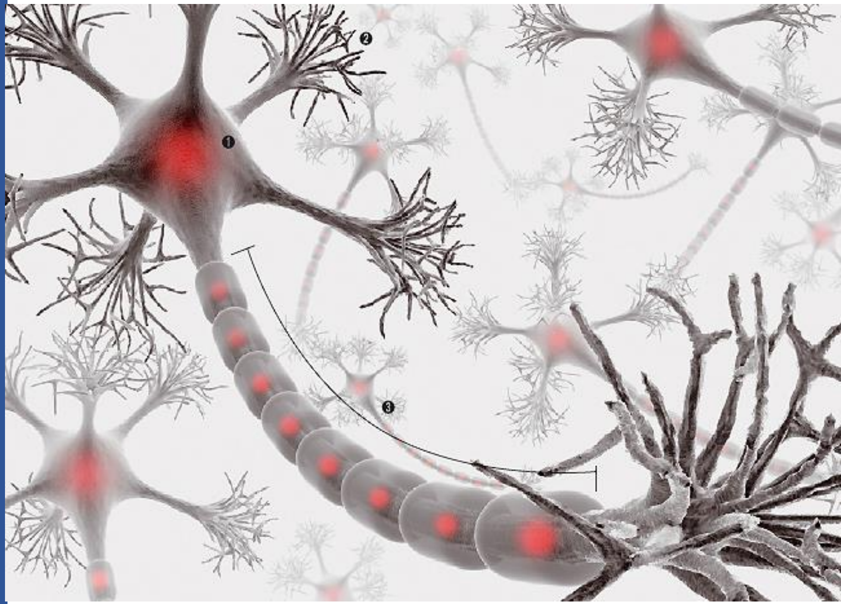
Tensorflow 2.0

학습 목표

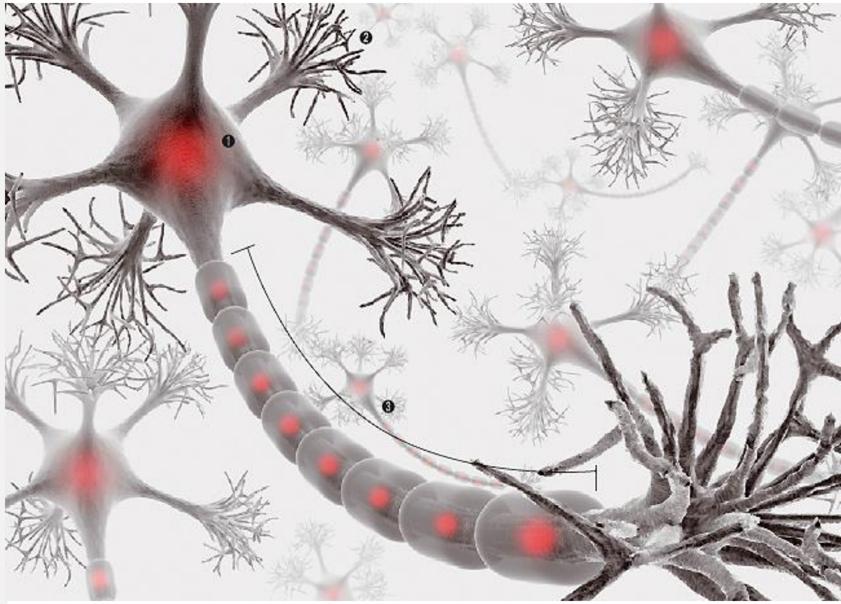
- 딥러닝 프레임워크의 역할과 Tensorflow의 사용법을 이해한다.
- .

