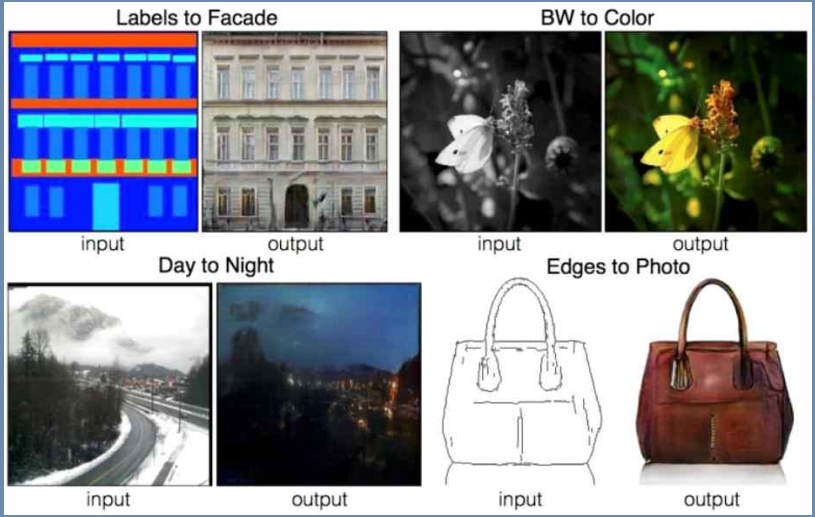
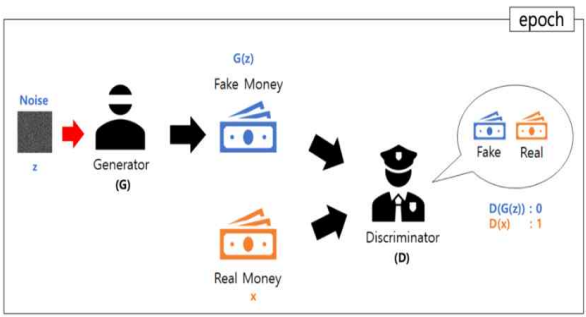
GAN

(Generative Adversarial Networks)

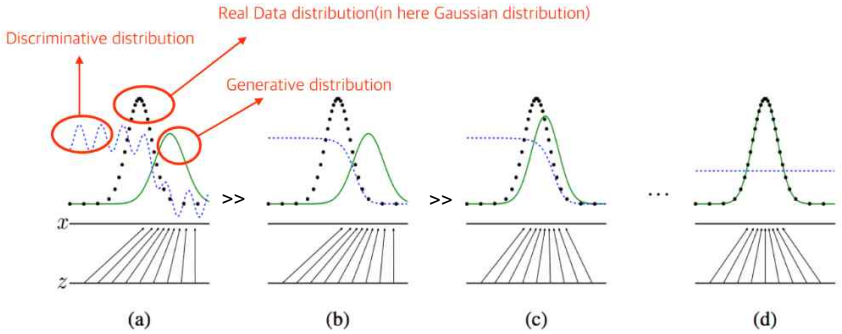
GAN은 Generative Adversarial Networks의 약자로 우리말로는
“적대적 생성 신경망”이라고 번역되는 기술 중 하나입니다.
Ian Goodfellow에 의해 2014년 처음으로 신경정보처리시스템
학회(NIPS)에서 제안되었고
이후 다양한 노문들로 파생되며 발전해 이미지 생성, 영상 생성,
텍스트 생성 등에 다양하게 응용되고 있습니다.

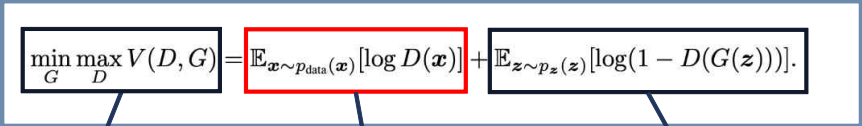

Labels to Facade
input
output
BW to Color
input
output
Day to Night
input
output
Edges to Photo
input
output

