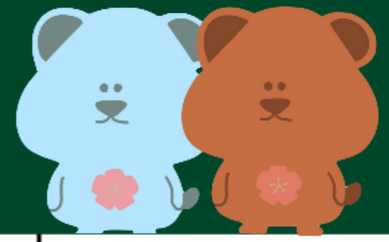
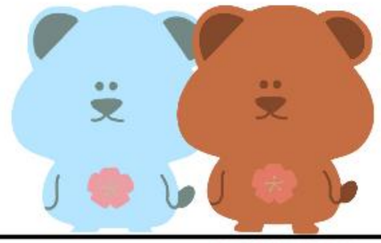


18주차 발표

ML팀 박지운 김예진 이서영

EWHA EURON

목차



EWHA EURON

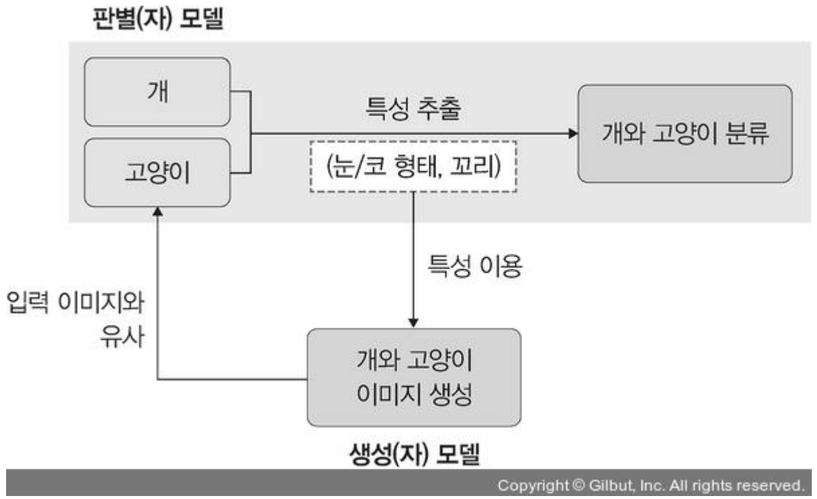
01. 생성 모델이란

EWHA EURON

#1.1 생성 모델 개념

생성 모델이란?

- 주어진 데이터를 학습하여 데이터 분포를 따르는 유사한 데이터를 생성하는 모델

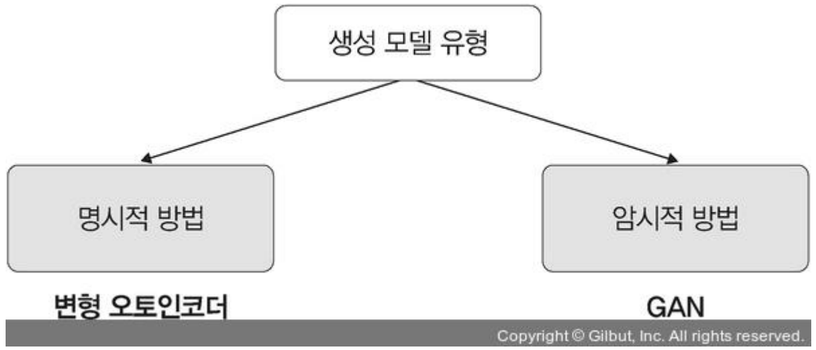


판별 모델 : 개와 고양이 이미지 데이터셋이 주어졌을 때, 그 이미지를 개와 고양이로 분류

생성 모델 : 판별자 모델에서 추출한 특성들의 조합을 이용하여 새로운 개와 고양이 이미지를 생성

-> 생성 모델은 입력 이미지에 대한 데이터 분포 p(x)를 학습하여 새로운 이미지를 생성하는 것이 목표

#1.2 생성 모델 유형



명시적 방법: 확률 변수 p(x)를 정의하여 사용
암시적 방법: 확률 변수 p(x)에 대한 정의 없이 p(x)를 샘플링하여 사용

변형 오토인코더 모델 : 모델의 확률 변수를 구함 -> 이미지의 잠재 공간(latent space)에서 샘플링하여 완전히 새로운 이미지나 기존 이미지를 변형하는 방식으로 학습을 진행

GAN 모델 : 확률 변수를 이용하지 않음 -> 생성자와 판별자가 서로 경쟁하면서 가짜 이미지를 진짜 이미지와 최대한 비슷하게 만들도록 학습을 진행

02. 변형 오토인코더

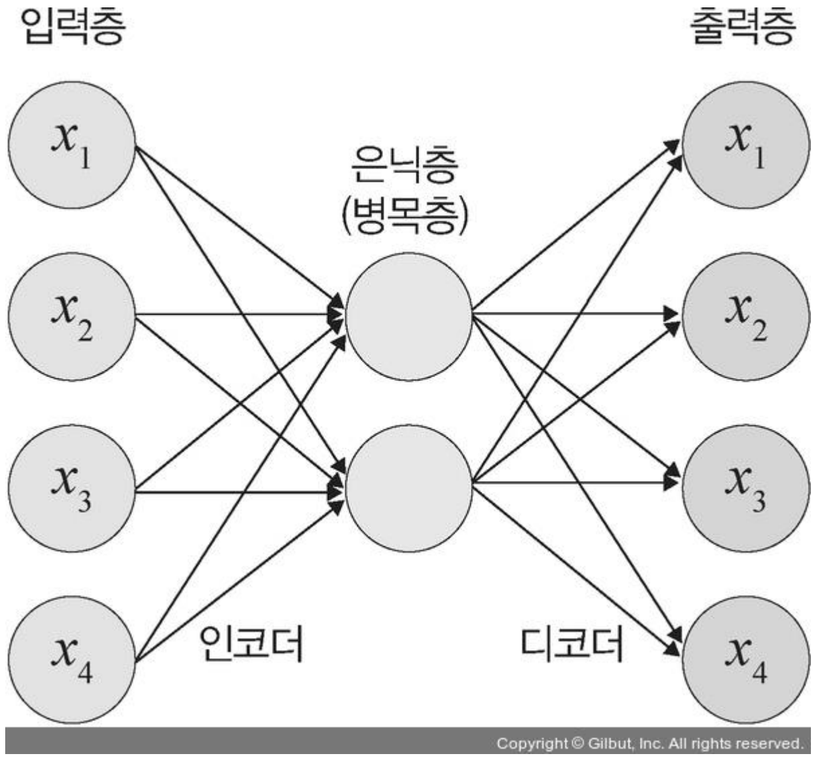
EWHA EURON

#2.1 오토인코더란

오토 인코더란?

