GAN

AILAB
Hanyang Univ.

오늘 실습 내용

1. fully connected GAN 구현

1. GAN 구현

GAN 이란

G = Generative : 그럴듯한 가짜를 생성하는

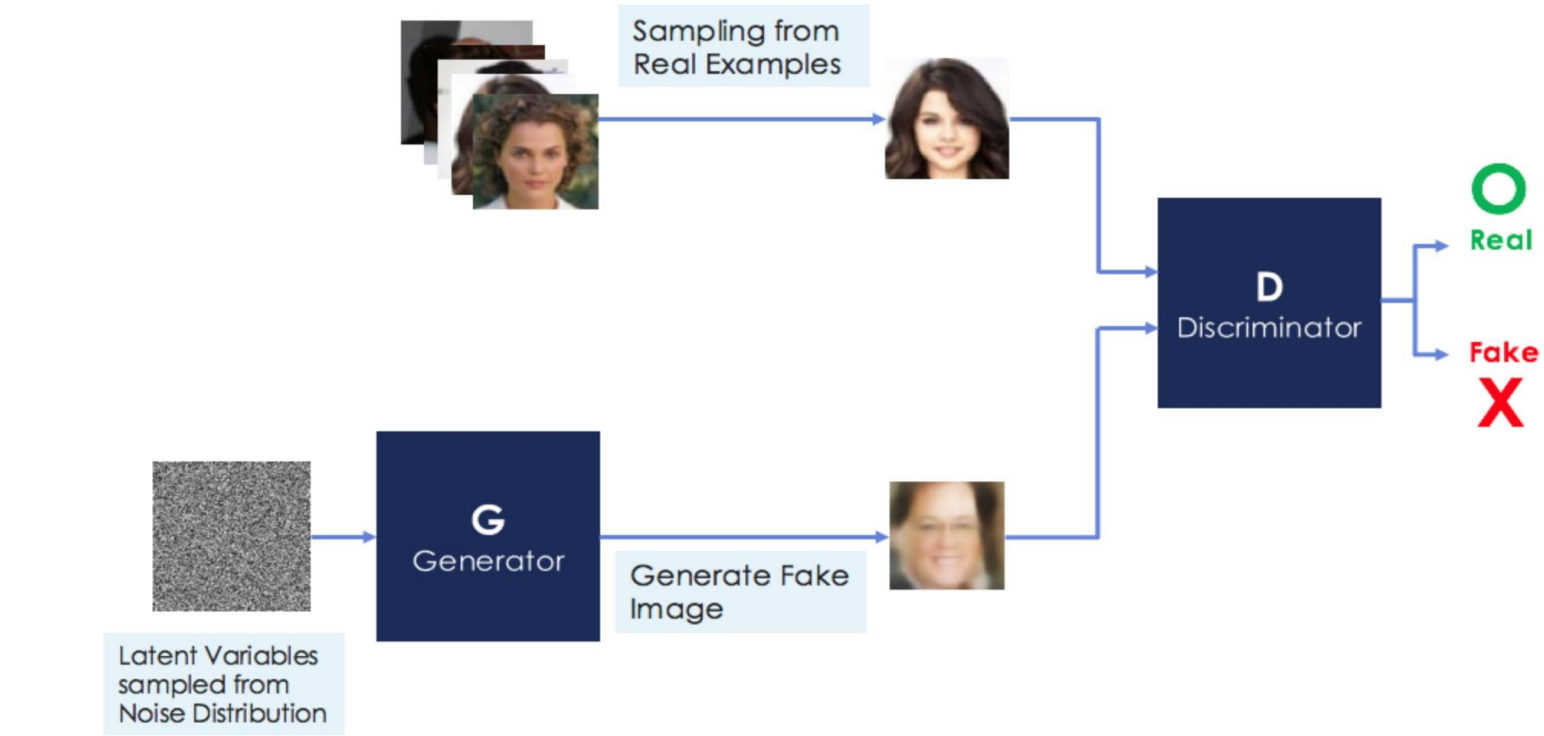
'그럴듯하다?'

