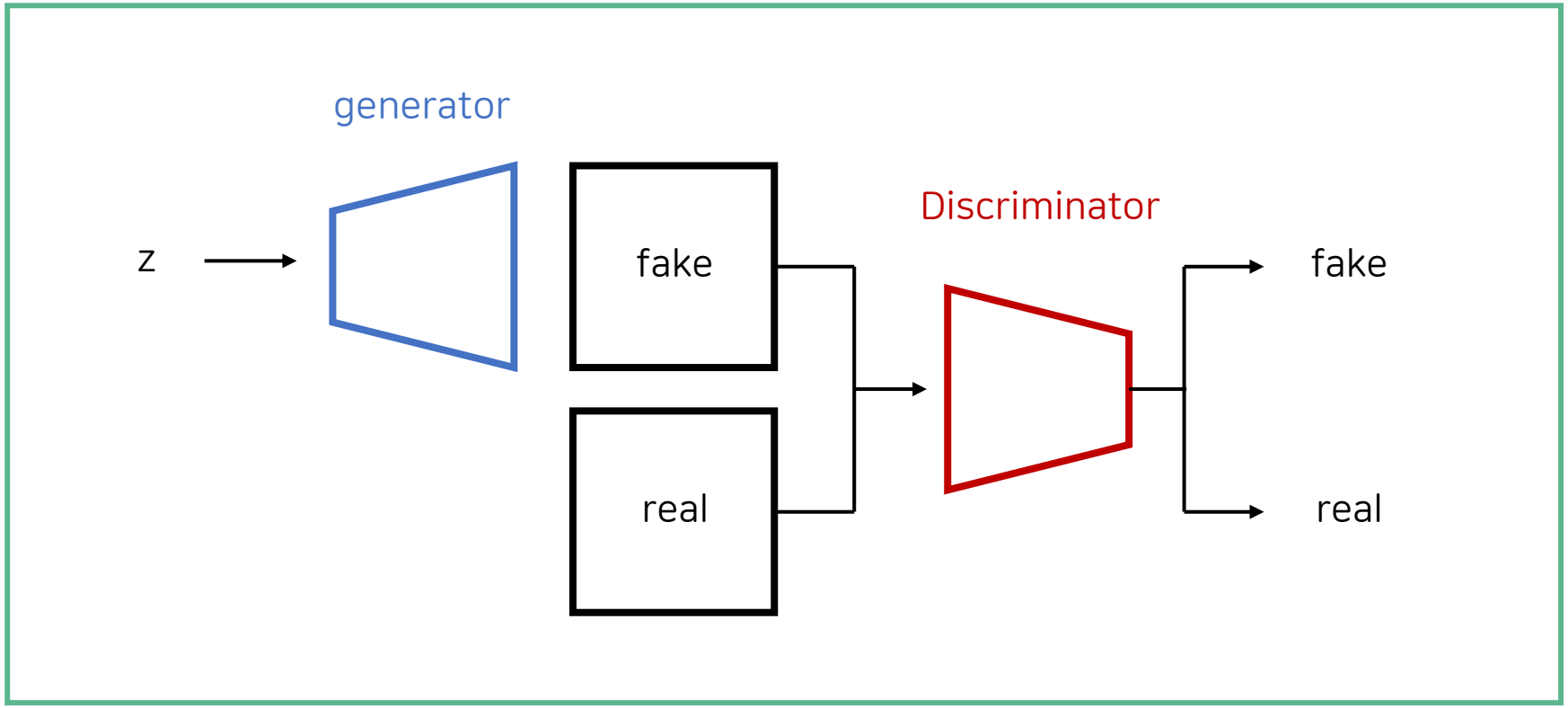
파이토치 첫걸음

chapter 10. 생성적 적대 신경망



GAN (Generative Adversarial Network)

"Generative Adversarial Network (생성적 적대 신경망)"

Generative: 생성적.