오토인코더는 단순히 입력을 출력으로 복사하는 신경망으로, 은닉층의 노드 수가 입력 값보다 적은 것이 특징
-> **입력과 출력이 동일한 이미지**

WHY 입력을 출력으로 복사하는 방법을 사용?
-> 오토인코더의 은닉층은 입력과 출력의 뉴런보다 훨씬 적으므로, 적은 수의 은닉층 뉴런으로 데이터를 가장 잘 표현할 수 있는 방법이 오토인코더~



1. 인코더: 인지 네트워크(recognition network)라고도 하며, 특성에 대한 학습을 수행하는 부분

2. 병목층(은닉층): 모델의 뉴런 개수가 최소인 계층입니다. 이 계층에서는 차원이 가장 낮은 입력 데이터의 압축 표현이 포함

3. 디코더: 생성 네트워크(generative network)라고도 하며, 이 부분은 병목층에서 압축된 데이터를 원래대로 재구성(reconstruction)하는 역할(최대한 입력에 가까운 출력을 생성)

4. 손실 재구성: 압축된 입력을 출력층에서 재구성하며, 손실 함수는 입력과 출력(인코더와 디코더)의 차이를 가지고 계산됨

#2.1 오토인코더란

오토인코더가 중요한 이유 ?

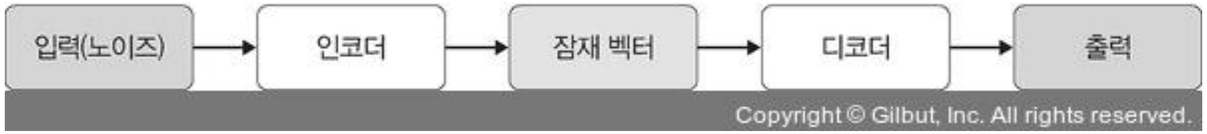
1. **데이터 압축**: 오토인코더를 이용하여 이미지나 음성 파일의 중요 특성만 압축하면 용량도 작고 품질도 더 좋아짐

2. 차원의 저주(curse of dimensionality) 예방: 오토인코더는 특성 개수를 줄여 주기 때문에 데이터 차원이 감소하여 차원의 저주를 피할 수 있음

3. 특성 추출: 오토인코더는 비지도 학습으로 자동으로 중요한 특성을 찾아 줌 ex)예를 들어 눈 모양, 털 색, 꼬리 길이 등 개의 중요한 특성을 자동으로 찾음

#2.1 오토인코더란

오토인코더 예제



케라스에 내장되어 제공하는 MNIST 데이터셋을 사용

```
transform = transforms.Compose([transforms.ToTensor()]) # transforms.ToTensor( )는 이미지를

train_dataset = datasets.MNIST(
    root="../chap13/data", train=True, transform=transform, download=True) # MNIST를 내려받

test_dataset = datasets.MNIST(
    root="../chap13/data", train=False, transform=transform, download=True)

train_loader = DataLoader(
    train_dataset, batch_size=128, shuffle=True, num_workers=4, pin_memory=False)

test_loader = DataLoader(
    test_dataset, batch_size=32, shuffle=False, num_workers=4)
```

```
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True,
                                         ⓐ              ⓑ           ⓒ
                          num_workers=4, pin_memory=False)
                                         ⓓ           ⓔ
```


ⓐ 첫 번째 파라미터: 훈련 데이터셋

ⓑ batch_size: 메모리로 한 번에 불러올 데이터의 크기

ⓒ shuffle: True로 지정하면 데이터를 무작위로 섞겠다는 의미

ⓓ num_workers: 데이터를 불러올 때 몇 개의 프로세스를 사용할지 지정하는 부분으로 병렬로 데이터를 불러오겠다는 의미. (일반적으로 GPU를 사용할 때 많이 사용하는 파라미터)

ⓔ pin_memory: CPU를 사용하다 GPU로 전환할 때 속도 향상을 위해 사용. (cpu나 gpu만 사용한다면 True로 지정할 필요 x)

#2.1 오토인코더란

인코더, 디코더 네트워크 생성

```python
class Encoder(nn.Module): #인코더 네트워크 생성
    def __init__(self, encoded_space_dim,fc2_input_dim):
        super().__init__()

        self.encoder_cnn = nn.Sequential(
            nn.Conv2d(1, 8, 3, stride=2, padding=1),
            nn.ReLU(True),
            nn.Conv2d(8, 16, 3, stride=2, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(True),
            nn.Conv2d(16, 32, 3, stride=2, padding=0),
            nn.ReLU(True)
        ) #이미지 데이터셋 처리를 위해 합성곱 신경망 이용

        self.flatten = nn.Flatten(start_dim=1) #완전연결층
        self.encoder_lin = nn.Sequential(
            nn.Linear(3 * 3 * 32, 128),
            nn.ReLU(True),
            nn.Linear(128, encoded_space_dim)
        ) #출력 계층

    def forward(self, x):
        x = self.encoder_cnn(x)
        x = self.flatten(x)
        x = self.encoder_lin(x)
        return x
```

```python
class Decoder(nn.Module): #디코더 네트워크 생성
    def __init__(self, encoded_space_dim, fc2_input_dim):
        super().__init__()
        self.decoder_lin = nn.Sequential(
            nn.Linear(encoded_space_dim, 128),
            nn.ReLU(True),
            nn.Linear(128, 3 * 3 * 32),
            nn.ReLU(True)
        ) #인코더의 출력을 디코더의 입력으로 사용

        self.unflatten = nn.Unflatten(dim=1, unflattened_size=(32, 3, 3)) #인코더의 완전연결층에 대응
        self.decoder_conv = nn.Sequential(
            nn.ConvTranspose2d(32, 16, 3, stride=2, output_padding=0),
            nn.BatchNorm2d(16),
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 8, 3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(8),
            nn.ReLU(True),
            nn.ConvTranspose2d(8, 1, 3, stride=2, padding=1, output_padding=1)
        ) #인코더의 합성곱층에 대응

    def forward(self, x):
        x = self.decoder_lin(x)
        x = self.unflatten(x)
        x = self.decoder_conv(x)
        x = torch.sigmoid(x)
        return x
```

인코더(데이터셋을 저차원으로 압축하는 것)

디코더(압축된 것을 다시 원래의 차원으로 복원하는 것)

=> 인코더와 디코더에서 사용하는 네트워크 계층은 같아야 함.

#2.1 오토인코더란

인코더, 디코더 객체 초기화 및 손실함수, 옵티마이저 지정

```
encoder = Encoder(encoded_space_dim=4, fc2_input_dim=128)
decoder = Decoder(encoded_space_dim=4, fc2_input_dim=128)
encoder.to(device)
decoder.to(device)

params_to_optimize = [
    {'params': encoder.parameters()},
    {'params': decoder.parameters()}
] # 인코더와 디코더에서 사용할 파라미터를 다르게 지정
optim = torch.optim.Adam(params_to_optimize, lr=0.001, weight_decay=1e-05)
loss_fn = torch.nn.MSELoss()
```

오토인코더에서 가장 널리 사용되는 손실 함수 => '평균 제곱 오차' 와 '이진 크로스 엔트로피'

- 입력 값이 (0,1) 범위에 있으면 이진 크로스 엔트로피를 사용
- 그렇지 않으면 평균 제곱 오차를 사용

옵티마이저는 아담을 사용했지만 알엠에스프롭(RMSProp) 또는 아다델타(adadelta) 같은 옵티마이저를 이용하여 성능을 비교해봐야 함

#2.1 오토인코더란

모델 학습에 대한 함수 생성

```
def train_epoch(encoder, decoder, device, dataloader, loss_fn, optimizer, noise_factor=0.3):
    encoder.train() # 인코더 훈련
    decoder.train() # 디코더 훈련
    train_loss = []
    for image_batch, _ in dataloader: # 훈련 데이터셋을 이용하여 모델 학습(비지도 학습으로 레이블은 필요하
        image_noisy = add_noise(image_batch, noise_factor)
        image_noisy = image_noisy.to(device) # 데이터셋이 CPU/GPU 장치를 사용하도록 지정
        encoded_data = encoder(image_noisy) # 노이즈 데이터를 인코더의 입력으로 사용
        decoded_data = decoder(encoded_data) # 인코더 출력을 디코더의 입력으로 사용
        loss = loss_fn(decoded_data, image_noisy)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss.append(loss.detach().cpu().numpy())
    return np.mean(train_loss)
```

