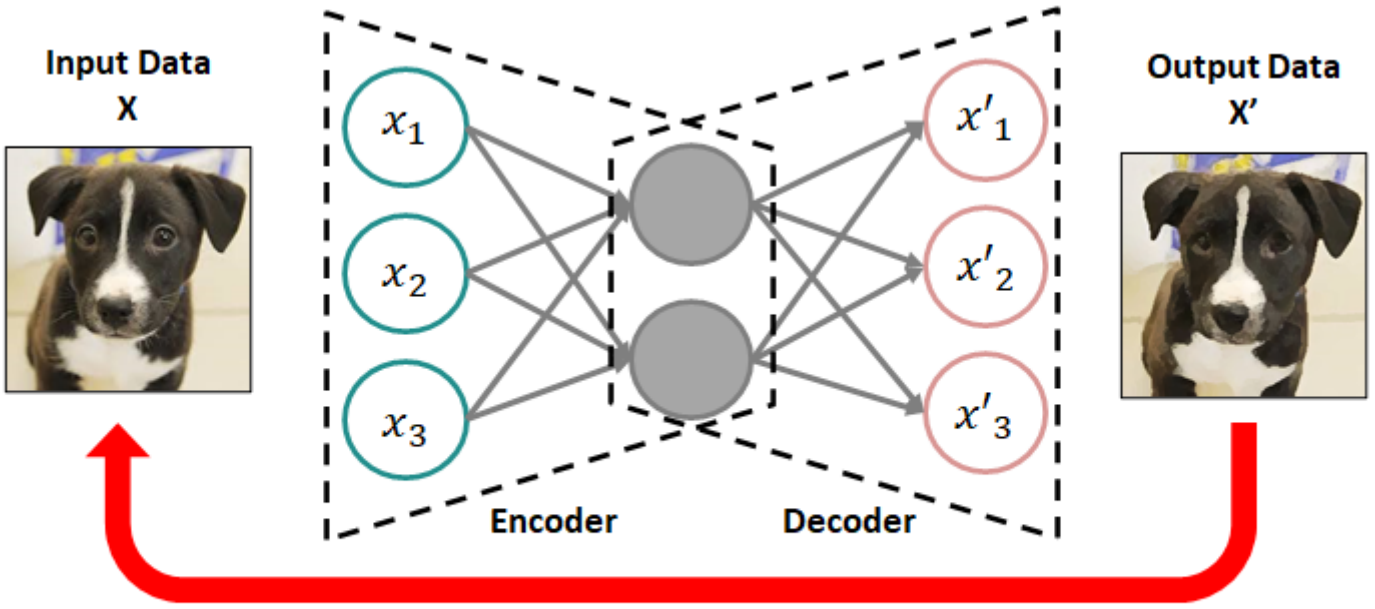
4일차 오토인코더 및 GAN
강의자료: https://github.com/nahyungsun/tutorial

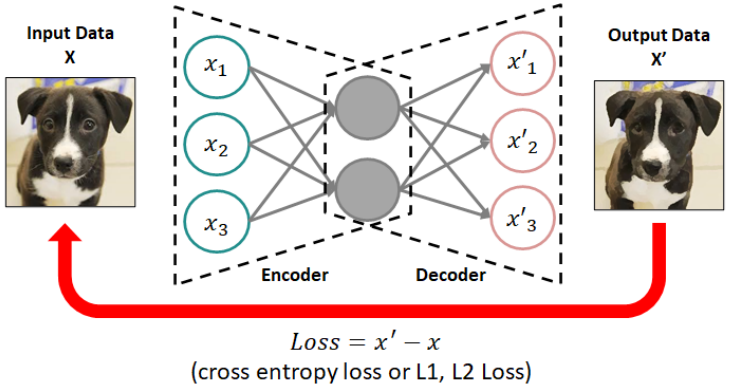
0. 오토인코더

- 레이블 없이 특징을 추출하는 신경망 / 비지도 학습
  - 이전까지 다뤘던 분류 모델들은 대다수 정답이 존재하는 데이터들로, 지도학습을 가능케 하는 문제들이다.
  - 하지만, 대다수 실세계의 수집 데이터는 특정 정답이 없으며, 단순히 주어진 데이터들을 구분할 수 있는 특정 패턴을 찾아야 한다.
  - 오토 인코더는 비지도학습을 진행하는 대표적인 네트워크 이다.



0. 오토인코더

오토 인코더 구조



- Encoder-Decoder 구조
  - 입력 받은 데이터를 압축하는 Encoder
  - 압축된 표현을 풀어 출력물을 생성하는 Decoder

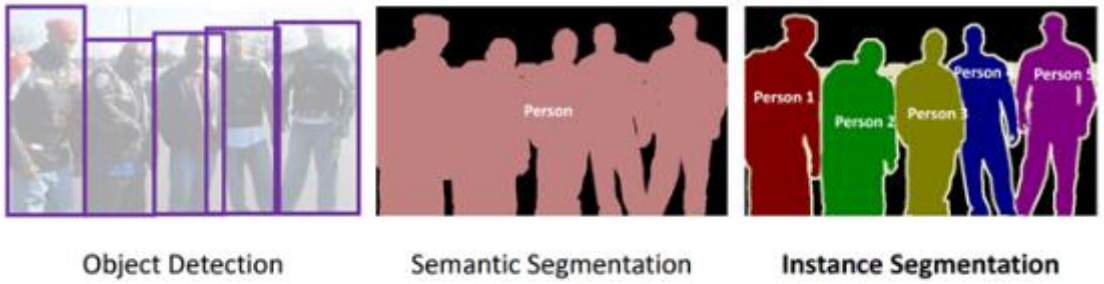
1) Input Data를 Encoder Network에 넣어, 압축된 특징 백터(회색 노드)를 생성
2) 압축된 벡터로부터 Decoder Network를 통해 Input과 유사한 Output Data를 생성

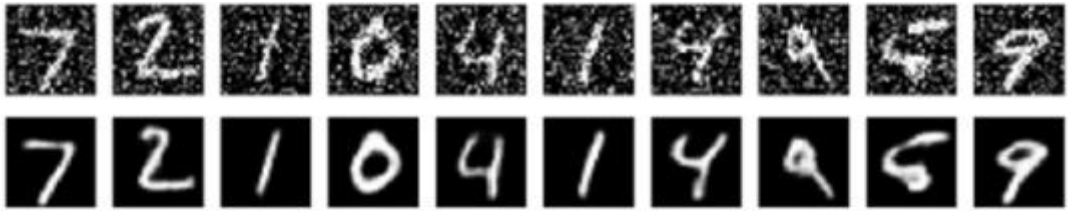
- 오토 인코더는 정답이 없는 x 데이터만으로 의미를 추출하기 위하여 x를 입력받아 다시 x를 복원할 수 있는 신경망을 학습하는 것이 핵심
- 오토 인코더는 입력, 출력의 크기는 같지만 중간으로 갈 수록 신경망의 차원이 줄어들면서, 입력 특징이 압축됨
- 압축된 데이터는 의미(Context, Semantic…)의 압축이 일어나게 되며 이는 중요한 정보만들 담는 압축이 일어나게 됨
- 즉 불필요한 정보에 대한 정보 손실이 일어나게 됨

0. 오토인코더

오토 인코더 구조

- Encoder-Decoder 특징
  - 오토 인코더는 주로 복잡한 비선형 데이터의 차원을 줄이는 용도로 쓰이며, 비정상적 거래 검출, 데이터 시각화-복원, 이미지 검색 등에 사용된다.
  - 이미지의 의미 있는 부분만 검출하는 Semantic Segmentation 과정,
  - 이미지의 노이즈를 제거하는 Noise Reduction 과정 에도 사용





0. 오토인코더

오토 인코더 실습

- 오토 인코더로 이미지의 특징 추출하기
- 이전 이미지 분류실습을 진행했던 FashionMNIST 데이터 셋을 활용해 AutoEncoder 모델을 실습해 보고자 함.

- 라이브러리 임포트, 하이퍼파라미터 설정

```
import torch
import torchvision
import torch.nn.functional as F
from torch import nn, optim
from torchvision import transforms, datasets

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import numpy as np
```

```
# 하이퍼파라미터
EPOCH = 10
BATCH_SIZE = 64
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
print("Using Device:", DEVICE)
```

```
Using Device: cpu
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더로 이미지의 특징 추출하기
- 이전 이미지 분류실습을 진행했던 FashionMNIST 데이터 셋을 활용해 AutoEncoder 모델을 실습해 보고자 함.

- Fashion MNIST 데이터 셋 불러오기
  – 이전에 실습했던 폴더와 같다면 추가 설치는 발생하지 않음

```
# Fashion MNIST 데이터셋
trainset = datasets.FashionMNIST(
    root      = './.data/',
    train     = True,
    download  = True,
    transform = transforms.ToTensor()
)
train_loader = torch.utils.data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE,
    shuffle     = True,
    num_workers = 2
)
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더 모듈 정의
- 784개의 차원으로 시작하여 3개의 차원으로 압축 후 다시 784개의 차원으로 복구

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 12),
            nn.ReLU(),
            nn.Linear(12, 3),   # 입력의 특징을 3차원으로 압축합니다
        )
        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(),
            nn.Linear(12, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid(),       # 픽셀당 0과 1 사이로 값을 출력합니다
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return encoded, decoded
```

0. 오토인코더

오토 인코더 실습

- 모델과 최적화 함수 객체 불러오기
- 최적화 함수는 Adam() 이며 SGD 의 변형 함수이다.
  - 학습 중인 기울기를 참고하여 학습 속도를 자동으로 변화시키는 특징이 있음.
- 원본 이미지 전처리
  - 픽셀의 색상 값은 0~255 이기 때문에 모델이 인식하는 0~1 사이의 값으로 전처리

```
autoencoder = Autoencoder().to(DEVICE)
optimizer = torch.optim.Adam(autoencoder.parameters(), lr=0.005)
criterion = nn.MSELoss()
```

```
# 원본 이미지를 시각화 하기 (첫번째 열)
view_data = trainset.data[:5].view(-1, 28*28)
view_data = view_data.type(torch.FloatTensor)/255.
```

0. 오토인코더

오토 인코더 실습

- 학습 코드

```python
def train(autoencoder, train_loader):
    autoencoder.train()
    for step, (x, label) in enumerate(train_loader):
        x = x.view(-1, 28*28).to(DEVICE)
        y = x.view(-1, 28*28).to(DEVICE)
        label = label.to(DEVICE)

        encoded, decoded = autoencoder(x)

        loss = criterion(decoded, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

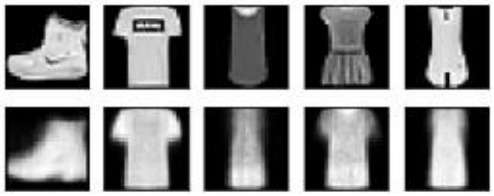
0. 오토인코더

오토 인코더 실습

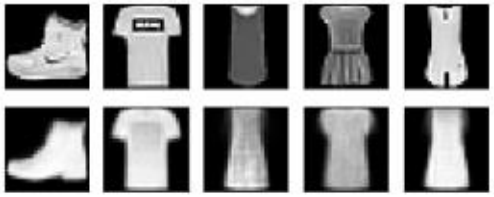
- 학습 코드
  - 학습 시 마다 원본 이미지와 복원 이미지 비교 / 위가 원본 아래가 복원 이미지

```
for epoch in range(1, EPOCH+1):
    train(autoencoder, train_loader)

    # 디코더에서 나온 이미지를 시각화 하기 (두번째 열)
    test_x = view_data.to(DEVICE)
    _, decoded_data = autoencoder(test_x)

    # 원본과 디코딩 결과 비교해보기
    f, a = plt.subplots(2, 5, figsize=(5, 2))
    print("[Epoch {}]".format(epoch))
    for i in range(5):
        img = np.reshape(view_data.data.numpy()[i],(28, 28))
        a[0][i].imshow(img, cmap='gray')
        a[0][i].set_xticks(()); a[0][i].set_yticks(())

    for i in range(5):
        img = np.reshape(decoded_data.to("cpu").data.numpy()[i], (28, 28))
        a[1][i].imshow(img, cmap='gray')
        a[1][i].set_xticks(()); a[1][i].set_yticks(())
    plt.show()
```

[Epoch 1]



[Epoch 10]



0. 오토인코더

오토 인코더 실습

- 특징 벡터 들여다 보기
  - 학습한 오토 인코더의 중간 부분인 특징 벡터들이 3차원에서 어떻게 분포 되는지 확인

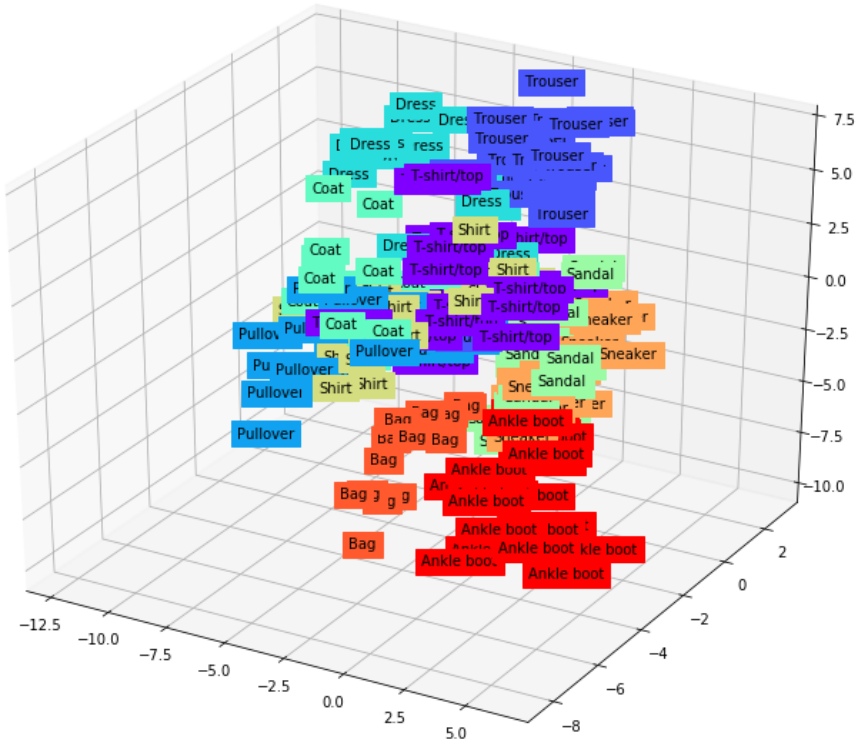
```python
# 잠재변수를 3D 플롯으로 시각화
view_data = trainset.data[:200].view(-1, 28*28)
view_data = view_data.type(torch.FloatTensor)/255.
test_x = view_data.to(DEVICE)
encoded_data, _ = autoencoder(test_x)
encoded_data = encoded_data.to("cpu")
```

```python
CLASSES = {
    0: 'T-shirt/top',
    1: 'Trouser',
    2: 'Pullover',
    3: 'Dress',
    4: 'Coat',
    5: 'Sandal',
    6: 'Shirt',
    7: 'Sneaker',
    8: 'Bag',
    9: 'Ankle boot'
}

fig = plt.figure(figsize=(10,8))
ax = Axes3D(fig)

X = encoded_data.data[:, 0].numpy()
Y = encoded_data.data[:, 1].numpy()
Z = encoded_data.data[:, 2].numpy()

labels = trainset.targets[:200].numpy()

for x, y, z, s in zip(X, Y, Z, labels):
    name = CLASSES[s]
    color = cm.rainbow(int(255*s/9))
    ax.text(x, y, z, name, backgroundcolor=color)

ax.set_xlim(X.min(), X.max())
ax.set_ylim(Y.min(), Y.max())
ax.set_zlim(Z.min(), Z.max())
plt.show()
```

0. 오토인코더

오토 인코더 실습

- 특징 벡터 들여다 보기
  - 학습한 오토 인코더의 중간 부분인 특징 벡터들이 3차원에서 어떻게 분포 되는지 확인
  - 같은 레이블을 가진 이미지의 잠재 변수는 서로 모여 있는 것을 확인할 수 있음
  - 또한 비슷한 특징을 가지는 이미지 또한 모여 있는 것을 확인 할 수 있다.



