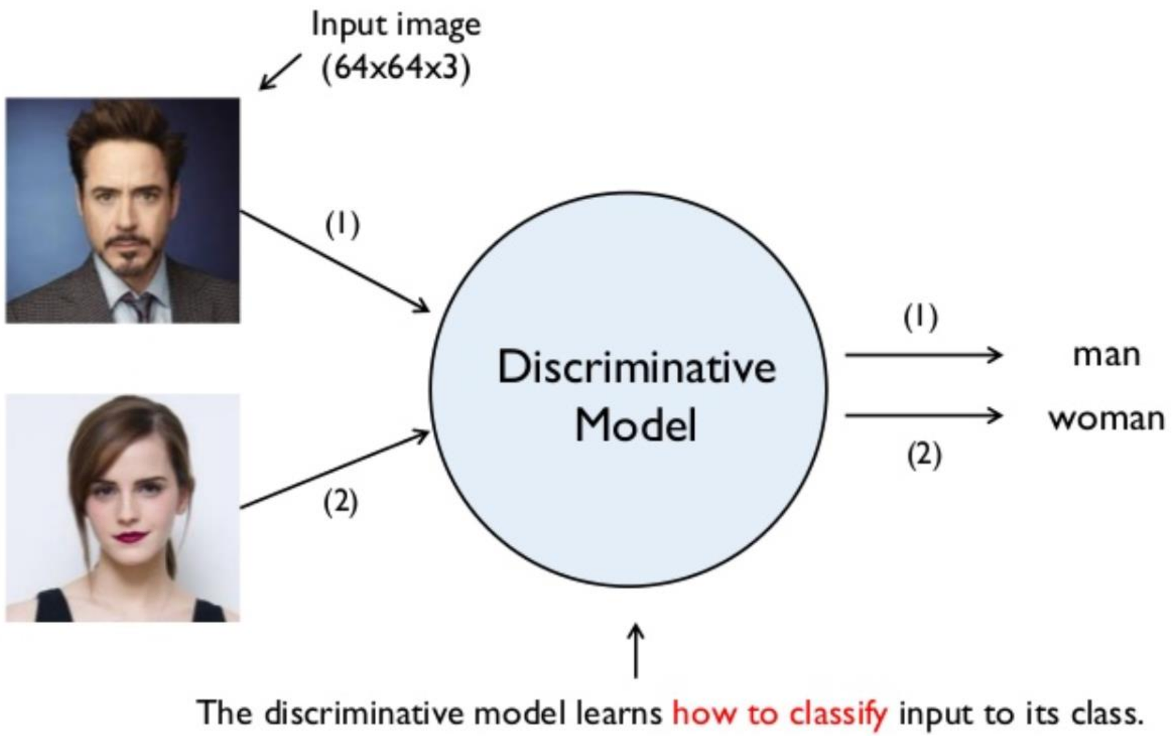
Generative Adversarial Networks (GAN)

Prof. Seungchul Lee
Industrial AI Lab.

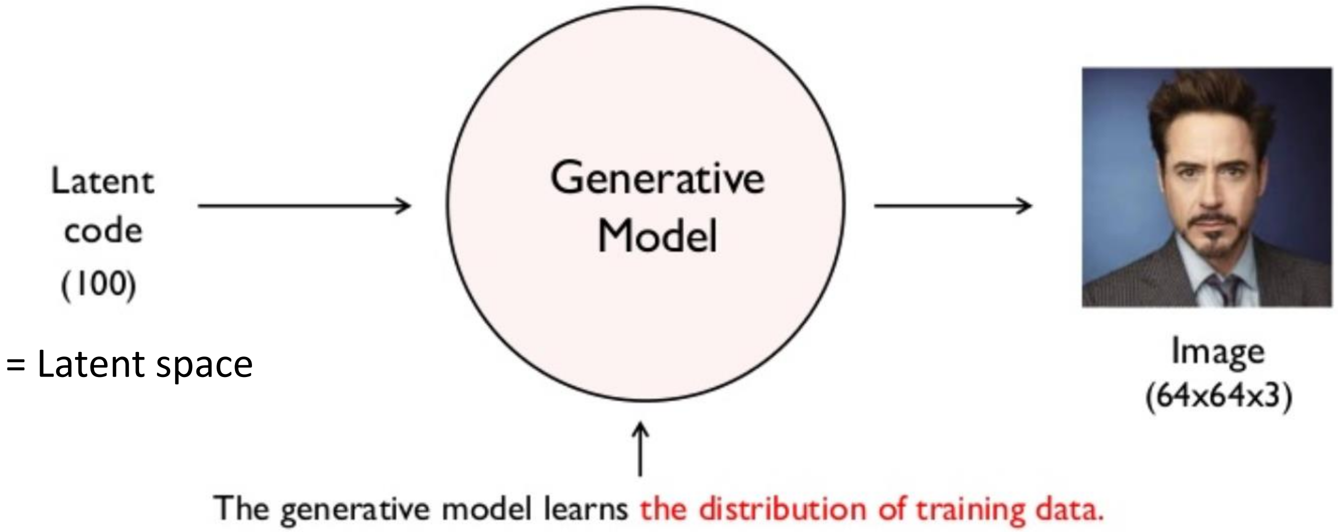
Supervised Learning

• Discriminative model



Unsupervised Learning

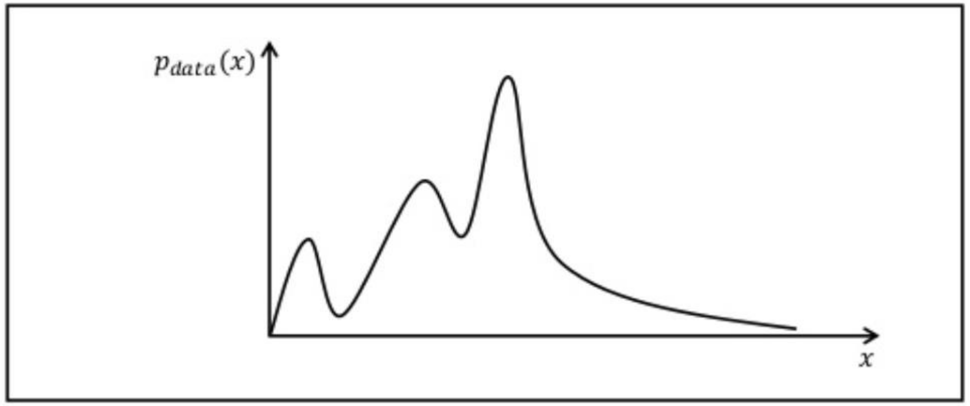
- Generative model



Probability Distribution

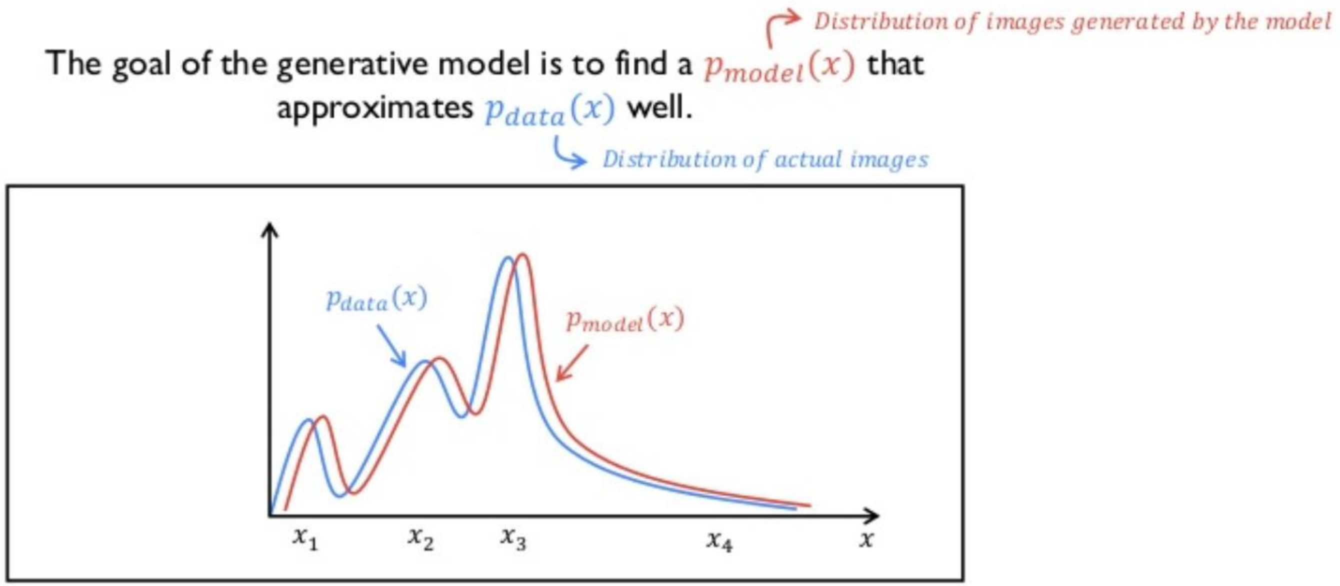
Probability density function

There is a $p_{data}(x)$ that represents the distribution of actual images.



