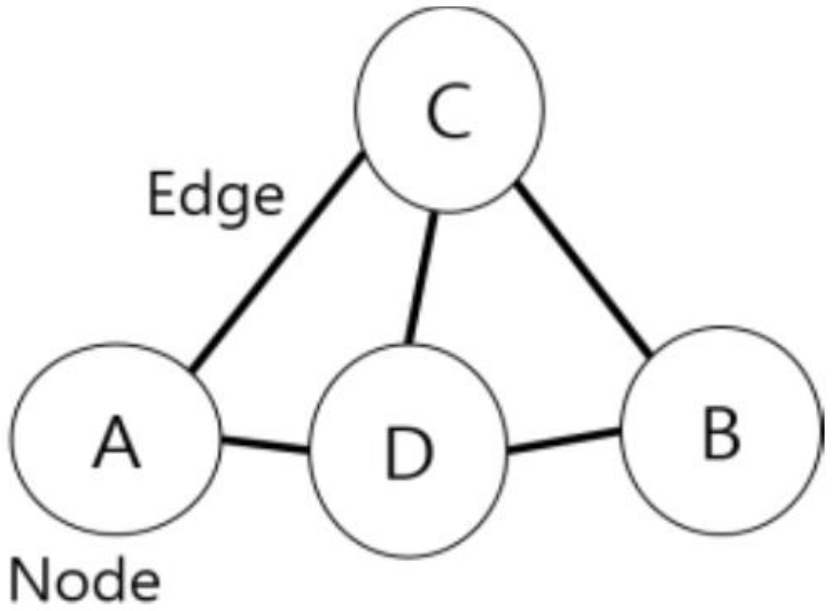
텐서플로 기본

- 자료형의 인식
  - 텐서플로는 스칼라, 벡터, 행렬과 같은 모든 데이터를 텐서(Tensor)로 인식하고 있다

데이터 값	데이터 타입
숫자 1, 2, 3...	스칼라 (또는 rank 0 텐서)
1 차원 배열 [1, 2]	벡터 (또는 rank 1 텐서)
2 차원 배열 [[1,2], [3,4]]	행렬 (또는 rank 2 텐서)
3 차원 배열 [[[1,2]], [[3,4]]]	텐서 (또는 rank 3 텐서)

텐서플로 기본

- 텐서플로의 처리구조
  - 텐서플로는 모든 처리를 그래프 구조를 기반으로 수행한다
  - 텐서플로의 기본 구조



- 원으로 표시된 4 개의 노드(Node)가 직선으로 표시된 5 개의 엣지(Edge)를 통해서 연결되어 있는 그래프(Graph) 구조

- 텐서플로는 이러한 그래프에서 엣지를 통해 텐서(Tensor)들을 노드에서 노드로 흘러(Flow)보내면서 딥러닝 학습을 진행

텐서플로 기본

- 텐서플로 코드 구현 순서

1. 상수, 변수, 텐서 연산 등의 Node와 이들을 연결해주는 Edge 정의
   (즉 값이 저장되는 노드와 이 값들이 전달되는 엣지를 먼저 정의)
2. 세션(Session)을 만들고, 그 세션을 통해 노드에 있는 데이터(텐서)가 전달되고 연산이 진행됨

텐서플로 기본

- **기본 자료형**
  - **상수형(Constant)**
    - tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False) 와 같은 형태로 정의. 각 정의되는 내용을 보면
      - value : 상수의 값
      - dtype : 상수의 데이터형. tf.float32와 같이 실수, 정수 등의 데이터 타입을 정의
      - shape : 행렬의 차원을 정의. shape=[3,3]으로 정의해주면, 이 상수는 3x3 행렬을 저장하게 됨
      - name : 상수의 이름을 정의

텐서플로 기본

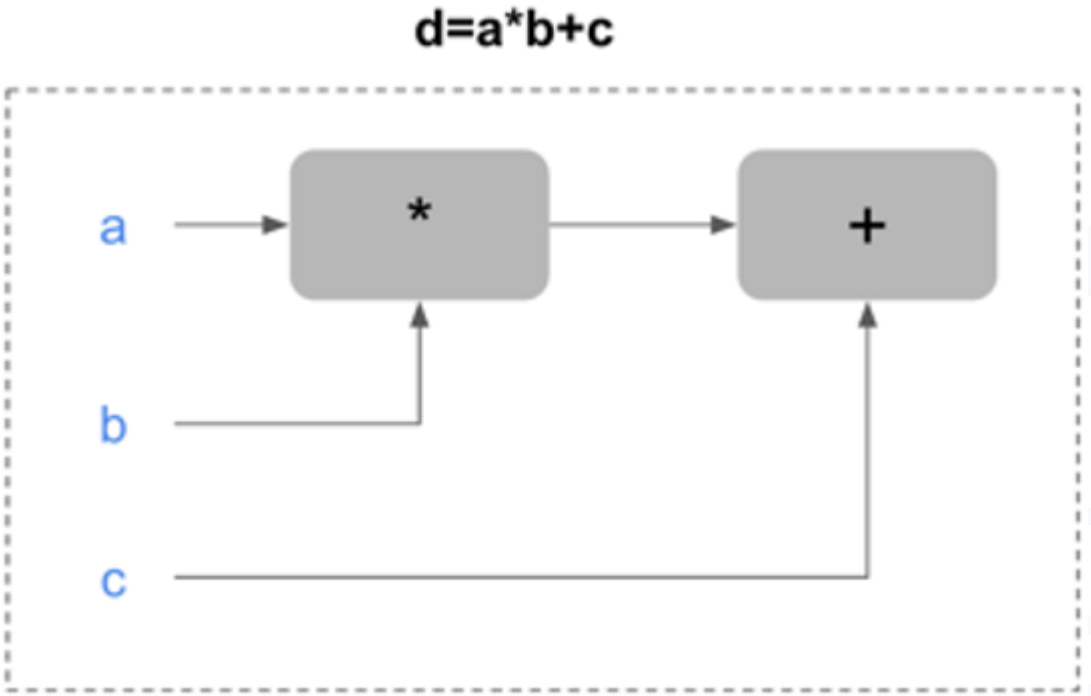
- 기본 자료형
  - 상수형(Constant) 사용 예시

```
import tensorflow as tf

a = tf.constant([5],dtype=tf.float32)
b = tf.constant([10],dtype=tf.float32)
c = tf.constant([2],dtype=tf.float32)

d = a*b+c

print(d)
```

```
tf.Tensor([52.], shape=(1,), dtype=float32)
```

텐서플로 기본

- 그래프와 세션의 개념
  - 텐서플로의 프로그래밍 모델을 이해하려면 먼저 그래프와 세션의 개념을 이해하여야 함
  - 앞 페이지 상수형 사용 예시의 d=a*b+c 에서 d는 계산을 수행하는 것이 아니라 a*b+c 그래프를 정의하는 것



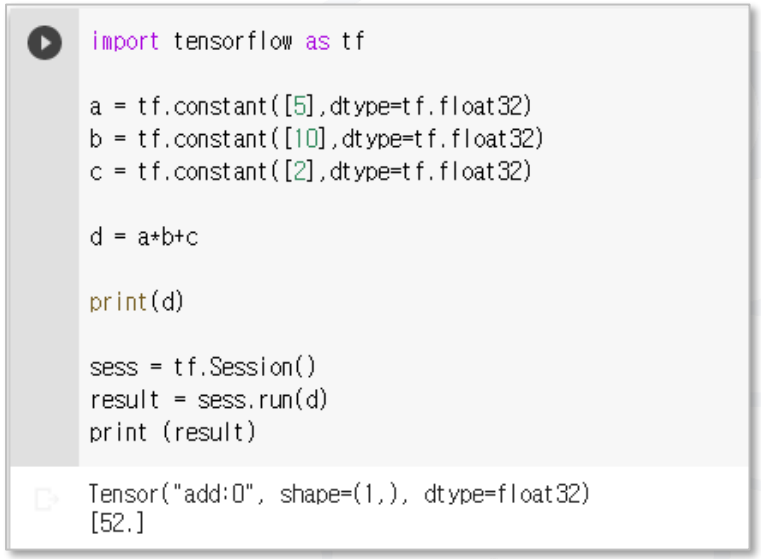
텐서플로 기본

- 실제로 값을 계산하여 얻기 위해서는 정의된 그래프에 a, b, c 값을 넣어서 실행해야 함
- 실행 방법은 먼저 세션 (Session)을 생성하고 다음으로 그래프를 실행함
- 세션: 그래프를 인자로 받아서 실행해 주는 러너(Runner)의 일종으로 볼 수 있음

텐서플로 기본

- 기본 자료형
  - 상수형(Constant) 사용 예시

```
import tensorflow as tf

a = tf.constant([5],dtype=tf.float32)
b = tf.constant([10],dtype=tf.float32)
c = tf.constant([2],dtype=tf.float32)

d = a*b+c

print(d)

sess = tf.Session()
result = sess.run(d)
print (result)
```



```
import tensorflow as tf

a = tf.constant([5],dtype=tf.float32)
b = tf.constant([10],dtype=tf.float32)
c = tf.constant([2],dtype=tf.float32)

d = a*b+c

print(d)

sess = tf.Session()
result = sess.run(d)
print (result)
```

```
Tensor("add:0", shape=(1,), dtype=float32)
[52.]
```

텐서플로 기본

- **AttributeError: module ‘tensorflow’ has no attribute ‘Session’**

```
[5]  import tensorflow as tf

     a = tf.constant([5],dtype=tf.float32)
     b = tf.constant([10],dtype=tf.float32)
     c = tf.constant([2],dtype=tf.float32)

     d = a*b+c

     print(d)

     sess = tf.Session()
     result = sess.run(d)
     print (result)
```

```
tf.Tensor([52.], shape=(1,), dtype=float32)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-a9855cfb3f73> in <module>()
      9 print(d)
     10
---> 11 sess = tf.Session()
     12 result = sess.run(d)
     13 print (result)

AttributeError: module 'tensorflow' has no attribute 'Session'
```

SEARCH STACK OVERFLOW

- 텐서플로는 1.x 버전에서 2.x 버전으로 바뀌면서 Session을 통해서 계산하는 구조를 자동으로 Eager Execution(즉시 실행 모드)을 적용함
- 이러한 문제로 1.x 버전과 일부 호환성의 문제를 일으키고 있음
- 그러나 여전히 1.x 버전의 예제 및 실무 코드가 보편화 되어 있으므로 1.x 버전을 학습할 필요가 있음
- Colab을 사용할 경우, 기본 설치된 텐서플로 2.5.x 버전을 제거하고 1.x버전을 재 설치할 필요가 있음

텐서플로 기본

- Colab에서 텐서플로 버전 변경하기
  - !pip uninstall tensorflow
  - !pip install tensorflow==1.15
  - 재설치 후에는 런타임 재시작

```
[2]  !pip uninstall tensorflow

    Found existing installation: tensorflow 2.5.0
    Uninstalling tensorflow-2.5.0:
      Would remove:
        /usr/local/bin/estimator_ckpt_converter
        /usr/local/bin/import_pb_to_tensorboard
        /usr/local/bin/saved_model_cli
        /usr/local/bin/tensorboard
        /usr/local/bin/tf_upgrade_v2
        /usr/local/bin/tflite_convert
        /usr/local/bin/toco
        /usr/local/bin/toco_from_protos
        /usr/local/lib/python3.7/dist-packages/tensorflow-2.5.0.dist-info/*
        /usr/local/lib/python3.7/dist-packages/tensorflow/*
    Proceed (y/n)? y
      Successfully uninstalled tensorflow-2.5.0

[3]  !pip install tensorflow==1.15

    Collecting tensorflow==1.15
      Downloading tensorflow-1.15.0-cp37-cp37m-manylinux2010_x86_64.whl (412.3 MB)
         |████████████████████████████████| 412.3 MB 23 kB/s
    Collecting keras-applications>=1.0.8
      Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
         |████████████████████████████████| 50 kB 6.1 MB/s
    Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (3.17.3)
    Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (1.12.1)
    Collecting tensorflow-estimator==1.15.1
      Downloading tensorflow_estimator-1.15.1-py2.py3-none-any.whl (503 kB)
         |████████████████████████████████| 503 kB 60.6 MB/s
    Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (1.15.0)
    Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (1.34.1)
    Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (0.12.0)
    Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (1.1.2)
    Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (0.8.1)
    Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow==1.15) (1.19.5)
    Collecting gast==0.2.2
      Downloading gast-0.2.2.tar.gz (10 kB)
    Collecting tensorboard<1.16.0,>=1.15.0
```

