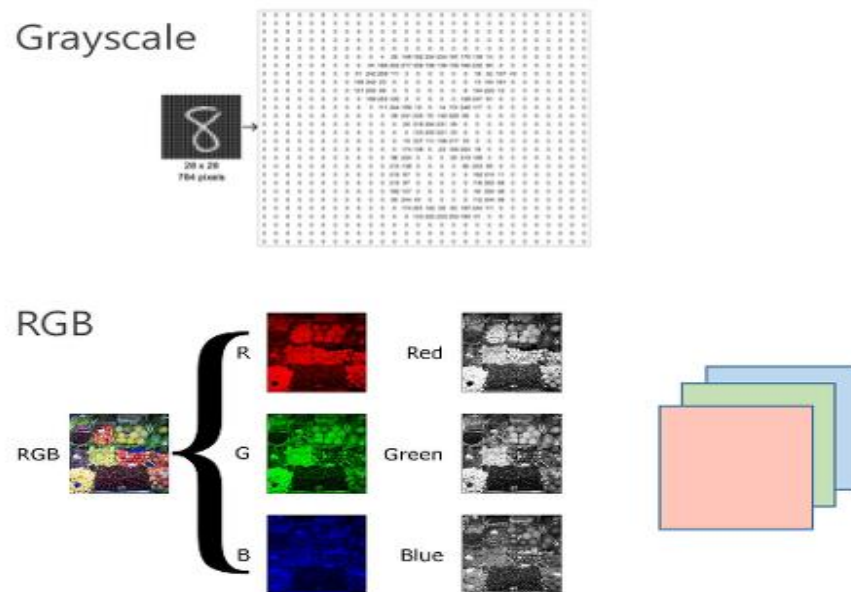
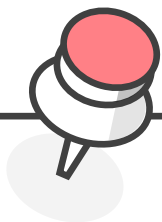


PPT PRESENTATION

-텐서플로우(Tensorflow) 란?

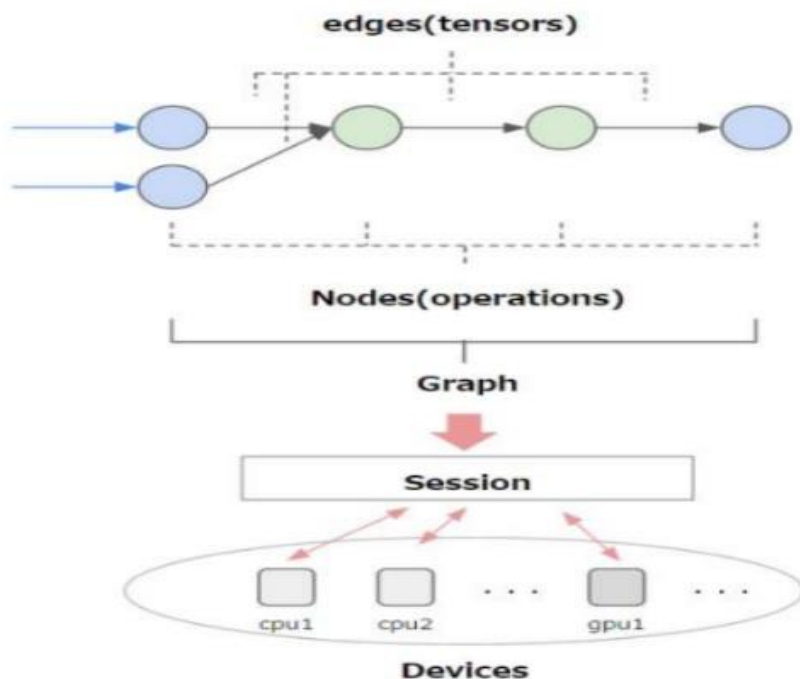
- 2015년에 오픈 소스로 공개된 구글(Google)에서 만든 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공하는 라이브러리
- Tensor(텐서)란 딥러닝에서 데이터를 표현하는 방식
- 행렬로 표현할 수 있는 2차원 형태의 배열을 높은 차원으로 확장한 다차원 배열
- 텐서 형태의 데이터들이 딥러닝 모델을 구성하는 연산들의 그래프를 따라 흐르면서 연산 진행
- Define and Run 방식





