

# 생성 모델과 시각 지능

Generative Model and Visual Intelligence

06 주차 |

GAN 발전 2

상명대학교 컴퓨터과학과  
민 경 하

# 학습목차

1. WGAN의 기본 원리

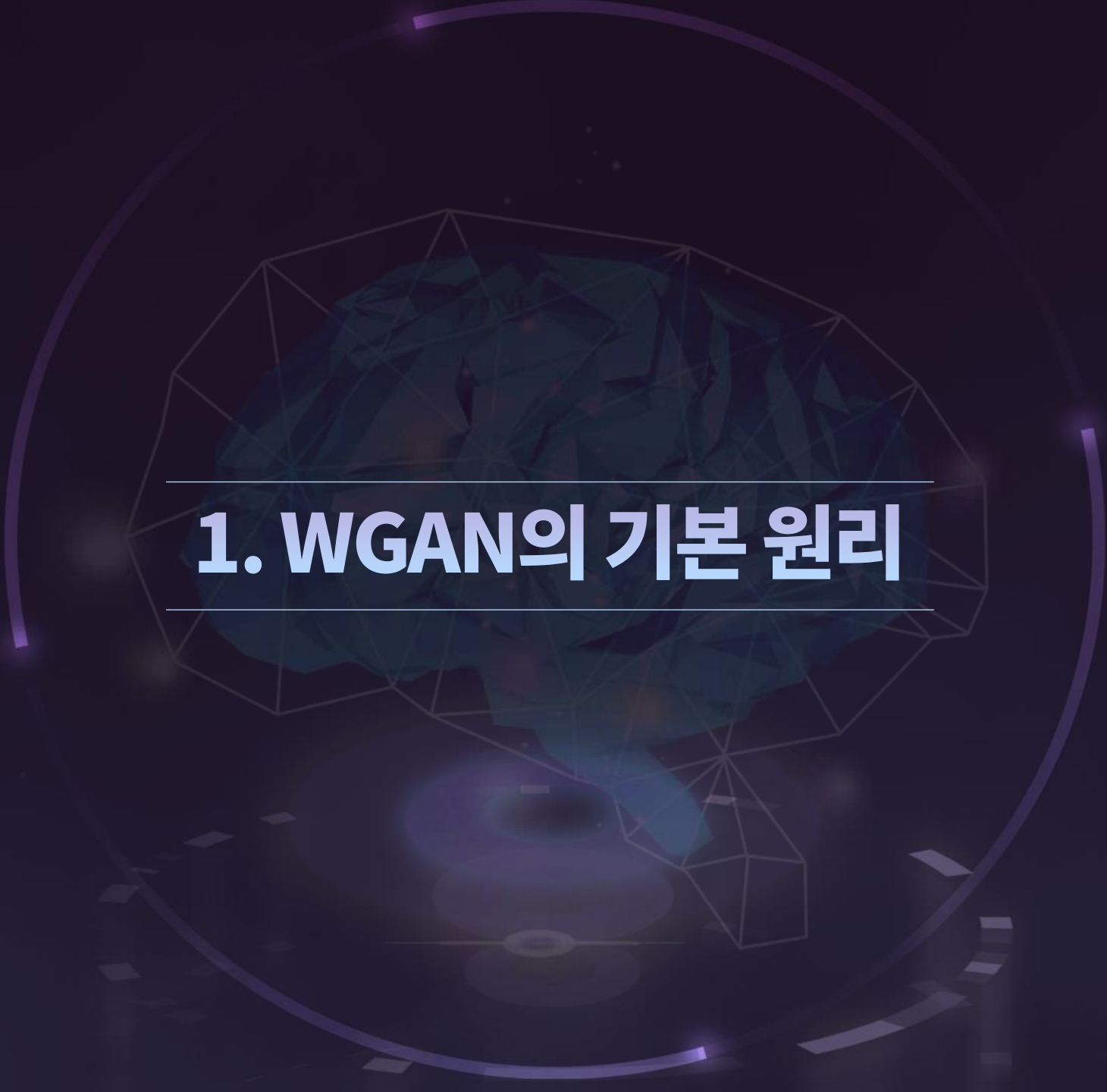
2. WGAN의 설계

3. WGAN의 구성 요소 (1): generator

4. WGAN의 구성 요소 (2): discriminator

5. WGAN의 구성 요소 (3): Gradient penalty

6. WGAN의 구성 요소 (4): loss 함수



# 1. WGAN의 기본 원리

# 1. WGAN의 기본 원리

## Entropy의 개념(기초)

### 정보 이론의 기본 개념

- ✓  $m$ : 정보를 포함하는 message
- ✓  $p(m)$ :  $m$ 의 확률
- ✓  $I(m)$ :  $m$ 에 대한 self-information

$$I(m) = \log \left( \frac{1}{p(m)} \right) = -\log(p(m))$$

#### 예

- ✓  $|m| = 1 \rightarrow$  항상 같은 message를 전달  $\rightarrow p(m) = 1 \rightarrow I(m) = 0$
- ✓  $|m| = 2 \rightarrow$  2 개의 message를 전달  $\rightarrow p(m) = 1/2 \rightarrow I(m) = \log 2 = 1$  (1 bit의 용량이 필요)
- ✓  $|m| = 8 \rightarrow$  8 개의 message를 전달  $\rightarrow p(m) = 1/8 \rightarrow I(m) = \log 8 = 3$  (3 bit의 용량이 필요)