텐서플로 기본

- 기본 자료형의 처리
  - 상수형(Constant) 사용 예시

```
import tensorflow as tf

# 상수 노드 정의
a = tf.constant(1.0)
b = tf.constant(2.0)
c = tf.constant([ [1.0, 2.0], [3.0, 4.0] ])

print(a)
print(a+b)
print(c)
```

```
# 세션 (session) 을 만들고 노드간의 텐서 연산 실행
sess = tf.Session()
print(sess.run([a, b]))
print(sess.run(c))
print(sess.run([a+b]))
print(sess.run(c+1.0)) # broadcast 수행

# 세션 close
sess.close()
```

```
[3]  import tensorflow as tf

     # 상수 노드 정의
     a = tf.constant(1.0)
     b = tf.constant(2.0)
     c = tf.constant([ [1.0, 2.0], [3.0, 4.0] ])

     print(a)
     print(a+b)
     print(c)

     # 세션 (session) 을 만들고 노드간의 텐서 연산 실행
     sess = tf.Session()
     print(sess.run([a, b]))
     print(sess.run(c))
     print(sess.run([a+b]))
     print(sess.run(c+1.0)) # broadcast 수행

     # 세션 close
     sess.close()

     Tensor("Const_3:0", shape=(), dtype=float32)
     Tensor("add_1:0", shape=(), dtype=float32)
     Tensor("Const_5:0", shape=(2, 2), dtype=float32)
     [1.0, 2.0]
     [[1. 2.]
      [3. 4.]]
     [3.0]
     [[2. 3.]
      [4. 5.]]
```

텐서플로 기본

- **기본 자료형의 처리**
  - **상수형**
    - 텐서플로는 tf.constant() 함수를 이용해서 상수 값을 저장하는 노드 생성

      → a = tf.constant(1.0) → 1.0 값을 가지는 상수 노드 a 를 생성하라

      → c = tf.constant([[1.0, 2.0], [3.0, 4.0]]) → [[1.0, 2.0], [3.0, 4.0]] 값을 가지는 노드 c 를 생성하라
    - 이렇게 상수를 저장하는 노드를 정의한 후에 print 명령문을 (세션을 만들지 않고) 바로 실행시키면 Tensor("a:0", shape=(), dtype=float32) 같은 결과가 출력됨

텐서플로 기본

- **기본 자료형의 처리**
  - **상수형**
    - 텐서플로는 세션(Session)을 만들지 않고 명령문을 바로 실행하면 노드에 저장되어 있는 값이 출력되는 것이 아니라, 현재 정의 되어있는 노드의 타입과 형상(shape) 등의 현재 상태 정보만이 출력됨

      → 이러한 노드에 실제의 값을 저장하고 노드 간의 연산을 하기 위해서는 sess = tf.Session() 명령문을 이용하여 세션을 반드시 만들어 주어야 함

텐서플로 기본

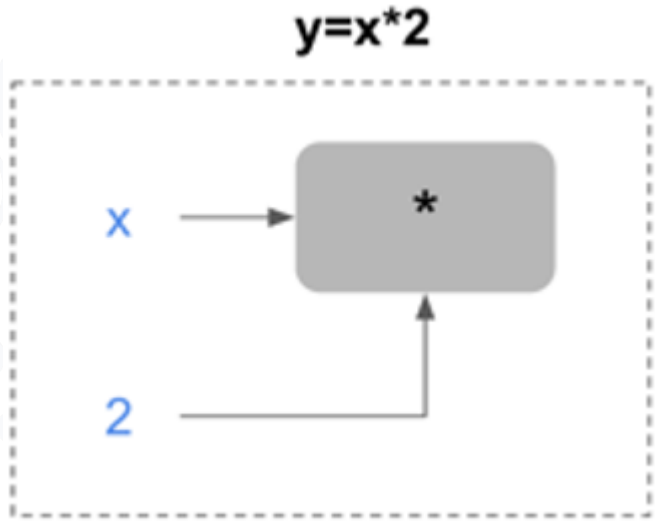
- **기본 자료형의 처리**
  - **상수형**
    - 이렇게 생성된 세션을 통해서, 즉 sess.run() 명령문을 통해서 노드에 상수 값이 할당되고 노드 간에 텐서를 흘려보내면서(tensor flow) a+b 수식과 연산과 print 명령문 등을 실
    - 생성된 세션에서 모든 연산을 마쳤다면 세션을 닫아주는 sess.close() 명령문을 실행시켜서 시스템자원(Resource)을 해제 시켜 주어야 함

텐서플로 기본

- **기본 자료형의 처리**
  - **플레이스 홀더(Placeholder)**
    - 학습용 데이터를 담는 그릇의 역할
    - tf.placeholder(dtype,shape,name) 의 형태로 정의
    - 플레이스 홀더 정의에 사용되는 변수들을 보면
      - dtype : 플레이스홀더에 저장되는 데이터형. tf.float32와 같이 실수, 정수 등의 데이터 타입을 정의
      - shape : 행렬의 차원을 정의. shapre=[3,3]으로 정의하면, 이 플레이스 홀더는 3x3 행렬을 저장
      - name : 플레이스 홀더의 이름을 정의

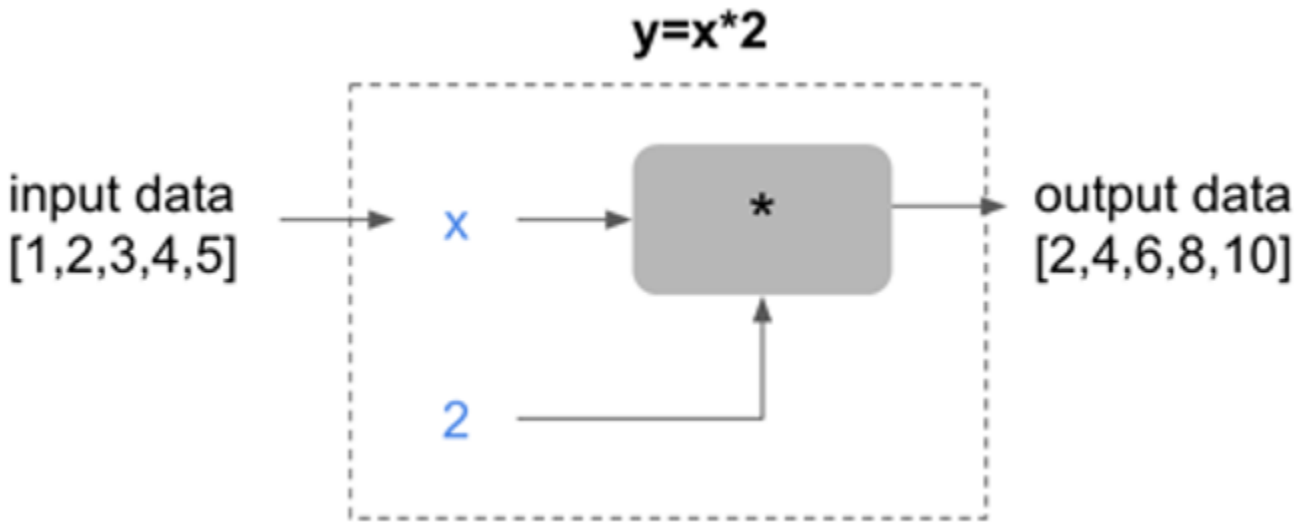
텐서플로 기본

- 기본 자료형의 처리
  - 플레이스 홀더(Placeholder)
    - y = x * 2 를 그래프를 통해서 실행하려면
      - 입력 값으로는 1,2,3,4,5를 넣고, 출력은 2,4,6,8,10을 기대함
    - 여러 입력값을 그래프에서 넣는 경우
      - 머신러닝에서 y=W*x + b 와 같은 그래프가 있다고 할 때,
      - x는 학습을 위한 데이터가 됨
    - y=x*2를 정의하면 내부적으로 오른쪽과 같은 그래프가 됨



텐서플로 기본

- 기본 자료형의 처리
  - 플레이스 홀더(Placeholder)

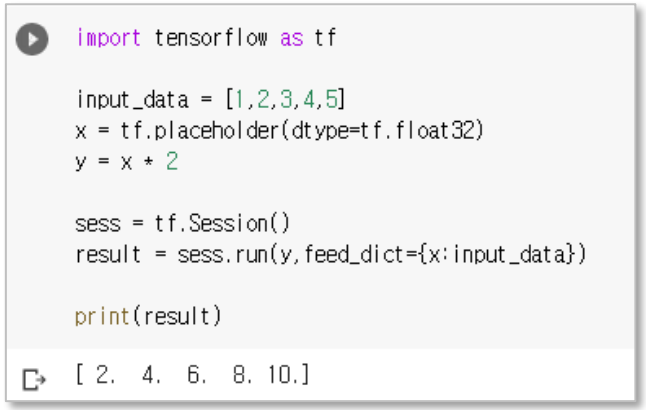


  - x에 학습용 데이터를 넣는 것을 피딩(feeding)이라고 함

텐서플로 기본

- 기본 자료형의 처리
  - 플레이스 홀더(Placeholder)

```
import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
y = x * 2

sess = tf.Session()
result = sess.run(y,feed_dict={x:input_data})

print(result)
```



- input_data=[1,2,3,4,5]으로 정의하고, 다음으로 x=tf.placeholder(dtype=tf.float32) 를 이용하여, x를 float32 데이터형을 가지는 플레이스 홀더로 정의함
- 그리고 y=x * 2 로 그래프를 정의함

- 세션이 실행될 때, x라는 통에 값을 하나씩 집어넣음
- sess.run(y,feed_dict={x:input_data}) 와 같이 세션을 통해서 그래프를 실행
- 이 때, feed_dict 변수를 이용해서 플레이스 홀더 x에, input_data를 피드하면
- 세션에 의해서 그래프가 실행되면서 x는 feed_dict에 의해서 정해진 피드 데이터 [1,2,3,4,5]를 하나씩 읽어서 실행

텐서플로 기본

- **기본 자료형의 처리**
  - **변수형(Variable)**
    - Variable 형의 객체로 생성됨

```
tf.Variable.__init__(initial_value=None, trainable=True, collections=None,
validate_shape=True, caching_device=None, name=None, variable_def=None,
dtype=None, expected_shape=None, import_scope=None)
```

    - 변수에 값을 넣는 방법

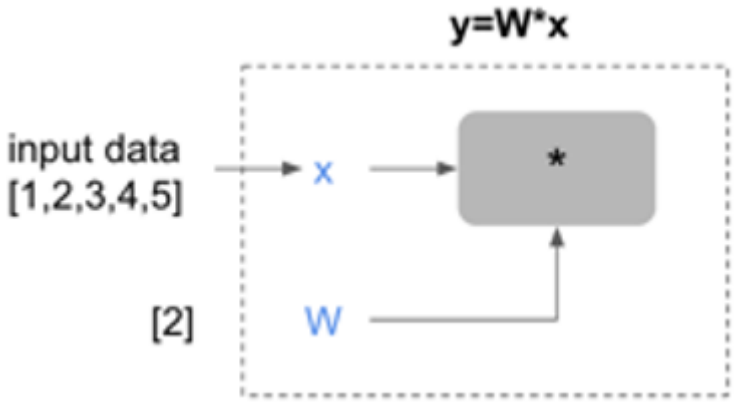
```
var = tf.Variable([1,2,3,4,5], dtype=tf.float32)
```