Tensorflow 구조

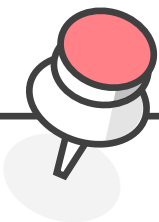
텐서플로우는 edges와 nodes로 구조화된 graph로 프로그램이 구성



```
import tensorflow as tf

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

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

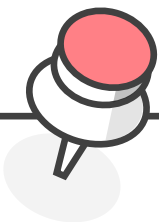


Tensorflow 란?

하나의 언어가 아닌 여러 언어에서 쓸 수 있다.

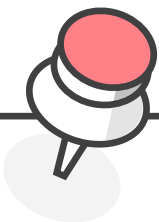
Productivity = efficient + flexibility + Scalability

간단하게 코딩 할 수 있어 새로운 방법에 적용, 구현이 쉽다.



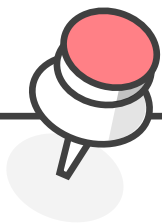
PPT PRESENTATION

- ▶ 데이터 플로우 그래프를 통한 풍부한 표현력
- ▶ 코드 수정 없이 CPU/GPU 모드로 동작
- ▶ 아이디어 테스트에서 서비스 단계까지 이용 가능
- ▶ 계산 구조와 목표 함수만 정의하면 자동으로 미분 계산을 처리
- ▶ Python/C++를 지원하며, SWIG를 통해 다양한 언어 지원 가능
- ▶ 커뮤니티가 형성되어있고, Torch -> TensorFlow migration도 이루어지고 있다.
- ▶ 구글에서 공식 릴리즈하였기 때문에, 그 전문성이 보장
- ▶ 텐서 보드(TensorBoard)를 통해서, 파라미터의 변화 양상이나 DNN에 대한 구조도를 그려줌으로써 Tensor들과의 연결관계, Tensor의 Flowing Status 를 잘 보여줌
- ▶ Threading이나 Queue 등의 메커니즘을 훨씬 디테일하게 사용



PPT PRESENTATION

- ▶ 메모리를 효율적으로 사용하지 못하고 있다
- ▶ intermediate layer의 값을 뽑아보려면 손이 많이 감 (그래프로 만들어줘야 하므로)
- ▶ 조건부 계층(layer)의 연산은 시간이 많이 소요됨
- ▶ 선언형 프레임워크 특성상 디버깅이 어렵다



PPT PRESENTATION

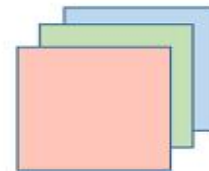
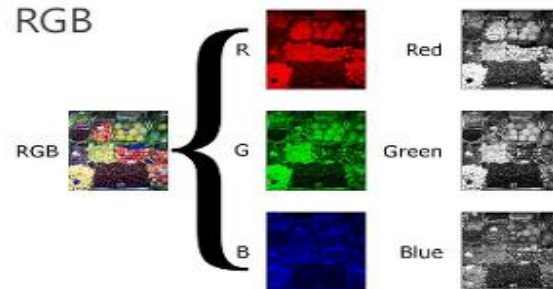
-파이토치(PyTorch) 란?

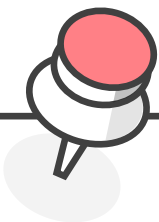
- 2016년에 발표된 딥러닝 구현을 위한 파이썬 기반의 오픈소스 머신러닝 라이브러리
- Facebook 인공지능 연구팀에 의해 개발
- Define by Run의 딥러닝 구현 패러다임
- 텐서 형태의 데이터들이 딥러닝 모델을 구성하는 연산들의 그래프를 따라 흐르면서 연산 진행
- NumPy를 대체하면서 GPU를 이용한 연산이 필요한 경우 사용
- 최대한의 유연성과 속도를 제공하는 딥러닝 연구 플랫폼이 필요한 경우 사용

Grayscale



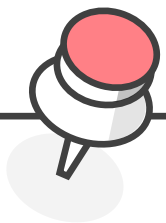
RGB





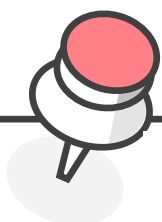
PPT PRESENTATION

- ▶ 설치가 간편하다
- ▶ 이해와 디버깅이 쉬운 직관적이고 간결한 코드로 구성되었다
- ▶ Define-by-Run 방식을 기반으로 한 실시간 결과값을 시각화한다
- ▶ 파이썬 라이브러리(Numpy, Scipy, Cython)와 높은 호환성을 가진다
- ▶ Winograd Convolution Algorithm 기본 적용을 통한 빠른 모델 훈련이 가능하다
- ▶ 모델 그래프를 만들 때 고정상태가 아니기 때문에 언제든지 데이터에 따라 조절이 가능하다
(유연성 지님)
- ▶ Numpy스러운 Tensor 연산이 GPU로도 가능하다
- ▶ Autograd System을 이용해 쉽게 DDN을 짤 수 있다
- ▶ 학습 및 추론 속도가 빠르고 다루기 쉽다

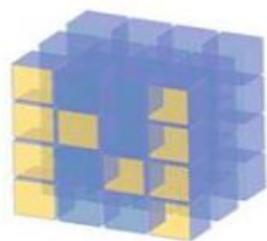


PPT PRESENTATION

▶ Tensorflow에 비해 커뮤니티 형성 ↓



PPT PRESENTATION



NumPy

VS

PYTORCH