- = 수학적으로 보면 실제 데이터의 분포와 비슷한 분포에서 나온 데이터를 의미
- Ex1) 키 172cm, 몸무게 70kg
- Ex2) 키 190cm, 몸무게 10kg => 이 조합은 실제 데이터 분포에서는 거의 나오지 않는 조합

GAN 이란

A = Adversarial : 두개의 모델을 적대적(adversarial)으로 경쟁시키면서 서로의 성능이 발전

(Generator vs. Discriminator)



latent variables = latent z

GAN 이란

N = Network

- Generator 와 Discriminator가 꼭 neural nets 일 필요는 없음
- 뉴럴넷의 장점이 있기 때문에 사용하는 것
  - Non-Linear Activation function
  - Hierarchy structure
  - Backpropagation

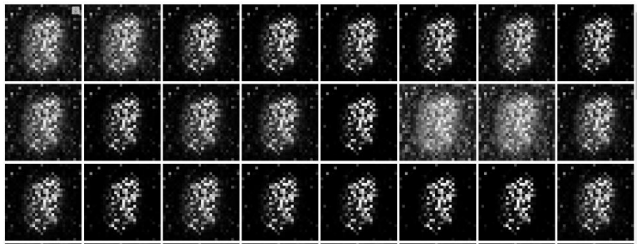
⇒ **Generative Adversarial network(GAN)**

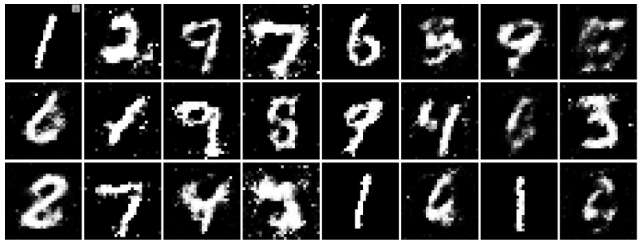
- 생성문제를 풀기위해 딥러닝으로 만들어진 모델을 Adversarial 방식으로 학습시키는 알고리즘

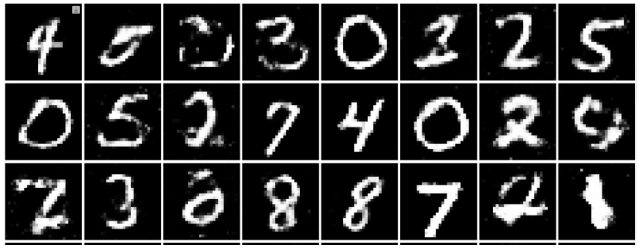
GAN 구현

Implement with MNIST

- MNIST 숫자 손글씨 데이터와 닮은 가짜 손글씨 데이터를 만들어내는 모델 만들기







GAN Loss

Implement with MNIST

Discriminator 의 손실함수

- 실제 이미지일 확률이 높으면 1에 가까운 값을 반환하고,
- 가짜 이미지일 확률이 높으면 0에 가까운 값을 반환하도록 구현

```
real_label = torch.ones((num_img,1)).to(device)
fake_label = torch.zeros((num_img,1)).to(device)
```

```
real_logit = discriminator(b_x)
d_real_loss = criterion(real_logit, real_label)
```

```
fake_data = generator(z)
fake_logit = discriminator(fake_data)
d_fake_loss = criterion(fake_logit, fake_label)

d_loss = d_real_loss + d_fake_loss
```

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

D(x) = 1 일때 Maximum

D(G(z)) = 0 일때 Maximum

GAN Loss

Implement with MNIST

Generator 목적은 Discriminator 속이기

- 생성된 가짜 이미지와 출력값 1과의 차이(gap)를 최소화하도록 손실함수 구현

```
fake_data = generator(z)
fake_logit=discriminator(fake_data)
g_loss = criterion(fake_logit, real_label)
```

1-D(G(z))를 최소화한다는 것은 D(G(z))를 1로 최대화 한다는 것과 같다!