0. 오토인코더

오토 인코더 실습

- 오토 인코더로 망가진 이미지 복원
- 이전실습과 빨간 박스를 제외하고 코드 동일

```
import torch
import torchvision
import torch.nn.functional as F
from torch import nn, optim
from torchvision import transforms, datasets

import matplotlib.pyplot as plt
import numpy as np
```

```
# 하이퍼파라미터
EPOCH = 50
BATCH_SIZE = 64
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
print("다음 기기로 학습합니다:", DEVICE)
```

```
다음 기기로 학습합니다: cpu
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더로 망가진 이미지 복원
- 이전 실습과 코드 동일

```
# Fashion MNIST 학습 데이터셋
trainset = datasets.FashionMNIST(
    root      = './.data/',
    train     = True,
    download  = True,
    transform = transforms.ToTensor()
)

train_loader = torch.utils.data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE,
    shuffle     = True,
    num_workers = 2
)
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더로 망가진 이미지 복원 / 이전 실습과 코드 동일

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

        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 12),
            nn.ReLU(),
            nn.Linear(12, 3),   # 입력의 특징을 3차원으로 압축합니다
        )
        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(),
            nn.Linear(12, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid(),       # 픽셀당 0과 1 사이로 값을 출력합니다
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return encoded, decoded
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더로 망가진 이미지 복원 / 빨간 박스를 제외하고 이전 실습과 코드 동일

```
autoencoder = Autoencoder().to(DEVICE)
optimizer = torch.optim.Adam(autoencoder.parameters(), lr=0.005)
criterion = nn.MSELoss()
```

```
def add_noise(img):
    noise = torch.randn(img.size()) * 0.2
    noisy_img = img + noise
    return noisy_img
```

```
def train(autoencoder, train_loader):
    autoencoder.train()
    avg_loss = 0
    for step, (x, label) in enumerate(train_loader):
        noisy_x = add_noise(x)  # 입력에 노이즈 더하기
        noisy_x = noisy_x.view(-1, 28*28).to(DEVICE)
        y = x.view(-1, 28*28).to(DEVICE)

        label = label.to(DEVICE)
        encoded, decoded = autoencoder(noisy_x)

        loss = criterion(decoded, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        avg_loss += loss.item()
    return avg_loss / len(train_loader)
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더로 망가진 이미지 복원

```
for epoch in range(1, EPOCH+1):
    loss = train(autoencoder, train_loader)
    print("[Epoch {}] loss:{}".format(epoch, loss))
    # 이번 예제에선 학습시 시각화를 건너 뜁니다
```

```
[Epoch 1] loss:0.038064478491462 75
[Epoch 2] loss:0.026071368966863226
[Epoch 3] loss:0.02439545210538 2614
[Epoch 4] loss:0.0235740480356 7286
[Epoch 5] loss:0.023224251918685334
[Epoch 6] loss:0.022898294619405703
[Epoch 7] loss:0.022739627257561 38
[Epoch 8] loss:0.022620197435193605
[Epoch 9] loss:0.022435167879819362
[Epoch 10] loss:0.022402050622951374
[Epoch 11] loss:0.022248590462354 58
[Epoch 12] loss:0.022100204423165272
[Epoch 13] loss:0.022072431424866 99
[Epoch 14] loss:0.022115336101589552
[Epoch 15] loss:0.021984517828487 89
[Epoch 16] loss:0.021982974323160105
[Epoch 17] loss:0.021864540557236 05
[Epoch 18] loss:0.021900656913071553
[Epoch 19] loss:0.021924663408558125
[Epoch 20] loss:0.021700548734873343
[Epoch 30] loss:0.021727535553943755
[Epoch 31] loss:0.021496980506648767
[Epoch 32] loss:0.021531763635496342
[Epoch 33] loss:0.02156935673135557
[Epoch 34] loss:0.021465798542062358
[Epoch 35] loss:0.022280702984997076
[Epoch 36] loss:0.022223254445312757
[Epoch 37] loss:0.021664978042721494
[Epoch 38] loss:0.021690005940923303
[Epoch 39] loss:0.02155636892374009
[Epoch 40] loss:0.021717974695680875
[Epoch 41] loss:0.021460372227817964
[Epoch 42] loss:0.02161121419045145
[Epoch 43] loss:0.02193874641577763
[Epoch 44] loss:0.021835189999770254
[Epoch 45] loss:0.022764335392412346
[Epoch 46] loss:0.022956009696064983
[Epoch 47] loss:0.021970841078472924
[Epoch 48] loss:0.021771560466762926
[Epoch 49] loss:0.02163266537111324
[Epoch 50] loss:0.021663608033456273
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더로 망가진 이미지 복원

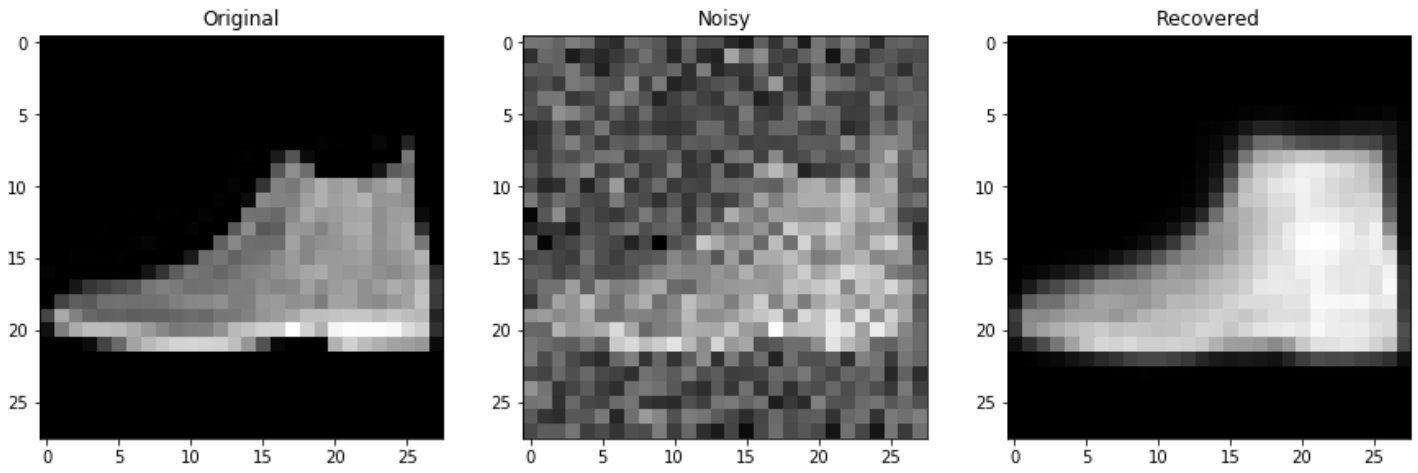
```
# 모델이 학습시 본적이 없는 데이터로 검증하기 위해 테스트 데이터셋을 가져옵니다.
testset = datasets.FashionMNIST(
    root      = './.data/',
    train     = False,
    download  = True,
    transform = transforms.ToTensor()
)

# 테스트셋에서 이미지 한장을 가져옵니다.
sample_data = testset.data[0].view(-1, 28*28)
sample_data = sample_data.type(torch.FloatTensor)/255.

# 이미지를 add_noise로 오염시킨 후, 모델에 통과시킵니다.
original_x = sample_data[0]
noisy_x = add_noise(original_x).to(DEVICE)
_, recovered_x = autoencoder(noisy_x)
```

0. 오토인코더

오토 인코더 실습

- 오토 인코더로 망가진 이미지 복원

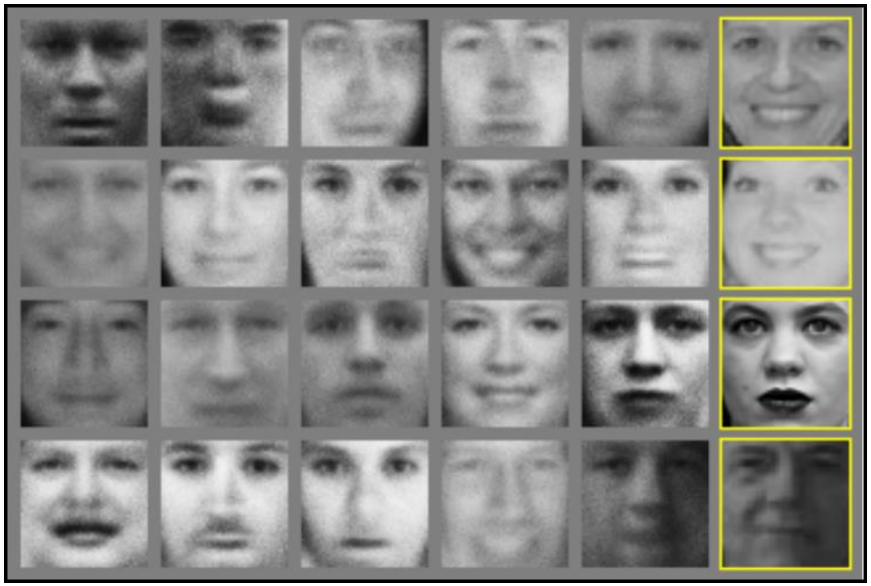
```python
f, a = plt.subplots(1, 3, figsize=(15, 15))

# 시각화를 위해 넘파이 행렬로 바꿔줍니다.
original_img = np.reshape(original_x.to("cpu").data.numpy(), (28, 28))
noisy_img = np.reshape(noisy_x.to("cpu").data.numpy(), (28, 28))
recovered_img = np.reshape(recovered_x.to("cpu").data.numpy(), (28, 28))

# 원본 사진
a[0].set_title('Original')
a[0].imshow(original_img, cmap='gray')

# 오염된 원본 사진
a[1].set_title('Noisy')
a[1].imshow(noisy_img, cmap='gray')

# 복원된 사진
a[2].set_title('Recovered')
a[2].imshow(recovered_img, cmap='gray')

plt.show()
```



1. GAN 예시

초기 GAN

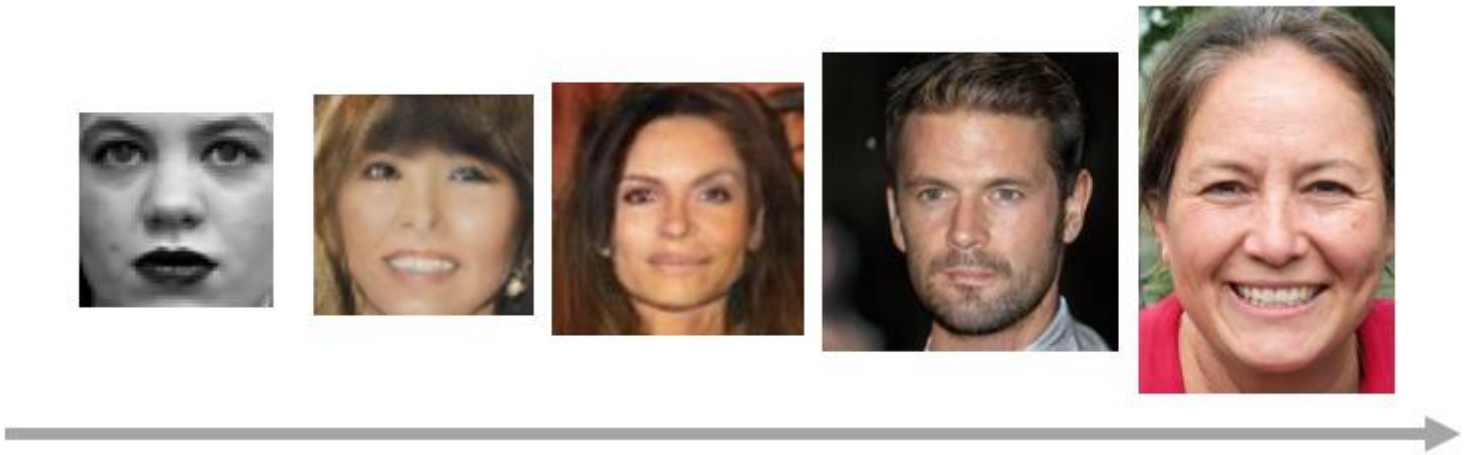
- 데이터를 학습하여 노란색 박스와 같은 이미지를 생성
- 초기 결과물이다 보니 색채가 없고 화질이 좋지 않음



