Vanilla GAN 실습

임세진

https://youtu.be/SL9s6hcdwzM

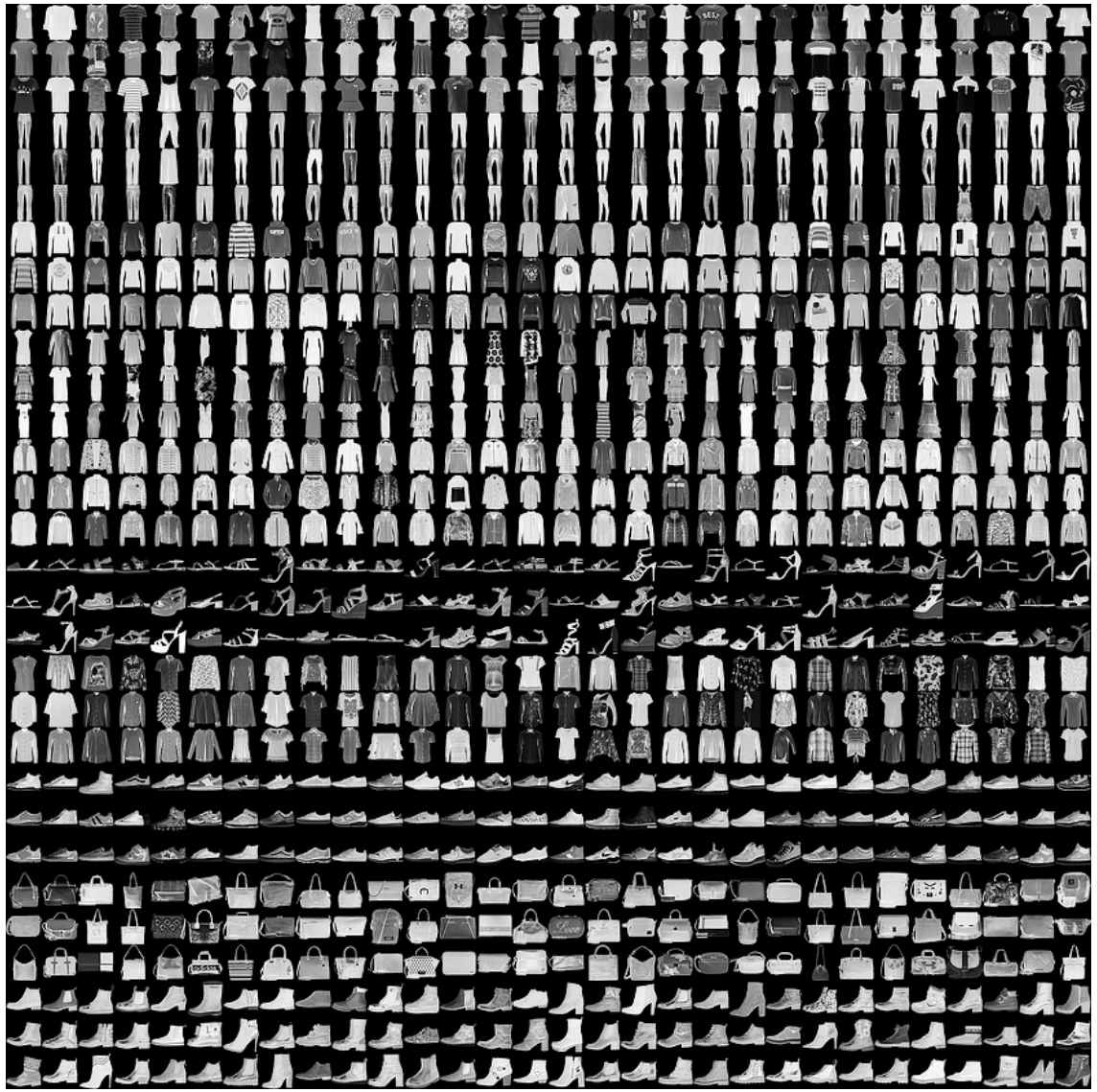
Contents

CryptoCraft LAB

데이터셋

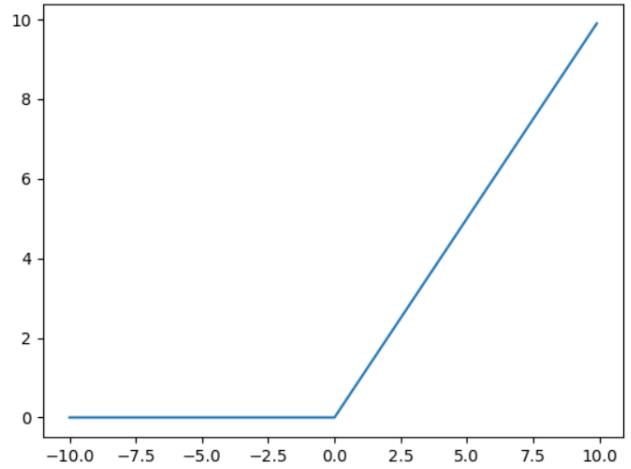
- Fashion-MNIST
- 10가지 패션 아이템에 대한 흑백 이미지와 라벨

