

```
/* elice */
```

양재 AI School 인공지능 캠프

텐서플로우 2.0 소개와 사용법



박상수 선생님

커리큘럼

1 ○ 텐서플로우 및 딥러닝 프레임워크

머신러닝과 딥러닝에 사용되는 딥러닝 프레임워크를 간단하게 살펴보고
텐서플로우의 특징에 대해 학습합니다.

2 ○ 텐서플로우 2.0 특징 및 사용법

텐서플로우 2.0 버전의 특징에 대해 살펴보고
간단한 예제를 통해 사용법을 파악합니다

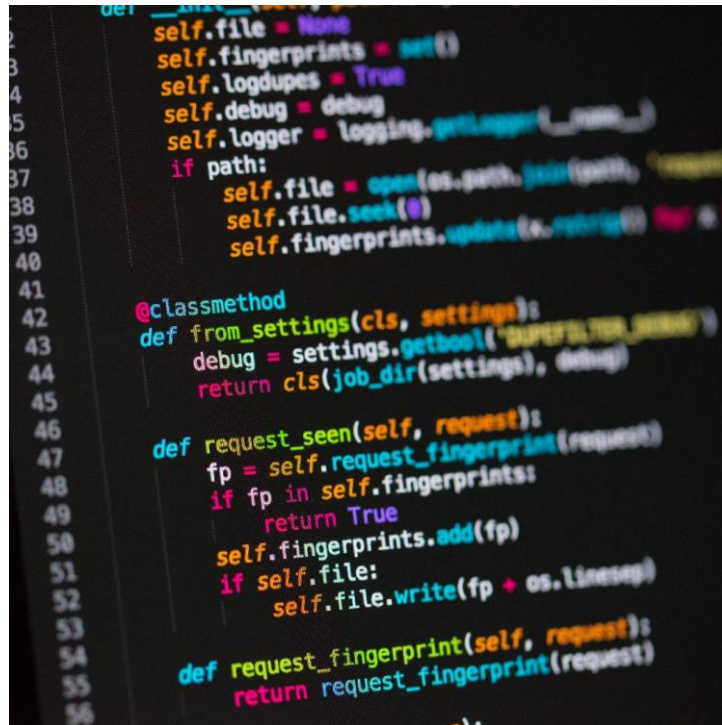
목차

1. 딥러닝 프레임워크
2. 텐서플로우의 특징
3. 텐서플로우 2.0 소개
4. 텐서플로우 2.0 사용법

딥러닝 프레임워크

러닝머신 (Running Machine)이 아니라 머신러닝

딥러닝 모델을 학습하기 위해서는



파이썬



하드웨어



C/C++

딥러닝 모델 (파이썬) +
좋은 연산 장치 (하드웨어) + 연산장치 제어 (C/C++) 등등
배울 것이 너무 많음 ? ?

배울 것은 너무 많지만



I'm deep learning framework

프레임워크 ???

frame·work

★  고교 영어1

+ 단어장 저장

미국식 [-wɜːrk]



영국식 ['freɪmwɜːk]



? 발음듣기

유익어/반의어 [명사] system, plan, order, ... [더보기](#)

출판사별 ?

옥스퍼드


동아출판

YBM

교학사

슈프림

영영사전

등급별 뜻보기 

명사 | 학습정보

✓ 예문달임

T ▾



명사

1. (건물 등의) 뼈대 [골조]

2. ~ (of/for sth) (판단·결정 등을 위한) 틀

The report provides a framework for further research.  

그 보고서는 추후 연구를 위한 틀을 마련해 주고 있다.

3. 체제, 체계

We need to establish a legal framework for the protection of the environment.  

우리는 환경 보호를 위한 법률 체계를 확립할 필요가 있다.