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

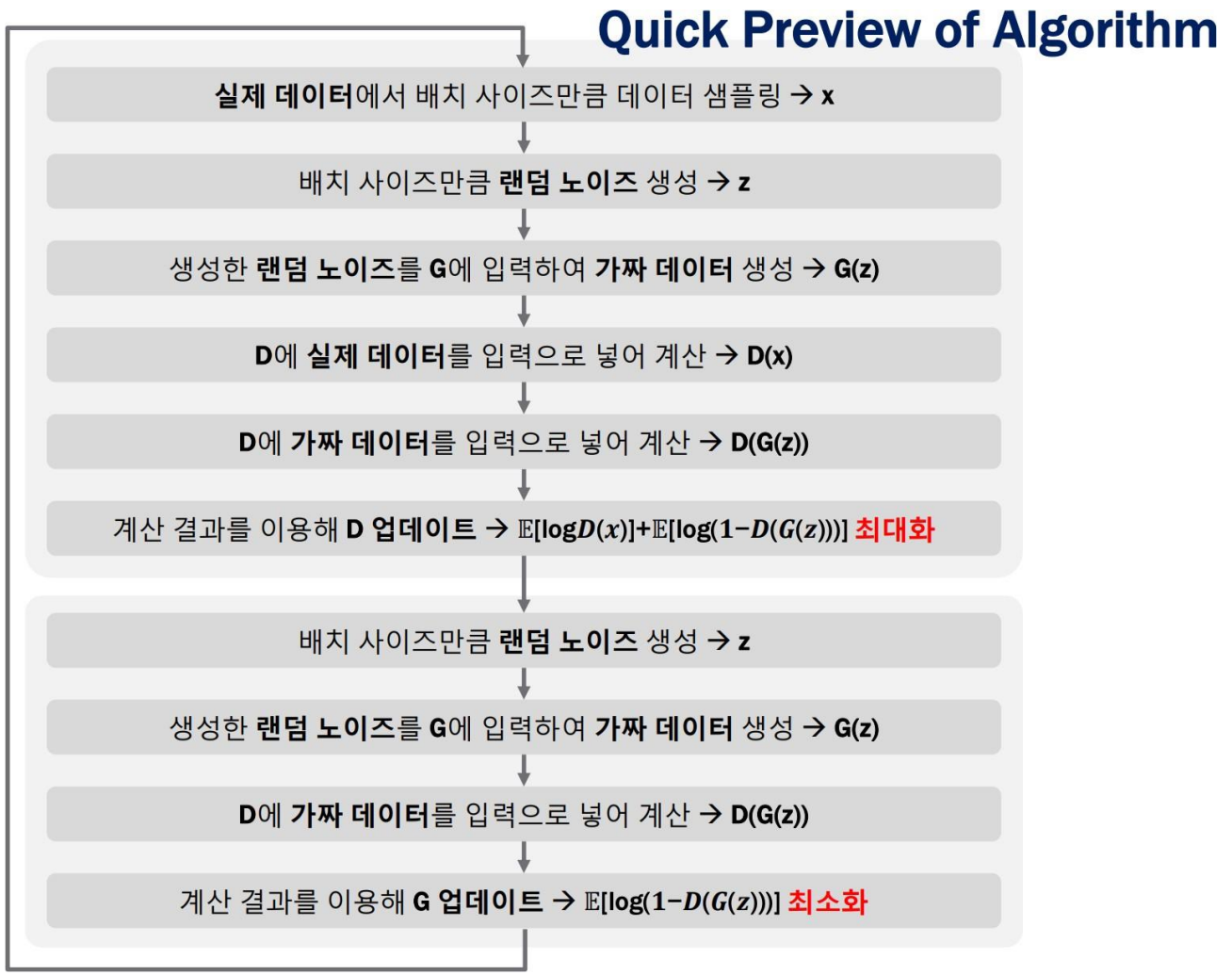
D는 G의 학습에 관여안함

D(G(z)) = 1로 Maximize

GAN 학습

- mini-batch 학습으로 구성
- 우선 매 Step마다 Mini Batch 로 G,D샘플링
  - Dataset으로부터 얻은 x 의 mini-batch
  - Normal distribution 으로부터 얻은 latent z의 mini-batch
- 두개의 Gradient step이 동시에 일어남
    - $J^{(D)}$를 감소시키는 방향으로 $\theta^{(D)}$만을 업데이트하는 gradient descent
    - $J^{(G)}$를 감소시키는 방향으로 $\theta^{(G)}$만을 업데이트하는 gradient descent

GAN 학습


Quick Preview of Algorithm
실제 데이터에서 배치 사이즈만큼 데이터 샘플링 → x
배치 사이즈만큼 랜덤 노이즈 생성 → z
생성한 랜덤 노이즈를 G에 입력하여 가짜 데이터 생성 → G(z)
D에 실제 데이터를 입력으로 넣어 계산 → D(x)
D에 가짜 데이터를 입력으로 넣어 계산 → D(G(z))
계산 결과를 이용해 D 업데이트 → 𝔼[log𝐷(𝑥)]+𝔼[log(1−𝐷(𝐺(𝑧)))] 최대화
배치 사이즈만큼 랜덤 노이즈 생성 → z
생성한 랜덤 노이즈를 G에 입력하여 가짜 데이터 생성 → G(z)
D에 가짜 데이터를 입력으로 넣어 계산 → D(G(z))
계산 결과를 이용해 G 업데이트 → 𝔼[log(1−𝐷(𝐺(𝑧)))] 최소화