```
In [4]: import numpy as np
from datetime import datetime
start = datetime.now()

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)

a = x * y
b = a * z
c = np.sum(b)

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

print(grad_x)
print(grad_y)
print(grad_z)
print(datetime.now()-start)

[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985  0.14404357  1.45427351]]
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186  0.8644362  -0.74216502]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.003751
```

```
In [6]: import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(), requires_grad=True)
y = Variable(torch.randn(N,D).cuda(), requires_grad=True)
z = Variable(torch.randn(N,D).cuda(), requires_grad=True)

a = x * y
b = a * z
c = torch.sum(b)

c.backward(torch.cuda.FloatTensor([1.0]))

print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)

Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
0.1152  1.1809  1.4522  1.9417
1.0845  0.1587 -1.6526  0.4031
1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
1 1 1 1
1 1 1 1
1 1 1 1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```



VS

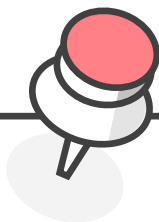
PYTORCH

0:00:00.003751

Gradient Calculation

0:00:00.003434

거리는 시간은 비슷(?)



PPT PRESENTATION



VS



```
In [5]: import tensorflow as tf
import numpy as np
from datetime import datetime
start = datetime.now()

with tf.device('/gpu:0'):
    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

print(grad_x_val)
print(grad_y_val)
print(grad_z_val)
print(datetime.now()-start)

[[-1.6138978 -0.21274029 -0.89546657  0.38690251]
 [-0.51080513 -1.18063223 -0.02818223  0.42833188]
 [ 0.06651722  0.30247191 -0.63432211 -0.36274117]]
[[ 1.23029065  1.20237982 -0.38732681 -0.30230275]
 [-1.04855299 -1.42001796 -1.70627022  1.95077538]
 [-0.5096522  -0.43807429 -1.25279534  0.77749038]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.046684
```

