

# CGAN & WGAN

---

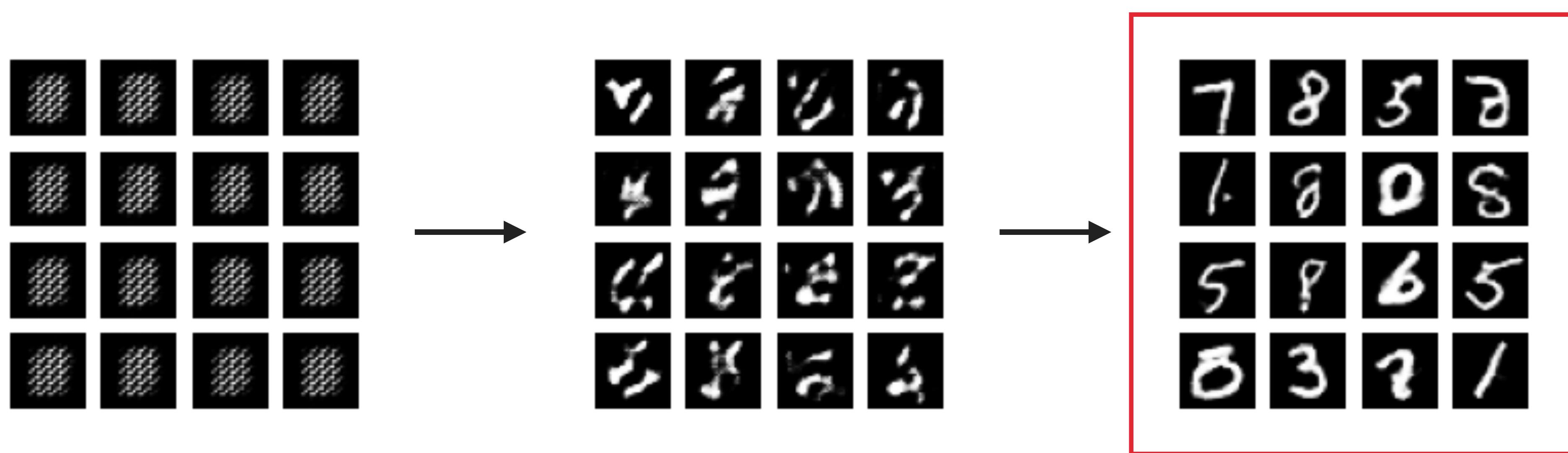
GAN: ONE STEP FURTHER

## GENERATIVE ADVERSARIAL NETWORKS

- ▶ 생성적 적대 신경망, 혹은 적대적 생성 신경망
- ▶ 임의의 **인코딩(노이즈)**이 주어졌을 때 새로운 유의미한 출력을 생성할 수 있다.
- ▶ 임의로 생성되는 출력을 제어할 수 있을까?