텐서플로 기본

- 기본 자료형의 처리
  - 변수형(Variable)

```
import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
y = x * 2

sess = tf.Session()
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

- 기대하는 결과



- 실행해 보면

텐서플로 기본

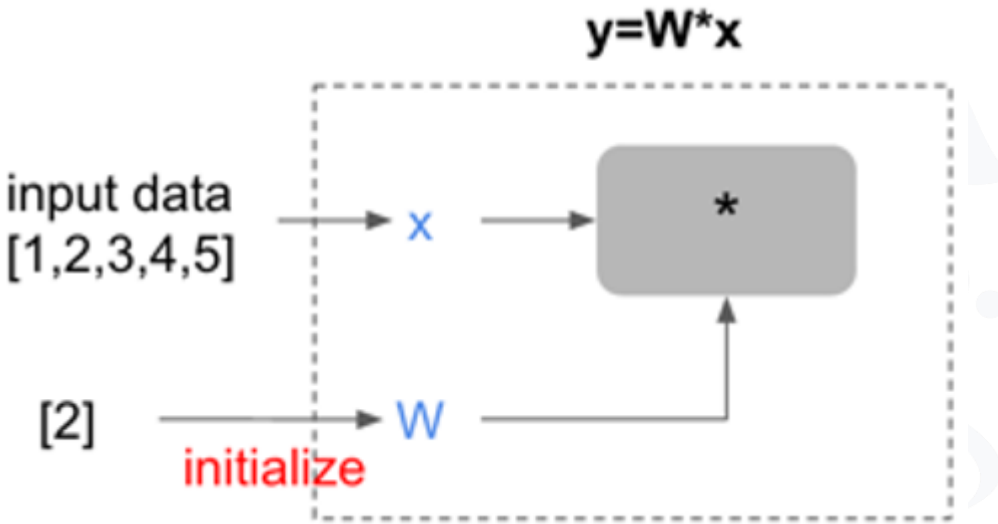
- 기본 자료형의 처리
  - 변수형(Variable)

```
[5]  import tensorflow as tf

     input_data = [1,2,3,4,5]
     x = tf.placeholder(dtype=tf.float32)
     W = tf.Variable([2],dtype=tf.float32)
     y = W*x

     sess = tf.Session()
     result = sess.run(y,feed_dict={x:input_data})

     print(result)
```

```
---------------------------------------------------------------------------
FailedPreconditionError                   Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/tensorflow_core/python/client/session.py in _do_call(self, fn, *args)
   1364     try:
-> 1365       return fn(*args)
   1366     except errors.OpError as e:

                                  6 frames
FailedPreconditionError: Attempting to use uninitialized value Variable
         [[{{node Variable/read}}]]

During handling of the above exception, another exception occurred:

FailedPreconditionError                   Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/tensorflow_core/python/client/session.py in _do_call(self, fn, *args)
   1382                    '\nsession_config.graph_options.rewrite_options.'
   1383                    'disable_meta_optimizer = True')
-> 1384       raise type(e)(node_def, op, message)
   1385
   1386   def _extend_graph(self):
```

텐서플로 기본

- 기본 자료형의 처리
  - 변수형(Variable)
    - 텐서플로에서 변수형은 그래프를 실행하기 전에 초기화 해줘야 그 값이 변수에 지정됨
    - 세션을 초기화 하는 순간 변수 W에 값이 지정됨
    - 초기화 방법
      - 변수들을 global_variables_initializer() 를 이용해서 초기화 한 후
      - 초기화된 결과를 세션에 전달해 주어야 함



텐서플로 기본

- 기본 자료형의 처리
  - 변수형(Variable)

```
import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
W = tf.Variable([2],dtype=tf.float32)
y = W*x

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

```
[6]  import tensorflow as tf

     input_data = [1,2,3,4,5]
     x = tf.placeholder(dtype=tf.float32)
     W = tf.Variable([2],dtype=tf.float32)
     y = W*x

     sess = tf.Session()
     init = tf.global_variables_initializer()
     sess.run(init)
     result = sess.run(y,feed_dict={x:input_data})

     print(result)

     [ 2.  4.  6.  8. 10.]
```

텐서플로 기본

- 텐서플로를 처음 시작할 때
  - Optimizer나 모델 등에 대해 이해하는 것도 중요하지만
  - “데이터를 가지고 학습을 시켜서 적정한 값을 찾는다“ 라는 머신러닝 학습 모델의 특성에 따라
  - 모델을 그래프로 정의하고, 세션을 만들어서 그래프를 실행하고, 세션이 실행될 때 그래프에 동적으로 값을 넣어가면서 (피딩) 실행한다는 기본 개념을 잘 이해해야
  - 텐서플로 프로그래밍을 제대로 시작할 수 있음

행렬과 텐서플로

- 텐서플로에서 행렬의 표현
  - 상수형 행렬

$$\begin{pmatrix} 1.0 & 2.0 & 3.0 \end{pmatrix} \times \begin{pmatrix} 2.0 \\ 2.0 \\ 2.0 \end{pmatrix}$$

```
import tensorflow as tf

input_data = [1,2,3,4,5]
x = tf.placeholder(dtype=tf.float32)
W = tf.Variable([2],dtype=tf.float32)
y = W*x

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

```
[8]  import tensorflow as tf

     x = tf.constant([ [1.0,2.0,3.0] ])
     w = tf.constant([ [2.0],[2.0],[2.0] ])
     y = tf.matmul(x,w)
     print(x.get_shape())

     sess = tf.Session()
     init = tf.global_variables_initializer()
     sess.run(init)
     result = sess.run(y)

     print(result)

     (1, 3)
     [[12.]]
```

- 텐서플로에서 행렬의 곱셈은 *이 아니라 tf.matmul을 사용함

행렬과 텐서플로

- 텐서플로에서 행렬의 표현
  - 변수형 행렬

```
import tensorflow as tf

x = tf.Variable([ [1.,2.,3.] ], dtype=tf.float32)
w = tf.constant([ [2.],[2.],[2.]], dtype=tf.float32)
y = tf.matmul(x,w)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y)

print(result)
```

```
[9]  import tensorflow as tf

     x = tf.Variable([ [1.,2.,3.] ], dtype=tf.float32)
     w = tf.constant([ [2.],[2.],[2.]], dtype=tf.float32)
     y = tf.matmul(x,w)

     sess = tf.Session()
     init = tf.global_variables_initializer()
     sess.run(init)
     result = sess.run(y)

     print(result)

     [[12.]]
```

행렬과 텐서플로

- 텐서플로에서 행렬의 표현
  - 플레이스홀더에도 행렬로 저장 가능
    - 예: 입력 데이터 행렬 x는 PlaceHolder 타입인 3x3 행렬이고 곱하는 값 w는 1x3인 행렬

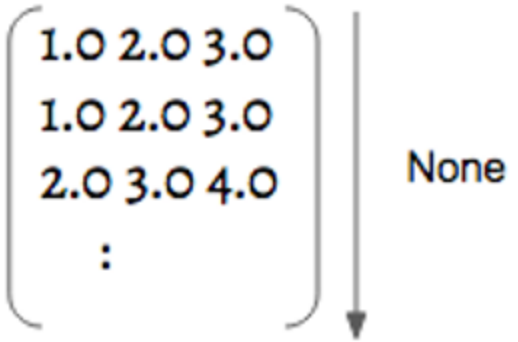
$$\begin{pmatrix} 1.0 & 2.0 & 3.0 \\ 1.0 & 2.0 & 3.0 \\ 2.0 & 3.0 & 4.0 \end{pmatrix} \times \begin{pmatrix} 2.0 \\ 2.0 \\ 2.0 \end{pmatrix}$$

```
import tensorflow as tf

input_data = [ [1.,2.,3.],[1.,2.,3.],[2.,3.,4.]
x = tf.placeholder(dtype=tf.float32,shape=[None,3])
w = tf.Variable([ [2.],[2.],[2.] ], dtype = tf.float32) y =
tf.matmul(x,w)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})

print(result)
```

```
[10]  import tensorflow as tf

      input_data = [ [1.,2.,3.],[1.,2.,3.],[2.,3.,4.] ] #3x3 matrix
      x = tf.placeholder(dtype=tf.float32,shape=[None,3])
      w = tf.Variable([ [2.],[2.],[2.] ], dtype = tf.float32) #3x1 matrix
      y = tf.matmul(x,w)

      sess = tf.Session()
      init = tf.global_variables_initializer()
      sess.run(init)
      result = sess.run(y,feed_dict={x:input_data})

      print(result)

      [[12.]
       [12.]
       [18.]]
```

행렬과 텐서플로

- **텐서플로에서 행렬의 표현**
  - **플레이스홀더 행렬**
    - placeholder x 를 정의할 때, shape=[None,3] 으로 정의함
    - None 이란, 갯수를 알수 없음을 의미
    - 텐서플로 머신러닝 학습에서 학습 데이터가 계속해서 들어오고
    - 학습 때마다 데이터의 양이 다를 수 있기 때문에
    - 이를 지정하하지 않고 None으로 해 놓으면
    - 들어오는 숫자 데이터에 맞춰서 저장함



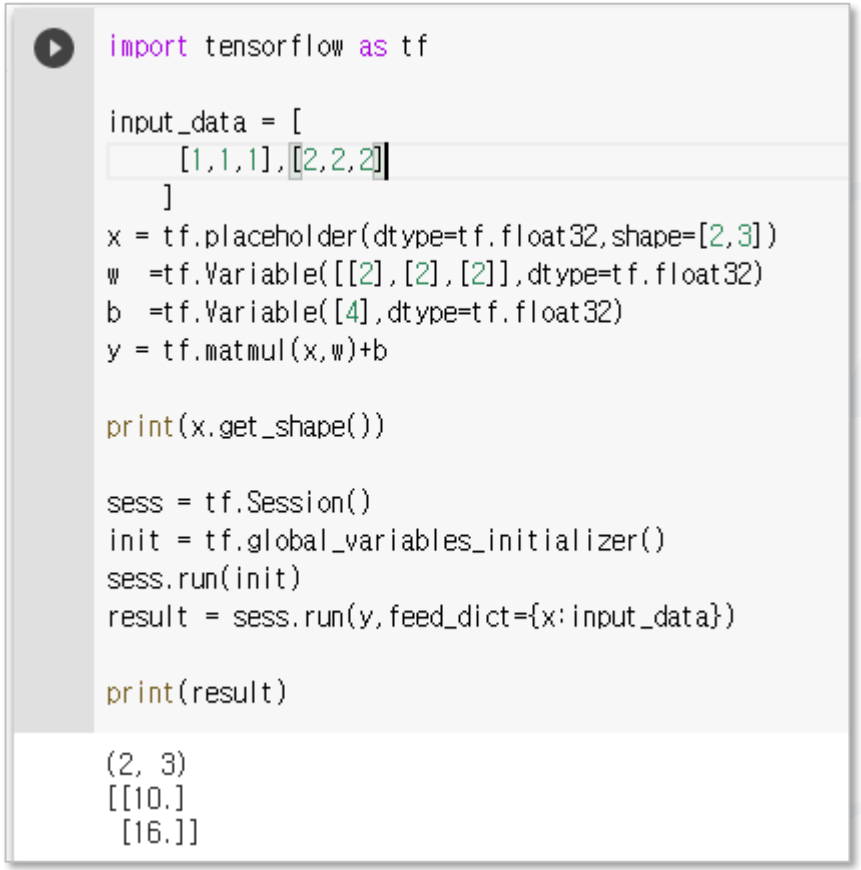
행렬과 텐서플로

- 브로드 캐스팅
  - 행렬 연산 (덧셈,뺄셈,곱셈)에서 차원이 맞지 않을 때, 행렬을 자동으로 늘려 줘서(Stretch) 차원을 맞춰주는 개념
  - 늘리는 것은 가능하지만 줄이는 것은 불가능함

행렬과 텐서플로

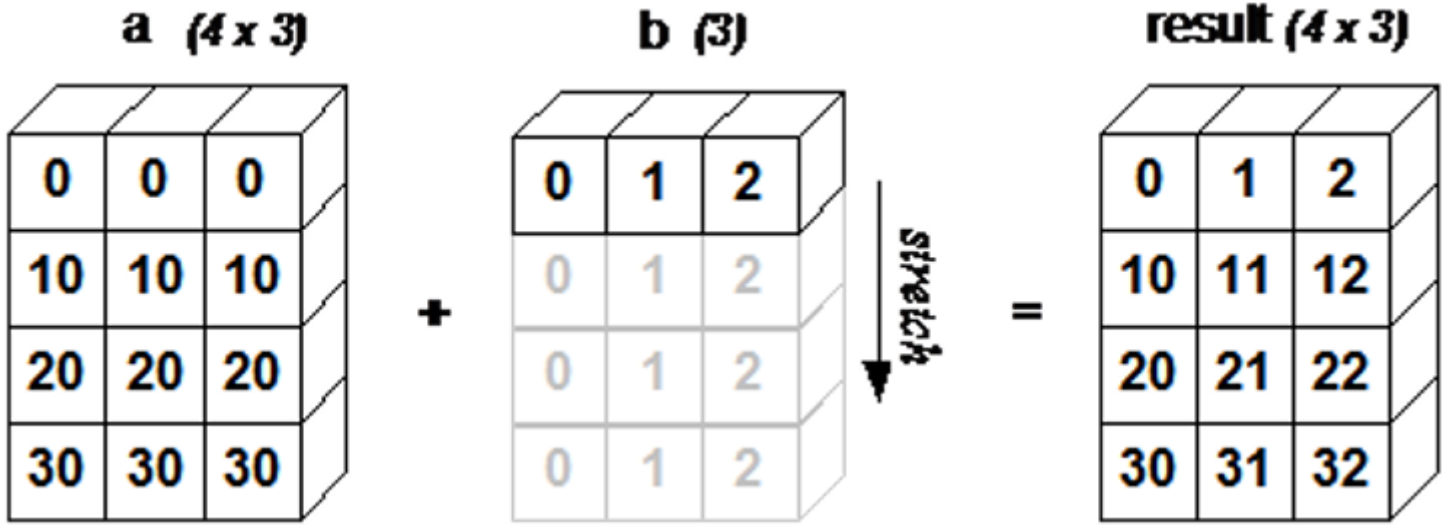
- 브로드 캐스팅