Define and Run

Define by Run

```
In [6]: import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(), requires_grad=True)
y = Variable(torch.randn(N, D).cuda(), requires_grad=True)
z = Variable(torch.randn(N, D).cuda(), requires_grad=True)

a = x * y
b = a * z
c = torch.sum(b)

c.backward(torch.cuda.FloatTensor([1.0]))

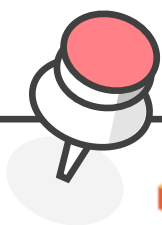
print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)

Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
0.1152  1.1809  1.4522  1.9417
1.0845  0.1587 -1.6526  0.4031
1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
1 1 1 1
1 1 1 1
1 1 1 1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

0:00:00.003434
```



PPT PRESENTATION

TensorFlow™

VS

PYTORCH

```
In [5]: import tensorflow as tf
import numpy as np
from datetime import datetime
start = datetime.now()

with tf.device('/gpu:0'):
    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

print(grad_x_val)
print(grad_y_val)
print(grad_z_val)
print(datetime.now()-start)
```

```
[[[-1.6138978 -0.21274029 -0.89546657 0.38690251]
 [-0.51080513 -1.18063223 -0.02818223 0.42833188]
 [ 0.06651722 0.30247191 -0.63432211 -0.36274117]]
 [[ 1.23029065 1.20237982 -0.38732681 -0.30230275]
 [-1.04855299 -1.42001796 -1.70627022 1.95077538]
 [-0.5096522 -0.43807429 -1.25279534 0.77749038]]
 [[ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]
 [ 1. 1. 1. 1.]]
0:00:00.046684
```

Define and Run

Define by Run

```
In [6]: import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(), requires_grad=True)
y = Variable(torch.randn(N,D).cuda(), requires_grad=True)
z = Variable(torch.randn(N,D).cuda(), requires_grad=True)

a = x * y
b = a * z
c = torch.sum(b)

c.backward(torch.cuda.FloatTensor([1.0]))

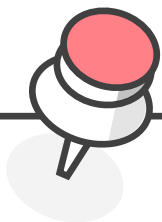
print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

```
Variable containing:
-0.6048 0.6640 -1.8035 1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247 0.6293 2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
0.1152 1.1809 1.4522 1.9417
1.0845 0.1587 -1.6526 0.4031
1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]