the basic framework of society  

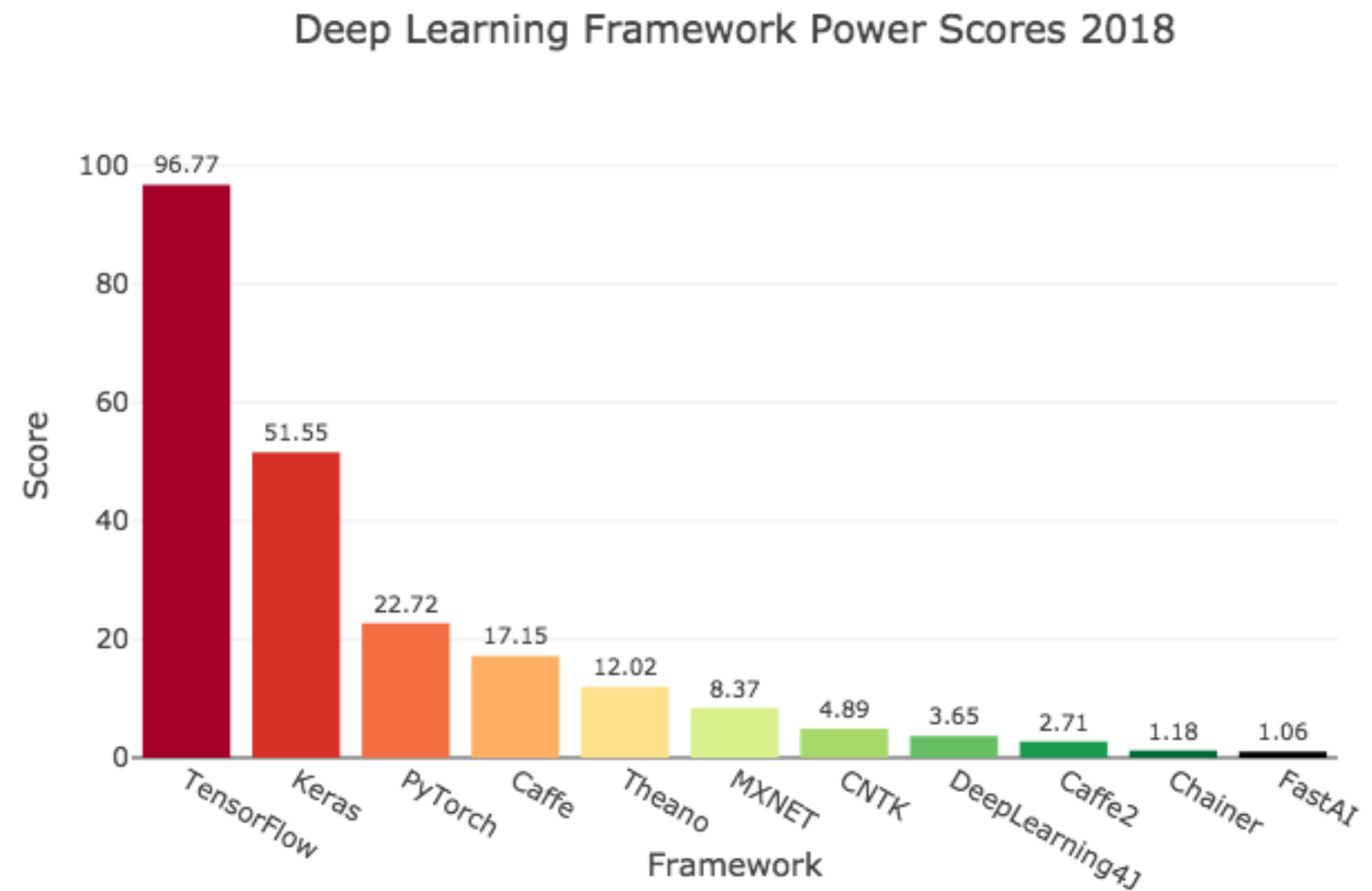
사회의 기본 체제

딥러닝 프레임워크



딥러닝 모델의 학습과 추론을 위한 프로그램
다양한 딥러닝 모델을 쉽게 구현, 사용 가능

많이 사용되는 딥러닝 프레임워크



Tensorflow (Keras + Theano) vs PyTorch (Caffe + Caffe2)

텐서플로우

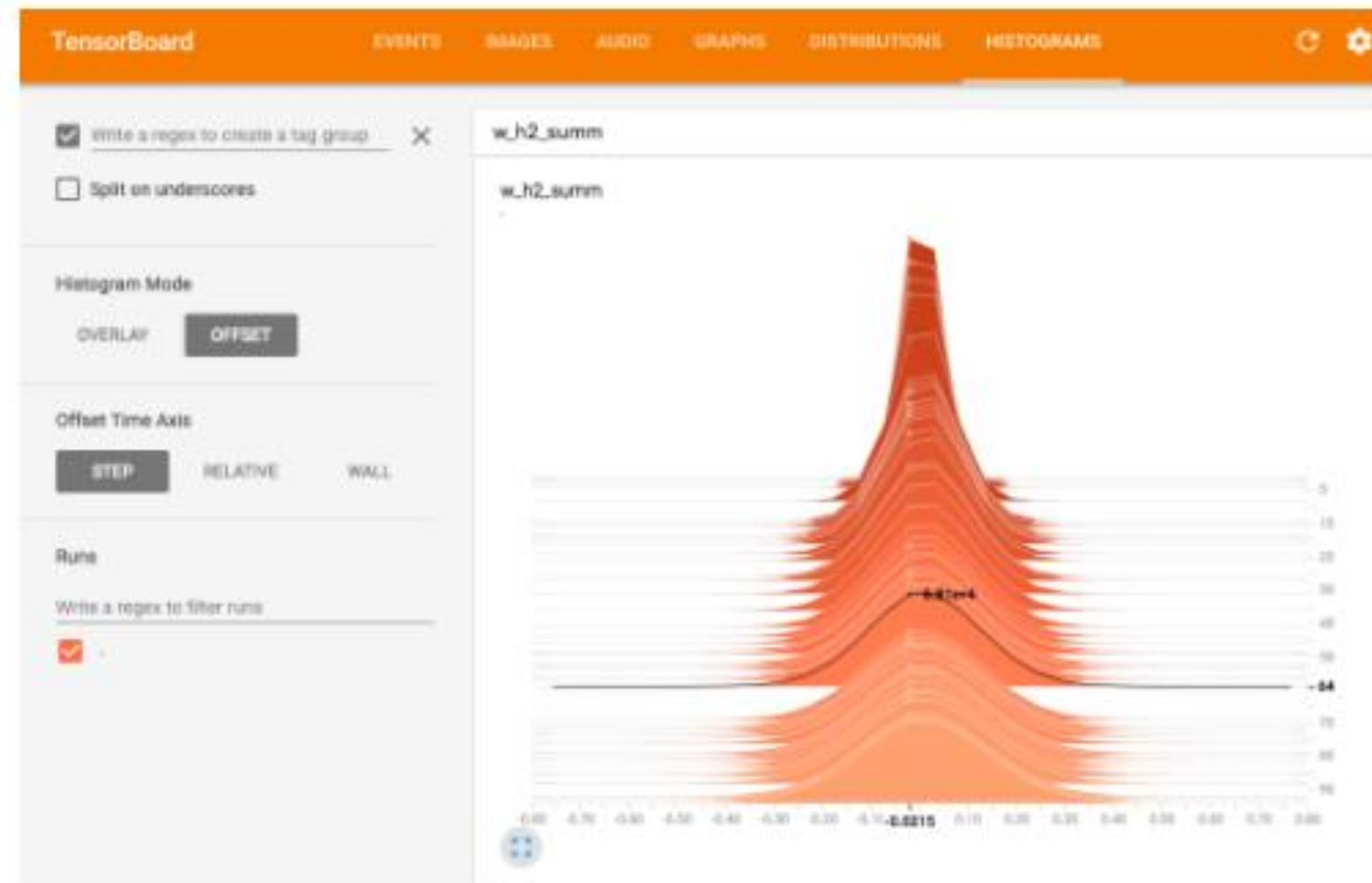
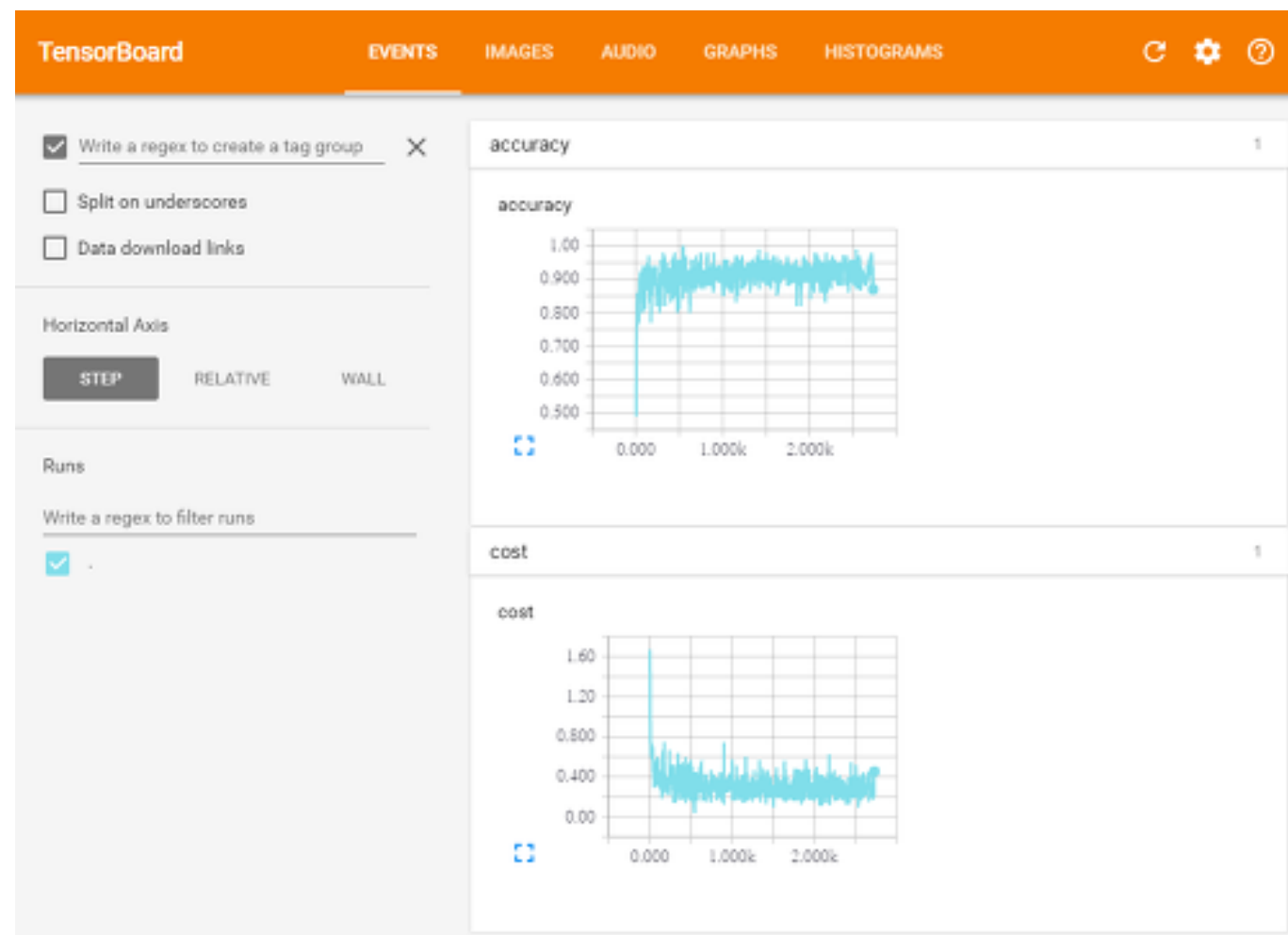
Tesor + Flow