GAN은 Generator와 Discriminator라는 서로 다른 2개의 네트워크로 이루어져 있으며 이 두 네트워크를 적대적으로 학습시키며 목적을 달성합니다. 예시로 생성 모델은 진짜 지폐와 비슷한 가짜 지폐를 만들어 경찰을 속이려 하는 위조지폐범과 같고, 반대로 판별모델은 위조지폐범이 만들어낸 가짜 지폐를 탐지하려는 경찰과 유사합니다. 이러한 경쟁이 계속됨에 따라 위조지폐범은 경찰을 속이지 못한 데이터를, 경찰은 위조지폐범에게 속은 데이터를 각각 입력받아 적대적으로 학습하게 되는 것입니다. 이 게임에서의 경쟁은 위조지폐가 진짜 지폐와 구별되지 않을 때까지 즉, 주어진 표본이 실제 표본이 될 확률이 0.5에 가까운 값을 가질 때까지 계속됩니다. 가짜로 확신하는 경우 판별기의 확률값이 0, 실제로 확신하는 경우 판별기의 확률값이 1을 나타내게 되며, 판별기의 확률값이 0.5라는 것은 가짜인지 진짜인지 판단하기 어려운 것을 의미하게 되는 것입니다.



Generative Model은 원래 데이터의 분포를 근사할 수 있도록 학습




$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$
V(G) : 값을 최소화 하도록 학습(빨간색 박스의 식에서는 Discriminator를 사용하지 않음으로 생략
V(D) : 값을 최대화 하도록 학습
원본데이터에서 한개의 데이터(x)를 꺼내 Discriminator에 넣어 log를 취한값의 기대값
하나의 노이즈(z)를 뽑아 Generator에 넣어 가짜 데이터를 생성한 다음 Discriminator에 넣은 값을 마이너스해 1를 더한값의 로그를 취한 값의 기대값

필요 라이브러리 , 데이터 불러오기

```python
import torch
import torch.nn as nn
import numpy as np
from torchvision import transforms
from torch.autograd import Variable
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
from torchvision import datasets
import torchvision
```

```python
transform = transforms.Compose([
        transforms.Resize(28),
        transforms.CenterCrop(28),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ToTensor(),
        transforms.Normalize(mean=(  0.5), std=( 0.5))
])
dataset = datasets.MNIST(root="./dataset", train=True, download=True, transform=transform)
data_loader = torch.utils.data.DataLoader(dataset, batch_size=64, shuffle=True, num_workers=4)
```

```python
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(794, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x, labels):
        x = x.view(x.size(0), 784)
        c = self.label_emb(labels)
        x = torch.cat([x, c], 1)
        out = self.model(x)
        return out.squeeze()
```

Discriminator클래스정의
활성화 함수 : LeakyReLU 사용
마지막에 Sigmiod함수를 사용해 0~1값 가지도록 설정

```python
class Generator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(110, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(1024, 784),
            nn.Tanh()
        )

    def forward(self, z, labels):
        z = z.view(z.size(0), 100)
        c = self.label_emb(labels)
        x = torch.cat([z, c], 1)
        out = self.model(x)
        return out.view(x.size(0), 28, 28)
```

Generator클래스정의
마지막에 Tanh를 사용해 -1~1의 값을 가지도록 설정

loss,optimizer 함수설정

```
criterion = nn.BCELoss()
d_optimizer = torch.optim.Adam(discriminator.parameters(), lr=1e-4)
g_optimizer = torch.optim.Adam(generator.parameters(), lr=1e-4)
```

Generator의 loss값 계산 함수

```
def generator_train_step(batch_size, discriminator, generator, g_optimizer, criterion):
    g_optimizer.zero_grad()
    z = Variable(torch.randn(batch_size, 100)).cuda()
    fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).cuda()
    fake_images = generator(z, fake_labels)
    validity = discriminator(fake_images, fake_labels)
    g_loss = criterion(validity, Variable(torch.ones(batch_size)).cuda())
    g_loss.backward()
    g_optimizer.step()
    return g_loss.data
```