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



안정적인 GAN training 기법

✓ Generator에는 ReLU를, Discriminator에서 Leaky ReLU를 사용함



ReLU : 임계값보다 작으면 0, 크면 입력값을 출력

→ Knockout 문제가 발생할 수 있음

어떤 layer에서 모든 노드의 미분 값이 0이 나오면 이후 layer는 학습 X

➔ Leaky ReLU 사용 : 임계치보다 작을 때 0.01을 곱함

✓ G에서 output layer에 Tanh(hyperbolic tangent)를 사용

✓ D에서 loss가 0.00이면 실패한 학습임

✓ G의 loss가 일정하게 감소할 경우, D가 fake img로 인해 잘못 학습되고 있을 가능성 ↑

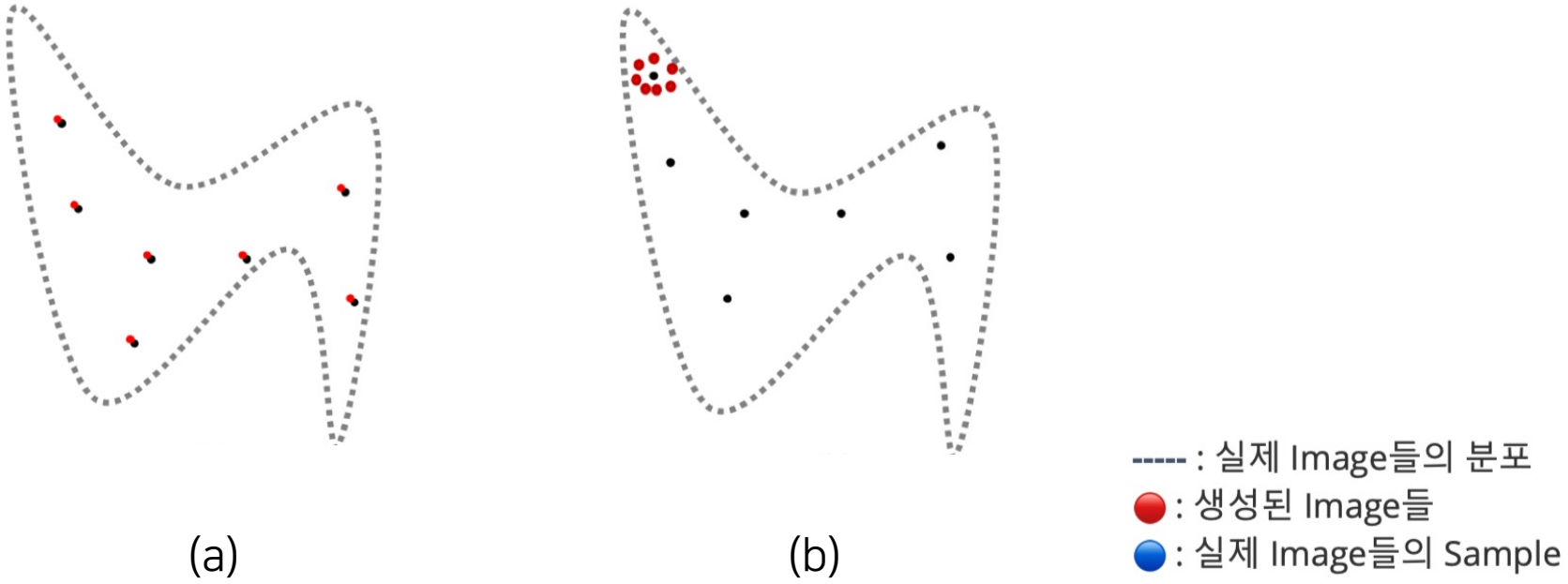
GAN의 평가 지표

GAN을 통해 생성된 img들의 문제점

(a) Overfitting

(b) Mode Collapse : G가 다양한 이미지를 만들어내지 못하고 비슷한 이미지만 계속 생성하는 경우

→D와 G가 학습이 서로 상호작용을 하면서 학습 되어야하는데 한쪽만 학습이 잘 된 경우 (학습의 불균형)



(a) (b)

GAN의 평가 지표

사람 육안으로 GAN 성능 평가 시 문제점

- 평가하는 사람에 따라 지표가 나눠지므로 주관적
- 평가하는 사람의 도메인 지식의 영향이 큼 (적용할 수 있는 분야가 한정적)
- Mode Collapse가 일어났는지 알 수 없음

GAN의 평가 지표

1) CID index : 3가지 평가지표를 곱한 값

$$CID = Creativity \times Inheritance \times Diversity$$

- Creativity : 실제 img와 중복이 있으면 안됨
- Inheritance : 생성된 img는 같은 스타일을 가져야함
- Diversity : 생성된 img들은 서로 달라야함

GAN의 평가 지표

2) Inception Score (IS)

- 이미지 분류 모델에 G가 생성한 이미지를 입력했을 때, 높은 확률로 해당 클래스로 예측되면, 생성 이미지가 실제 이미지와 비슷하다고 평가

- 이미지 분류 모델의 weight에 따라 score가 크게 달라짐

- G가 img의 Class당 하나의 img만 생성하고 그 이미지를 여러 번 반복해서 생성하는 경우에도 IS를 높게 받을 수 있음 → Class 내의 이미지 다양성 측정 X

- 실제 이미지를 고려하지 않음

GAN의 평가 지표

3) Fréchet Inception Distance (FID)

- IS는 생성된 이미지만 사용하여 성능을 평가하지만, FID는 대상 도메인의 실제 이미지 모음 통계와 생성된 이미지 모음 통계를 비교해 평가 수행
- 점수가 낮을수록 GAN의 성능이 좋은 것을 의미