모델을 검증하기 위한 함수 생성

```
def test_epoch(encoder, decoder, device, dataloader, loss_fn, noise_factor=0.3):
    # Set evaluation mode for encoder and decoder
    encoder.eval() # 인코더 테스트
    decoder.eval() # 디코더 테스트
    with torch.no_grad():
        conc_out = [] # 각 배치에 대한 출력을 저장하기 위해 리스트 형식의 변수 정의
        conc_label = []
        for image_batch, _ in dataloader:
            image_batch = image_batch.to(device)
            encoded_data = encoder(image_batch)
            decoded_data = decoder(encoded_data)
            conc_out.append(decoded_data.cpu())
            conc_label.append(image_batch.cpu())
        conc_out = torch.cat(conc_out) # 리스트 형식으로 저장된 모든 값을 하나의 텐서로 생성
        conc_label = torch.cat(conc_label)
        val_loss = loss_fn(conc_out, conc_label) # 손실 함수를 이용하여 오차 계산
    return val_loss.data
```

입력 데이터셋에 추가할 노이즈를 생성하기 위한 함수 정의

```
def add_noise(inputs, noise_factor=0.3):
    noisy = inputs + torch.randn_like(inputs) * noise_factor # 입력과 동일한 크기의 노이즈 텐서 생성
    noisy = torch.clip(noisy, 0., 1.) # 데이터 값의 범위를 조정할 때 사용 (노이즈 값 범위 조정)
    return noisy
```

EWHA EURON

#2.1 오토인코더란

에포크가 진행될수록 노이즈 데이터로 새로운 이미지가 어떻게 만들어지는지 확인하기 위한 함수 생성

- 원래의 이미지, 노이즈가 적용되어 손상된 데이터(이미지), 노이즈 데이터를 이용하여 새롭게 생성된 데이터(이미지)

```
def plot_ae_outputs(encoder, decoder, n=5, noise_factor=0.3):
    plt.figure(figsize=(10,4.5))
    for i in range(n):
        ax = plt.subplot(3, n, i+1) # subplot에서 사용하는 파라미터는 (행, 열, 인덱스)
        img = test_dataset[i][0].unsqueeze(0)
        image_noisy = add_noise(img, noise_factor)
        image_noisy = image_noisy.to(device)

        encoder.eval() # 인코더 평가
        decoder.eval() # 디코더 평가
        with torch.no_grad():
            rec_img = decoder(encoder(image_noisy))

        plt.imshow(img.cpu().squeeze().numpy(), cmap='gist_gray') # 테스트 데이터셋을 출력
        ax.get_xaxis().set_visible(False) #- set_visible(False)는 그래프의 눈금을 표시하지 않겠다는 의미
        ax.get_yaxis().set_visible(False)
        if i == n//2:
            ax.set_title('원래 이미지')
        ax = plt.subplot(3, n, i + 1 + n)
        plt.imshow(image_noisy.cpu().squeeze().numpy(), cmap='gist_gray') # 테스트 데이터셋에 노이즈가 적용된 결과
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        if i == n//2:
            ax.set_title('노이즈가 적용되어 손상된 이미지')

        ax = plt.subplot(3, n, i + 1 + n + n)
        plt.imshow(rec_img.cpu().squeeze().numpy(), cmap='gist_gray') # 노이즈가 추가된 이미지를 인코더와 디코더에
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        if i == n//2:
            ax.set_title('재구성된 이미지')
    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.7,
                        top=0.9,
                        wspace=0.3,
                        hspace=0.3) # subplots_adjust()를 이용하여 subplot들이 겹치지 않도록 최소한의 여백을 만듦
    plt.show()
```

#2.1 오토인코더란

모델 학습

```
import numpy as np

num_epochs = 30
history_da = {'train_loss':[],'val_loss':[]}
loss_fn = torch.nn.MSELoss()

for epoch in range(num_epochs):
    print('EPOCH %d/%d' % (epoch + 1, num_epochs))
    train_loss=train_epoch(
        encoder=encoder,
        decoder=decoder,
        device=device,
        dataloader=train_loader,
        loss_fn=loss_fn,
        optimizer=optim, noise_factor=0.3) # 모델 학습 함수(train_epoch)를 이용하여 모델 학습
    val_loss = test_epoch(
        encoder=encoder,
        decoder=decoder,
        device=device,
        dataloader=test_loader,
        loss_fn=loss_fn, noise_factor=0.3) # 모델 검증(테스트) 함수(test_epoch)를 이용하여 테스트
    history_da['train_loss'].append(train_loss)
    history_da['val_loss'].append(val_loss)
    print('₩n EPOCH {}/{} ₩t train loss {:.3f} ₩t val loss {:.3f}'.format(epoch + 1, num_epochs,train_loss,val_los
    plot_ae_outputs(encoder, decoder, noise_factor=0.3)
```

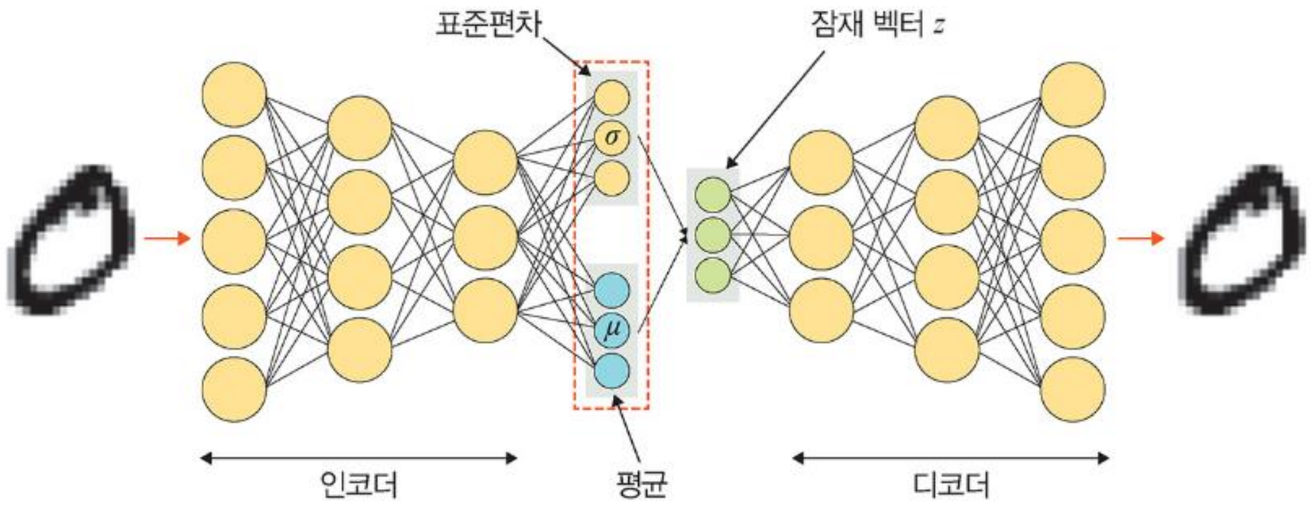
에포크가 진행될수록 훈련과 검증(테스트) 데이터셋에 대한 오차가 줄어들고 있음.
또한, 노이즈를 이용하여 생성된 이미지 역시 선명해지는 것을 확인

EWHA EURON

#2.2 변형 오토인코더란

변형 오토인코더란?