# 1. WGAN의 기본 원리

## Entropy의 개념(기초)

- 메시지  $m$ 의 집합  $M$ 에 대한 정보량의 평균값  $\rightarrow \sum p(m) * I(m)$

$$H(M) = E[I(M)] = \sum_{m \in M} p(m) I(m) = - \sum_{m \in M} p(m) \log p(m)$$

### 예

- ✓  $M = (0, 1)$  with  $p(0) = 0.001$ ,  $p(1) = 0.999$  (최소)  
»  $H(M) = -(0.001 * \log(0.001) + 0.999 * \log(0.999)) \approx 0.011$
- ✓  $M = (0, 1)$  with  $p(0) = 0.5$ ,  $p(1) = 0.5$  (최대)  
»  $H(M) = -(0.5 * \log 0.5 + 0.5 * \log 0.5) = 1$
- ✓  $M = (0, 1)$  with  $p(0) = 0.25$ ,  $p(1) = 0.75$   
»  $H(M) = -(0.25 * \log 0.25 + 0.75 * \log 0.75) = 0.811$

# 1. WGAN의 기본 원리

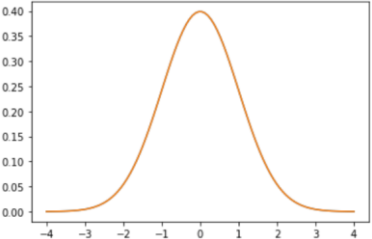
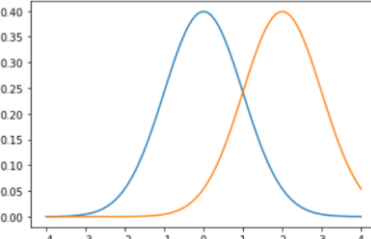
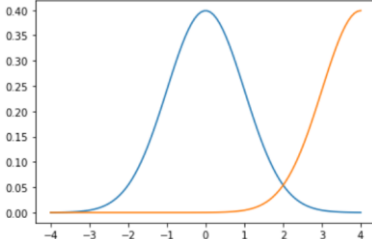
## Entropy의 개념(기초)

### – Cross entropy

- 두 확률 분포  $p$ 와  $q$ 에 대해서  $q$ 를 이용해서  $p$ 를 설명할 때 필요한 정보량

$$H(p, q) = E_p[-\log(q)] = - \sum_x p(x) \log q(x)$$

예

$p, q$	$p = G(0, 1), q = G(0, 1)$	$p = G(0, 1), q = G(2, 1)$	$p = G(0, 1), q = G(4, 1)$
$H(p, q)$	17.55	42.30	116.54
분포			





## 2. WGAN의 설계

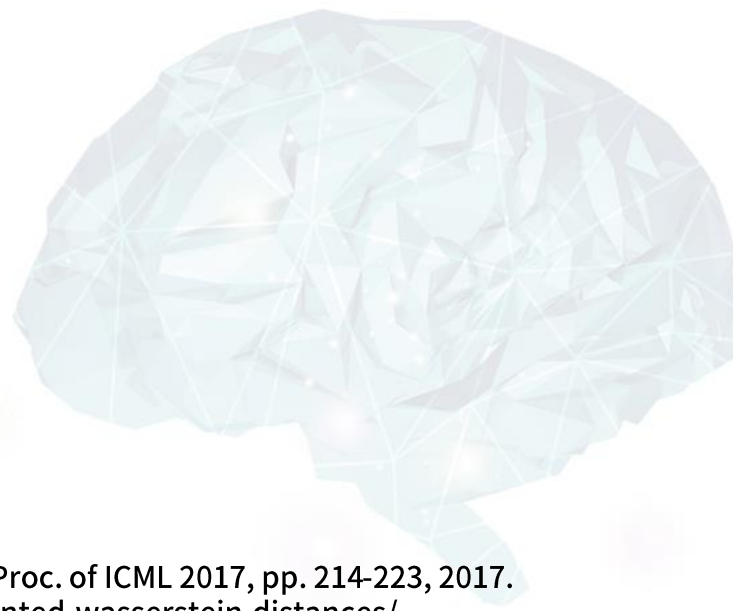
# Wasserstein GAN

- ⊖ Improves the stability of GAN training
  - ☑ Resolve mode collapsing
  - ☑ Avoid gradient vanishing



Leonid Wasserstein  
(1944 ~ )