데이터 자체를 생성하는 역할

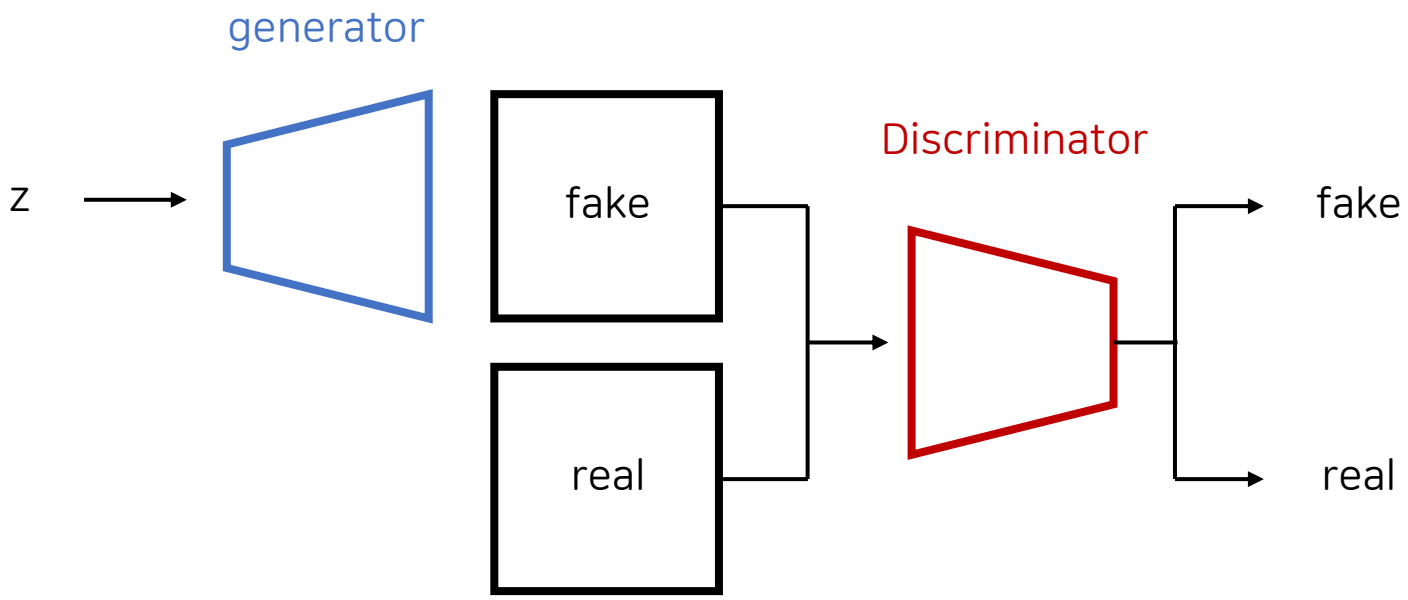
Adversarial: 적대적.

생성 네트워크와 구분 네트워크 간의 상반되는 목적함수가 존재

Network: 네트워크.

신경망의 형태를 가진 네트워크

GAN (Generative Adversarial Network)

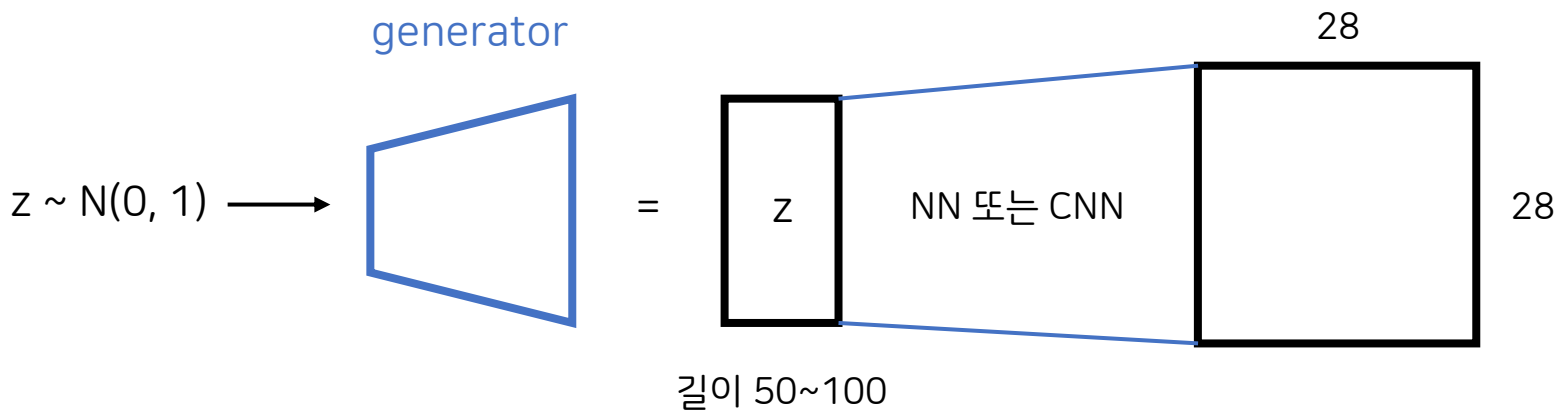


generator가 어떠한 입력 z를 받아서 fake data를 생성

↓

discriminator는 real data와 fake data를 받아 각 data가 real인지 fake인지 판별

GAN (Generative Adversarial Network)