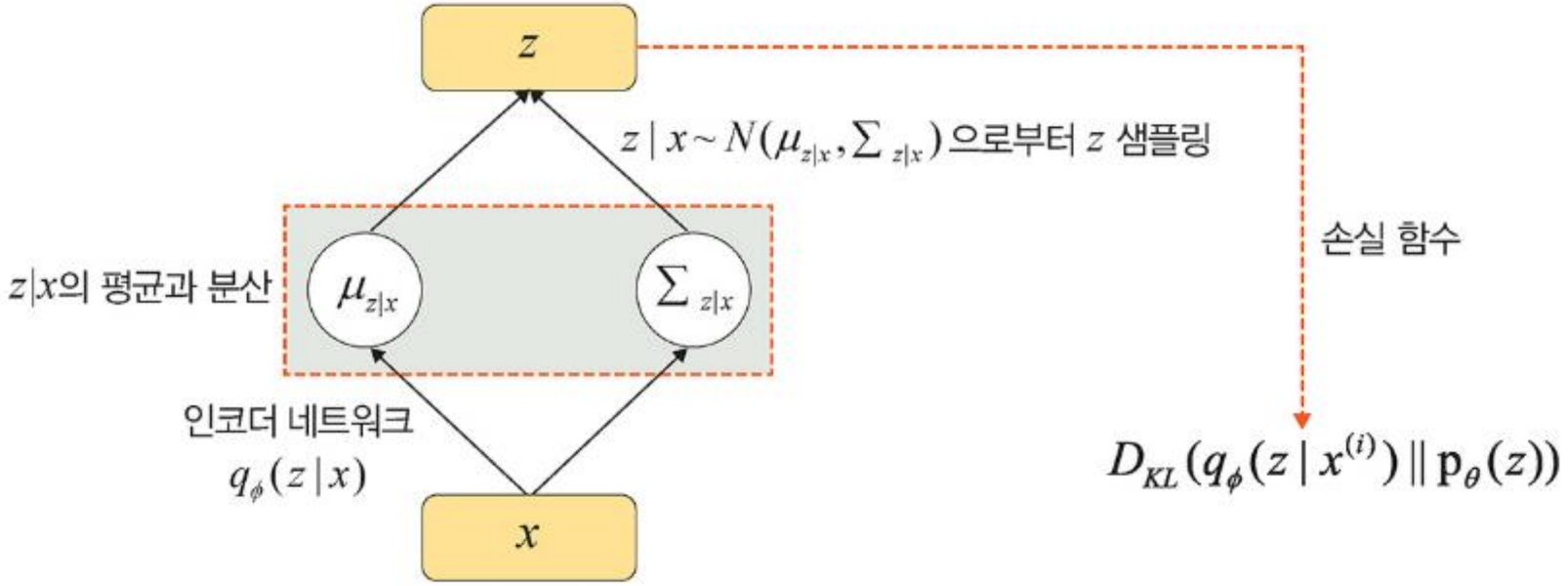
- 오토인코더 : 단순히 입력을 출력으로 복사
- 변형 오토인코더 : 입력 데이터와 조금 다른 출력 데이터
- 표준 편차와 평균으로 확률 분포 만듦 → z라는 가우시안 분포 (잠재 벡터 z)
- z분포에서 벡터를 랜덤하게 샘플링 → 이 분포의 오차를 이용하여 유사한 다양한 데이터 생성



#2.2 변형 오토인코더란

인코더 네트워크

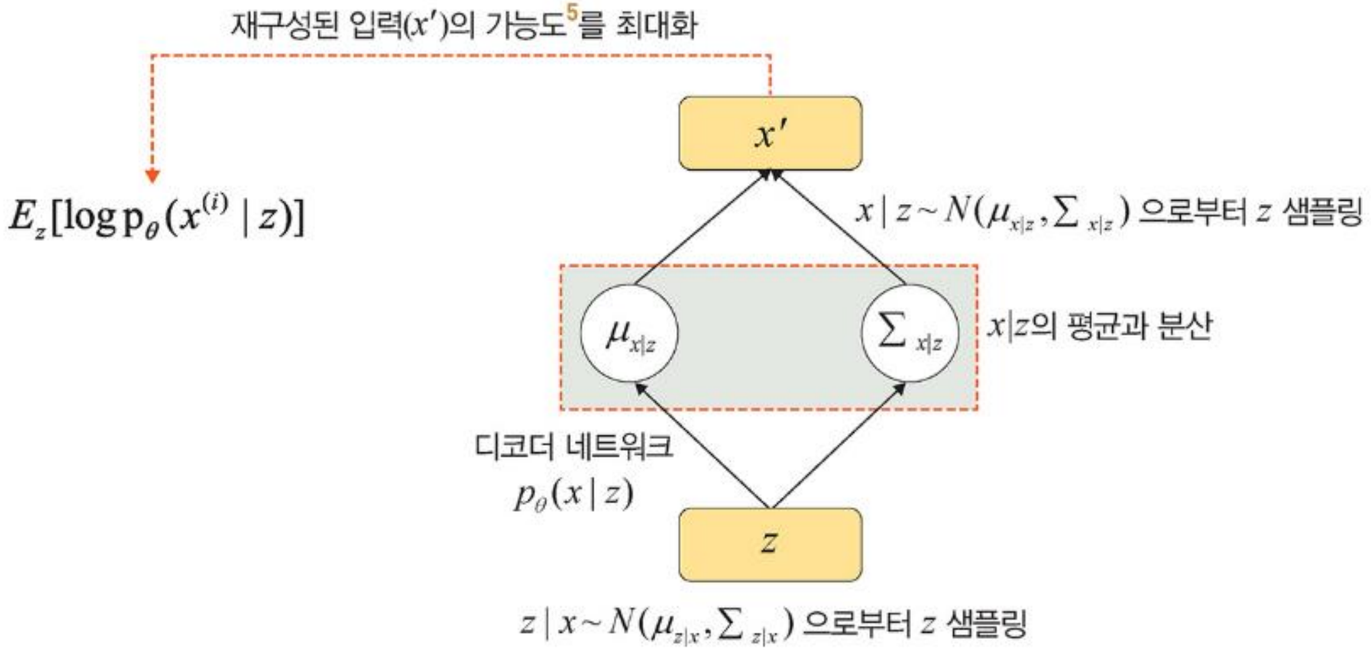
- 인코더 네트워크 : x를 입력 받아 잠재 벡터 z와 대응되는 평균, 분산 구함
- 입력 x → 인코더 네트워크 → 출력 → ②항 계산
- 가우시안 분포에서 z 샘플링