  (FID가 0일 경우 가장 이상적인 모델. 평균적으로 FID가 10 내외이면 좋은 모델)

- 특정 생성 기법이 적용된 이미지에만 의미가 있음

실습 코드

<모델 구조>

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.n_features = 128 # latent space에서 나가는 벡터의 크기
        self.n_out = 784 # 28X28
        self.linear = nn.Sequential(
                    nn.Linear(self.n_features, 256),
                    nn.ReLU(),
                    nn.Linear(256, 512),
                    nn.ReLU(),
                    nn.Linear(512, 1024),
                    nn.ReLU(),
                    nn.Linear(1024, self.n_out),
                    nn.Tanh() # 하이퍼볼릭 탄젠트 함수 -1 ~ 1 사이의 값
                    )

    def forward(self, z):
        x = self.linear(z)
        x = x.view(-1, 1, 28, 28)
        return x
```

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.n_in = 784
        self.n_out = 1
        self.linear = nn.Sequential(
                    nn.Linear(self.n_in, 1024),
                    nn.LeakyReLU(0.2),
                    nn.Dropout(0.3),
                    nn.Linear(1024, 512),
                    nn.LeakyReLU(0.2),
                    nn.Dropout(0.3),
                    nn.Linear(512, 256),
                    nn.LeakyReLU(0.2),
                    nn.Dropout(0.3),
                    nn.Linear(256, self.n_out),
                    nn.Sigmoid()
                    )
    def forward(self, x):
        x = x.view(-1, 784)
        x = self.linear(x)
        return x
```

