Generative Adversarial Networks (GANs)

Prof. Seungchul Lee

POSTECH

Source

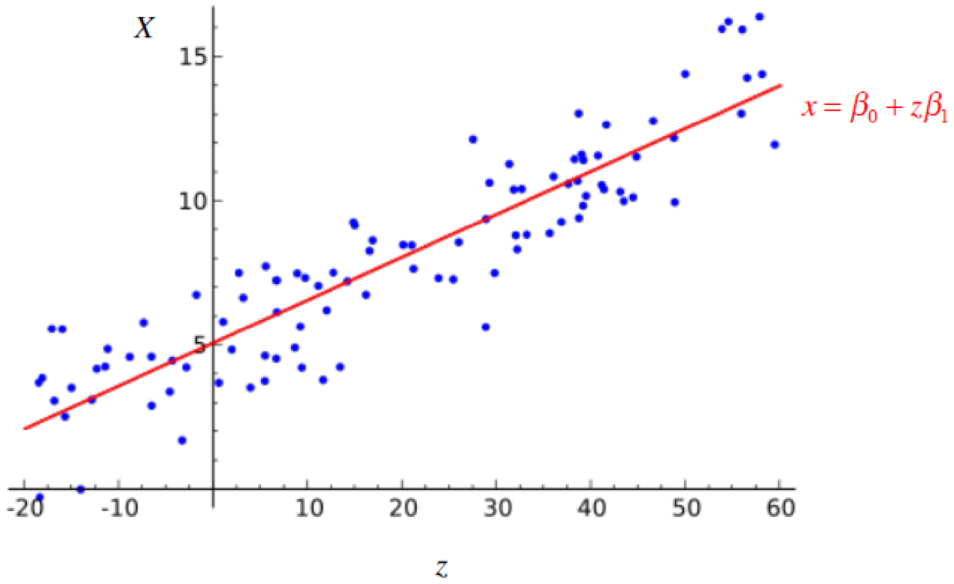
- 1시간만에 GAN(Generative Adversarial Network) 완전 정복하기
  - by 최윤제 (고려대 석사생)
  - YouTube: https://www.youtube.com/watch?v=odpjk7_tGY0
  - Slides: https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

- CSC321 Lecture 19: GAN
  - By Prof. Roger Grosse at Univ. of Toronto
  - http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/

- CS231n: CNN for Visual Recognition
  - Lecture 13: Generative Models
  - By Prof. Fei-Fei Li at Stanford University
  - http://cs231n.stanford.edu/

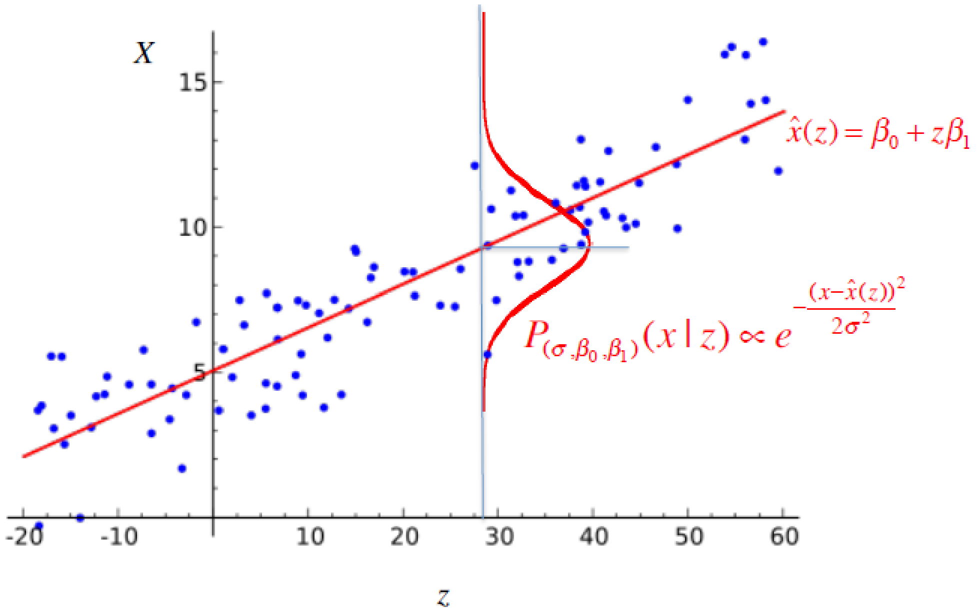
Recap: Linear Regression

- Most people think of linear regression as points and a straight line:

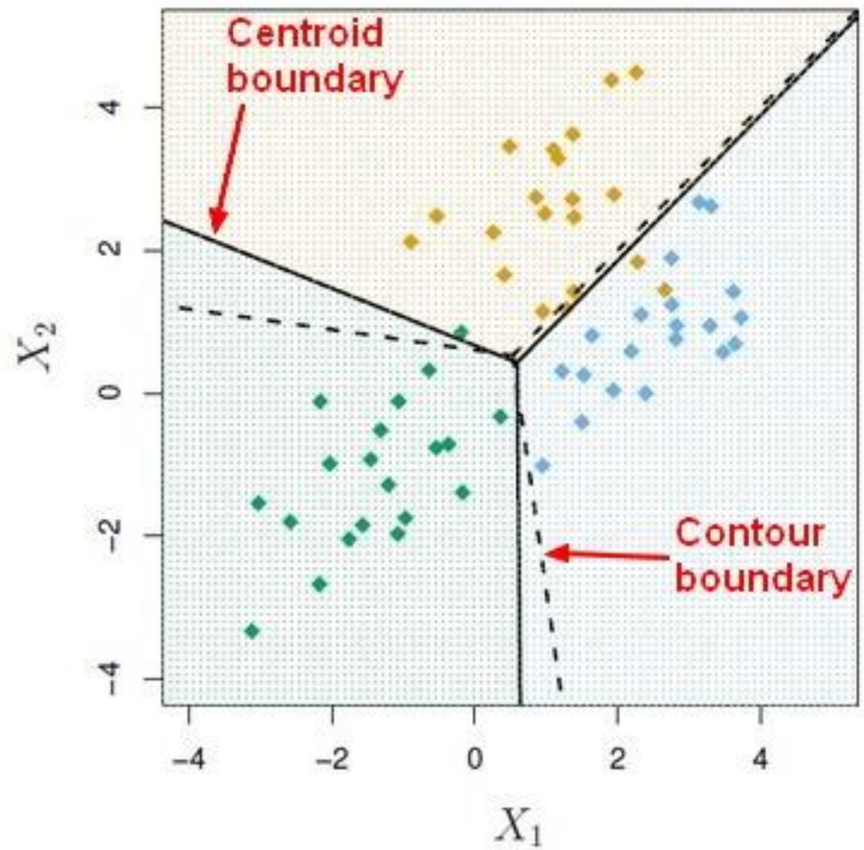


Recap: Linear Regression

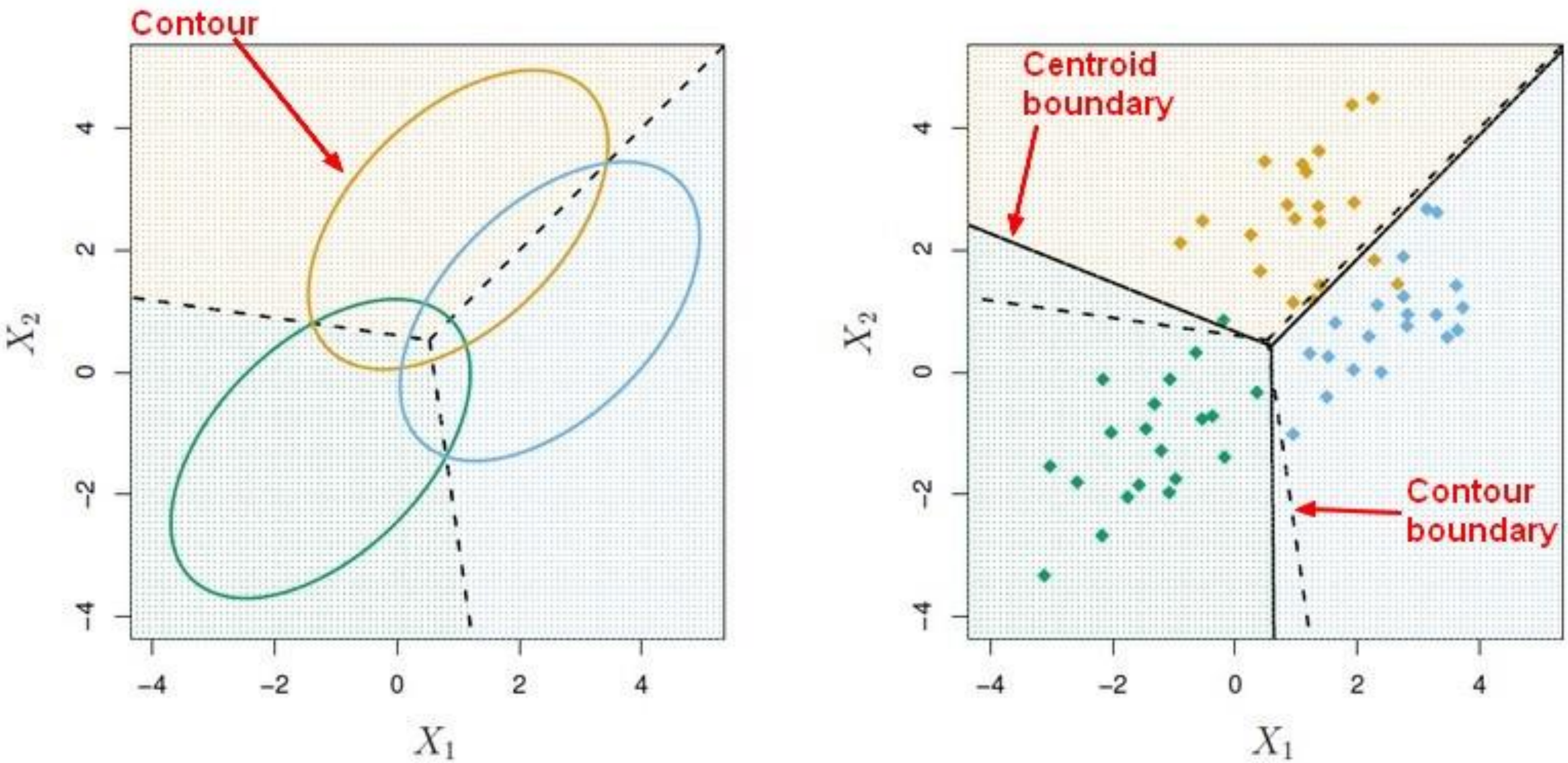
- Statisticians additionally have $P_\theta(X|Z)$
- Benefits of having an error model:
  - How likely is a data point
  - Confidence bounds
  - Compare models