1. GAN 예시

GAN 발전과정

- GAN 모델의 단점을 극복하는 다양한 연구들이 진행됨에 따라 아래와 같이 진짜 이미지와 구분하기 힘들 정도로 발전하게 됨



GAN 모델 성능 발전 과정

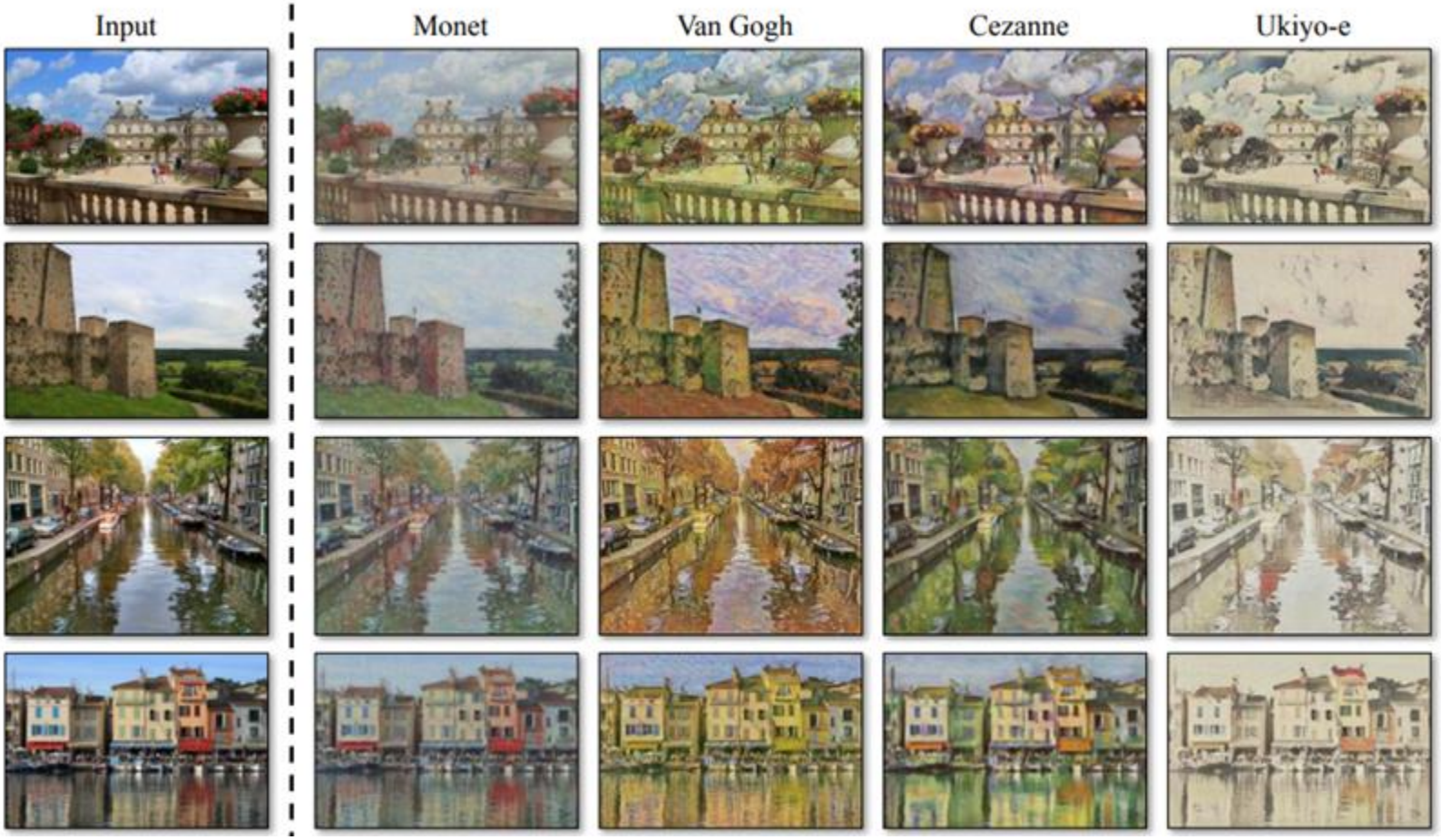
1. GAN 예시

GAN 예시

1. GAN 예시 / 화질 개선

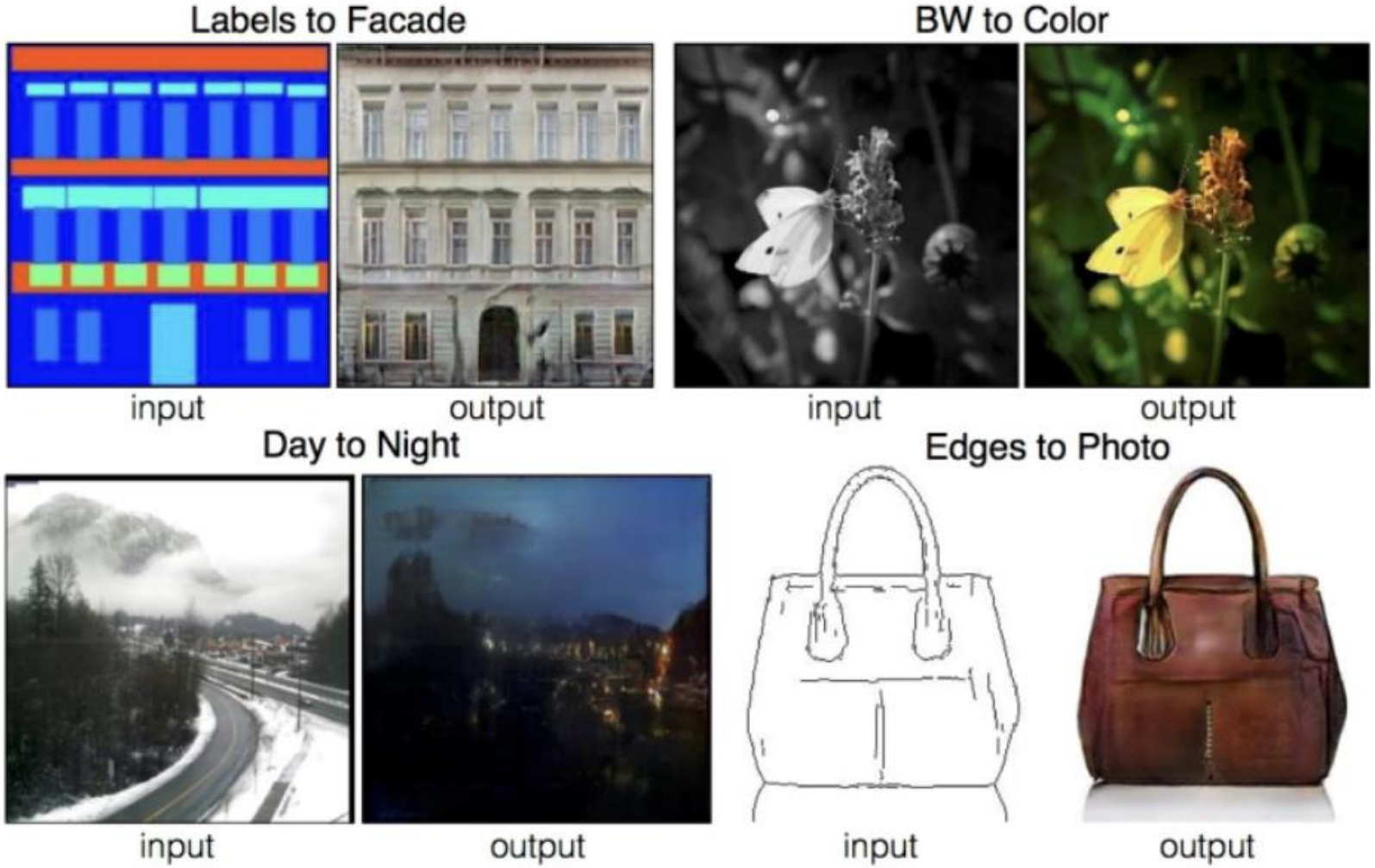
1. GAN 예시 / 사진을 특정 화가의 화풍으로 그리기

GAN 예시



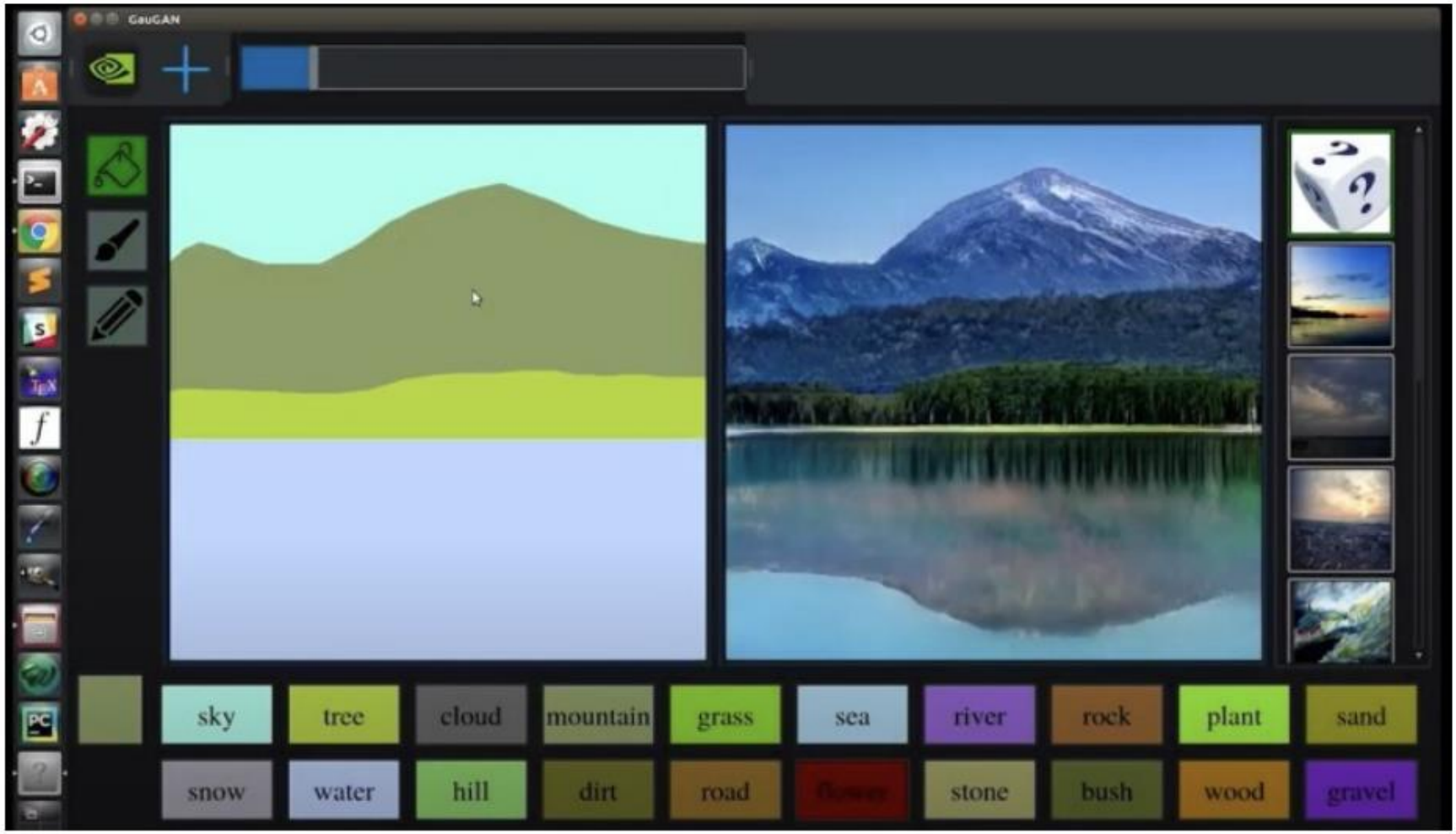
1. GAN 예시 / 도형 건물 / 흑백 컬러 / 낮 밤 / 스케치 색 부여

GAN 예시



1. GAN 예시 / 단순한 이미지를 실제 사진처럼 변환

GAN 예시



< © Nvidia >

2. GAN 개념

이전 모델

Cat

입력 데이터 판별 결과

- 이전에 배운 모델은 데이터를 특정한 클래스로 판별 하는 모델

2. GAN 개념

이전 모델

Cat

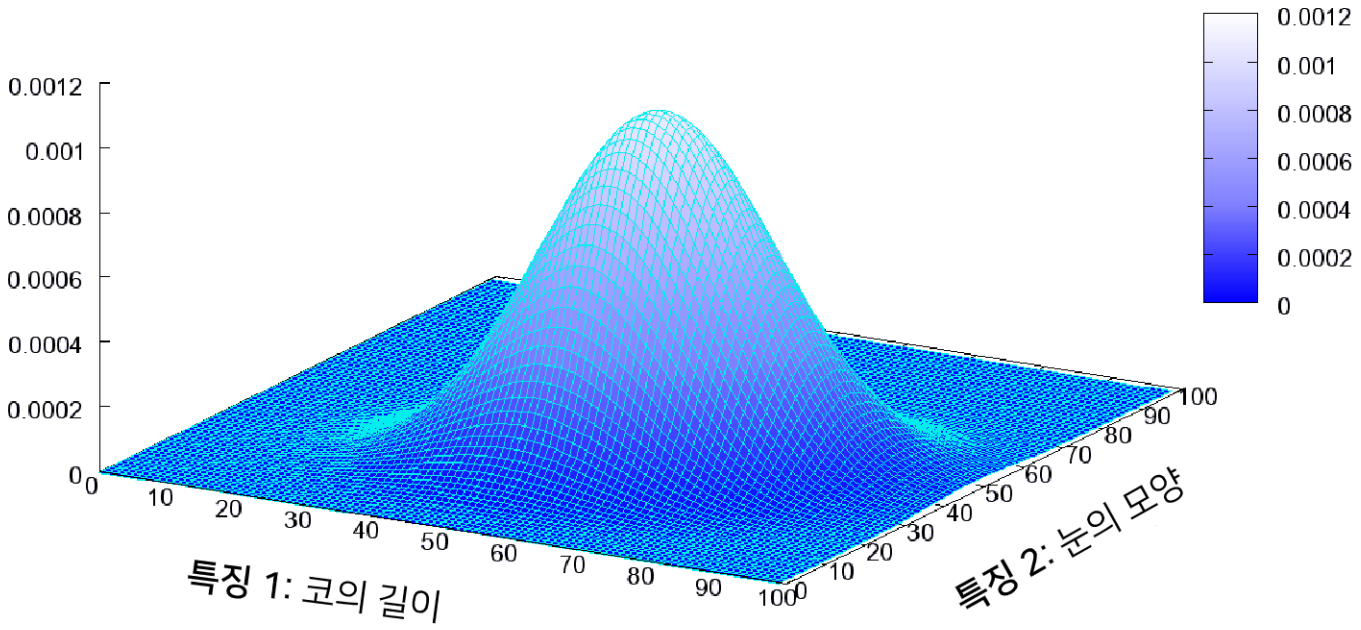
입력 데이터
x

판별 결과
y

- 이러한 모델은 입력 데이터(x)에 대하여 결과(y)가 될 확률, 즉 조건부 확률 $P(y|x)$ 을 구하는 것이 된다.
- 이 말은 같은 결과(y)를 출력하는 데이터(x)는 특정한 확률 분포를 가지고 있게 된다.

2. GAN 개념

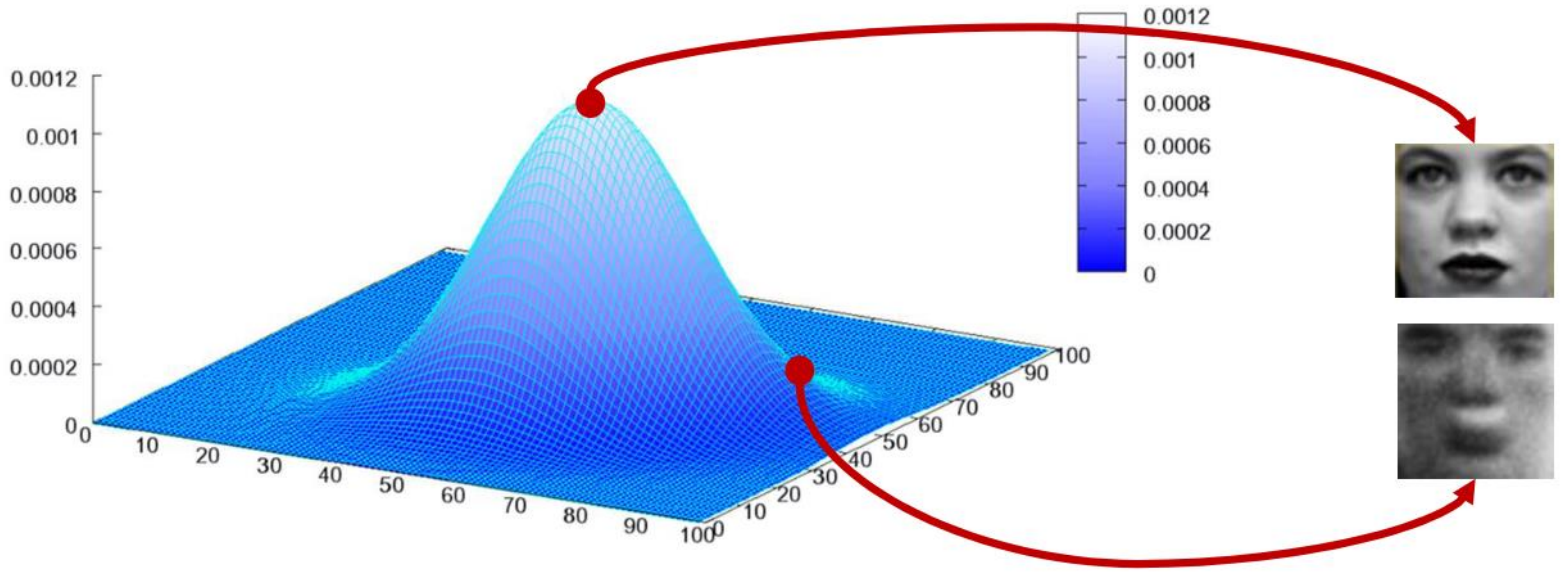
생성모델



- 특정한 확률 분포를 이용해 유사한 확률 분포를 갖는 데이터를 생성할 수 있게 된다.
- 추가로 인위적으로 특정한 확률 분포를 합치거나 제거해 줌으로써 이전에 없던 새로운 데이터를 생성할 수 있다.

2. GAN 개념

생성모델

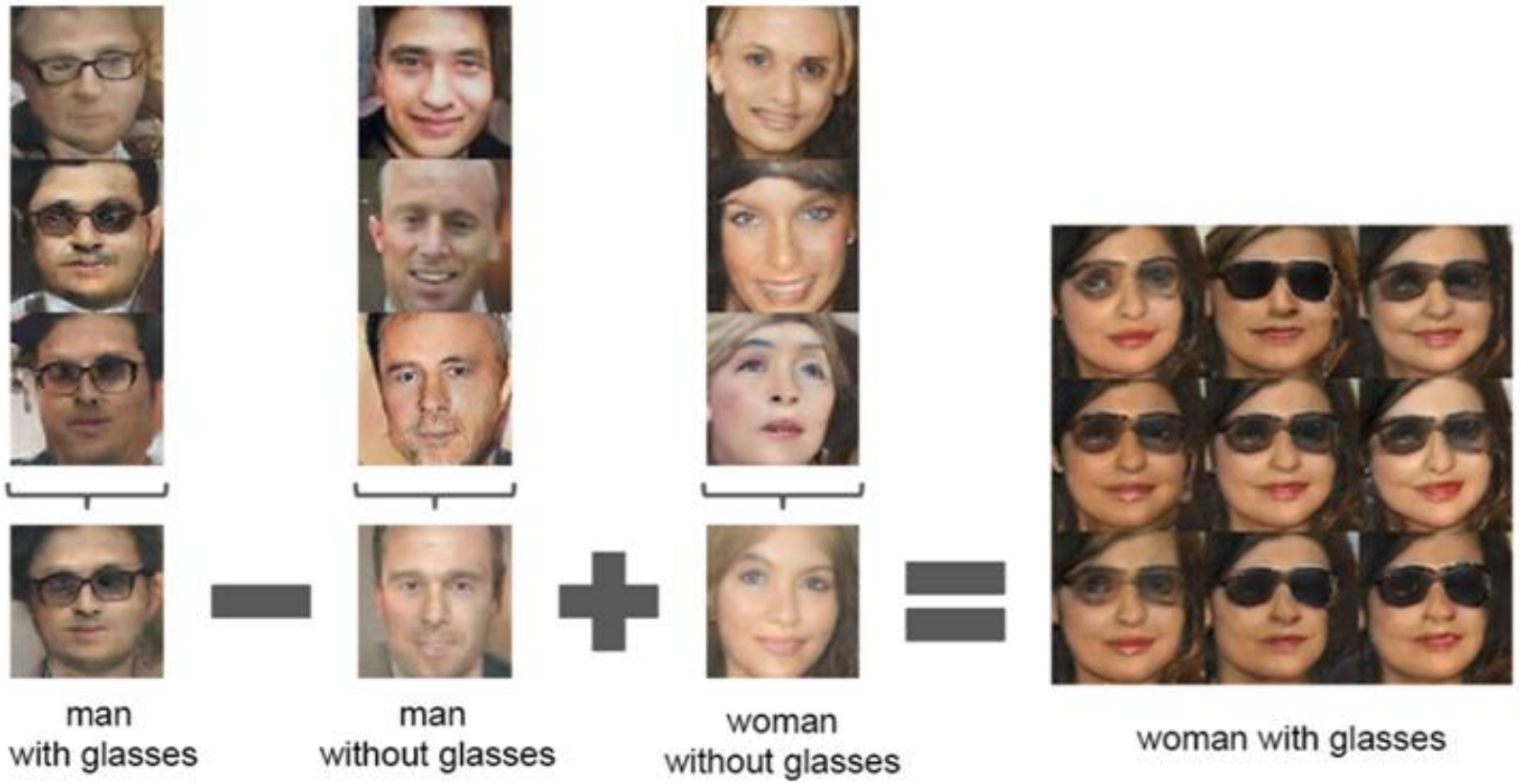


- 즉 생성 모델은 실존하지 않지만 있을 법한 이미지를 생성할 수 있는 모델을 의미한다.