MNIST dataset

28 x 28 x 1

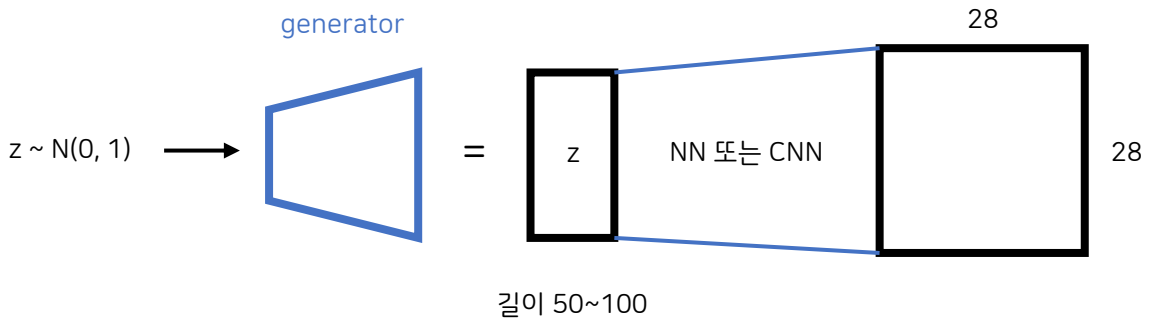
[1, 28, 28]

([채널, 가로, 세로])

generator	
input	output
noise z (잠재변수)	MNIST 데이터와 같은 형태의 데이터

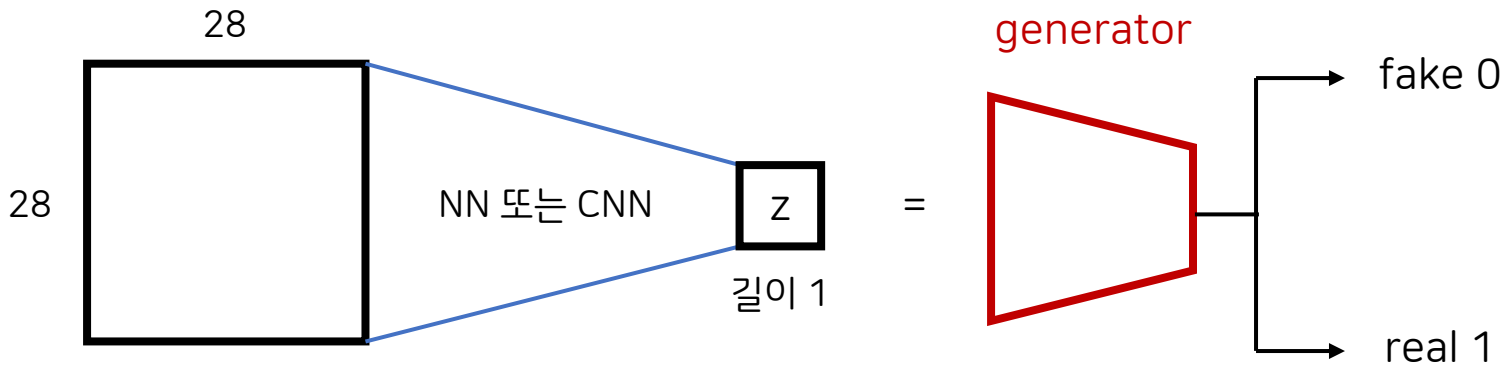
GAN (Generative Adversarial Network)

(2) generator




```
class Generator(nn.Module):
  def __init__(self):
    super(Generator, self).__init__()
    self.layer1 = nn.Seqeuntial(OrderedDict([
                                ('fc1', nn.Linear(z_size, middle_size)), # z_size = 50, middel_size = 200
                                ('bn1', nn.BatchNorm1d(middle_size)),
                                ('act1', nn.ReLU()),
    ]))
    self.layer2 = nn.Sequential(OrderedDict([
                                ('fc2', nn.Linear(middle_size, 784)),# MNIST data size로 바꿔줌
                                #('bn2', nn.BatchNorm2d(784)),
                                ('tanh', nn.Tanh()),
    ]))
  def forward(self, z):
    out = self.layer(z)
    out = self.layer2(out)
    out = out.view(batch_size, 1, 28, 28)
    return out
```

GAN (Generative Adversarial Network)



MNIST dataset

28 x 28 x 1

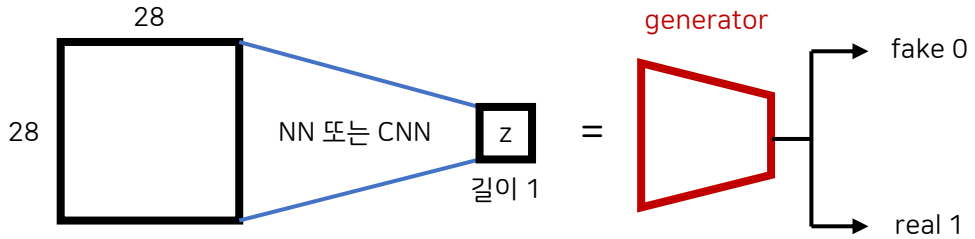
[1, 28, 28]