텐서플로우 (Tensorflow)



유연하고, 효율적이며, 확장성이 있는 딥러닝 프레임워크
대형 클러스터 컴퓨터부터 스마트폰까지 다양한 디바이스에서 동작
Python 2/3, C/C++ 지원

텐서플로우의 강력한 툴: 텐서보드



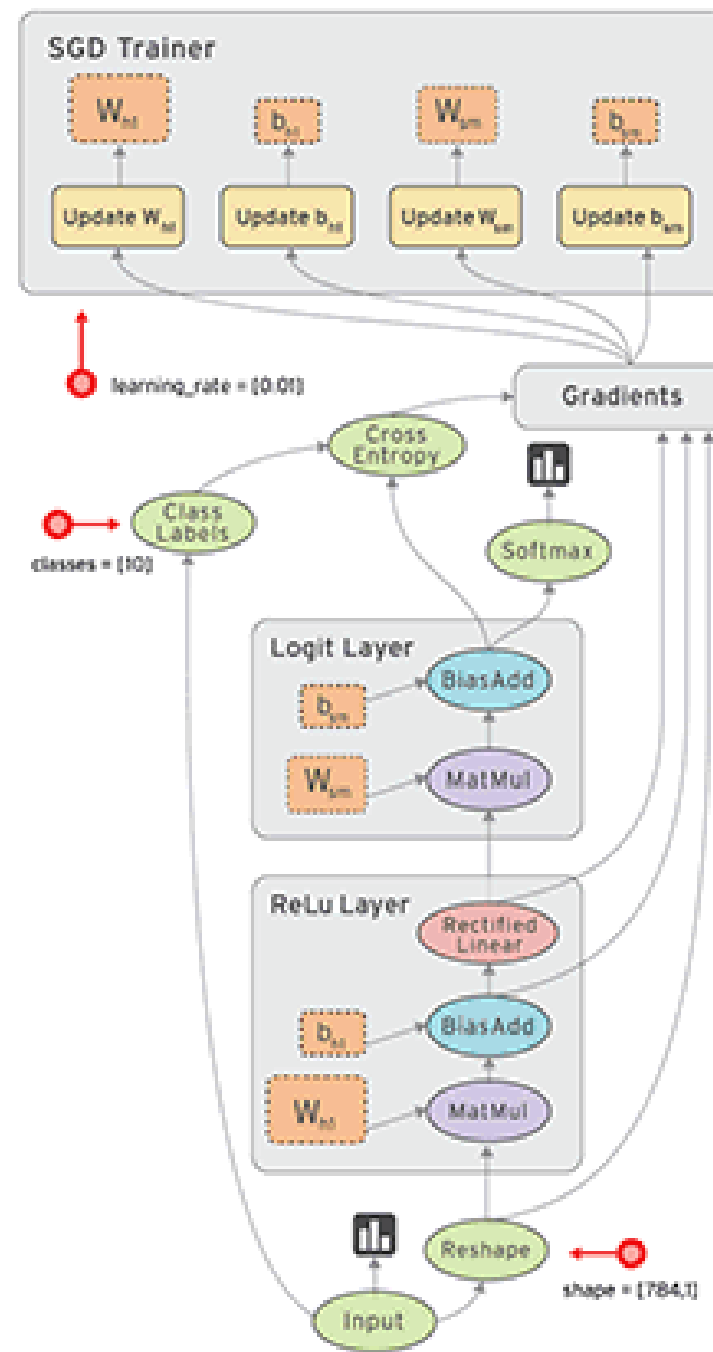
학습 과정에서 **시각적인 분석 방법**을 쉽게 사용 가능

텐서플로우 (TensorFlow)

“TensorFlow (Tensor + Flow) is an open source software library for **numerical computation** using **data flow graphs**”