Probability Density Estimation Problem

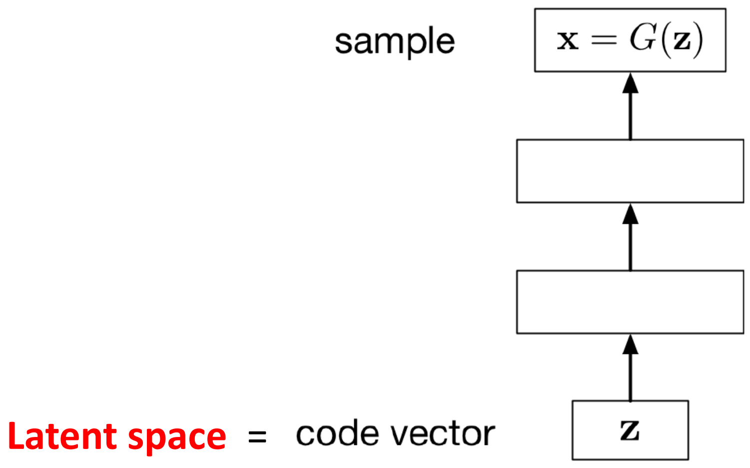
- If $P_{model}(x)$ can be estimated as close to $P_{data}(x)$, then data can be generated by sampling from $P_{model}(x)$



Generative Models from Lower Dimension

- Learn transformation via a neural network

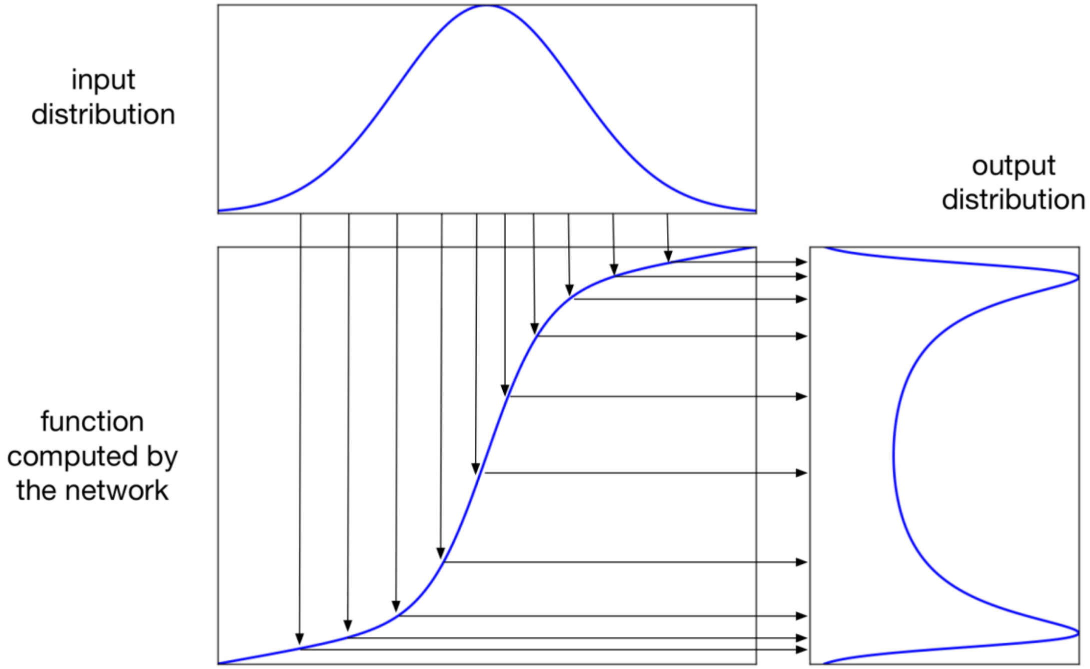
- Start by sampling the code vector z from a fixed, simple distribution (e.g. uniform distribution or Gaussian distribution)
- Then this code vector is passed as input to a deterministic generator network G, which produces an output sample $x = G(z)$



Source: Prof. Roger Grosse at U of Toronto

Deterministic Transformation (by Network)

- 1-dimensional example:

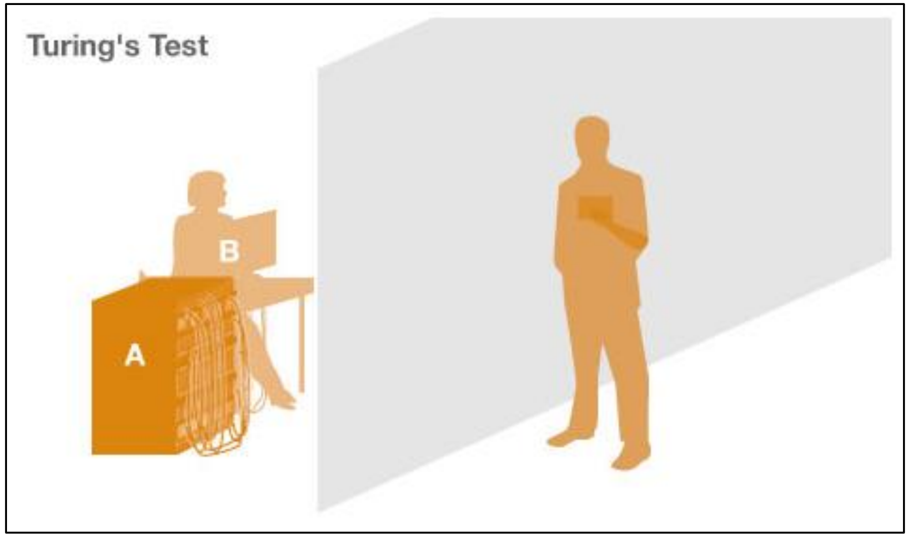


- Remember
  - Network does not generate distribution, but
  - It maps known distribution to target distribution

Source: Prof. Roger Grosse at U of Toronto

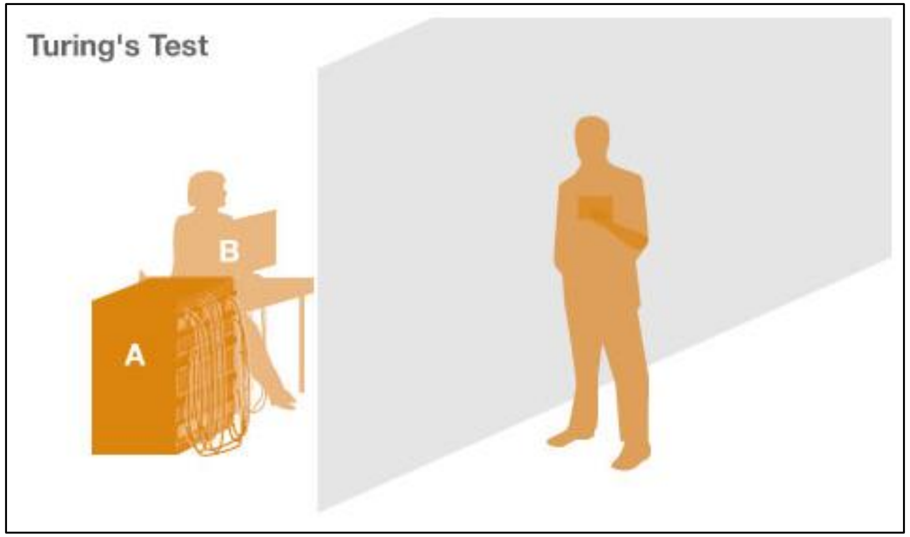
Generative Adversarial Networks (GANs)

- In generative modeling, we'd like to train a network that models a distribution, such as a distribution over images.
- GANs do not work with any explicit density function !
  - Instead, take game-theoretic approach



Turing Test

- One way to judge the quality of the model is to sample from it.
- GANs are based on a very different idea:
  - Model to produce samples which are indistinguishable from the real data, as judged by a discriminator network whose job is to tell real from fake



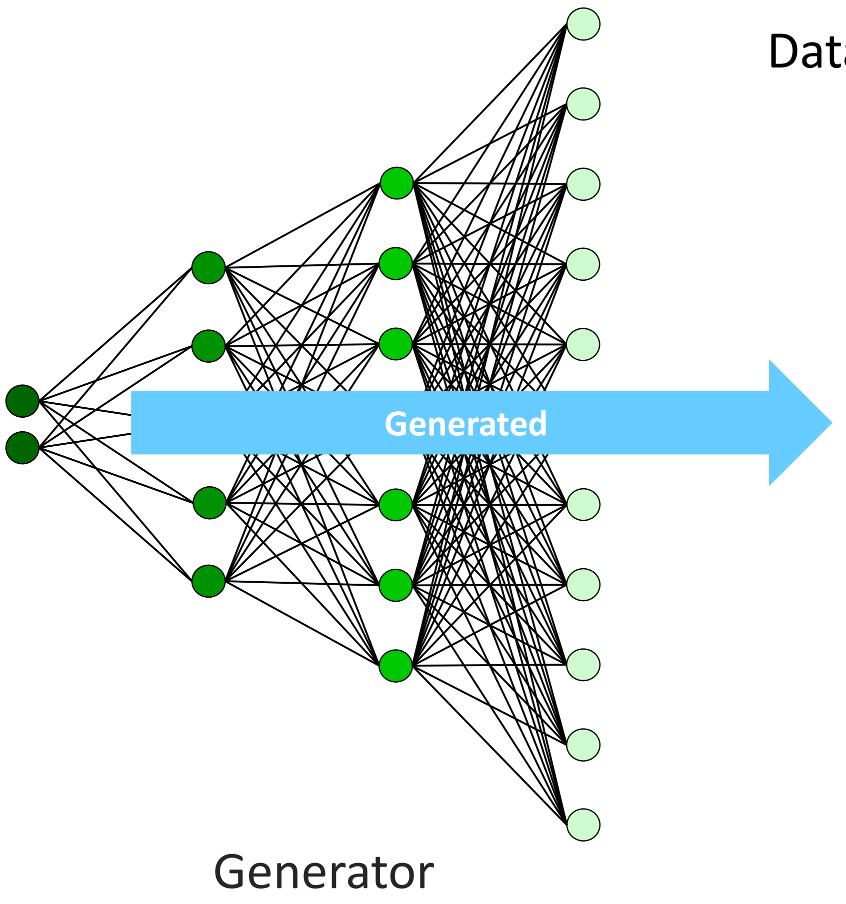
Generative Adversarial Networks (GAN)