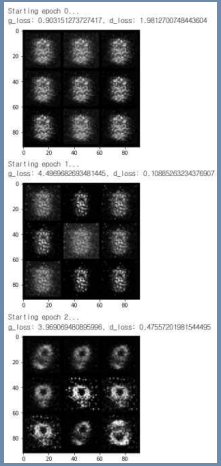
Discriminator의 loss값 계산 함수

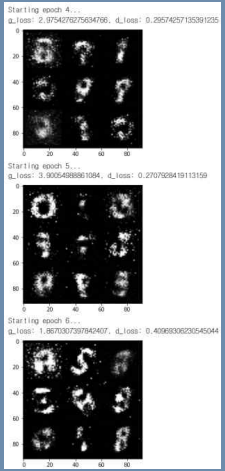
```
def discriminator_train_step(batch_size, discriminator, generator, d_optimizer, criterion, real_images, labels):
    d_optimizer.zero_grad()

    # train with real images
    real_validity = discriminator(real_images, labels)
    real_loss = criterion(real_validity, Variable(torch.ones(batch_size)).cuda())

    # train with fake images
    z = Variable(torch.randn(batch_size, 100)).cuda()
    fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).cuda()
    fake_images = generator(z, fake_labels)
    fake_validity = discriminator(fake_images, fake_labels)
    fake_loss = criterion(fake_validity, Variable(torch.zeros(batch_size)).cuda())

    d_loss = (real_loss + fake_loss) / 2
    d_loss.backward()
    d_optimizer.step()
    return d_loss.data
```

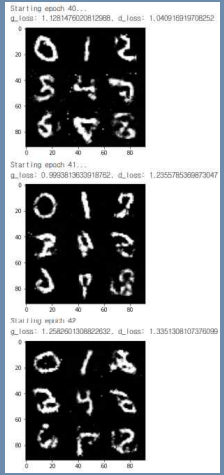
학습

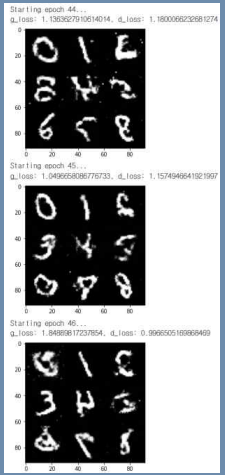
```
num_epochs = 30
n_critic = 5
display_step = 300
for epoch in range(num_epochs):
    print('Starting epoch {}...'.format(epoch))
    for i, (images, labels) in enumerate(data_loader):
        real_images = Variable(images).cuda()
        labels = Variable(labels).cuda()
        generator.train()
        batch_size = real_images.size(0)
        d_loss = discriminator_train_step(len(real_images), discriminator,
                                          generator, d_optimizer, criterion,
                                          real_images, labels)


        g_loss = generator_train_step(batch_size, discriminator, generator, g_optimizer, criterion)

    generator.eval()
    print('g_loss: {}, d_loss: {}'.format(g_loss, d_loss))
    z = Variable(torch.randn(9, 100)).cuda()
    labels = Variable(torch.LongTensor(np.arange(9))).cuda()
    sample_images = generator(z, labels).unsqueeze(1).data.cpu()
    grid = make_grid(sample_images, nrow=3, normalize=True).permute(1,2,0).numpy()
    plt.imshow(grid)
    plt.show()
```

학습결과


Starting epoch 0...
g_loss: 0.9031527327417, d_loss: 1.9812700748443604
Starting epoch 1...
g_loss: 4.499968283081445, d_loss: 0.10885263234376907
Starting epoch 2...
g_loss: 3.9606948089996, d_loss: 0.47557219815444946

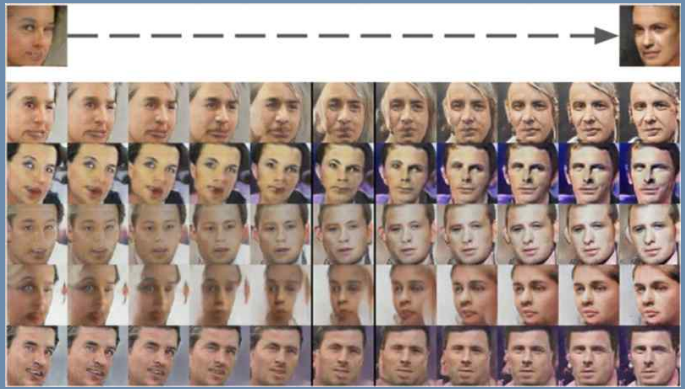

Starting epoch 4...
g_loss: 2.975427627563476, d_loss: 0.2557425713539123
Starting epoch 5...
g_loss: 3.900548886108494, d_loss: 0.27079284191113159
Starting epoch 6...
g_loss: 1.8670307397842407, d_loss: 0.4096930623054504