2. GAN 개념

생성모델

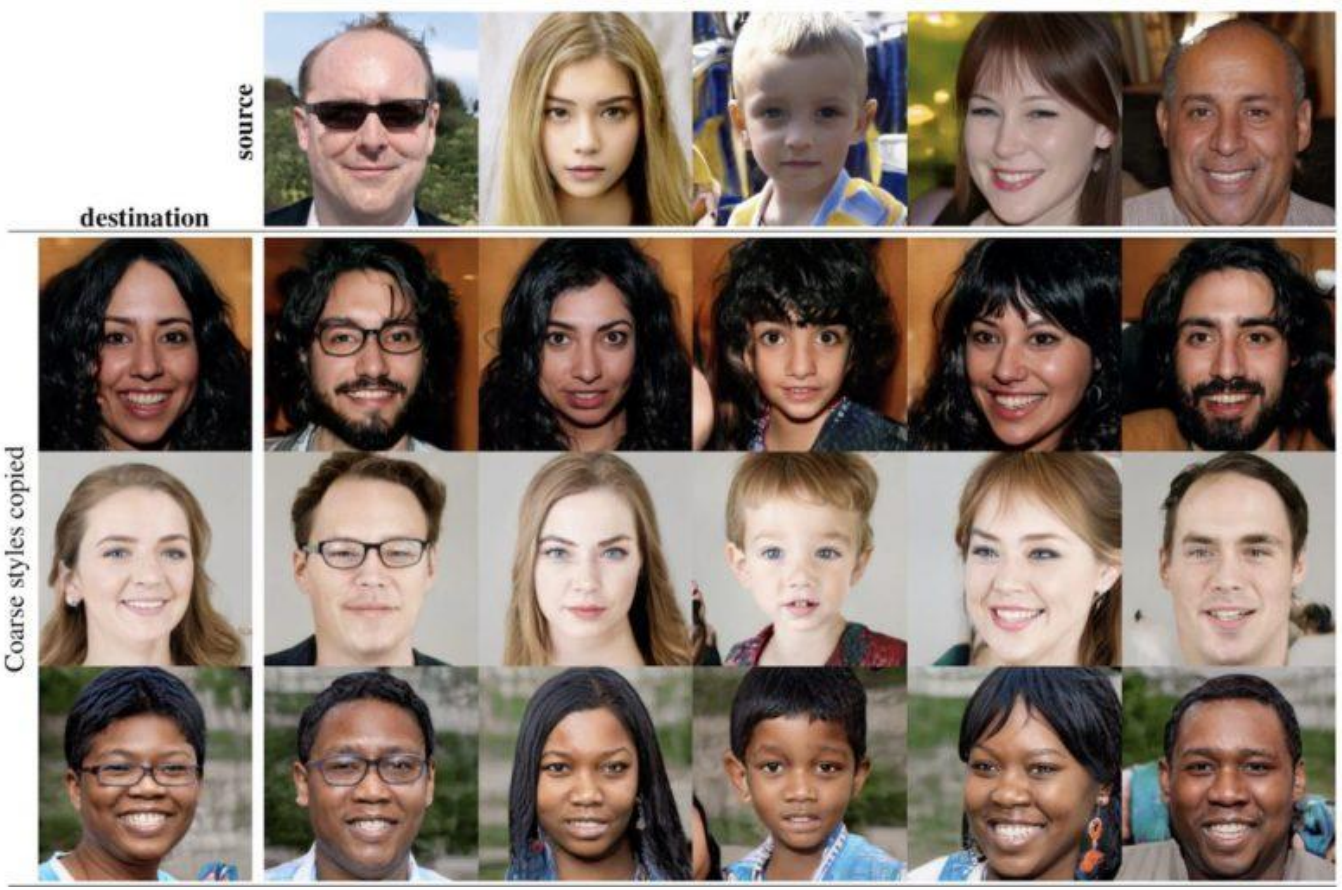
- 이미지의 특징을 추출하여 벡터 산술 연산이 가능하게 됨



2. GAN 개념

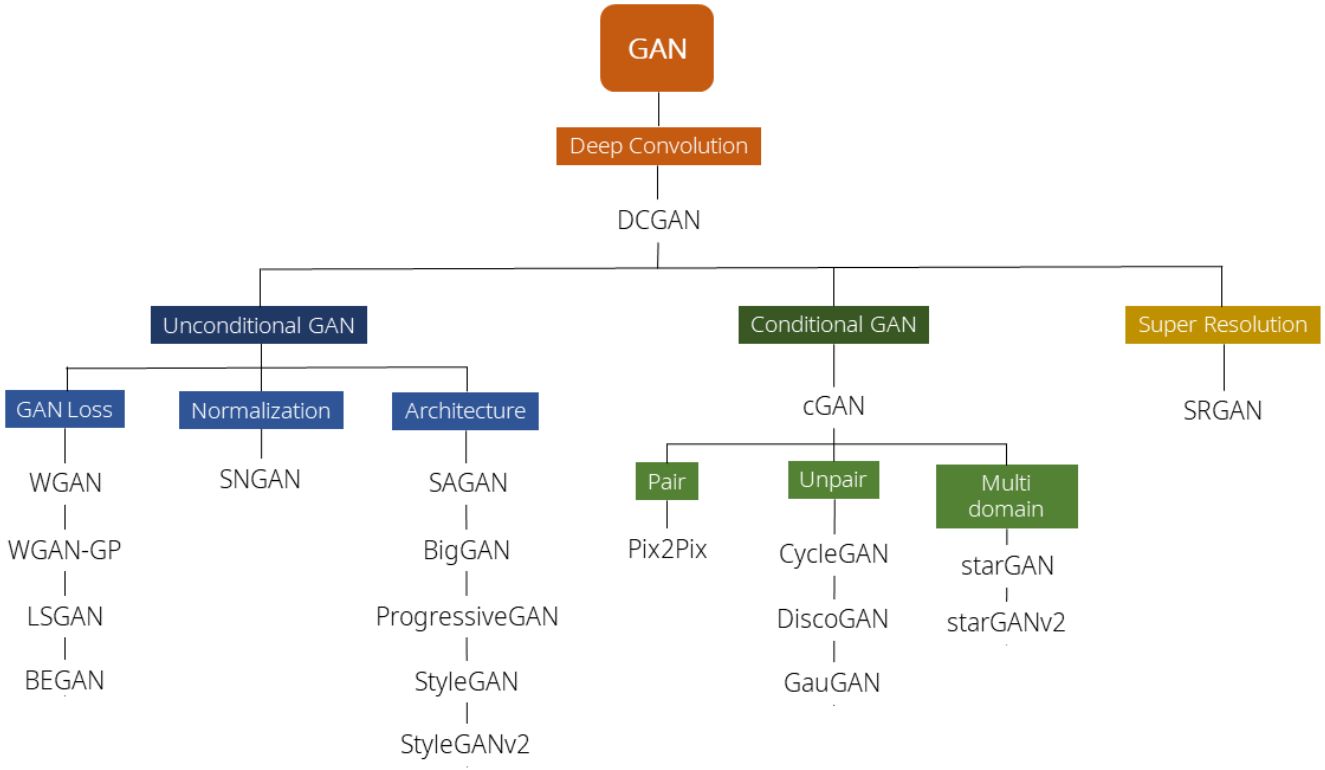
생성모델

- 이미지의 특징을 추출하여 벡터 산술 연산이 가능하게 됨



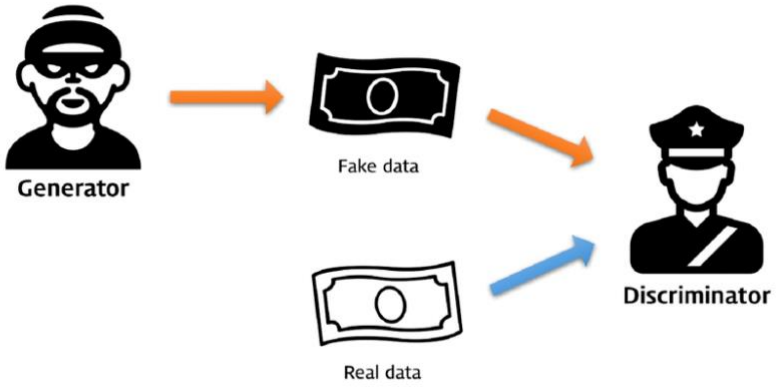
2. GAN 개념

GAN 파생 모델



2. GAN 개념

핵심 아이디어

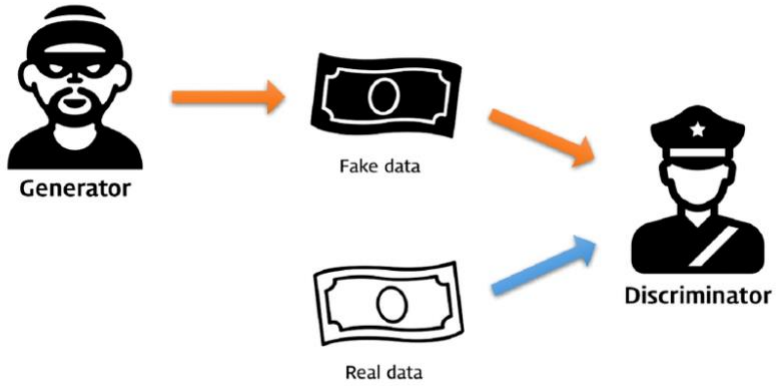


- 위조 지폐범은 최대한 진짜 같은 화폐를 만드는 것이 목적 . (생성)
- 경찰은 진짜 화폐와 가짜 화폐를 완벽히 판별(분류) 하여 위조 지폐범을 검거하는 것이 목적.

- 위와 같은 경쟁 학습이 지속되면 어느 순간 위조지폐범을 진짜와 다를 바 없는 위조지폐를 만들 수 있게 되고 경찰은 위조지폐를 분류할 확률이 50%로 수렴하게 된다.
- 즉 경찰은 위조지폐와 실제 지폐를 구분할 수 없는 상태에 이르게 된다.

2. GAN 개념

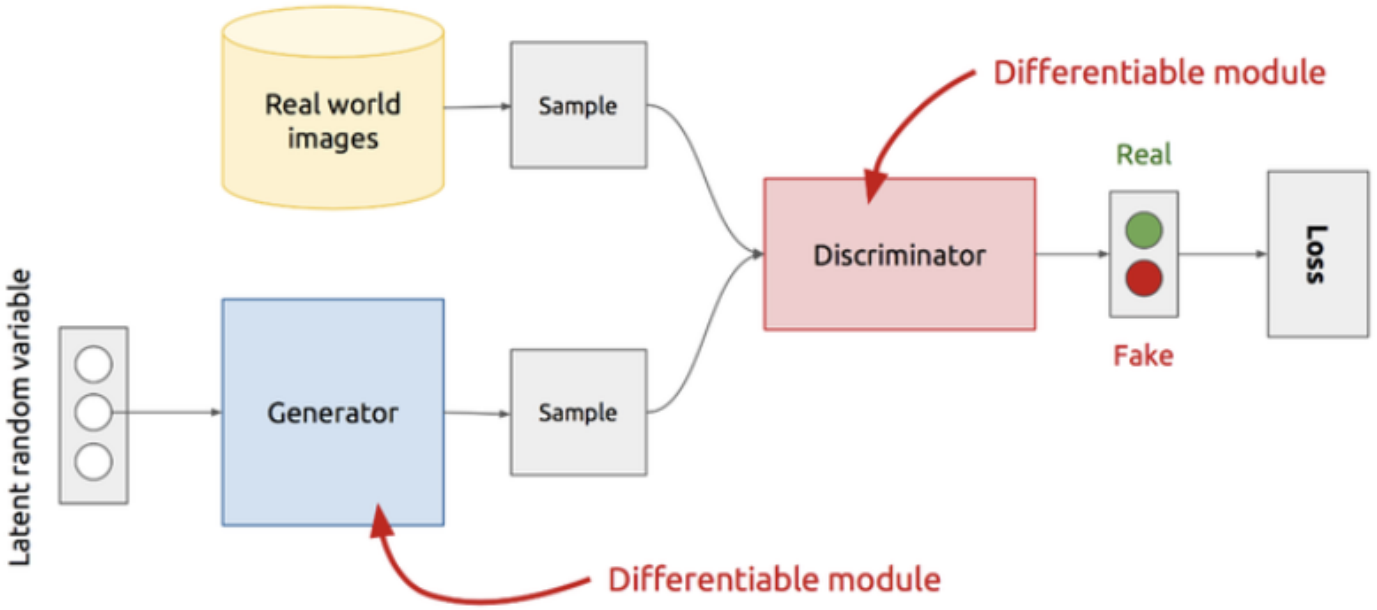
핵심 아이디어



- 여기서 경찰은 분류 모델 Discriminator 위조지폐범은 생성 모델 Generator를 의미한다.
- GAN 에서는 진짜 같은 데이터를 생성하려는 생성 모델과 진짜와 가짜를 판별하려는 분류 모델이 각각 존재하여 서로 적대적으로 학습한다.

2. GAN 개념

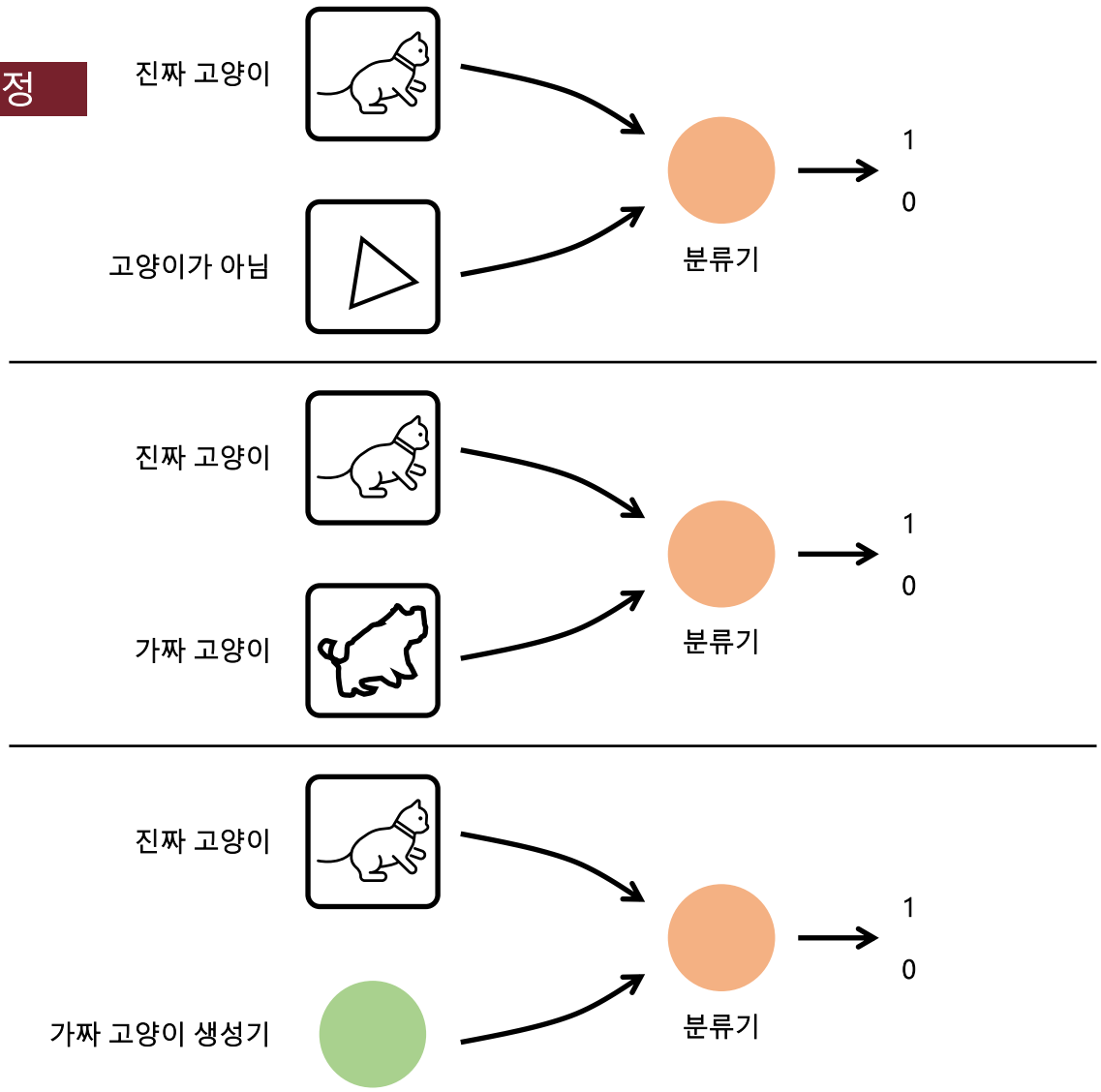
모델 도식화



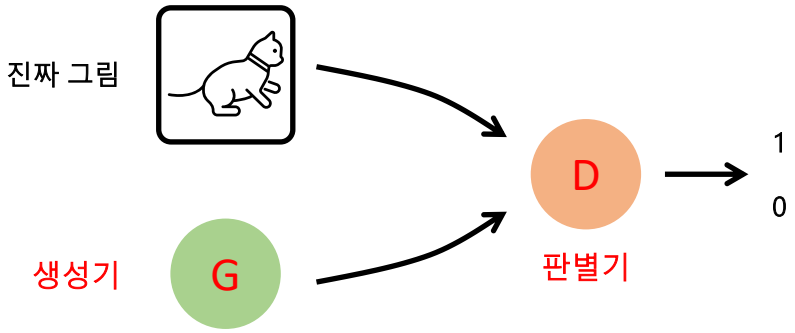
- 적대적 학습(GAN) 에서는 분류 모델을 먼저 학습시킨 후, 생성 모델을 학습시키는 과정을 서로 주고받으면서 반복한다.

2. GAN 개념

GAN 학습 과정



2. GAN 개념



훈련하는 과정은 어떻게 보상을 주고 어떻게 벌을 줄지에 대한 과정 (손실함수)

- 판별기를 무사히 속이면 생성기에 보상을 줍니다.
- 판별기에 잡히면 생성기에 벌을 줍니다.

판별기의 역할은 가짜로 생성된 이미지와 진짜 이미지를 구분하는 것
만약 생성기가 그다지 성능이 좋지 않다면, 이 일은 굉장히 쉬울 것임
하지만 생성기를 계속 훈련시키면, 점차 진짜와 구분하기 어려운 이미지를 만들어낼 것임

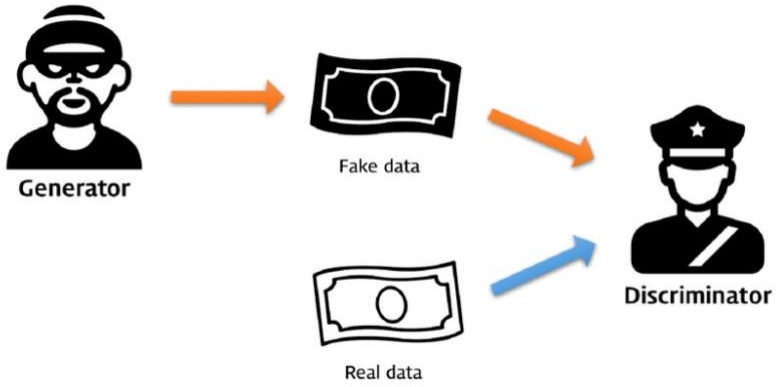
2. GAN 개념

판별기가 훈련을 거치며 점점 성능이 좋아질수록, 생성기 또한 보상과 벌을 통해 훈련이 되어 성능이 증가할 것임

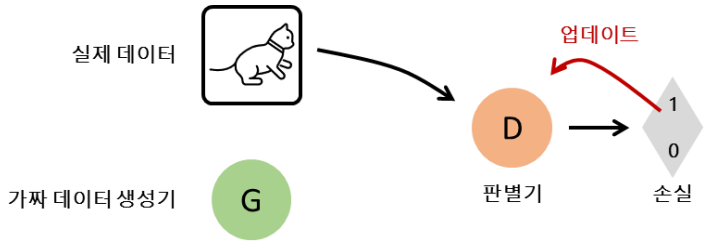
궁극적으로 생성기는 진짜 이미지와 분간이 가지 않는 이미지들을 만들기 시작할 것임

판별기와 생성기는 서로 적대적 관계로 경쟁을 하게 되며,
서로를 뛰어넘으려고 노력하기 때문에 결과적으로는 둘 다 성능이 좋아지게 됨

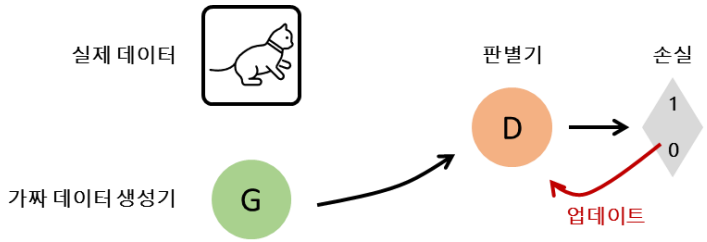
이러한 구조가 바로 생성적 적대 신경망 generative adversarial network



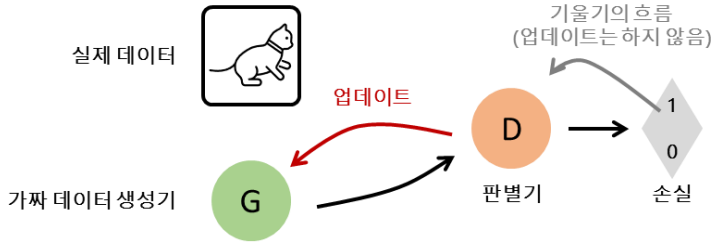
2. GAN 개념 – 훈련 방법



1단계 : 판별기에 실제 데이터를 보여주고 1 이라는 값이어야 한다고 알려준다.