PennState 수학과 교수

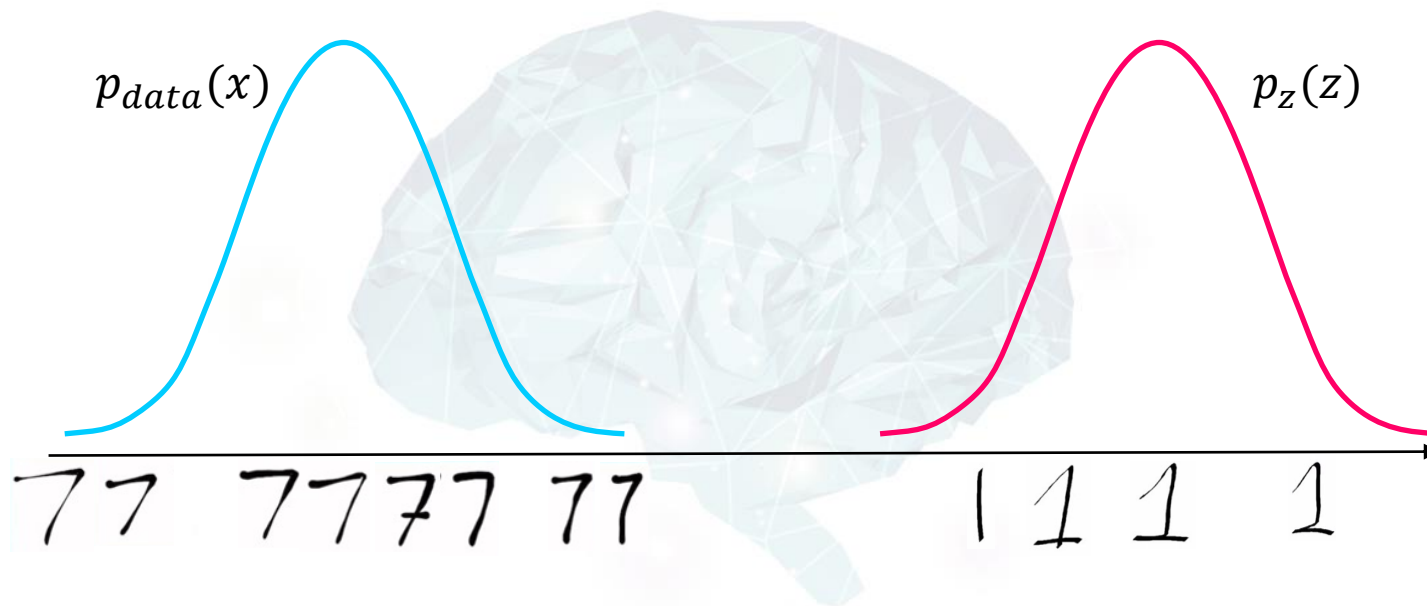




## GAN의 목적(복습)

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

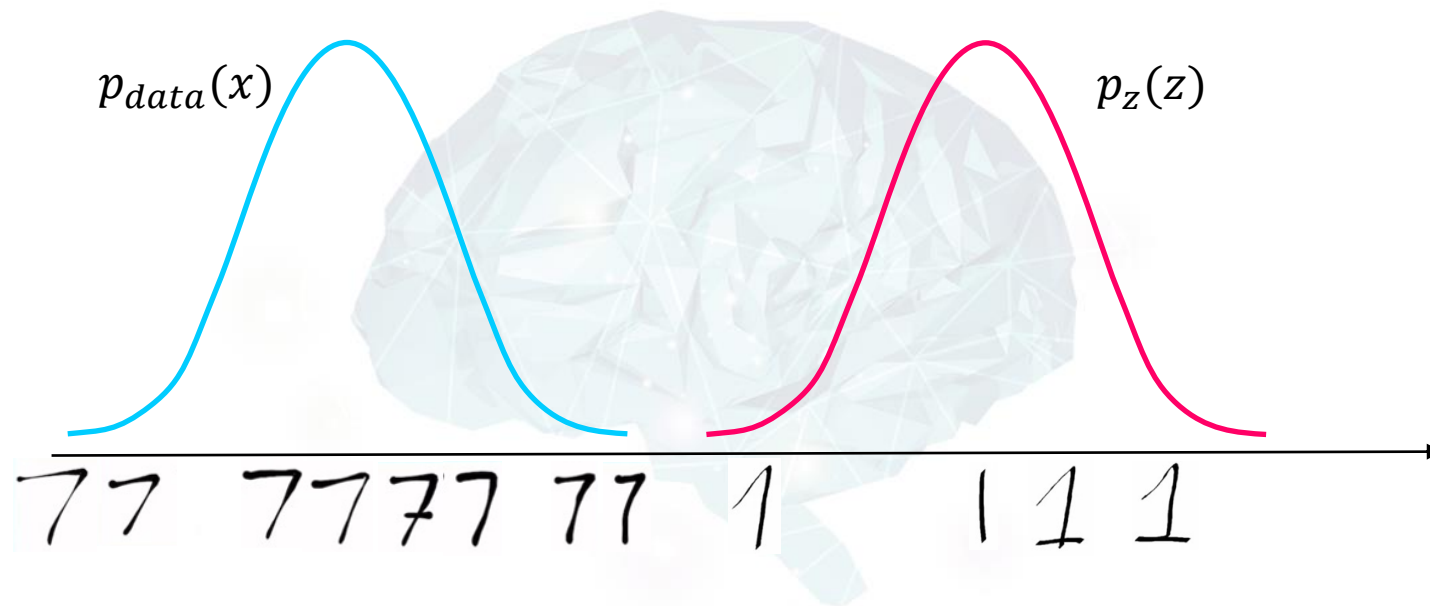
- $p_{data}(x)$ 와 최대한 비슷한  $p_z(z)$ 를 만들자