```
import tensorflow as tf

input_data = [
     [1,1,1],[2,2,2]
    ]
x = tf.placeholder(dtype=tf.float32,shape=[2,3])
w   =tf.Variable([[2],[2],[2]],dtype=tf.float32)
b   =tf.Variable([4],dtype=tf.float32)
y = tf.matmul(x,w)+b

print(x.get_shape())

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
result = sess.run(y,feed_dict={x:input_data})

print(result)
```



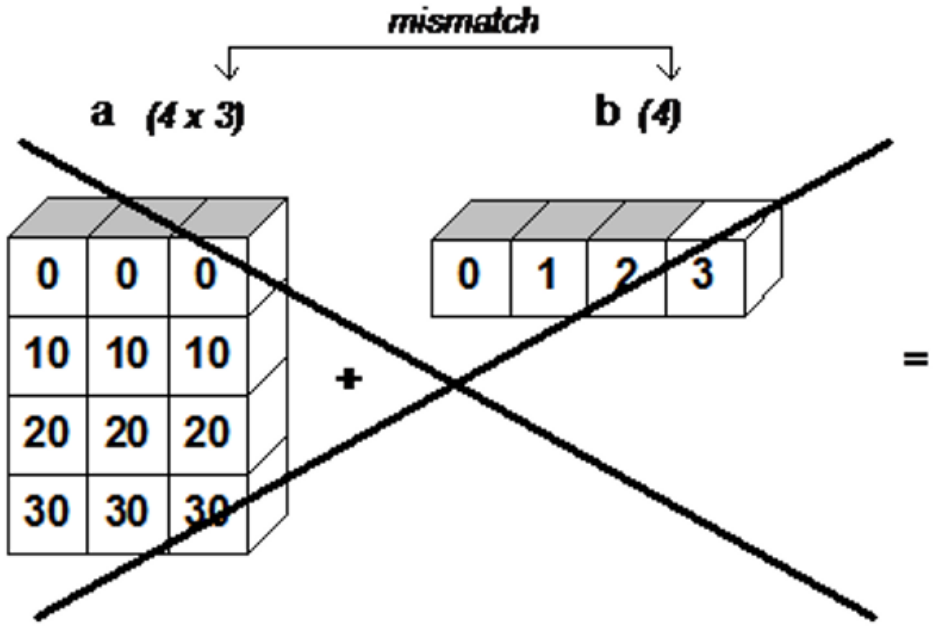
행렬과 텐서플로

- 브로드 캐스팅
  - 4x3 행렬 a와 1x3 행렬 b를 더하는 연산의 예
    - 차원이 맞지 않기 때문에, 행렬 b의 열을 늘려서 1x3 → 4x3 으로 맞춰서 연산함



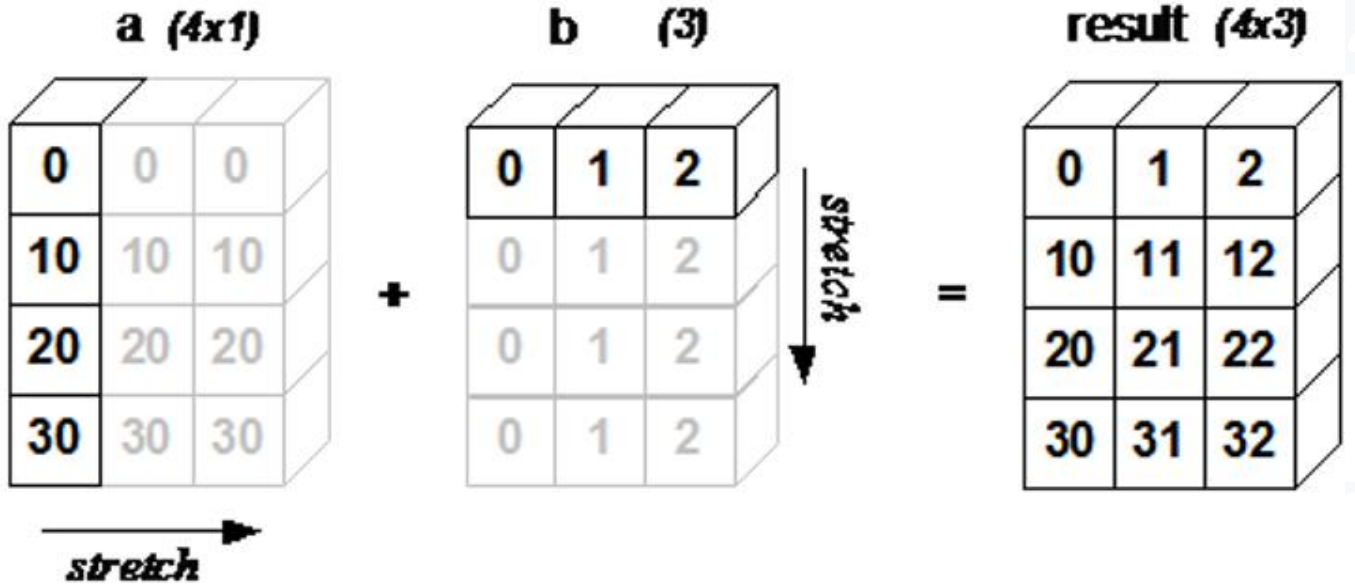
행렬과 텐서플로

- **브로드 캐스팅**
  - **행렬 b가 아래 그림과 같이 1x4 일 경우**
    - 열을 4 → 3으로 줄이고, 세로 행을 1→ 4 로 늘려야 하는데, 앞에서 언급한바와 같이, 브로드 캐스팅은 행이나 열을 줄이는 것은 불가능함



행렬과 텐서플로

- 브로드 캐스팅
  - 양쪽 행렬을 둘 다 늘린 예
    - 4x1 행렬 a와 1x3 행렬 b를 더하면 양쪽을 다 수용할 수 있는 큰 차원인 4x3 행렬로 변환하여 덧셈을 수행



행렬과 텐서플로

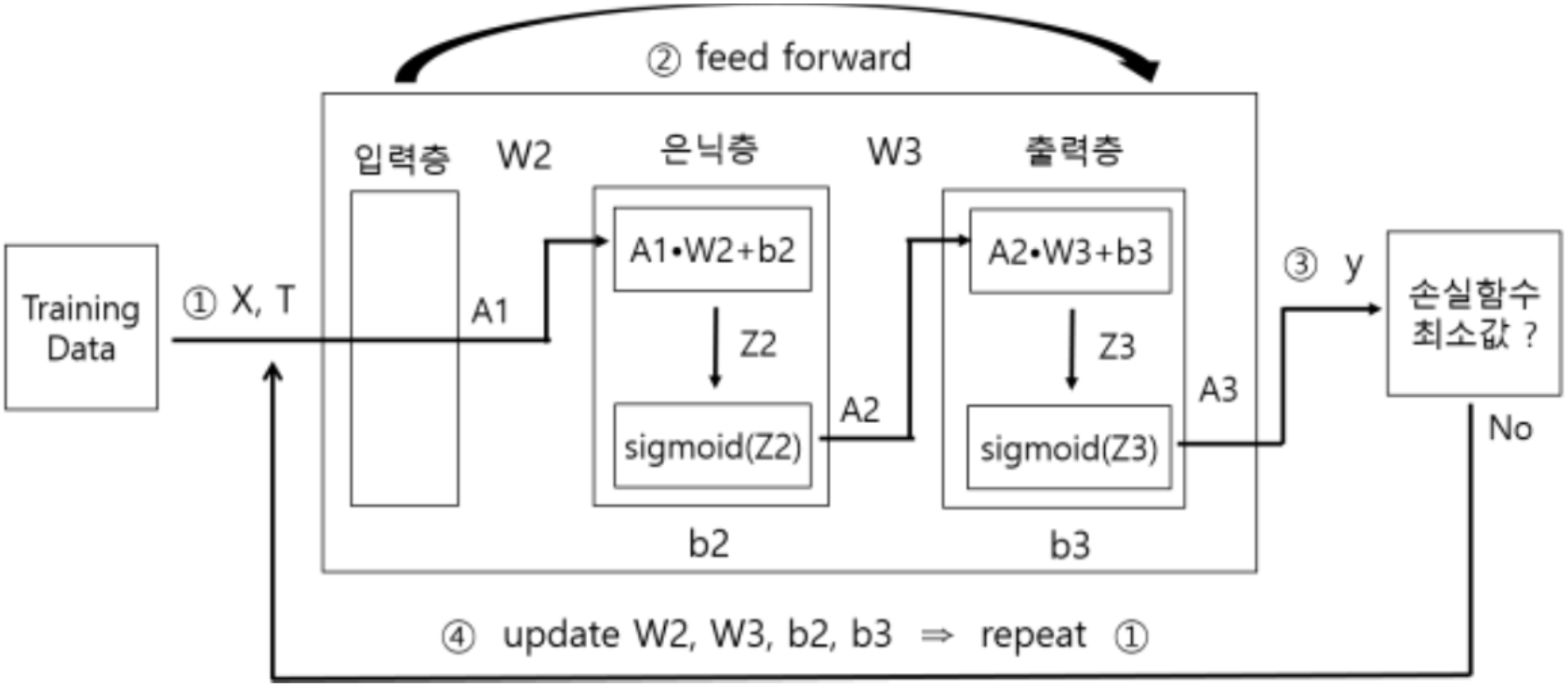
- 텐서플로 용어 참고
  - 행렬이 아닌 숫자나 상수는 Scalar, 1차원 행렬을 Vector, 2차원 행렬을 Matrix, 3차원 행렬을 3-Tensor 또는 cube, 그리고 이 이상의 다차원 행렬을 N-Tensor

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

  - 행렬의 차원은 Rank. scalar는 Rank가 0, Vector는 Rank 가 1, Matrix는 Rank가 2

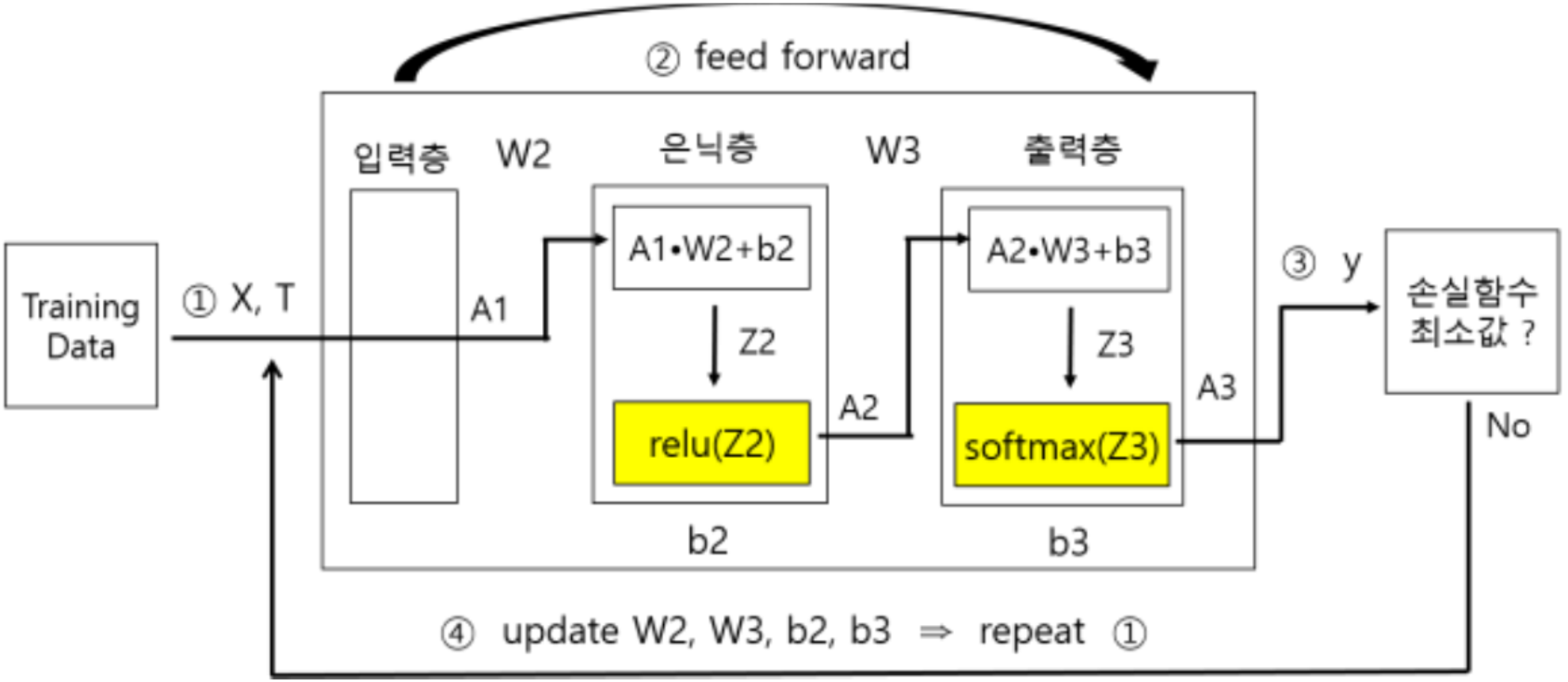
텐서플로를 이용한 MNIST

- 딥러닝 기본 아키텍처



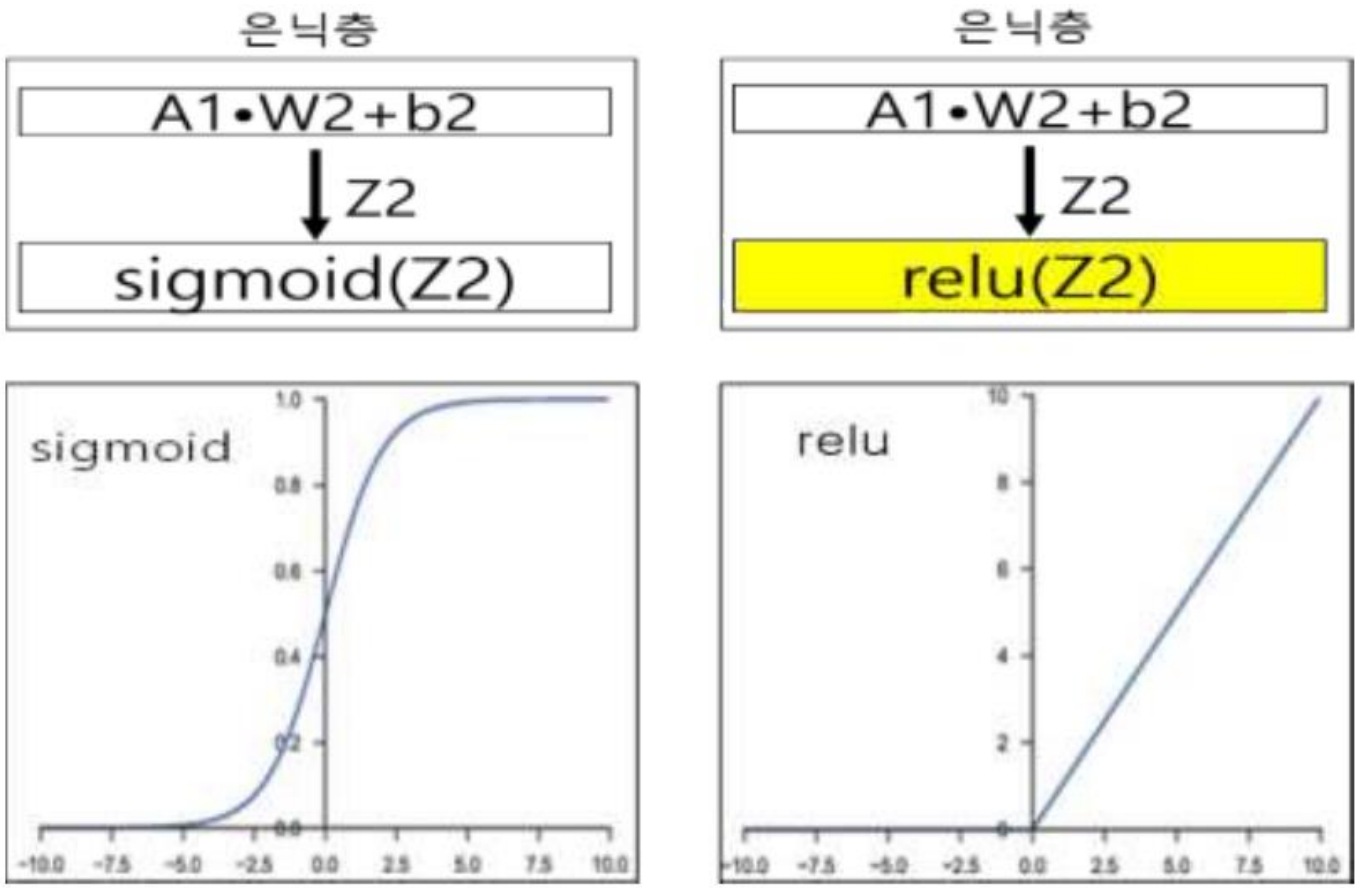
텐서플로를 이용한 MNIST

- 텐서플로에서의 딥러닝 아키텍처



텐서플로를 이용한 MNIST

- **Sigmoid 함수 vs ReLU 함수**



속도가 빠르다

텐서플로를 이용한 MNIST

- **Softmax 함수**

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}}$$

- 분모 부분에서는 모든 입력 값을 더하고 있으며
- 분자 부분에서는 특정한 하나의 입력 값만 나타남

출력층

index	0	1	2	3	4	5	6	7	8	9
z3	0.1	0.5	1.0	1.5	0.1	2.1	1.1	0.9	1.8	0.2

↓

softmax

↓

index	0	1	2	3	4	5	6	7	8	9
softmax(Z3)	0.03	0.05	0.09	0.14	0.03	0.26	0.09	0.08	0.19	0.04

정답이 5일 확률이 26%

- 입력 값의 대소(大小) 순서가 출력 값의 대소 순서와 같음
- 각각의 입력 값에 대한 Softmax 함수의 출력은 0~1 사이
- 각각의 입력에 대한 Softmax 출력 값들을 모두 더하면 1

  → 출력 값들의 총합이 1 이라는 것은 각각의 출력 값이 확률(probability)을 나타낸다고 해석 가능

텐서플로를 이용한 MNIST

- 실습 코드

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np

filepath = '{}/{}/{}/'.format(ROOT_PATH, 'data', 'mnist')
mnist = input_data.read_data_sets(filepath, one_hot=True)

print("₩ntrain image shape = ", np.shape(mnist.train.images))
print("train label shape = ", np.shape(mnist.train.labels))
print("test image shape = ", np.shape(mnist.test.images))
print("test label shape = ", np.shape(mnist.test.labels))

# 신경망 구조 및 하이퍼 파라미터 설정
learning_rate = 0.1 # 학습율
epochs = 100 # 반복횟수
batch_size = 100 # 한번에 입력으로 주어지는 MNIST 데이터 개수
input_nodes = 784 # 입력 층 노드 개수
hidden_nodes = 100 # 은닉 층 노드 개수
output_nodes = 10 # 출력 층 노드 개수
```

```
# 입력과 출력을 위한 플레이스홀더(placeholder) 정의
X = tf.placeholder(tf.float32, [None, input_nodes])
T = tf.placeholder(tf.float32, [None, output_nodes])

# 가중치와 바이어스 노드 정의
W2 = tf.Variable(tf.random_normal([input_nodes, hidden_nodes]))
b2 = tf.Variable(tf.random_normal([hidden_nodes]))
W3 = tf.Variable(tf.random_normal([hidden_nodes, output_nodes]))
b3 = tf.Variable(tf.random_normal([output_nodes]))

# 피드포워드 수행 노드 정의
Z2 = tf.matmul(X, W2) + b2 # 은닉층 선형회귀 값 Z2
A2 = tf.nn.relu(Z2) # 은닉층 출력 값 A2
Z3 = logits = tf.matmul(A2, W3) + b3 # 출력층 선형회귀 값 Z3
y = A3 = tf.nn.softmax(Z3) # 출력층 출력 값 A3
```