- The idea behind Generative Adversarial Networks (GANs): train two different networks
  - Generator network: try to produce realistic-looking samples
  - Discriminator network: try to distinguish between real and fake data

- The generator network tries to fool the discriminator network

Generative Adversarial Networks (GAN)

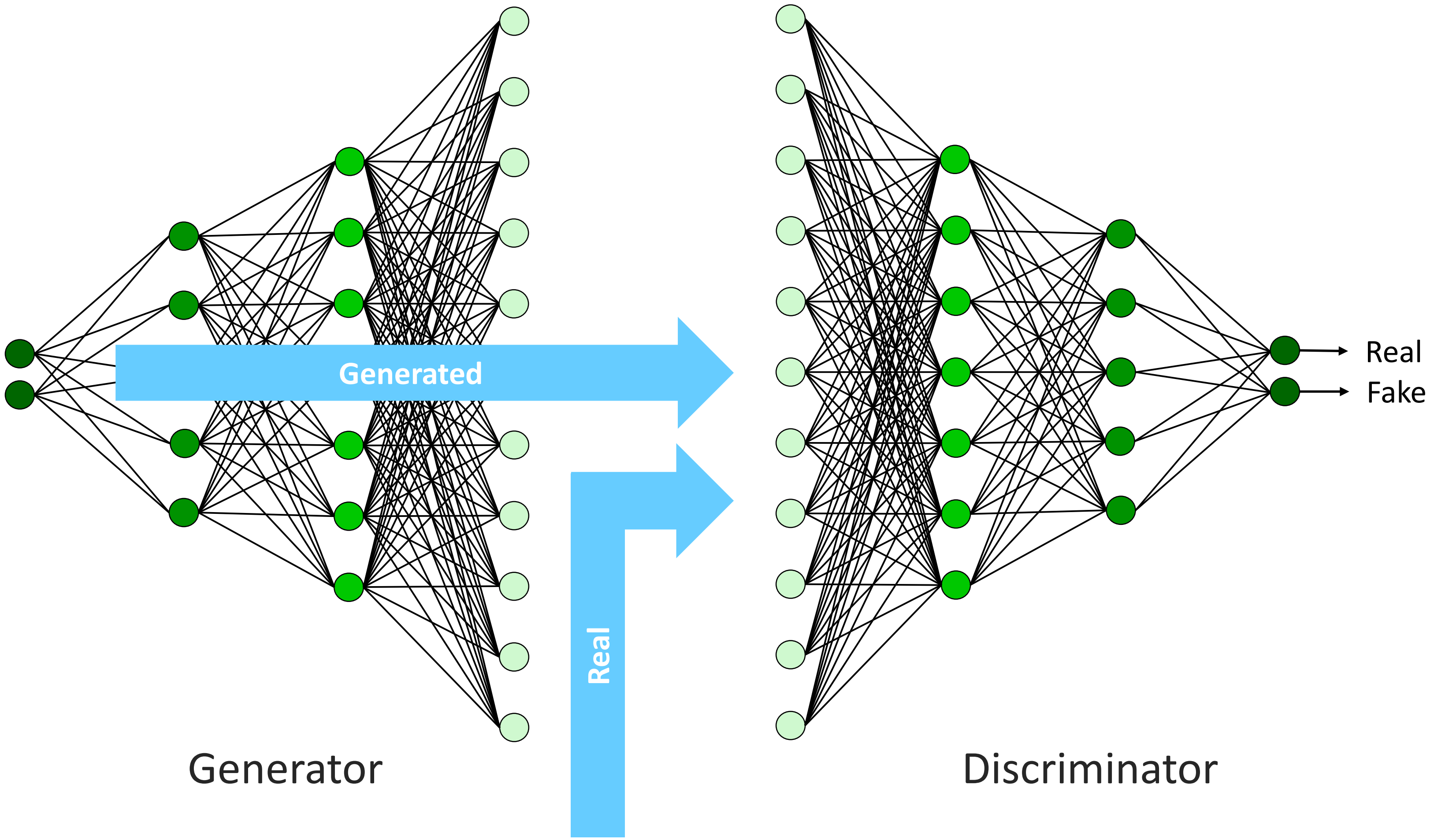
- Analogous to Turing Test



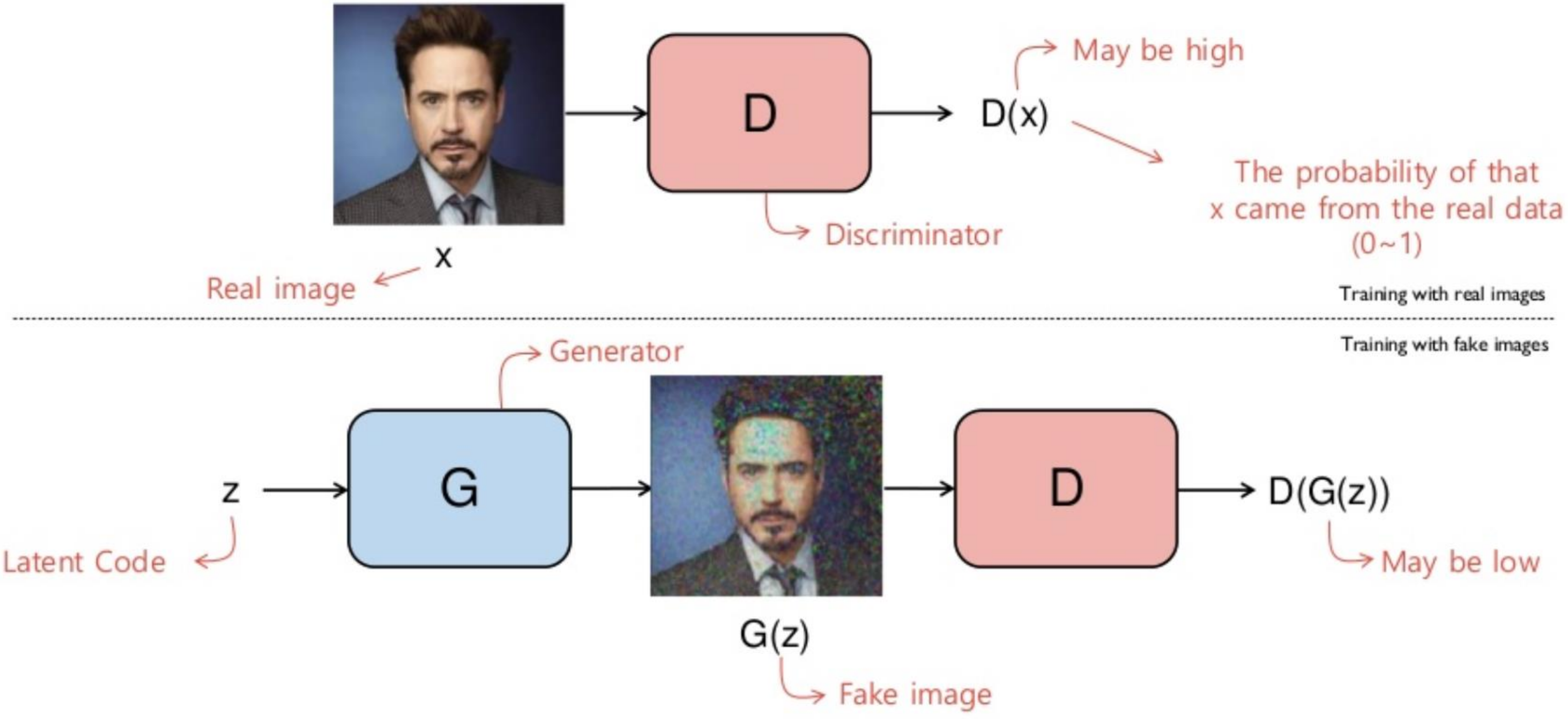
Data Generator

Generative Adversarial Networks (GAN)