주요 내용

- 1. 하드웨어
- 2. 소프트웨어
- 3. Tensorflow
- 4. Keras

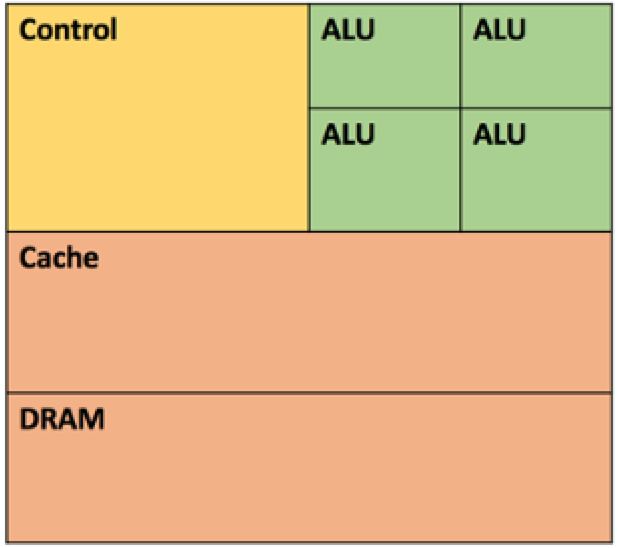


1 하드웨어

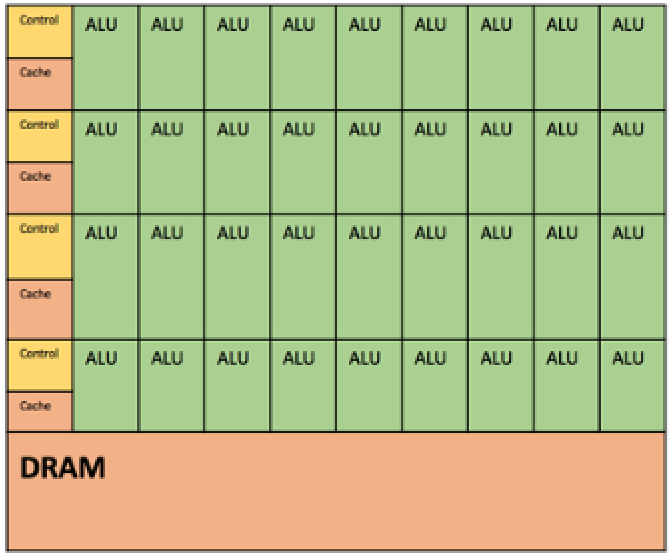


CPU vs GPU

CPU



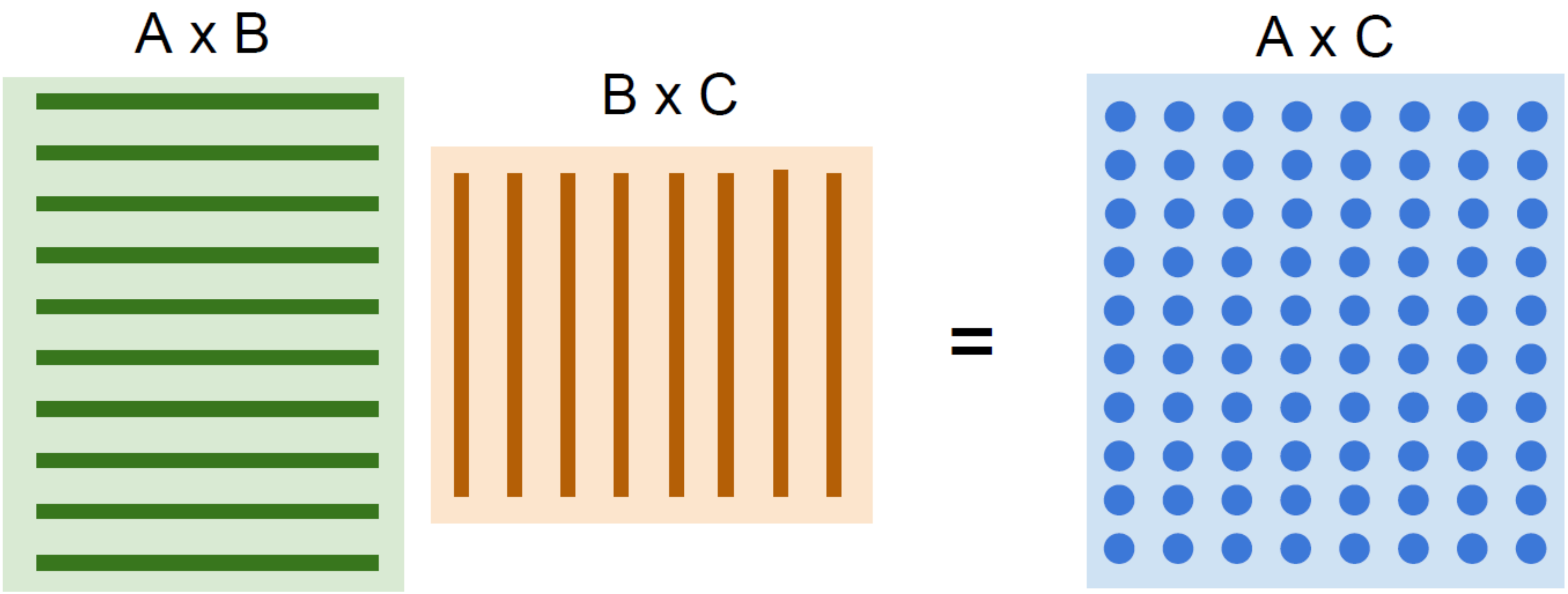
GPU



- Core 개수가 적음
- 각 Core는 매우 빠르고 범용적
- **Sequential task에 적합**

- Core 개수가 매우 많음
- 각 Core는 느리고 제한적
- **Parallel task에 적합**

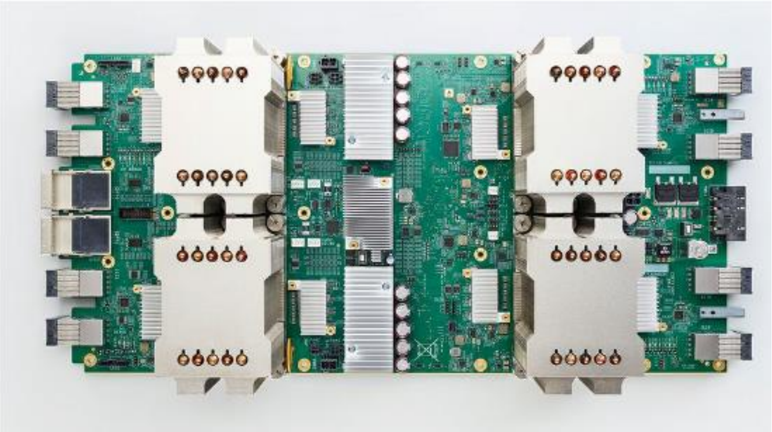
Matrix Multiplication



Stanford CS231n: Convolutional Neural Networks for Visual Recognition

TPU (Tensor Processing Unit)

Deep Learning 전용 Processor



Google Cloud TPU 2.0 = 180 TFLOP!
Google Cloud TPU 3.0 = 2.0보다 8배 빨라짐

NVIDIA TITAN V =
14 TFLOP (FP32),
112 TFLOP (FP16)

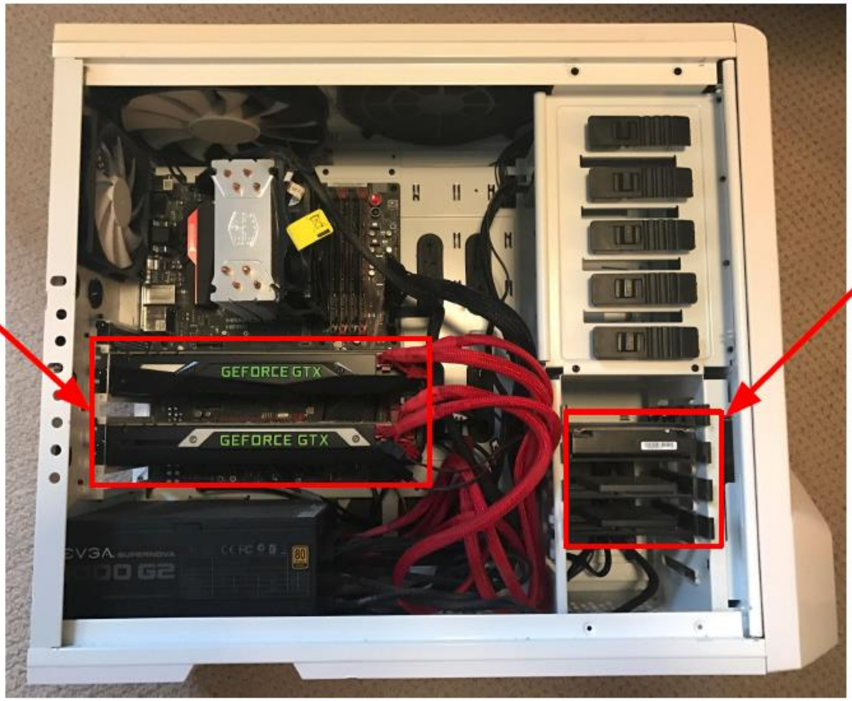
Stanford CS231n: Convolutional Neural Networks for Visual Recognition

CPU vs GPU vs TPU

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$385	~540 GFLOPs FP32
GPU (NVIDIA RTX 2080 Ti)	3584	1.6 GHz	11 GB GDDR6	$1199	~13.4 TFLOPs FP32
TPU NVIDIA TITAN V	5120 CUDA, 640 Tensor	1.5 GHz	12GB HBM2	$2999	~14 TFLOPs FP32 ~112 TFLOP FP16
TPU Google Cloud TPU	?	?	64 GB HBM	$4.50 per hour	~180 TFLOP

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

CPU / GPU Communication



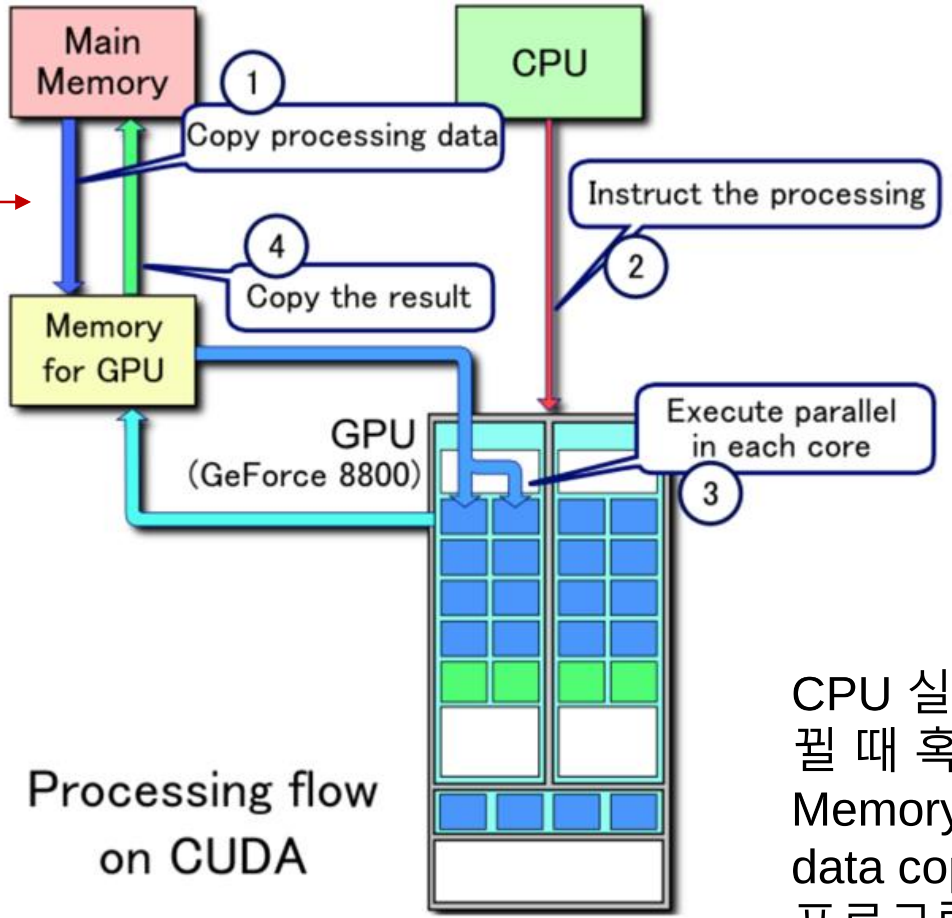
훈련 시 데이터를 읽어서 GPU로 보내는 것이 bottleneck이 될 수 있음.

