Generative Adversarial Network (GAN)

https://youtu.be/Qy3PGvwGCYc

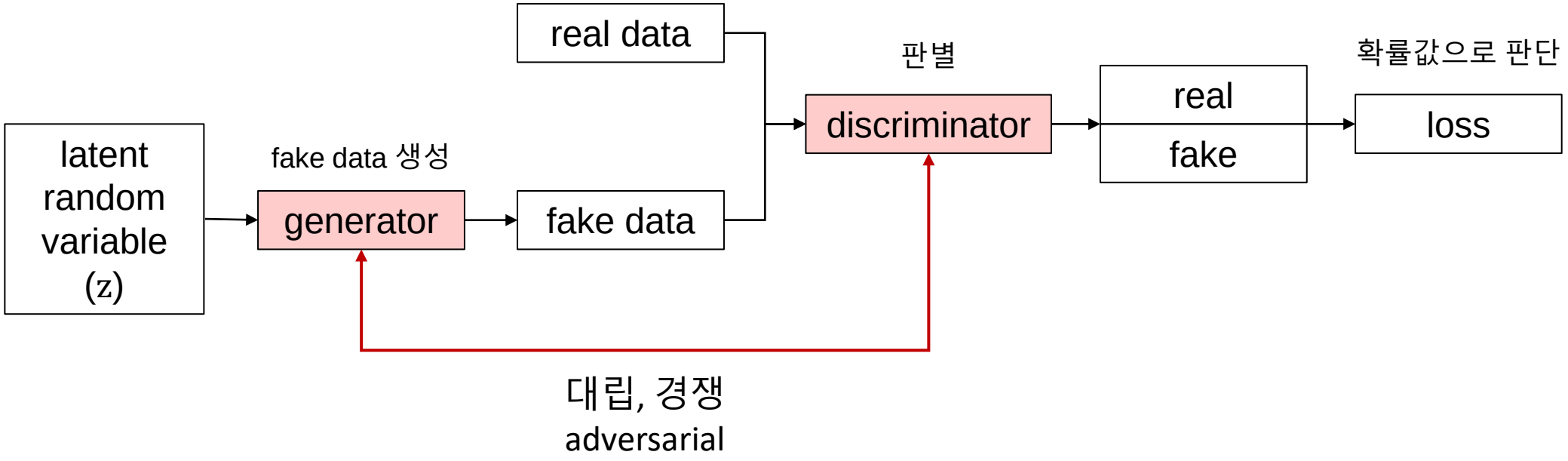
Contents

CryptoCraft LAB

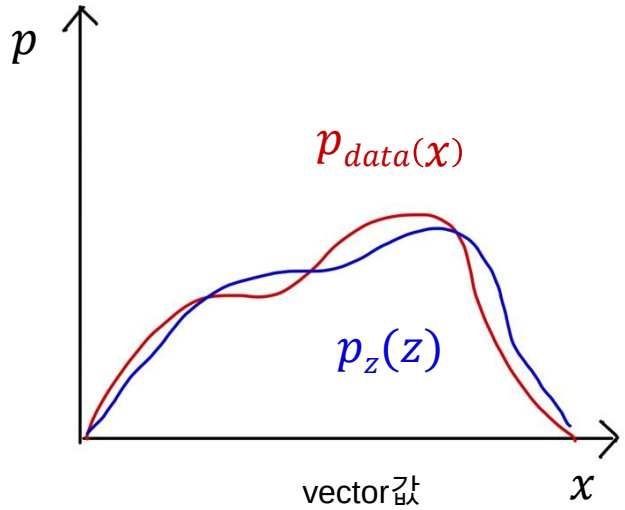
Generative Adversarial Network

- 비지도학습(Unsupervised learning)과 지도학습(Supervised learning)으로 구성
- 두 신경망이 서로 경쟁하며 generator는 점점 real data 같은 데이터를 생성하고 discriminator의 분류 성능은 점점 더 향상되도록 하는 것이 목적