# GAN의 목적(복습)

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

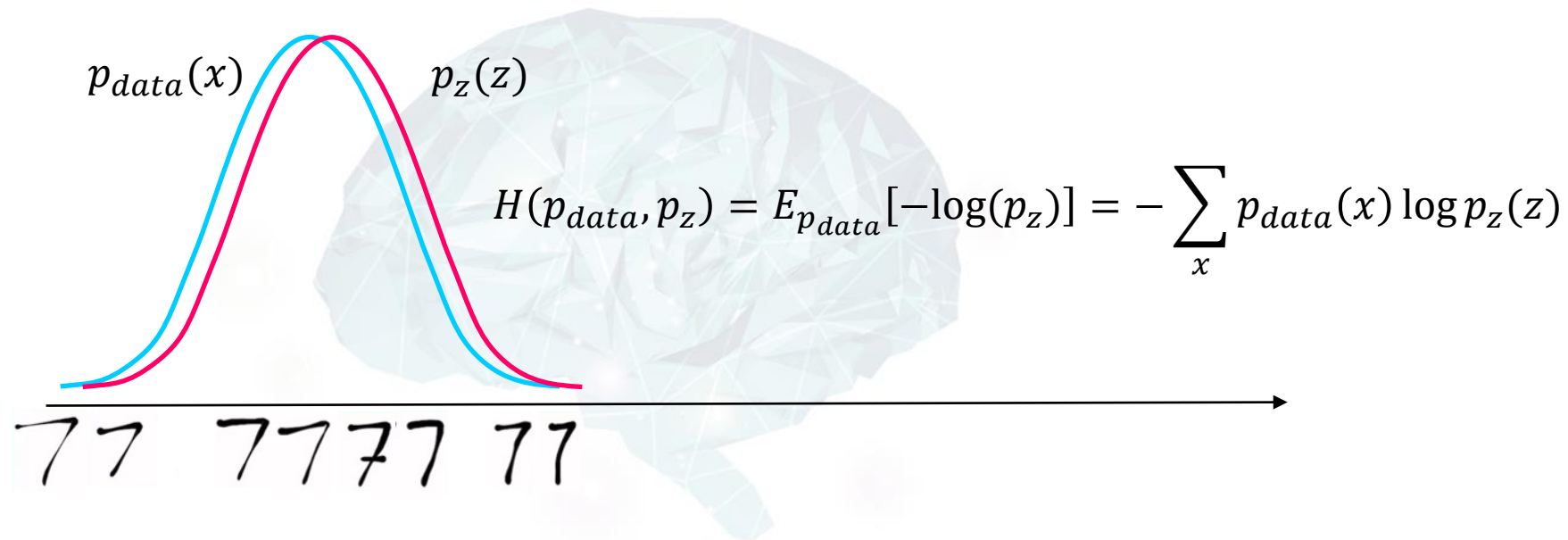
- $p_{data}(x)$ 와 최대한 비슷한  $p_z(z)$ 를 만들자



## GAN의 목적(복습)

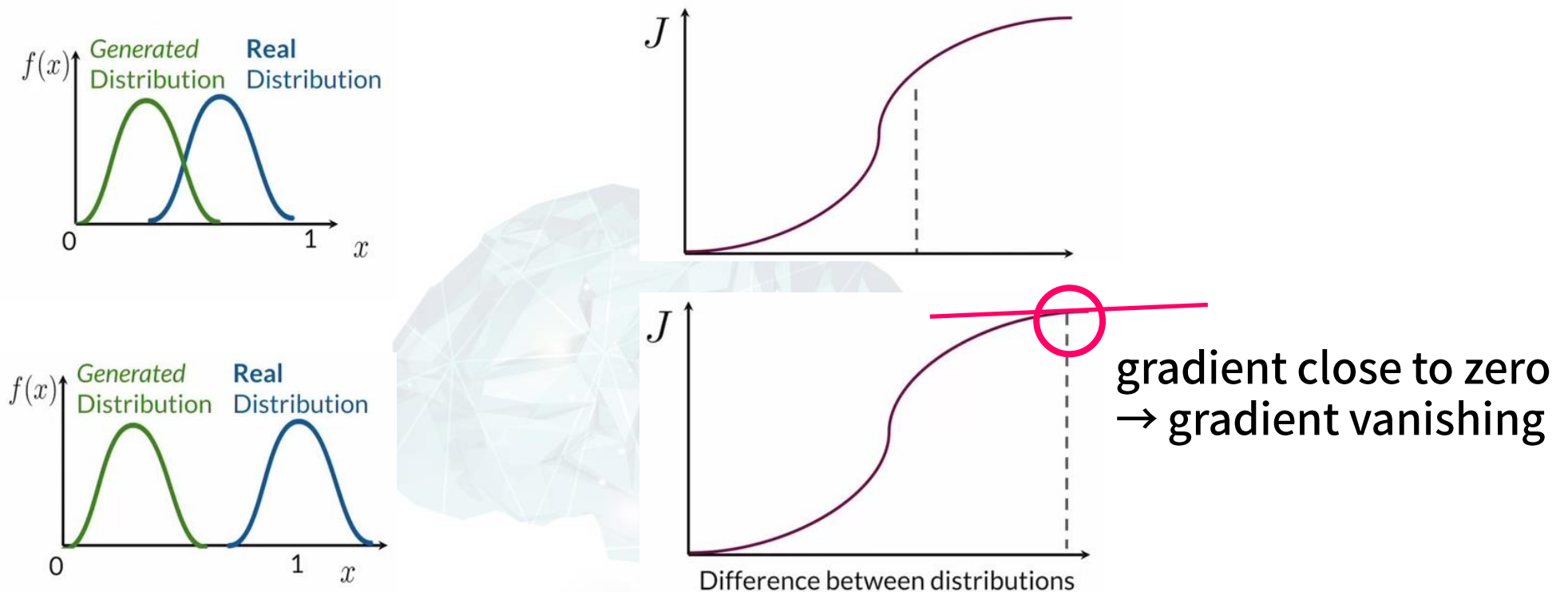
$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim P_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim P_Z(z)} \left[ \log (1 - D(G(z))) \right]$$

- $p_{data}(x)$ 와 최대한 비슷한  $p_z(z)$ 를 만들자  
➔  $p_{data}(x)$ 와  $p_z(z)$ 의 거리를 최소로 만들자