#2.2 변형 오토인코더란

디코더 네트워크

- 디코더 네트워크 : z를 입력 받아 x와 대응되는 평균, 분산 구함
- 샘플링한 z 입력 → 디코더 네트워크 → 출력 → ①항 계산
- 가우시안 분포에서 z 샘플링 → x’ 구함
- 역전파를 이용하여 가능도가 증가하는 방향으로 파라미터 업데이트



#2.2 변형 오토인코더란

$$L\left(x^{(i)}, \theta, \phi\right) = E_z[\log p_\theta(x^{(i)}|z)] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))$$

- z가 주어졌을 때 x' 를 표현하기 위한 확률밀도 함수
- 디코더 네트워크
- 클수록 모델 가능도 커짐

KLD (쿨백-라이블러 발산)

- X에서 z를 표현하는 확률밀도 함수
- 인코더 네트워크와 가우시안 분포의 유사도
- 작을수록 유사도와 모델 가능도 커짐

EWHA EURON

#2.3 변형 오토인코더 실습 예제

인코더 네트워크 생성

```
class Encoder(nn.Module):
    def __init__(self, input_dim, hidden_dim, latent_dim):
        super(Encoder, self).__init__()
        self.input1 = nn.Linear(input_dim, hidden_dim)
        self.input2 = nn.Linear(hidden_dim, hidden_dim)
        self.mean = nn.Linear(hidden_dim, latent_dim)
        self.var = nn.Linear (hidden_dim, latent_dim)

        self.LeakyReLU = nn.LeakyReLU(0.2)
        self.training = True

    def forward(self, x):
        h_ = self.LeakyReLU(self.input1(x))
        h_ = self.LeakyReLU(self.input2(h_))
        mean = self.mean(h_)
        log_var = self.var(h_)
        return mean, log_var
```

평균, 분산 반환

EWHA EURON

#2.3 변형 오토인코더 실습 예제

디코더 네트워크 생성

```
class Decoder(nn.Module):
    def __init__(self, latent_dim, hidden_dim, output_dim):
        super(Decoder, self).__init__()
        self.hidden1 = nn.Linear(latent_dim, hidden_dim)
        self.hidden2 = nn.Linear(hidden_dim, hidden_dim)
        self.output = nn.Linear(hidden_dim, output_dim)
        self.LeakyReLU = nn.LeakyReLU(0.2)

    def forward(self, x):
        h = self.LeakyReLU(self.hidden1(x))
        h = self.LeakyReLU(self.hidden2(h))
        x_hat = torch.sigmoid(self.output(h))
        return x_hat
```

추출한 샘플을 다시 원본으로 반환

시그모이드 통과

#2.3 변형 오토인코더 실습 예제

변형 오토인코더 네트워크 생성

```
class Model(nn.Module):
    def __init__(self, Encoder, Decoder):
        super(Model, self).__init__()
        self.Encoder = Encoder
        self.Decoder = Decoder

    def reparameterization(self, mean, var):
        epsilon = torch.randn_like(var).to(device)
        z = mean + var*epsilon
        return z

    def forward(self, x):
     ① mean, log_var = self.Encoder(x)
     ② z = self.reparameterization(mean, torch.exp(0.5 * log_var))
     ③ x_hat = self.Decoder(z)
        return x_hat, mean, log_var
```

reparameterization 함수 :