GAN 구현

FC로 이루어진 모델

Generator(생성자)

INPUT : z from Latent Variable OUTPUT : 생성된 MNIST 이미지

```
class Generator(nn.Module):
  def __init__(self):
    super(Generator, self).__init__()
    self.fc1 = nn.Linear(100,256)
    self.fc2 = nn.Linear(256,512)
    self.fc3 = nn.Linear(512,1024)
    self.fc4 = nn.Linear(1024,784)
    self.leakyrelu = nn.LeakyReLU(0.2)
    self.tanh = nn.Tanh()

  def forward(self, z):
    x = self.fc1(z)
    x = self.leakyrelu(x)

    x = self.fc2(x)
    x = self.leakyrelu(x)

    x = self.fc3(x)
    x = self.leakyrelu(x)

    x = self.fc4(x)
    x = self.tanh(x)

    return x
```

Discriminator(분류자)

INPUT : 인풋 이미지 OUTPUT : 예측값 : 0 or 1

```
class Discriminator(nn.Module):
  def __init__(self):
    super(Discriminator, self).__init__()
    self.fc1 = nn.Linear(784,1024)
    self.fc2 = nn.Linear(1024,512)
    self.fc3 = nn.Linear(512,256)
    self.fc4 = nn.Linear(256,1)
    self.leakyrelu = nn.LeakyReLU(0.2)
    self.dropout = nn.Dropout(0.3)
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    x = self.fc1(x)
    x = self.leakyrelu(x)
    x = self.dropout(x)

    x = self.fc2(x)
    x = self.leakyrelu(x)
    x = self.dropout(x)

    x = self.fc3(x)
    x = self.leakyrelu(x)
    x = self.dropout(x)

    x = self.fc4(x)
    x = self.sigmoid(x)

    return x
```

GAN 코드 1/5

Wandb 설정

- Loss
- 생성한 이미지 시각화

```
!pip install -q wandb
import wandb
wandb.login()
```

```
[ ]  lr = 0.0002
     batch_size = 128
     config = {
       "dataset": "MNIST",
       "gpu": "colab",
       "model": "GAN",
       "learning_rate": lr,
       "batch_size": batch_size,
     }
     wandb.init(project="week13_gan", config=config)
     wandb.run.name = "mnist gan"
```

GAN 코드 2/5

환경 및 데이터 설정

```
[ ]  import torch
     import torch.nn as nn
     import torch.optim as optim
     import numpy as np
```

```
[ ]  torch.manual_seed(0)
     torch.cuda.manual_seed(0)
     torch.cuda.manual_seed_all(0)
```

```
[ ]  if torch.cuda.is_available():
         device = torch.device('cuda')
     else:
         device = torch.device('cpu')
```

```
     import torchvision
     import torchvision.transforms as transforms
     transform = transforms.Compose([
         transforms.ToTensor(),
         transforms.Normalize((0.5),(0.5))
     ])
     train_dataset = torchvision.datasets.MNIST(root="MNIST_data/",
                         train=True,
                         transform=transform,
                         download=True)
```

```
[ ]  train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

GAN 코드 3/5

$$\text{LeakyRELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$

모델 코드

```
class Generator(nn.Module):
  def __init__(self):
    super(Generator, self).__init__()
    self.fc1 = nn.Linear(100,256)
    self.fc2 = nn.Linear(256,512)
    self.fc3 = nn.Linear(512,1024)
    self.fc4 = nn.Linear(1024,784)
    self.leakyrelu = nn.LeakyReLU(0.2)
    self.tanh = nn.Tanh()

  def forward(self, z):
    x = self.fc1(z)
    x = self.leakyrelu(x)

    x = self.fc2(x)
    x = self.leakyrelu(x)

    x = self.fc3(x)
    x = self.leakyrelu(x)

    x = self.fc4(x)
    x = self.tanh(x)