→ generator 1번, discriminator 1번 번갈아가며 학습



Generative Adversarial Network



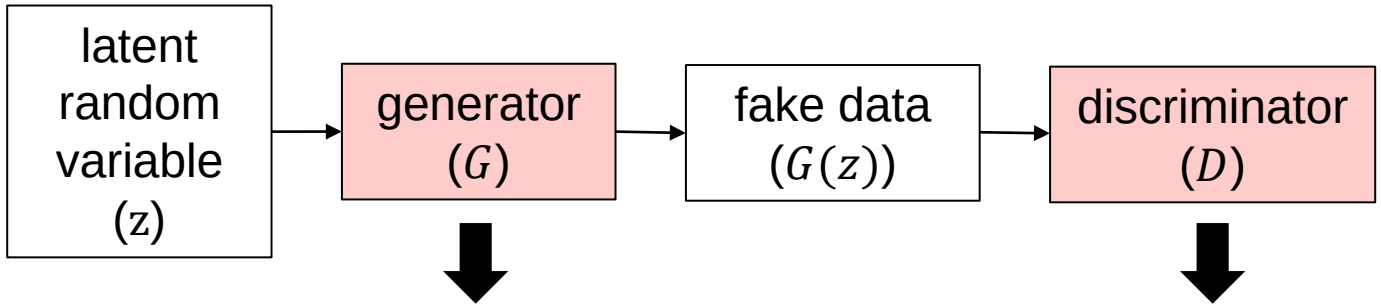
- data의 특징을 나타내는 vector값의 분포 (p)
- real data의 분포($p_{data}(x)$)와 fake data의 분포($p_z(z)$)를 비슷하게 만드는 것이 목적
  → label을 통한 분류가 아닌 training data의 분포를 학습

➢ 확률 분포가 정확히 일치하면 real data와 fake data를 구분할 수 없음

Generative Adversarial Network

random vector, noise (z)

1. 보통 uniform distribution
2. 고차원 벡터 → z=100인 경우 100차원 vector



object function(loss)

- **discriminator (D)**

$$min_G max_D V(D,G) = \mathrm{E}_{x\sim pdat_{a(x)}}[\log D(x)] + \mathrm{E}_{z\sim pz(z)}[\log(1 - D(G(z)))]$$

1. 실제 데이터로부터 뽑은 특징 벡터 x는 $D(x) = 1$이 되도록 학습

   → real data가 real data로 분류되도록

2. 임의의 z(random vector)로부터 만들어진 fake data에 대해 $D(G(z)) = 0$이 되도록 학습

   → fake data를 fakedata로

- **generator (G)**

$$min_G max_D V(D,G) = \mathrm{E}_{x\sim pdat_{a(x)}}[\log D(x)] + \mathrm{E}_{z\sim pz(z)}[\log(1 - D(G(z)))]$$

$D\big(G(z)\big) = 1$ 이 되도록 하는 것이 목적 (최소화)

→ fake data를 real data로 착각하도록 (discriminator를 속이기 위함)

* $\mathrm{E}_{x\sim pdata_{(x)}}$, $\mathrm{E}_{z\sim pz(z)}$
  real data, fake data에 대한 확률 밀도 함수

original GAN의 한계

- 복잡한 이미지에 대해서는 그다지 좋지 않은 성능을 보임
- 학습 시 안정성이 떨어짐
  - real vs fake만을 분류

    → sigmoid 활성화 함수

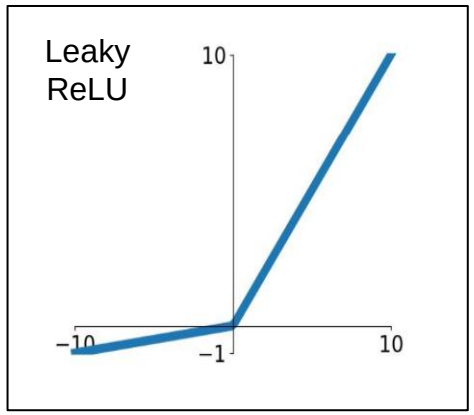
    → 기울기 소실 문제 (0~1 사이의 기울기들이 곱해져 0으로 수렴 → 학습 저하)