Recap: Linear Classifier



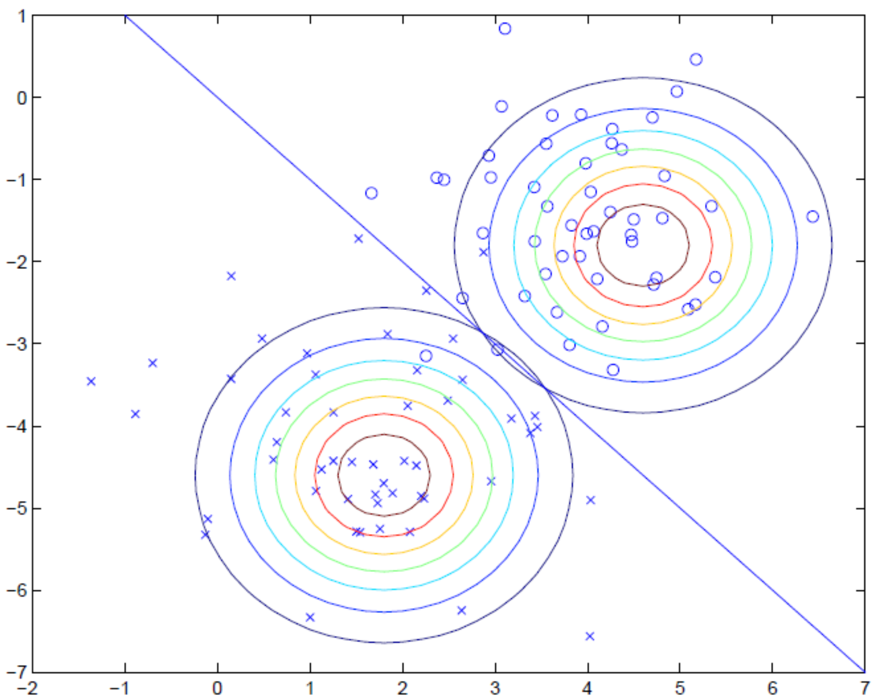
Recap: Linear Classifier



Recap: Linear Classifier

- Think about how data is generated

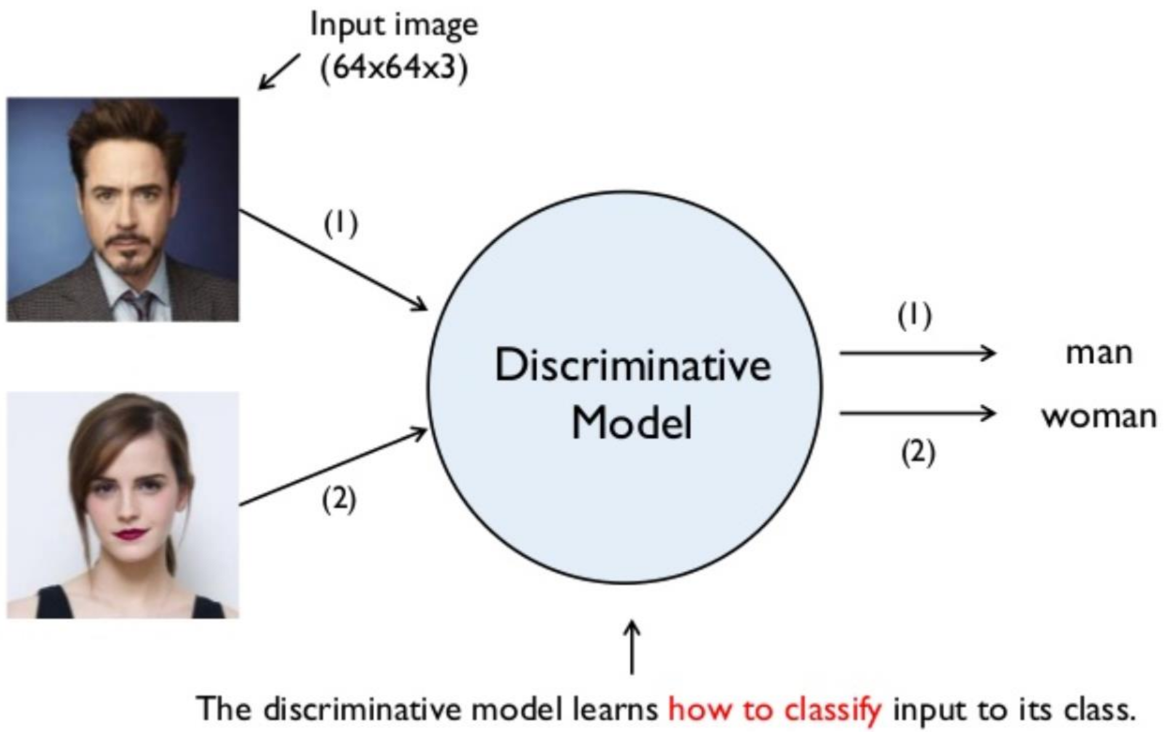
$$
\begin{aligned}
y &\sim \text{Bernoulli}(\phi) \\
x|y=0 &\sim \mathcal{N}(\mu_0, \Sigma) \\
x|y=1 &\sim \mathcal{N}(\mu_1, \Sigma)
\end{aligned}
$$



POSTECH

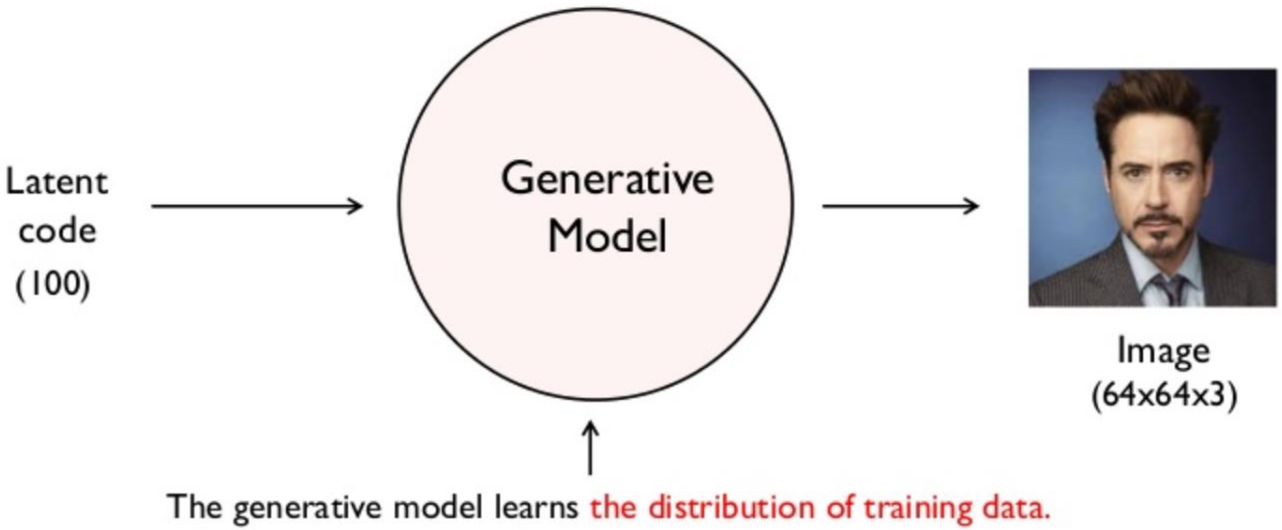
Supervised Learning

- Discriminative model



Unsupervised Learning

- Generative model



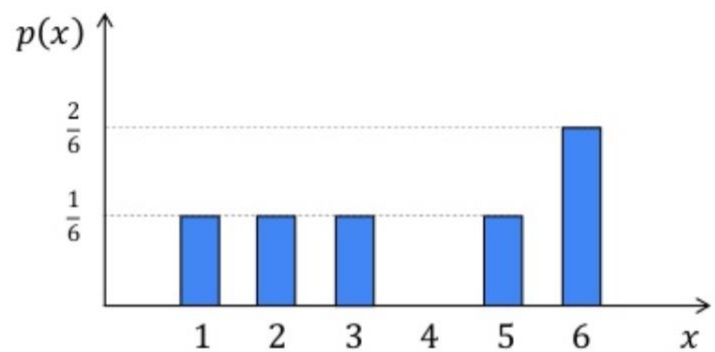
Probability Distribution

Probability Basics (Review)

Random variable

X	1	2	3	4	5	6
$P(X)$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{0}{6}$	$\frac{1}{6}$	$\frac{2}{6}$

Probability mass function



Probability Distribution

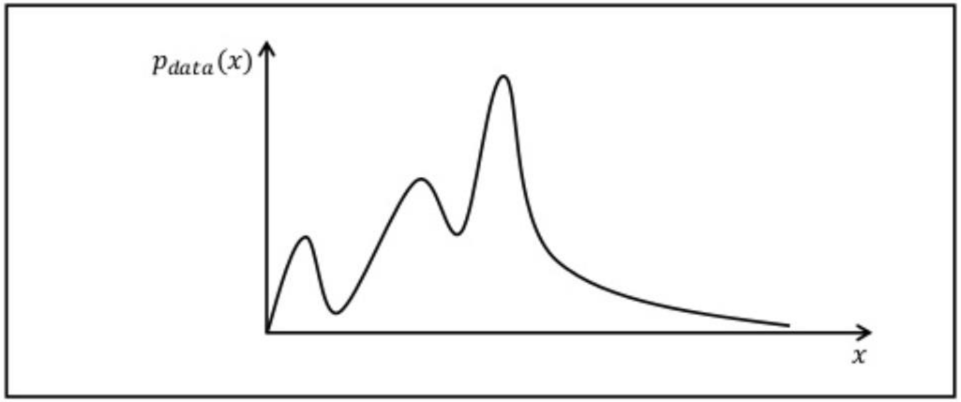
What if x is actual images in the training data?

At this point, x can be represented as a (for example) 64x64x3 dimensional vector.

Probability Distribution

Probability density function

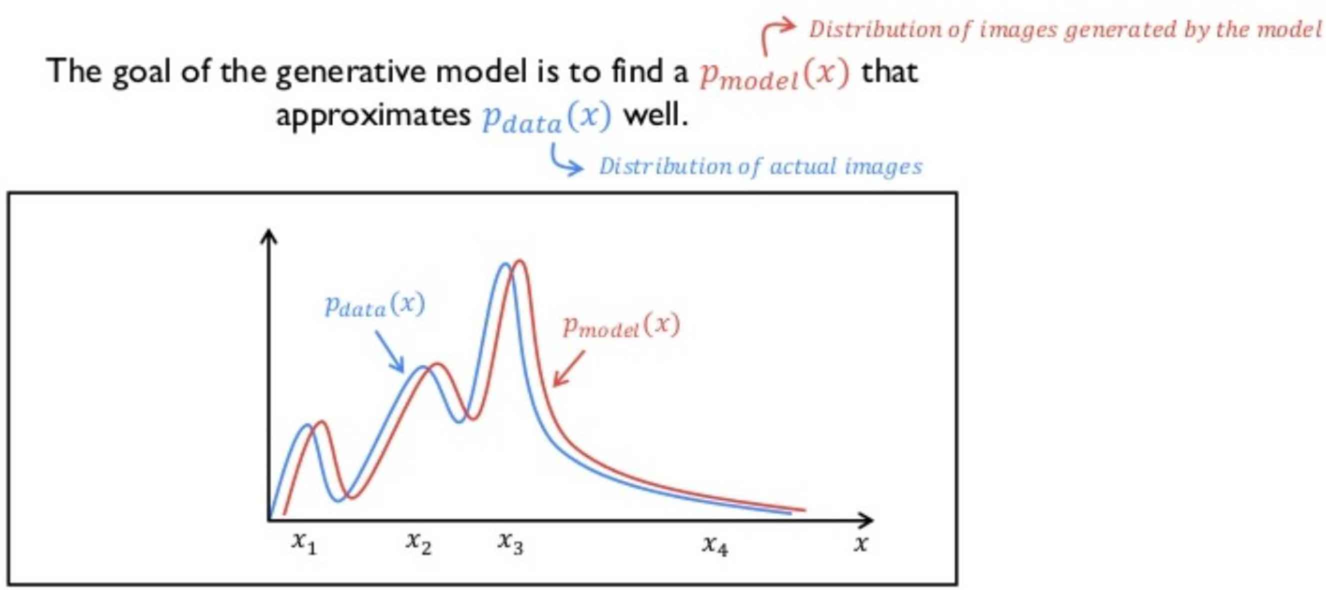
There is a $p_{data}(x)$ that represents the distribution of actual images.