    return x
```

```
class Discriminator(nn.Module):
  def __init__(self):
    super(Discriminator, self).__init__()
    self.fc1 = nn.Linear(784,1024)
    self.fc2 = nn.Linear(1024,512)
    self.fc3 = nn.Linear(512,256)
    self.fc4 = nn.Linear(256,1)
    self.leakyrelu = nn.LeakyReLU(0.2)
    self.dropout = nn.Dropout(0.3)
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    x = self.fc1(x)
    x = self.leakyrelu(x)
    x = self.dropout(x)

    x = self.fc2(x)
    x = self.leakyrelu(x)
    x = self.dropout(x)

    x = self.fc3(x)
    x = self.leakyrelu(x)
    x = self.dropout(x)

    x = self.fc4(x)
    x = self.sigmoid(x)

    return x
```

GAN 코드 4/5

모델 코드

```
[ ]  generator = Generator().to(device)
     discriminator = Discriminator().to(device)
```

```
[ ]  criterion = torch.nn.BCELoss()
     g_optimizer = torch.optim.Adam(generator.parameters(), lr=lr)
     d_optimizer = torch.optim.Adam(discriminator.parameters(), lr=lr)
```

GAN 코드 5-1/5

학습 코드

```
[ ]  epochs = 200
     total_batch_num = len(train_dataloader)

     for epoch in range(epochs):
       generator.train()
       discriminator.train()

       avg_g_cost = 0
       avg_d_cost = 0

       for step, batch in enumerate(train_dataloader):
         b_x, _ = batch
         b_x = b_x.view(-1, 784).to(device)
         num_img = len(b_x)
         real_label = torch.ones((num_img,1)).to(device)
         fake_label = torch.zeros((num_img,1)).to(device)

         real_logit = discriminator(b_x)
         d_real_loss = criterion(real_logit, real_label)

         z = torch.randn((num_img,100),requires_grad=False).to(device)
         fake_data = generator(z)
         fake_logit = discriminator(fake_data)
         d_fake_loss = criterion(fake_logit, fake_label)

         d_loss = d_real_loss + d_fake_loss
         d_optimizer.zero_grad()
         d_loss.backward()
         d_optimizer.step()
```

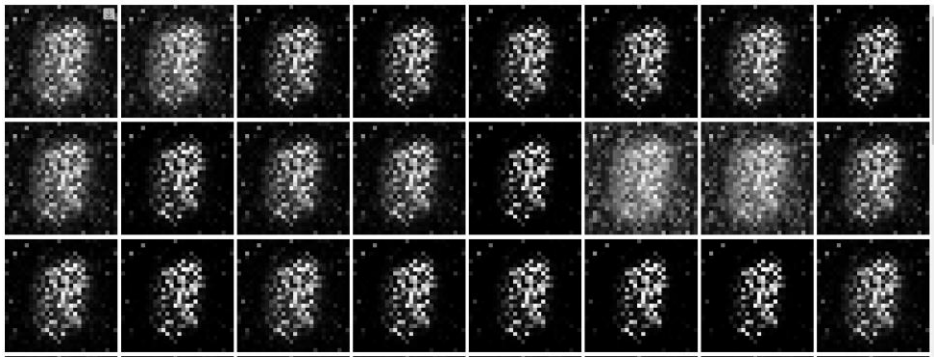
GAN 코드 5-2/5

학습 코드

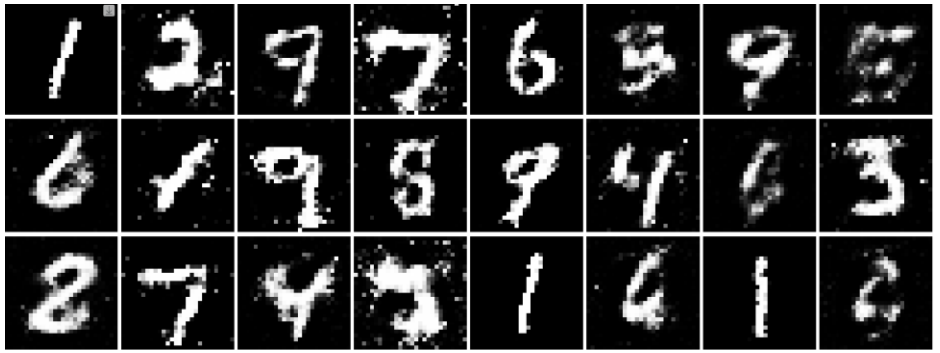
```
        z = torch.randn((num_img,100),requires_grad=False).to(device)
        fake_data = generator(z)
        fake_logit=discriminator(fake_data)
        g_loss = criterion(fake_logit, real_label)

