



# Generative Adversarial Networks (GANs)

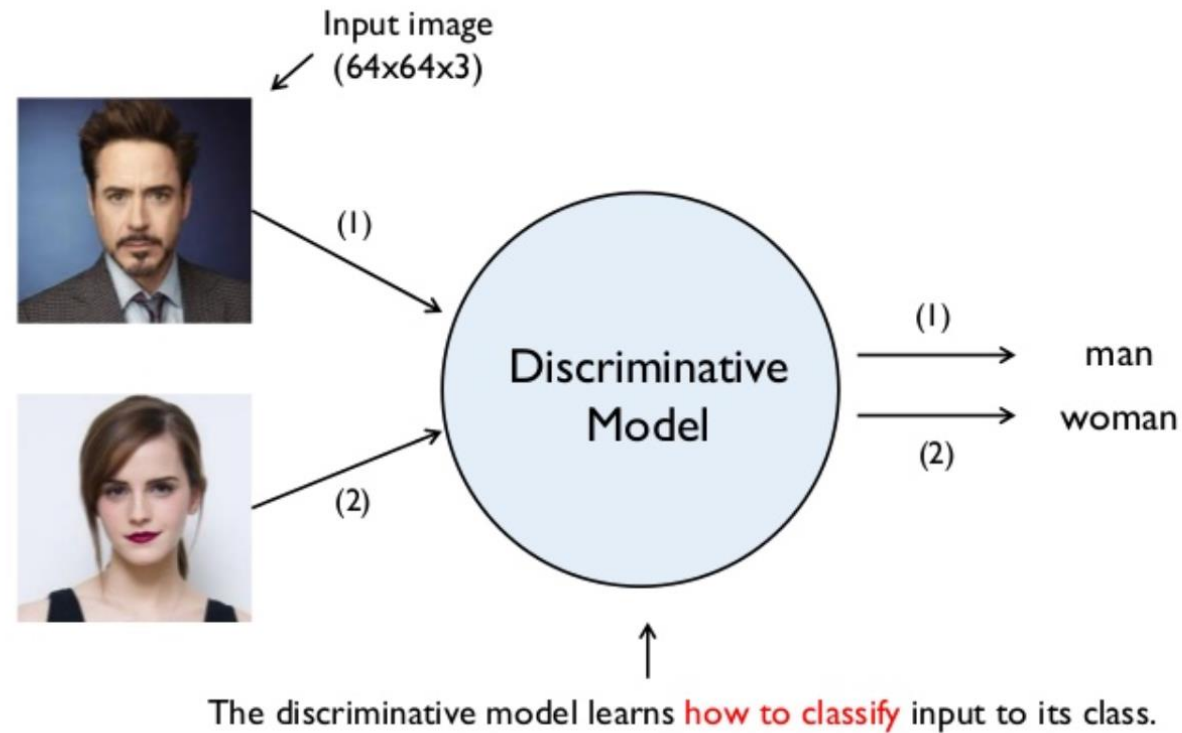
**Prof. Seungchul Lee**  
**Industrial AI Lab.**

# Source

- 1시간만에 GAN (Generative Adversarial Network) 완전 정복하기
  - by 최윤제
  - YouTube: [https://www.youtube.com/watch?v=odpjk7\\_tGY0](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/](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/>

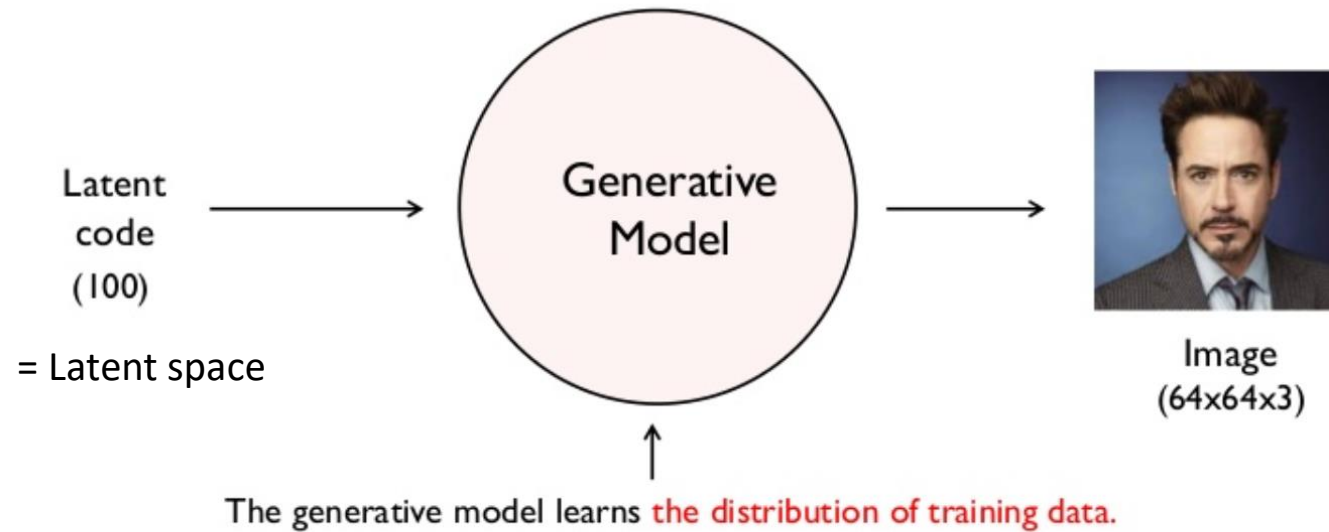
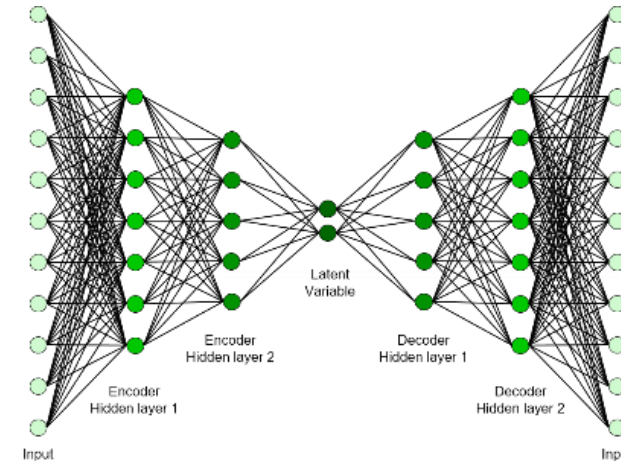
# Supervised Learning

- Discriminative model



# Unsupervised Learning

- Generative model



# Model Distribution vs. Data Distribution

# Probability Distribution

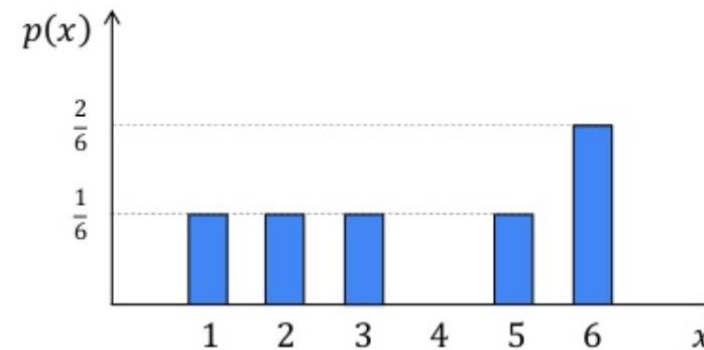
## Probability Basics (Review)



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?