평균, 표준편차를 이용해 가우시안 분포 생성

→ z 벡터 샘플링

① 인코더가 평균, 표준편차 반환

② reparameterization 함수가 z 샘플링

③ 디코더가 다시 원본으로 변환

#2.3 변형 오토인코더 실습 예제

손실함수 정의

```
def loss_function(x, x_hat, mean, log_var):
  ① reproduction_loss = nn.functional.binary_cross_entropy(x_hat, x, reduction='sum')
  ② KLD = - 0.5 * torch.sum(1+ log_var - mean.pow(2) - log_var.exp())
    return reproduction_loss, KLD

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

① 재구성 오차 : 복원한 결과물(x_hat)을 원본 데이터(x)와 비교하여 얼마나 비슷한지 (크로스엔트로피 사용)

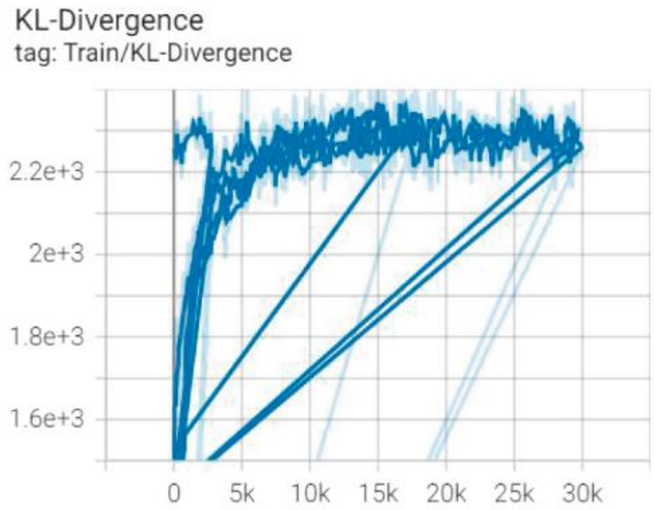
② KLD (쿨백-라이블러 발산) : 인코더 네트워크와 가우시안 분포의 유사도

loss = BCE+ KLD 즉, reproduction loss + KLD

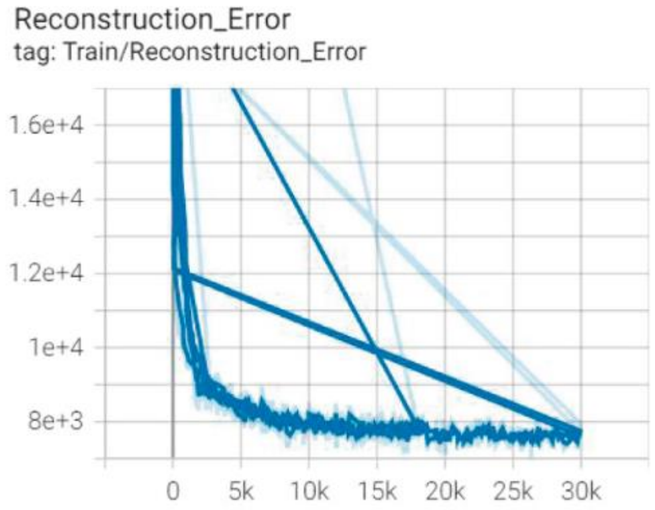
$$L = -E_{z\sim q(z|x)}[\log p_\theta(x|z)] + D_{KL}(q(z|x)\|p_\theta(z))$$

#2.3 변형 오토인코더 실습 예제

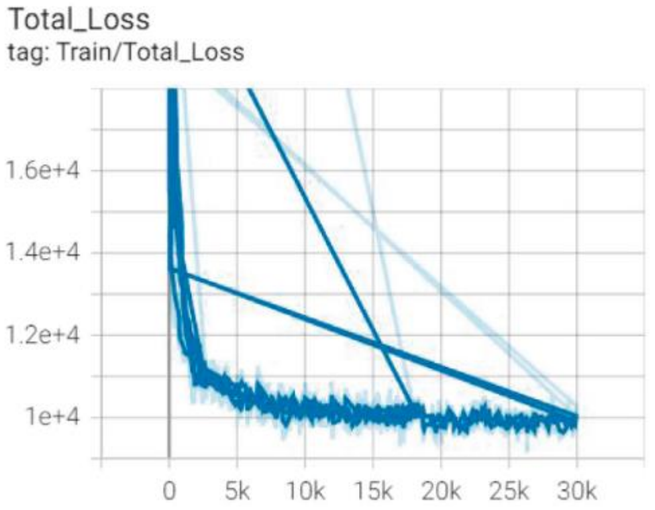
Training set



- 증가하다가 일정 범위에서 수렴
- 인코더 네트워크와 가우시안 분포가 가까워지고 있음



- 기존 이미지를 이용한 새로운 이미지 생성이 잘 진행되고 있음



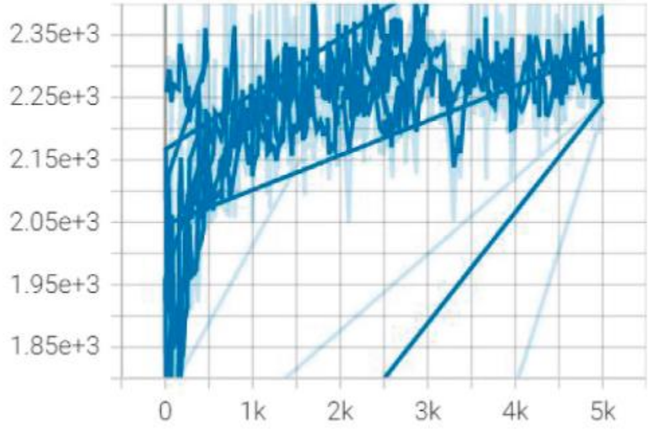
- 전체 오차 작아지고 있음!

EWHA EURON

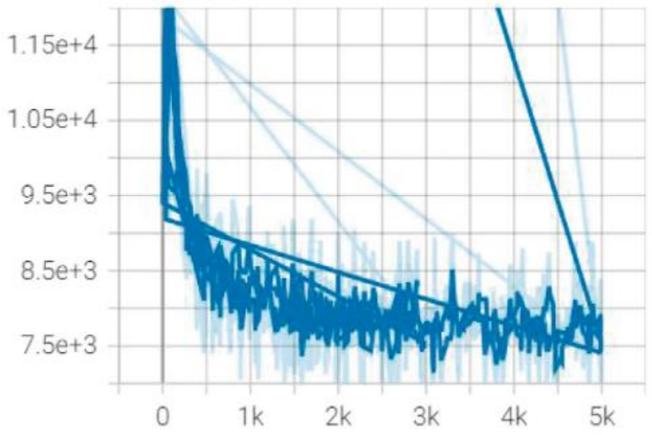
#2.3 변형 오토인코더 실습 예제

Test set

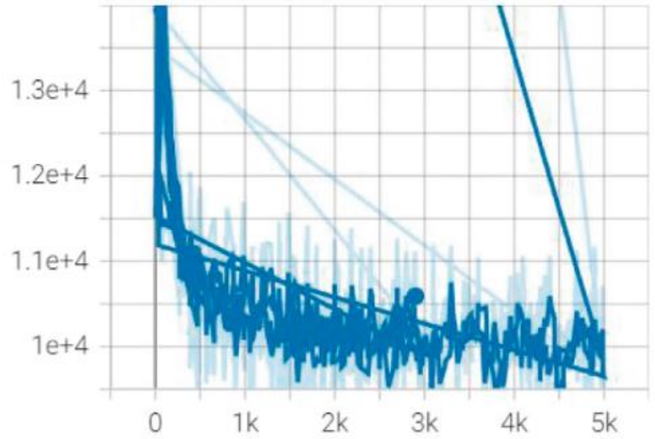
KL-Divergence
tag: Test/KL-Divergence



Reconstruction_Error
tag: Test/Reconstruction_Error



Total_Loss
tag: Test/Total_Loss



03. 적대적 생성 신경망

EWHA EURON

3.1 적대적 생성 신경망

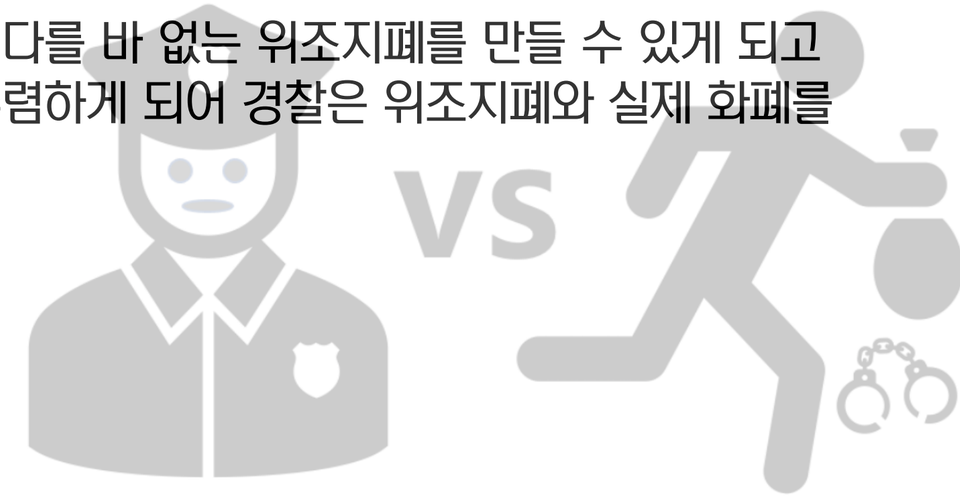
: Generative Adversarial Networks(GAN)