        g_optimizer.zero_grad()
        g_loss.backward()
        g_optimizer.step()

        avg_d_cost += d_loss
        avg_g_cost += g_loss

    avg_d_cost /= total_batch_num
    avg_g_cost /= total_batch_num

    # observe fake images
    generator.eval()
    with torch.no_grad():
        z = torch.randn((64,100),requires_grad=False).to(device)
        fake_data = generator(z)

        fake_img = fake_data.detach().cpu().numpy().reshape(64,28,28)
        wandb.log({"discriminator loss": avg_d_cost, "generator loss":avg_g_cost, 'fake image': [wandb.Image(i) for i in fake_img]})
```

GAN wandb

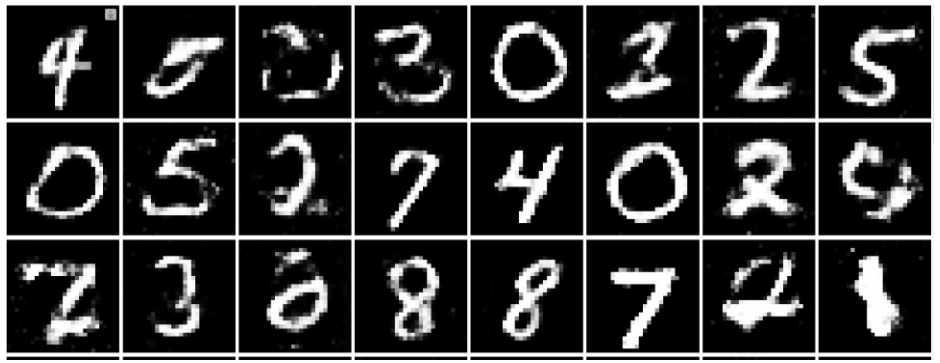
학습 결과



epoch 0



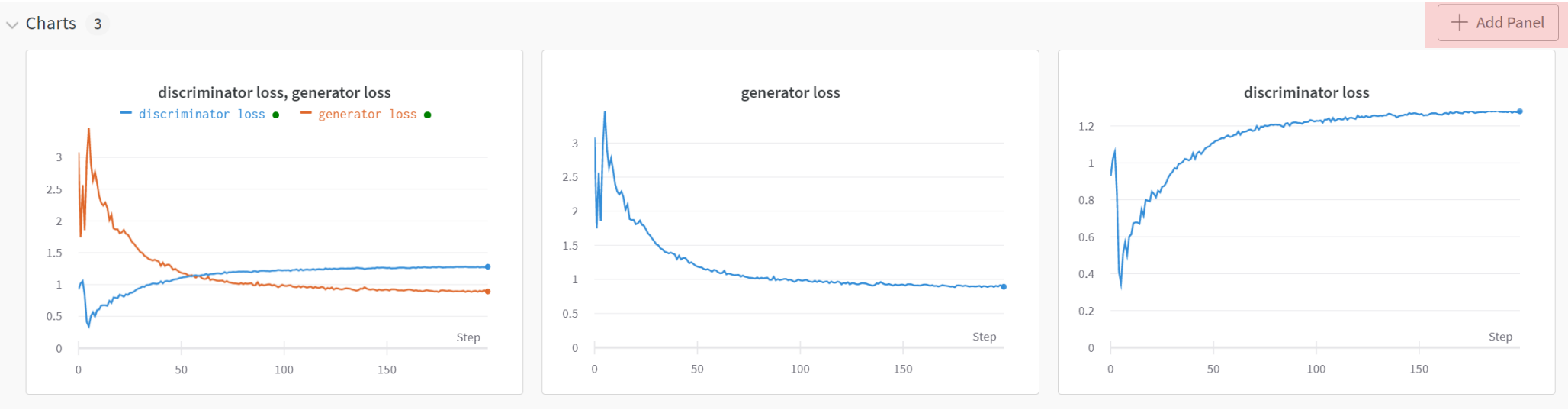
epoch 50

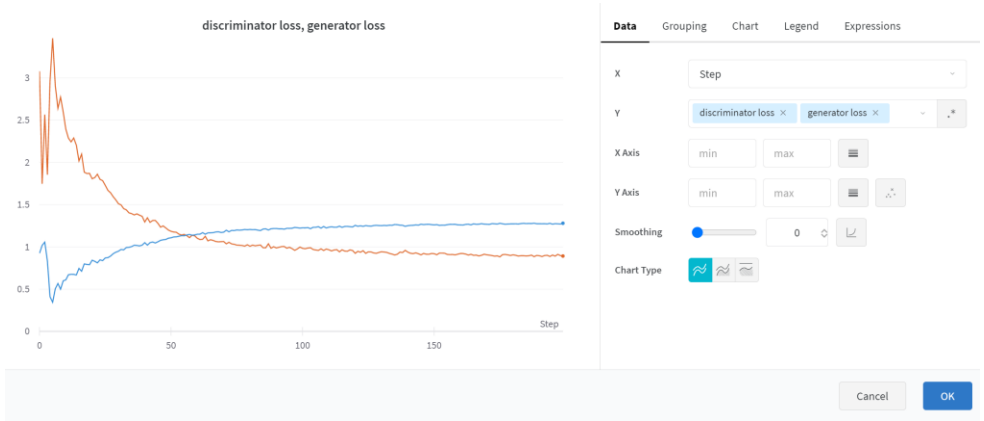


epoch 199

GAN wandb

학습 결과





FID

FID(Frechet Inception distance)

- GAN의 Generator가 생성하는 이미지를 평가하는 metric
  - fake image가 real image처럼 잘 만들었는지
  - fake image가 다양하게 만들어졌는지

- Imagenet 데이터세트에서 사전 훈련된 Inception V3 모델
  - 모델 끝에서 두 번째 풀링 레이어의 결과를 가져옴 shape (2048,)
  - 이를 사용하여 real, fake image의 관계 계산

$$\mathrm{FID} = ||\mu_X - \mu_Y||^2 - \mathrm{Tr}(\sum_X + \sum_Y - 2 \quad \sum_X \sum_Y)$$

μ_X : X의 평균 벡터
μ_Y: Y의 평균 벡터
Σ_X : X의 공분산
Σ_Y : Y의 공분산

  - 값이 작을 수록 좋음

FID 코드

- pytorch_fid install