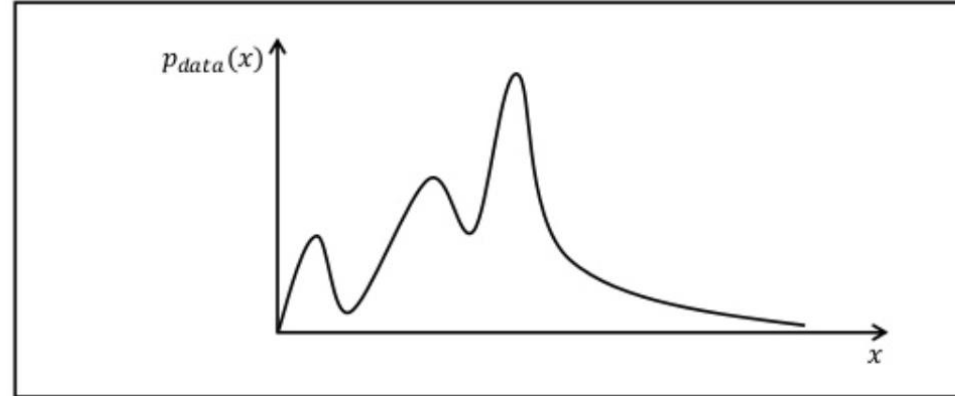
At this point,  $x$  can be represented as a (for example) 64x64x3 dimensional vector.



# Probability Distribution

Probability density function

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



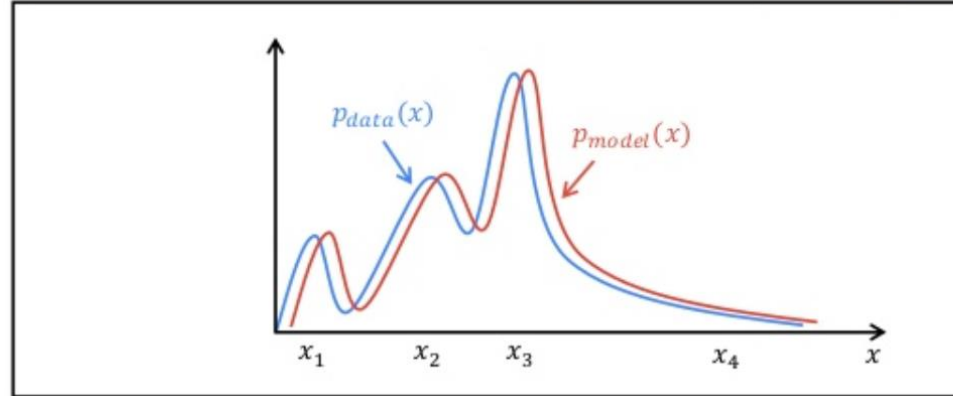


# Probability Density Estimation Problem

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

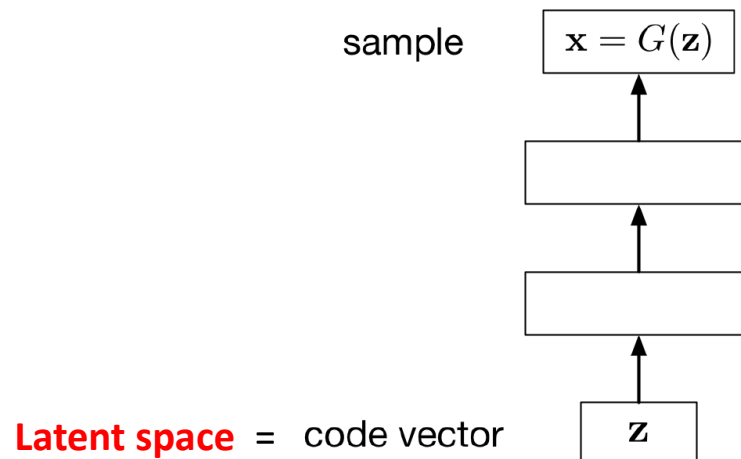
The goal of the generative model is to find a  $p_{model}(x)$  that approximates  $p_{data}(x)$  well.

↗ Distribution of images generated by the model  
↘ Distribution of actual images



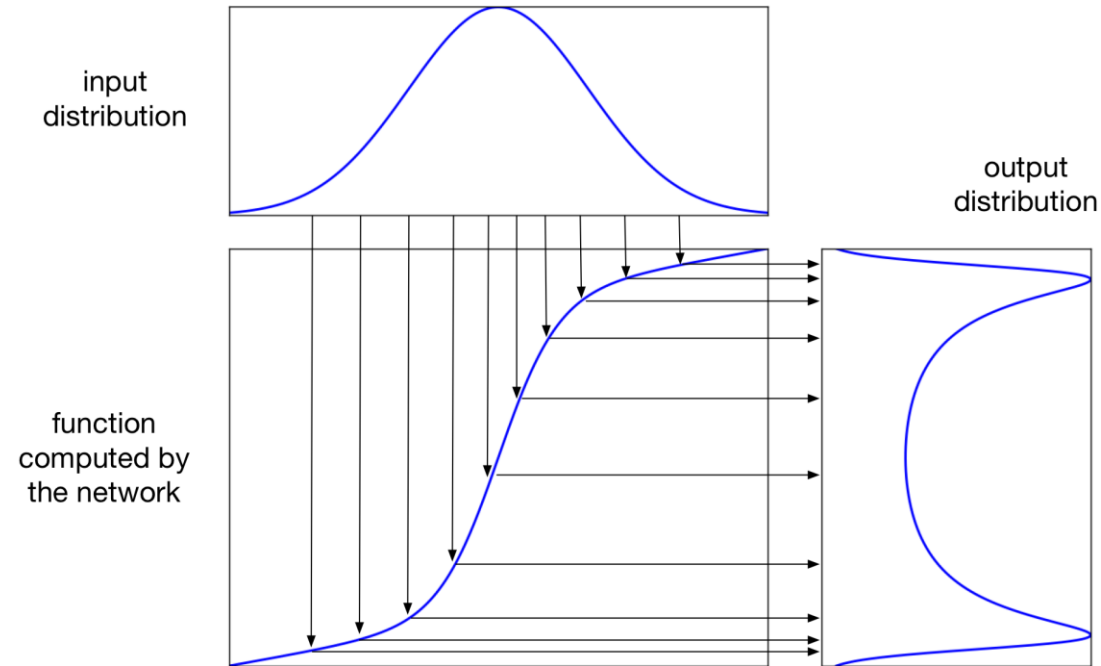
# Generative Models from Lower Dimension