해결책:
- 전체 데이터를 RAM으로 읽기
- HDD 대신 SSD 사용하기
- 데이터를 읽을 때 여러 CPU thread 사용

Stanford CS231n: Convolutional Neural Networks for Visual Recognition

CPU / GPU Communication

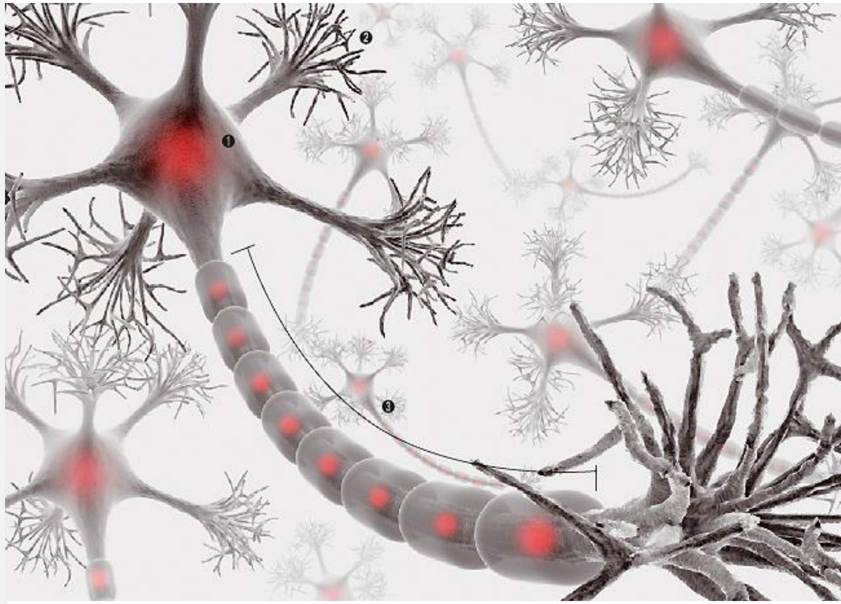
병목지점 (Bottleneck)



CPU 실행에서 GPU 실행으로 바뀔 때 혹은 그 반대의 경우 Main Memory와 GPU Memory 사이의 data copy가 bottleneck이 되므로 프로그램 설계를 잘해야 함

https://www.datascience.com/blog/cpu-gpu-machine-learning

2 소프트웨어



딥러닝 프레임워크

Caffe
(UC Berkeley)

Caffe2
(Facebook)

Paddle
(Baidu)

Chainer

Torch
(NYU / Facebook)

PyTorch
(Facebook)

개발하기 편한 프레임워크

MXNet
(Amazon)
Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

CNTK
(Microsoft)

Theano
(U Montreal)

TensorFlow
(Google)

가장 보편적인 프레임워크

Deeplearning4j

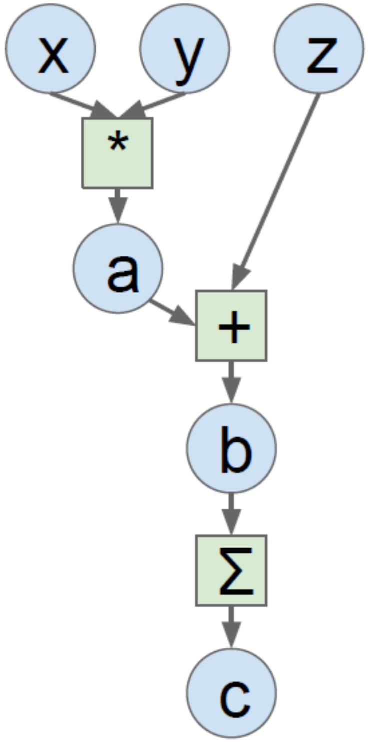
Stanford CS231n: Convolutional Neural Networks for Visual Recognition

딥러닝 프레임워크를 왜 사용해야 하는가?

Quick	새로운 아이디어를 신속하게 개발하고 테스트 할 수 있다
Automatic	복잡한 Gradient 계산을 자동으로 해준다.
Efficient	GPU를 효율적 활용할 수 있다. (cuDNN, cuBLAS, etc Wrapper)

딥러닝 구현 시 NumPy의 한계

```
import numpy as np
np.random.seed(0)

N, D = 3, 4
x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

# Forward Pass
a = x * y
b = a + z
c = np.sum(b)

# Gradient 계산
grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



장점:
- Clean API
- 수치를 다루는 코드를 쉽게 작성할 수 있음

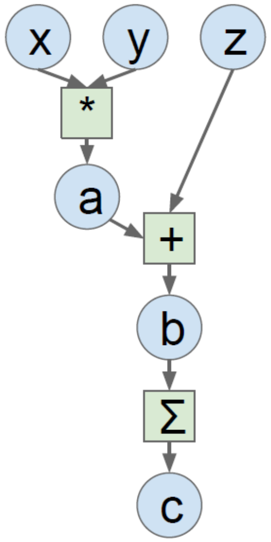
단점:
- Gradient를 직접 계산해야 함
- GPU에서 실행할 수 없음

참고 Gradient 계산

$$c = \sum_i \sum_j b_{i,j}$$

$$b = a + z$$

$$a = x \, * y$$



Local Gradient

$$\frac{\partial c}{\partial b_{i,j}} = 1$$

$$\frac{\partial c}{\partial b} = \begin{bmatrix} \frac{\partial c}{\partial b_{1,1}} & \frac{\partial c}{\partial b_{1,2}} & \cdots & \frac{\partial c}{\partial b_{1,D}} \\ \frac{\partial c}{\partial b_{2,1}} & \frac{\partial c}{\partial b_{2,2}} & \cdots & \frac{\partial c}{\partial b_{2,D}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial c}{\partial b_{N,1}} & \frac{\partial c}{\partial b_{N,2}} & \cdots & \frac{\partial c}{\partial b_{N,D}} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$\frac{\partial b}{\partial a} = 1_{[N,D]} \qquad \frac{\partial b}{\partial z} = 1_{[N,D]}$$

$$\frac{\partial a}{\partial x} = y \qquad \frac{\partial a}{\partial y} = x$$

Global Gradient

$$\frac{\partial c}{\partial b} = 1_{[N,D]}$$

$$\frac{\partial c}{\partial a} = \frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial a} = 1_{[N,D]}$$

$$\frac{\partial c}{\partial z} = \frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial z} = 1_{[N,D]}$$

$$\frac{\partial c}{\partial x} = \frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial a} \cdot \frac{\partial a}{\partial x} = y$$

$$\frac{\partial c}{\partial y} = \frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial a} \cdot \frac{\partial a}{\partial y} = x$$

딥러닝 프레임워크 Gradient 자동 계산

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4
x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

# Forward Pass
a = x * y
b = a + z
c = np.sum(b)

# Gradient 계산
grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

TensorFlow

```
import tensorflow as tf

N, D = 3, 4
x = tf.Variable(tf.random.normal((N, D)))
y = tf.Variable(tf.random.normal((N, D)))
z = tf.Variable(tf.random.normal((N, D)))

# Forward Pass
with tf.GradientTape() as tape:
    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

# Gradient 계산
grad_x, grad_y, grad_z = tape.gradient(c, [x, y, z])
```

딥러닝 프레임워크 GPU 실행

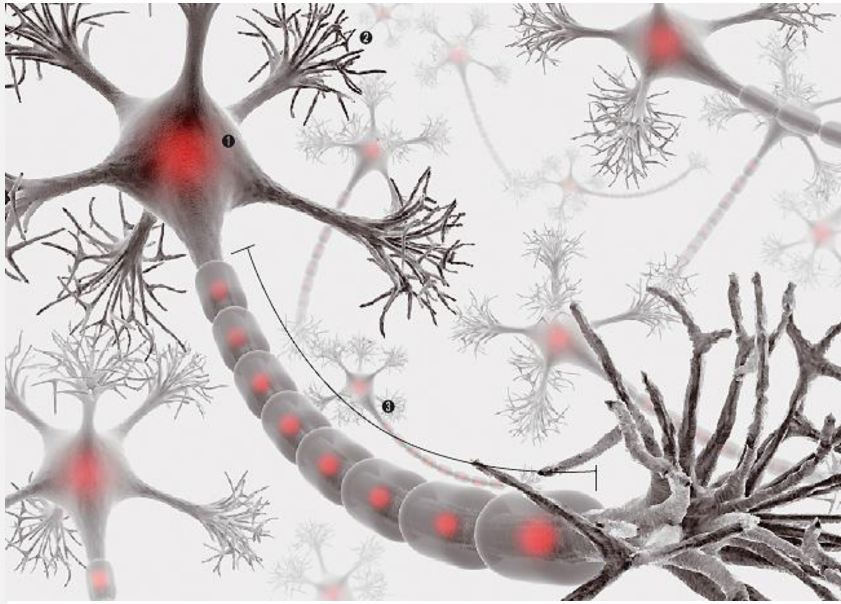
CPU 사용