## DCGAN USING KERAS ONLY

- ▶ Deep Convolution-NN GAN
- ▶ <https://github.com/lukepark327/keras-only-GAN>



**CGAN**

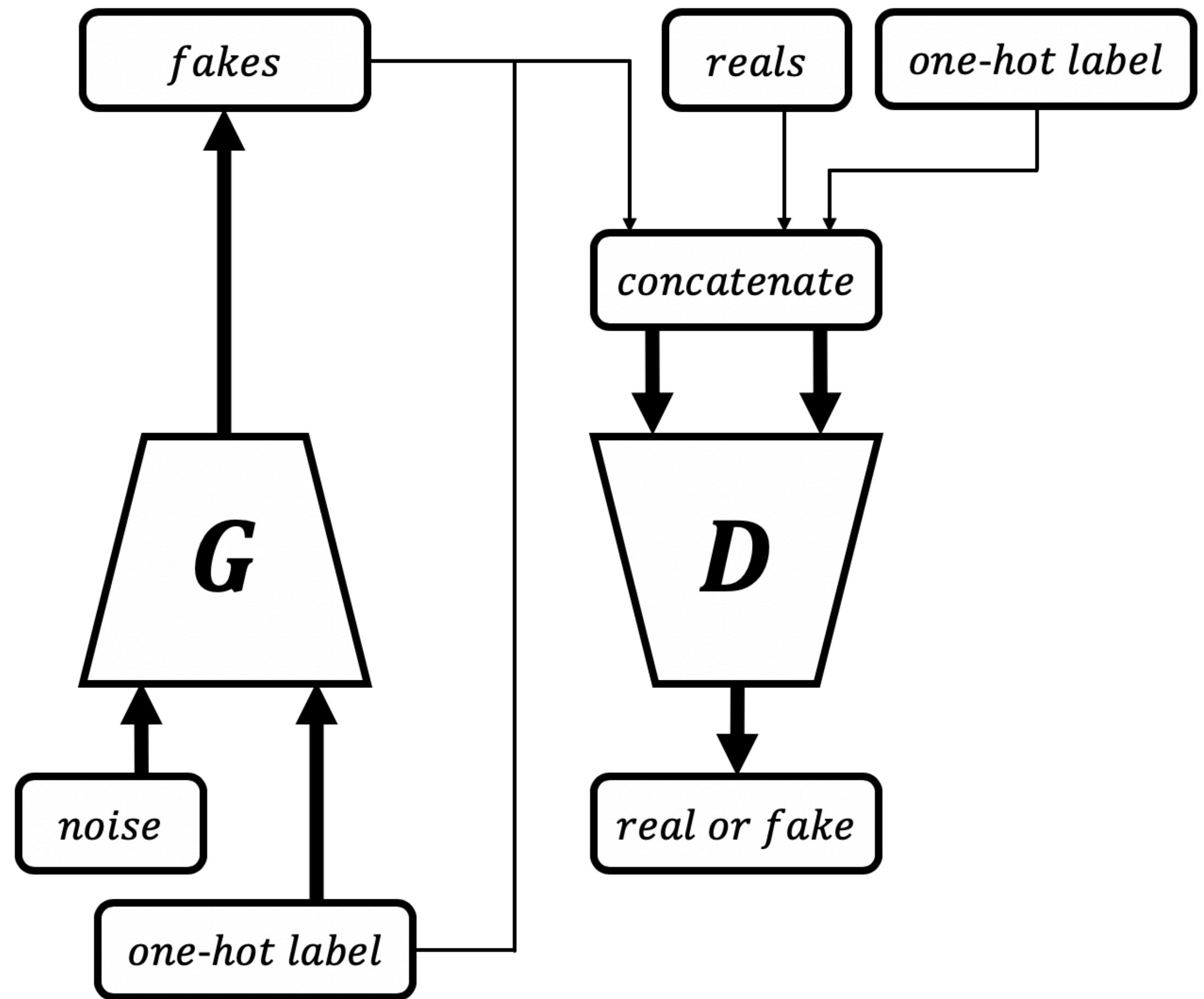
## CONDITIONAL GAN

- ▶ DCGAN과 거의 동일한 구조를 사용
  - ▶ ‘조건’에 해당하는 입력이 추가됨
  - ▶ 조건: 숫자를 one-hot-encoding

## CONDITIONAL GAN

- ▶ DCGAN과 거의 동일한 구조를 사용
  - ▶ ‘조건’에 해당하는 입력이 추가됨
  - ▶ 조건: 숫자를 one-hot-encoding

## CONDITIONAL GAN



## CONDITIONAL GAN

- ▶ Training CGAN:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- ▶ 원-핫 레이블 조건인  $y$ 와  $y'$ 이 붙음

# IMPLEMENTATION

## CGAN USING KERAS ONLY

► <https://github.com/lukepark327/keras-only-GAN>



## CGAN USING KERAS ONLY

```
# Discriminator
inputs = Input(shape=img_shape, name='discriminator_input')
y_labels = Input(shape=label_shape, name='discriminator_class')
discriminator = build_discriminator(inputs, y_labels)
optimizer = RMSprop(lr=lr, decay=decay)
discriminator.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
discriminator.summary()

# Generator
inputs = Input(shape=z_shape, name='generator_input')
y_labels = Input(shape=label_shape, name='generator_class')
generator = build_generator(inputs, y_labels, img_shape[0])
generator.summary()
```

- ▶ There is no `generator.compile()`

## DCGAN USING KERAS ONLY

```
# Adversarial
discriminator.trainable = False # Fix weights # Boolean flag at compiling

inputs = Input(shape=z_shape, name='generator_input')
y_labels = Input(shape=label_shape, name='generator_class')
adversarial = Model([inputs, y_labels], discriminator(generator([inputs, y_labels]), y_labels), name=model_name)
optimizer = RMSprop(lr=lr * 0.5, decay=decay * 0.5)
adversarial.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
adversarial.summary()
```