- 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  $x = G(z)$



# Deterministic Transformation (by Network)

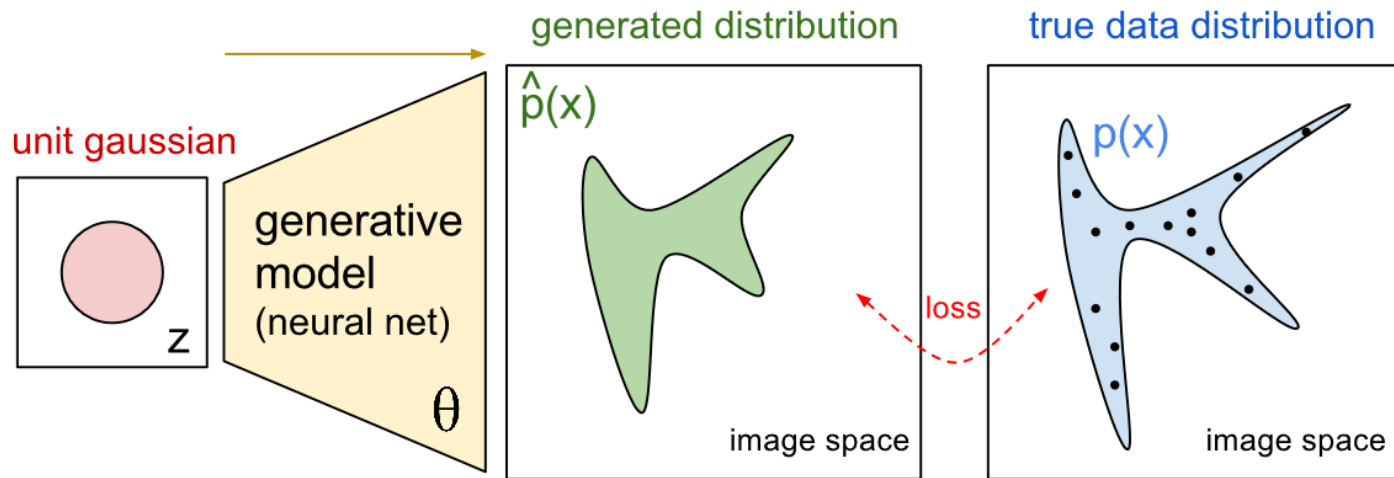
- 1-dimensional example:



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

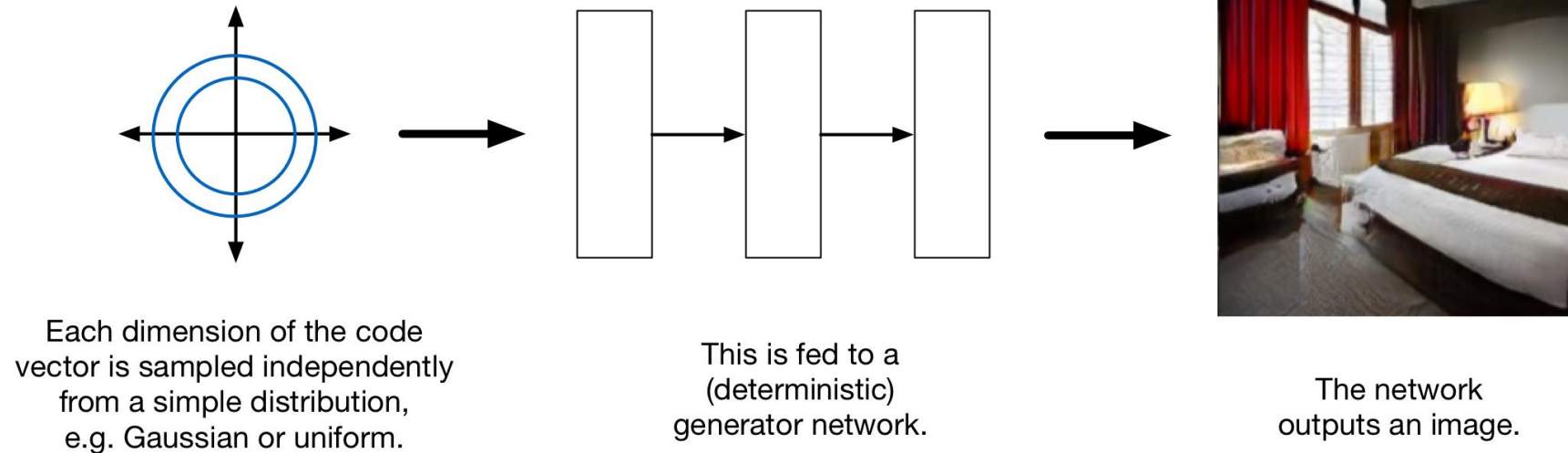
# Deterministic Transformation (by Network)

- High dimensional example:



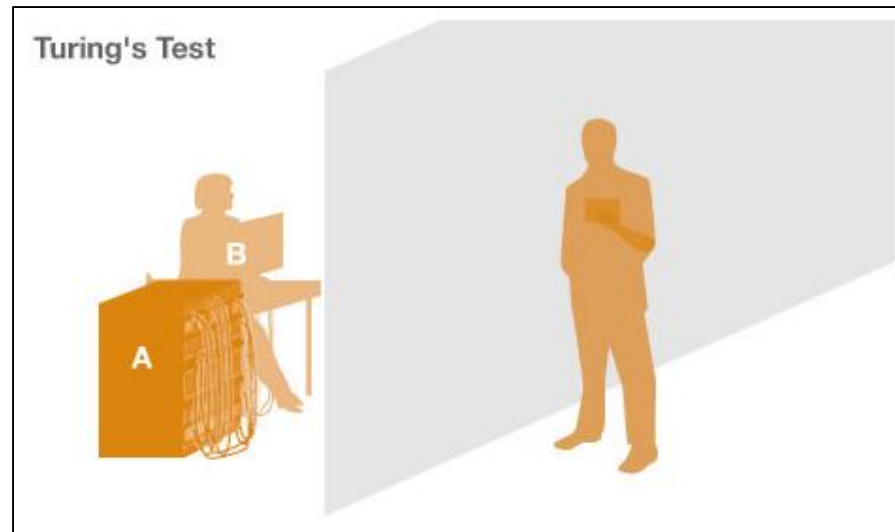
# Prob. Density Function by Deep Learning

- Generative model of image



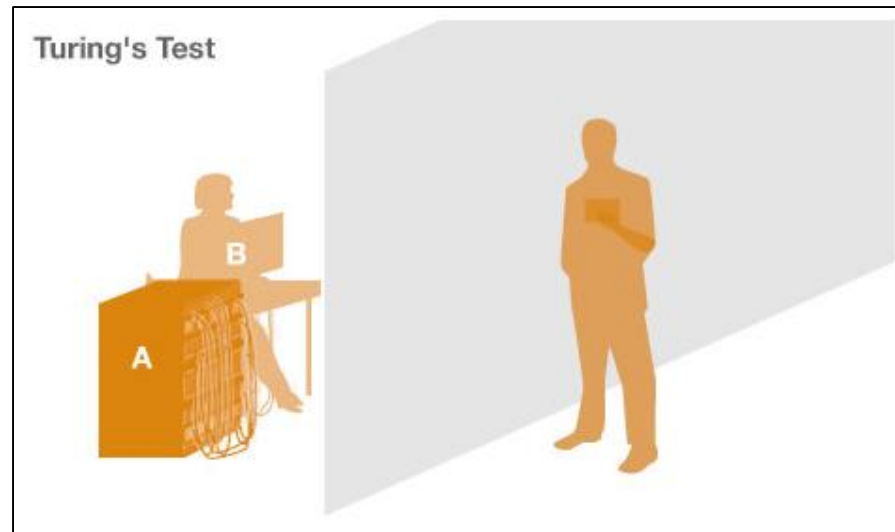
# Generative Adversarial Networks (GANs)