텐서플로를 이용한 MNIST

- 실습 코드

```
# 손실함수 계산 및 가중치, 바이어스 업데이트
cross_entropy =
tf.nn.softmax_cross_entropy_with_logits_v2(logits=Z3, labels=T)
loss = tf.reduce_mean(cross_entropy)
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
train = optimizer.minimize(loss)

# 정확도 검증 노드
predicted_val = tf.equal( tf.argmax(A3, 1), tf.argmax(T, 1) )
accuracy = tf.reduce_mean(tf.cast(predicted_val, dtype=tf.float32))
```

```
# MNIST 인식 정확도 검증
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for I in range(epochs):
        total_batch = int(mnist.train.num_examples / batch_size) # 55,000 / 100

        for step in range(total_batch):
            batch_x, batch_t = mnist.train.next_batch(batch_size)
            loss_val, _ = sess.run([loss, train], feed_dict={X: batch_x, T: batch_t})

            if step % 100 == 0:
            print("step = ", step, " loss_val = ", loss_val)

        # Accuracy 확인
        test_x_data = mnist.test.images
        test_t_data = mnist.test.labels
        accuracy_val = sess.run(accuracy, feed_dict={X: test_x_data, T: test_t_data})
        print("₩nAccuracy = ", accuracy_val)
```

텐서플로를 이용한 MNIST

- 실습 코드

```
step =  0 , loss_val =  102.172585
step =  100 , loss_val =  3.9716425
step =  200 , loss_val =  3.0084152
step =  300 , loss_val =  2.6459744
step =  400 , loss_val =  1.0566043
step =  500 , loss_val =  2.128511

Accuracy =  0.8597
step =  0 , loss_val =  0.39965266
step =  100 , loss_val =  1.6609597
step =  200 , loss_val =  0.46147424
step =  300 , loss_val =  0.85154253
step =  400 , loss_val =  1.1934104
step =  500 , loss_val =  0.85407317

Accuracy =  0.8693
step =  0 , loss_val =  0.9349872
```

▪▪▪▪

```
step =  500 , loss_val =  0.09209129

Accuracy =  0.9504
step =  0 , loss_val =  0.058875456
step =  100 , loss_val =  0.013295699
step =  200 , loss_val =  0.09775752
step =  300 , loss_val =  0.16363683
step =  400 , loss_val =  0.048410796
step =  500 , loss_val =  0.05772085

Accuracy =  0.9509
step =  0 , loss_val =  0.06759764
step =  100 , loss_val =  0.044439774
step =  200 , loss_val =  0.07914443
step =  300 , loss_val =  0.013826843
step =  400 , loss_val =  0.05356615
step =  500 , loss_val =  0.04511414

Accuracy =  0.9494
step =  0 , loss_val =  0.05250498
```

▪▪▪▪

```
Accuracy =  0.9506
step =  0 , loss_val =  0.057981644
step =  100 , loss_val =  0.07193121
step =  200 , loss_val =  0.03811815
step =  300 , loss_val =  0.088065736
step =  400 , loss_val =  0.105421096
step =  500 , loss_val =  0.057689935

Accuracy =  0.9486
step =  0 , loss_val =  0.14923517
step =  100 , loss_val =  0.18706508
step =  200 , loss_val =  0.010605138
step =  300 , loss_val =  0.032077584
step =  400 , loss_val =  0.21106303
step =  500 , loss_val =  0.09802035

Accuracy =  0.9484
```

epoch =100 이므로 100번 반복

텐서플로 실습 문제

- MNIST에서 accuracy_index, predicted_list 값과 개수를 출력하고, 학습을 마친 후 가중치 (W2, W3) / 바이어스 (b2, b3) 값을 출력하시오