```
import tensorflow as tf

N, D = 3, 4
with tf.device("CPU:0"):
    x = tf.Variable(tf.random.normal((N, D)))
    y = tf.Variable(tf.random.normal((N, D)))
    z = tf.Variable(tf.random.normal((N, D)))

    # Forward Pass
    with tf.GradientTape() as tape:
        a = x * y
        b = a + z
        c = tf.reduce_sum(b)

    # Gradient 계산
    grad_x, grad_y, grad_z = tape.gradient(c, [x, y, z])
```

GPU 사용

```
import tensorflow as tf

N, D = 3, 4
with tf.device("GPU:0"):
    x = tf.Variable(tf.random.normal((N, D)))
    y = tf.Variable(tf.random.normal((N, D)))
    z = tf.Variable(tf.random.normal((N, D)))

    # Forward Pass
    with tf.GradientTape() as tape:
        a = x * y
        b = a + z
        c = tf.reduce_sum(b)

    # Gradient 계산
    grad_x, grad_y, grad_z = tape.gradient(c, [x, y, z])
```

3 TensorFlow



What is TensorFlow

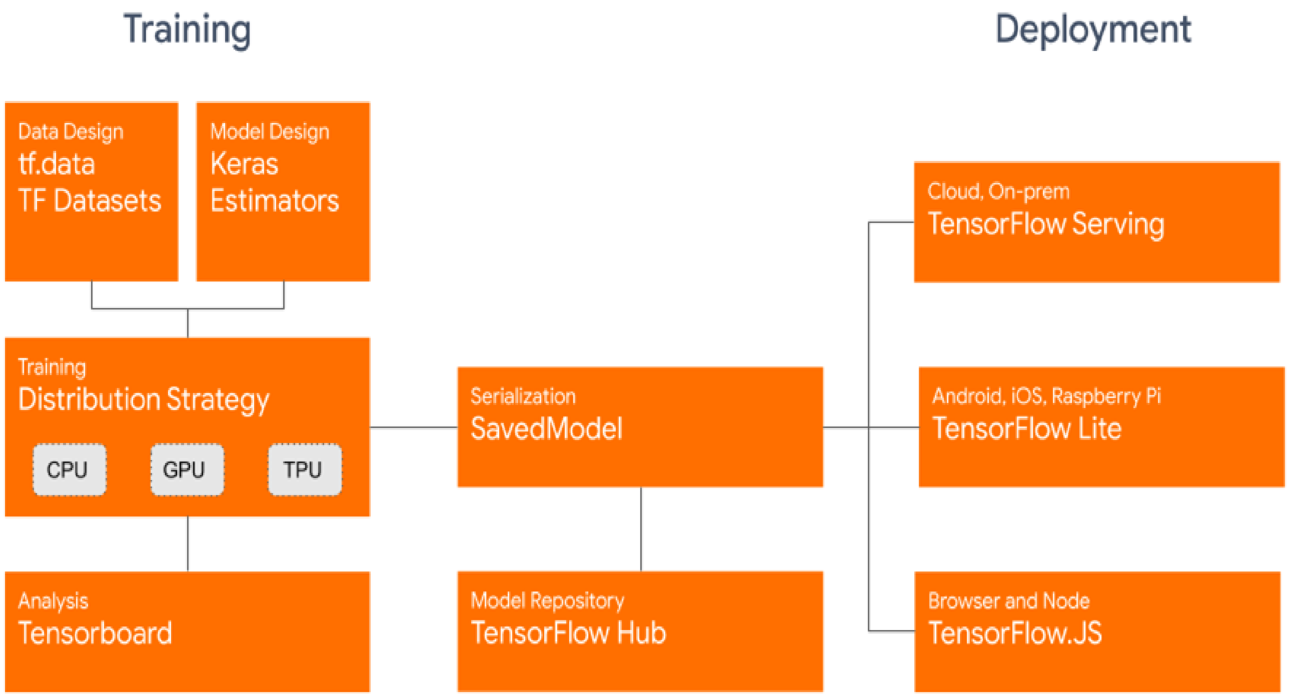
- TensorFlow 0.5 Release (2015. 11)
- **TensorFlow 2.0** Release (2019. 10)
- C++ core, Python API
- High-level API : **Keras**
- Community
  - 117,000+ GitHub stars
  - TensorFlow.org : Blogs, Documentation, DevSummit, YouTube talks

TensorFlow

http://tensorflow.org

TensorFlow 2.0

- **Ecosystem**
  - Keras : high-level API
  - TensorFlow.js : in the browser
  - TensorFlow Lite : on the phone
  - Colaboratory : in the cloud
  - TPU : optimized hardware