- Analogous to Turing Test

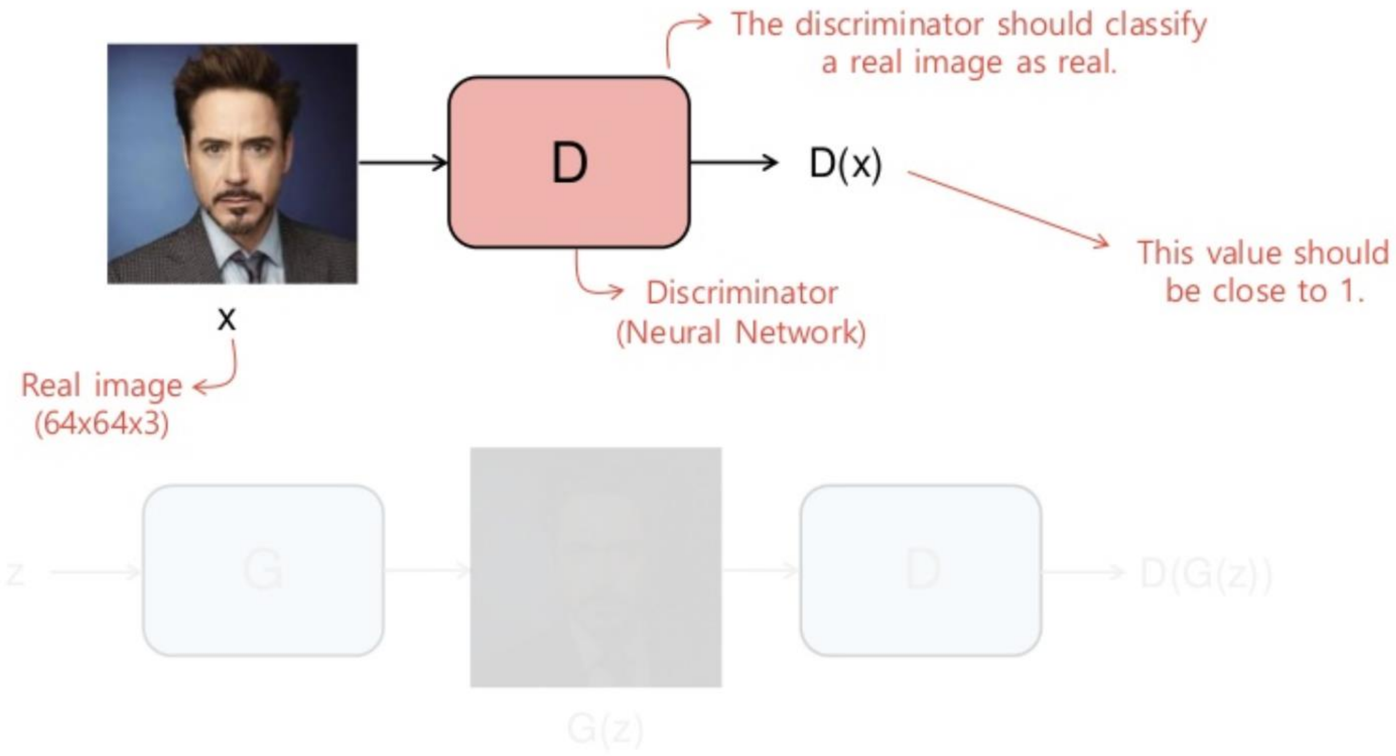


Intuition for GAN



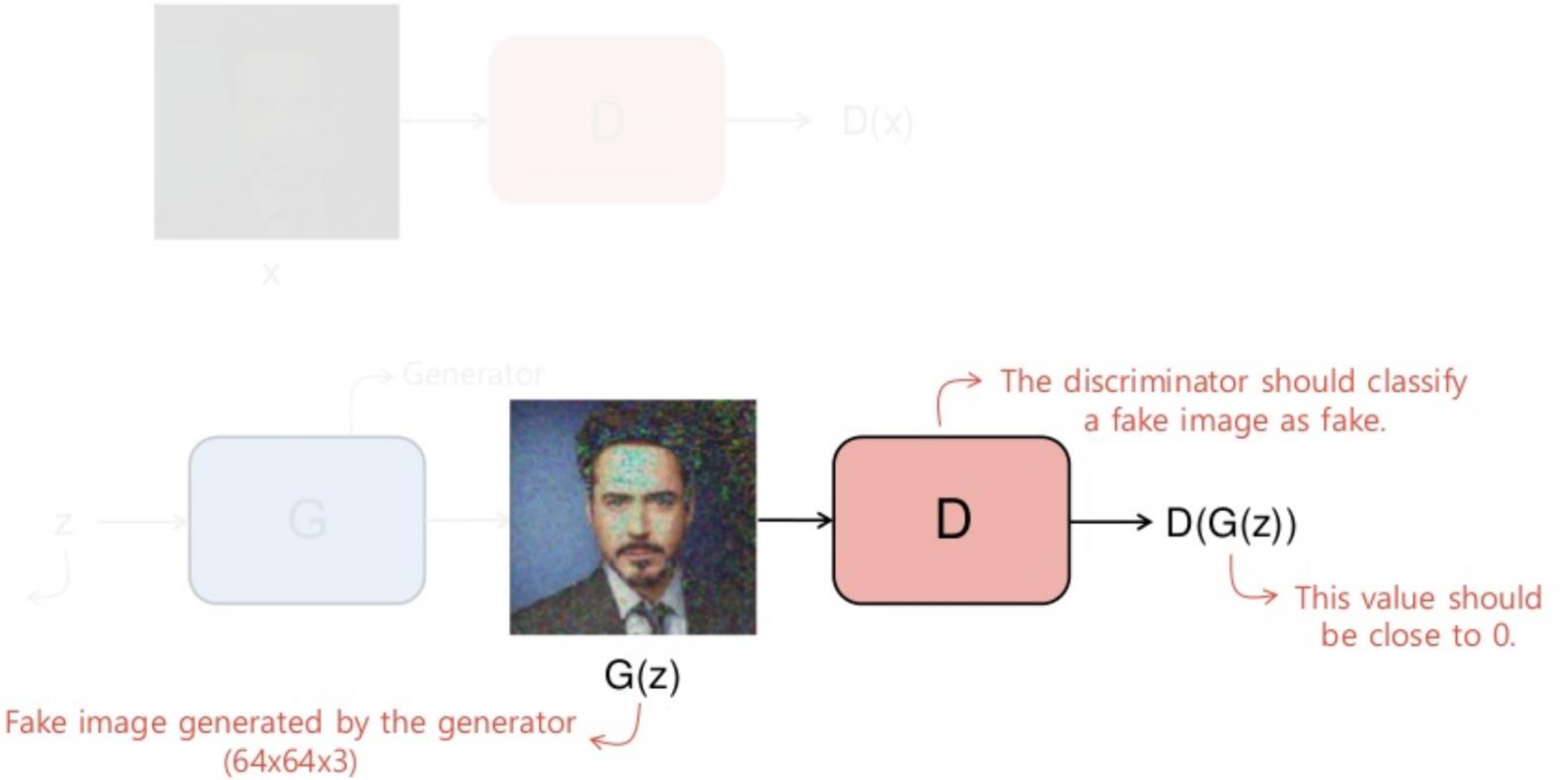
https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Discriminator Perspective (1/2)



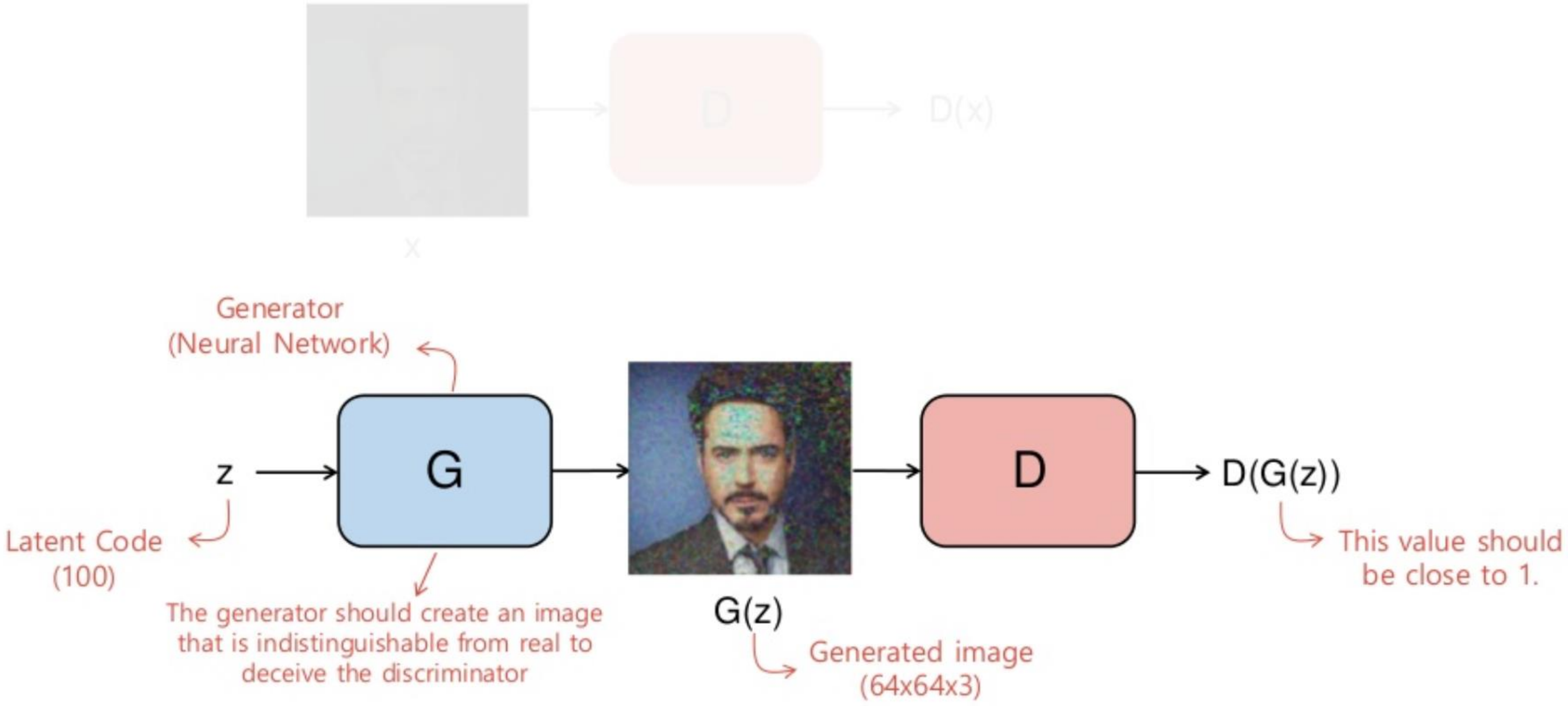
https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Discriminator Perspective (2/2)