2단계 : 판별기에 생성기로부터 만들어진 가짜 데이터를 보여주고 0이어야 한다고 알려준다.

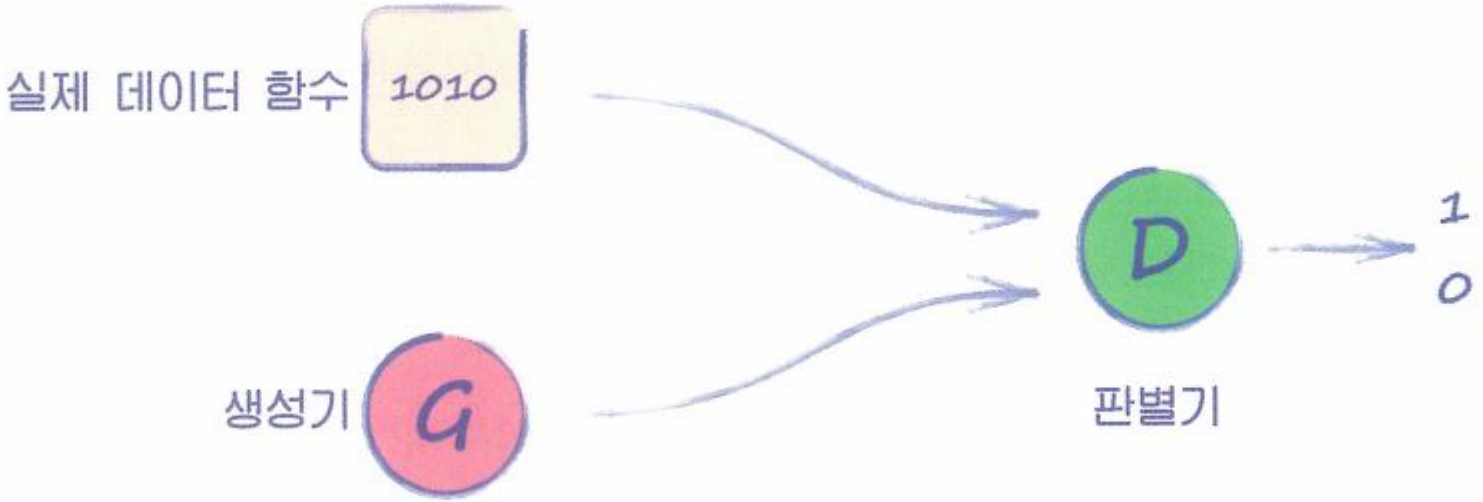


3단계 : 판별기에 생성기의 결과를 보여주고, 생성기에 결과가 1이어야 한다고 알려준다.

3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- GAN 코드가 일반적으로 어떻게 생겼는지, 그리고 GAN이 어떻게 학습하는지 확인하는 것에 초점을 맞추고자 함.



3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 학습에 필요한 라이브러리 임포트

```python
# 라이브러리 임포트
import torch
import torch.nn as nn
import pandas
import matplotlib.pyplot as plt
import random
import numpy
```

- Generate_real() // 실제 데이터 생성

- 실제 데이터를 1010으로 생성하는 것이 아닌
- 실제 세계 데이터처럼 약간의 오차를 부여

- Generate_random(size)

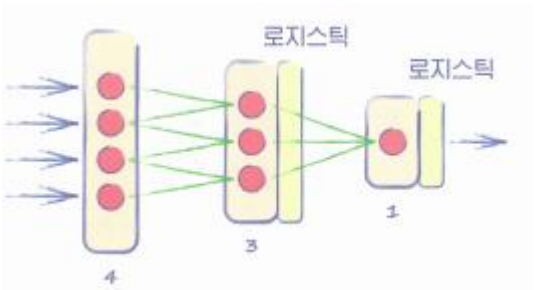
- size의 크기를 갖는 0~1 랜덤 데이터 생성

```python
# 실제 데이터를 생성하기 위한 코드

def generate_real():
    real_data = torch.FloatTensor(
        [random.uniform(0.8, 1.0),
         random.uniform(0.0, 0.2),
         random.uniform(0.8, 1.0),
         random.uniform(0.0, 0.2)])
    return real_data

# 동일한 임의의 데이터를 생성하기 위한 코드

def generate_random(size):
    random_data = torch.rand(size)
    return random_data
```

3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 판별기 클래스 생성
  - 4개의 입력을 받아
  - 참인 경우 1출력
  - 거짓인 경우 0 출력




```python
class Discriminator(nn.Module):
    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()
        # 신경망 레이어 정의
        self.model = nn.Sequential(
            nn.Linear(4, 3),
            nn.Sigmoid(),
            nn.Linear(3, 1),
            nn.Sigmoid()
        )
        # 손실함수 생성
        self.loss_function = nn.MSELoss()
        # SGD 옵티마이저 생성
        self.optimiser = torch.optim.SGD(
            self.parameters(), lr=0.01)
        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []
    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs)
    def train(self, inputs, targets):
        # 신경망 출력 계산
        outputs = self.forward(inputs)
        # 손실 계산
        loss = self.loss_function(outputs, targets)
        # 매 10회마다 에러를 누적하고 카운터를 증가
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
            pass
        if (self.counter % 10000 == 0):
            print("counter = ", self.counter)
            pass
        # 기울기를 초기화 하고 역전파 후 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()
    def plot_progress(self):
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0, 1.0), figsize=(16,8), alpha=0.1,
                marker='.', grid=True, yticks=(0, 0.25, 0.5))
```

3. GAN 실습

GAN 실습 1

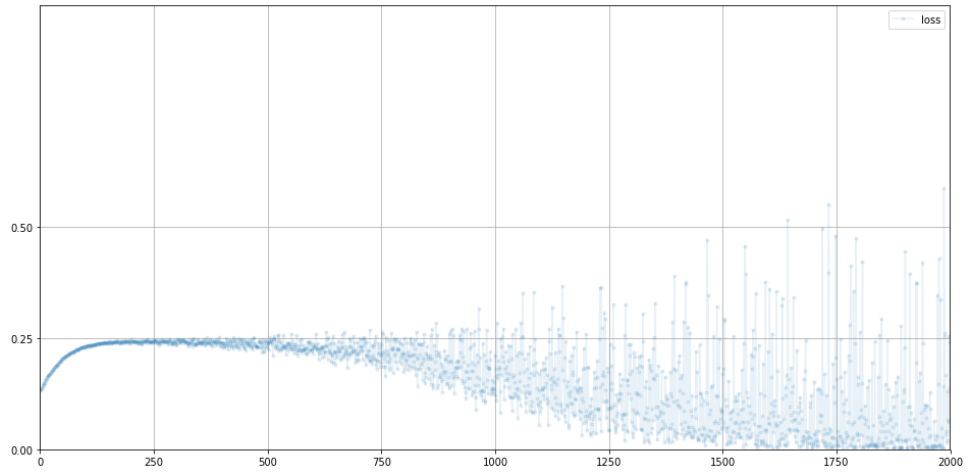
- GAN을 이용해 1010패턴 생성
- 판별기 테스트

- 실제 데이터와 가짜 데이터를
- 사용하여 판별기 테스트

```
# 판별기가 임의의 노이즈로부터 실제 데이터를 구별할수 있는지 확인
D = Discriminator()
for i in range(10000):
    # 실제 데이터
    D.train(generate_real(), torch.FloatTensor([1.0]))
    # 생성된 데이터
    D.train(generate_random(4), torch.FloatTensor([0.0]))
# 판별기 손실 플롯
D.plot_progress()
```

```
counter =  10000
counter =  20000
```

- 판별기의 오차가 0.25에서 시작해
- 0으로 수렴한다.



3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 판별기 테스트
  - 판별기에 직접 값을 넣어 확인해 보기

```
1  # 가짜와 진짜를 판별할수 있는지 판별기 직접 구동
2
3  print( D.forward( generate_real() ).item() )
4  print( D.forward( generate_random(4) ).item() )
```

```
0.8129775524139404
0.3490223288536072
```

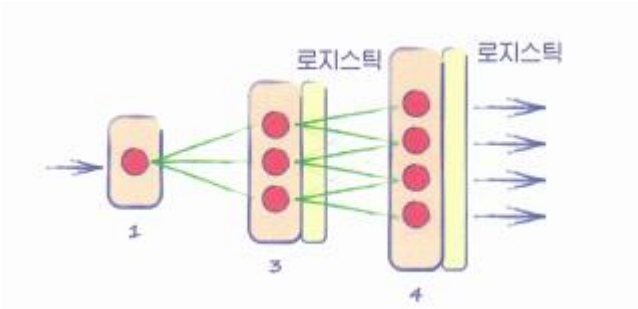
- 판별기에 1010 패턴에 맞는 값을 넣을 경우 결과는 1.0
- 완전 임의의 패턴을 넣을 경우 결과는 0.0에 가깝게 출력될 것이다.

- 실제 데이터를 넣은 경우 0.81
- 가짜 데이터를 넣은 경우 0.34

3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 생성기 만들기
- 판별기로부터 흘러온 기울기 오차를
- 통해 업데이트가 진행됨




```python
class Generator(nn.Module):
    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()
        # 신경망 레이어 정의
        self.model = nn.Sequential(
            nn.Linear(1, 3),
            nn.Sigmoid(),
            nn.Linear(3, 4),
            nn.Sigmoid()
        )
        # SGD 옵티마이저 생성
        self.optimiser = torch.optim.SGD(
            self.parameters(), lr=0.01)
        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []
    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs)
    def train(self, D, inputs, targets):
        # 신경망 출력 계산
        g_output = self.forward(inputs)
        # 판별기에 값 전달
        d_output = D.forward(g_output)
        # 오차 계산
        loss = D.loss_function(d_output, targets)
        # 매 10회마다 에러를 누적하고 카운터를 증가
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
        # 기울기를 초기화 하고 역전파 후 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()
    def plot_progress(self):
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0, 1.0), figsize=(16,8), alpha=0.1,
                marker='.', grid=True, yticks=(0, 0.25, 0.5))
```

3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 생성기 결과 확인하기

```
1  # 생성기의 출력이 올바른 타입과 형태를 지니고 있는지 확인
2  G = Generator()
3  G.forward(torch.FloatTensor([0.5]))
```

```
tensor([0.5336, 0.5202, 0.4646, 0.5578], grad_fn=<SigmoidBackward>)
```

- 정상적으로 4개의 텐서를 반환하는 것을 확인할 수 있다.
- 1010 패턴이 아닌 이유는 학습이 진행되지 않았기 때문!

3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- GAN 학습 / %%time – 학습에 걸린 시간 측정

```
%%time
# 판별기 및 생성기 생성
D = Discriminator()
G = Generator()
image_list = []
# 판별기와 생성기 훈련
for i in range(10000):
    # 참일 경우 판별기 훈련
    D.train(generate_real(), torch.FloatTensor([1.0]))
    # 거짓일 경우 판별기 훈련
    # G의 기울기가 계산되지 않도록 detach() 함수를 이용
    D.train(G.forward(torch.FloatTensor([0.5])).detach(), torch.FloatTensor([0.0]))
    # 생성기 훈련
    G.train(D, torch.FloatTensor([0.5]), torch.FloatTensor([1.0]))
    # 매 1000 회마다 이미지 추가
    if (i % 1000 == 0):
        image_list.append( G.forward(torch.FloatTensor([0.5])).detach().numpy() )

```

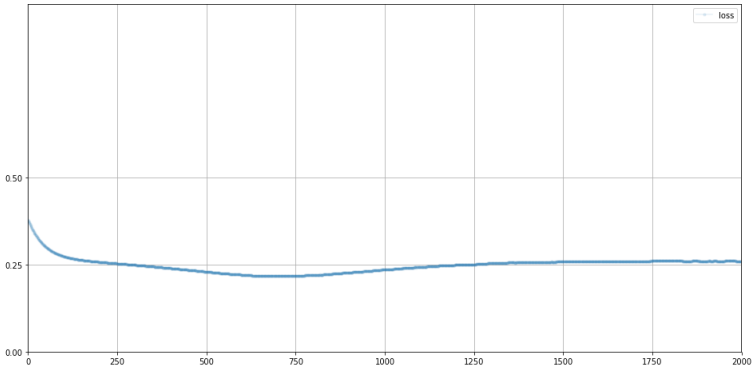
```
counter =  10000
counter =  20000
Wall time: 11.6 s
```

3. GAN 실습

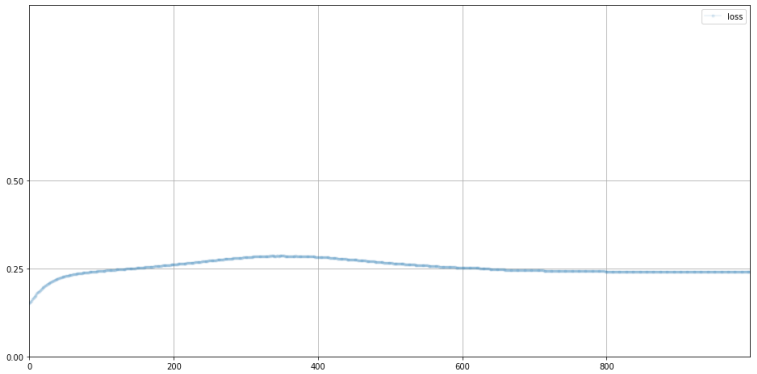
GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 훈련 결과 출력

```
1  # 판별기 오차 플롯
2
3  D.plot_progress()
```




```
1  # 생성기 손실 출력
2
3  G.plot_progress()
```



- 0.25 로 수렴하는 것을 볼 수 있다. 이전 인공지능 모델들과는 다르게 GAN의 경우 loss 값이 0.5 로 수렴하게 되는데(판별기가 조작된 데이터와 실제 데이터를 판별하지 못 할 경우 0.5를 반환하게 됨) 여기서 평균 제곱 오차를 사용하여 0.5**2 로 loss 값이 수렴하게 된다.

3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 직접 생성기를 구동해 출력 확인

```
1  # 직접 생성기를 구동해 출력 확인
2
3  G.forward(torch.FloatTensor([0.5]))
```

```
tensor([0.9459, 0.0408, 0.9418, 0.0405], grad_fn=<SigmoidBackward>)
```

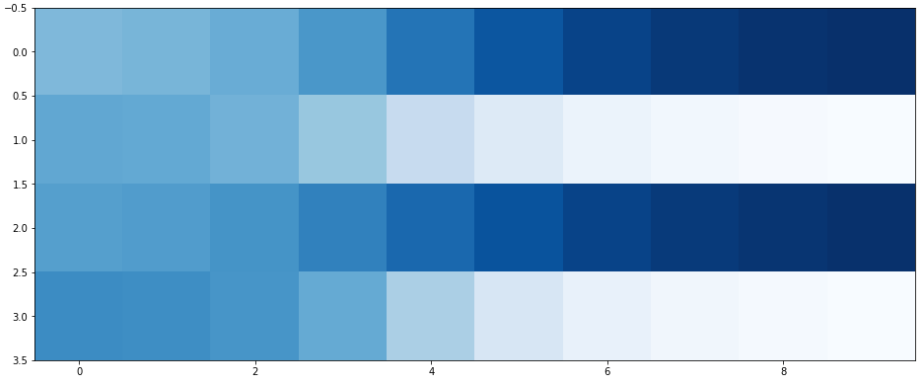
- 1010 패턴을 출력하는 것을 확인할 수 있다.

3. GAN 실습

GAN 실습 1

- GAN을 이용해 1010패턴 생성
- 훈련 시 패턴 시각화

```
1  # 훈련시 패턴 시각화
2
3  plt.figure(figsize = (16,8))
4
5  plt.imshow(numpy.array(image_list).T, interpolation='none', cmap='Blues')
```



- 진할 수록 1, 옅을수록 0 을 의미하고 처음에는 패턴을 찾지 못하다가 학습회수가 증가함에 따라 패턴을 잘 찾은 것을 확인할 수 있다.