- ▶ Compiling `adversarial` instead of `generator`

## DCGAN USING KERAS ONLY

```
x = np.concatenate([real_imgs, fake_imgs])
y = np.concatenate([np.ones([batch_size]), np.zeros([batch_size])]).reshape(-1, 1)
y_labels = np.concatenate([real_labels, fake_labels])
loss, acc = discriminator.train_on_batch([x, y_labels], y)
log = "%05d: [discriminator loss: %f, acc: %f]" % (i, loss, acc)

# Train adversarial
x = np.random.uniform(-1.0, 1.0, size=[batch_size, latent_size]) # noise vectors
y = np.ones([batch_size, 1])
y_labels = np.eye(10)[np.random.choice(10, batch_size)]
loss, acc = adversarial.train_on_batch([x, y_labels], y)
log = "%s [adversarial loss: %f, acc: %f]" % (log, loss, acc)
print(log)
```

# WGAN

## WASSERSTEIN GAN

- ▶ GAN의 훈련은 매우 어려움
  - ▶ 판별기와 생성기가 서로 상반된 목표를 가지고 있음
  - ▶ 두 신경망의 수렴 속도를 잘 맞춰줘야 함

## WASSERSTEIN GAN

- ▶ GAN에서 말하는 데이터의 분포
  - ▶ 데이터는 **다차원 공간**에서의 **벡터**로 표현 가능
  - ▶ 벡터의 방향 혹은 축은 features를 의미
  - ▶ 벡터의 크기는 해당하는 features의 정도를 의미