텐서플로 2.x 소개

- **텐서플로 2.x**
  - **텐서플로 2.0 버전은 2019 년 9 월 30 일에 전 세계에 정식으로 배포**
  - **즉시 실행 모드라고 불리는 Eager Execution 모드가 적용되어서 코드의 직관성이 매우 높아짐**
  - **사용자 친화적이어서 개발자들이 쉽게 배울 수 있는 케라스(Keras)만을 High-Level API 로 공식적으로 지원**
    - 기존의 강력한 텐서플로는 백 엔드(Back End) 엔진으로 사용
    - 케라스는 프론트 엔드 (Front End) 로 사용

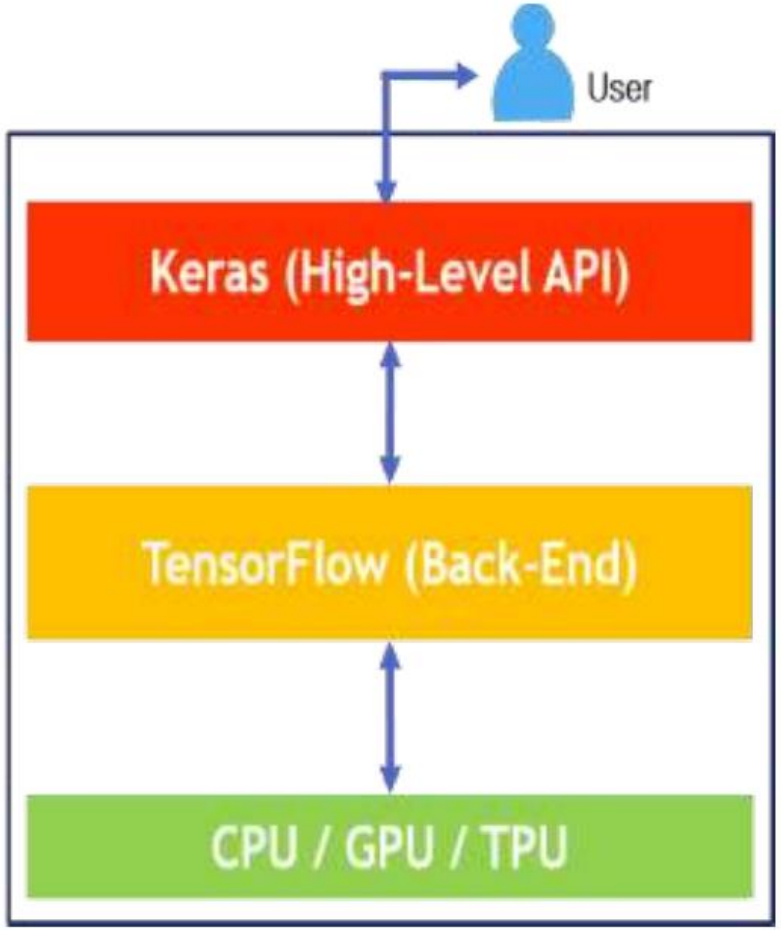
텐서플로 2.x 소개

- 텐서플로 2.0 주요 특징

Eager Execution

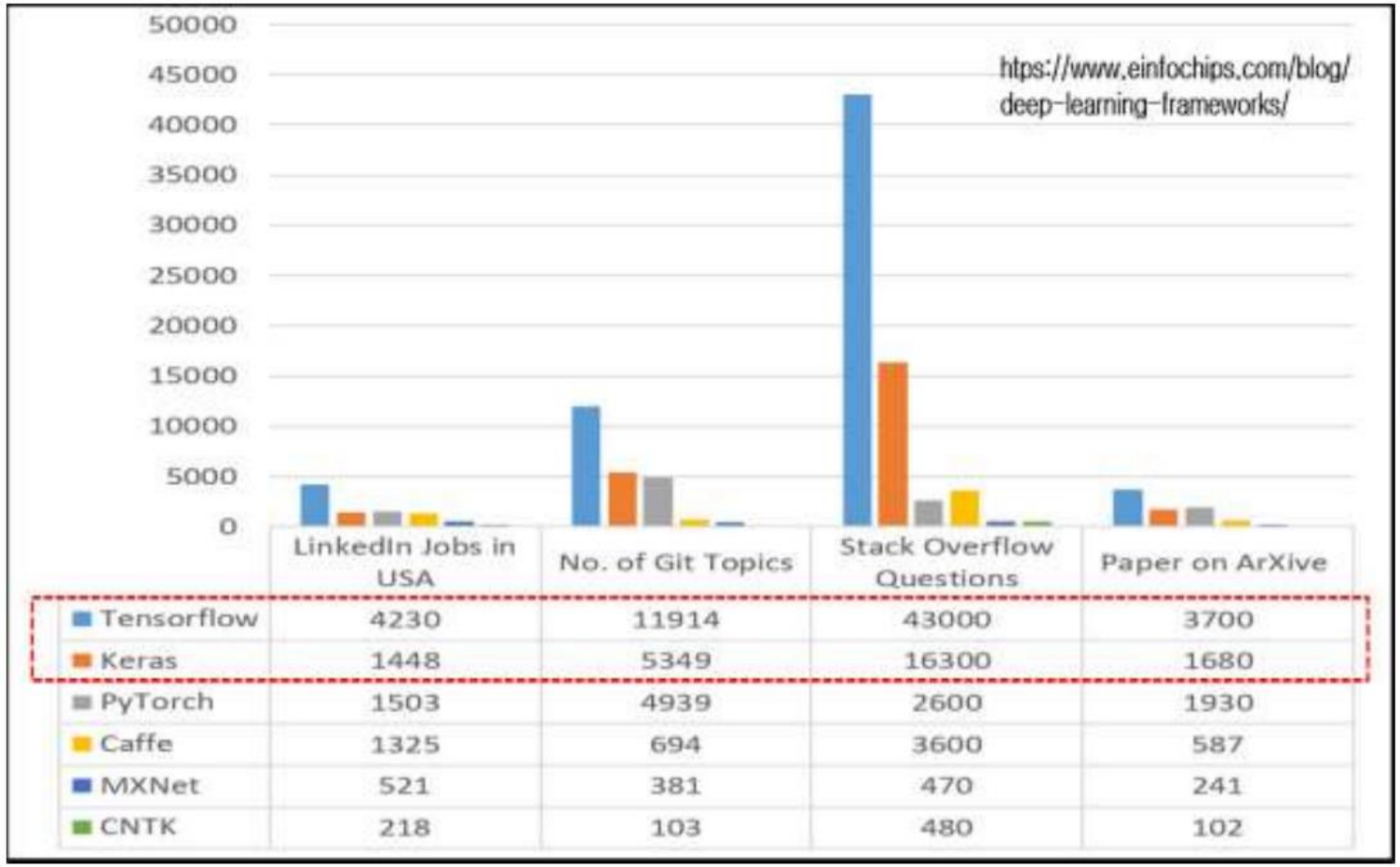
```
W = tf.Variable(tf.random.normal([1]))  # 가우시안 분포
print('initial W = ', W.numpy())
print('==========================================')

# 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]
```



텐서플로 2.x 소개

- 딥러닝 프레임워크 현황(2020년 12월 기준)



	LinkedIn Jobs in USA	No. of Git Topics	Stack Overflow Questions	Paper on ArXive
Tensorflow	4230	11914	43000	3700
Keras	1448	5349	16300	1680
PyTorch	1503	4939	2600	1930
Caffe	1325	694	3600	587
MXNet	521	381	470	241
CNTK	218	103	480	102

텐서플로 2.x

- **즉시 실행 모드(Eager Execution Mode)**
  - **기존의 텐서플로 1.x 에서는**
    - 계산 그래프(Computational Graph)와 세션(Session)을 생성하는 방식인 Lazy Evaluation 방식으로 코드 작성
  - **텐서플로 2.x에서는**
    - 코드가 순차적으로 실행되는, 즉 코드가 작성된 순서대로 즉시 실행 가능한 Eager Execution Mode 가 기본으로 적용됨

"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

텐서플로 2.x

- **즉시 실행 모드(Eager Execution Mode)**

```
import tensorflow as tf
import numpy as np

tf.__version__
```

Eager Execution

```
'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)
```

numpy() 메서드는 numpy 값을 리턴

tf.convert_to_tensor() 메서드는 numpy 값을 tensor 값으로 변환

```
<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)
```

- 자동으로 Eager Execution Mode(즉시 실행 모드)가 적용되기 때문에 그래프와 세션을 만들지 않아도 텐서 값을 계산해 줌
- Eager Execution 모드로 계산되어 있는 텐서는 언제라도 numpy() 함수를 이용하여 파이썬의 넘파이 타입으로 변환할 수 있음

텐서플로 2.x

- 상수 노드(tf.constant())의 처리

TensorFlow 1.15

```
%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))
```

세션을 만들지 않고 print와 같은 명령문 실행하면, 저장된 값이 아닌 현재 정의되어 있는 노드의 상태 출력됨

```
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
```

텐서플로우 2.x 에서는 세션을 생성하지 않고,

오퍼레이션을 실행하는 순간 연산이 수행되기 때문에 오퍼레이션 실행결과를 numpy 값으로 즉시 알 수 있음. 즉 모든 코드는 쓰여진 순서에 따라 즉시 실행되는 Eager Execution으로 인해서 결과를 바로 확인 할 수 있음

텐서플로 2.x

- 변수 노드(tf.Variable())의 처리

TensorFlow 1.15

```
%tensorflow_version 1.x

import tensorflow as tf

print(tf.__version__)

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))
```

변수 W 초기화 코드는 세션 내에서 반드시 실행되어야 함

```
TensorFlow 1.x selected.
1.15.2
<tf.Variable 'Variable:0' shape=(1,) dtype=float32_ref>
==========
step =  0 , W =  [2.2812839]
step =  1 , W =  [3.2812839]
```

TensorFlow 2.x

```
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())
```

변수 W 초기화 없이 즉시 실행 (Eager Execution)

```
initial W =  [0.588869]
==========
step =  0 , W =  [1.588869]
step =  1 , W =  [2.588869]
```

텐서플로 2.x

- **tf.placeholder() 삭제**

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 세션내에서 입력받기 받는 용도로 사용됨 (Lazy Evalution)

플레이스홀더 노드에 대입되는 값

```
tensorflow version =  1.15.2
==========================================
[4.]
```

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

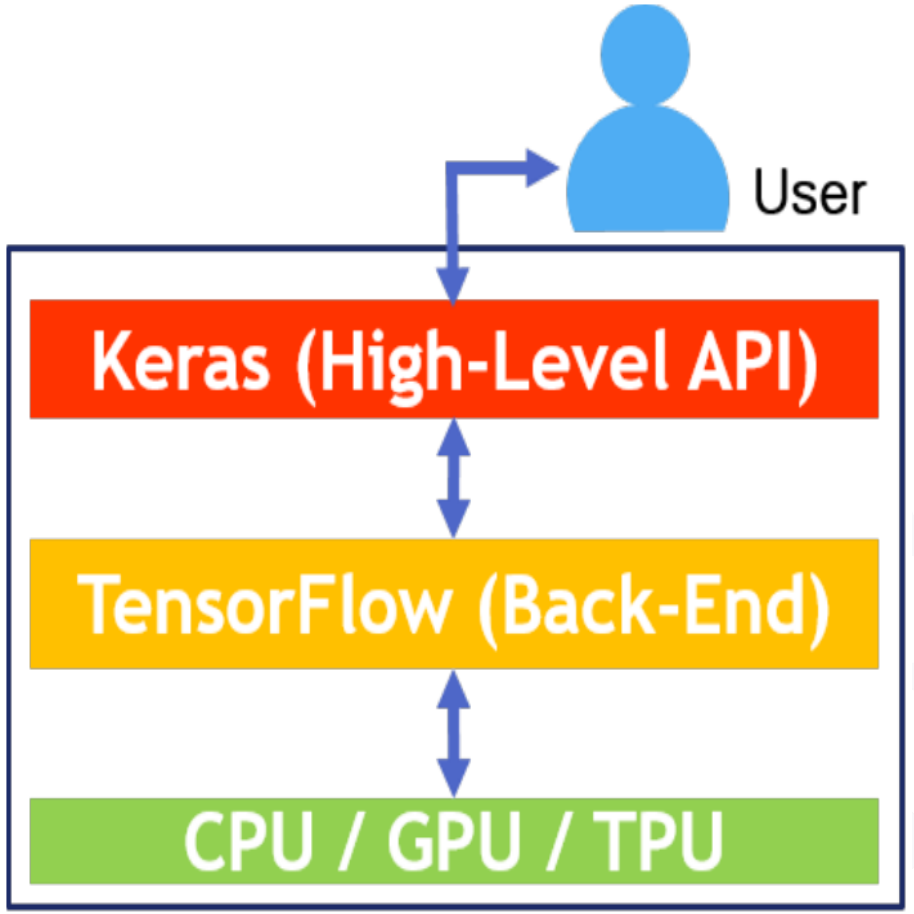
```
<class 'tensorflow.python.framework.ops.EagerTensor'>
4.0
```