```
!pip install -q pytorch_fid
```

```
!mkdir fake_img
!mkdir real_img
```

Scaling up to 0 to 255
RGB

```
# FID score 측정에 사용할 fake 이미지를 생성하는 코드 입니다.
# generator의 학습을 완료한 뒤 마지막에 실행하여 fake 이미지를 저장하시기 바랍니다.
from PIL import Image

num_img = 1000
test_noise = torch.randn(num_img, 100, device=device)
with torch.no_grad():
  test_fake = generator(test_noise).detach().cpu().numpy()

  for index, img in enumerate(test_fake):
    fake = np.reshape(img, (28,28))
    fake = (fake * 127.5 + 127.5).astype(np.uint8)
    fake = np.expand_dims(fake,axis=2)
    fake=np.repeat(fake,3,axis=2)
    im = Image.fromarray(fake)
    im.save("./fake_img/fake_sample{}.jpeg".format(index))
```

Scaling up to 0 to 255
RGB

```
for i in range(num_img):
    real = np.reshape(train_dataset[i][0].detach().cpu().numpy(), (28,28))
    real = (real * 127.5 + 127.5).astype(np.uint8)
    real = np.expand_dims(real,axis=2)
    real=np.repeat(real,3,axis=2)
    im = Image.fromarray(real)
    im.save("./real_img/real_sample{}.jpeg".format(i))
```

- 함수 사용하기 위해서 fake image가 저장된 path와 real image가 저장된 path각각 필요하다.

FID 코드

- calculate_fid_given_paths 사용해서 FID 계산

```
100%|██████████| 8/8 [00:03<00:00,  2.20it/s]
100%|██████████| 8/8 [00:03<00:00,  2.20it/s]
fid score : 280.422095531941
```

```python
import os
import torch

from pytorch_fid.fid_score import *

os.environ['KMP_DUPLICATE_LIB_OK']='True'

real_img_path = 'real_img/'
fake_img_path = 'fake_img/'

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.cuda.manual_seed_all(0)

if __name__ == "__main__":
    fid = calculate_fid_given_paths(
        paths=[real_img_path, fake_img_path],
        batch_size=128,
        device=device,
        dims=2048
    )

    print("fid score : {}".format(fid))
```

오늘 실습 내용

1. MNIST FC GAN 학습해보기