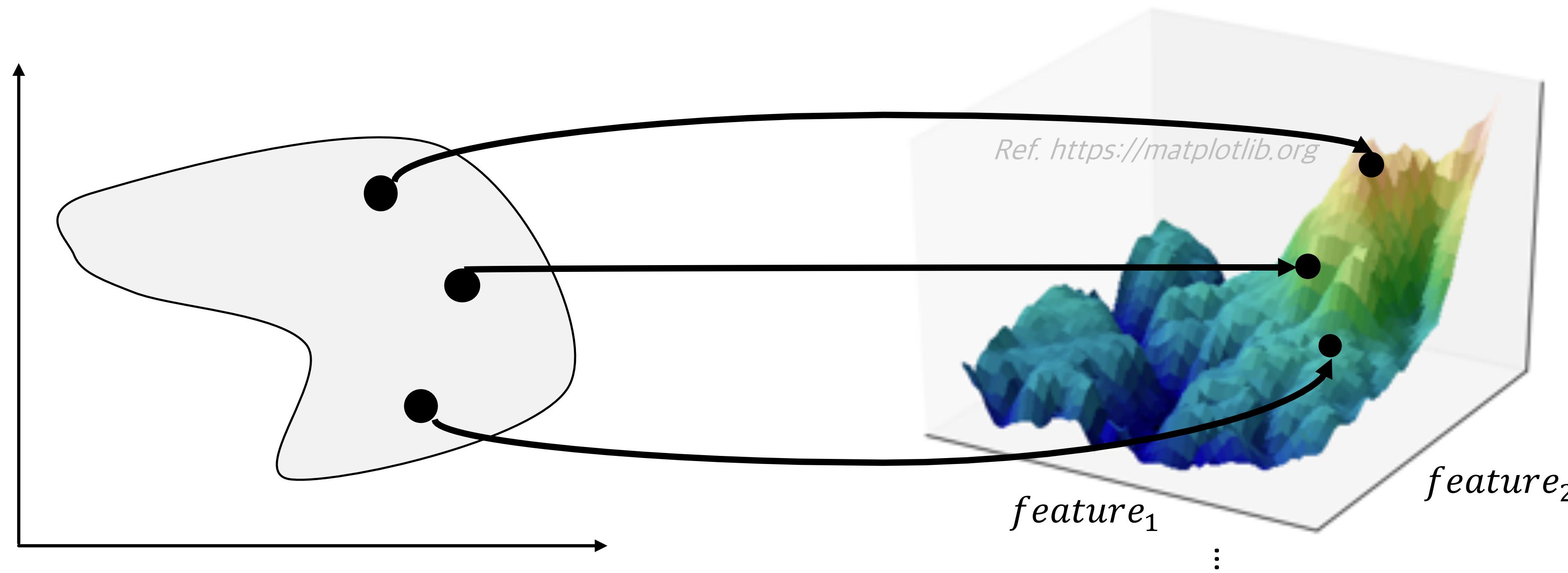
## WASSERSTEIN GAN

- ▶ 데이터를 수치화(벡터화)하면 여러 해석이 가능
  - ▶ 분포
  - ▶ 확률밀도, 확률분포

## WASSERSTEIN GAN

- ▶ GAN이 근사하고자 하는 함수의 역할:
  - ▶ 랜덤 노이즈의 분포를 `reals`에 가까운 확률분포로 변환하는 함수
  - ▶ 판별기가 `real`이라고 판단하는 영역에 점차 가까워 짐

## WASSERSTEIN GAN



## WASSERSTEIN GAN

- ▶ 어떻게 fakes가 reals의 분포를 근사할 수 있을까?
- ▶ **분포 사이의 거리를 측정할 수 없을까?**
- ▶ Wasserstein 또는 Earth-Mover Distance(EMD)

A yellow WERKLUST WG 35 D wheel loader is shown from a side-front angle, working at a construction site. The loader has a large bucket attachment and is positioned next to a large mound of dirt. The background shows a hilly landscape with some buildings and power lines under a clear sky.