POSTECH

Density Estimation

- Probability density estimation problem

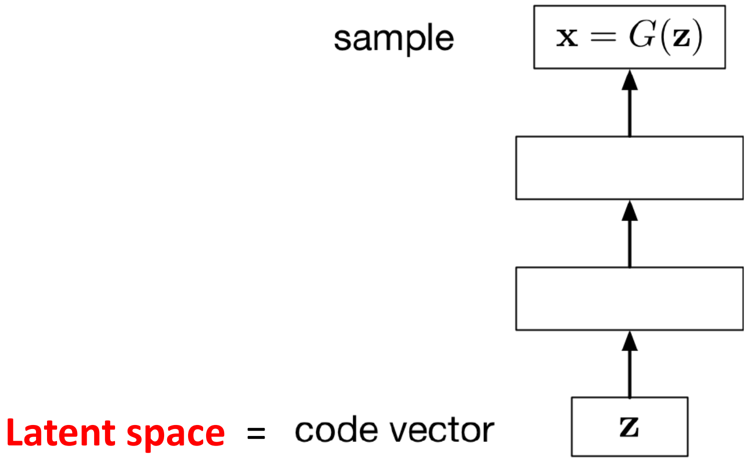


- If $P_{model}(x)$ can be estimated as close to $P_{data}(x)$, then data can be generated by sampling from $P_{model}(x)$

Generative Models from Lower Dim.

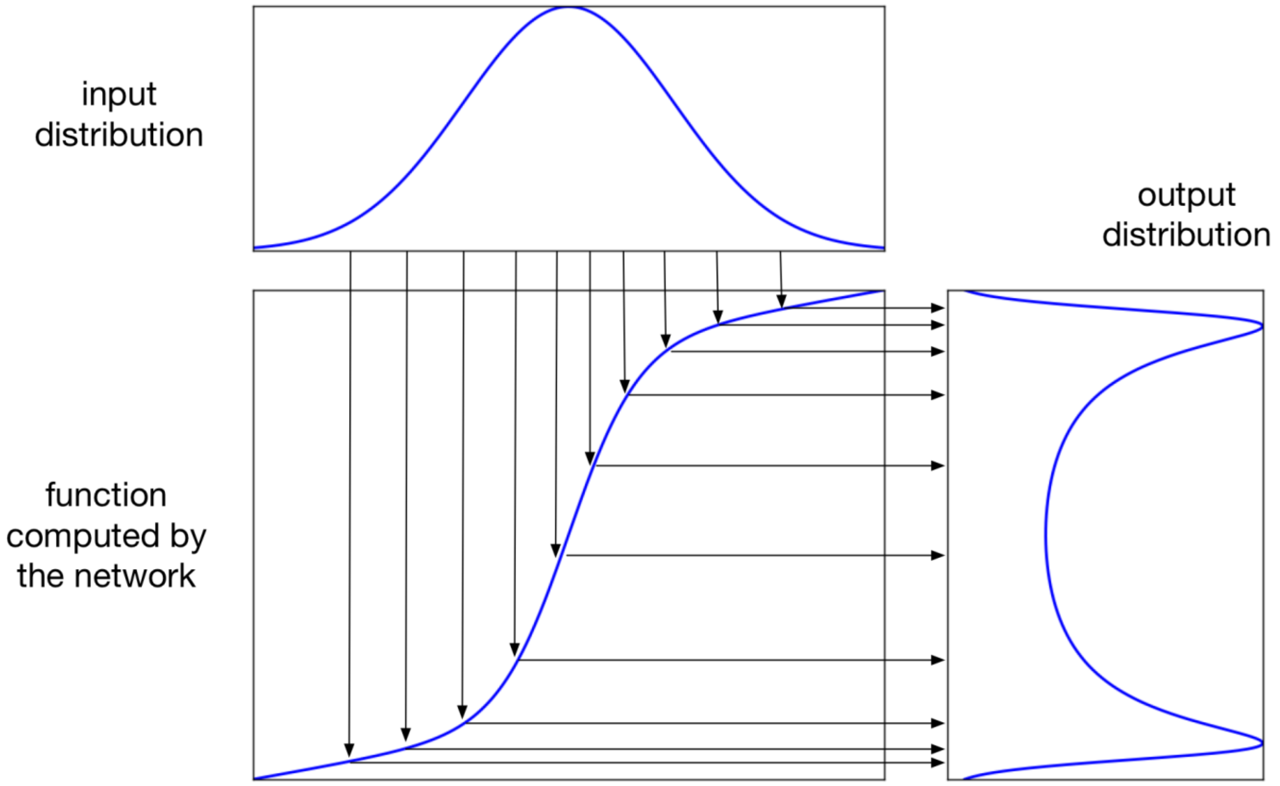
- Learn transformation via a neural network

- 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 $x = G(z)$



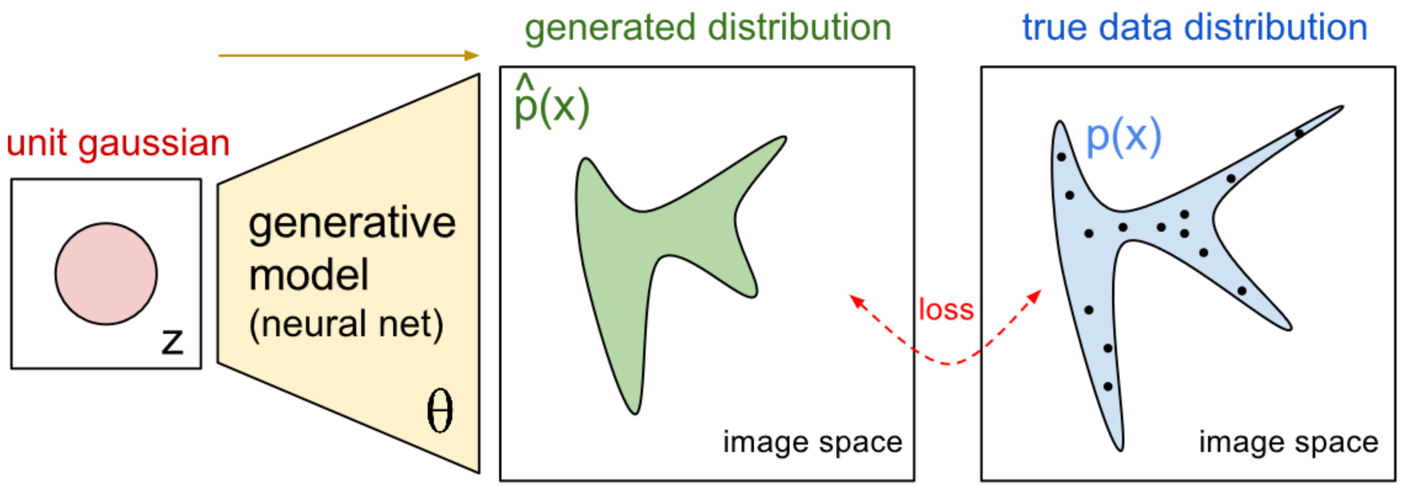
Deterministic Transformation (by Network)

- 1-dimensional example:



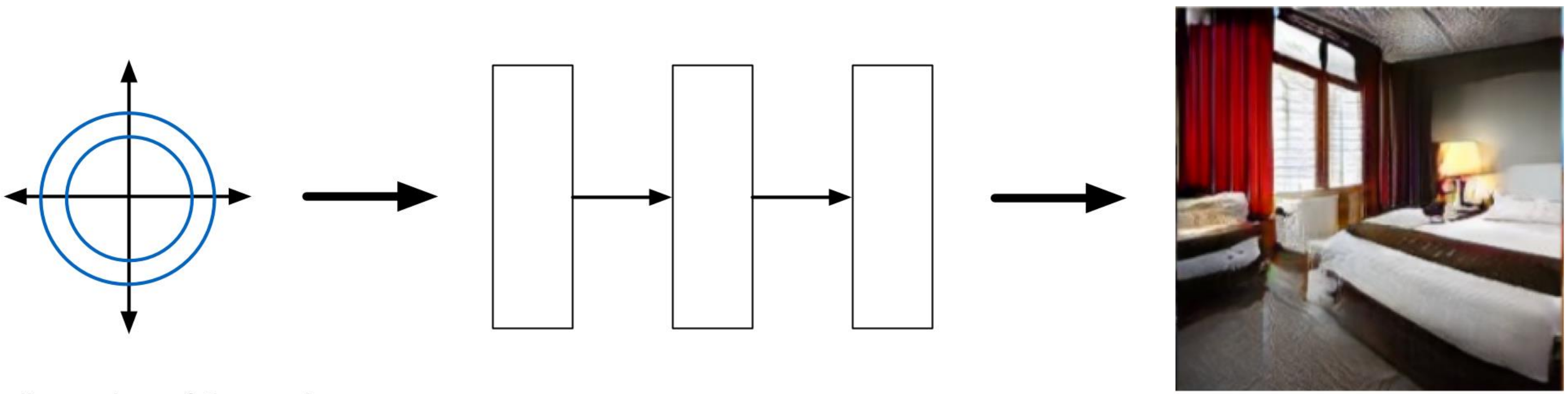
Deterministic Transformation (by Network)

- High dimensional example:



Prob. Density Function by Deep Learning

- Generative model of image



Each dimension of the code vector is sampled independently from a simple distribution, e.g. Gaussian or uniform.

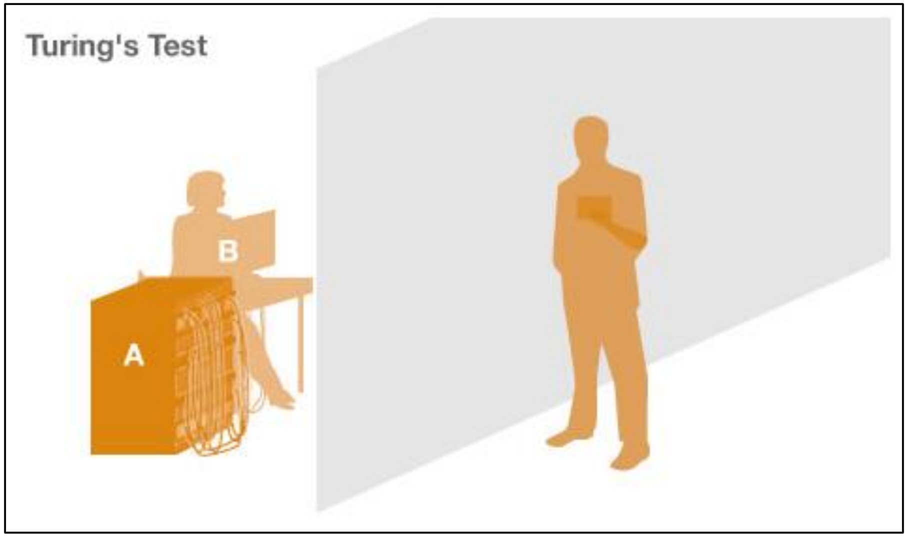
This is fed to a (deterministic) generator network.