https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Generator Perspective

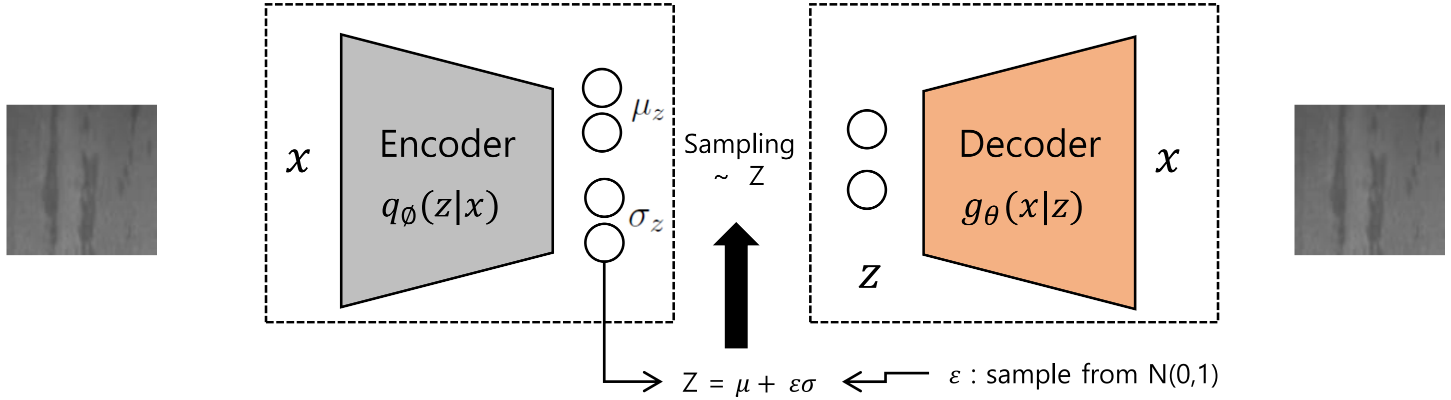


https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

실시간 강의자료

Variational auto-encoder

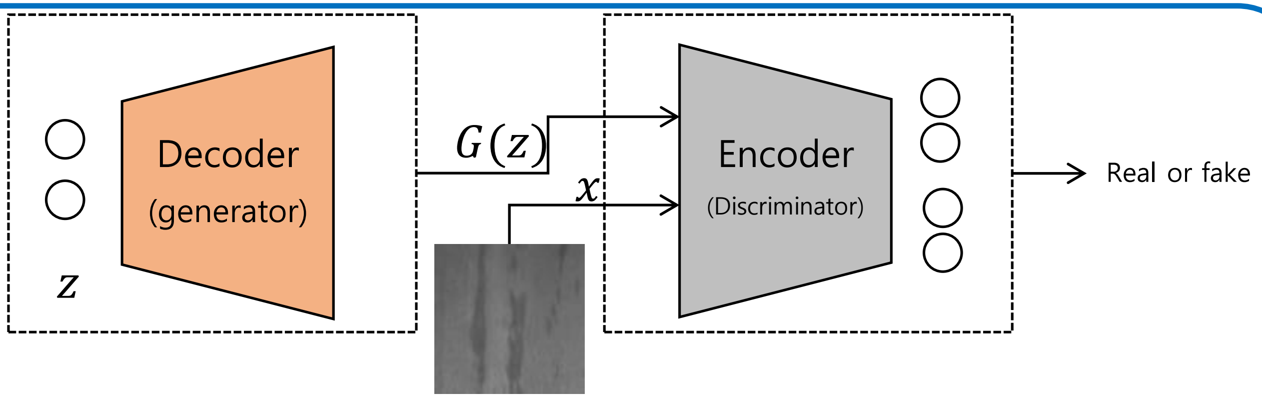
[VAE 모델 개요도 및 Loss]



$$L_i(\phi, \theta, x_i) = -\mathbb{E}_{q_\phi(z|x)}\left[\log\left(p_\theta(x_i|z)\right)\right] + KL(q_\phi(z|x_i) \parallel p(z))$$

Reconstruction error(복원 오차)   Regularization(제약조건)

[GAN 모델 개요도 및 Loss]



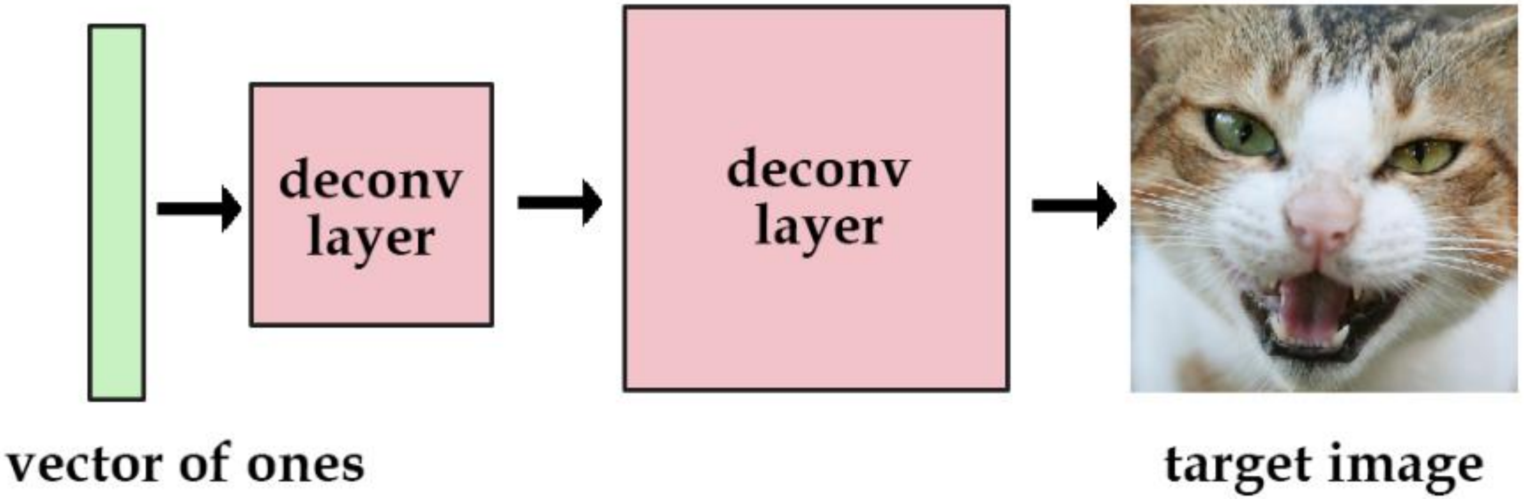
$$V(D, G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