  - TensorBoard : visualization
  - TensorFlow Hub : graph modules
- **Extras**
  - Swift for TensorFlow
  - TensorFlow Serving
  - TensorFlow Extended (TFX)
  - TensorFlow Probability
  - Tensor2Tensor



https://medium.com/tensorflow/tensorflow-2-0-is-now-available-57d706c2a9ab

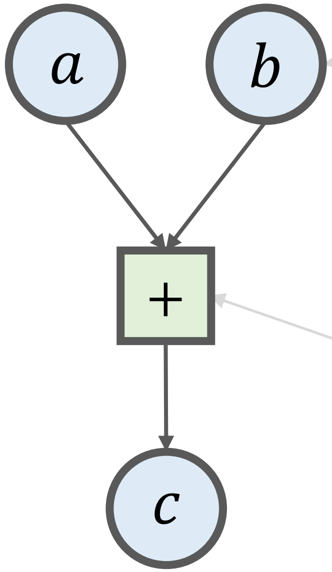
TensorFlow 2.0

- **Eager execution** (Define by Run)
- Functions, not session
- AutoGraph
- API Cleanup (no more globals)

Computational Graph

신경망에 필요한 계산 과정을 계산 그래프(Computational Graph)로 정의

<u>**계산 그래프**</u>



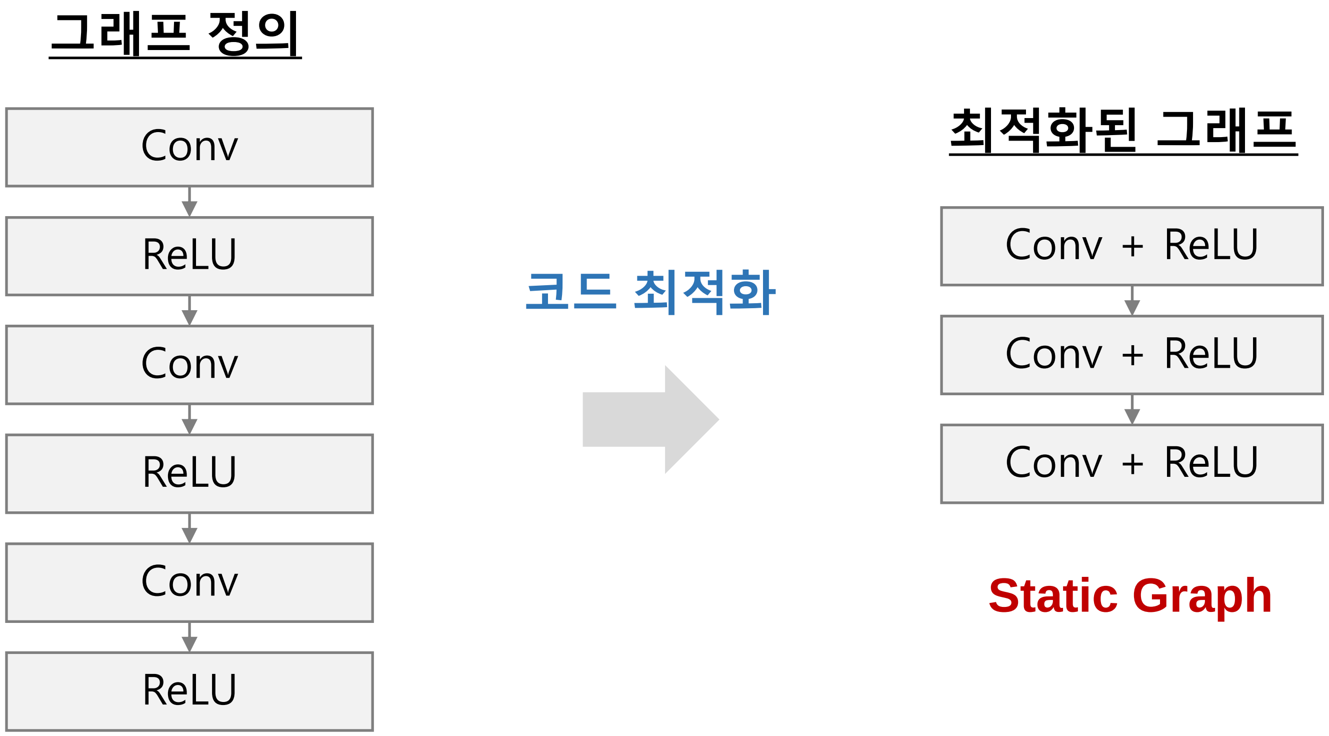
예제 : c = a + b 계산

텐서 (Tensor) :
- 데이터가 저장된 다차원 배열

오퍼레이션 (Operator) :
- 다차원 배열에 대한 연산을 수행
- 산술연산, 논리연산, 행렬연산 등

Define and Run

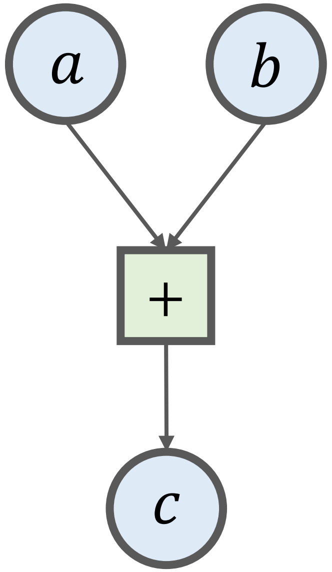
실행에 최적화된 Static Graph 생성



Define and Run

세션을 통해 계산 그래프를 실행

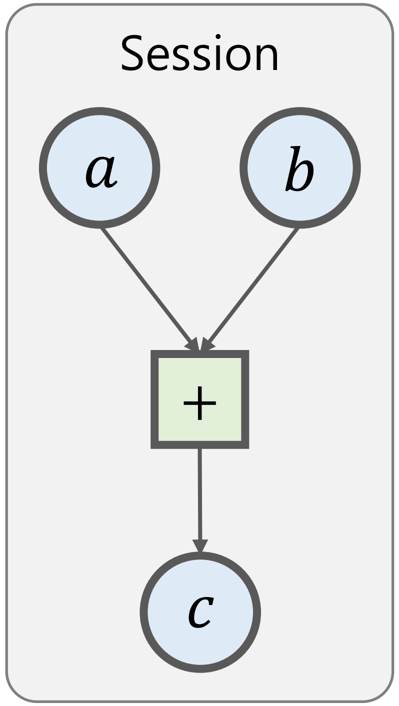
<u>계산 그래프 정의</u>



<u>세션 생성</u>

세션 (Session) :
- 계산 그래프를 실행하는 단위
- 실행 환경을 추상화 한 개념

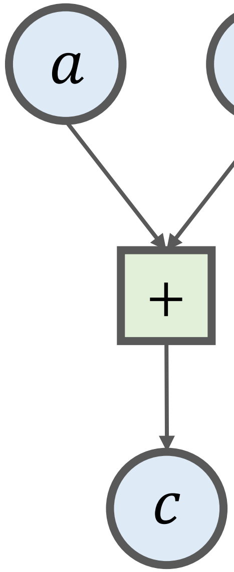
<u>세션 실행</u>



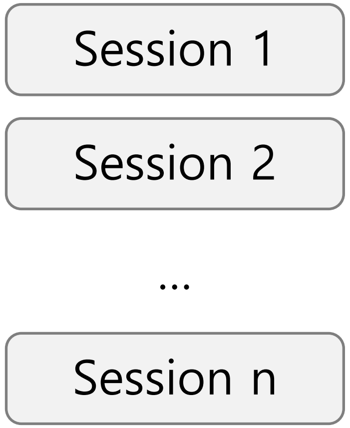
Define and Run

동일 계산 그래프를 여러 세션으로 동시에 실행 가능

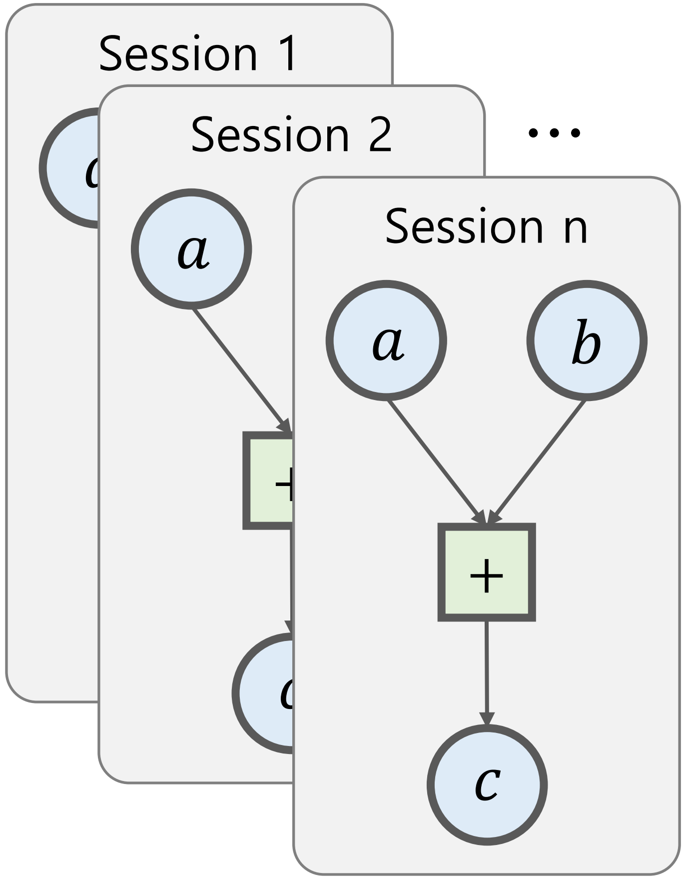
계산 그래프 정의



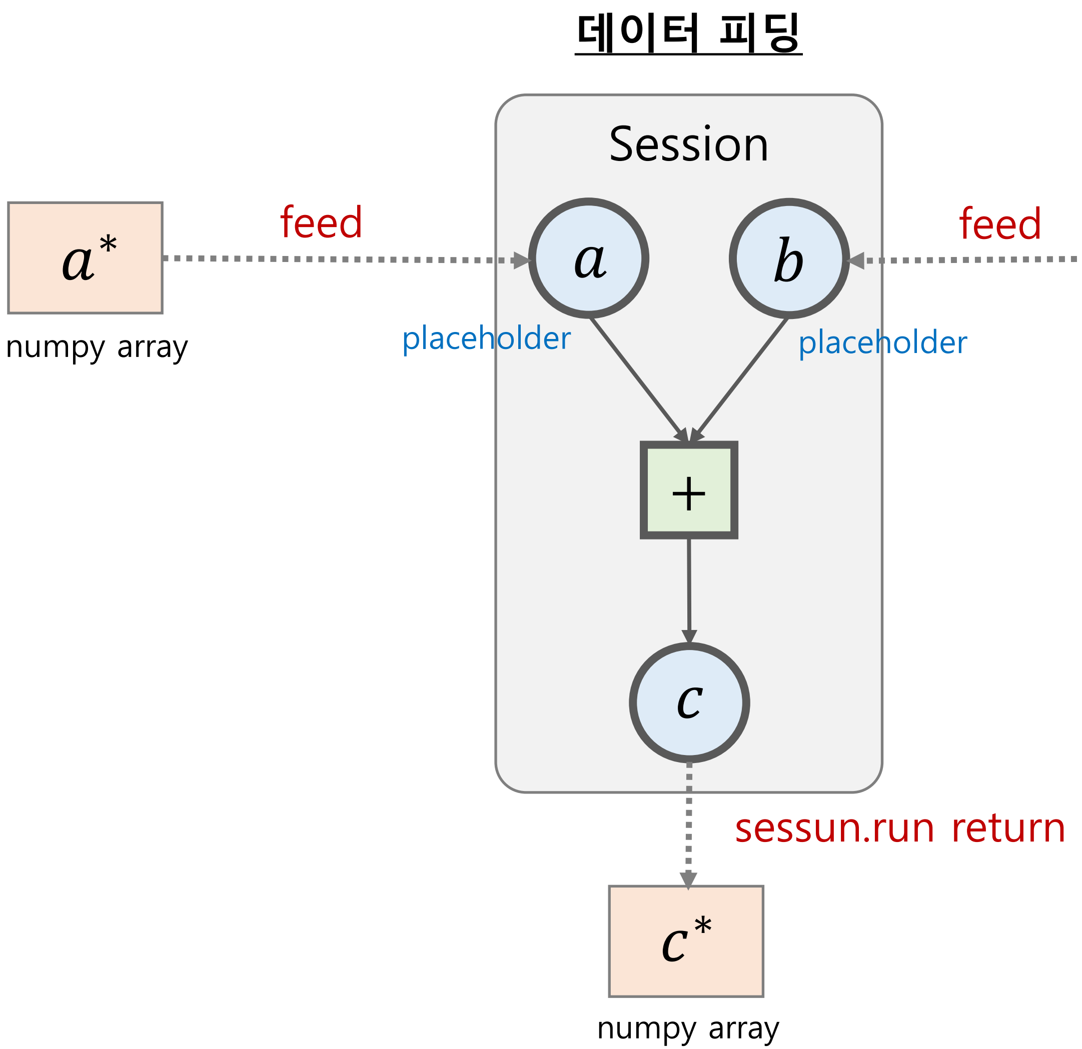
세션 생성



세션 실행



Define and Run



Tensor의 종류

PlaceHolder :
그래프 외부에서 데이터를 전달 받기위한 텐서

Variable :
그래프 내부 데이터를 저장하기 위한 텐서
초기화가 필요

Constant :
그래프 내부 상수 데이터를 갖고 있는 텐서

Define and Run

TensorFlow 1.x 코드 형태