The network outputs an image.

POSTECH

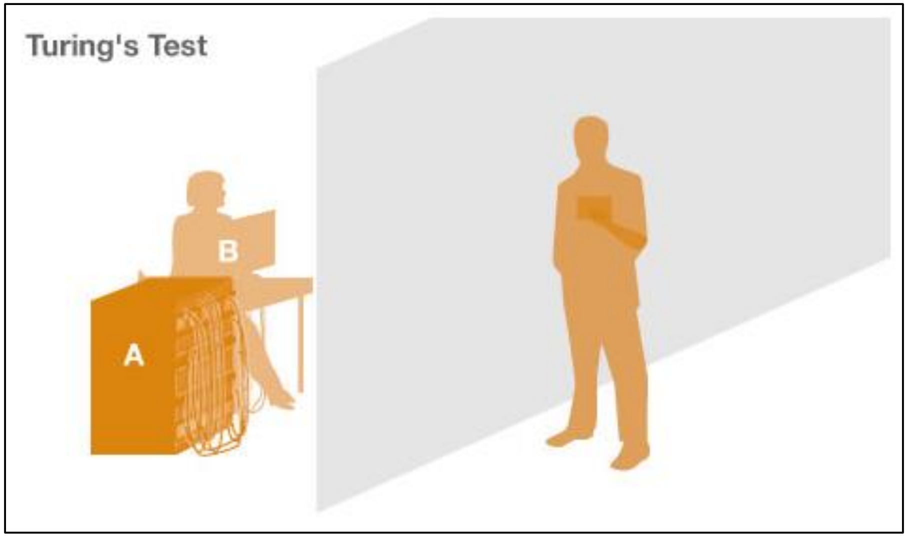
Generative Adversarial Networks (GANs)

- 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

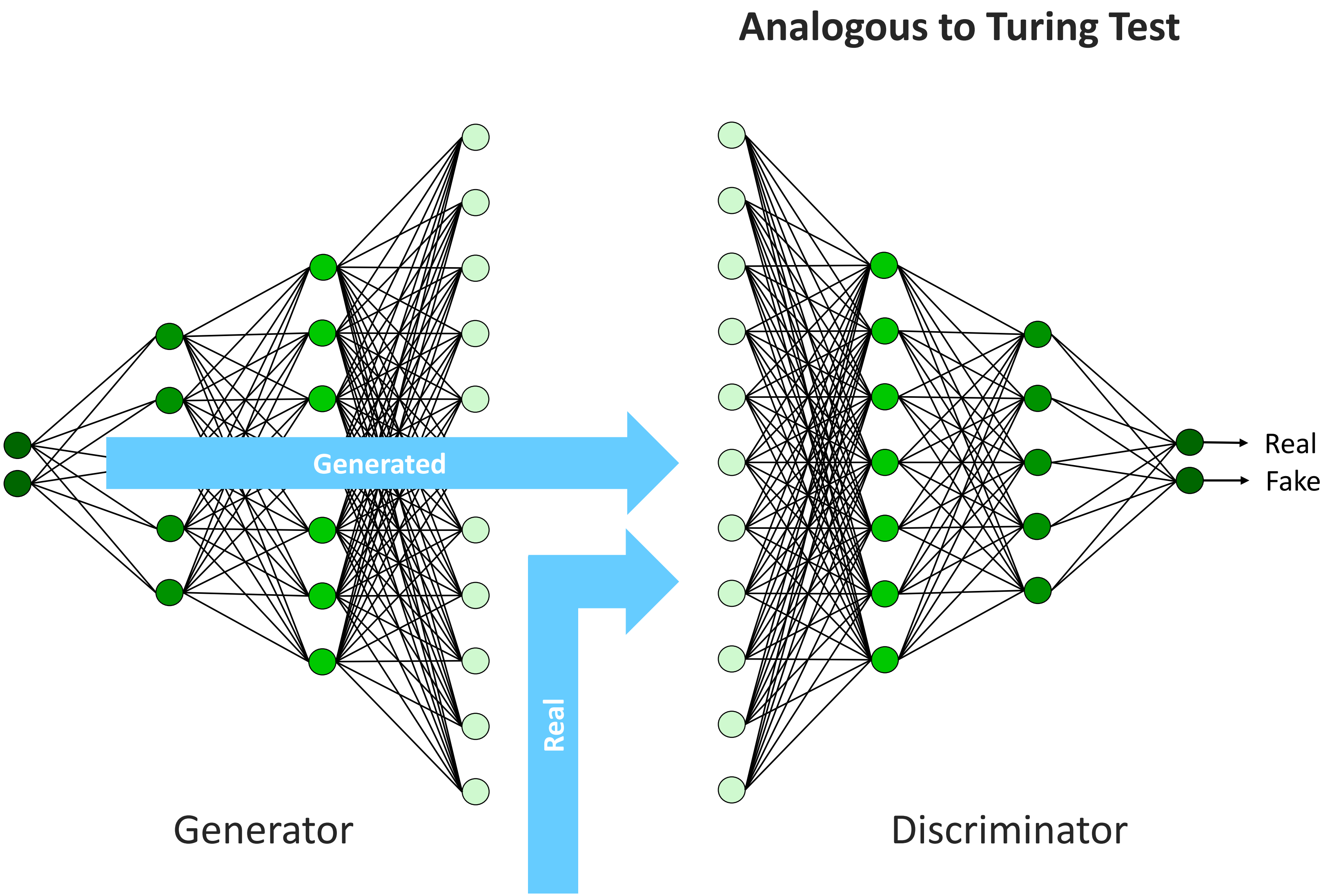


Turing Test

- 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)



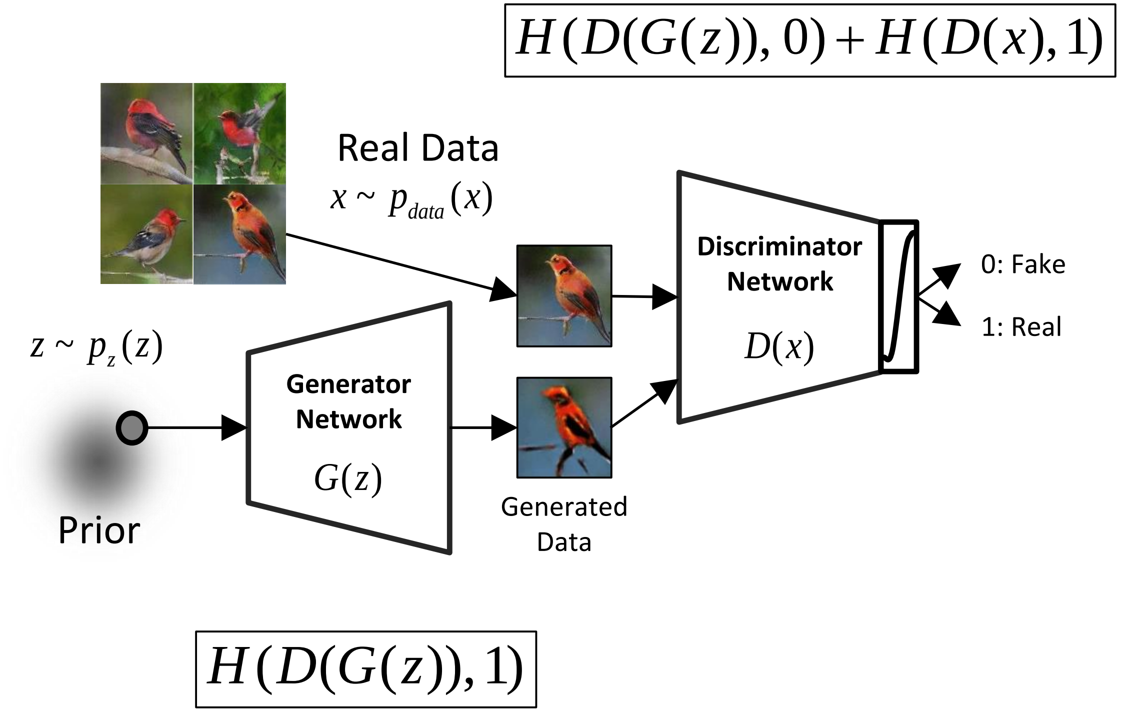
POSTECH

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

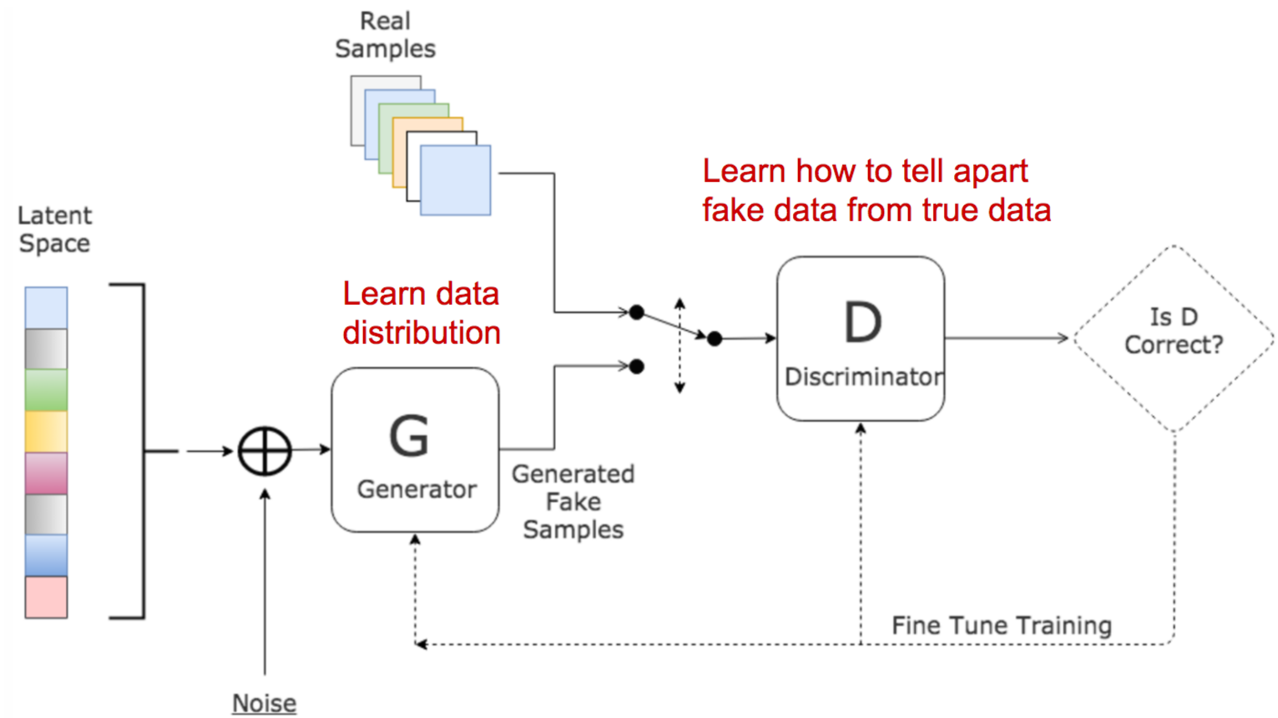
Generative Adversarial Networks (GAN)