3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성

```python
# 라이브러리 임포트
import torch
import torch.nn as nn
from torch.utils.data import Dataset
import pandas, numpy, random
import matplotlib.pyplot as plt
# 데이터 로드
# MNIST dataset
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)
data_loader = DataLoader(dataset=mnist_train,
                                          batch_size=1, # 배치 크기는 100
                                          shuffle=True,
                                          drop_last=True)
```

```python
# 동일한 임의 데이터를 생성하기위한 함수
def generate_random(size):
    random_data = torch.rand(size)
    return random_data
```

3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성

```python
# 분류기 클래스
class Discriminator(nn.Module):
    def __init__(self):
        # 부모 클래스 초기화
        super().__init__()
        # 신경망 레이어 정의
        self.model = nn.Sequential(
            nn.Linear(784, 200),
            nn.Sigmoid(),
            nn.Linear(200, 1),
            nn.Sigmoid()
        )
        # 손실함수 설정
        self.loss_function = nn.MSELoss()
        # 옵티마이저 설정
        self.optimiser = torch.optim.SGD(
            self.parameters(), lr=0.01)
        # 변수 초기화
        self.counter = 0;
        self.progress = []
    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs)
    def train(self, inputs, targets):
        # 신경망의 결과 계산
        outputs = self.forward(inputs)
        # 손실 계산
        loss = self.loss_function(outputs, targets)
        # 카운터를 증가시키고 10회마다 오차 저장
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
        if (self.counter % 10000 == 0):
            print("counter = ", self.counter)
        # 기울기 초기화, 역전파 실행, 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()
    def plot_progress(self):
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0, 1.0), figsize=(16,8), alpha=0.1,
                marker='.', grid=True, yticks=(0, 0.25, 0.5))
```

3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성

```
%%time
# 판별기가 임의의 노이즈로부터 실제 데이터를 구별할수 있는지 확인
D = Discriminator()
for image_data_tensor, target_tensor in data_loader:
    # 실제 데이터
    D.train(image_data_tensor.view(784), torch.FloatTensor([1.0]))
    # 생성된 데이터
    D.train(generate_random(784), torch.FloatTensor([0.0]))
    pass
```

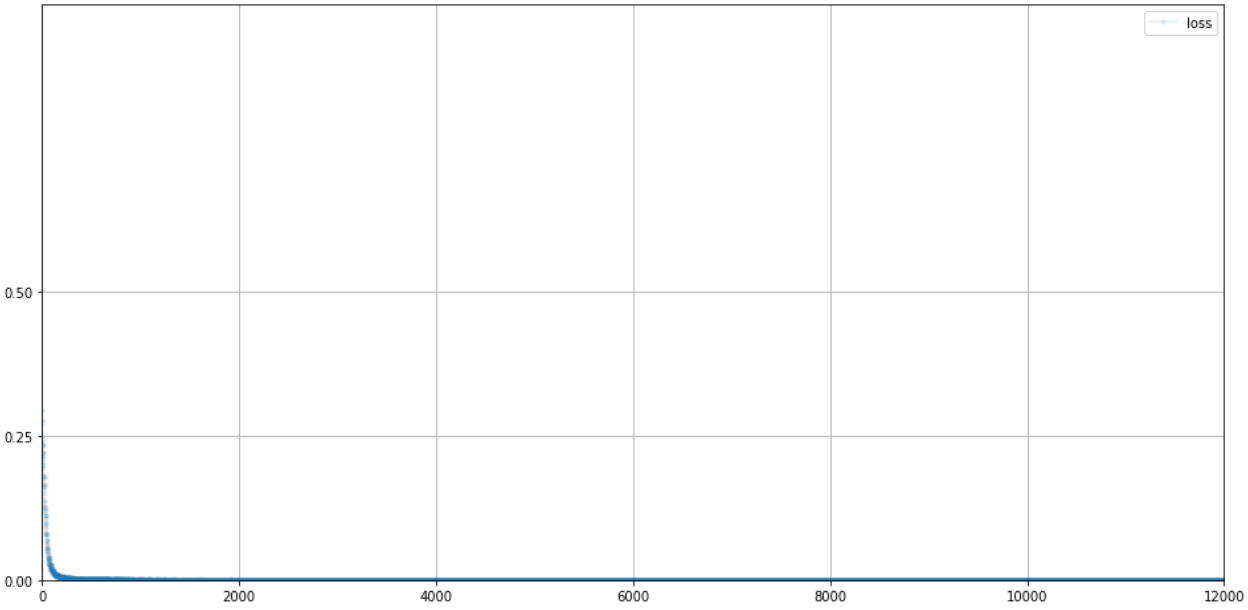
```
counter =  10000
counter =  20000
counter =  30000
counter =  40000
counter =  50000
counter =  60000
counter =  70000
counter =  80000
counter =  90000
counter =  100000
counter =  110000
counter =  120000
Wall time: 1min 13s
```

3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- 판별기 테스트
- 원하던대로 판별기의 손실함수가 0으로 수렴하는 모습을 보인다

```
1  # 판별기 손실 플롯
2  D.plot_progress()
```



3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- 판별기 테스트
- 진짜 데이터는 1에 가까운 값이 나오고
- 가짜 데이터는 0에 가까운 값이 나온다.

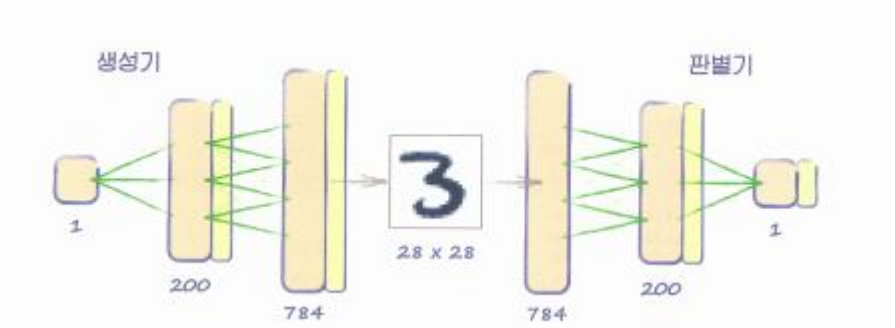
```
# 가짜와 진짜를 판별할수 있는지 판별기 직접 구동
for i in range(4):
    image_data_tensor,_ = mnist_train[i]
    print( D.forward( image_data_tensor.view(784) ).item() )
for i in range(4):
    print( D.forward( generate_random(784) ).item() )
```

```
0.996203601360321
0.9965772032737732
0.9856284856796265
0.9950453042984009
0.005787729285657406
0.006164430174976587
0.005636881105601788
0.004627532325685024
```

3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- 생성기 생성




```python
# 생성기 class
class Generator(nn.Module):
    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()
        # 신경망 레이어 정의
        self.model = nn.Sequential(
            nn.Linear(1, 200),
            nn.Sigmoid(),
            nn.Linear(200, 784),
            nn.Sigmoid()
        )
        # SGD 옵티마이저 생성
        self.optimiser = torch.optim.SGD(
            self.parameters(), lr=0.01)
        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []
    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs)
    def train(self, D, inputs, targets):
        # 신경망 출력 계산
        g_output = self.forward(inputs)

        # 판별기에 값 전달
        d_output = D.forward(g_output)

        # 오차 계산
        loss = D.loss_function(d_output, targets)

        # 매 10회마다 에러를 누적하고 카운터를 증가
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
        # 기울기를 초기화 하고 역전파 후 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()
    def plot_progress(self):
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0, 1.0), figsize=(16,8), alpha=0.1,
                marker='.', grid=True, yticks=(0, 0.25, 0.5))
```

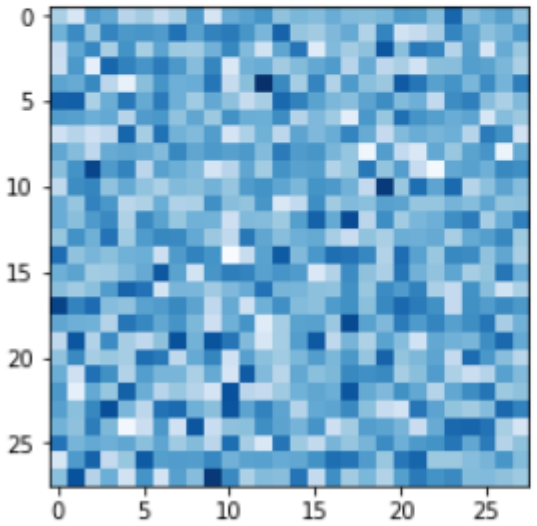
3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- 생성기 확인
- 훈련전의 생성기 결과값을 확인할 수 있다.

```
# 생성기의 출력이 올바른 타입과 형태를 지니고 있는지 확인
G = Generator()
output = G.forward(generate_random(1))
img = output.detach().numpy().reshape(28,28)
plt.imshow(img, interpolation='none', cmap='Blues')
```

```
<matplotlib.image.AxesImage at 0x22157159248>
```



3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- GAN 모델 학습 진행
  - MNIST DATA 60000개
  - 생성기 이미지 60000개

```
%%time
# 판별기 및 생성기 생성
D = Discriminator()
G = Generator()
for image_data_tensor, target_tensor in data_loader:
    # 참일 경우 판별기 훈련
    D.train(image_data_tensor.view(784), torch.FloatTensor([1.0]))
    # 거짓일 경우 판별기 훈련
    # G의 기울기가 계산되지 않도록 detach() 함수를 이용
    D.train(G.forward(generate_random(1)).detach(), torch.FloatTensor([0.0]))
    # 생성기 훈련
    G.train(D, generate_random(1), torch.FloatTensor([1.0]))
```

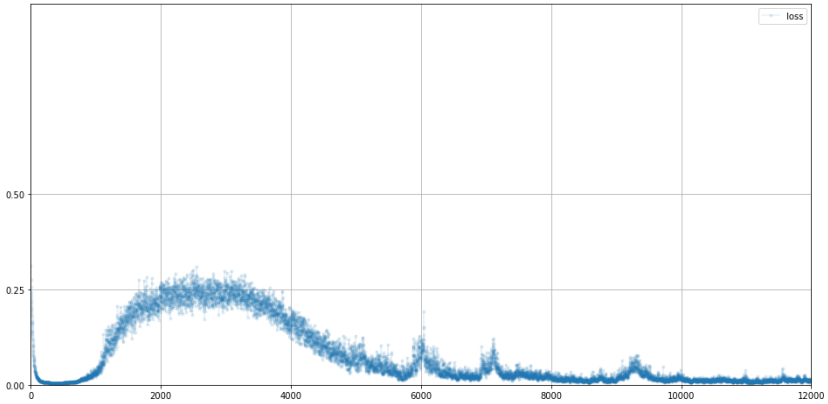
```
counter =  10000
counter =  20000
counter =  30000
counter =  40000
counter =  50000
counter =  60000
counter =  70000
counter =  80000
counter =  90000
counter =  100000
counter =  110000
counter =  120000
Wall time: 2min 23s
```

3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- 판별기 해석 – 손실 값이 0에 가까울 때는 생성기를 성능으로 앞서는 상태이며 0.25로 상승 할 때는 생성기와 판별기가 균형이 맞기 시작한 상태임
- 그 이후는 판별기가 다시 생성기를 앞서0에 수렴하는 형태를 보임

```
1  # 판별기 오차 플롯
2  D.plot_progress()
```

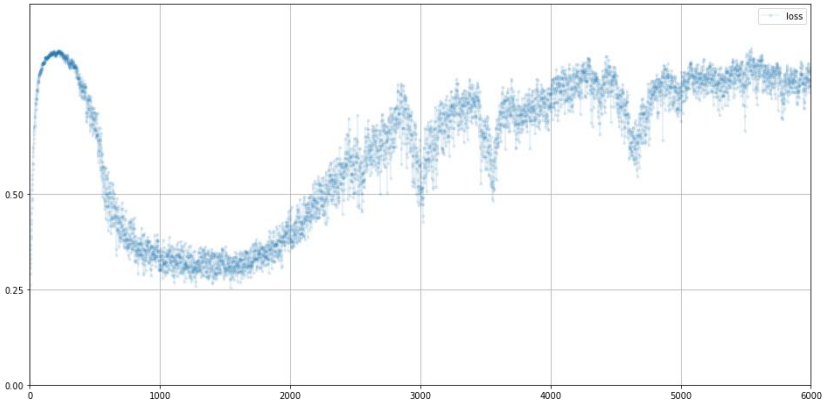


3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- 생성기 해석 – 초기에 손실 값이 치솟은 이유는 판별기가 생성기에서 나온 이미지를 잘 구별했기 때문, 그 후 손실 값은 0.25로 하락하며 판별기와 균형을 이루게 됨.
- 그후 손실 값이 상승하는 이유는 판별기의 성능이 생성기의 성능을 뛰어 넘었기 때문

```
1  # 생성기 오차 플롯
2  G.plot_progress()
```

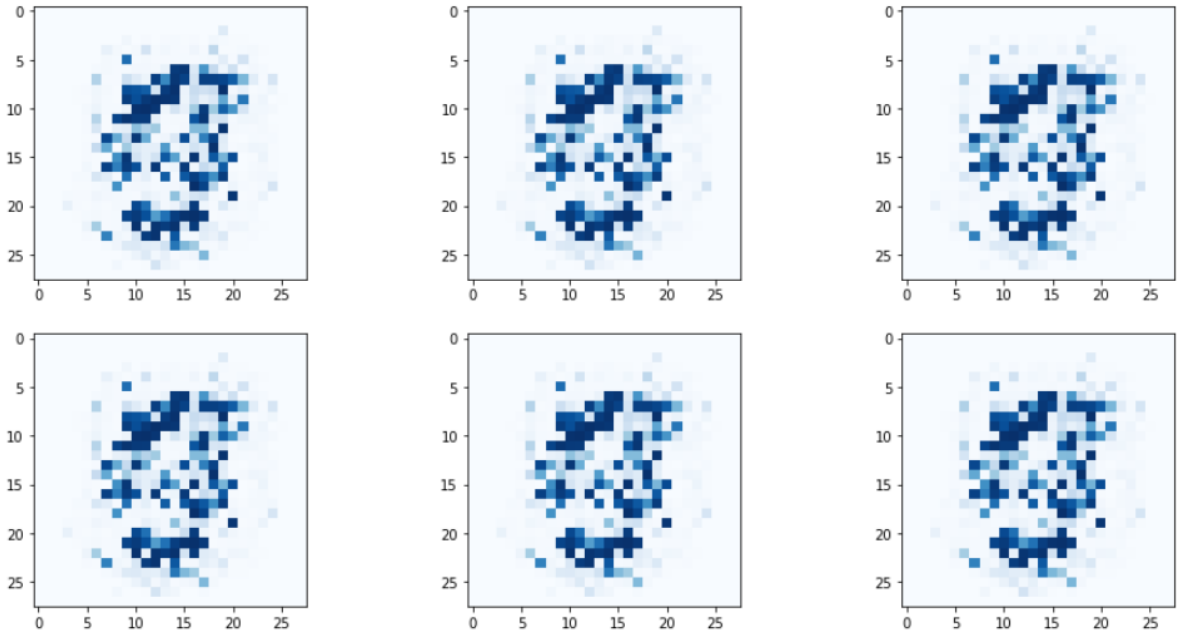


3. GAN 실습

GAN 실습 2

- GAN을 이용해 MNIST 데이터 생성
- 생성기가 노이즈가 아닌 어떤 형태를 가지는 이미지를 생성해 냄 (9, 5)
- 하지만 이는 GAN의 가장 큰 어려움인 모드 붕괴에 해당

```
# 훈련된 생성기로부터 몇개의 출력을 플롯
# plot a 3 column, 2 row array of generated images
f, axarr = plt.subplots(2,3, figsize=(16,8))
for i in range(2):
    for j in range(3):
        output = G.forward(generate_random(1))
        img = output.detach().numpy().reshape(28,28)
        axarr[i,j].imshow(img, interpolation='none', cmap='Blues')
```

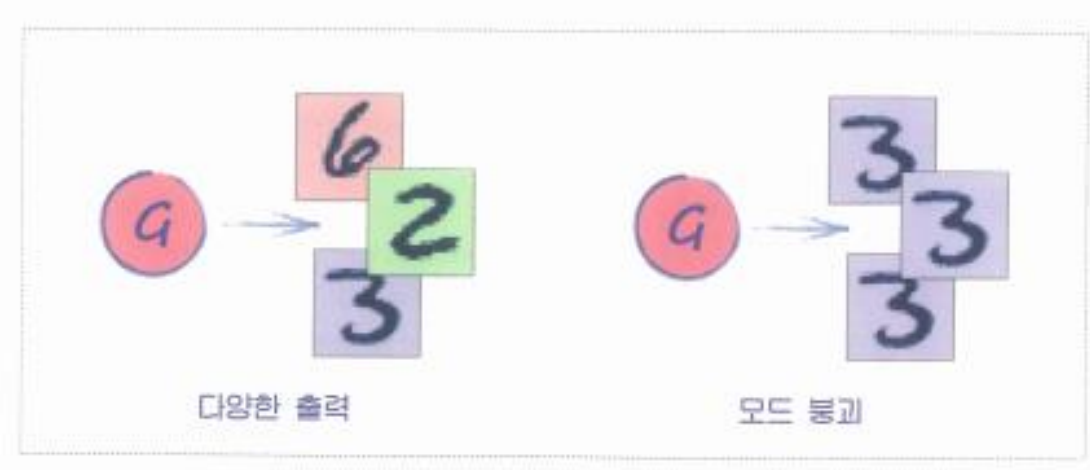


3. GAN 실습

GAN 실습 2

- 모드 붕괴(Mode collapse)?
  - MNIST 예제 상 생성기는 10개의 숫자를 다양하게 생성해야 함
  - 하지만 모드붕괴가 일어나면 오직 하나만 만들게 되거나 선택지 중 극소수만 만들어 내게 됨

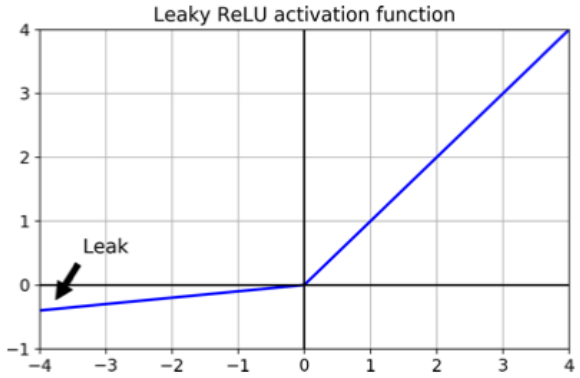
  - 아직 모드붕괴가 일어나는 명확한 이유는 밝혀지지 않음
  - 생성기가 판별기 보다 성능이 좋아졌을 때 과적합이 된다는 설이 있음
  - 즉 판별기가 제대로 일을 하지 못해 생기는 상황이라고도 생각할 수 있음



3. GAN 실습

GAN 실습 3

- GAN 성능 향상 실습
  - MSELoss 손실함수 대신 분류문제에서 더 잘 작동하는 BCELoss 손실함수 사용
  - Sigmoid 대신 LeakyReLU 활성화 함수 사용