텐서플로 2.x-케라스

- 케라스(Keras)
  - 파이썬으로 작성된 오픈 소스 신경망 라이브러리
  - MXNet, Deeplearning4j, 텐서플로, Microsoft Cognitive Toolkit 또는 Theano 등 다양한 플랫폼 위에서 수행할 수 있음
  - 딥 신경망과의 빠른 실험을 가능케 하도록 설계되었으며 최소한의 모듈 방식의 확장 가능성 중시
  - ONEIROS(Open-ended Neuro-Electronic Intelligent Robot Operating System) 프로젝트의 연구적 노력의 일환으로 개발됨
  - 주 개발자이자 유지보수자는 구글의 엔지니어 프랑소아 숄레(Francois Chollet)

텐서플로 2.x-케라스

- 케라스(Keras)
  - 케라스 창시자 프랑소와 숄레가 텐서플로우 2.0 개발에 직접 참여
  - 텐서플로 2.0 에서 공식적이고 유일한 High Level API (Application Programming Interface)로서 케라스가 선정됨



텐서플로 2.x-케라스

- 케라스(Keras)
  - 프랑소와 숄레는 앞으로 케라스를 직접 프로그램에서 임포트 시켜 사용하는 native Keras 보다는 텐서플로우 2.0 을 이용한, 즉 tensorflow.keras 라이브러리를 임포트 시켜서 프로그래밍 하는 것을 적극적으로 권장

텐서플로 2.x-케라스

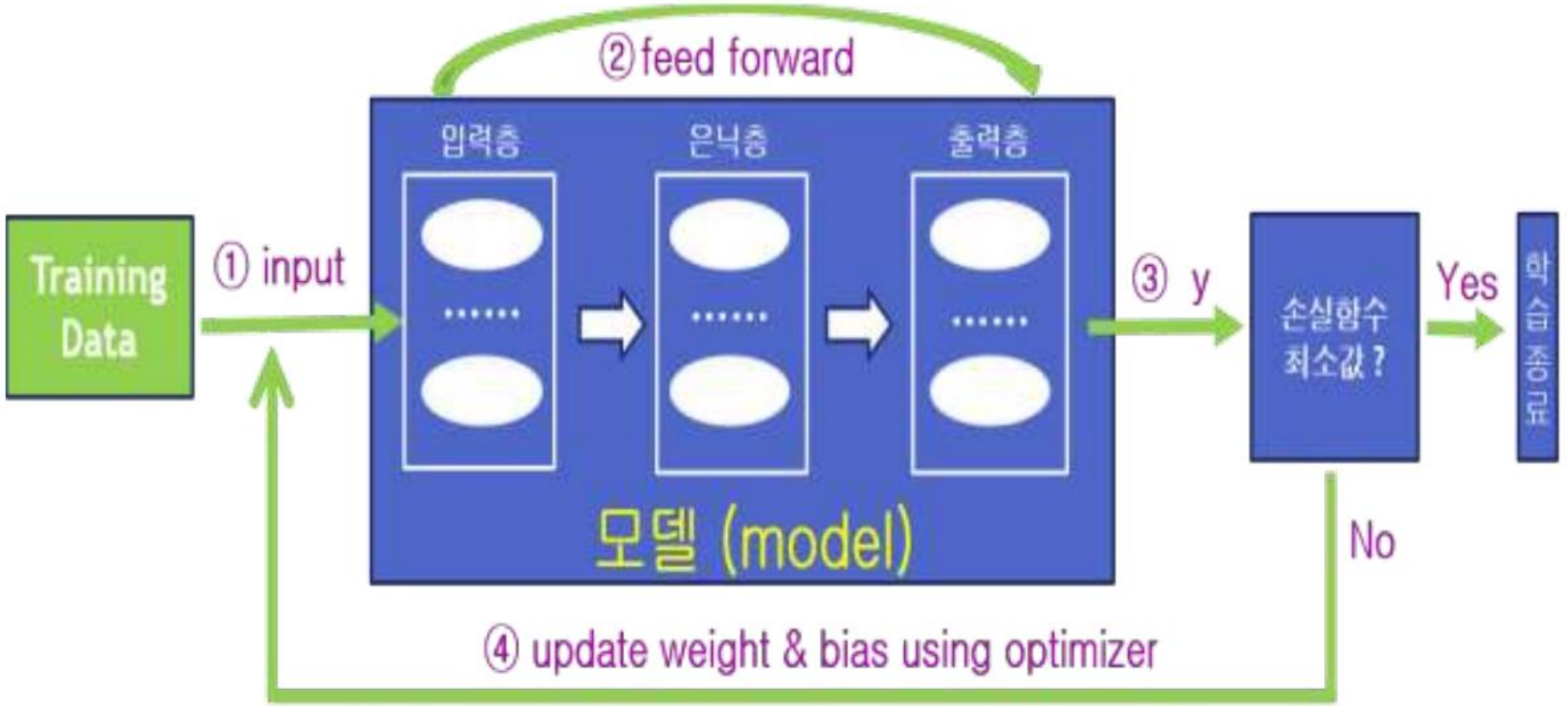
- **케라스의 두 가지 특징**
  - **사용자 친근성 (User Friendliness)**
    - 케라스의 직관적인 API 를 이용하면 일반 신경망(ANN), 합성곱 신경망(CNN) 그리고 순환 신경망(RNN) 모델 또는 이들을 조합한 다양한 딥러닝 모델을 구축하고 학습시키는데 있어 단지 몇 줄의 케라스 함수 만을 이용하여 매우 쉽게 구축 할 수 있음

텐서플로 2.x-케라스

- **케라스의 두 가지 특징**
  - **모듈성 (Modularity)**
    - 케라스에서 제공하는 다양한 모듈은 독립적으로 설정하고 연결 할 수 있음
    - 즉 입력 층 / 다수의 은닉 층 / 출력 층 으로 구성된 신경망 층(layer), 정답과 계산 값의 차이를 나타내는 손실 함수(loss function)만이 아니라, 활성화 함수와 최적화 알고리즘 그리고 정규화 기법 등은 모두가 개별적으로 독립적인 모듈이기 때문에 이러한 모듈을 서로 조합하기만 하면 새로운 딥 러닝 모델을 쉽게 만들고 학습시킬 수 있음
  - **케라스에서 가장 핵심적인 데이터 구조는 모델(model)**

텐서플로 2.x-케라스

- 케라스 기본 아키텍처인 모델 개념도



텐서플로 2.x-케라스

- 케라스의 기본 아키텍처는
  - 모델(model)을 기반으로
  - 입력 층, 다수의 은닉 층 그리고 출력 층으로 구성되어 있으며
  - 피드 포워드(feed forward)를 통해서 데이터를 전파하고
  - 오차 역전파 알고리즘을 이용하여 가중치(weight)와 바이어스(bias)를 최적의 값으로 업데이트 하는 과정

텐서플로 2.x-케라스

- 케라스의 모델은
  - 인공 신경망 자체를 나타낸다고 볼 수 있으며,
  - 모든 모델의 기본 단위는 층(layer)으로 구성됨
  - 이러한 층을 레고 블록처럼 순차적으로 쌓기만 하면
  - 일반 신경망(ANN), 합성곱 신경망(CNN) 그리고 순환 신경망(RNN) 또는 이들을 다양하게 조합한 아키텍처를 구성하는 것이 가능함

텐서플로 2.x-케라스

- 케라스의 구조와 기본 API

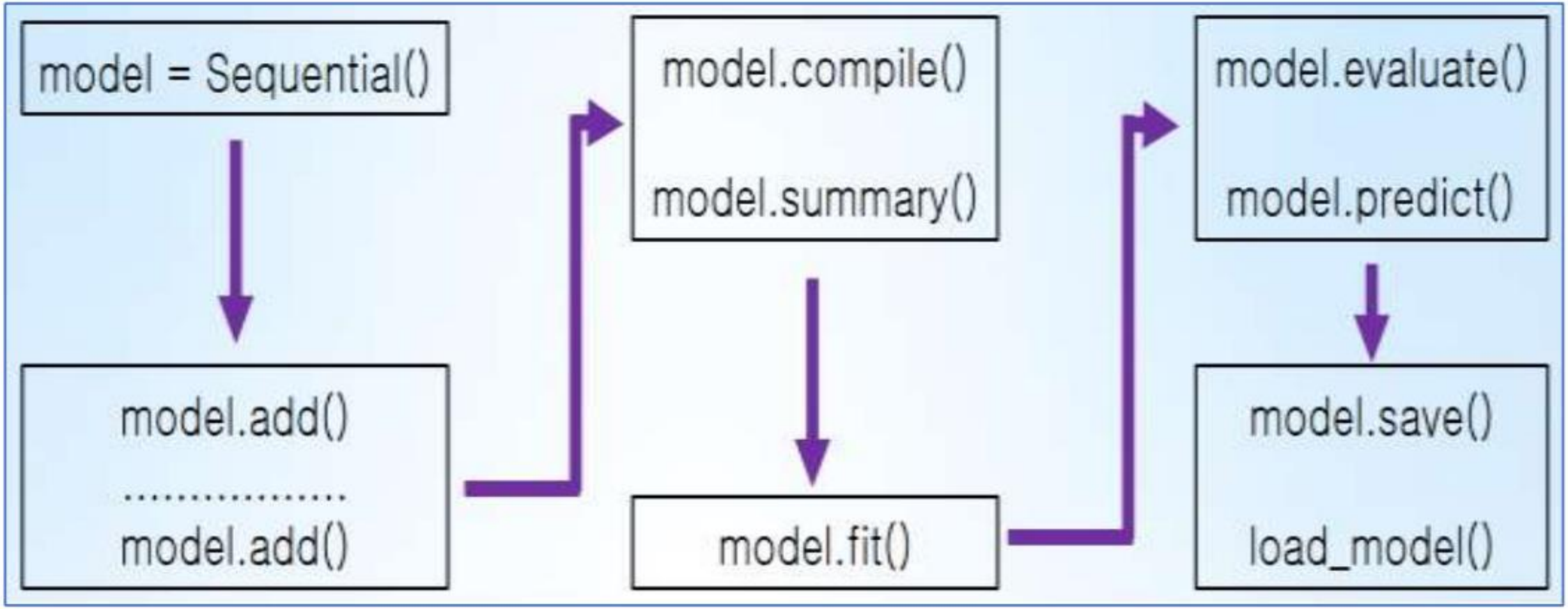
케라스 구조	기본 API
케라스 모델(model)이라는 것은 인공적인 신경망 자체를 나타내고 있음	model = Sequential() # 순차 모델 생성
모든 모델의 기본 단위는 층(layer)이며, 이러한 층을 레고 블럭처럼 순차적으로 쌓기만 하면 일반 ANN, CNN 그리고 RNN 또는 이들을 조합한 모델 구축이 가능함	model.add(...) # 층 1 추가 model.add(...) # 층 2 추가 ..... model.add(...) # 층 n 추가
모델이 구축되면 손실 함수 (loss function) 값이 최소가 될 때 까지 ①~④과정을 반복 하며 최적의 가중치 (weight)와 바이어스 (bias) 값을 찾는 학습(learning)이 진행됨	model.compile(...) # 손실함수 정의 # 학습 알고리즘 정의 model.fit(...) # 학습진행 ①~④

텐서플로 2.x-케라스

- 케라스를 이용한 개발 과정은
  - 기본적인 API 를 이용해서
  - 순차적인 모델을 만들어
  - 트레이닝 데이터를 학습하고
  - 손실 함수 값이 최소가 될 때까지
  - 가중치와 바이어스를 최적화하는 과정