- How to generate data?
  - Train through competition
  - Generator vs. Discriminator

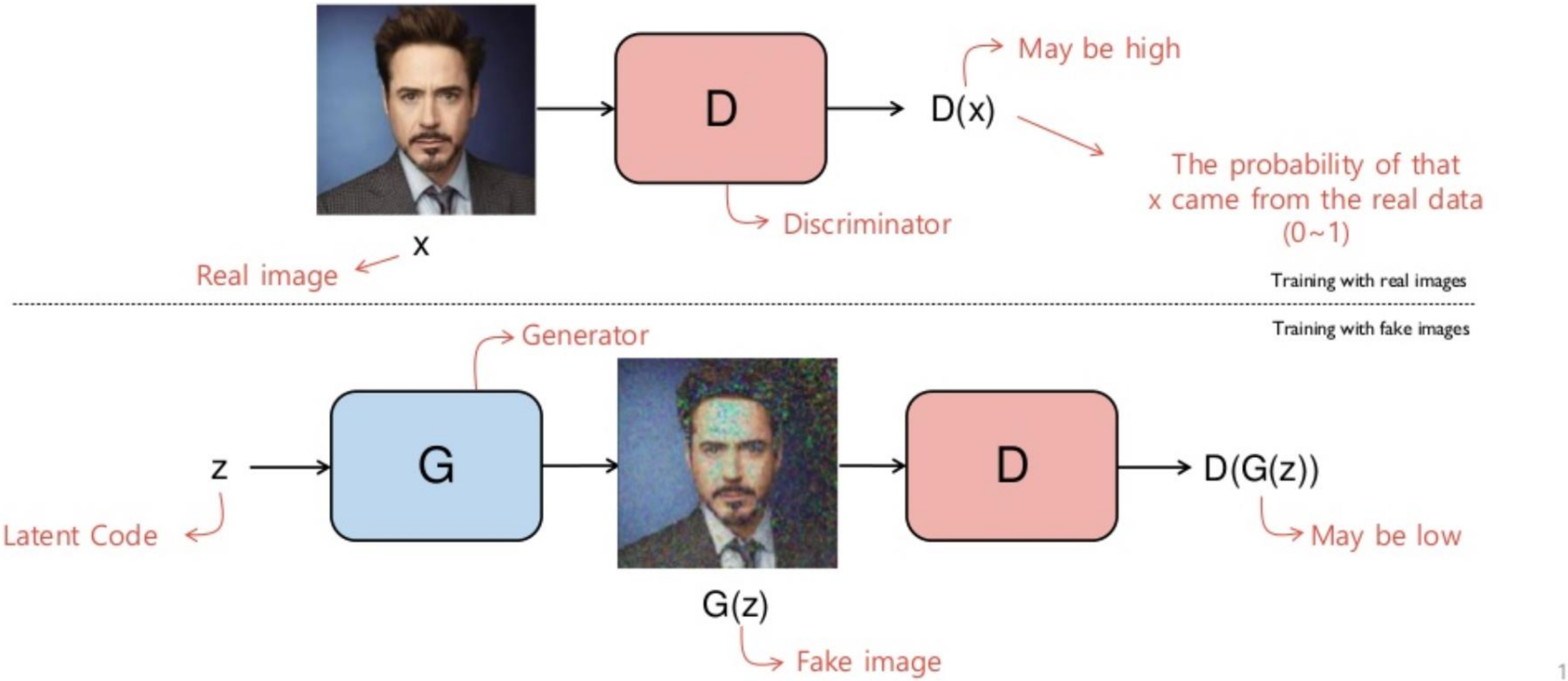


Generative Adversarial Networks (GAN)

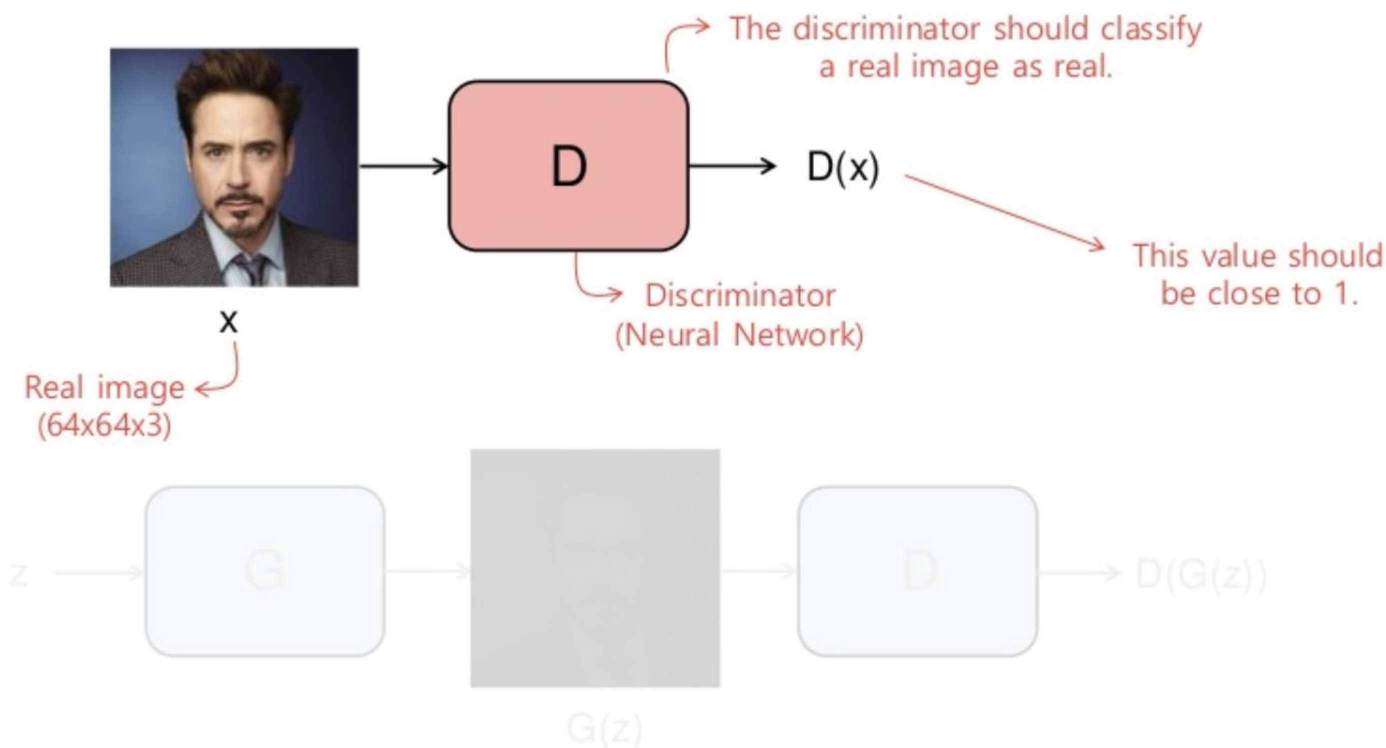
- How to generate data?
  – Train through competition
  – Generator vs. Discriminator