# Wasserstein distance

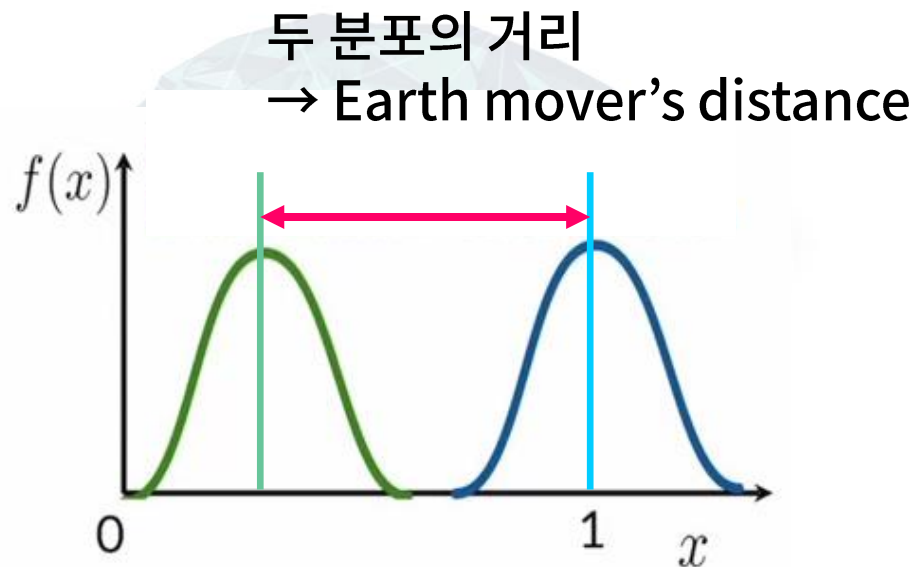
- ➡ Cross entropy를 이용한 Distribution의 거리 측정의 문제점



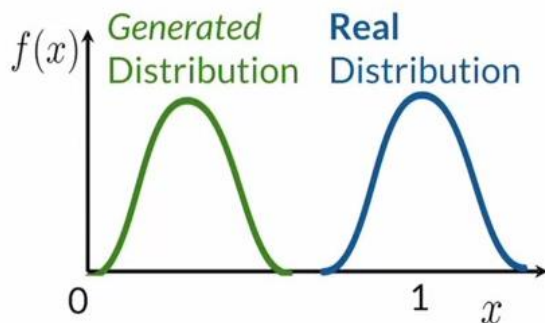
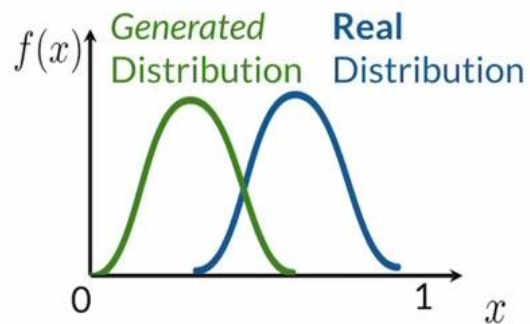
# Wasserstein distance

## ➡ 해결책 → Earth Mover's Distance

- ✓ 두 분포의 차이를 두 분포 사이의 거리로 정의
- ✓ 하나의 분포를 다른 분포로 옮기는 작업을 Earth move로 간주



# Earth Mover's Distance



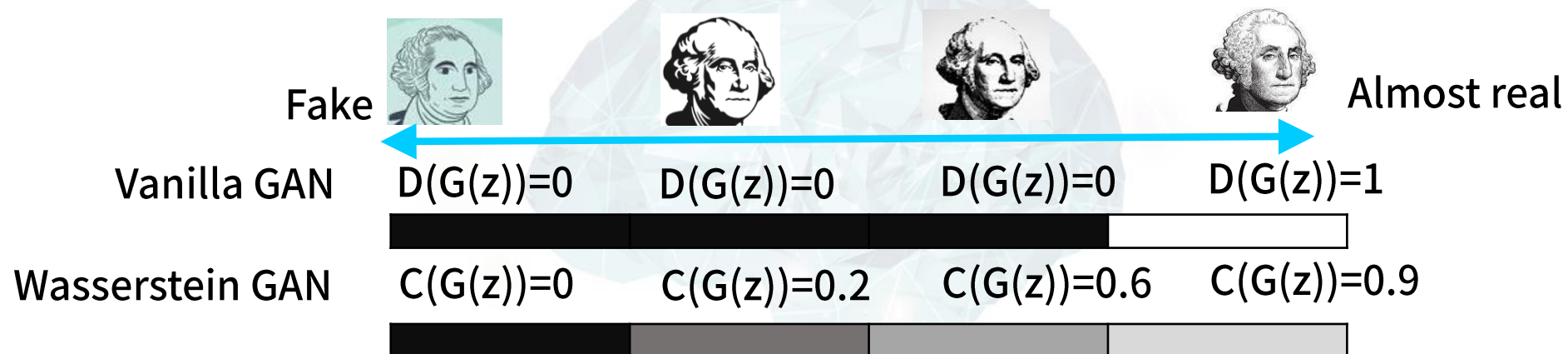
Gradient not close to zero even for very different distributions!