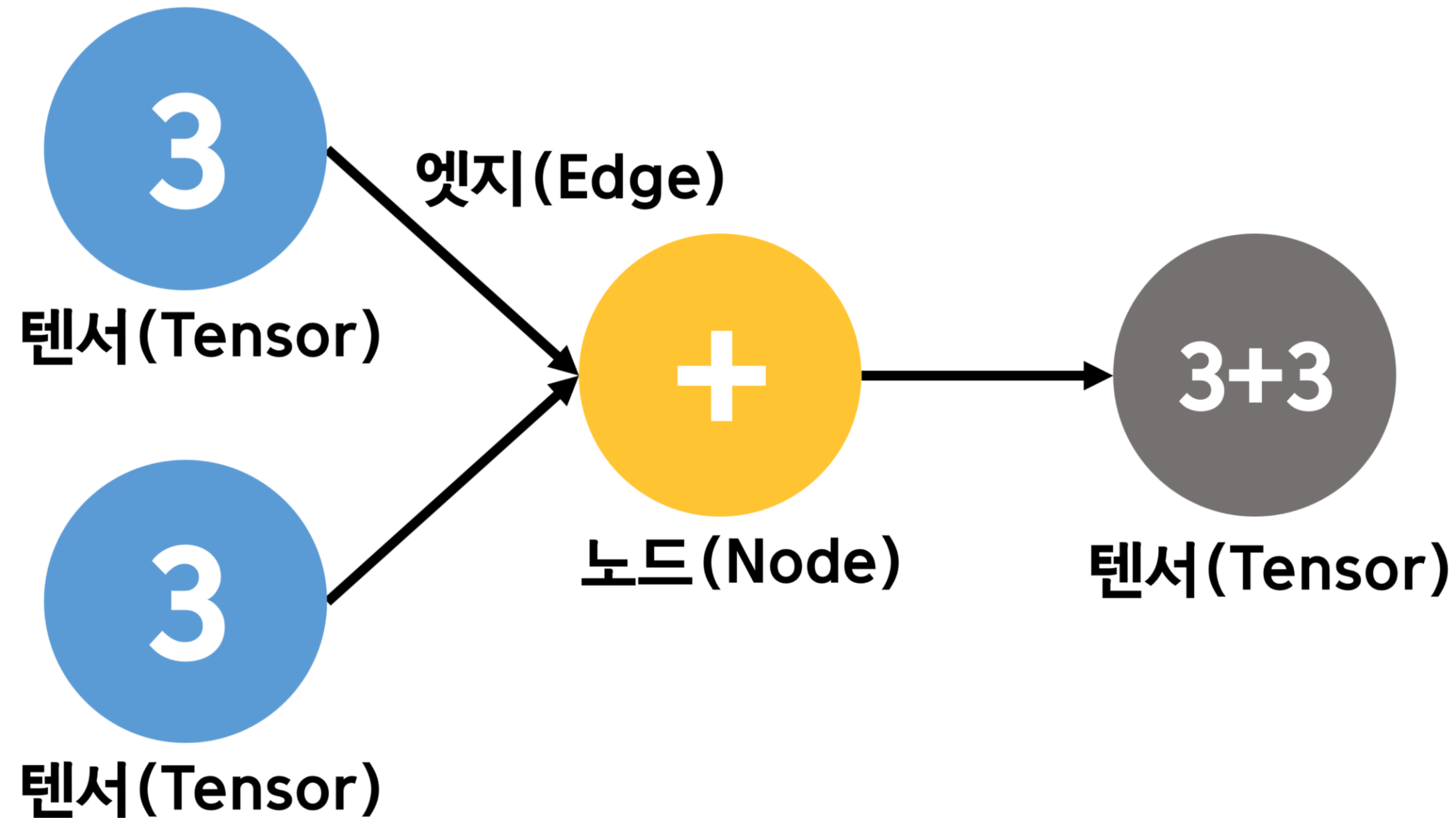
텐서: 수학적 의미로 여러 방향을 가진 벡터 (다차원 배열 데이터)
플로우: 데이터의 흐름, 즉 **텐서**라는 **데이터의 흐름**

텐서플로우의 설계방법: Data Flow



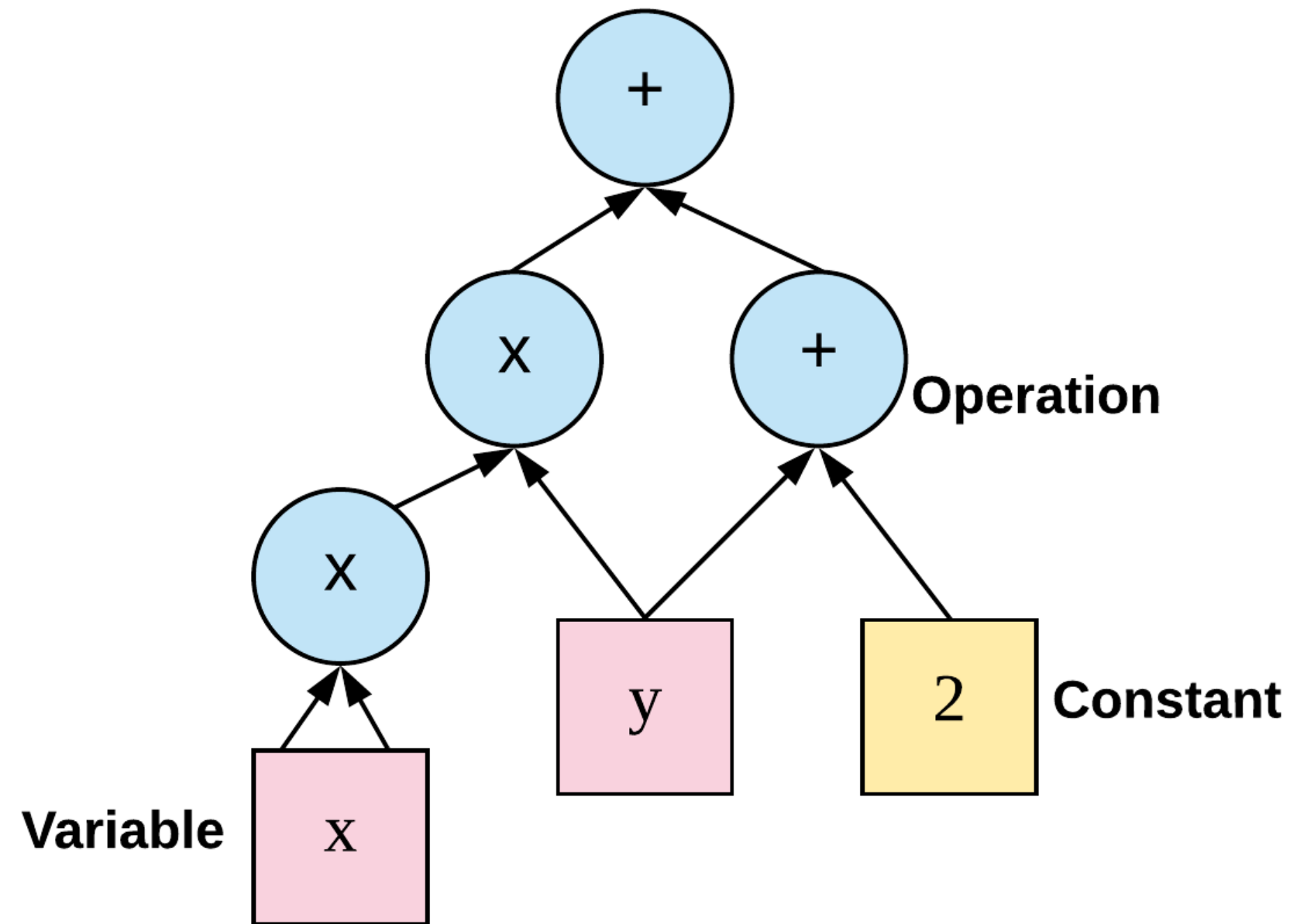
“TensorFlow (Tensor + Flow) is an open source software library for numerical computation using data flow graphs”