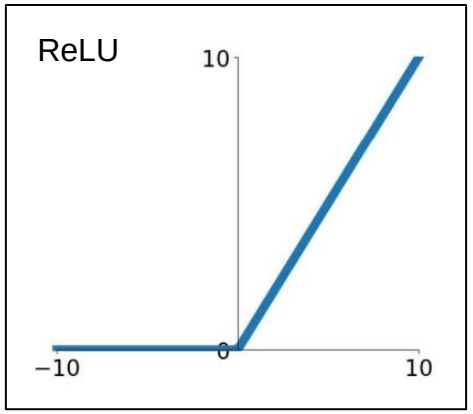
    → 이미지 품질 저하
- 이러한 문제 해결을 위해 CNN을 적용한 Deep Convolutional GAN (DCGAN)을 고안
- 그 외에도 여러 형태의 GAN 등장

  LSGAN, SGAN, ACGAN, WGAN, BEGAN …

variants of GANs

- **Deep Convolutional GAN (DCGAN)**
  - 간단하면서 선호되는 모델
  - CNN + 다음과 같은 방법 적용하여 성능 개선 → 고화질 이미지
    1. MaxPooling layer 제거
    2. Batch normalization 적용
    3. Fully connected hidden layer 제거
    4. 활성화 함수
       - tanh, sigmoid
       - Leaky ReLU, ReLU





*Leaky ReLu

ReLu에서 $x < 0$인 경우, 뉴런이 죽는 현상을 해결하기 위함

$f(x) = max(0.01\,x, x)$: 0.01 등 매우 작은 값 사용

$x < 0$ 에서 기울기가 0이 되지 않음

variants of GANs

- **Least Squares GAN (LSGAN)**
  - discriminator를 속여 real data로 분류된다 하더라도 original GAN의 경우 기울기 소실로 인해 실제 데이터와 차이 존재
  - 거리에 따른 패널티를 주어 실제 데이터와 근접(비슷)하도록 생성
  - D를 위한 object(loss) function을 least squares로 대체

$$min_G max_D V(D,G) = \mathrm{E}_{x \sim pdat_{a(x)}}[\log D(x)] + \mathrm{E}_{z \sim pz(z)}[\log(1 - D(G(z)))]$$

*original GAN의 object function

$$min_D V_{\mathrm{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{x \sim p_{data}(x)}[(D(x) - b)^2] + \frac{1}{2}\mathbb{E}_{x \sim p_z(z)}[D(G(z)) - a)^2]$$

$$min_G V_{\mathrm{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{x \sim p_z(z)}[(D(G(z)) - c)^2]$$