Difference between distributions



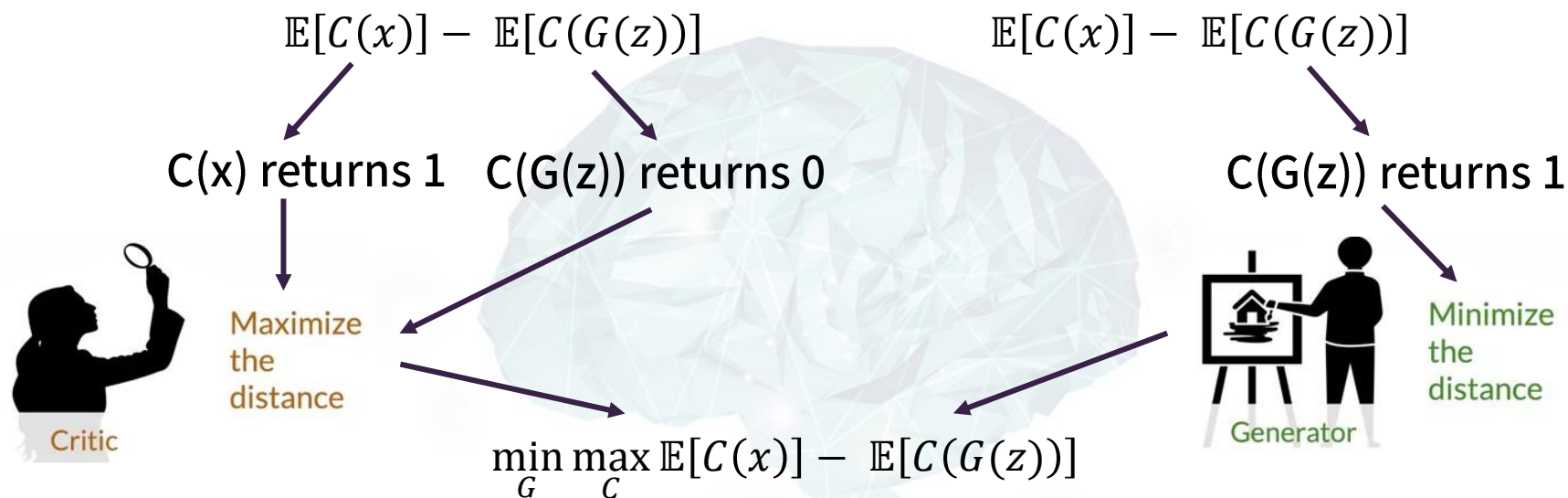
# Earth Mover's Distance

- Earth Mover's Distance를 이용해서 정의
- Discriminator → Critic
  - ✓ Discriminator return 0 (Fake) or 1 (real)
  - ✓ Critic returns the quality of the result in (0, 1)



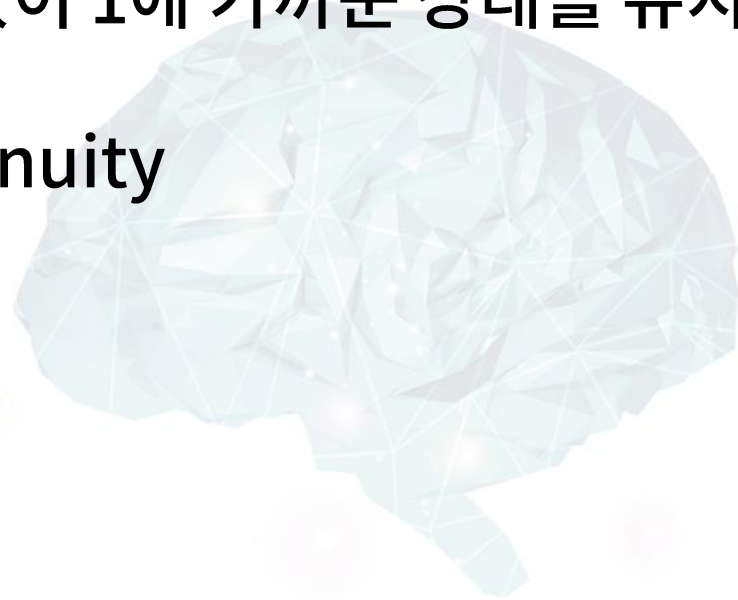
# WassersteinGAN의 loss 함수

- Real data의 distribution과 Fake data의 Distribution의 차이
  - Real data의 distribution:  $\mathbb{E}[C(x)]$
  - Fake data의 distribution:  $\mathbb{E}[C(G(z))]$



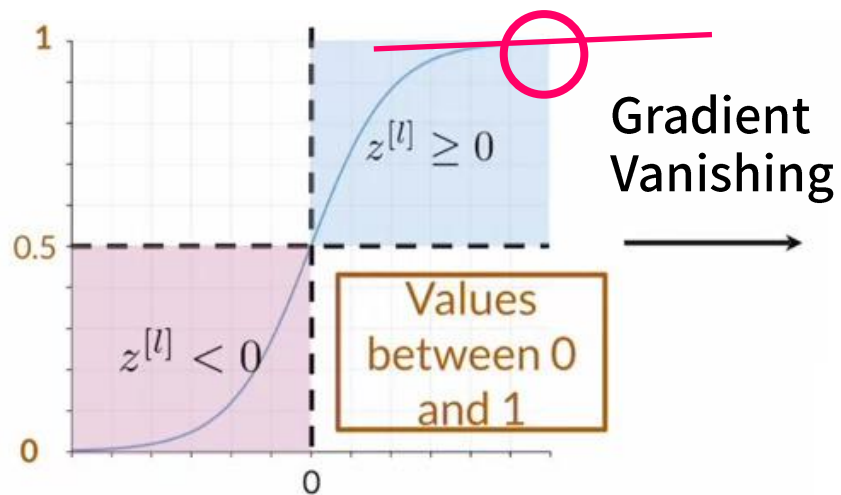
# WassersteinGAN의 loss 함수

- ⊖ Gradient exploding
  - ☑ Gradient가 큰 경우,  
Gradient descent method를 이용해서 구하는 값이 지나치게 커져서 수렴하지 못하는 경우
- ⊖ Gradient의 절대값이 1에 가까운 상태를 유지할 것
- ⊖ 1-Lipschitz continuity