실습 코드

<손실함수 + optimizer 정의>

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

# 둘 다 학습 시켜줘야해서 옵티마이저가 2개
g_optim = optim.Adam(generator.parameters(), lr=2e-4)
d_optim = optim.Adam(discriminator.parameters(), lr=2e-4)

g_losses = []
d_losses = []

criterion = nn.BCELoss()

# latent space에 있는 벡터
def noise(n, n_features=128):
    return Variable(torch.randn(n, n_features)).to(device)

def label_ones(size):
    data = Variable(torch.ones(size, 1))
    return data.to(device)

def label_zeros(size):
    data = Variable(torch.zeros(size, 1))
    return data.to(device)
```

실습 코드

<Training 함수>

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

- Expectation
- x is sampled from real data
- Probability of D(real)
- z is sampled from N(0, 1)
- Probability of D(fake)
- fake

```python
def train_discriminator(optimizer, real_data, fake_data):
    n = real_data.size(0)

    optimizer.zero_grad()

    prediction_real = discriminator(real_data)
    # 진짜 이미지 기준 Loss 구하기
    d_loss = criterion(prediction_real, label_ones(n))

    prediction_fake = discriminator(fake_data)
    # 가짜 이미지 기준 Loss 구하기
    g_loss = criterion(prediction_fake, label_zeros(n))

    loss = d_loss + g_loss

    loss.backward()
    optimizer.step()

    return loss.item()
```

```python
def train_generator(optimizer, fake_data):
    n = fake_data.size(0)
    optimizer.zero_grad()

    prediction = discriminator(fake_data)
    # G와 D 모두 동일한 Loss 사용 > 진짜 라벨과 비교하여 로스 계산
    # 이 로스가 작다는 것은 D가 잘 구별을 못한다는 뜻
    loss = criterion(prediction, label_ones(n))

    loss.backward()
    optimizer.step()

    return loss.item()
```

실습 코드

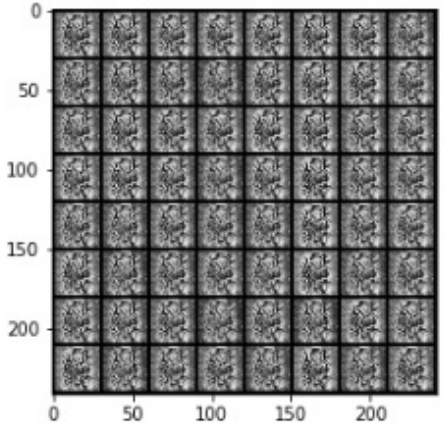
<학습>

```
num_epochs = 101
test_noise = noise(64) # G에 들어가는 input, 128짜리 벡터를 64개 생성

l = len(trainloader)

for epoch in range(num_epochs):
    g_loss = 0.0
    d_loss = 0.0

    for data in trainloader:
        imgs, _ = data # 비지도 학습이라 라벨 필요X
        n = len(imgs)

        fake_data = generator(noise(n)).detach()
        real_data = imgs.to(device)
        d_loss += train_discriminator(d_optim, real_data, fake_data) # D 학습
        fake_data = generator(noise(n))
        g_loss += train_generator(g_optim, fake_data) # G 학습

    img = generator(test_noise).cpu().detach()
    img = make_grid(img)
    g_losses.append(g_loss/l)
    d_losses.append(d_loss/l)