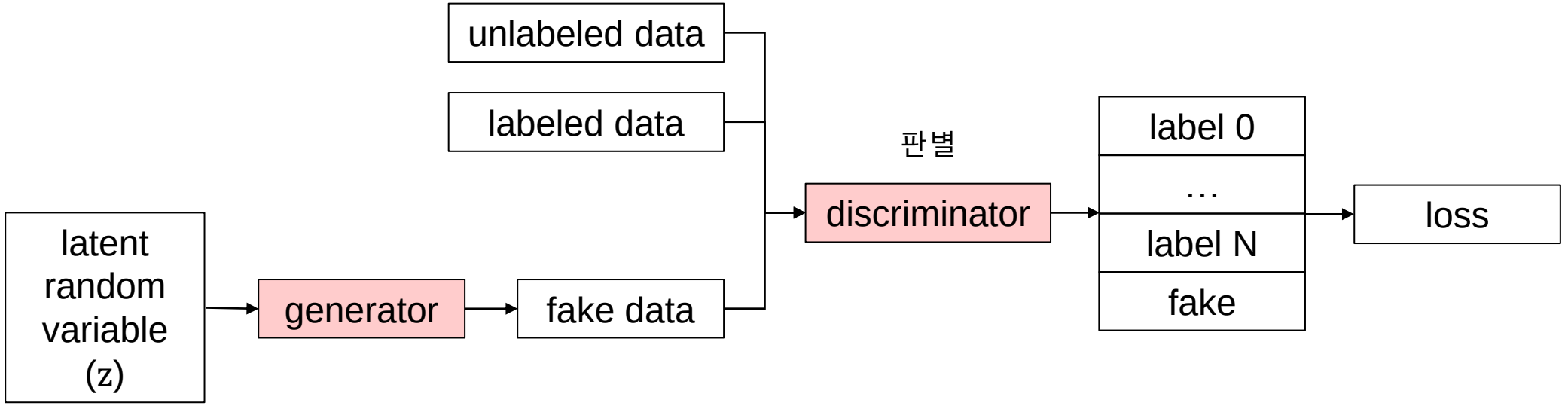
*LSGAN의 object function

variants of GANs

- **Semi-Supervised GAN (SGAN)**
  - discriminator에서 real vs fake가 아니라 classification 가능

    → N개의 class와 1개의 fake data로 분류

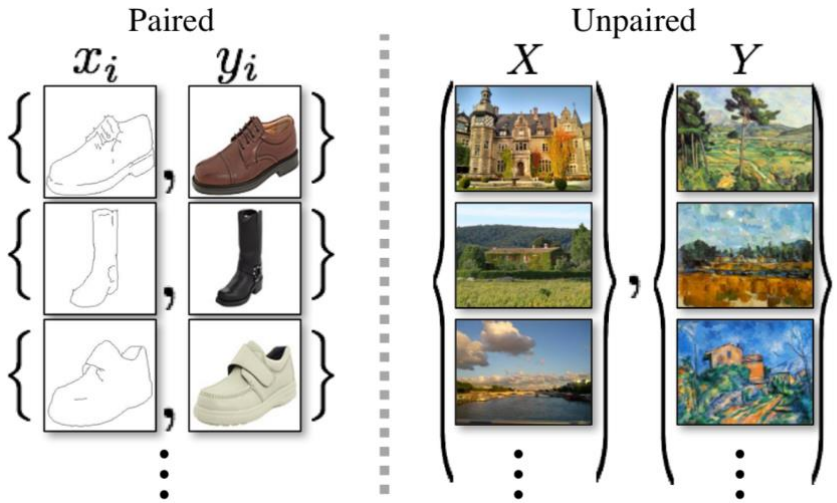
    → softmax 활성화 함수 사용
  - DCGAN을 기본 구조로 함



Extensions of GANs

- **CycleGAN : Unpaired Image to Image Translation**
  - 입출력 데이터가 pair가 아닌 경우

    입력으로 출력 생성, 만들어진 출력을 입력으로 다시 사용
  - 모양을 바꾸기는 어려움



완전히 같은 형태가 아님

Extensions of GANs

- **StackGAN**
  - 텍스트를 입력하면 그에 해당하는 이미지를 생성
  - stage를 나누어 저해상도(stage 1) → 고해상도 + 세부정보 (stage 2) → 사실적
  - stage 1, 2는 각각 generator와 discriminator로 구성

부리가 짧고 몸통이 녹색과 검은색인 새

generator

```
def create_g(self):
    self.G = Sequential()
    dropout = 0.4
    depth = 64+64+64+64
    dim = 8
    self.G.add(Dense(dim*dim*depth, input_dim=self.noise_size))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))
    self.G.add(Reshape((dim, dim, depth)))
    self.G.add(Dropout(dropout))
    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/2), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))
    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/4), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))
    self.G.add(Conv2DTranspose(int(depth/8), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))
    self.G.add(Conv2DTranspose(self.channel, 5, padding='same'))
    self.G.add(Activation('sigmoid'))
    self.G.summary()

    return self.G
```