Variable containing:
1 1 1 1
1 1 1 1
1 1 1 1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]
0:00:00.003434
```

TF가 10배 이상 느리다??



PPT PRESENTATION



VS




[tensorflow / tensorflow](#)

[Watch](#) 5,262 [Unstar](#) 59,483 [Fork](#) 28,477

[Code](#) [Issues 1,153](#) [Pull requests 103](#) [Projects 0](#) [Insights](#)

pytorch 2.5x faster on VGG16 #7065

[Closed](#) SeguinBe opened this issue on 26 Jan · 20 comments



SeguinBe commented on 26 Jan · edited

What related GitHub issues or StackOverflow threads have you found by searching the web for your problem?


Started on SO, and was told to post here ([SO post](#))

Environment info

Operating System:
Ubuntu 14.04 + Maxwell Titan X

Installed version of CUDA and cuDNN:
CUDA 8.0, cuDNN 5.1

Assignees

 tfboyd

Labels

type:bug/performance

Projects

None yet

Milestone

No milestone

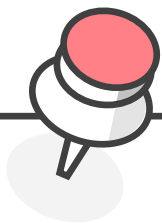


SeguinBe commented on 27 Jan · edited

So I think that was mainly the solution, the tensorflow definition of the network was using a convolution instead of the fully connected linear matrix multiplication for fc6 fc7 fc8 ([here](#)). Did not think originally it would be a big problem but to recapitulate :

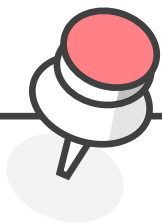
Model	Timing
TF-slim default	160ms
TF-slim + NCHW	150ms
fc layers instead of conv	94ms
fc layers instead of conv + NCHW	82ms
pytorch	65ms

There is still a gap but it is definitely more acceptable, should we consider this as resolved?



PPT PRESENTATION

구분	Tensorflow	Pytorch
패러다임	Define and Run	Define by Run
그래프 형태	Static graph	Dynamic graph
현재 사용자	많음	적음
자체 운영 포럼	없음	있음
한국 사용자 모임	Tensorflow Korea(TF-KR)	Pytorch Korea(Pytorch-KR)



PPT PRESENTATION

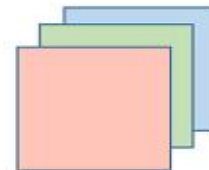
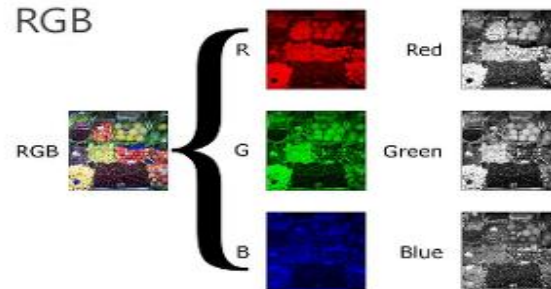
-케라스(Keras) 란?

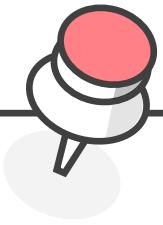
- 파이썬 기반의 오픈소스 머신러닝 라이브러리
- Keras는 Tensorflow 위에서 동작하는 프레임워크
- 딥 신경망과의 빠른 실험을 가능케 하도록 설계
- 최소한의 모듈 방식의 확장 가능성에 초점

Grayscale



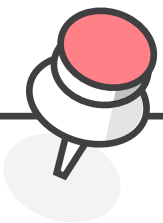
RGB





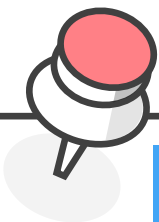
PPT PRESENTATION

- ▶ 빠른 시간 내에 프로토타이핑 가능
- ▶ 이해와 디버깅이 쉬운 직관적이고 간결한 코드로 구성되었다
- ▶ Tensorflow, MXNet, TypeScript, JavaScript, CNTK 등 프레임워크 지원
- ▶ 사용자 친화적 인터페이스



PPT PRESENTATION

- ▶ Tensorflow를 사용하는 쪽이 훨씬 더 **디테일한 조작**이 가능
- ▶ Pytorch가 더 우수한 디버깅 기능 제공, 훈련속도 우세



PPT PRESENTATION

```
In [4]: (train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5
BUFFER_SIZE = 60000
BATCH_SIZE = 256
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

▶ tf.keras.dataset

케라스에서 불러온 데이터셋을 $[-1, 1]$ 로 normalize

▶ Train_dataset

60000개 단위로 섞고 256개의 샘플 단위마다 모델의 가중치를 업데이트

```
In [5]: def make_generator_model():
model = tf.keras.Sequential() #여러개의 레이어
model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
```

```
In [7]: def make_discriminator_model():
model = tf.keras.Sequential() #여러개의 레이어
model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
input_shape=[28, 28, 1]))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))
```



Training data



(Real) data



Latent sample, z



Generator

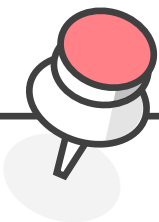


(Fake) data



Discriminator

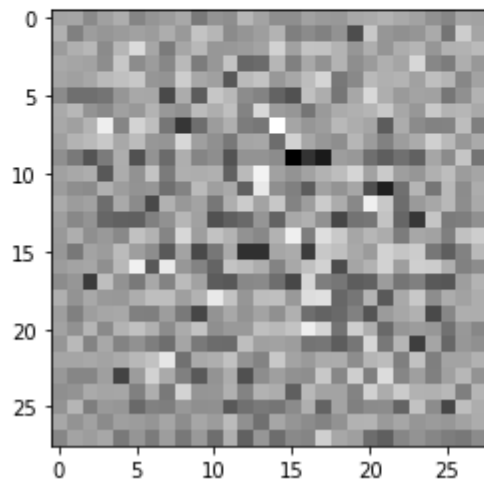
0	fake
1	real



PPT PRESENTATION

```
In [6]: generator = make_generator_model()
noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
Out [6]: <matplotlib.image.AxesImage at 0x298d6380820>
```