    if epoch % 10 == 0:
        print('Epoch {}: g_loss: {:.3f} d_loss: {:.3f}\r'.format(epoch, g_loss/l, d_loss/l))
        plt.imshow(np.array(to_image(img)))
        plt.show()
```

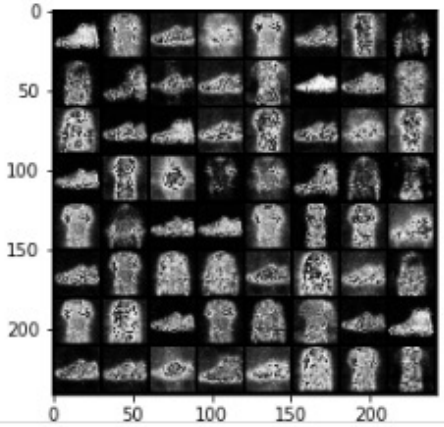
실습 코드

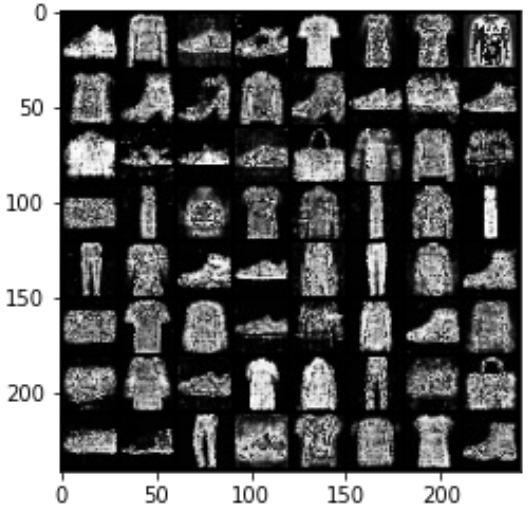
<결과>

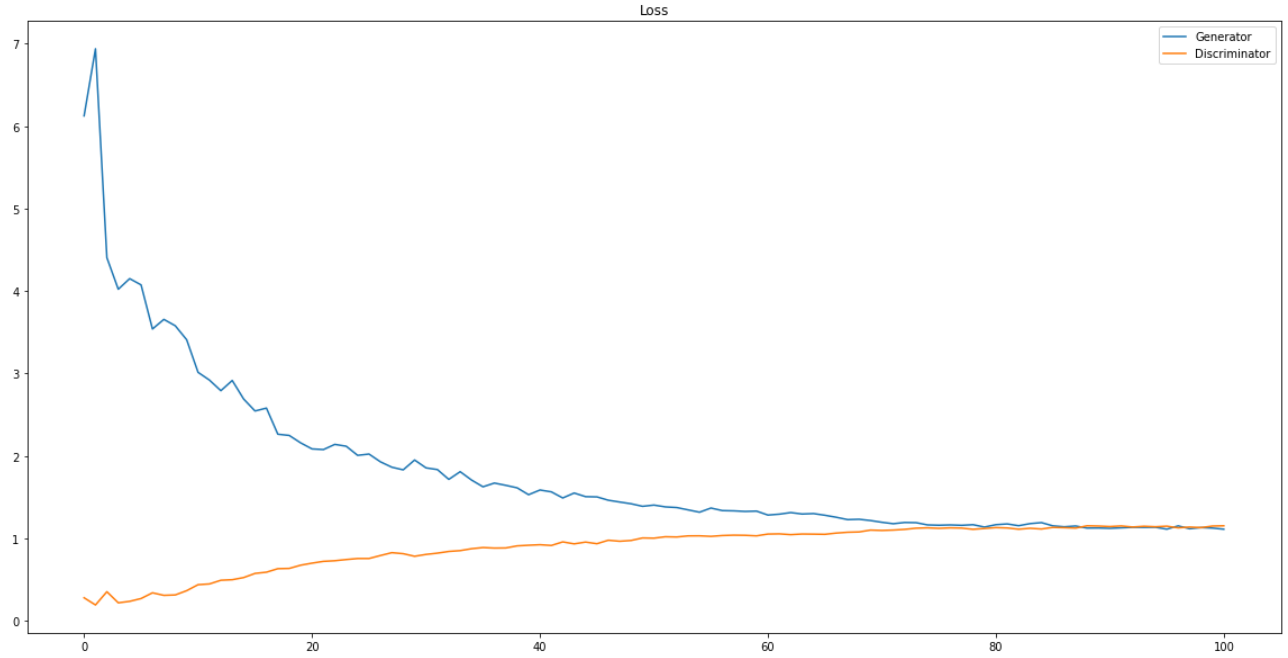
Epoch 0: g_loss: 6.123 d_loss: 0.281



Epoch 10: g_loss: 3.014 d_loss: 0.440







<loss>

감사합니다