G(generator) 모델 생성 후

필요한 layer 추가하여 구성

Deep Convolutional GAN

→ convolution layer 사용

discriminator

```
def create_d(self):
    self.D = Sequential()
    depth = 64
    dropout = 0.4
    self.D.add(Conv2D(depth*1, 5, strides=2, input_shape=self.input_shape, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))
    self.D.add(Conv2D(depth*2, 5, strides=2, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))
    self.D.add(Conv2D(depth*4, 5, strides=2, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))
    self.D.add(Conv2D(depth*8, 5, strides=1, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))
    self.D.add(Flatten())
    self.D.add(Dense(1))
    self.D.add(Activation('sigmoid'))
    self.D.summary()
    return self.D
```

D(discriminator) 모델 생성 후

필요한 layer 추가하여 구성

은닉층 활성화 함수

LeakyReLU

real vs fake : 2진 분류

DCGAN model

```
# Build model to train D.
optimizer = Adam(lr=0.0008)
self.D.compile(loss='binary_crossentropy', optimizer=optimizer)

# Build model to train G.
optimizer = Adam(lr=0.0004)
self.D.trainable = False
self.AM = Sequential()
self.AM.add(self.G)
self.AM.add(self.D)
self.AM.compile(loss='binary_crossentropy', optimizer=optimizer)
```

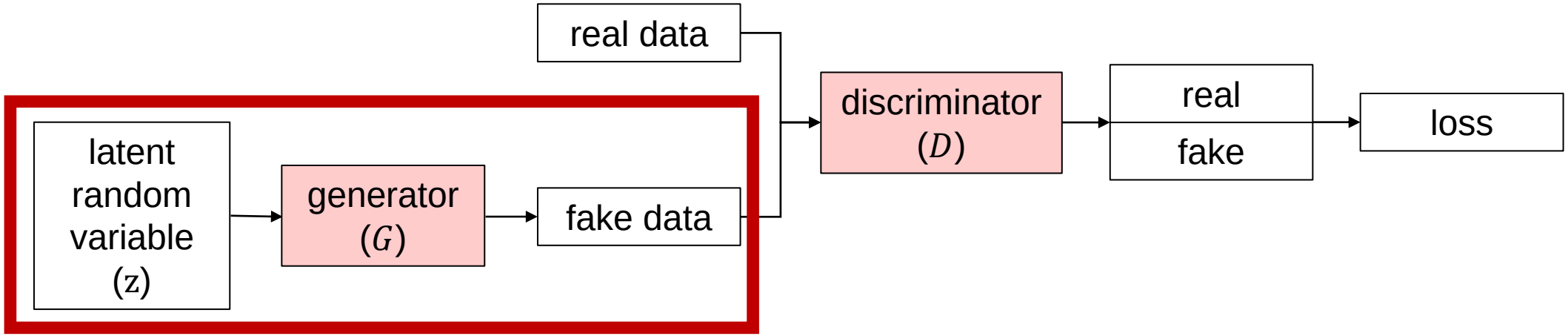
generator와 이미 학습된 discriminator을 연결 후 compile

training

latent random variable
(random vector z)

```
noise = np.random.uniform(-1.0, 1.0, size=[batch_size, self.noise_size])
images_fake = self.G.predict(noise)
```