([채널, 가로, 세로])

discriminator	
input	output
real data, fake data	0 (fake), 1 (real)

GAN (Generative Adversarial Network)

(3) discriminator




```
class Discriminator(nn.Module):
  def __init__(self):
    super(Discriminator, self).__init__()
    self.layer1 = nn.Seqeuntial(OrderdDict([
                                        ('fc1', Linear(784, middle_size)),
                                        #('bn1', nn.BatchNorm1d(middle_size)),
                                        ('act1', nn.LeakyReLU()),
    ]))
    self.layer2 == nn.Sequential(OrderedDict([
                                        ('fc2', nn.Linear(middle_size, 1)),
                                        ('bn2', nn.BatchNorm1d(1)),
                                        ('act2', nn.Sigmoid()), # 0~1 사이의 값
    ]))

  def forward(self, x):
    out = x.view(batch_size, -1)
    out = self.layer1(out)
    out = self.layer2(out)
    return out
```

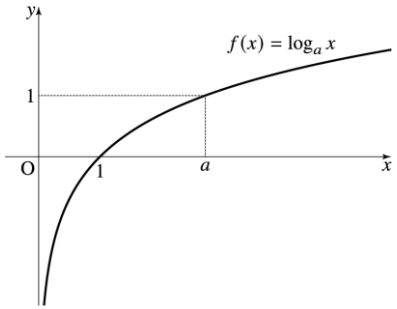
GAN (Generative Adversarial Network)

목적함수

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P(x)}[\log D(x)] + \mathbb{E}_{z\sim P_z(z)}[\log(1 - D(G(Z))]$$

Discriminator: $\max_D V(D,G) = \mathbb{E}_{x\sim P(x)}[\log D(x)] + \mathbb{E}_{z\sim P_z(z)}[\log(1 - D(G(Z))]$

↳ real data는 1, fake data는 0이 되도록 학습 ($D(x)$=1, $D(G(Z))$=0)



Generator: $\min_G V(D,G) = \mathbb{E}_{x\sim P(x)}[\log D(x)] + \mathbb{E}_{z\sim P_z(z)}[\log(1 - D(G(Z))]$

↳ fake data가 1이 되도록 학습 ($D(G(Z))$=1)

discriminator의 목적 함수를 달성한 최적의 상태일 때, generator의 목적 함수를 달성하는 것이 실제 데이터의 분포와 생성된 데이터의 분포가 같아지게 만든다.

GAN (Generative Adversarial Network)

목적함수 ⇒ 손실함수(최소화)

Discriminator: $\max_D V(D,G) = \mathbb{E}_{x\sim P(x)}[\log D(x)] + \mathbb{E}_{z\sim P_z(z)}[\log(1 - D(G(Z))]$

$\min_D V(D,G) = -\mathbb{E}_{x\sim P(x)}[\log D(x)]$ (교차 엔트로피 식과 같은 형태)

Generator: $\min_G V(D,G) = \mathbb{E}_{x\sim P(x)}[\log D(x)] + \mathbb{E}_{z\sim P_z(z)}[\log(1 - D(G(Z))]$

$\max_G V(D,G) = \mathbb{E}_{z\sim P_z(z)}[\log(D(G(Z))]$ (손실함수로 사용시 - 붙임)

➡ torch.nn.BCELoss( )

(교차 엔트로피 손실 함수)

GAN (Generative Adversarial Network)

L2 손실 함수

(LSGAN 등에서 L2 손실 함수를 이용해 안정적인 학습을 진행함)

```
for i in range(epoch):
  for j, (image, label) in enumerate(train_loader):
    image = image.to(device)
    # discriminator
    dis_optim.zero_grad()

    # z 샘플링
    z = init.normal_(torch.Tensor(batch_size, z_size), mean=0, std=0.1).to(device)
    gen_fake = generator.forward(z) # fake image 생성
    dis_fake = discriminator.forward(gen_fake) # fake image 판별

    dis_real = discriminator.forward(image) # real image 판별

    dis_loss = torch.sum(loss_func(dis_fake, zeros_label)) + torch.sum(loss_func(dis_real, ones_label))
    dis_loss.backward(retain_graph = True)
    dis_optim.step()
```

GAN (Generative Adversarial Network)

L2 손실 함수

(LSGAN 등에서 L2 손실 함수를 이용해 안정적인 학습을 진행함)

```
# generator
gen_optim.zero_grad()

z = init.normal_(torch.Tensor(batch_size, z_size), mean=0, std=0.1).to(device)
gen_fake = generator.forward(z) # fake image 생성
dis_fake = discriminator.forward(gen_fake) # fake image 판별

gen_loss = torch.sum(loss_func(dis_fake, ones_label))
gen_loss.backward
gen_optim.step()
```

The END