$D^*, G^* = \min_G \max_D V(D, G)$ **minmax problem!!!**

Variational auto-encoder

- Step 1

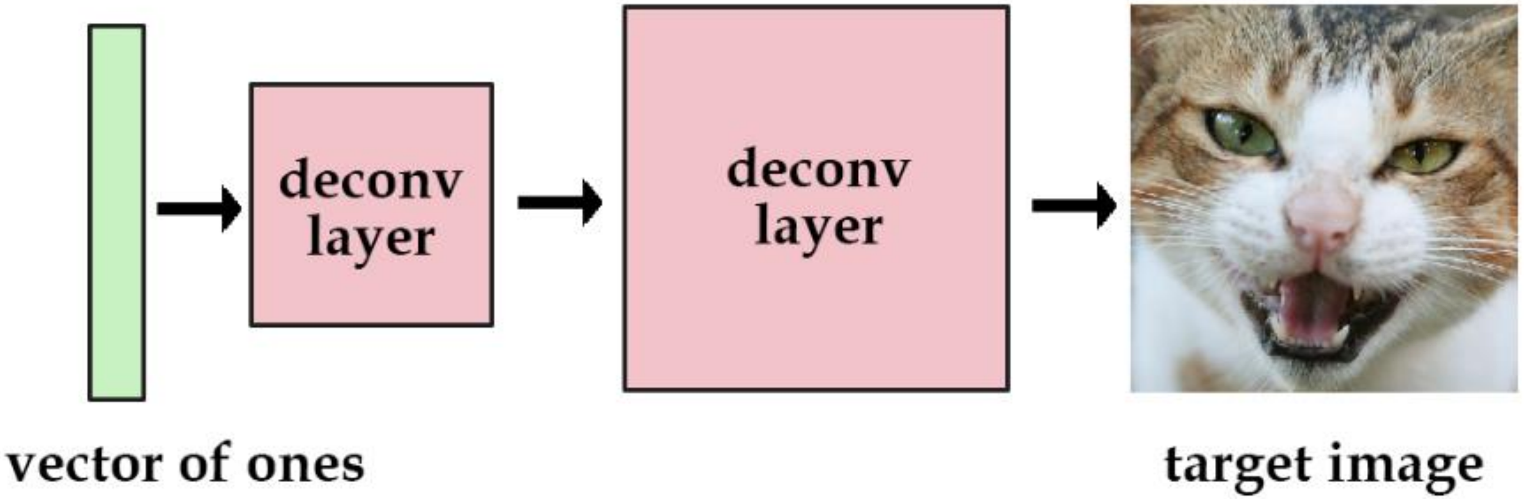
  : 고정된 입력에서 데이터 1개 생성



Variational auto-encoder

- Step 2

  : 고정된 입력에서 데이터 다수 생성

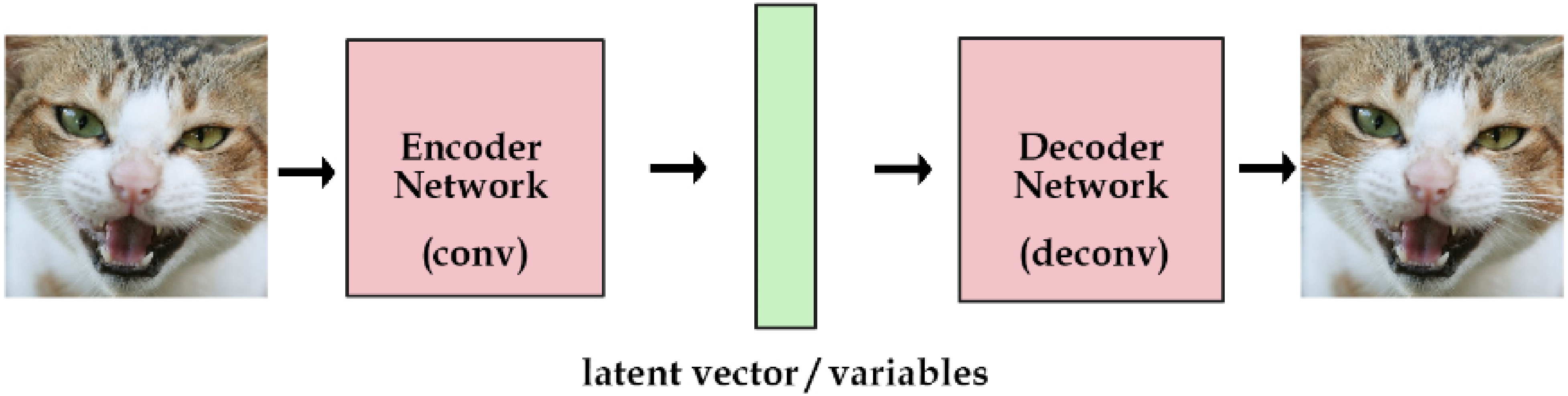


http://kvfrans.com/variational-autoencoders-explained/

Variational auto-encoder

- Step 3

  : 입력의 특징으로 latent vector 구성

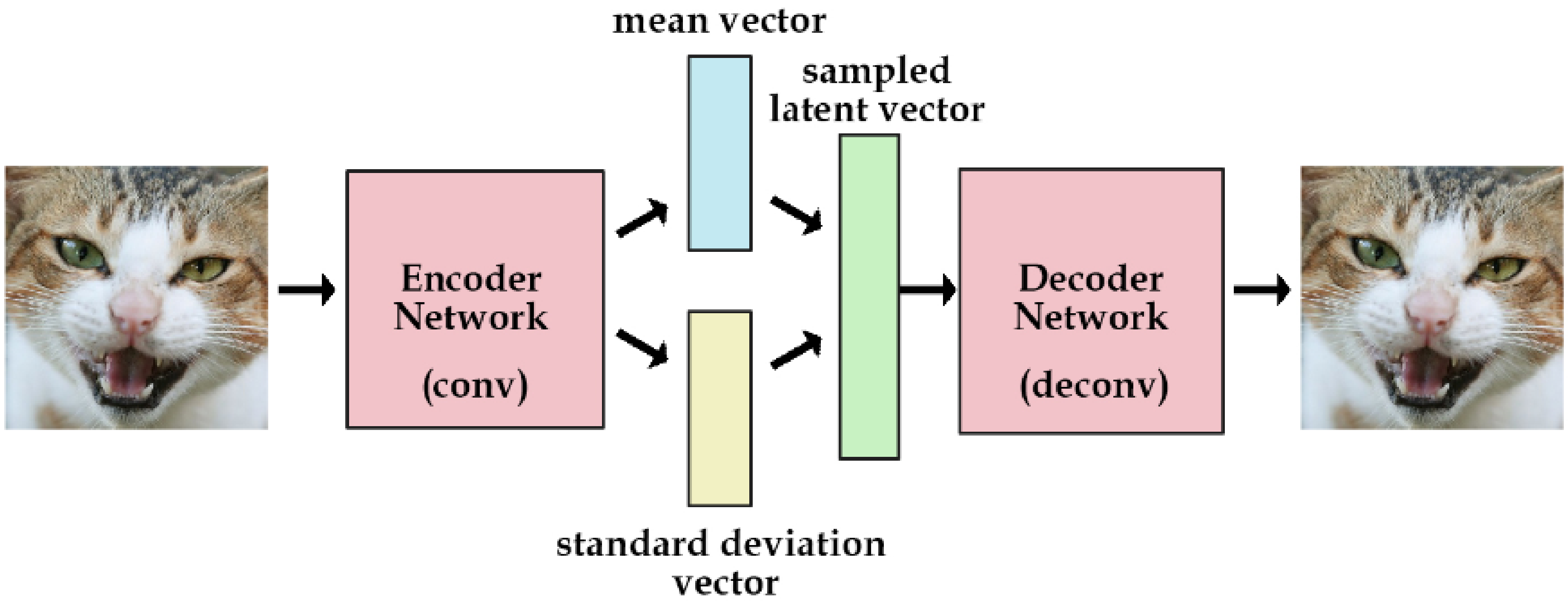


http://kvfrans.com/variational-autoencoders-explained/

Variational auto-encoder

- Step 4

  : Latent vector with simple distribution (e.g. uniform distribution or Gaussian distribution)

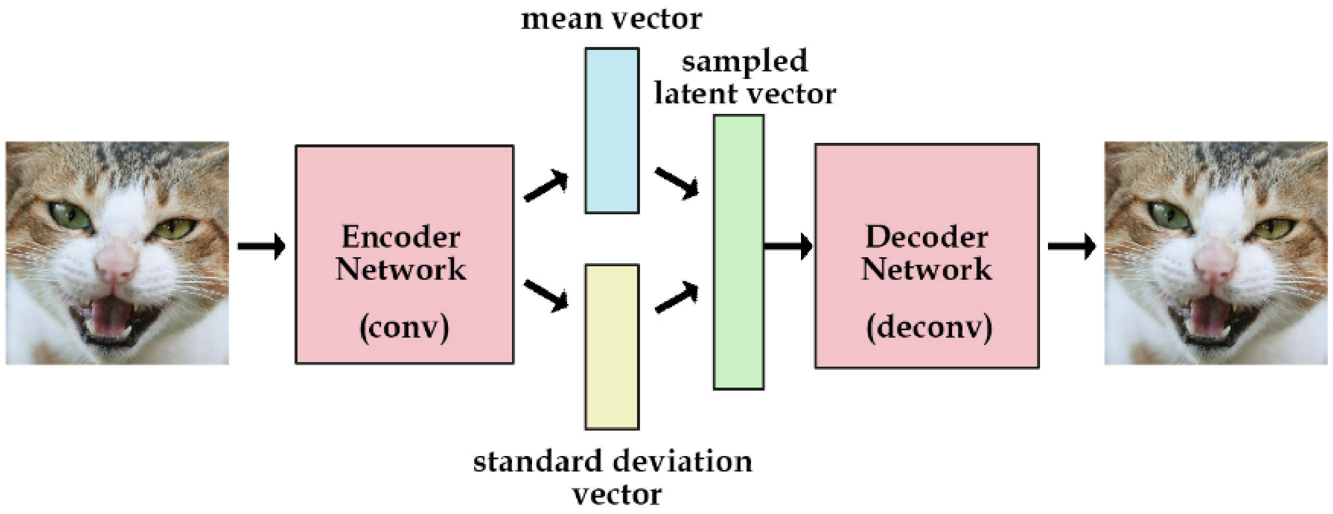


http://kvfrans.com/variational-autoencoders-explained/

Variational auto-encoder

- Step 5
  : KL-divergence

```
generation_loss = mean(square(generated_image - real_image))
latent_loss = KL-Divergence(latent_variable, unit_gaussian)
loss = generation_loss + latent_loss
```