Starting epoch 40...
g_loss: 1.1281476020812988, d_loss: 1.0409169197082522
Starting epoch 41...
g_loss: 0.9993813633918762, d_loss: 1.2355785369873047
Starting epoch 42...
g_loss: 1.2526901388922632, d_loss: 1.3351308107376099


Starting epoch 44...
g_loss: 1.1363929510614014, d_loss: 1.1800066232681274
Starting epoch 45...
g_loss: 1.0496958088776733, d_loss: 1.1574996641921997
Starting epoch 46...
g_loss: 1.0488981723785, d_loss: 0.9966501998996489

생성된 FAKE 이미지

https://www.kaggle.com/code/arturlacerda/pytorch-conditional-gan



생성 모델로서의 **GAN**이 데이터를 우연히 만들어 내는 것인지, 데이터를 완벽히 이해하고 있는 가치 있는 모델인지를 알아보기 위해 **G**의 출력이 사람의 얼굴이라고 했을 때, 왼쪽을 바라보는 얼굴을 만들어 내는 **z**(왼쪽)들의 평균벡터와 오른쪽을 보고 있는 얼굴에 대응하는 **z**(오른쪽)들의 평균을 계산하고 이 두 벡터 사이의 축을 중간에서 **interpolation**하여 생성자로 입력하면 천천히 회전하는 얼굴이 나오는 것을 확인할 수 있습니다.
이 결과는 **GAN**생성자가 학습한 딥러닝 알고리즘이 정확히 데이터의 의미를 이해하고 데이터의 확률분포를 정확히 표현하고 있어서, 입력에서의 약간의 변화가 출력에서도 부드러운 변화로 표현 가능하다는 사실을 보여줍니다.

한계점

기존의 GAN의 한계점은 크게 2가지로 나뉩니다.

1. (성능 평가)

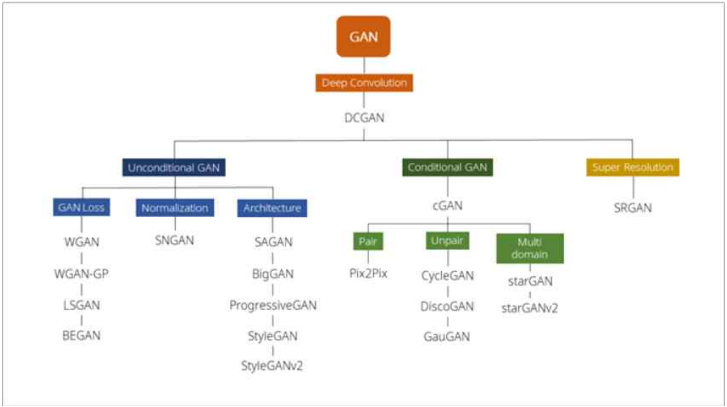
GAN 모델의 성능을 객관적 수치로 표현할 수 있는 방안이 부재 GAN의 경우 결과 자체가 새롭게 만들어진 데이터이기 때문에 비교 가능한 정량적 척도가 없음

2. (성능 개선)

GAN은 기존 네트워크 학습 방법과 다른 구조로 학습이 불안정함. GAN은 Saddle Problem 혹은 Minmax를 풀어야 하는 태생적으로 불안정한 구조

하지만 이의 두 단점을 모두 개선하여 GAN의 후속 연구가 줄줄이 이어나올 수 있도록 DCGAN(Deep Convolutional GAN)이 개발 되었습니다.

GAN 종류


GAN
Deep Convolution
DCGAN
Unconditional GAN
Conditional GAN
Super Resolution
GAN Loss
Normalization
Architecture
cGAN
SRGAN
WGAN
SNGAN
SAGAN
Pair
Unpair
Multi domain
WGAN-GP
BigGAN
Pix2Pix
CycleGAN
starGAN
LSGAN
ProgressiveGAN
DiscoGAN
starGANv2
BEGAN
StyleGAN
GauGAN
StyleGANv2

감사합니다.