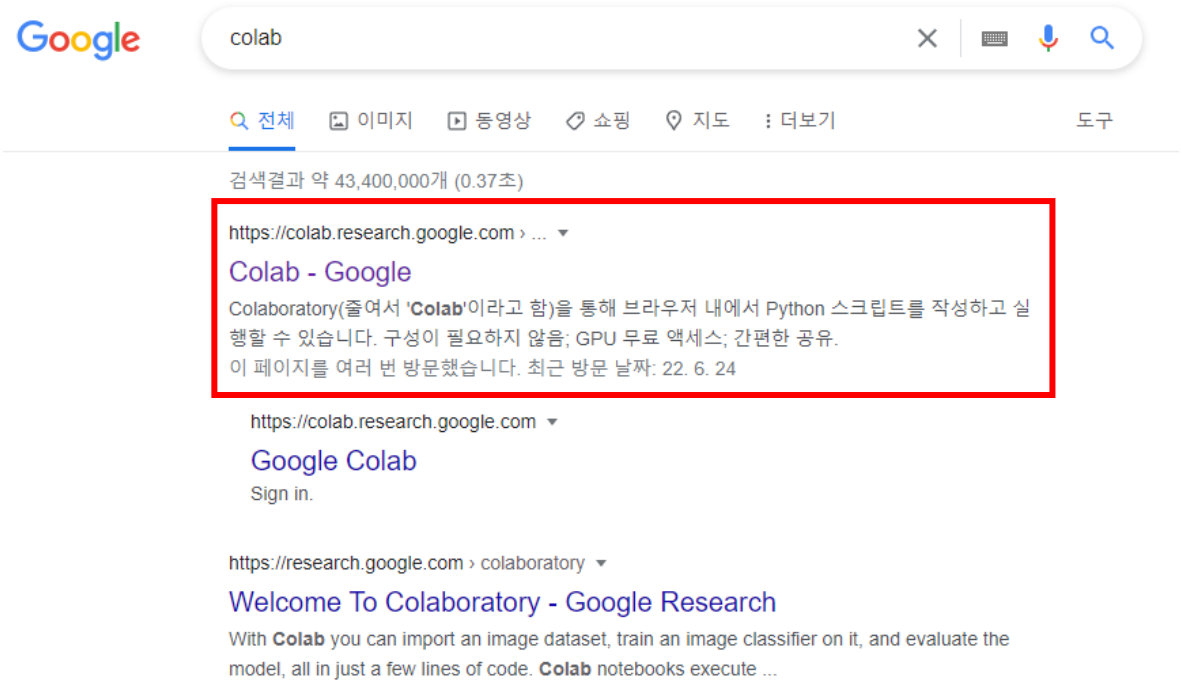


  - 정규화 사용
  - SGD 대신 Adam 옵티마이저 사용
  - 생성기의 입력 시드의 크기 키우기
  - 판별기 훈련때와 생성기훈련때에 생성기에게 입력 다르게 주기
  - 손실 범위 상한을 없애고 수평 격자선 추가

3. GAN 실습

GAN 실습 3

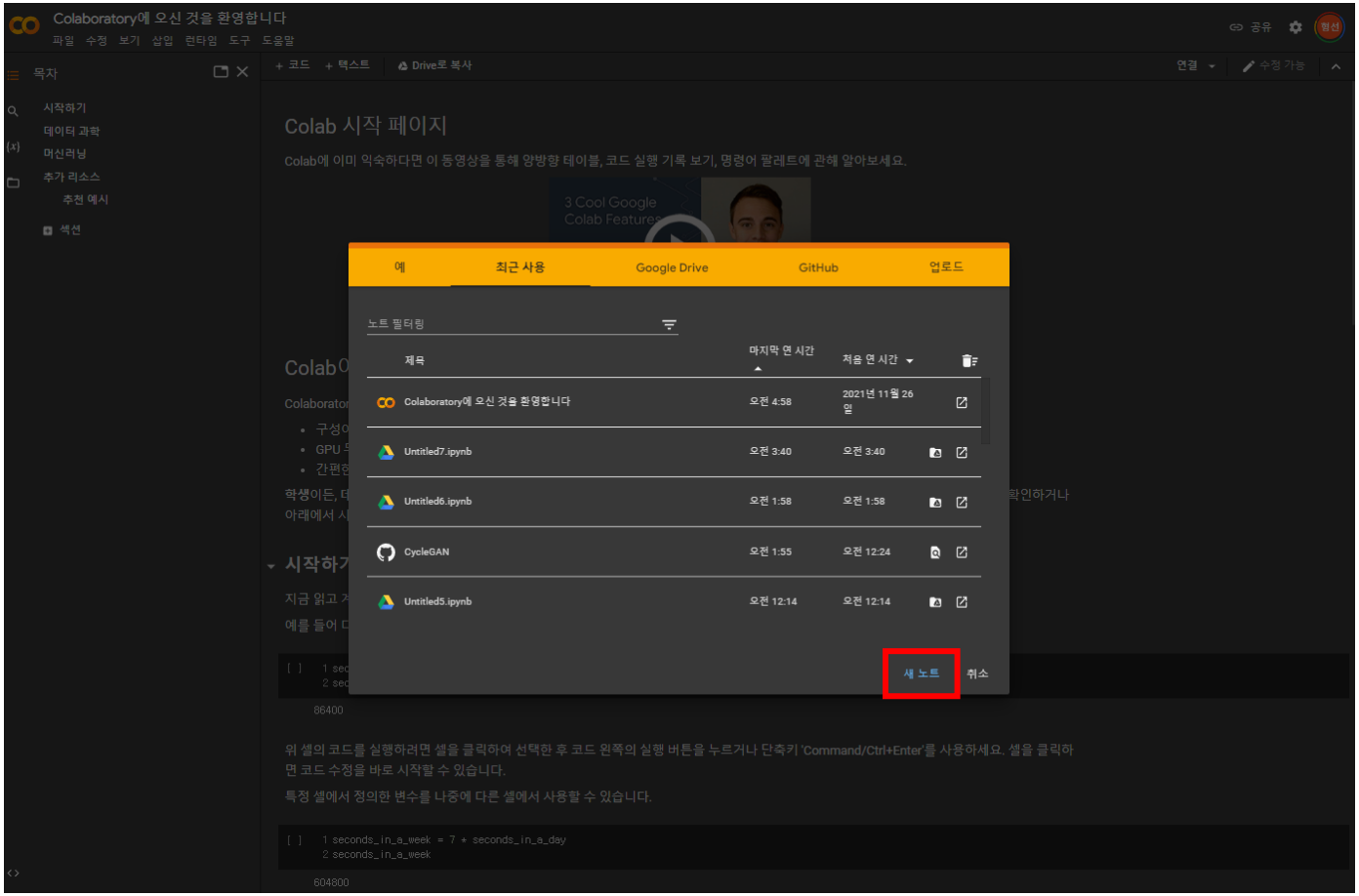
- GAN 성능 향상 실습
  - 또한 CPU로 진행할 경우 속도가 너무 느려 GPU 사용을 해야 함.
- 구글에 코랩 입력
  - 첫번째 페이지 클릭 // 구글 로그인 필요



3. GAN 실습

GAN 실습 3

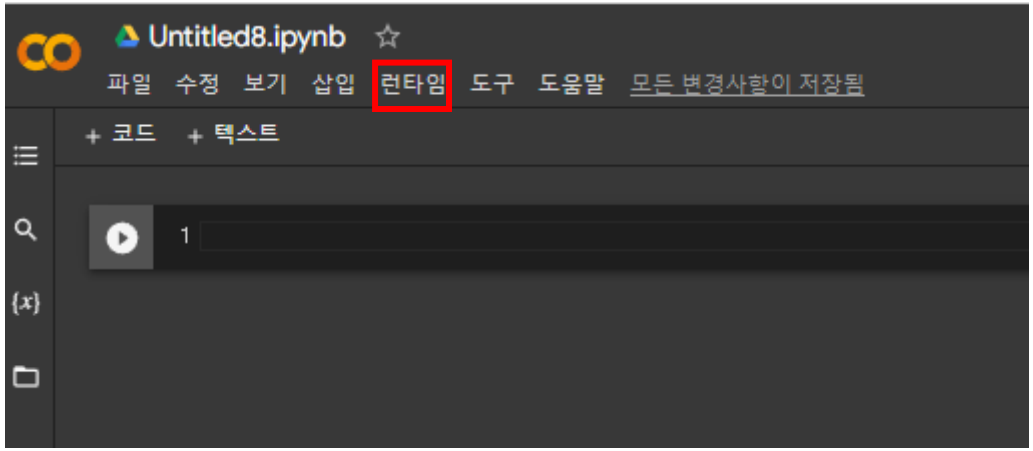
- 새 노트 클릭



3. GAN 실습

GAN 실습 3

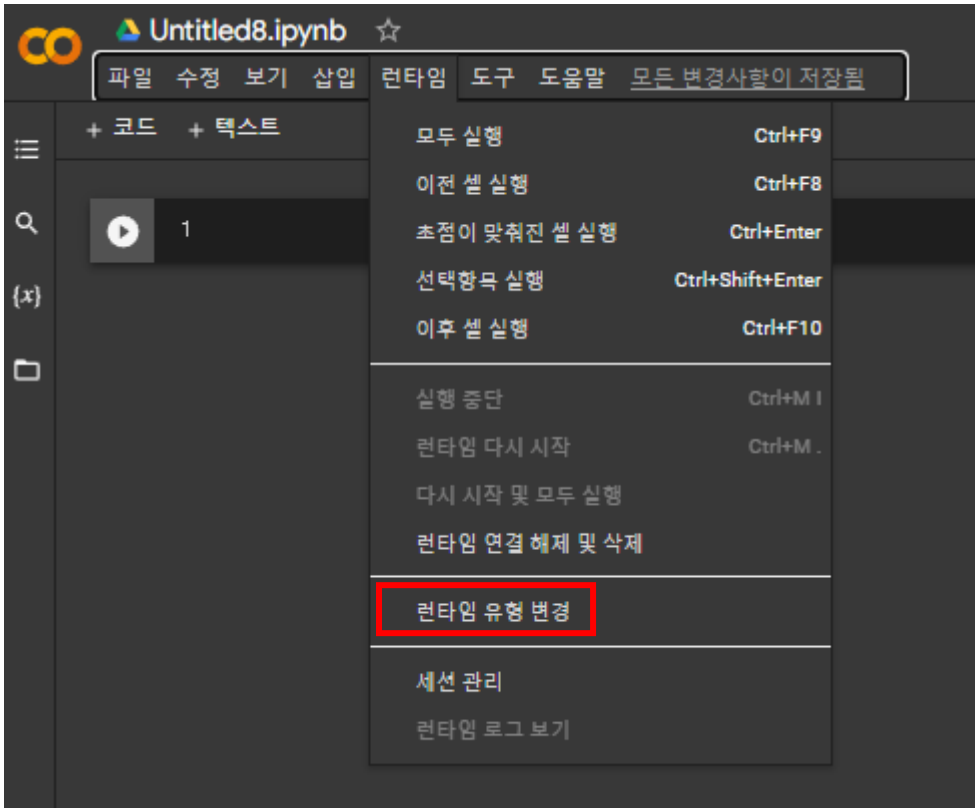
- Jupyter Notebook과 비슷한 환경이 나오면 런타임 클릭



3. GAN 실습

GAN 실습 3

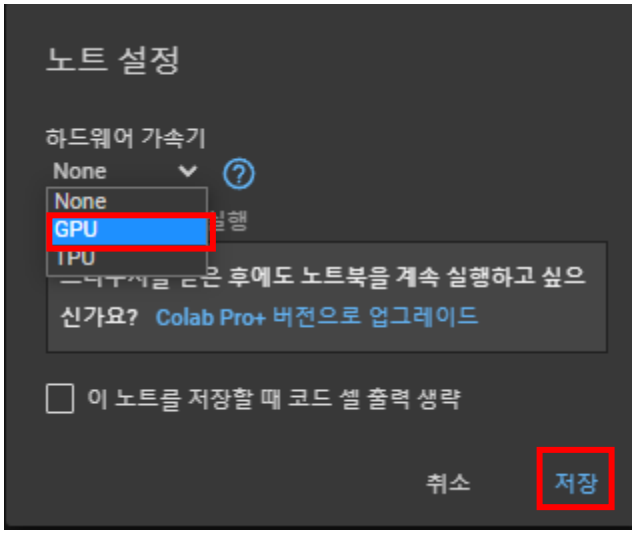
- 런타임 유형 변경 클릭



3. GAN 실습

GAN 실습 3

- 하드웨어 가속기를 GPU로 설정하고 저장



3. GAN 실습

GAN 실습 3

- cuda 출력시 성공

```
[29]  1 import torch
      2 USE_CUDA = torch.cuda.is_available() # GPU를 사용가능하면 True, 아니라면 False를 리턴
      3 device = torch.device("cuda" if USE_CUDA else "cpu") # GPU 사용 가능하면 사용하고 아니면 CPU 사용
      4 print("다음 기기로 학습합니다:", device)

      다음 기기로 학습합니다: cuda
```

3. GAN 실습

GAN 실습 3

- 빨간 박스가 이전과 바뀐 부분

```python
# 라이브러리 임포트
import torch
import torch.nn as nn
from torch.utils.data import Dataset
import pandas, numpy, random
import matplotlib.pyplot as plt
# 데이터 로드
# MNIST dataset
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
USE_CUDA = torch.cuda.is_available() # GPU를 사용가능하면 True, 아니라면 False를 리턴
device = torch.device("cuda" if USE_CUDA else "cpu") # GPU 사용 가능하면 사용하고 아니면 CPU 사용
print("다음 기기로 학습합니다:", device)

mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)
data_loader = DataLoader(dataset=mnist_train,
                                          batch_size=1, # 배치 크기는 100
                                          shuffle=True,
                                          drop_last=True)

# 동일한 임의 데이터를 생성하기위한 함수
def generate_random_image(size):
    random_data = torch.rand(size)
    return random_data

def generate_random_seed(size):
    random_data = torch.randn(size)
    return random_data
```

```
다음 기기로 학습합니다: cuda
```

3. GAN 실습

GAN 실습 3

- 빨간 박스가 이전과 바뀐 부분

```python
# 라이브러리 임포트
import torch
import torch.nn as nn
from torch.utils.data import Dataset
import pandas, numpy, random
import matplotlib.pyplot as plt
# 데이터 로드
# MNIST dataset
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
USE_CUDA = torch.cuda.is_available() # GPU를 사용가능하면 True, 아니라면 False를 리턴
device = torch.device("cuda" if USE_CUDA else "cpu") # GPU 사용 가능하면 사용하고 아니면 CPU 사용
print("다음 기기로 학습합니다:", device)

mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)
data_loader = DataLoader(dataset=mnist_train,
                                          batch_size=1, # 배치 크기는 100
                                          shuffle=True,
                                          drop_last=True)

# 동일한 임의 데이터를 생성하기위한 함수
def generate_random_image(size):
    random_data = torch.rand(size)
    return random_data

def generate_random_seed(size):
    random_data = torch.randn(size)
    return random_data
```

```
다음 기기로 학습합니다: cuda
```

3. GAN 실

```
class Discriminator(nn.Module):
    def __init__(self):
        # 부모 클래스 초기화
        super().__init__()
        # 신경망 레이어 정의
        self.model = nn.Sequential(
            nn.Linear(784, 200),
            nn.LeakyReLU(0.02),
            nn.LayerNorm(200),
            nn.Linear(200, 1),
            nn.Sigmoid())
        # 손실함수 설정
        self.loss_function = nn.BCELoss()
        # 옵티마이저 설정
        self.optimiser = torch.optim.Adam(self.parameters(), lr=0.0001)
        # 변수 초기화
        self.counter = 0;
        self.progress = []
    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs.to(device))
    def train(self, inputs, targets):
        # 신경망의 결과 계산
        outputs = self.forward(inputs).to(device)
        # 손실 계산
        loss = self.loss_function(outputs, targets)
        # 카운터를 증가시키고 10회마다 오차 저장
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
        if (self.counter % 10000 == 0):
            print("counter = ", self.counter)
        # 기울기 초기화, 역전파 실행, 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()
    def plot_progress(self):
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0), figsize=(16,8), alpha=0.1, marker='.', grid=True, yticks=(0, 0.25, 0.5, 1.0, 5.0))
```

```python
# 생성기 class
class Generator(nn.Module):
    def __init__(self):
        # 파이토치 부모 클래스 초기화
        super().__init__()
        # 신경망 레이어 정의
        self.model = nn.Sequential(
            nn.Linear(100, 200),
            nn.LeakyReLU(0.02),
            nn.LayerNorm(200),
            nn.Linear(200, 784),
            nn.Sigmoid())
        # 옵티마이저 생성
        self.optimiser = torch.optim.Adam(self.parameters(), lr=0.0001)
        # 진행 측정을 위한 변수 초기화
        self.counter = 0;
        self.progress = []
    def forward(self, inputs):
        # 모델 실행
        return self.model(inputs.to(device))
    def train(self, D, inputs, targets):
        # 신경망 출력 계산
        g_output = self.forward(inputs).to(device)
        # 판별기에 값 전달
        d_output = D.forward(g_output).to(device)
        # 오차 계산
        loss = D.loss_function(d_output, targets)
        # 매 10회마다 에러를 누적하고 카운터를 증가
        self.counter += 1;
        if (self.counter % 10 == 0):
            self.progress.append(loss.item())
        # 기울기를 초기화 하고 역전파 후 가중치 갱신
        self.optimiser.zero_grad()
        loss.backward()
        self.optimiser.step()
    def plot_progress(self):
        df = pandas.DataFrame(self.progress, columns=['loss'])
        df.plot(ylim=(0), figsize=(16,8), alpha=0.1, marker='.', grid=True, yticks=(0, 0.25, 0.5, 1.0, 5.0))
```