Graph model: Node and Edge



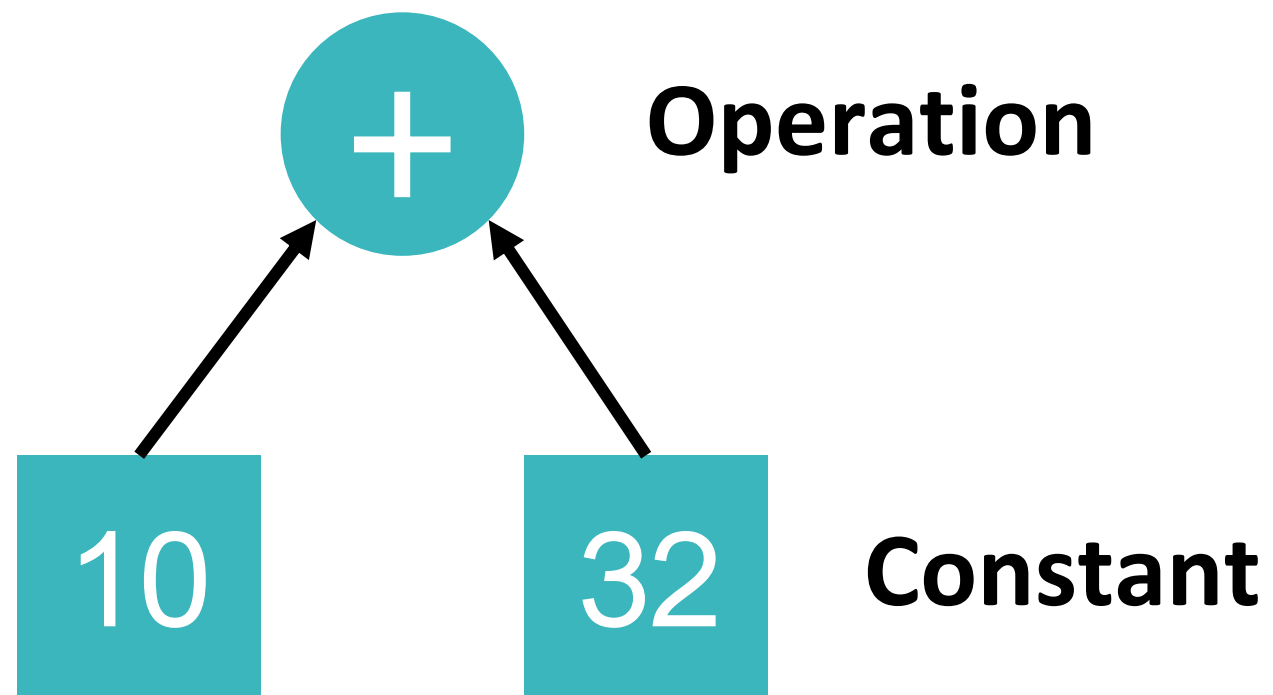
그래프는 노드 (Node)와 엣지 (Edge)로 구성
노드는 연산 (Operation), 엣지는 (데이터가 오가는 통로)

Graph model을 사용하는 이유 ?



‘y+2’로 표시해도 되는데, 그래프로 표현하는 이유는 ?
복잡한 수식을 잘게 쪼개어 단순화, 단순화한 값은 미분에 유리함

Graph model의 모델 예시



```
>>> import tensorflow as tf
>>> a = tf.constant (10)
>>> b = tf.constant (32)
>>> sess1 = tf.Session ()
>>> print (sess1.run (a+b))
```

세션 (Session): 그래프를 실행하기 위해 필요
오퍼레이션의 실행 환경을 캡슐화

텐서플로우 처리 순서

1. Tensorflow 모듈 호출
2. x라는 변수 생성, 35 값으로 초기화
3. y라는 변수 생성, 방정식 $x+5$ 정의
4. 모델 내부의 변수 초기화
5. 값을 계산하기 위해 세션 생성
6. 4에서 만든 모델 실행
7. 변수 y만 실행, 현재 값 출력

```
1 import tensorflow as tf
2 x = tf.constant(35, name='x')
3 y = tf.Variable(x+5,name='y')
4 model = tf.global_variables_initializer()
5 with tf.Session() as session:
6     session.run(model)
7     print(session.run(y))
```

XOR 문제 예시

```
import tensorflow as tf

T = 1. F = 0.

train_in = [ [T, T], [T, F], [F, T], [F, F], ]
train_out = [ [F], [T], [T], [F], ]

// w1, w2는 정규분포를 따르는 값을 저장 (텐서 크기 [2,2] , [2,1])
// b1, b2는 0인 값으로 모두 저장 (b1은 2개, b2는 1개)
w1 = tf.Variable(tf.random_normal([2, 2]))
b1 = tf.Variable(tf.zeros([2]))

w2 = tf.Variable(tf.random_normal([2, 1]))
b2 = tf.Variable(tf.zeros([1]))
```

XOR 문제 예시

```
out1 = tf.nn.relu(tf.matmul(train_in, w1) + b1)
```

```
out2 = tf.nn.relu(tf.matmul(out1, w2) + b2)
```

```
// Loss function
```

```
error = tf.subtract(train_out, out2)
```

```
mse = tf.reduce_mean(tf.square(error))
```

```
// Hyperparameter
```

```
err = 1.0
```

```
target = 0.01
```

```
epoch = 0
```

```
max_epochs = 1000 // 학습 데이터 전체를 몇 번 학습에 사용했는지 의미함
```

XOR 문제 예시

```
// Optimization 설정
train = tf.train.GradientDescentOptimizer(0.01).minimize(mse)

sess = tf.Session()           // Session(): 그래프 실행
sess.run(tf.global_variables_initializer()) // 그래프 초기화

while err > target and epoch < max_epochs: // 학습을 언제까지 진행하는 것인지 정의
    epoch += 1
    err, _ = sess.run([mse, train])

print("epoch:", epoch, "mse:", err)
print("result: ", out1)
```

텐서플로우 2.0

1.x 버전과 2.0의 차이점은 ?