- 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



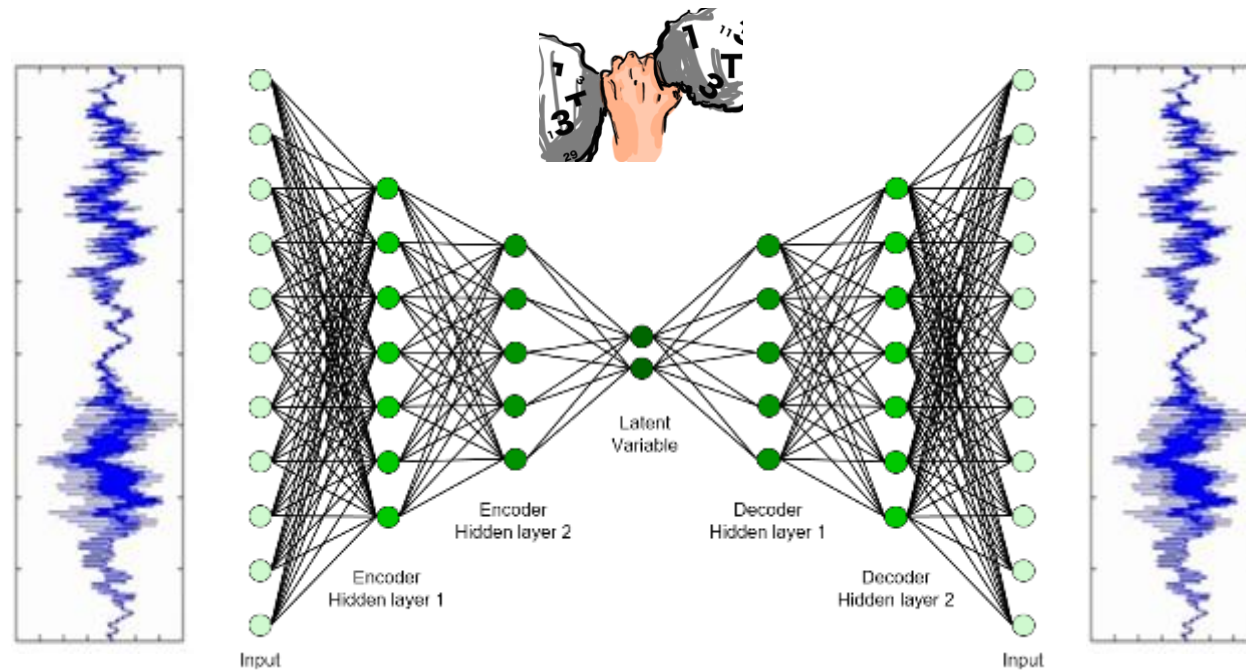
# Generative Adversarial Networks (GAN)

- 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



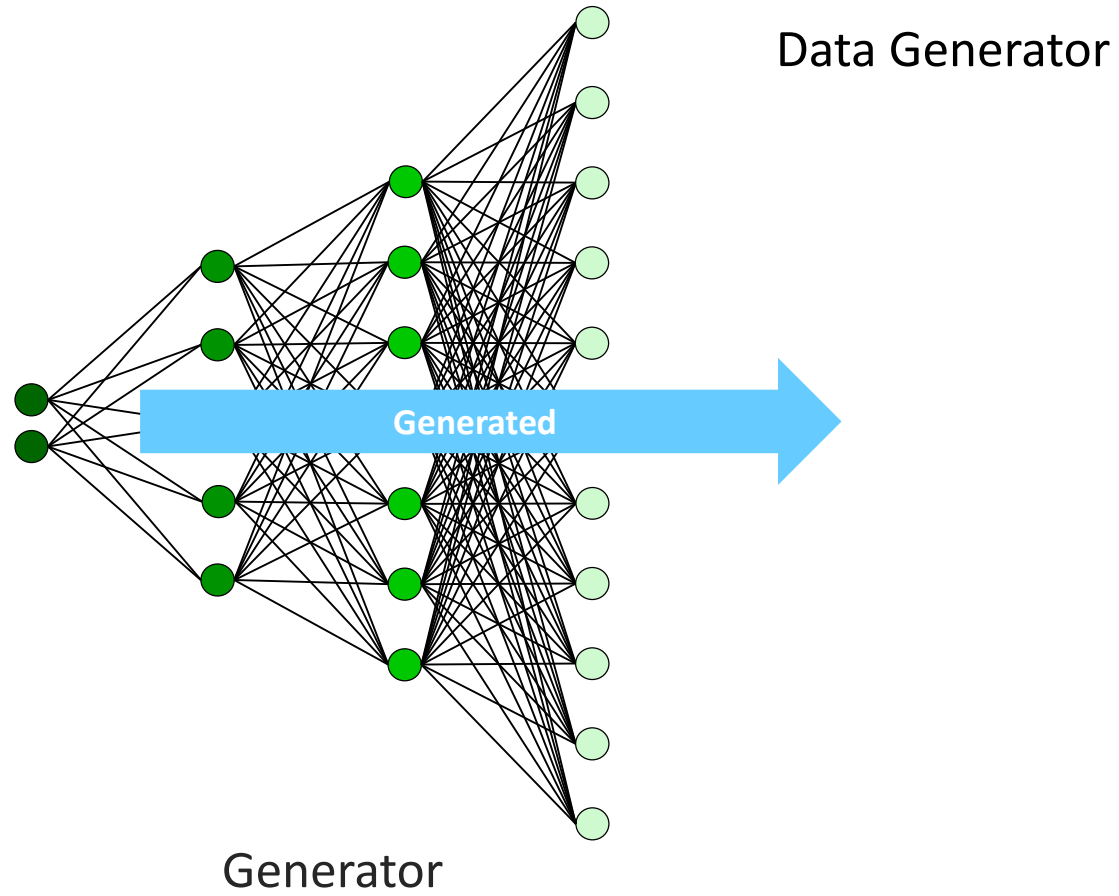
# Autoencoder

- Dimension reduction
- Recover the input data
  - Learns an encoding of the inputs so as to recover the original input from the encodings as well as possible