3. GAN 실습

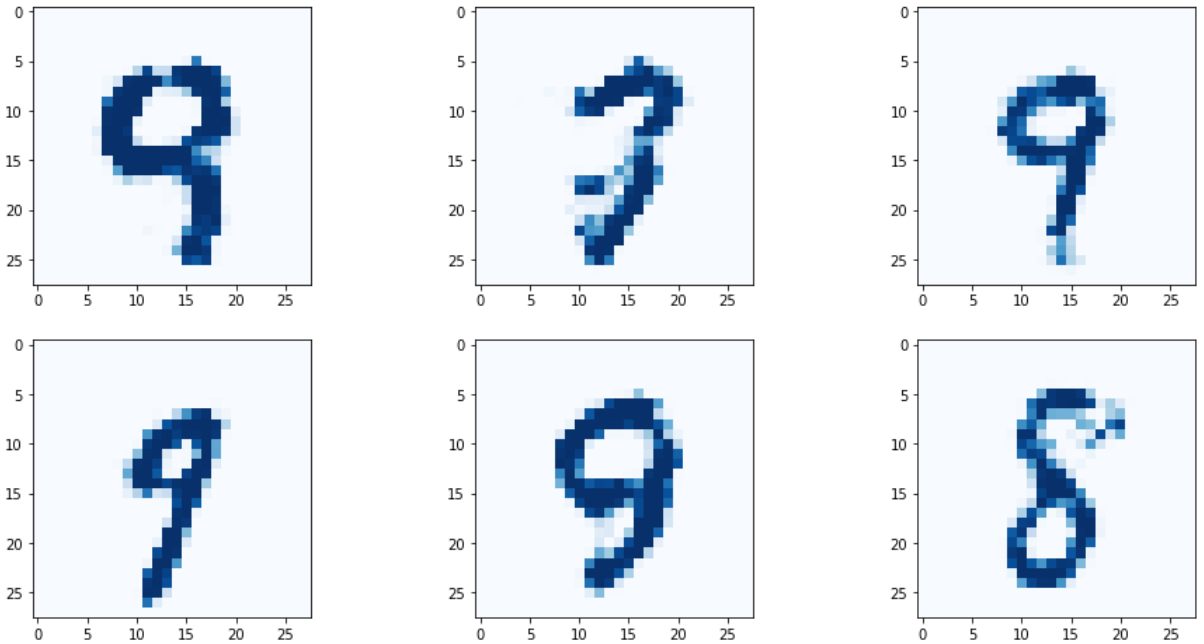
```
%%time
# 판별기 및 생성기 생성
D = Discriminator().to(device)
G = Generator().to(device)
epochs = 4
for epoch in range(epochs):
    print ("epoch = ", epoch + 1)
    for image_data_tensor, target_tensor in data_loader:
        # 참일 경우 판별기 훈련
        D.train(image_data_tensor.view(784).to(device), torch.FloatTensor([1.0]).to(device))
        # 거짓일 경우 판별기 훈련
        # G의 기울기가 계산되지 않도록 detach() 함수를 이용
        D.train(G.forward(generate_random_seed(100)).detach().to(device), torch.FloatTensor([0.0]).to(device))
        # 생성기 훈련
        G.train(D, generate_random_seed(100).to(device), torch.FloatTensor([1.0]).to(device))
```

```
epoch =  1
counter =  10000
counter =  20000
counter =  30000
counter =  40000
counter =  50000
counter =  60000
counter =  70000
counter =  80000
counter =  90000
counter =  100000
counter =  110000
counter =  120000
epoch =  4
counter =  370000
counter =  380000
counter =  390000
counter =  400000
counter =  410000
counter =  420000
counter =  430000
counter =  440000
counter =  450000
counter =  460000
counter =  470000
counter =  480000
CPU times: user 19min 38s, sys: 38.4 s, total: 20min 16s
Wall time: 20min 5s
```

3. GAN 실습

GAN 실습 3

```
# 훈련된 생성기로부터 몇개의 출력을 플롯
# plot a 3 column, 2 row array of generated images
f, axarr = plt.subplots(2,3, figsize=(16,8))
for i in range(2):
    for j in range(3):
        output = G.forward(generate_random_seed(100)).to("cpu")
        img = output.detach().numpy().reshape(28,28)
        axarr[i,j].imshow(img, interpolation='none', cmap='Blues')
```



3. GAN 실습

GAN 실습

- GAN을 이용해 패션 아이템 생성
- 학습에 필요한 라이브러리와 EPOCHS, BATCH_SIZE 같은 하이퍼파라미터 설정

```
import os
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, datasets
from torchvision.utils import save_image
import matplotlib.pyplot as plt
import numpy as np
```

```
# 하이퍼파라미터
EPOCHS = 500
BATCH_SIZE = 100
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
print("Using Device:", DEVICE)
```

```
Using Device: cpu
```

3. GAN 실습

GAN 실습

- GAN을 이용해 패션 아이템 생성
- 학습에 필요한 데이터셋 로딩

```
# Fashion MNIST 데이터셋
trainset = datasets.FashionMNIST(
    './.data',
    train=True,
    download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])
)
train_loader = torch.utils.data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE,
    shuffle     = True
)
```

3. GAN 실습

GAN 실습

- 생성자는 64차원의 랜덤한 텐서를 입력 받아 이에 행렬 곱(Linear)과 활성화 함수(ReLU, Tanh) 연산을 실행
- 생성자의 결과값은 784차원, 즉 Fashion MNIST 속의 이미지와 같은 차원의 텐서
- 판별자는 784차원의 텐서를 입력 받음
- 판별자 역시 입력된 데이터에 행렬 곱과 활성화 함수를 실행시키지만, 생성자와 달리 판별자의 결과값은 입력 받은 텐서가 진짜인지 구분하는 예측 값

```
# 생성자 (Generator)
G = nn.Sequential(
        nn.Linear(64, 256),
        nn.ReLU(),
        nn.Linear(256, 256),
        nn.ReLU(),
        nn.Linear(256, 784),
        nn.Tanh())
```

```
# 판별자 (Discriminator)
D = nn.Sequential(
        nn.Linear(784, 256),
        nn.LeakyReLU(0.2),
        nn.Linear(256, 256),
        nn.LeakyReLU(0.2),
        nn.Linear(256, 1),
        nn.Sigmoid())
```

※ 생성자와 판별자는 코드 순서 상관 없습니다.

3. GAN 실습

GAN 실습

- 생성자와 판별자 학습에 쓰일 오차 함수와 최적화 알고리즘 정의

```
# 모델의 가중치를 지정한 장치로 보내기
D = D.to(DEVICE)
G = G.to(DEVICE)

# 이진 크로스 엔트로피 (Binary cross entropy) 오차 함수와
# 생성자와 판별자를 최적화할 Adam 모듈
criterion = nn.BCELoss()
d_optimizer = optim.Adam(D.parameters(), lr=0.0002)
g_optimizer = optim.Adam(G.parameters(), lr=0.0002)
```

3. GAN 실습

GAN 실습

- GAN 학습 코드

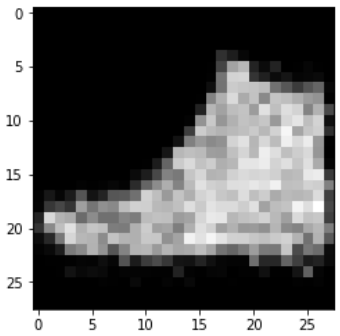
```
total_step = len(train_loader)
for epoch in range(EPOCHS):
    for i, (images, _) in enumerate(train_loader):
        images = images.reshape(BATCH_SIZE, -1).to(DEVICE)
        # '진짜'와 '가짜' 레이블 생성
        real_labels = torch.ones(BATCH_SIZE, 1).to(DEVICE)
        fake_labels = torch.zeros(BATCH_SIZE, 1).to(DEVICE)
        # 판별자가 진짜 이미지를 진짜로 인식하는 오차를 예산
        outputs = D(images)
        d_loss_real = criterion(outputs, real_labels)
        real_score = outputs
        # 무작위 텐서로 가짜 이미지 생성
        z = torch.randn(BATCH_SIZE, 64).to(DEVICE)
        fake_images = G(z)
        # 판별자가 가짜 이미지를 가짜로 인식하는 오차를 계산
        outputs = D(fake_images)
        d_loss_fake = criterion(outputs, fake_labels)
        fake_score = outputs
        # 진짜와 가짜 이미지를 갖고 낸 오차를 더해서 판별자의 오차 계산
        d_loss = d_loss_real + d_loss_fake
        # 역전파 알고리즘으로 판별자 모델의 학습을 진행
        d_optimizer.zero_grad()
        g_optimizer.zero_grad()
        d_loss.backward()
        d_optimizer.step()
        # 생성자가 판별자를 속였는지에 대한 오차를 계산
        fake_images = G(z)
        outputs = D(fake_images)
        g_loss = criterion(outputs, real_labels)
        # 역전파 알고리즘으로 생성자 모델의 학습을 진행
        d_optimizer.zero_grad()
        g_optimizer.zero_grad()
        g_loss.backward()
        g_optimizer.step()
    # 학습 진행 알아보기
    print('Epoch [{}/{}], d_loss: {:.4f}, g_loss: {:.4f}, D(x): {:.2f}, D(G(z)): {:.2f}'
          .format(epoch, EPOCHS, d_loss.item(), g_loss.item(),
                  real_score.mean().item(), fake_score.mean().item()))
```

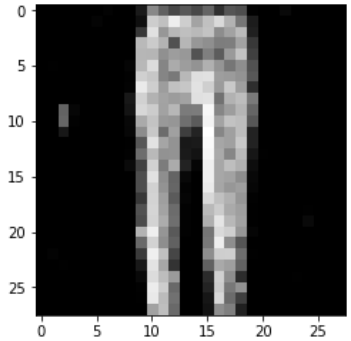
3. GAN 실습

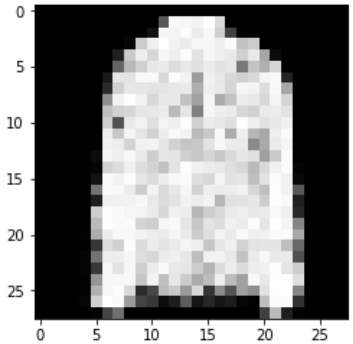
GAN 실습

- 학습이 끝난 새성자의 결과물 확인

```
z = torch.randn(BATCH_SIZE, 64).to(DEVICE)
fake_images = G(z)
for i in range(10):
    fake_images_img = np.reshape(fake_images.data.cpu().numpy()[i],(28, 28))
    plt.imshow(fake_images_img, cmap = 'gray')
    plt.show()
```







3. GAN 실습

GAN 실습

- cGAN으로 내가 원하는 패션 이미지 생성하기
- 학습에 필요한 라이브러리와 EPOCHS, BATCH_SIZE 같은 하이퍼파라미터 설정
- 기존과 동일

```
import os
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, datasets
from torchvision.utils import save_image
import matplotlib.pyplot as plt
import numpy as np
```

```
# 하이퍼파라미터
EPOCHS = 500
BATCH_SIZE = 100
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
print("Using Device:", DEVICE)
```

```
Using Device: cpu
```

3. GAN 실습

GAN 실습

- cGAN으로 내가 원하는 패션 이미지 생성하기
- 학습에 필요한 데이터셋을 로딩
- 이전과 동일

```
# Fashion MNIST 데이터셋
trainset = datasets.FashionMNIST(
    './.data',
    train=True,
    download=True,
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])
)
train_loader = torch.utils.data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE,
    shuffle     = True
)
```

3. GAN 실습

GAN 실습

- cGAN으로 내가 원하는 패션 이미지 생성하기
- 생성자와 판별자 정의

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

        self.embed = 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):
        c = self.embed(labels)
        x = torch.cat([z, c], 1)
        return self.model(x)
```

3. GAN 실습

GAN 실습

- cGAN으로 내가 원하는 패션 이미지 생성하기
- 생성자와 판별자 정의

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

        self.embed = 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):
        c = self.embed(labels)
        x = torch.cat([x, c], 1)
        return self.model(x)
```

3. GAN 실습

GAN 실습

- cGAN으로 내가 원하는 패션 이미지 생성하기
- 생성자와 판별자 학습에 쓰일 오차 함수와 최적화 알고리즘 정의

```
# 모델 인스턴스를 만들고 모델의 가중치를 지정한 장치로 보내기
D = Discriminator().to(DEVICE)
G = Generator().to(DEVICE)

# 이진 교차 엔트로피 함수와
# 생성자와 판별자를 최적화할 Adam 모듈
criterion = nn.BCELoss()
d_optimizer = optim.Adam(D.parameters(), lr =0.0002)
g_optimizer = optim.Adam(G.parameters(), lr =0.0002)
```

3. GAN 실습

GAN 실습

- 학습 진행

```
[0/300] d_loss:0.3160 g_loss: 6.8871 D(x):0.89 D(G(z)):0.04
[1/300] d_loss:0.5910 g_loss: 3.9762 D(x):0.87 D(G(z)):0.19
[2/300] d_loss:0.3091 g_loss: 4.8783 D(x):0.92 D(G(z)):0.08
[3/300] d_loss:0.1595 g_loss: 4.3155 D(x):0.95 D(G(z)):0.06
[4/300] d_loss:0.5849 g_loss: 3.0070 D(x):0.78 D(G(z)):0.13
[5/300] d_loss:0.4532 g_loss: 3.0106 D(x):0.82 D(G(z)):0.10
[6/300] d_loss:1.0214 g_loss: 2.8349 D(x):0.64 D(G(z)):0.09
[7/300] d_loss:0.6247 g_loss: 2.1058 D(x):0.79 D(G(z)):0.20
[8/300] d_loss:0.6507 g_loss: 1.7656 D(x):0.82 D(G(z)):0.27
[9/300] d_loss:0.6669 g_loss: 1.7414 D(x):0.81 D(G(z)):0.25
[10/300] d_loss:0.6982 g_loss: 2.1408 D(x):0.81 D(G(z)):0.26
```

```
[291/300] d_loss:1.3586 g_loss: 1.0188 D(x):0.53 D(G(z)):0.41
[292/300] d_loss:1.2751 g_loss: 0.8963 D(x):0.57 D(G(z)):0.44
[293/300] d_loss:1.1592 g_loss: 1.0320 D(x):0.64 D(G(z)):0.43
[294/300] d_loss:1.2472 g_loss: 0.8309 D(x):0.61 D(G(z)):0.47
[295/300] d_loss:1.2423 g_loss: 1.0166 D(x):0.63 D(G(z)):0.43
[296/300] d_loss:1.3765 g_loss: 0.9225 D(x):0.49 D(G(z)):0.44
[297/300] d_loss:1.2616 g_loss: 0.8626 D(x):0.56 D(G(z)):0.45
[298/300] d_loss:1.2170 g_loss: 0.8461 D(x):0.57 D(G(z)):0.43
[299/300] d_loss:1.1471 g_loss: 1.2162 D(x):0.60 D(G(z)):0.38
```

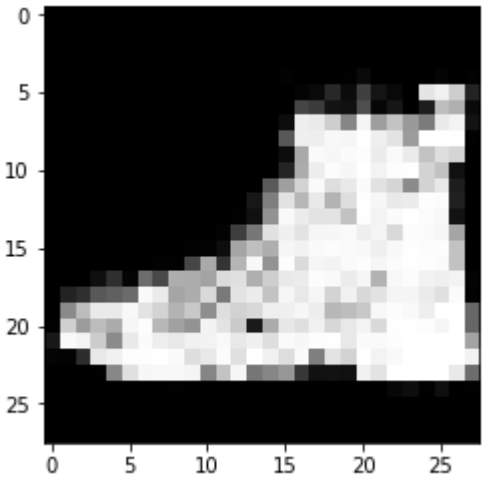
```python
total_step = len(train_loader)
for epoch in range(EPOCHS):
    for i, (images, labels) in enumerate(train_loader):
        images = images.reshape(BATCH_SIZE, -1).to(DEVICE)
        # '진짜'와 '가짜' 레이블 생성
        real_labels = torch.ones(BATCH_SIZE, 1).to(DEVICE)
        fake_labels = torch.zeros(BATCH_SIZE, 1).to(DEVICE)
        # 판별자가 진짜 이미지를 진짜로 인식하는 오차 계산 (데이터셋 레이블 입력)
        labels = labels.to(DEVICE)
        outputs = D(images, labels)
        d_loss_real = criterion(outputs, real_labels)
        real_score = outputs
        # 무작위 텐서와 무작위 레이블을 생성자에 입력해 가짜 이미지 생성
        z = torch.randn(BATCH_SIZE, 100).to(DEVICE)
        g_label = torch.randint(0, 10, (BATCH_SIZE,)).to(DEVICE)
        fake_images = G(z, g_label)
        # 판별자가 가짜 이미지를 가짜로 인식하는 오차 계산
        outputs = D(fake_images, g_label)
        d_loss_fake = criterion(outputs, fake_labels)
        fake_score = outputs
        # 진짜와 가짜 이미지를 갖고 낸 오차를 더해서 판별자의 오차 계산
        d_loss = d_loss_real + d_loss_fake
        # 역전파 알고리즘으로 판별자 모델의 학습을 진행
        d_optimizer.zero_grad()
        g_optimizer.zero_grad()
        d_loss.backward()
        d_optimizer.step()
        # 생성자가 판별자를 속였는지에 대한 오차 계산(무작위 레이블 입력)
        fake_images = G(z, g_label)
        outputs = D(fake_images, g_label)
        g_loss = criterion(outputs, real_labels)
        # 역전파 알고리즘으로 생성자 모델의 학습을 진행
        d_optimizer.zero_grad()
        g_optimizer.zero_grad()
        g_loss.backward()
        g_optimizer.step()
    print('epoch [{}/{}] d_loss:{:.4f} g_loss: {:.4f} D(x):{:.2f} D(G(z)):{:.2f}'
          .format(epoch,
                  EPOCHS,
                  d_loss.item(),
                  g_loss.item(),
                  real_score.mean().item(),
                  fake_score.mean().item()))
```

3. GAN 실습

GAN 실습

- 결과 확인

```
# 만들고 싶은 아이템 생성하고 시각화하기
item_number = 9 # 아이템 번호
z = torch.randn(1, 100).to(DEVICE) # 배치 크기 1
g_label = torch.full((1,), item_number, dtype=torch.long).to(DEVICE)
sample_images = G(z, g_label)

sample_images_img = np.reshape(sample_images.data.cpu().numpy()
                               [0],(28, 28))
plt.imshow(sample_images_img, cmap = 'gray')
plt.show()
```



Q & A

Thank you