텐서플로우 2.0



TensorFlow
2.0

베타 버전 배포 중 (19.06 ~)

편리성이 향상된 텐서플로우



Keras와 즉시 실행 (Eager execution)을 이용한 쉬운 모델 작성
어떤 플랫폼에서든 튼튼한 (Robust) 모델 배포
Deprecated된 API를 정리하고 중복을 줄여서 **API 단순화**

즉시 실행 (Eager execution)

예를 들면, 텐서플로우에서 간단한 계산을 수행한다고 생각해봅시다:

```
x = tf.placeholder(tf.float32, shape=[1, 1])
m = tf.matmul(x, x)

with tf.Session() as sess:
    print(sess.run(m, feed_dict={x: [[2.]]}))

# Will print [[4.]]
```

즉시 실행 (Eager execution)은 이 작업을 단순하게 할 수 있습니다:

```
x = [[2.]]
m = tf.matmul(x, x)

print(m)
```

**Eager execution은 그래프 생성 없이 연산을 즉시 실행
실행할 계산 그래프 없이, 실제 값을 얻는 것 가능
(직관적인 인터페이스, 쉬운 디버깅)**

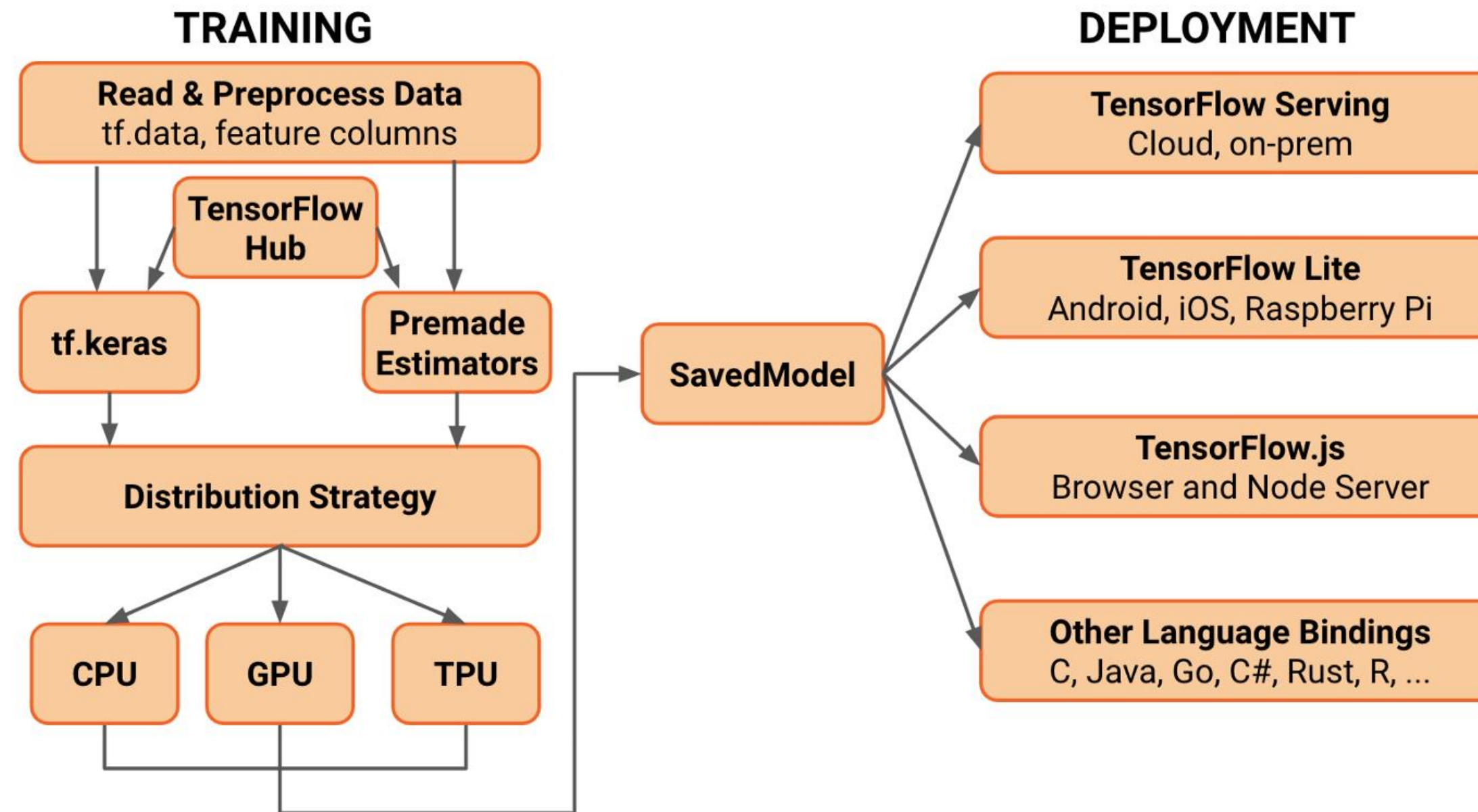
High level APIs (Keras API)

```
1 import numpy as np
2 from keras.models import Sequential
3 from keras.layers.core import Activation, Dense
4
5 training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
6 target_data = np.array([[0],[1],[1],[0]], "float32")
7
8 model = Sequential()
9 model.add(Dense(32, input_dim=2, activation='relu'))
10 model.add(Dense(1, activation='sigmoid'))
11
12 model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])
13
14 model.fit(training_data, target_data, nb_epoch=1000, verbose=2)
15
16 print model.predict(training_data)
```

```
1 import tensorflow as tf
2
3 input_data = [[0., 0.], [0., 1.], [1., 0.], [1., 1.]] # XOR input
4 output_data = [[0.], [1.], [1.], [0.]] # XOR output
5
6 n_input = tf.placeholder(tf.float32, shape=[None, 2], name="n_input")
7 n_output = tf.placeholder(tf.float32, shape=[None, 1], name="n_output")
8
9 hidden_nodes = 5
10
11 b_hidden = tf.Variable(tf.random_normal([hidden_nodes]), name="hidden_bias")
12 W_hidden = tf.Variable(tf.random_normal([2, hidden_nodes]), name="hidden_weights")
13 hidden = tf.sigmoid(tf.matmul(n_input, W_hidden) + b_hidden)
14
15 W_output = tf.Variable(tf.random_normal([hidden_nodes, 1]), name="output_weights") # output layer's weight matrix
16 output = tf.sigmoid(tf.matmul(hidden, W_output)) # calc output layer's activation
17
18 cross_entropy = tf.square(n_output - output) # simpler, but also works
19
20 loss = tf.reduce_mean(cross_entropy) # mean the cross_entropy
21 optimizer = tf.train.AdamOptimizer(0.01) # take a gradient descent for optimizing with a "stepsize" of 0.1
22 train = optimizer.minimize(loss) # let the optimizer train
23
24 init = tf.initialize_all_variables()
25
26 sess = tf.Session() # create the session and therefore the graph
27 sess.run(init) # initialize all variables
28
29 for epoch in xrange(0, 2001):
30     # run the training operation
31     cvalues = sess.run([train, loss, W_hidden, b_hidden, W_output],
32                        feed_dict={n_input: input_data, n_output: output_data})
33
34     if epoch % 200 == 0:
35         print("")
36         print("step: {:>3}".format(epoch))
37         print("loss: {}".format(cvalues[1]))
38
39 print("")
40 print("input: {} | output: {}".format(input_data[0], sess.run(output, feed_dict={n_input: [input_data[0]]})))
41 print("input: {} | output: {}".format(input_data[1], sess.run(output, feed_dict={n_input: [input_data[1]]})))
42 print("input: {} | output: {}".format(input_data[2], sess.run(output, feed_dict={n_input: [input_data[2]]})))
43 print("input: {} | output: {}".format(input_data[3], sess.run(output, feed_dict={n_input: [input_data[3]]})))
```