Intuition for GAN

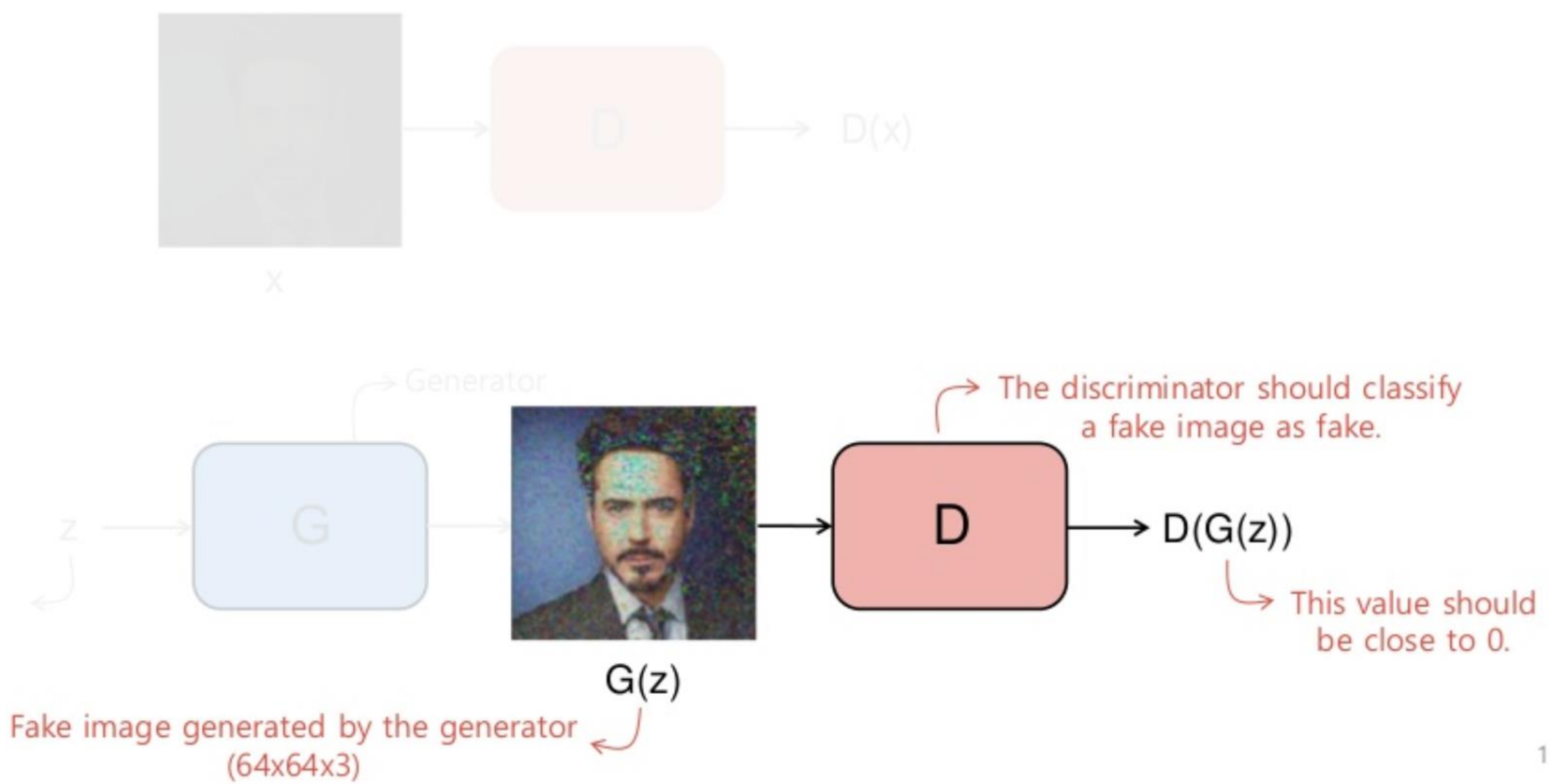


Discriminator Perspective

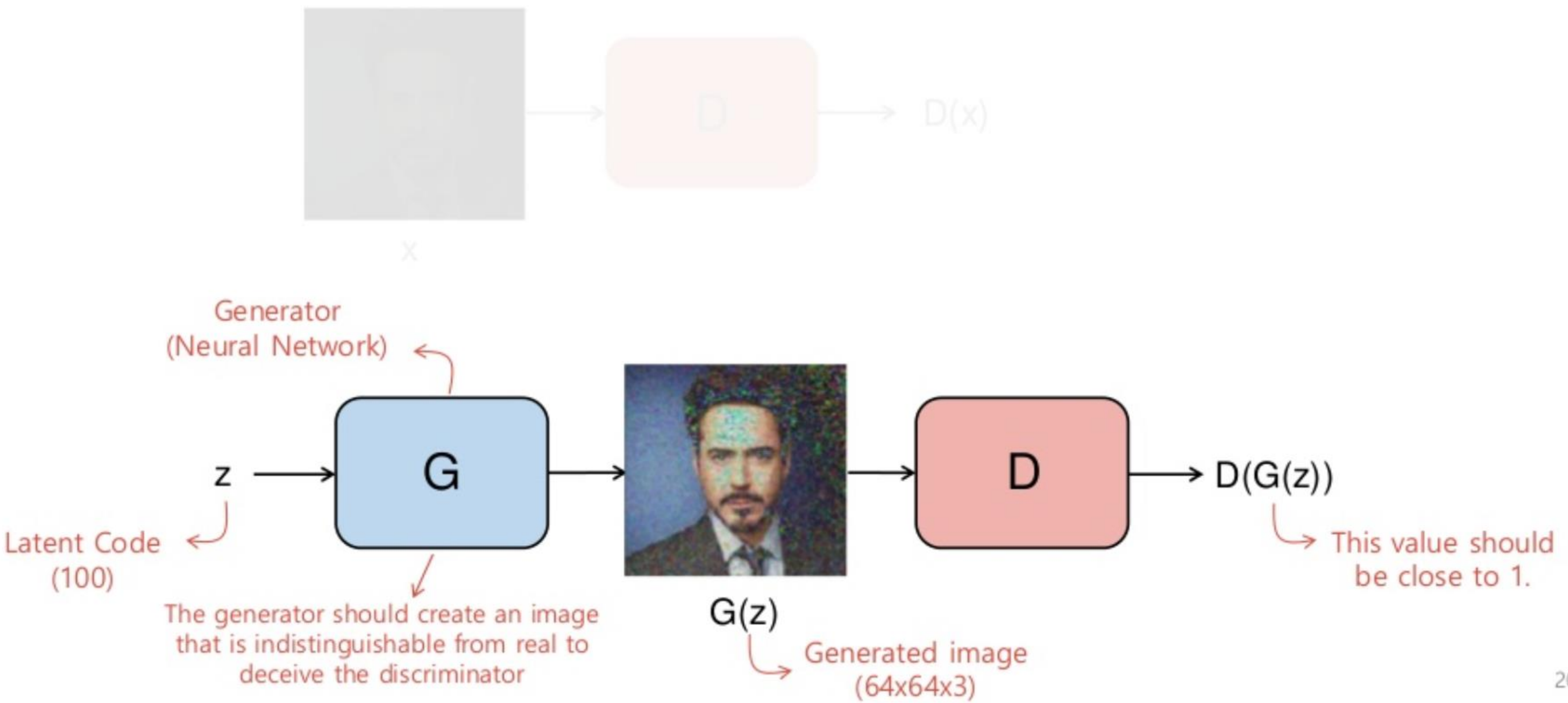


18

Discriminator Perspective



Generator Perspective



Objective Function of GAN

$$\text{loss} = -y \log h(x) - (1 - y) \log(1 - h(x))$$

Sample x from real data distribution

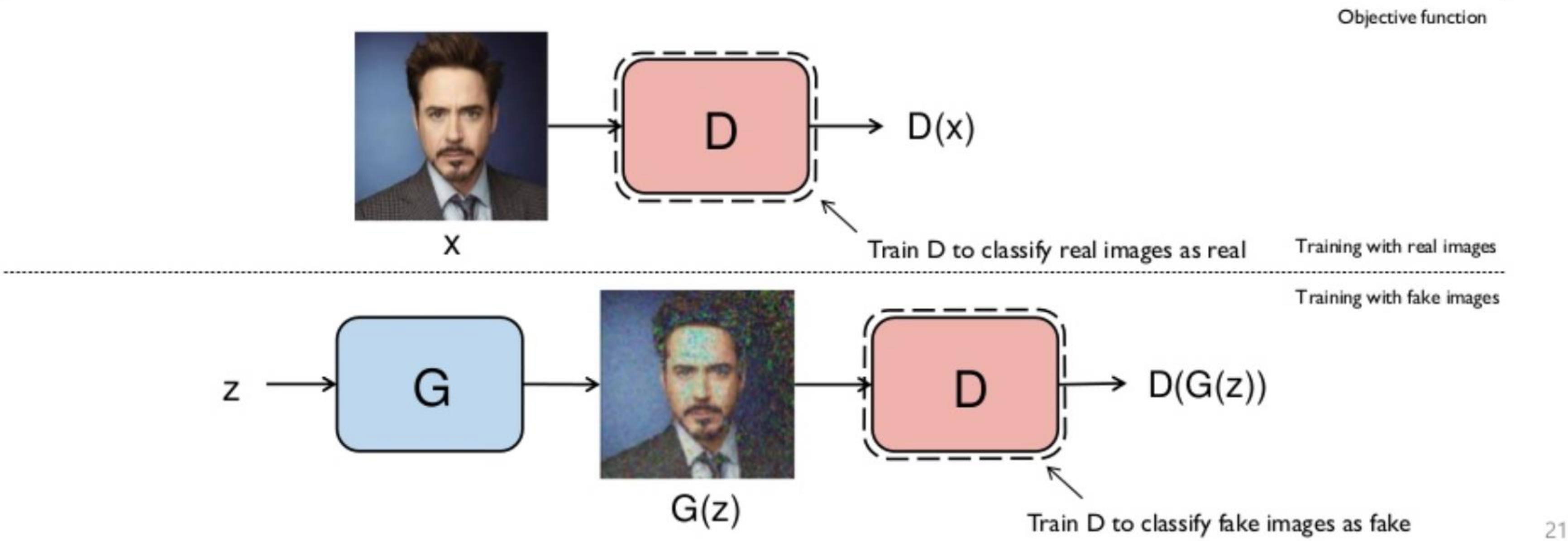
Sample latent code z from Gaussian distribution

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

D should maximize $V(D,G)$

Maximum when $D(x) = 1$

Maximum when $D(G(z)) = 0$



Objective Function of GAN

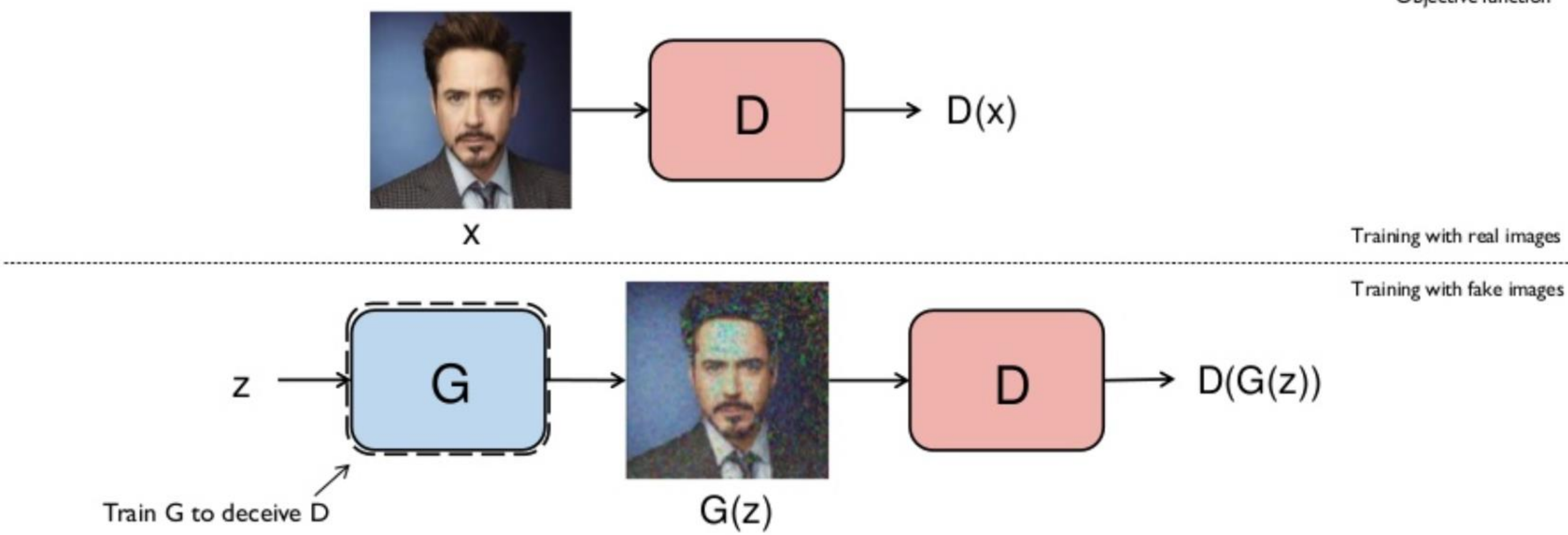
G is independent of this part

$$\min_{G} \max_{D} V(D,G) = \cancel{E_{x\sim p_{data}(x)}[\log D(x)]} + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

G should minimize $V(D,G)$

Minimum when $D(G(z)) = 1$

Objective function



22

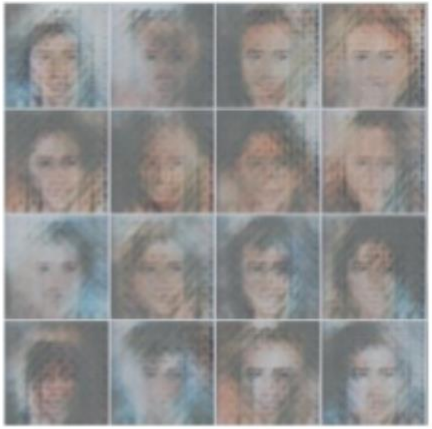
Non-Saturating Game

$$\min_G E_{z\sim p_z(z)}[\log(1 - D(G(z))]$$

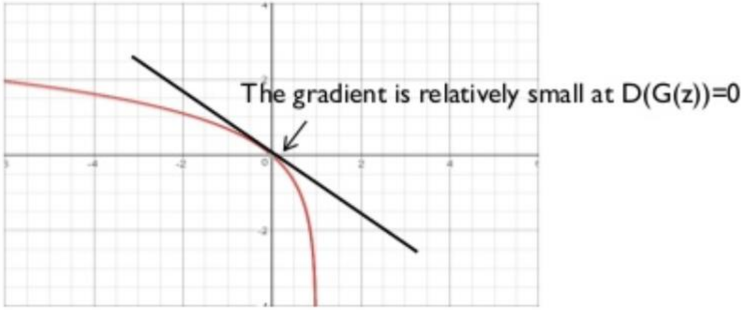
Objective function of G

At the beginning of training, the discriminator can clearly classify the generated image as fake because the quality of the image is very low.