구글의 “Ian Goodfellow”에 의해 2014년 신경정보처리시스템학회(NIPS)에서 제안

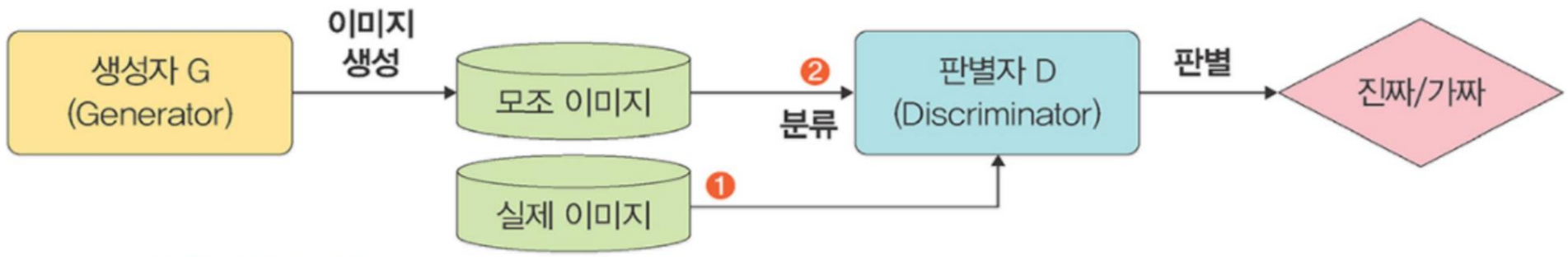
- 두 개의 네트워크를 적대적으로 학습시키며 실제 데이터와 비슷한 가짜 데이터를 생성해내는 모델
- Label 없는 비지도 학습

- Ian은 경찰과 위조지폐범 사이 게임에 GAN을 비유
  - 위조지폐범은 최대한 진짜 같은 화폐를 만들어 경찰을 속이려고 하고
  - 경찰은 진짜와 가짜 화폐를 완벽히 판별하여 위조지폐범을 검거하려 함

⇒ 경쟁적인 학습이 지속되다 보면 어느 순간 위조지폐범은 진짜와 다를 바 없는 위조지폐를 만들 수 있게 되고 경찰이 위조지폐를 구별할 수 있는 확률도 가장 헷갈리는 50%로 수렴하게 되어 경찰은 위조지폐와 실제 화폐를 구분할 수 없는 상태에 이르게 됨



3.2 판별자와 생성자



▲ 그림 13-21 적대적 생성 신경망 학습 과정

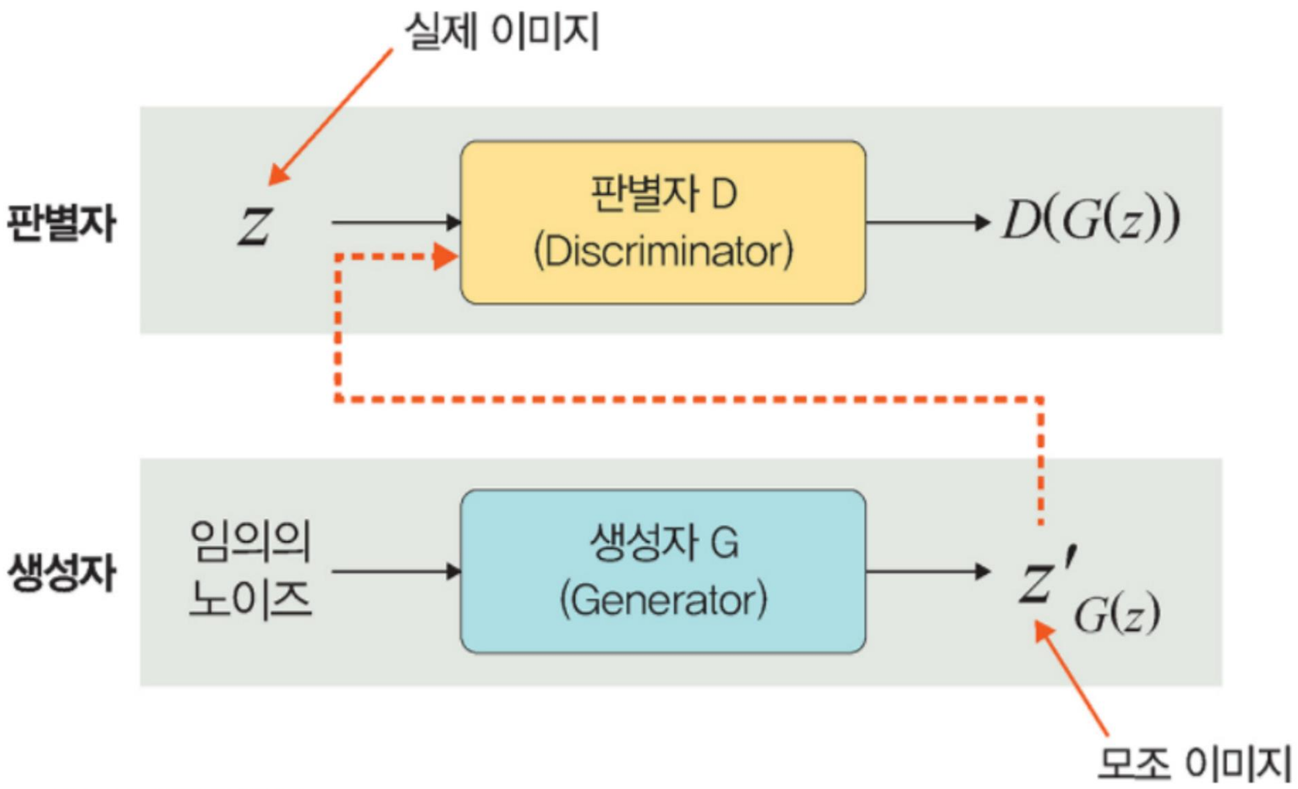
판별자를 먼저 학습시킨 후 생성자를 학습시키는 과정 반복

1) 판별자 D
   - 목표 : 진짜 분포인지 가짜 분포인지 맞히기
   - 과정 : 실제 이미지 입력을 통해 분류 학습 → 생성자가 만든 모조 이미지를 입력하여 가짜로 분류하도록 학습 → 반복하면 판별자는 실제와 모조를 구분할 수 없게 됨

2) 생성자 G
   - 목표 : 진짜 분포에 가까운 가짜 분포 생성하기
   - 과정 : 판별자가 구분할 수 없게끔 진짜에 가까운 모조 이미지를 만듦

☑ 생성자는 분류에 성공할 확률을 낮추고 판별자는 분류에 성공할 확률을 높이면서 서로 경쟁적으로 발전시키는 구조
☑ 판별자가 가짜와 진짜를 판단하지 못하는 경계가 최적 지점

3.2 판별자와 생성자



▲ 그림 13-23 생성자와 판별자

- 판별자는 D(x)가 진짜 이미지일 확률 반환
- 생성자는 노이즈 데이터를 사용하여 모조 이미지 z' (G(z)) 생성

☑ D 학습 시 G 고정시킨 후 실제 이미지는 높은 확률, 모조 이미지는 낮은 확률로 반환하게끔 가중치 업데이트

3.3 minmax probelm

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[log\ D(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z))]$$

① ② ③ ④

① 실제 데이터에 대한 확률 분포에서 샘플링한 데이터
② 가우시안 분포를 사용하는 임의의 노이즈에서 샘플링한 데이터
③ 판별자 D(x)가 1에 가까우면 진짜 데이터로 0에 가까우면 가짜 데이터로 판단, 0이면 가짜를 의미