텐서플로우에 비해 **고차원 (High-Level) API**
고차원 API의 사용을 통해 사용자가 쉽게 개발 가능

튼튼한 (Robust)한 모델 배포



학습된 모델을 다양한 Deployment에서 동작 가능

텐서플로우 2.0 사용법

이론수업에서 맛보는 예제

텐서플로우 2.0 개발환경 특징



파이썬 3.0 버전 (파이썬 2.x 버전은 2020 이후부터는 지원 x)
정식 버전은 올해 8월 발표 예정

Eager execution: 기본

```
from __future__ import absolute_import, division, print_function, unicode_literals  
import tensorflow as tf  
tf.enable_eager_execution() // eager execution 활성화  
print(tf.executing_eagerly()) // eager execution 가능한지 확인
```

Eager execution을 사용하기 위해서는 지원 여부 파악 필요 !

Eager execution: 기본

```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf

tf.enable_eager_execution() // eager execution 활성화

x = tf.matmul([[1,2],[3,4]], [[4,5],[6,7]])
y = tf.add(x, 1)
z = tf.random_uniform([5, 3])

print(x)
print(y)
print(z)
```

Eager execution: NumPy

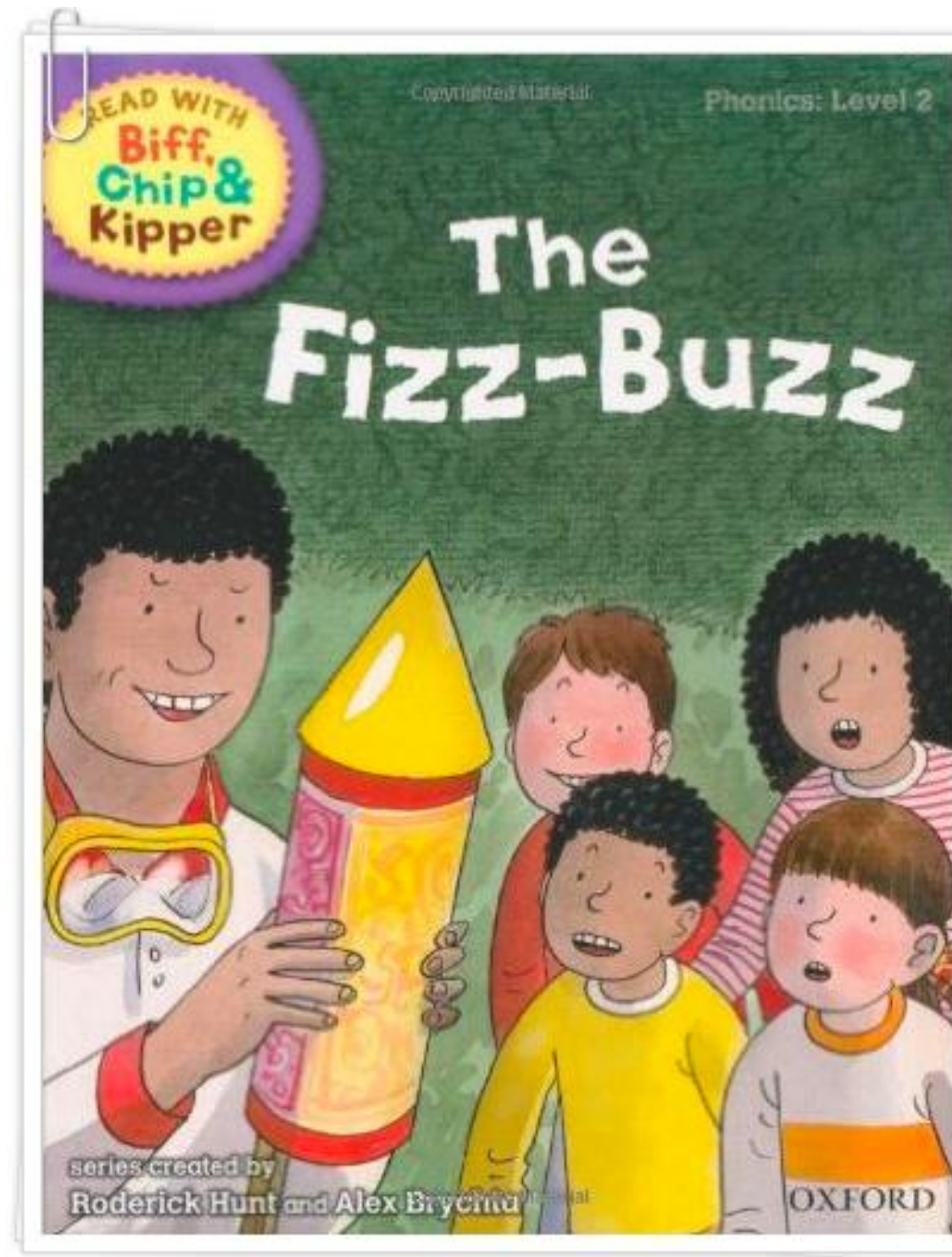
```
import numpy as np
import tensorflow as tf
tf.enable_eager_execution()

x = tf.add(1, 1)          # tf.Tensor with a value of 2
y = tf.add(np.array(1), np.array(1)) # tf.Tensor with a value of 2
z = np.multiply(x, y)      # numpy.int64 with a value of 4

print(y)
print(y.numpy)
```

NumPy 배열과 Tensor는 자동으로 호환 가능 (1.x에서는 불가능)
NumPy operation에 의해 Tensor가 들어와도 NumPy 배열로 변경

Eager execution: 동적 흐름 제어



동적 흐름 제어는 Fizzbuzz와 같은 조건이 있는 문제 해결에 적합
Fizzbuzz ???

