GAN

강예준

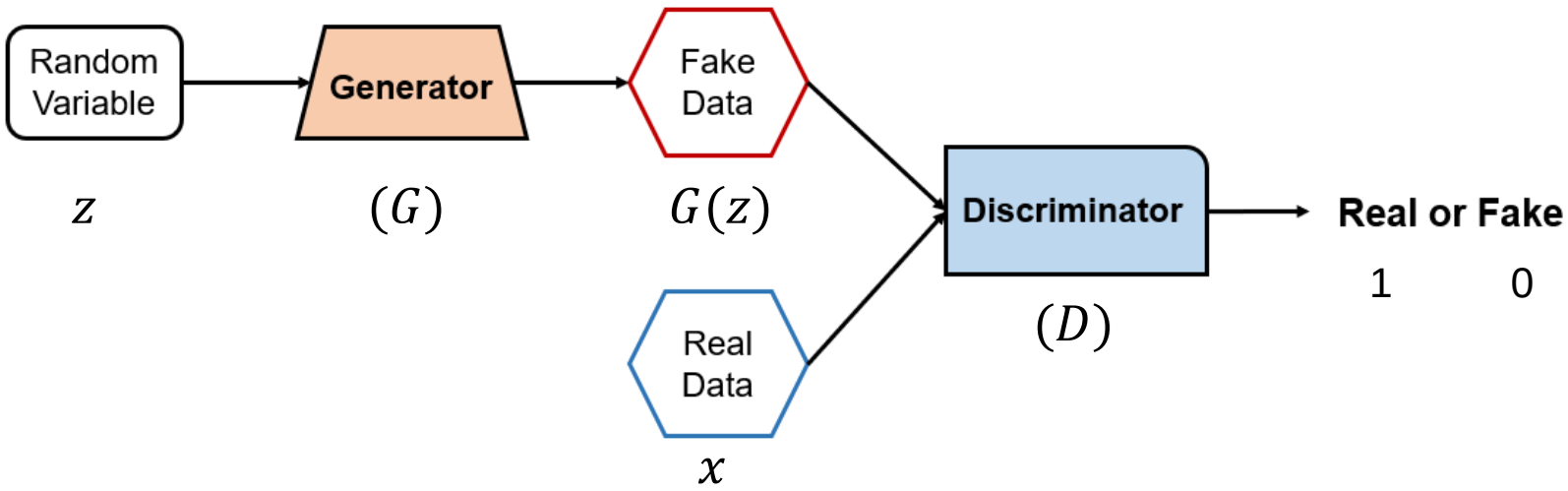
https://youtu.be/oZ8qxJwop60

GAN의 구조

- GAN (Generative Adversarial Network)
  - Discriminator : 경찰
  - Generator : 위조지폐범

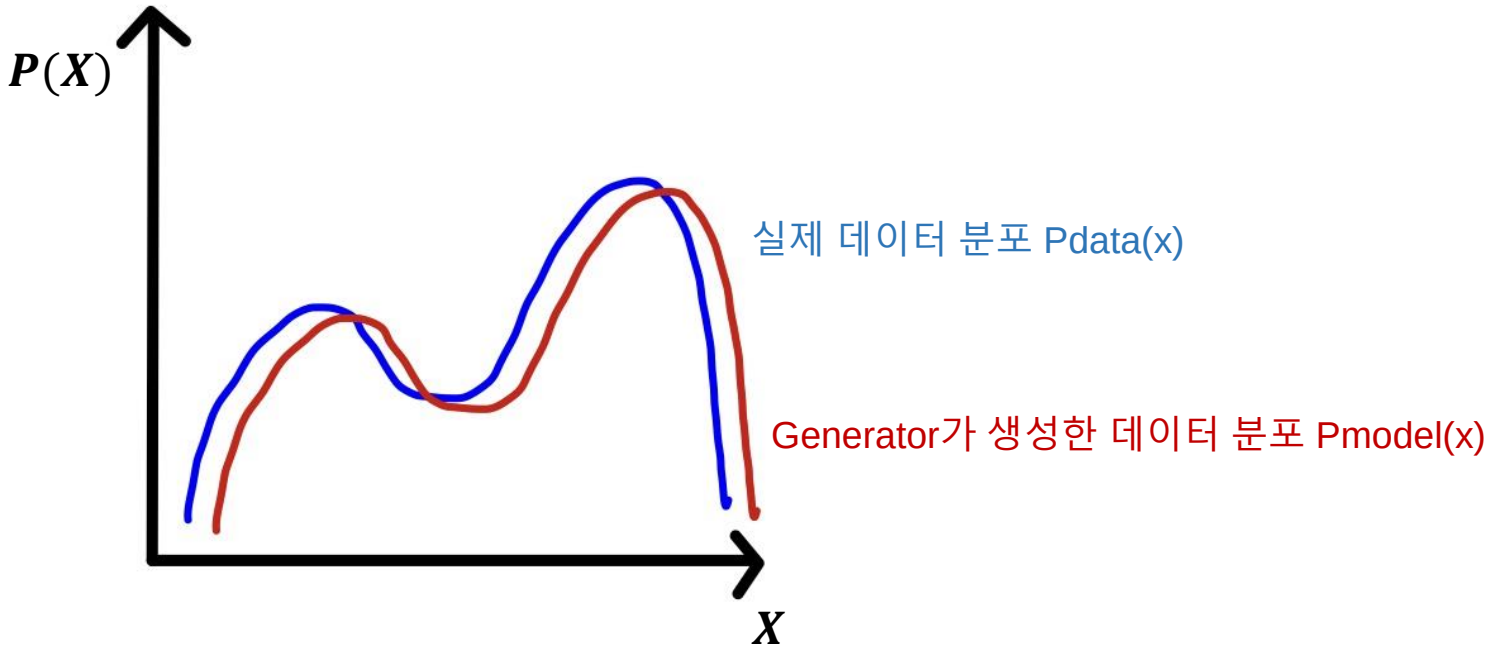
GAN의 구조

- GAN (Generative Adversarial Network)
  - Discriminator : 진짜 · 가짜 데이터를 **구별**
  - Generator : 가짜 데이터를 **생성해** Discriminator를 **속임**