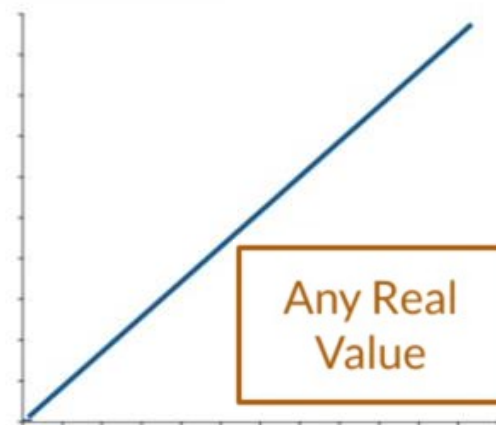


# WassersteinGAN의 loss 함수

- Comparison of discriminator & critic outputs



Discriminator output



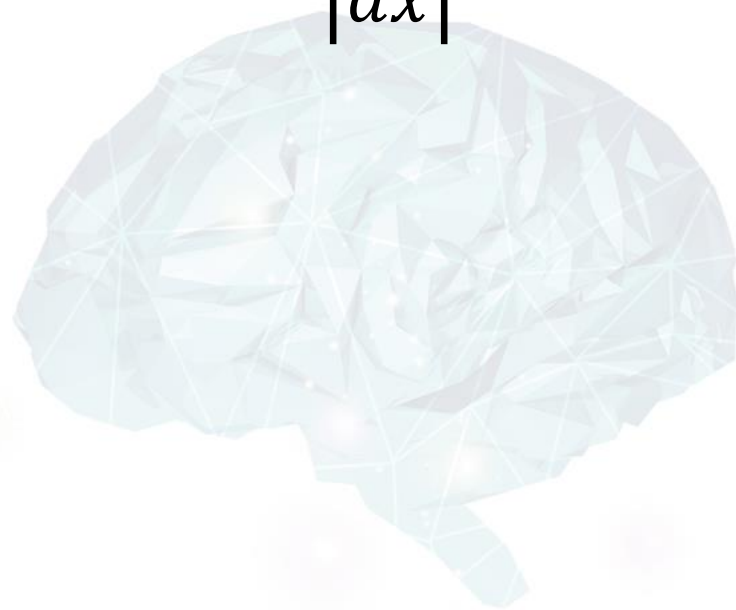
Critic output

Gradient vanishing은 피함  
Gradient exploding은?

# Lipschitz 연속 함수

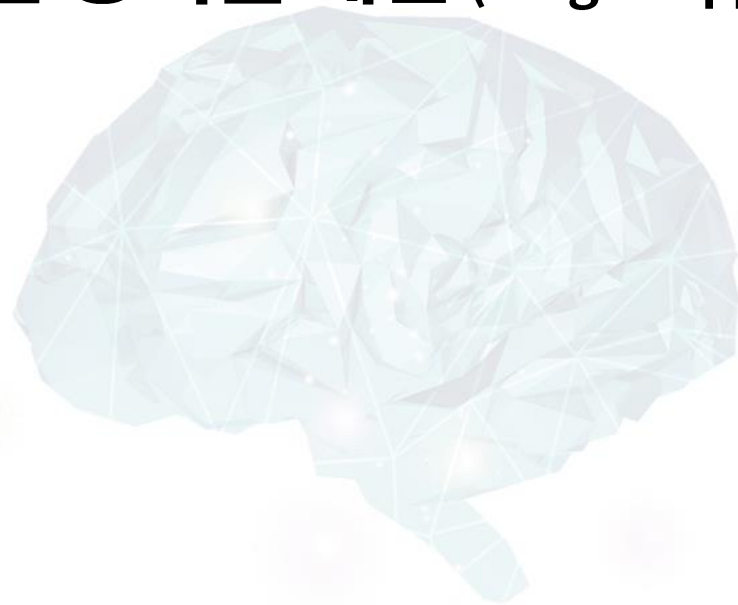
- 두 점 사이의 거리가 일정한 비율 이상으로 증가하지 않는 함수

$$\left| \frac{dy}{dx} \right| \leq K$$



# WGAN의 loss함수의 gradient에 1-L 연속을 유지

- Gradient의 크기를 1이하로 유지
  - ! Critic의 학습 능력을 제한 (Weight clipping)





# Gradient penalty

- Critic의 gradient에 대한 regularization을 이용해서 1-L 연속을 유지

$$\min_G \max_C \mathbb{E}[C(x)] - \mathbb{E}[C(G(z))] + \lambda (\|\nabla C(\hat{x})\|_2 - 1)^2$$

Regularization term  
→ Gradient penalty term

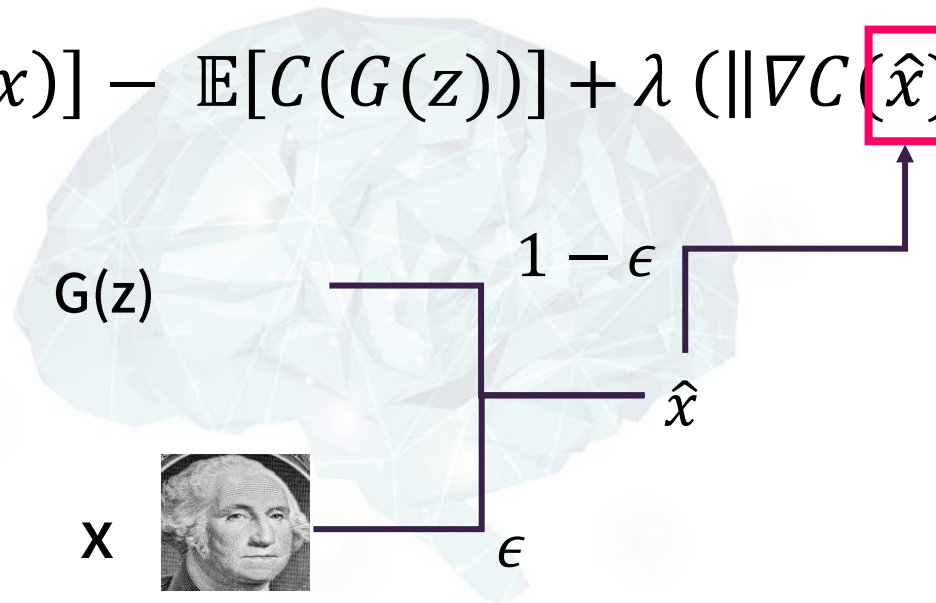


# Gradient penalty에 대한 $\hat{x}$

- $G(z)$ 와  $x$ 를 보간해서  $\hat{x}$  생성  
→  $G(z)$ 의 품질을 높여서 critic의 학습 속도를 조절

$$\hat{x} = \epsilon x + (1 - \epsilon)G(z)$$

$$\min_G \max_C \mathbb{E}[C(x)] - \mathbb{E}[C(G(z))] + \lambda (\|\nabla C(\hat{x})\|_2 - 1)^2$$