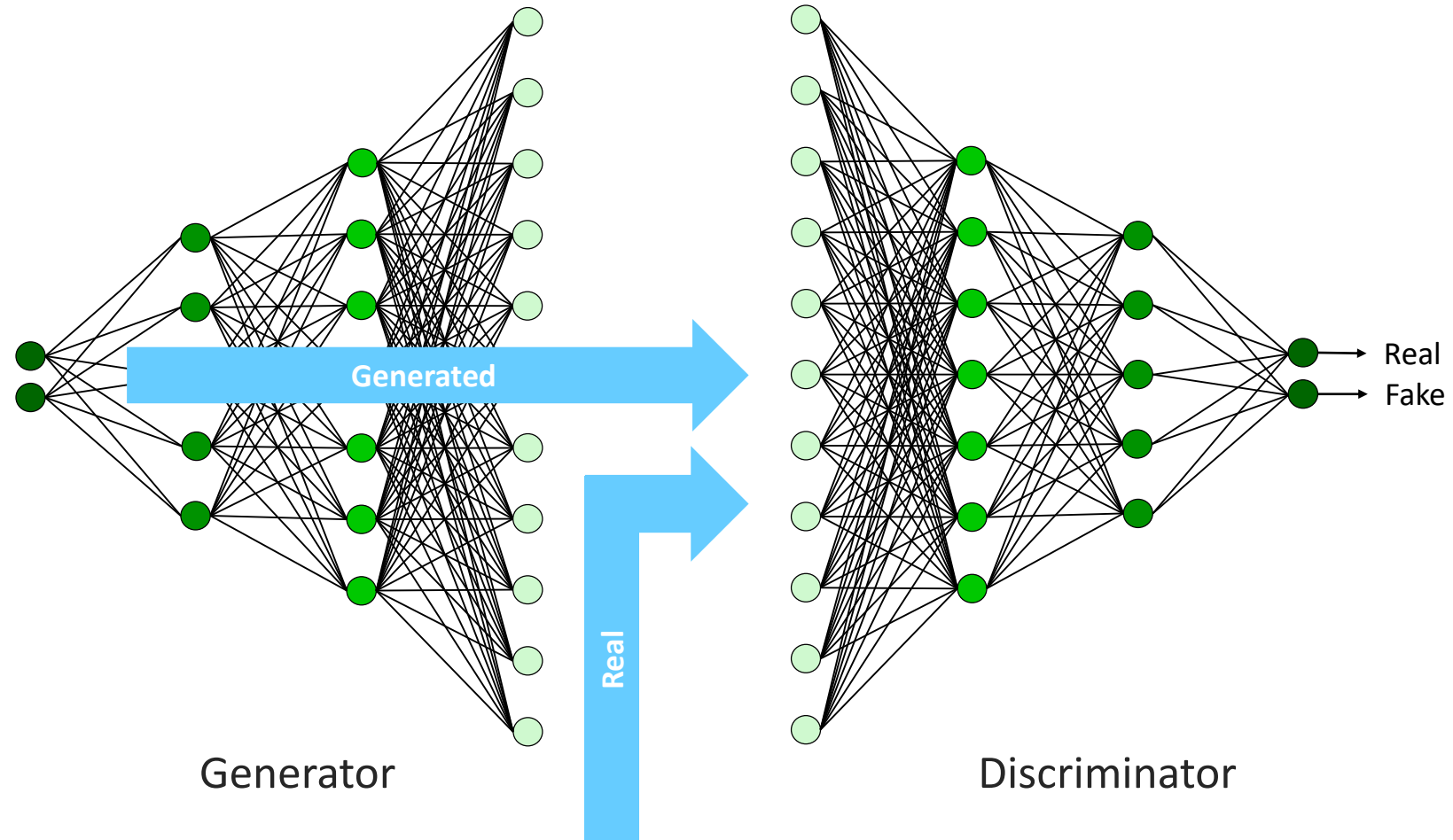
# Generative Adversarial Networks (GAN)