Fizzbuzz problem ?

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

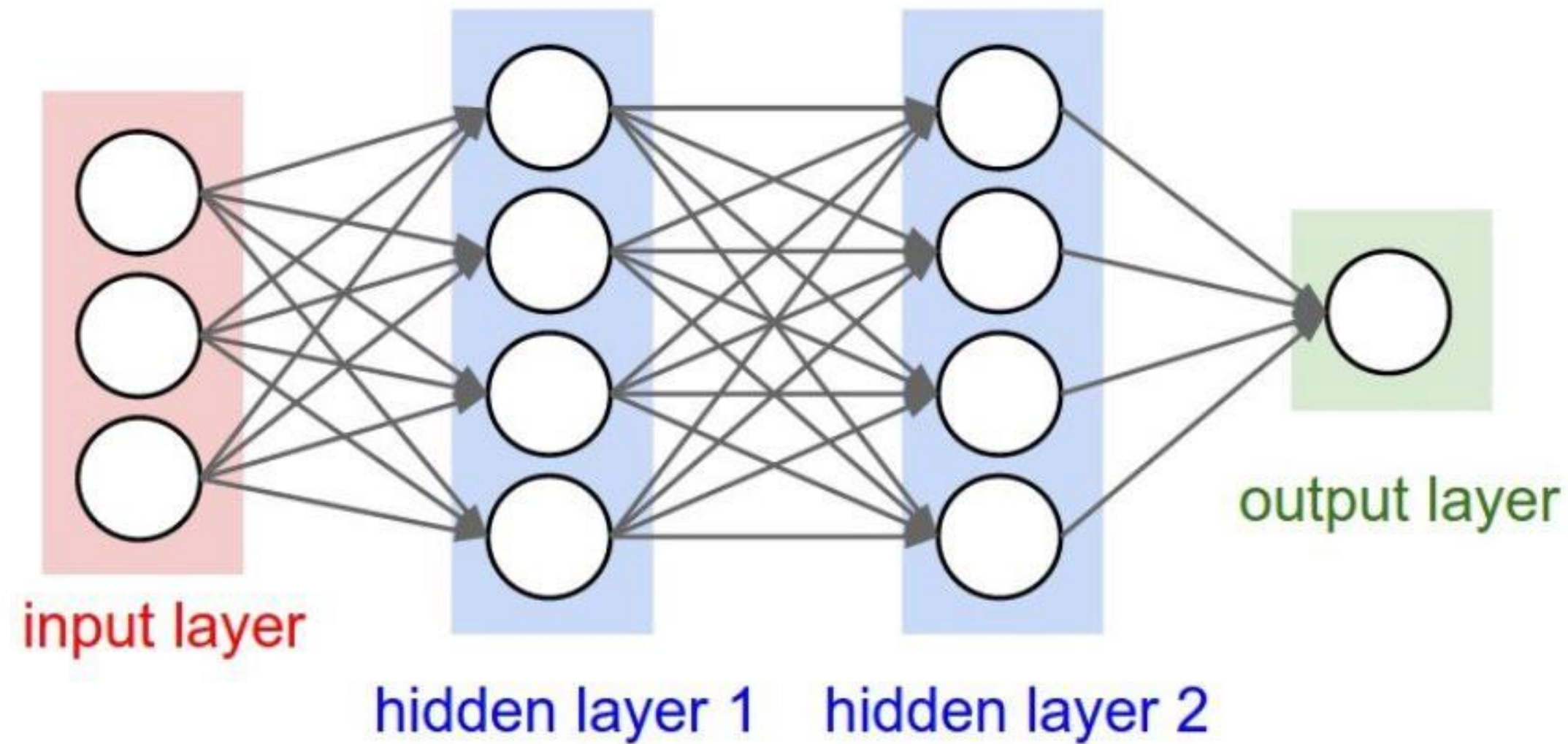
1~100 사이의 숫자를 프린트 하는 프로그램
3의 배수이면 'Fizz', 5의 배수이면 'Buzz', 둘다 이면 'FizzBuzz'

Eager execution: Fizzbuzz

```
def fizzbuzz(max_num):  
    counter = tf.constant(0)  
    max_num = tf.convert_to_tensor(max_num)  
    for num in range(1, max_num.numpy()+1):  
        num = tf.constant(num)  
        if int(num % 3) == 0 and int(num % 5) == 0:    print('FizzBuzz')  
        elif int(num % 3) == 0:                      print('Fizz')  
        elif int(num % 5) == 0:                      print('Buzz')  
        else:                                         print(num.numpy())  
        counter += 1
```

Session()에서는 동적 제어 불가능
Eager execution에서는 가능

케라스 API: Sequential 모델



텐서플로우는 그래프 기반으로 모델을 설계
케라스는 레이어들을 선형으로 쌓는 Sequential 모델

Sequential 모델: 레이어 쌓기

```
from tensorflow.keras import layers

model = tf.keras.Sequential()           // Sequential 모델
model.add(layers.Dense(16, input_dim=2, activation='relu ' )) // 레이어 추가
model.summary()
```

케라스에서는 레이어를 생성하는데 **Sequential.add()** 사용
입출력을 모두 연결해주는 Dense 레이어
Dense(출력 뉴런 갯수, 입력 뉴런 갯수, 활성화 함수)

Sequential 모델: 학습 방법

```
from tensorflow.keras import layers

model = tf.keras.Sequential()                // Sequential 모델
model.add(layers.Dense(16, input_dim=2, activation='relu ' )) // 레이어 추가
model.compile(loss='mse ' ,
               optimizer='adam ' ,
               metrics=['binary_accuracy'])
```

Sequential.compile()을 사용해 학습 방법 설정
(손실 함수 종류, Optimizer, 정확도 평가 방법)

Sequential 모델: 훈련 시작

```
from tensorflow.keras import layers

model = tf.keras.Sequential()                // Sequential 모델
model.add(layers.Dense(16, input_dim=2, activation='relu ' )) // 레이어 추가
model.compile(loss='mse ' ,
               optimizer='adam ' ,
               metrics=['binary_accuracy'])
model.fit(training_data, target_data, nb_epoch=20)
```

Sequential.fit()을 사용해 학습 시작
(학습 데이터, 성능 평가 데이터, epoch)

케라스 API: XOR 문제

```
import tensorflow as tf
import numpy as np
from tensorflow.keras import layers

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

model = tf.keras.Sequential()
model.add(layers.Dense(16, input_dim=2, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='mse ', optimizer='adam ', metrics=['binary_accuracy'])
model.fit(training_data, target_data, nb_epoch=20)
```

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

medium.com/elice