```
# Basic computational graph
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.placeholder(tf.float32)

a = x * y
b = a + z
c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

계산 그래프 정의

Gradient 계산

세션 실행

장점:
- 실행 성능을 최적화 할 수 있다.
- 확장성이 좋다.

단점:
- 프로그램 방식이 익숙하지 않다.
- 디버깅이 어렵다.
- 조건에 따라 동적으로 변화하거나 반복적으로 확장되는 Dynamics Graph를 만들기 어렵다.

Static Graph 방식의 한계

Conditional

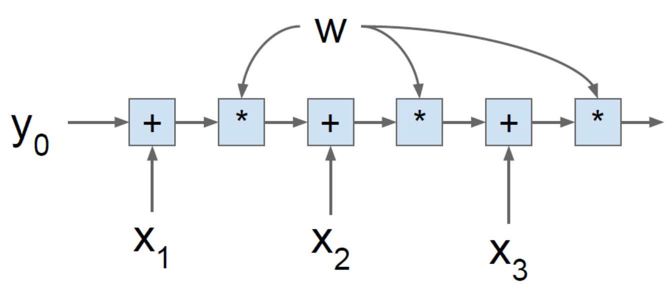
- 조건에 따라 실행되는 그래프가 변경되어야 하는 경우

$$y = \begin{cases} w1 * x & \text{if } z > 0 \\ w2 * x & \text{otherwise} \end{cases}$$

Loop

- 입력의 크기에 따라 그래프가 재귀적으로 확장되어야 하는 경우

$$y_t = (y_{t-1} + x_t) * w$$



Dynamic Graph 방식이 필요! Eager Execution

Eager Execution

- 1.x : tf.enable_eager_execution()
- 2.x: Default
- **Define by Run** support (like PyTorch, Chainer)
- Rapid Development
- **Easy Debugging** (use Python toolchain) → easy to check bottlenecks
- Native Control Flow (if, for etc) → easy to make complex model
- Boost performance by **AutoGraph**

Keras in TensorFlow2.0 by 박해선님

Eager Execution

Define and Run에서 Define by Run 으로!

```python
import tensorflow as tf

a = tf.constant(5)
b = tf.constant(3)

c = a + b

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

symbolic

```python
import tensorflow as tf

a = tf.constant(5)
b = tf.constant(3)

c = a + b

print(c)
```

concrete

TensorFlow 1.x :	8	Tensor("add_2:0", shape=(), dtype=int32)
TensorFlow 2.x :	Error	tf.Tensor(8, shape=(), dtype=int32)

Eager Execution

TensorFlow 1.x　　　z = w * x + b 구현　　　TensorFlow 2.x

```python
import tensorflow as tf

## 그래프 정의
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(dtype=tf.float32,
                       shape=(None), name='x')
    w = tf.Variable(2.0, name='weight')
    b = tf.Variable(0.7, name='bias')
    z = w * x + b
    init = tf.global_variables_initializer()

## 세션 생성 및 그래프 g 전달
with tf.Session(graph=g) as sess:
    ## w와 b 초기화
    sess.run(init)
    ## z 평가
    for t in [1.0, 0.6, -1.8]:
        print('x=%4.1f --> z=%4.1f'%(
              t, sess.run(z, feed_dict={x:t})))
```

```python
import tensorflow as tf

w = tf.Variable(2.0, name='weight')
b = tf.Variable(0.7, name='bias')

# ## z 평가
for x in [1.0, 0.6, -1.8]:
    z = w * x + b
    print('x=%4.1f --> z=%4.1f'%(x, z))
```

AutoGraph

tf.Graph() + tf.Session() → @tf.function

```
# TensorFlow 2.x
@tf.function
def simple_func():
    # complex computation with pure python
    ...
    return z

output = simple_func(input)
```

- for/while → tf.while_loop
- if → tf.cond

- @tf.function를 붙이면 그래프 생성해서 GPU나 TPU를 사용해서 작동
- @tf.function가 붙은 함수로부터 호출된 모든 함수들은 그래프 모드에서 동작
- 조건, 루프와 같은 제어문 사용 시 tf.cond, tf.while_loop와 같은 텐서플로 연산으로 변환

Keras in TensorFlow2.0 by 박해선님

AutoGraph

TensorFlow 1.x

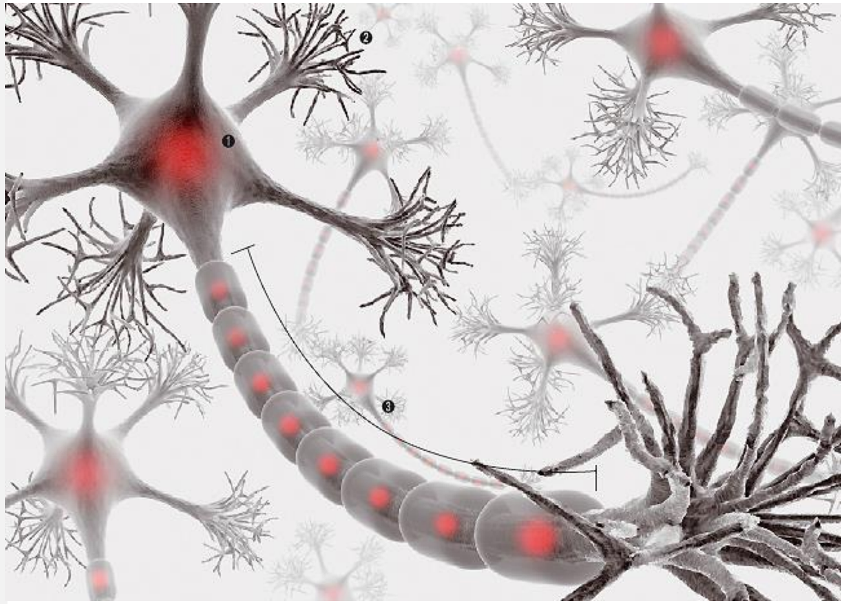
```
# 텐서플로 1.x 방식
g = tf.Graph()

# 그래프에 노드를 추가합니다.
with g.as_default():
...

g.as_graph_def()

node {
  name: "a"
  op: "Const"
  attr {
    key: "dtype"
    value {
      type: DT_INT32
    }
  }
```

TensorFlow 2.x