## 3. WGAN의 구성 요소 (1): generator

### 3. WGAN의 구성 요소 (1): generator

# Generator block

```
def gen_block(self, input_channels, output_channels, kernel_size=3, stride=2, final_layer=False):  
    if not final_layer:  
        return nn.Sequential (  
            nn.ConvTranspose2d (input_channels, output_channels, kernel_size, stride),  
            nn.BatchNorm2d(output_channels),  
            nn.ReLU(inplace=True),  
        )  
    else:  
        return nn.Sequential (  
            nn.ConvTranspose2d(input_channels, output_channels, kernel_size, stride),  
            nn.Tanh(),  
        )
```



### 3. WGAN의 구성 요소 (1): generator

# Generator

```
class Generator(nn.Module):
    def __init__(self, z_dim=10, im_chan=1, hidden_dim=64):
        super(Generator, self).__init__()
        self.z_dim = z_dim

        self.gen = nn.Sequential (
            self.gen_block(z_dim, hidden_dim*4),
            self.gen_block(hidden_dim*4, hidden_dim*2, kernel_size=4, stride=1),
            self.gen_block(hidden_dim*2, hidden_dim),
            self.gen_block(hidden_dim, im_chan, kernel_size=4, final_layer=True),
        )

    # def gen_block

    def forward(self, noise):
        x = noise.view(len(noise), self.z_dim, 1, 1)
        result = self.gen(x)
        return result
```



## 4. WGAN의 구성 요소 (2): discriminator



## 4. WGAN의 구성 요소 (2): discriminator

# Critic block

```
def crit_block(self, input_channels, output_channels, kernel_size=4, stride=2, final_layer=False):  
    if not final_layer:  
        return nn.Sequential (  
            nn.Conv2d(input_channels, output_channels, kernel_size, stride),  
            nn.BatchNorm2d(output_channels),  
            nn.LeakyReLU(0.2, inplace=True),  
        )  
    else:  
        return nn.Sequential (  
            nn.Conv2d(input_channels, output_channels, kernel_size, stride),  
        )
```



## 4. WGAN의 구성 요소 (2): discriminator

# Critic

```
class Critic(nn.Module):
    def __init__(self, im_chan=1, hidden_dim=64):
        super(Critic, self).__init__()
        self.crit = nn.Sequential(
            self.crit_block(im_chan, hidden_dim),
            self.crit_block(hidden_dim, hidden_dim*2),
            self.crit_block(hidden_dim*2, 1, final_layer=True),
        )

    # def crit_block

    def forward(self, image):
        crit_pred = self.crit(image)
        return crit_pred.view(len(crit_pred), -1)
```

## 5. WGAN의 구성 요소 (3): Gradient penalty

## 5. WGAN의 구성 요소 (3): Gradient penalty

# Gradient 계산

$$\min_G \max_C \mathbb{E}[C(x)] - \mathbb{E}[C(G(z))] + \lambda (\|\nabla C(\hat{x})\|_2 - 1)^2$$

$$\hat{x} = \epsilon x + (1 - \epsilon)G(z)$$

```
def get_gradient(crit, real, fake, epsilon):  
    # Mix the images together  
    mixed_images = real * epsilon + fake * (1 - epsilon)  
  
    # Calculate the critic's scores on the mixed images  
    mixed_scores = crit(mixed_images)  
  
    # Take the gradient of the scores with respect to the images  
    gradient = torch.autograd.grad(  
        inputs=mixed_images,  
        outputs=mixed_scores,  
  
        grad_outputs=torch.ones_like(mixed_scores),  
        create_graph=True,  
        retain_graph=True,  
    )[0]  
    return gradient
```

## 5. WGAN의 구성 요소 (3): Gradient penalty

# Gradient penalty

$$\min_G \max_C \mathbb{E}[C(x)] - \mathbb{E}[C(G(z))] + \lambda (\|\nabla C(\hat{x})\|_2 - 1)^2$$

```
def gradient_penalty(gradient):  
    # Flatten the gradients so that each row captures one image  
    gradient = gradient.view(len(gradient), -1)  
  
    # Calculate the magnitude of every row  
    gradient_norm = gradient.norm(2, dim=1)  
  
    # Penalize the mean squared distance of the gradient norms from 1  
    penalty = torch.mean((gradient_norm - 1)**2)  
    return penalty
```

## 6. WGAN의 구성 요소 (4): loss 함수



## 6. WGAN의 구성 요소 (4): loss 함수

# loss 함수 계산

```
def get_gen_loss(crit_fake_pred):  
    gen_loss = -1. * torch.mean(crit_fake_pred)  
    return gen_loss
```

```
def get_crit_loss(crit_fake_pred, crit_real_pred, gp, c_lambda):  
    crit_loss = torch.mean(crit_fake_pred) - torch.mean(crit_real_pred) + c_lambda * gp  
    return crit_loss
```

$$\min_G \max_C \mathbb{E}[C(x)] - \mathbb{E}[C(G(z))] + \lambda (\|\nabla C(\hat{x})\|_2 - 1)^2$$