- Analogous to Turing Test

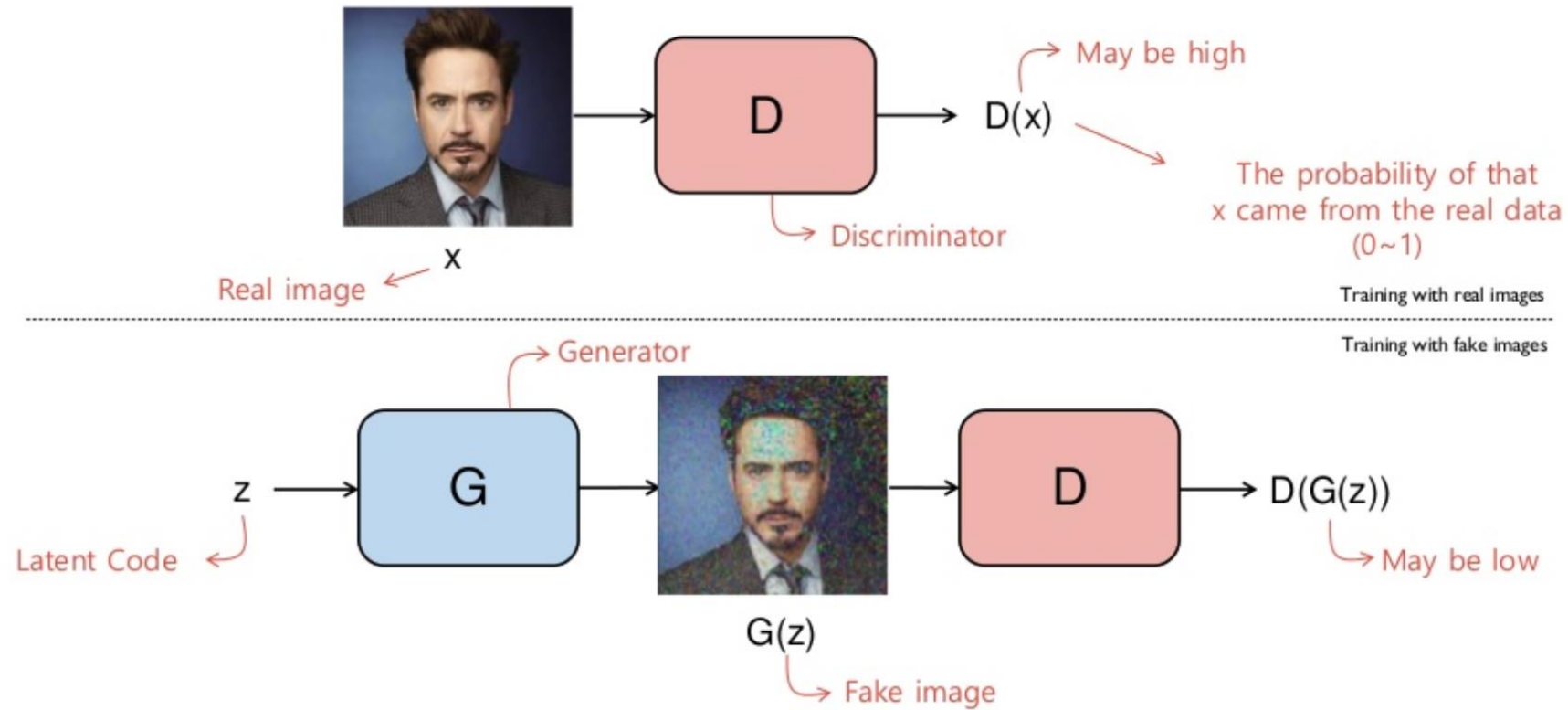


# Generative Adversarial Networks (GAN)

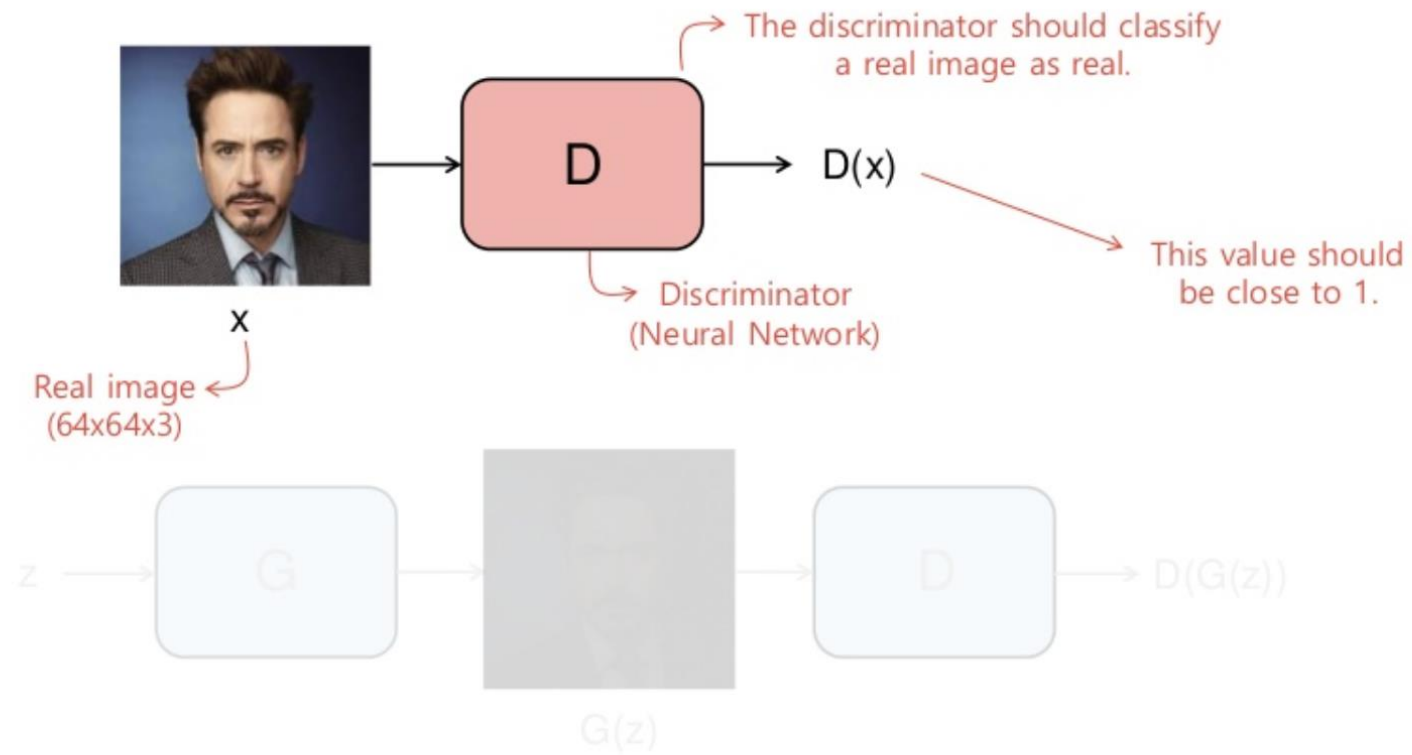
- Analogous to Turing Test



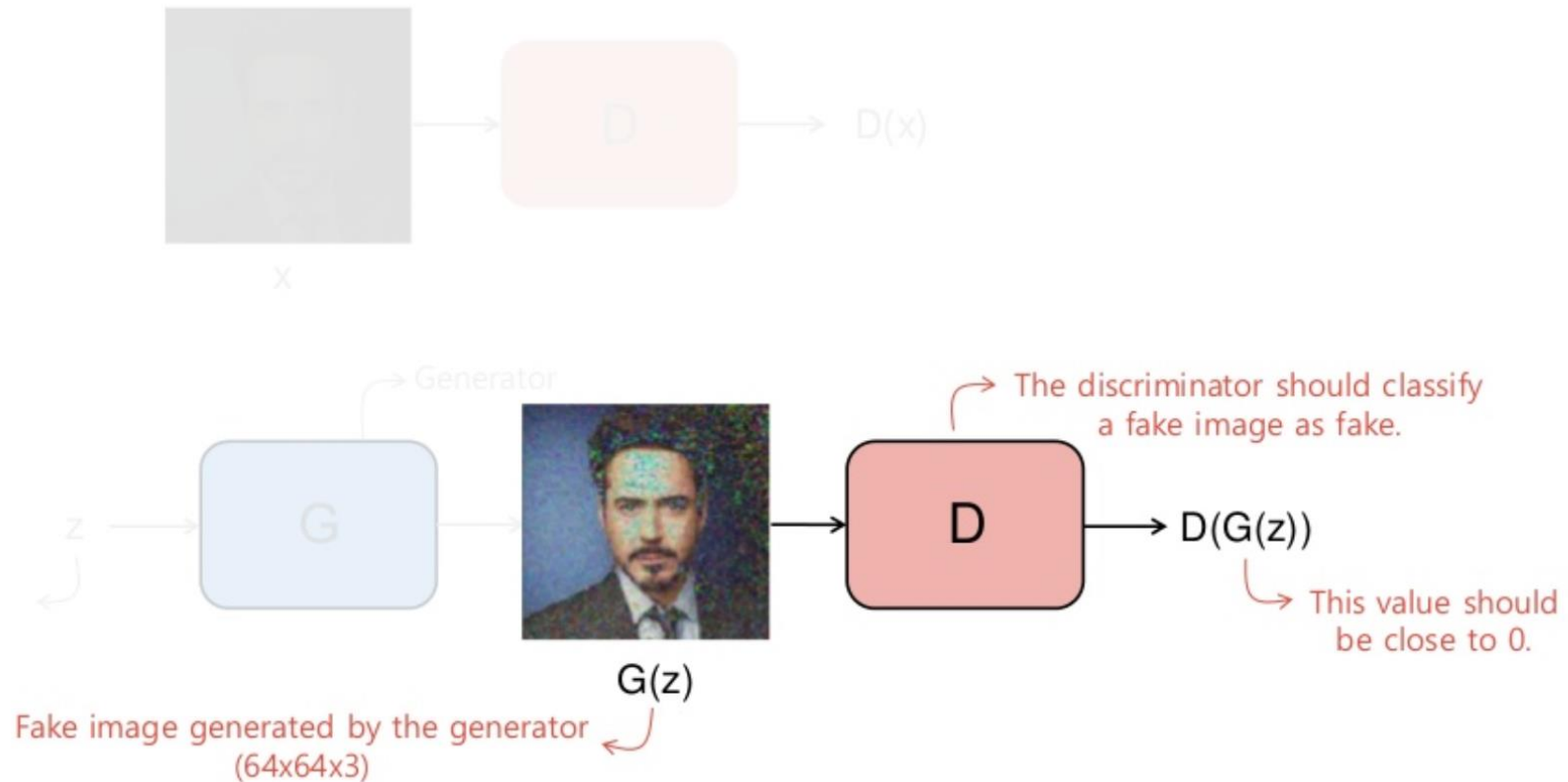
# Intuition for GAN



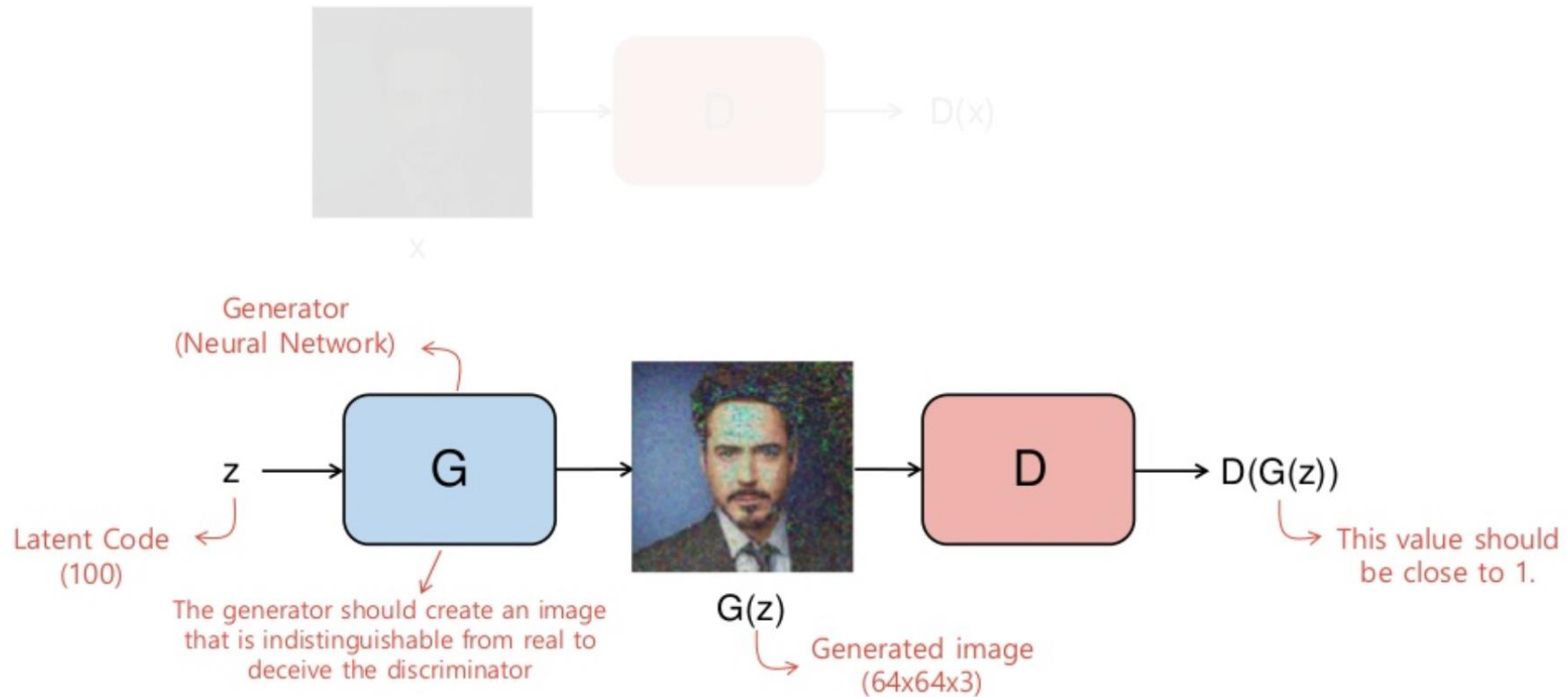