Variational auto-encoder

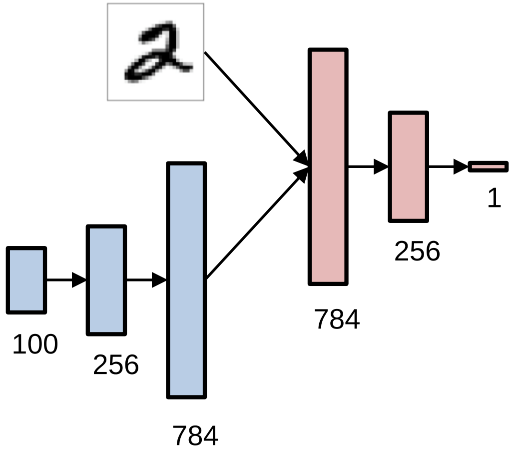
- VAE vs GAN

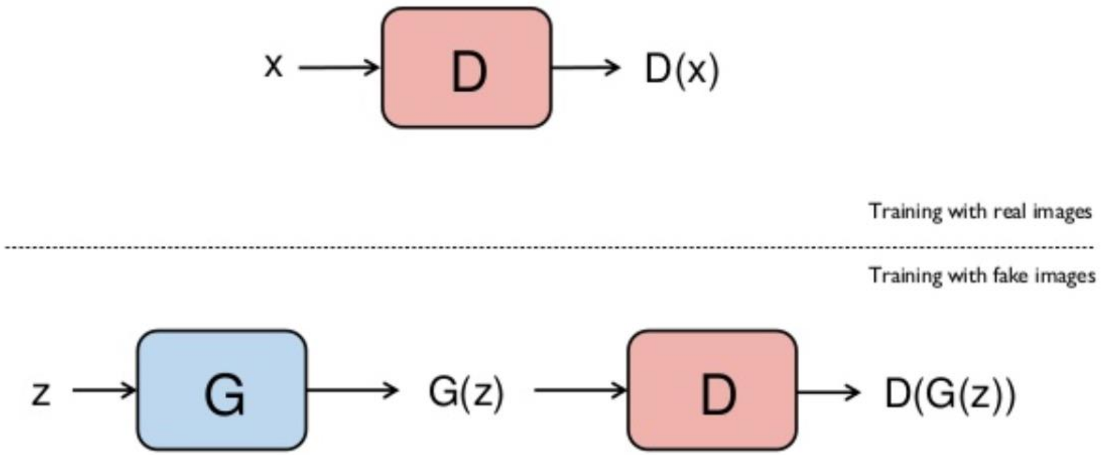
Model	Optimization	Image Quality	Generalization
VAE	• Stochastic gradient descent • Converge to local minimum • Easier	• Smooth • Blurry	• Tend to remember input images
GAN	• Alternating stochastic gradient descent • Converge to saddle points • Harder ❖ Model collapsing ❖ Unstable convergence	• Sharp • Artifact	• Generate new unseen images

그림 그리는 AI (Naver Clova 이활석) http://aliensunmin.github.io/project/accv16tutorial/media/generative.pdf

GAN Implementation in TensorFlow

TensorFlow Implementation

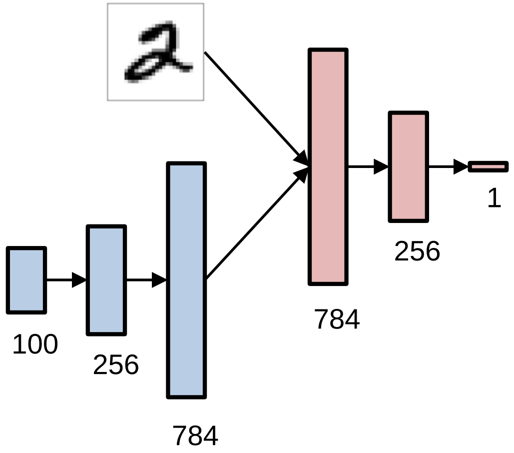


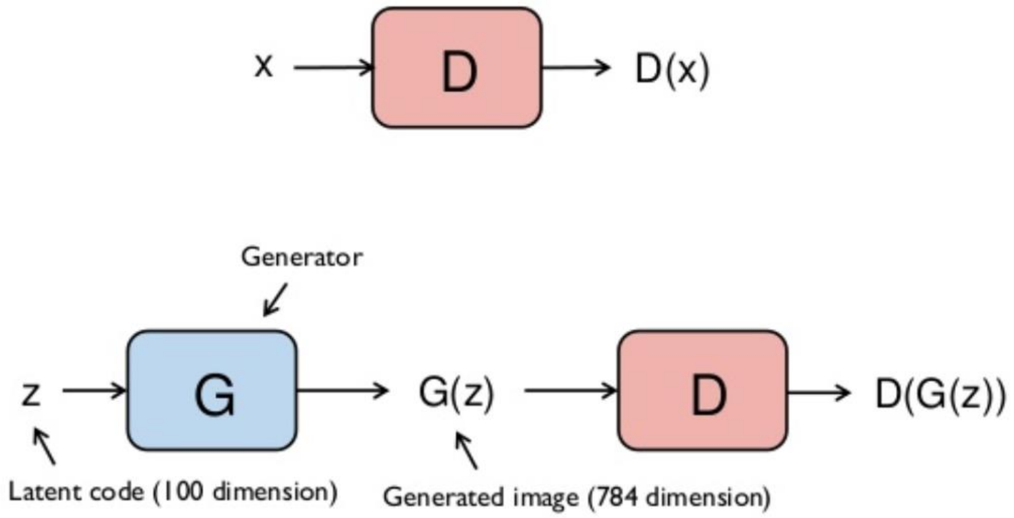


https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Generator

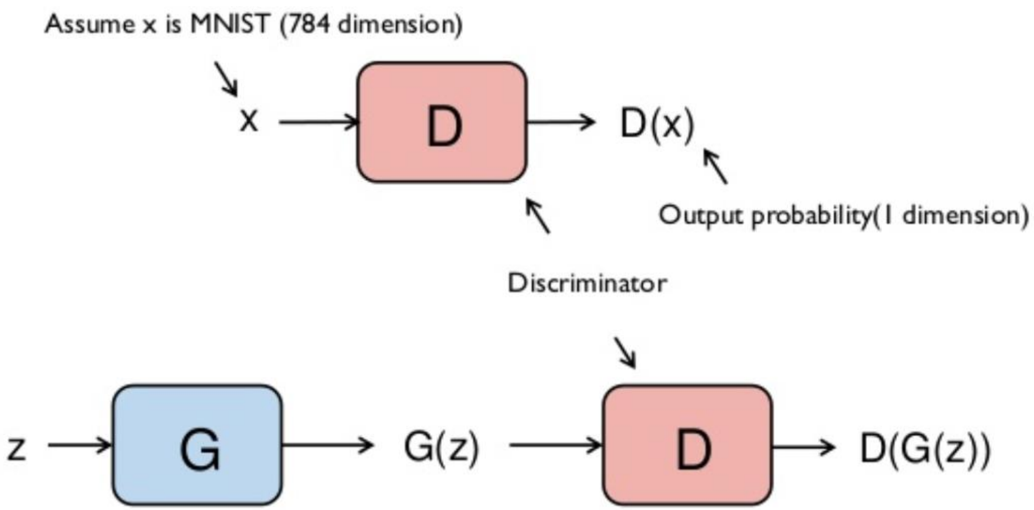
```
generator = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units = 256, input_dim = 100, activation = 'relu'),
    tf.keras.layers.Dense(units = 784, activation = 'sigmoid')
])
```

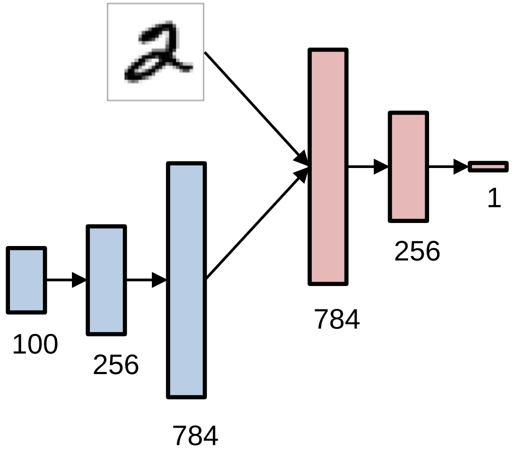




Discriminator

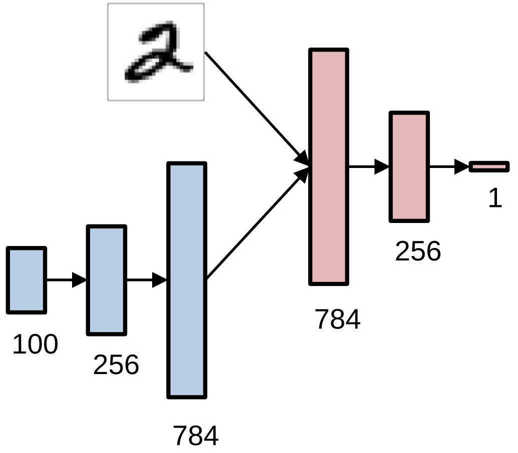
```
discriminator = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units = 256, input_dim = 784, activation = 'relu'),
    tf.keras.layers.Dense(units = 1, activation = 'sigmoid'),
])
```