This means that D(G(z)) is almost zero at early stages of training.



Images created by the generator at the beginning of training



$$y = \log(1 - x)$$

32

Non-Saturating Game

$$\cancel{\min_G E_{z\sim p_z(z)}[\log(1 - D(G(z))]}$$

$$\Big\downarrow \text{ Modification (heuristically motivated)}$$

$$\max_G E_{z\sim p_z(z)}[\log D(G(z))]$$

```
# tensorflow
tf.losses.sigmoid_cross_entropy()

# pytorch
nn.BCELoss()
```

- Practical Usage

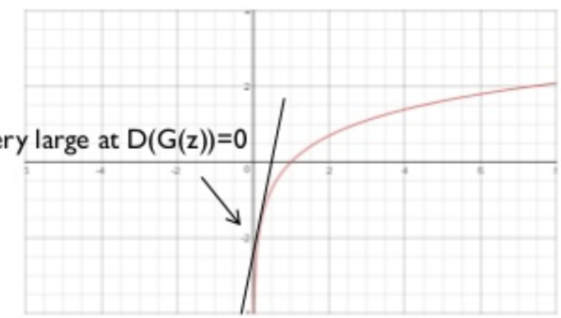
Use binary cross entropy loss function with fake label (1)

$$\min_G E_{z\sim p_z(z)}[-y \log D(G(z)) - (1-y)\log(1 - D(G(z))]$$

$$\Big\downarrow \; y = 1$$

$$\min_G E_{z\sim p_z(z)}[-\log D\big(G(z)\big)]$$

The gradient is very large at D(G(z))=0



$y = \log(x)$

Solving a Minmax Problem

Step 1: Fix G and perform a gradient step to

$$\max_D E_{x\sim p_{\text{data}}(x)}\left[\log D(x)\right] + E_{x\sim p_z(z)}\left[\log(1 - D(G(z)))\right]$$

Step 2: Fix D and perform a gradient step to

$$\max_G E_{x\sim p_z(z)}\left[\log D(G(z))\right]$$

OR

Step 1: Fix G and perform a gradient step to

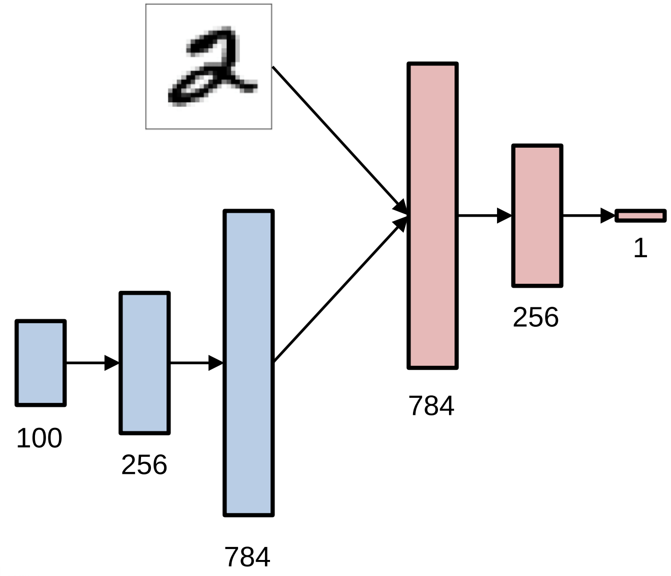
$$\min_D E_{x\sim p_{\text{data}}(x)}\left[-\log D(x)\right] + E_{x\sim p_z(z)}\left[-\log(1 - D(G(z)))\right]$$

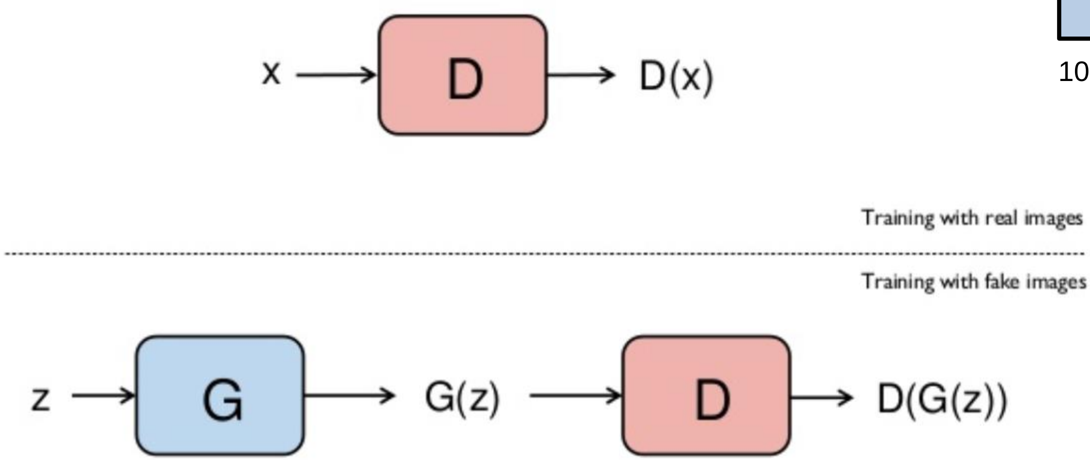
Step 2: Fix D and perform a gradient step to

$$\min_G E_{x\sim p_z(z)}\left[-\log D(G(z))\right]$$

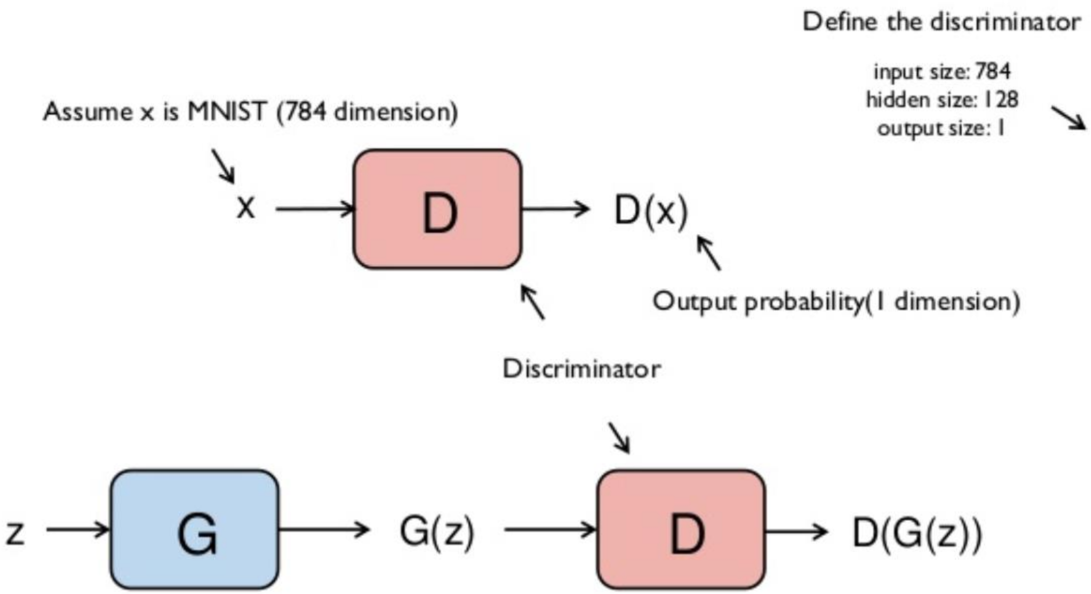
POSTECH

TensorFlow Implementation

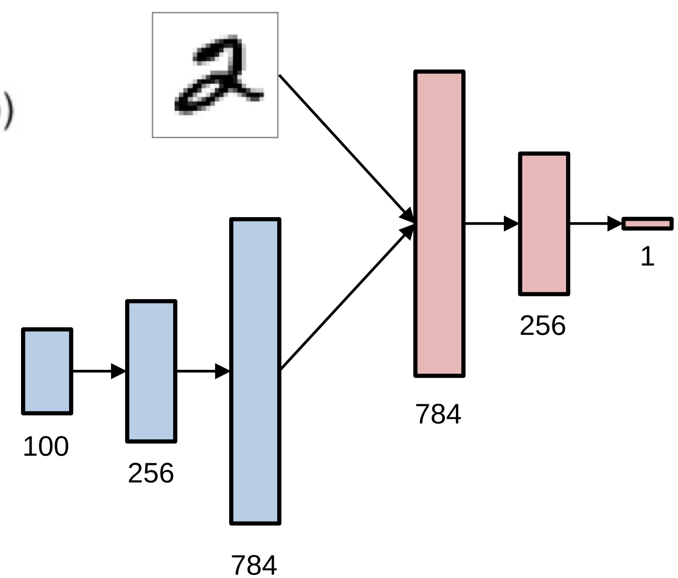




TensorFlow Implementation

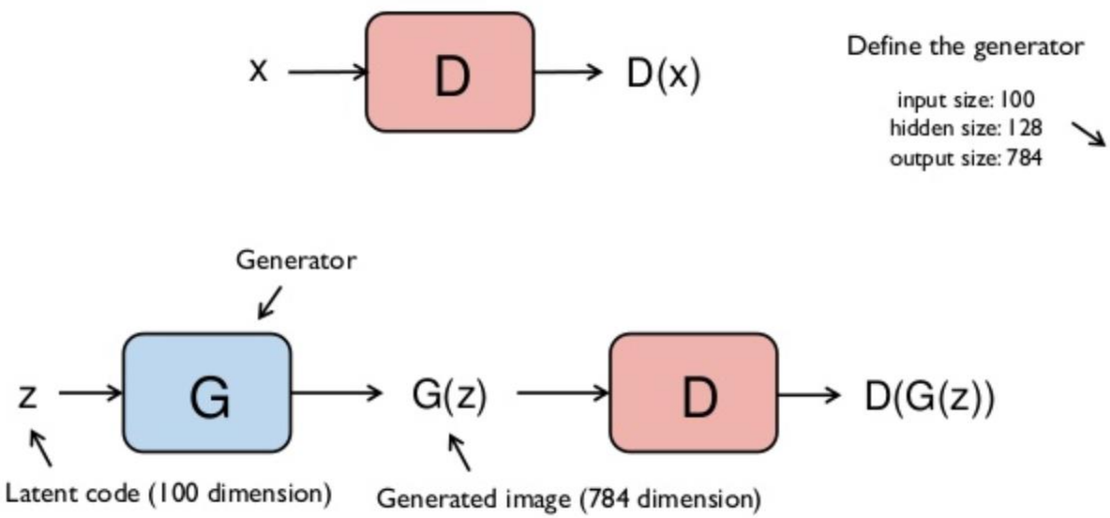


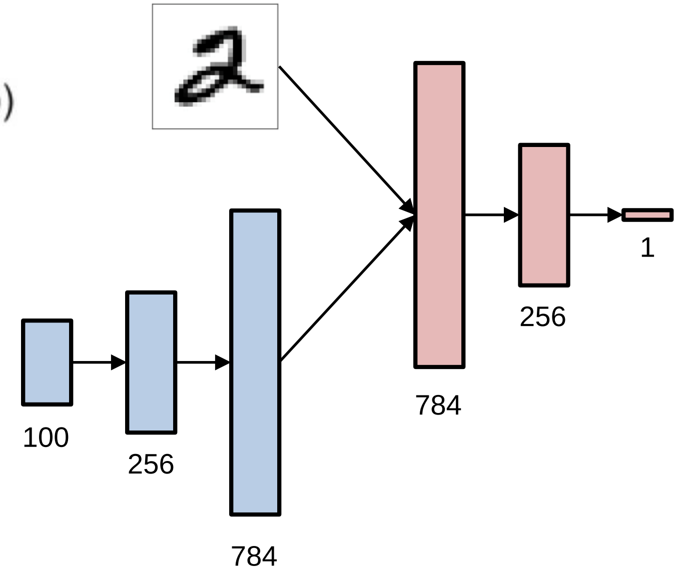

```
n_D_input = 28*28
n_D_hidden = 256
n_D_output = 1

n_G_input = 100
n_G_hidden = 256
n_G_output = 28*28
```