# Discriminator Perspective (1/2)



## Discriminator Perspective (2/2)



# Generator Perspective



# Loss Function of Discriminator

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

Sample  $x$  from real data distribution

Sample latent code  $z$  from Gaussian distribution

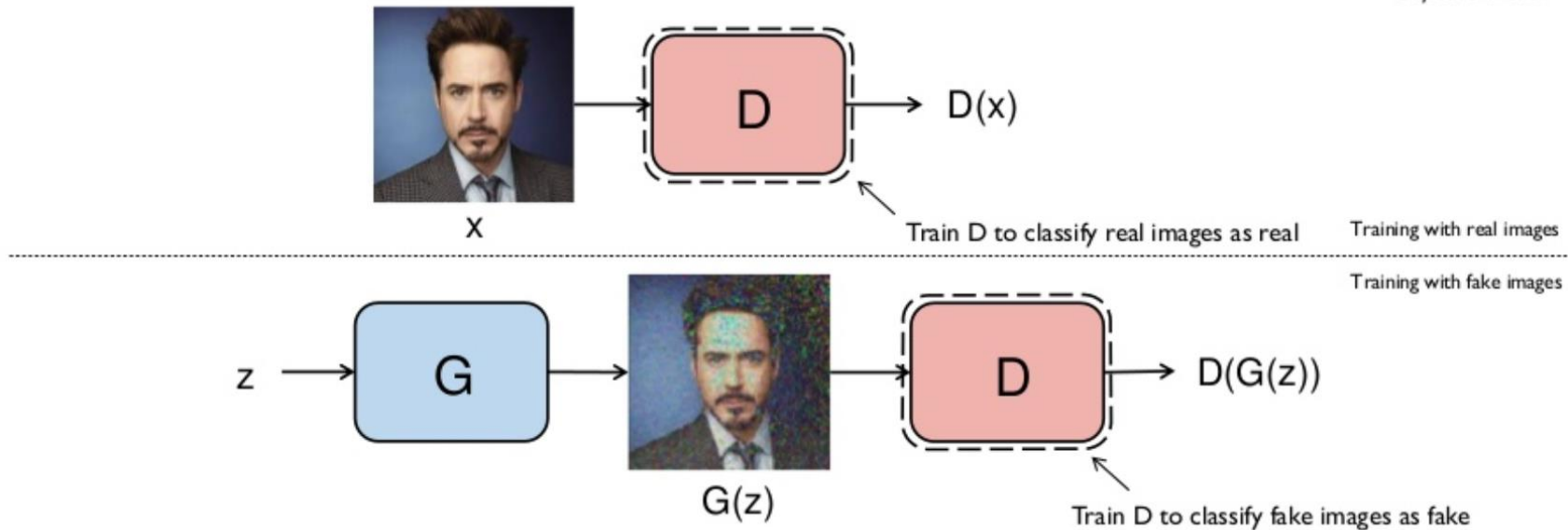
$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