▶ `tf.random.normal`

정규 분포를 통한 난수를 노이즈로 두고
Generator 함수 통한 이미지 생성

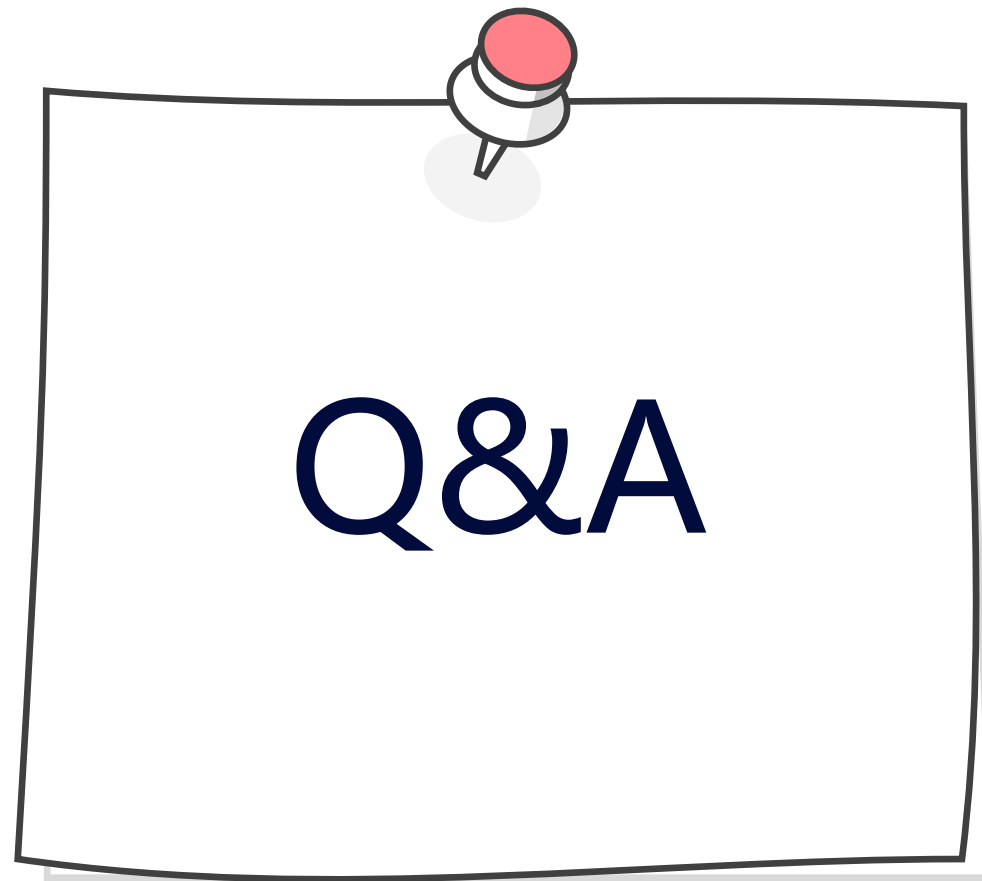
▶ `tf.keras.losses.Binarycrossentropy`

이미지 분류가 가능하도록 loss 함수를 이용한
새로운 Discriminator를 정의

```
In [8]: discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

tf.Tensor([[0.00091641]], shape=(1, 1), dtype=float32)
```

Discriminator는 해당 생성 이미지를 보고 판단!



감사합니다.

감사합니다.