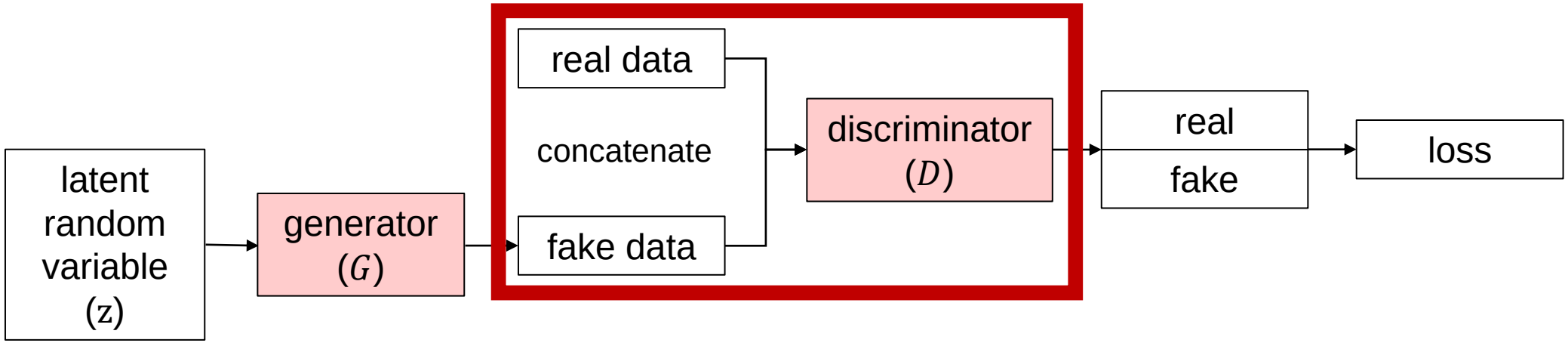
generator 통해 random vector를 예측하여 fake data 생성



training

real data(training data)와 앞서 생성한 fake data를 학습

```
# Train D.
x = np.concatenate((images_train, images_fake))
y = np.ones([2*batch_size, 1])
y[batch_size:, :] = 0
self.D.trainable = True
d_loss = self.D.train_on_batch(x, y)
```

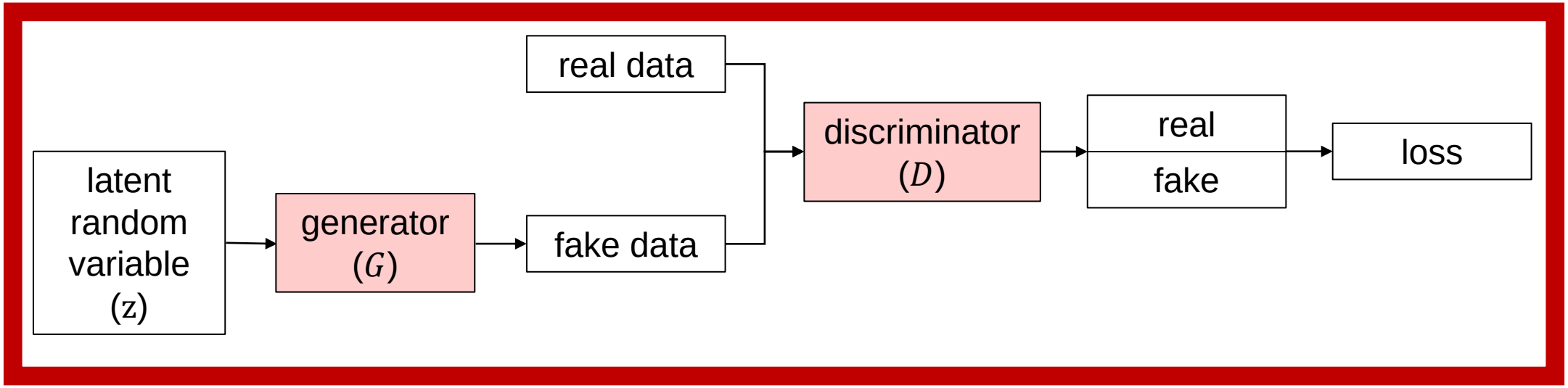


training

새로운 random vector 생성

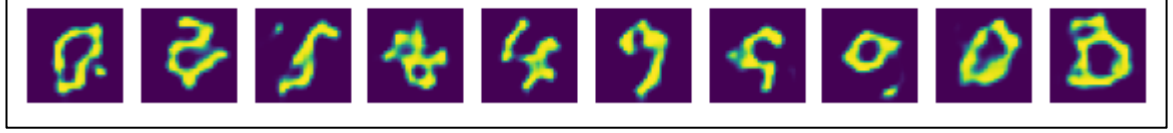
```
# Train G.
y = np.ones([batch_size, 1])
noise = np.random.uniform(-1.0, 1.0, size=[batch_size, self.noise_size])
self.D.trainable = False
a_loss = self.AM.train_on_batch(noise, y)
```