텐서플로 2.x-케라스

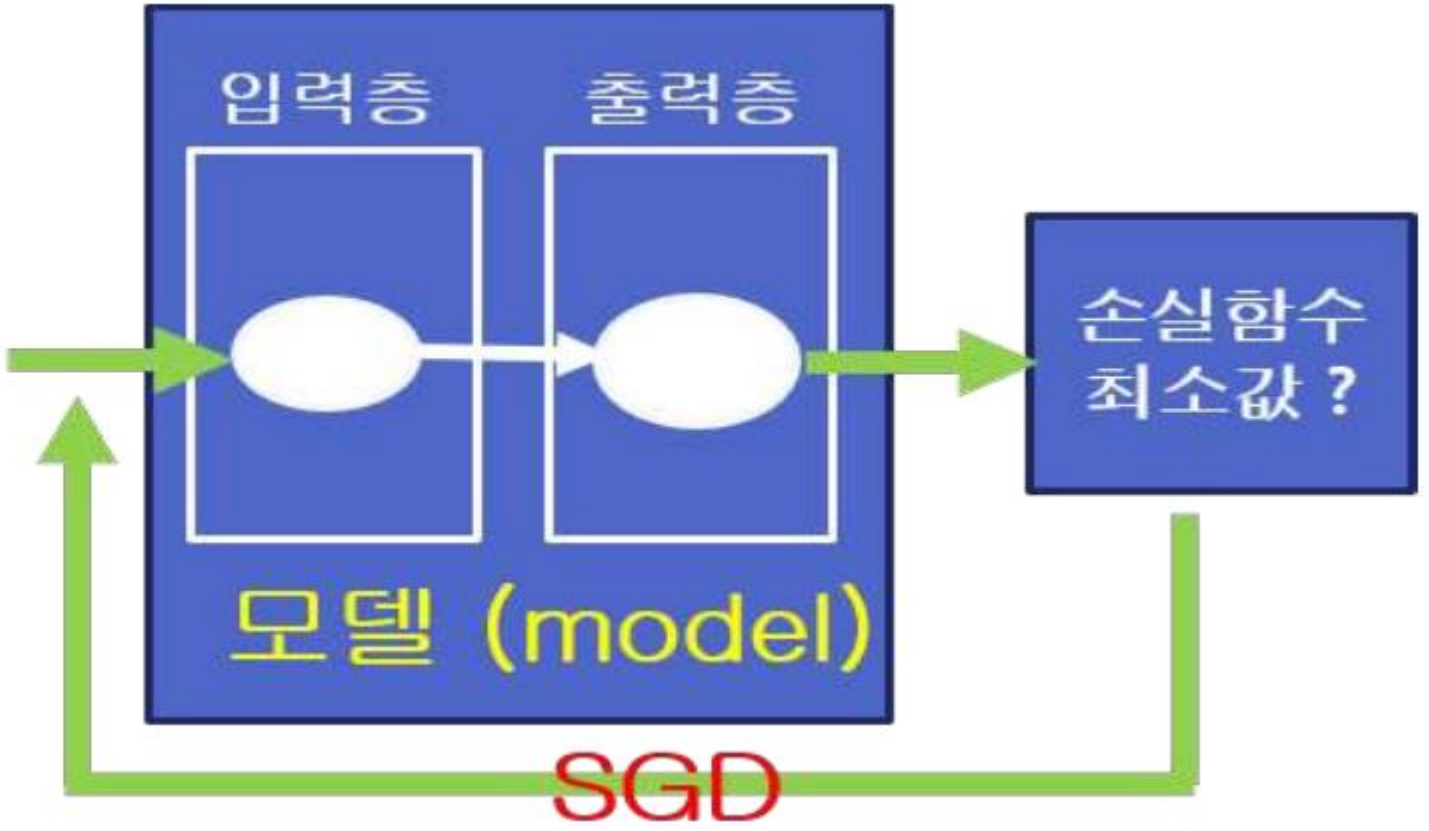
- API 관점에서 보는 케라스 개발 과정



텐서플로 2.x-케라스

- 케라스를 이용한 선형회귀 예제 (실습코드)

입력	정답
1	3
2	4
3	5
4	6
5	7
6	8



텐서플로 2.x-케라스-실습문제

- [문제 1]
  - 1) training data 입력과 정답에는 어떤 관계가 있는가?
  - 2) TensorFlow 2.x 이용하여, training data 를 학습한 후에, 다음의 test data 결과를 예측하시오
    - (5, 5, 0) , (2, 3, 1) , (-1, 0, -1) , (10, 5, 2) , (4, -1, -2)

텐서플로 2.x-케라스-실습문제

- [문제 2] Diabetes.csv 파일을 읽어 Training Data 생성 후에
  - 1) 25% 비율로 validation data 생성해서 학습함
  - 2) hist = model.fit(...) 에서 리턴 값 hist 데이터 타입을 확인 후, 각각의 값을 출력하시오
  - 3) list comprehension 을 이용하여 index_label_prediction_list 구현하시오

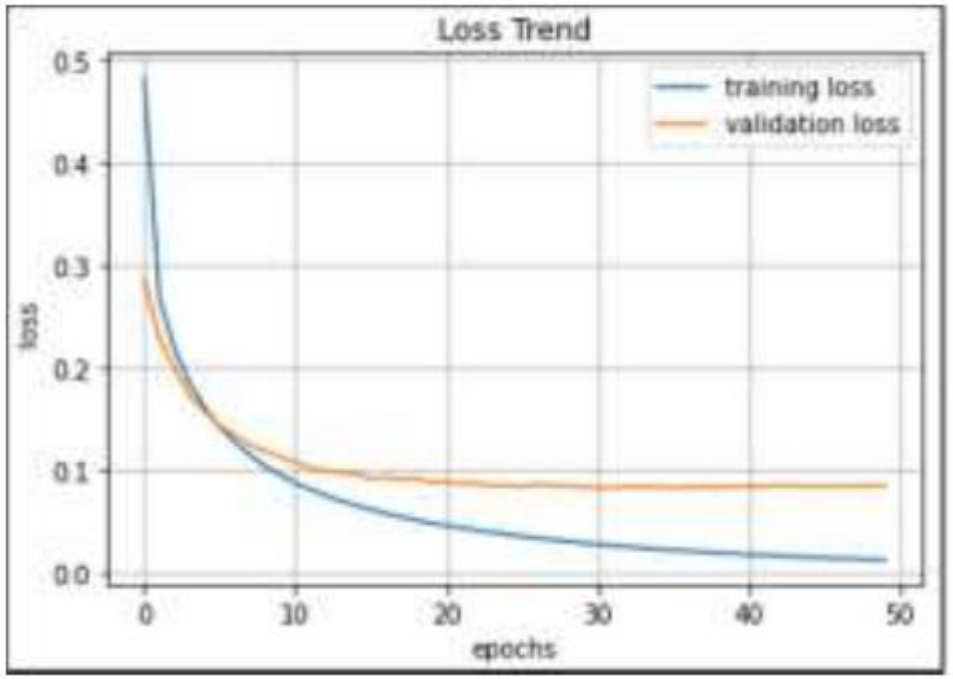
텐서플로 2.x-케라스-실습문제

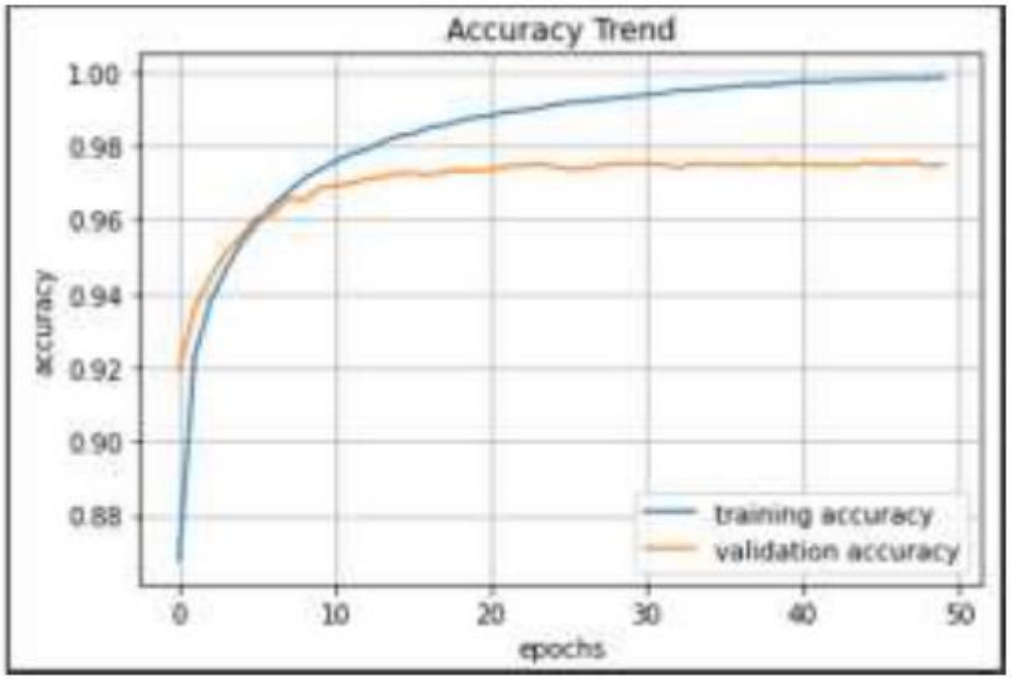
- [문제 3] 다음과 같은 조건을 만족하는 코드를 구현하시오.
  - [1] epoch 5 번마다 train data 와 validation data 에 대한 accuracy 를 출력하시오.

```
epochs =  5 , accuracy =  0.9537083506584167  , val_accuracy =  0.9555000066757202
epochs =  10 , accuracy =  0.973562479019165  , val_accuracy =  0.9685000181198112
epochs =  15 , accuracy =  0.9826666712760925  , val_accuracy =  0.9725000262260437
epochs =  20 , accuracy =  0.9877291917800903  , val_accuracy =  0.9732499718666077
epochs =  25 , accuracy =  0.991020858287813  , val_accuracy =  0.9745000004768372
epochs =  30 , accuracy =  0.9933541417121887  , val_accuracy =  0.9751666784286499
epochs =  35 , accuracy =  0.995520830154419  , val_accuracy =  0.9751666784286499
epochs =  40 , accuracy =  0.9969583153724667  , val_accuracy =  0.9749166369438171
epochs =  45 , accuracy =  0.9978749752044678  , val_accuracy =  0.9754999876022339
epochs =  50 , accuracy =  0.9984583258628845  , val_accuracy =  0.9750000238418579
```

텐서플로 2.x-케라스-실습문제

- [2] 다음과 같이 overfitting 확인할 수 있는 그래프를 출력하시오.





텐서플로 2.x-케라스-실습문제

- [문제 4] 다음과 같은 데이터가 diabetes.csv 파일로 현재 디렉토리에 저장되어 있다.
  - 학습율 0.01 을 가지는 확률적 경사하강법을 이용하여 Logistic Regression을 구현하고
  - 정확도와 오버피팅 여부를 확인하시오

텐서플로 2.x-케라스-실습문제

임신 횟수	포도당 농도	이완기 혈압	삼두근 피부두께	2시간 인슐린	체질량	당뇨 직계 가족력	나이	5년내 당뇨병 발병여부
-0.29412	0.487437	0.180328	-0.29293	0	0.00149	-0.53117	-0.03333	0
-0.88235	-0.14573	0.081967	-0.41414	0	-0.20715	-0.76687	-0.66667	1
-0.05882	0.839196	0.04918	0	0	-0.30551	-0.49274	-0.63333	0
-0.88235	-0.10553	0.081967	-0.53535	-0.77778	-0.16244	-0.924	0	1
0	0.376884	-0.34426	-0.29293	-0.60284	0.28465	0.887276	-0.6	0
.....								
-0.88235	0.899497	-0.01639	-0.53535	1	-0.10283	-0.72673	0.266667	0
-0.17647	0.005025	0	0	0	-0.10581	-0.65329	-0.63333	0
0	0.18593	0.377049	-0.05051	-0.45627	0.365127	-0.59607	-0.66667	0
-0.17647	0.075377	0.213115	0	0	-0.11774	-0.8497	-0.66667	0

759

9

- [문제 5] 다음과 같은 MNIST 데이터에 대해서 은닉 층 노드 100 개를 가지는 인공신경망 코드를 구현하고, 정확도와 오버피팅 여부를 확인하시오. (단 정답데이터는 One-Hot Encoding 되어 있음)