```
# 텐서플로 2.x 방식
@tf.function
def simple_func():
    ...
    return z

con_func = simple_func.get_concrete_function()
con_func.graph.as_graph_def()

node {
  name: "a"
  op: "Const"
  attr {
    key: "dtype"
    value {
      type: DT_INT32
    }
  }
```

Keras in TensorFlow2.0 by 박해선님

4 Keras



Keras

- **High-Level Neural Networks Specification** (https://keras.io) (2015. 03)
- Add to tf.contrib.keras at TensorFlow 1.2
- Promote to tf.keras at TensorFlow 1.4 (tf.layers → tf.keras)
- 1st Class Python API for TensorFlow 2.0
- Deprecated tf.layer, tf.contrib.layers(Slim)
- Keras 2.3.x is last major release of multi-backend Keras.
  Instead use tf.keras

Keras in TensorFlow2.0 by 박해선님

Class Hierarchy

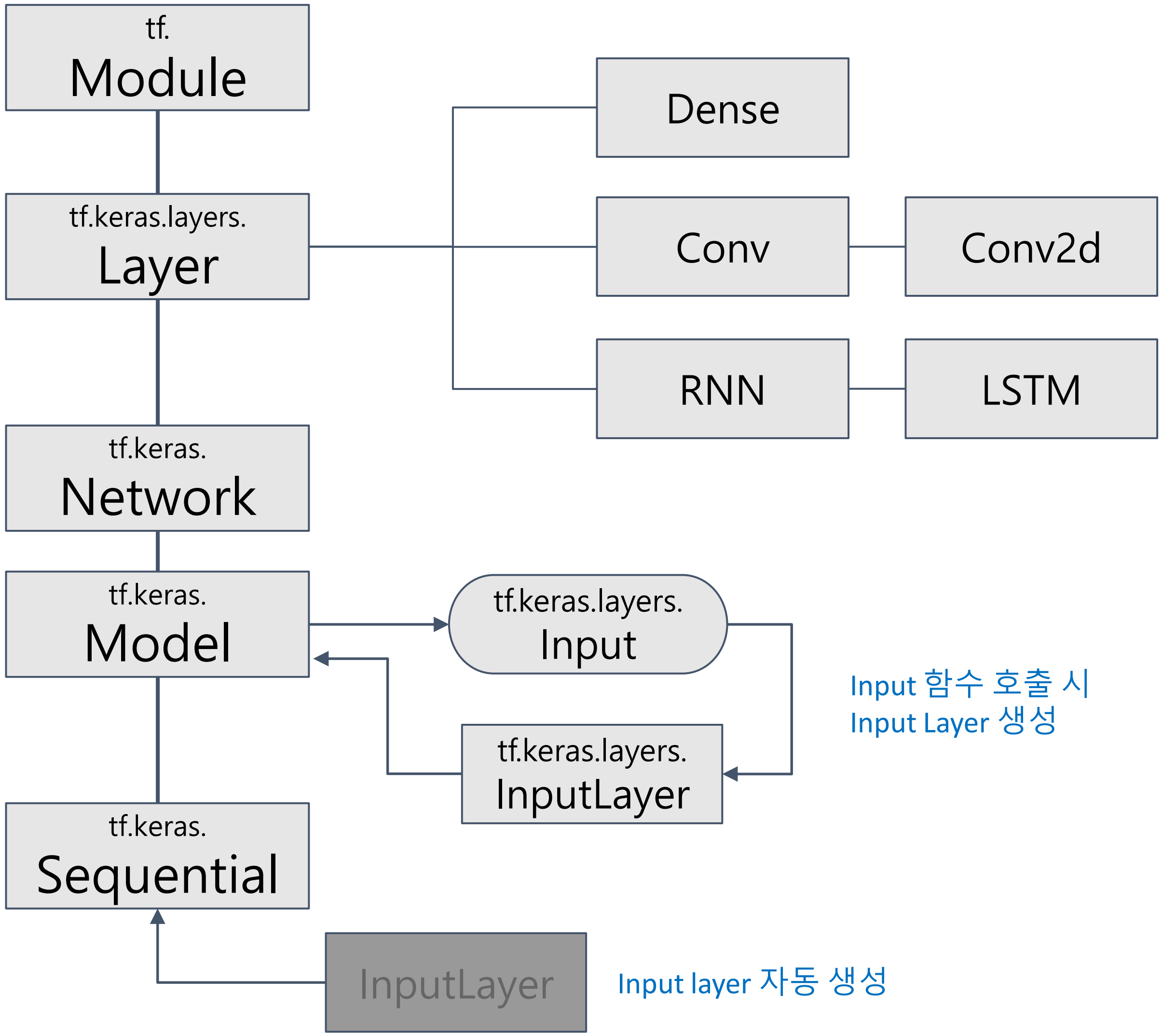
변수 컨테이너 (tf.Variable)
variables(), trainable_variables()

계층 정의 (파라미터, Forward Pass)
__call__() → build() → add_weights()
|→ call()
add_loss()

신경망 계층 통합
layers(), summary(), save()

모델 훈련/검증/테스트
compile(), fit(), evaluate(), predict()

순차 모델 구성
add()



Keras in TensorFlow2.0 by 박해선님

Keras 모델 정의

Simple Models

Sequential API + Built-in Layers

Functional API + Built-in Layers

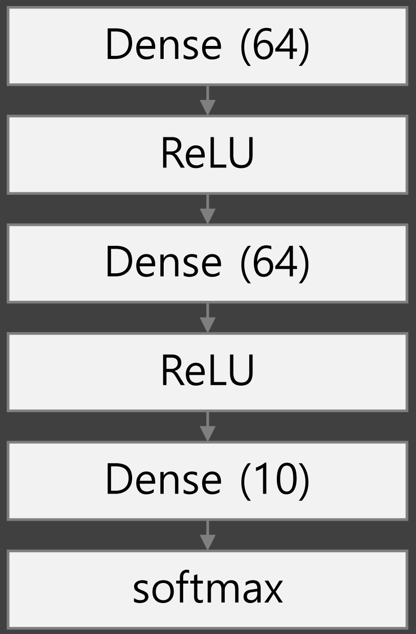
Functional API + Custom layers, Custom metrics, Custom losses

Subclassing

Complex Model

Sequential API

```
from tensorflow import tf

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# 훈련 설정
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 모델 훈련
model.fit(train_data, labels, epochs=10, batch_size=32)

# 모델 평가
model.evaluate(test_data, labels)

# 샘플 예측
model.predict(new_sample)
```




```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(10),
])
```

참고 tf.keras.layers

- **class Conv2D:** 2D convolution layer (e.g. spatial convolution over images).
- **class Dense**: Just your regular densely-connected NN layer.

Layer

- **class Flatten**: Flattens the input. Does not affect the batch size.
- class Reshape: Reshapes an output to a certain shape.
- class InputLayer: Layer to be used as an entry point into a Network (a graph of layers).

모양 변경

- class **MaxPool2D**: Max pooling operation for spatial data.
- class AveragePooling2D: Average pooling operation for spatial data.
- **class GlobalAveragePooling2D**: Global average pooling operation for spatial data.

Pooling

- **class BatchNormalization:** Normalize and scale inputs or activations. (Ioffe and Szegedy, 2014).
- class **Dropout**: Applies Dropout to the input.

정규화

- **class Embedding**: Turns positive integers (indexes) into dense vectors of fixed size.
- class SimpleRNN: Fully-connected RNN where the output is to be fed back to input.
- **class LSTM**: Long Short-Term Memory layer - Hochreiter 1997.
- class GRU: Gated Recurrent Unit - Cho et al. 2014.

RNN 계열

https://www.tensorflow.org/api_docs/python/tf/keras/layers

참고 tf.keras.layers

- class Softmax: Softmax activation function.
- **class ReLU**: Rectified Linear Unit activation function.
- class LeakyReLU: Leaky version of a Rectified Linear Unit.
- class ELU: Exponential Linear Unit.

Activation Function

https://www.tensorflow.org/api_docs/python/tf/keras/layers

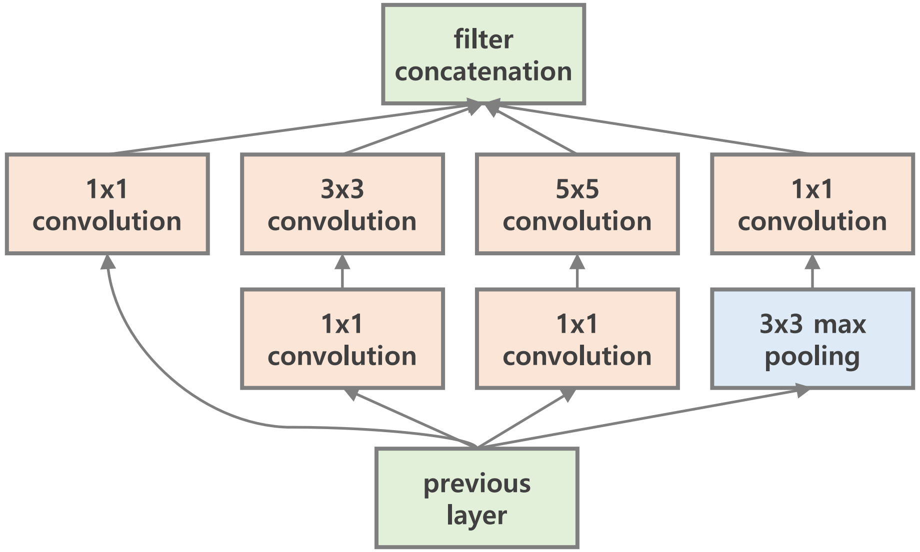
Functional API