TensorFlow Implementation

```
n_D_input = 28*28
n_D_hidden = 256
n_D_output = 1

n_G_input = 100
n_G_hidden = 256
n_G_output = 28*28
```





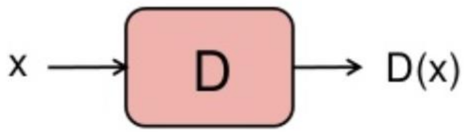
TensorFlow Implementation

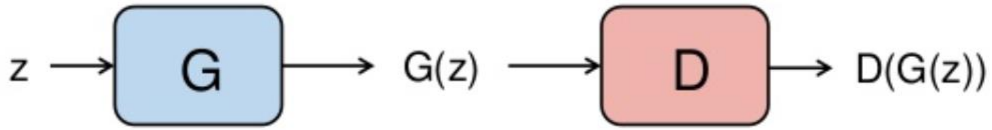
Step 1: Fix G and perform a gradient step to

$$\min_D E_{x\sim p_{\text{data}}(x)}\left[-\log D(x)\right] + E_{x\sim p_z(z)}\left[-\log(1 - D(G(z)))\right]$$

Step 2: Fix D and perform a gradient step to

$$\min_G E_{x\sim p_z(z)}\left[-\log D(G(z))\right]$$

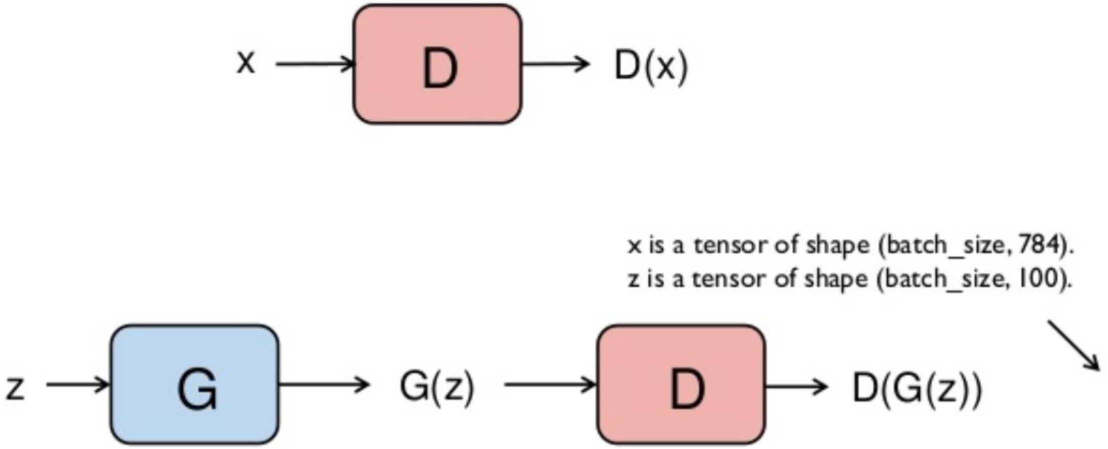





```
cost_D = tf.reduce_mean(tf.log(D_real) + tf.log(1 - D_fake))
cost_G = tf.reduce_mean(tf.log(D_fake))
```

```
LR = 0.0002
D_train = tf.train.AdamOptimizer(LR).minimize(-cost_D, var_list = D_var_list)
G_train = tf.train.AdamOptimizer(LR).minimize(-cost_G, var_list = G_var_list)
```

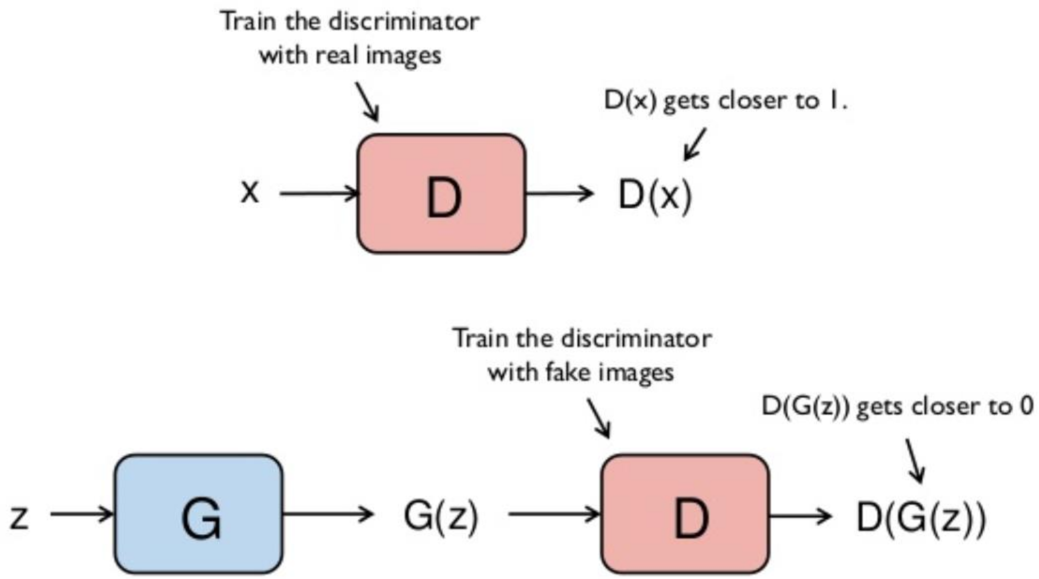
TensorFlow Implementation




```
z = tf.placeholder(tf.float32, [None, n_G_input])
x = tf.placeholder(tf.float32, [None, n_D_input])
```

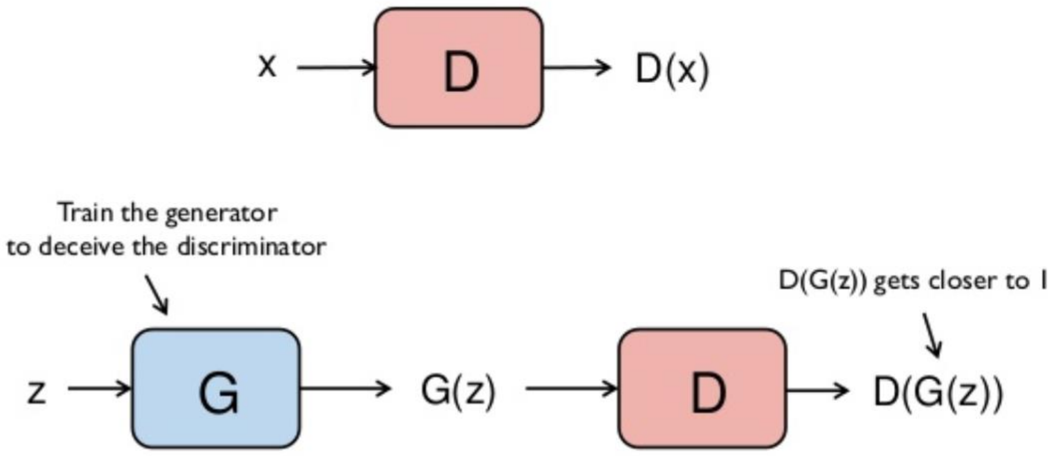
TensorFlow Implementation

```
_, D_loss_val = sess.run([D_train, cost_D], feed_dict={x: train_x, z: noise})
```



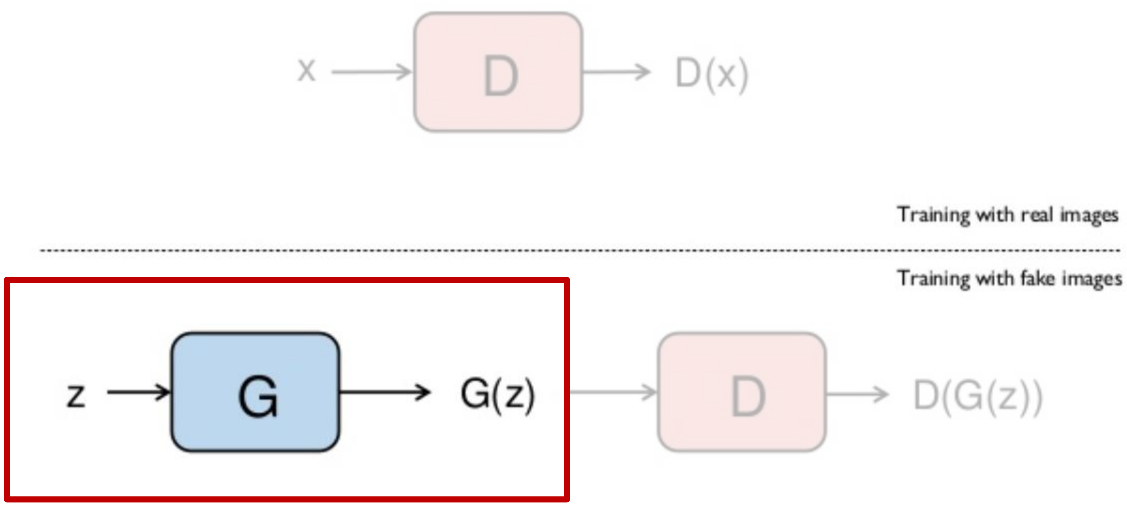
TensorFlow Implementation

```
_, D_loss_val = sess.run([D_train, cost_D], feed_dict={x: train_x, z: noise})
_, G_loss_val = sess.run([G_train, cost_G], feed_dict={z: noise})
```



After Training

- After training, use generator network to generate new data




```
noise = make_noise(n_batch, n_G_input)
G_img = sess.run(G_output, feed_dict={z: noise})
plt.imshow(G_img[0,:].reshape(28,28),'gray')
plt.show()
```

GAN Samples

2009

2015

2018