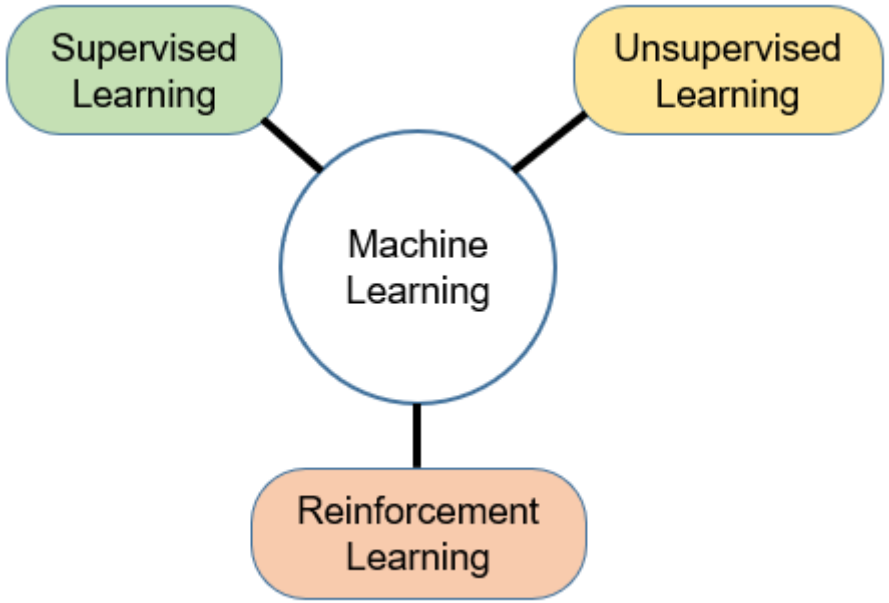
Generator

- Generator는 데이터의 분포를 학습해 실제 데이터 분포와 비슷해지도록 가짜 데이터 생성



GAN의 구조

- 학습의 종류
  - **Supervised Learning** : 정답이 있는 데이터를 활용해 데이터를 학습 **(Discriminator)**
  - **Unsupervised Learning** : 정답을 따로 알려주지 않고, 비슷한 데이터들을 군집화 **(Generator)**



Objective Function of GAN

- Objective Function of Discriminator

$$min_G max_D V(D,G) = \mathrm{E}_{x\sim pdata_{(x)}}[\log D(x)] + \mathrm{E}_{z\sim pz(z)}[\log(1 - D(G(z)))]$$

  - Discriminator는 Objective Function이 최대가 되도록 학습
  - 따라서 D(x)가 1, D(G(z))가 0에 가깝게 됨

- Objective Function of Generator

$$min_G max_D V(D,G) = \mathrm{E}_{x\sim pdata_{(x)}}[\log D(x)] + \mathrm{E}_{z\sim pz(z)}[\log(1 - D(G(z)))]$$

  - Generator는 실제 데이터를 고려할 필요 X
  - Objective Function이 최소가 되도록 학습
  - 따라서 D(G(z))가 1에 가깝게 됨

Model 구성

```python
def create_generator():
    generator=Sequential()
    generator.add(Dense(units=256,input_dim=100))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=512))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=1024))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=784, activation='tanh'))

    generator.compile(loss='binary_crossentropy', optimizer=adam_optimizer())
    return generator
```

```python
def create_discriminator():
    discriminator=Sequential()
    discriminator.add(Dense(units=1024,input_dim=784))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units=512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units=256))
    discriminator.add(LeakyReLU(0.2))

    discriminator.add(Dense(units=1, activation='sigmoid'))

    discriminator.compile(loss='binary_crossentropy', optimizer=adam_optimizer())
    return discriminator
```

GAN 구성 및 생성

```
def create_gan(discriminator, generator):
    discriminator.trainable=False
    gan_input = Input(shape=(100,))
    x = generator(gan_input)
    gan_output= discriminator(x)
    gan= Model(inputs=gan_input, outputs=gan_output)
    gan.compile(loss='binary_crossentropy', optimizer='adam')
    return gan
```

```
# Creating GAN
generator = create_generator()
discriminator = create_discriminator()
gan = create_gan(discriminator, generator)
```

Discriminator Training

```
noise= np.random.normal(0,1, [batch_size, 100])

# Generate fake MNIST images from noised input
generated_images = generator.predict(noise)

# Get a random set of  real images
image_batch =X_train[np.random.randint(low=0,high=X_train.shape[0],size=batch_size)]

# Construct different batches of real and fake data
X= np.concatenate([image_batch, generated_images])

# Labels for generated and real data
y_dis=np.zeros(2*batch_size)
y_dis[:batch_size]=0.9

# Pretrain discriminator on  fake and real data before starting the gan.
discriminator.trainable=True
discriminator.train_on_batch(X, y_dis)
```

Generator Training

```
y_gen = np.ones(batch_size)
discriminator.trainable=False
gan.train_on_batch(noise, y_gen)
```

Q & A