```
from tensorflow import tf

# 입력과 출력을 연결해서 임의의 모델 그래프 생성
input = tf.keras.Input(shape=(784,), name='img’)  # 입력 플레이스 홀더 반환
h1 = tf.keras.layers.Dense(64, activation='relu')(inputs) # 각 계층 별로 Tensor를 전달하고 리턴 받음
h2 = tf.keras.layers.Dense(64, activation='relu')(h1)
output = tf.keras.layers.Dense(10, activation='softmax')(h2)

# 모델 생성
model = tf.keras.Model(input, output) # 입력 Tensor와 Output Tensor를 모델에 지정

# 훈련 설정
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                    loss='categorical_crossentropy',
                    metrics=['accuracy’])
…
```

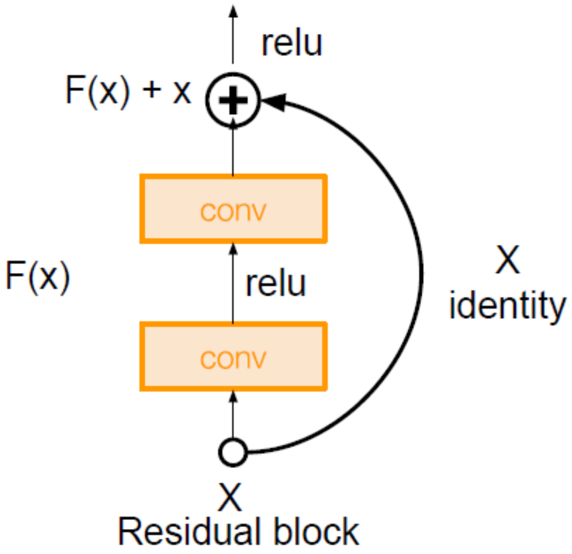
Keras in TensorFlow2.0 by 박해선님

Functional API

“Inception module”



“Residual block”



- 다중 입력 모델
- 다중 출력 모델
- 층을 공유하는 모델 (동일한 층을 여러 번 호출합니다)
- 데이터 흐름이 차례대로 진행되지 않는 모델 (예를 들면 잔차 연결(residual connections)).

Custom Layer

```
from tensorflow import tf
class MyLayer(tf.keras.layers.Layer):

    def __init__(self, units, activation=None, **kwargs):
        self.units = units
        self.activation = keras.activations.get(activation)
        super().__init__(**kwargs)

    def build(self, input_shape):
        self.weight = self.add_weight(name='kernel',
                                      shape=(input_shape[1], self.units),
                                      initializer='uniform')
        self.bias = self.add_weight(name='bias',
                                    shape=(self.units,),
                                    initializer='zeros')
        super().build(input_shape)

    def call(self, X):
        z = tf.matmul(X, self.weight) + self.bias

        return self.activation(z)
```

Keras in TensorFlow2.0 by 박해선님

Custom Model

```
from tensorflow import tf

class MyModel(tf.keras.Model):

    def __init__(self, **kwargs):
        self.hidden = MyLayer(10, activation="relu")
        self.output = MyLayer(1)
        super().__init__(**kwargs)

    def call(self, input):
        h = self.hidden(input)
        return self.output(h)

model = MyModel()
```

Keras in TensorFlow2.0 by 박해선님

Training 방식

Quick Experiment

model.fit()

model.fit()

callbacks

- Checkpoint
- Early stopping
- Tensorboard
- Slack notification

Iterate on the data

train_on_batch()

test_on_batch()

predict_on_batch()

- GAN
- Curriculum Learning

Custom Training Loop

GradientTape

- New optimization algorithm
- Learn to learn (meta learning)

Advanced Training

model.compile

훈련에 필요한 Optimizer, Loss, Metric을 설정하는 단계

1. 이름으로 지정 (Default 값으로 실행할 때)

회귀 모델 예시

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='mse',       # 평균 제곱 오차
              metrics=['mae'])  # 평균 절댓값 오차
```

2. 객체를 생성해서 전달 (파라미터를 지정할 필요가 있을 때)

분류 모델 예시

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.01),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])
```

참고 tf.keras.optimizers

- class SGD: Stochastic gradient descent and momentum optimizer.
- class Adagrad: Optimizer that implements the Adagrad algorithm.
- **class RMSprop**: Optimizer that implements the RMSprop algorithm.
- **class Adam**: Optimizer that implements the Adam algorithm.

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

참고 tf.keras.losses

- **class MeanSquaredError**: Computes the mean of squares of errors between labels and predictions.
- class MeanAbsoluteError: Computes the mean of absolute difference between labels and predictions.

- **class BinaryCrossentropy:** Computes the cross-entropy loss between true labels and predicted labels.
- **class CategoricalCrossentropy**: Computes the crossentropy loss between the labels and predictions.
- **class SparseCategoricalCrossentropy**: Computes the crossentropy loss between the labels and predictions.

https://www.tensorflow.org/api_docs/python/tf/keras/losses

참고 tf.keras.metrics

- **class Accuracy**: Calculates how often predictions matches labels.
- **class MeanAbsoluteError**: Computes the mean absolute error between the labels and predictions.
- **class MeanSquaredError**: Computes the mean squared error between y_true and y_pred.

https://www.tensorflow.org/api_docs/python/tf/keras/metrics

model.fit

모델을 고정된 epoch 수로 훈련

```
history = model.fit( normed_train_data, train_labels,
                     epochs=1000, validation_split = 0.2, verbose=0,
                     callbacks=[Earlystopping(),
                                Tensorboard(),
                                ModelCheckpoint()])
```

- **batch_size**: 배치 크기 (default 32)
- **epochs**: 총 epoch 수 (epoch는 training set을 한번 실행하는 단위)
- **validation_split**: training set에서 validation set으로 사용할 비율 ((0,1) 사이의 값)
- **verbose**: 훈련 진행 상황 모드 0 = silent, 1 = progress bar, 2 = one line per epoch
- **callbacks**: 훈련하면서 실행할 콜백 리스트

참고 tf.keras.callbacks

- **class EarlyStopping**: Stop training when a monitored quantity has stopped improving.
- **class ModelCheckpoint**: Save the model after every epoch.
- class TensorBoard: Enable visualizations for TensorBoard.
- class LearningRateScheduler: Learning rate scheduler.

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks

tf.GradientTape

```
@tf.function
def train_step(input, target):
 with tf.GradientTape() as tape:
   # forward Pass
   predictions = model(input)
   # compute the loss
   loss = tf.reduce_mean(
           tf.keras.losses.sparse_categorical_crossentropy(
           target, predictions, from_logits=True))
 # compute gradients
 grads = tape.gradient(loss, model.trainable_variables)
 # perform a gradient descent step
 optimizer.apply_gradients(zip(grads, model.trainable_variables))
 return loss
```

Forward Pass

Gradient 계산

Parameter Update

Keras + eager mode

- Keras use @tf.function by default.

```
class CustomModel(tf.keras.models.Model):

  @tf.function
  def call(self, input_data):
    if tf.reduce_mean(input_data) > 0:
      return input_data
    else:
      return input_data // 2
```

- For using eager mode,
  - model = Model(dynamic=True) or
  - model.compile(..., run_eagerly=True)

Keras in TensorFlow2.0 by 박해선님

Thank you!