④ 생성자 G가 생성한 이미지인 G(z)가 1에 가까우면 진짜 데이터로, 0에 가까우면 가짜 데이터로 구분

$$\max_D \log(D(x)) + \log(1 - D(G(z)))$$

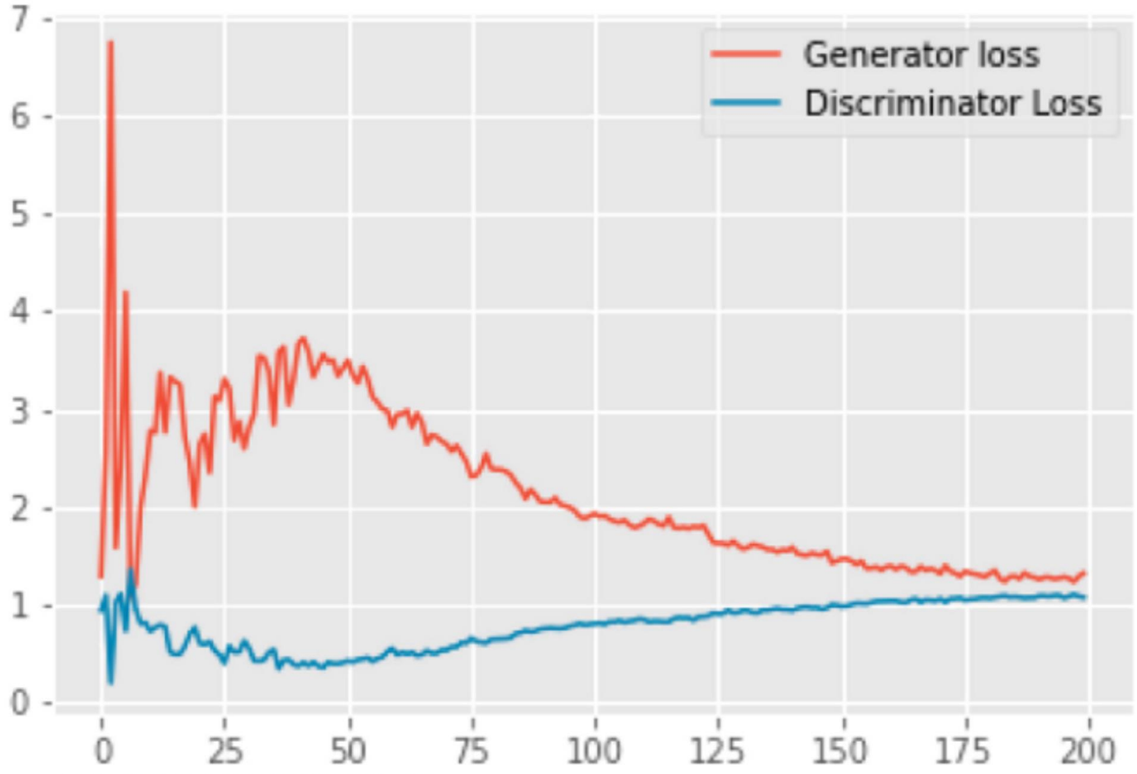
$$\min_G \log(1 - D(G(z)))$$

☑ D(x)=1, 1-D(G(z))=1이어야 최적이므로 판별자는 최대값으로 업데이트
☑ D(G(z))=1이어야 최적이므로 판별자는 최소값으로 업데이트
☑ D/G 파라미터 번갈아 업데이트하며, 그 때 나머지 네트워크는 고정

** 원 논문에서는 최소최대문제가 global optimum인 unique solution을 갖는지 & 알고리즘이 전역 최고점으로 수렴하는지에 대한 이론적 증명도 포함되지만, 함수 공간을 현실적인 뉴럴 네트워크가 표현할 수 있는 공간으로 좁히면 G가 모수 공간에서 다중임계점을 가진다고 함
⇒ 이론적 보장은 부족하나 성능 면에서 실용적이므로 합리적 모델로서 사용한다는 뜻
https://brunch.co.kr/@kakao-it/145

EWHA EURON

3.4 구현결과



▲ 그림 13-31 생성자와 판별자에 대한 오차

? 초반 에포크 동안 생성자의 오차는 증가하고 판별자의 오차는 감소
⇒ 학습 초기 단계에 생성자는 좋은 가짜 이미지를 생성하지 못하기에 판별자가 실제 이미지와 가짜 이미지를 쉽게 구분
⇒ 하지만 학습이 진행됨에 따라 생성자는 진짜와 같은 가짜 이미지를 만들며 판별자는 가짜 이미지 중 일부를 진짜로 분류하므로 생성자의 오차가 감소하면 판별자의 오차 증가

04. GAN 파생 기술

EWHA EURON

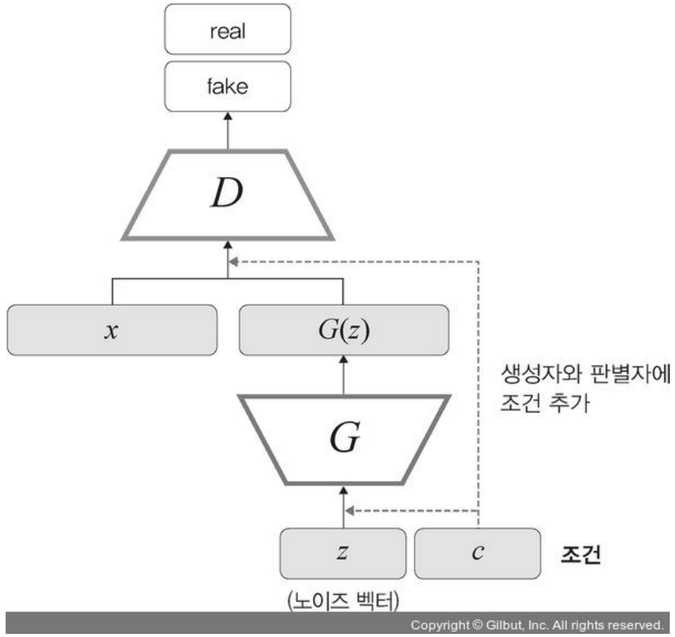
#4.1 DCGAN, cGAN

DCGAN :

GAN과 동일하게 입력된 이미지를 바탕으로 그것과 매우 유사한 가짜 이미지를 만들고, 이를 평가하는 과정을 반복하여 실제와 매우 유사한 이미지를 생산하는 학습법

cGAN :

입력 이미지에 새로운 객체를 추가하거나 이미지에 자동으로 문자열 태그를 붙이고 싶을때 사용

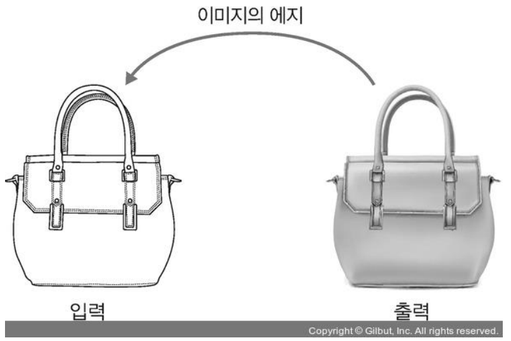


#4.2 CycleGAN

PIX2PIX:

임의의 노이즈 벡터가 아닌 이미지를 입력으로 받아 다른 스타일의 이미지를 출력하는 지도 학습 알고리즘

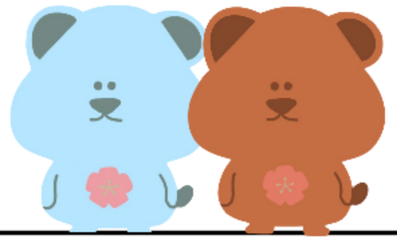
-> 학습하려면 입력을 위한 데이터셋과 PIX2PIX를 거쳐서 나올 정답 이미지가 필요



CycleGAN:

쌍(paired)을 이루지 않는 이미지(unpaired image)로 학습할 수 있는 방법이 필요할때 사용하는 방법

THANK YOU



EWHA EURON