DISTANCE OF

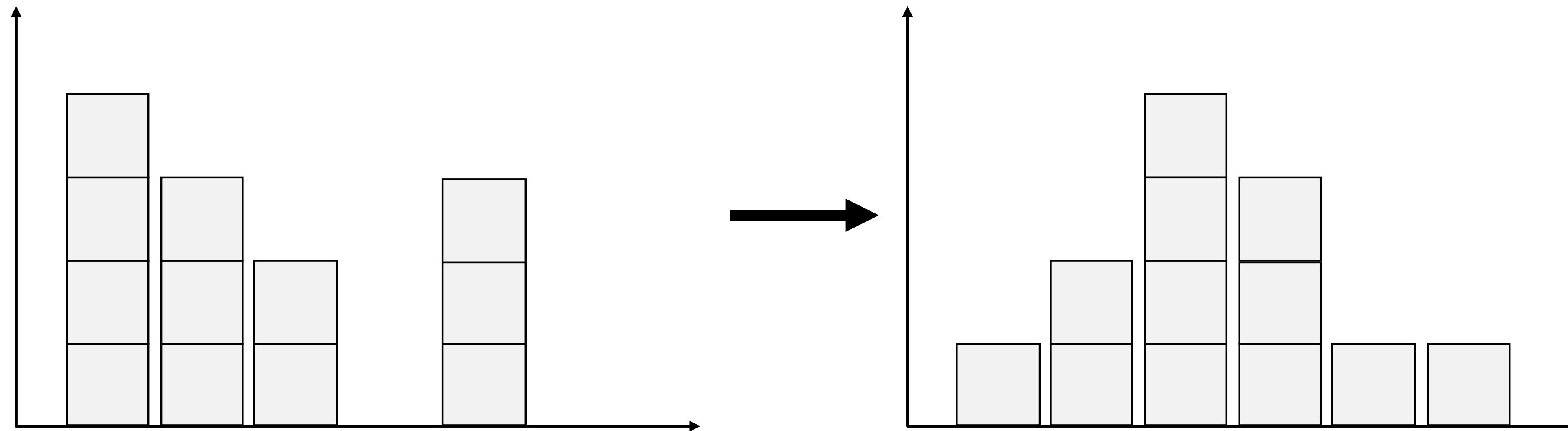
---

# EARTH-MOVER

## EMD

- ▶ EMD의 의미
  - ▶ 확률분포를 다른 확률분포로 바꾸기 위해
  - ▶ 얼마나 많은 질량을
  - ▶ 얼마의 거리만큼 옮겨야 하는가

# EMD

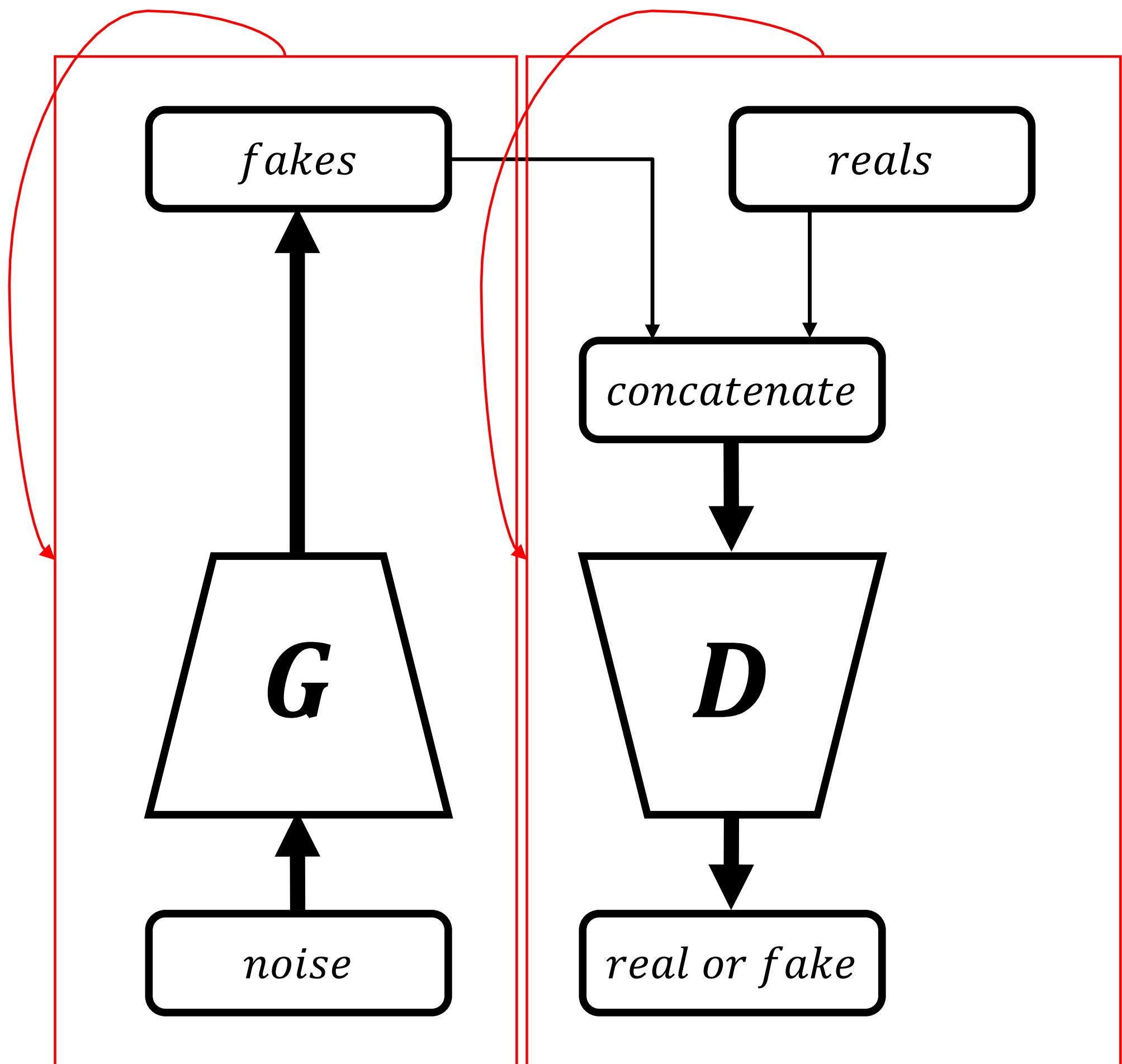


# IMPLEMENTATION

## WASSERSTEIN GAN

- ▶ 판별기와 생성기를 교대로 학습
  - ▶ DCGAN과는 다르게, 서로 다른 횟수로 학습
  - ▶ 판별기를 조금 더 적게 학습하고
  - ▶ 생성기를 조금 더 많이 학습함

## WASSERSTEIN GAN



## WASSERSTEIN GAN

- ▶ reals의 label은 1.0, fakes의 label은 -1.0
- ▶  $-\frac{1}{n} \sum_{i=1}^n (y_i p_i)$

## WASSERSTEIN GAN

```
def wasserstein_loss(y_label, y_pred):  
    return -K.mean(y_label * y_pred)
```

# CGAN & WGAN

---

GAN: ONE STEP FURTHER