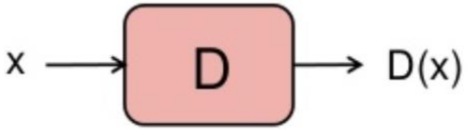


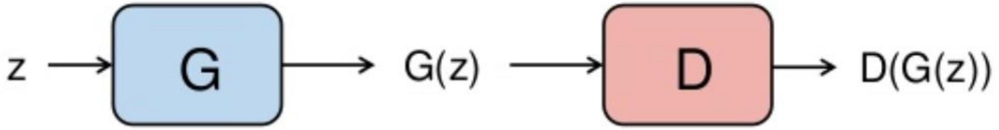


Combined

```
combined_input = tf.keras.layers.Input(shape = (100,))
generated = generator(combined_input)
discriminator.trainable = False
combined_output = discriminator(generated)

combined = tf.keras.models.Model(inputs = combined_input, outputs = combined_output)
```

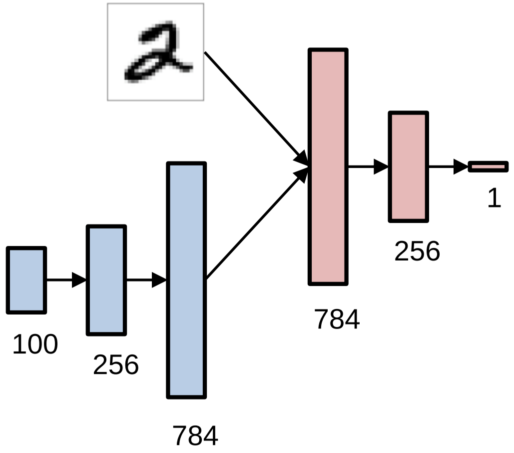


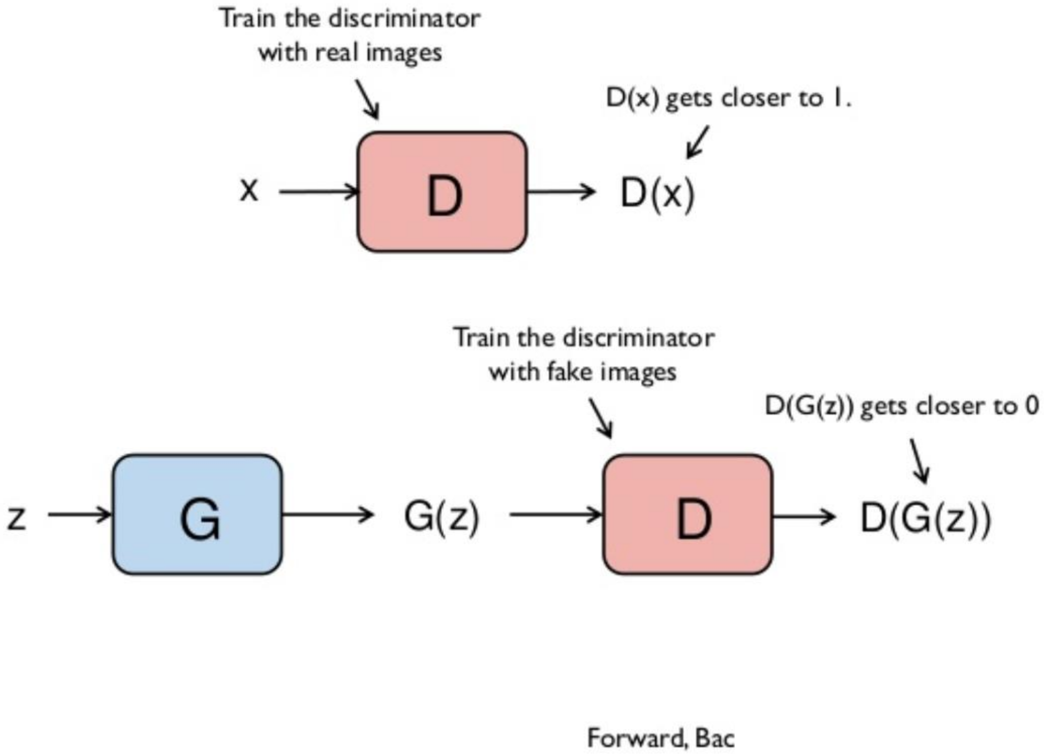




https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Training: Discriminator

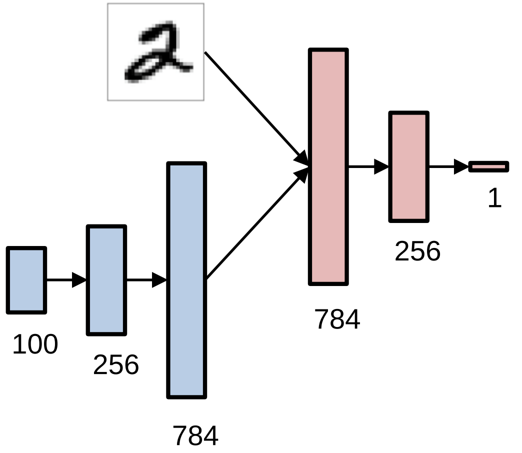


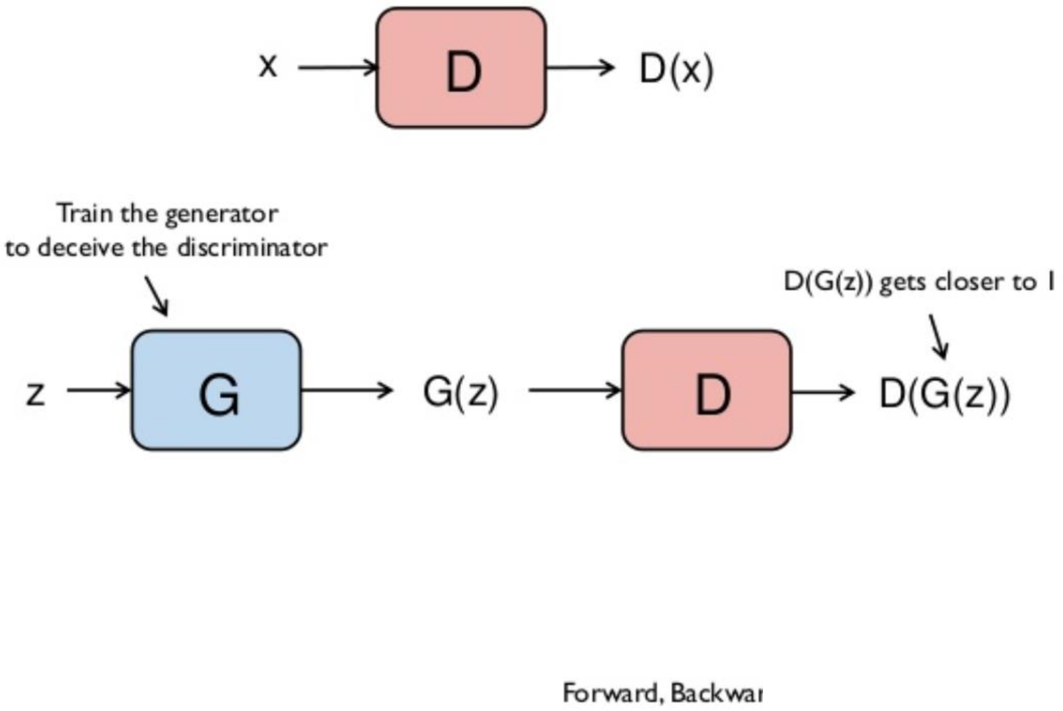



```python
n_iter = 5000
batch_size = 50

fake = np.zeros(batch_size)
real = np.ones(batch_size)

for i in range(n_iter):

    # Train Discriminator
    noise = make_noise(batch_size)
    generated_images = generator.predict(noise)

    idx = np.random.randint(0, train_x.shape[0], batch_size)
    real_images = train_x[idx]

    D_loss_real = discriminator.train_on_batch(real_images, real)
    D_loss_fake = discriminator.train_on_batch(generated_images, fake)
    D_loss = D_loss_real + D_loss_fake

    # Train Generator
    noise = make_noise(batch_size)
    G_loss = combined.train_on_batch(noise, real)
```


https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Training: Generator






```python
n_iter = 5000
batch_size = 50

fake = np.zeros(batch_size)
real = np.ones(batch_size)

for i in range(n_iter):

    # Train Discriminator
    noise = make_noise(batch_size)
    generated_images = generator.predict(noise)

    idx = np.random.randint(0, train_x.shape[0], batch_size)
    real_images = train_x[idx]

    D_loss_real = discriminator.train_on_batch(real_images, real)
    D_loss_fake = discriminator.train_on_batch(generated_images, fake)
    D_loss = D_loss_real + D_loss_fake

    # Train Generator
    noise = make_noise(batch_size)
    G_loss = combined.train_on_batch(noise, real)
```

Generated Images

```
Discriminator Loss:  0.371719628572464
Generator Loss:  2.474647283554077
```

```
Discriminator Loss:  0.38926680386066437
Generator Loss:  2.30230712890625
```