DCGAN 모델 학습

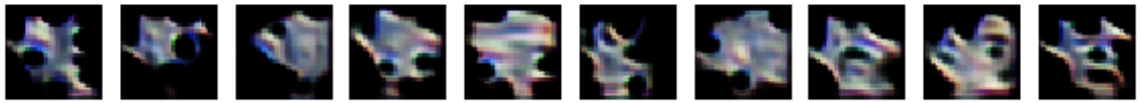


result

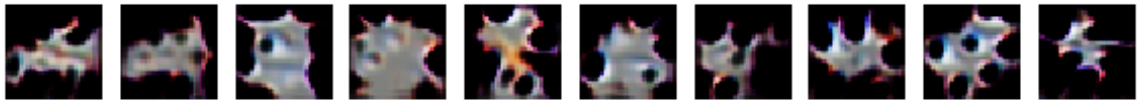
D Loss: 0.09376799690776579, AM Loss: 8.693308387001355



Epoch: 220, D Loss: 0.26622369419783354, AM Loss: 3.145884335041046

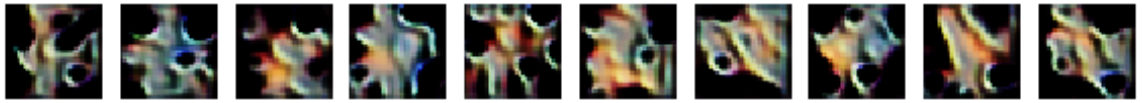


Epoch: 240, D Loss: 0.361137792468071, AM Loss: 1.8742984682321548

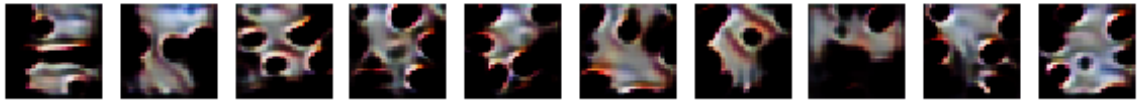


Epoch: 260, D Loss: 0.24683966673910618, AM Loss: 2.22111377120018

Epoch: 280, D Loss: 0.31206763163208960, AM Loss: 2.227735549211502



Epoch: 300, D Loss: 0.2658086307346821, AM Loss: 3.7545968890190125

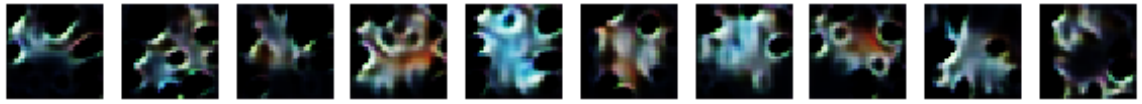


Epoch: 320, D Loss: 0.1451576016843319, AM Loss: 4.480963855981827

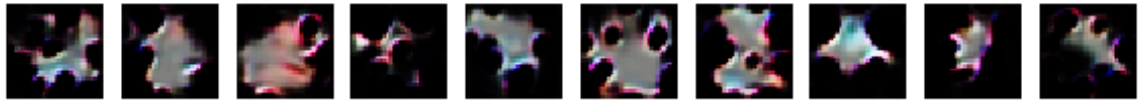
Epoch: 340, D Loss: 0.5095665194094181, AM Loss: 2.240627035498619

Epoch: 360, D Loss: 0.1802350990474224, AM Loss: 2.7137776017189026

Epoch: 380, D Loss: 0.3175997529178858, AM Loss: 3.1885876059532166



Epoch: 400, D Loss: 0.15155761037021875, AM Loss: 4.314605861902237



Q & A