$D$  should maximize  $V(D, G)$

Maximum when  $D(x) = 1$

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

Objective function





# Loss Function of Generator

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

$G$  is independent of this part

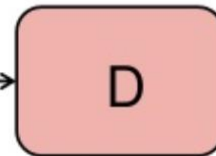
$G$  should minimize  $V(D, G)$

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

Objective function

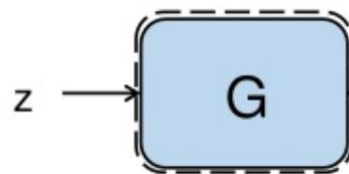


x



$D(x)$

Training with real images



z

Train G to deceive D



$G(z)$



$D(G(z))$

Training with fake images

# Supplementary - Non-Saturating Game

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

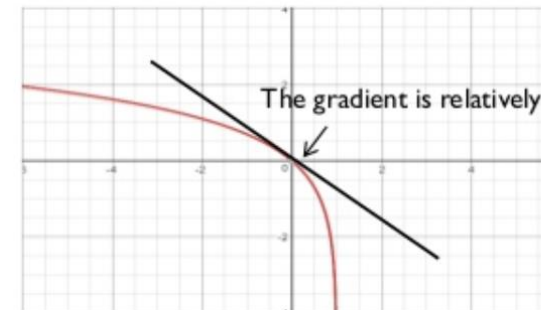
Objective function of G

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



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

# Supplementary - Non-Saturating Game

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

↓ Modification (heuristically motivated)

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

- Practical Usage

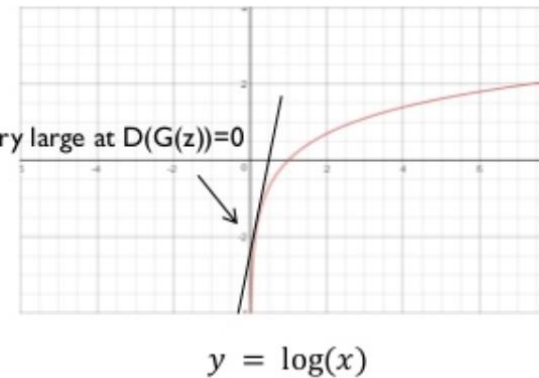
Use **binary cross entropy loss function** with fake label (1)

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

↓  $y = 1$

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

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



# Solving a MinMax Problem

Step 1: Fix  $G$  and perform a gradient step to

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

Step 2: Fix  $D$  and perform a gradient step to

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

OR

Step 1: Fix  $G$  and perform a gradient step to

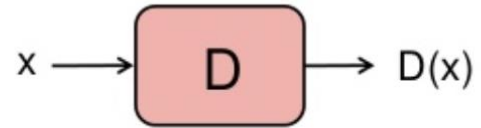
$$\min_D E_{x \sim p_{\text{data}}(x)} [-\log D(x)] + E_{x \sim p_z(z)} [-\log(1 - D(G(z)))]$$

Step 2: Fix  $D$  and perform a gradient step to

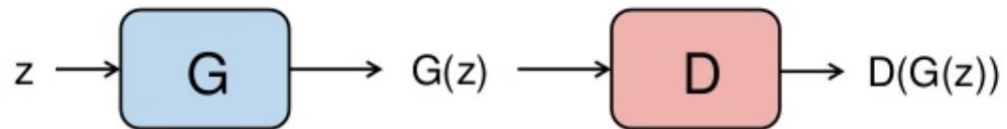
$$\min_G E_{x \sim p_z(z)} [-\log D(G(z))]$$

# GAN Implementation in TensorFlow

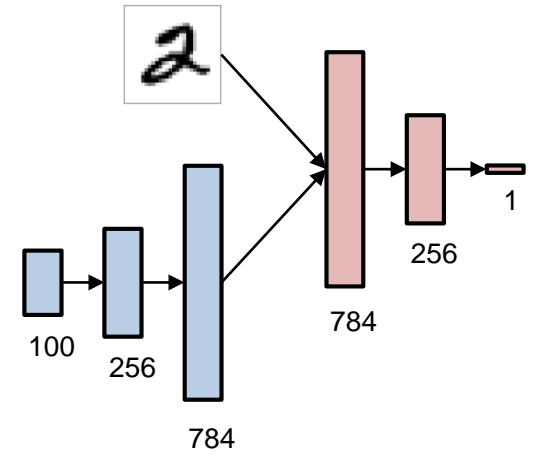
# TensorFlow Implementation



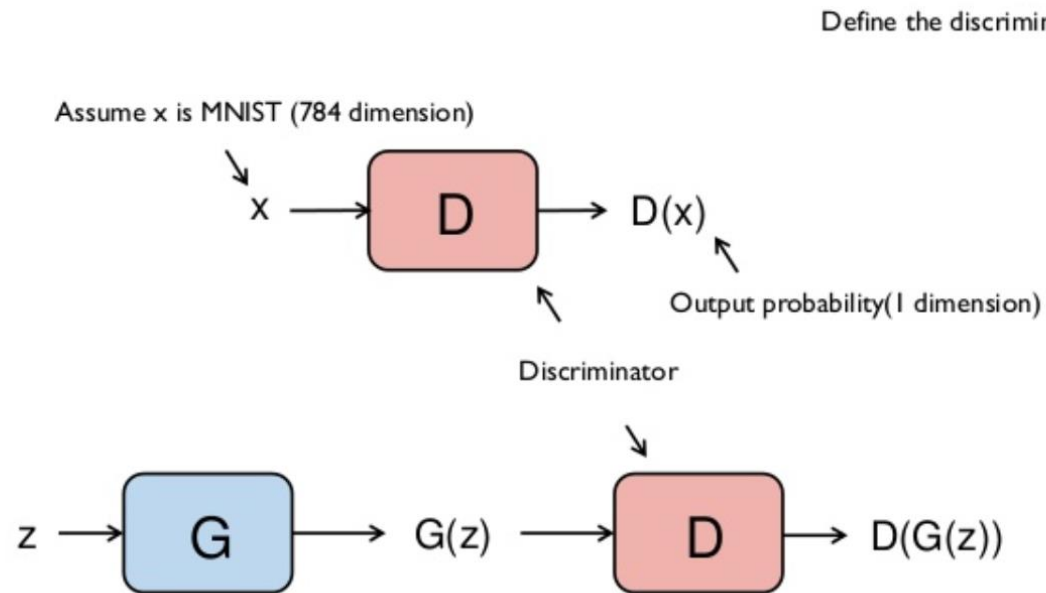
Training with real images



Training with fake images



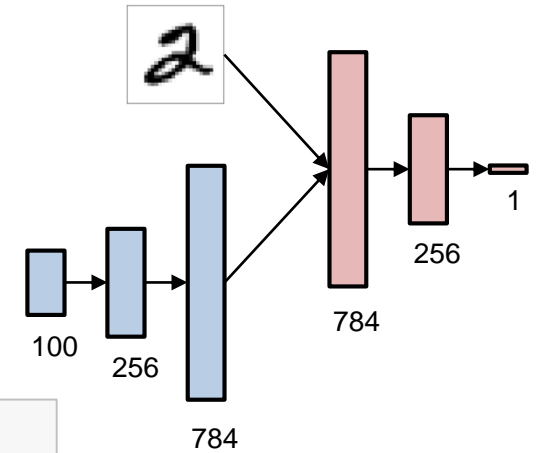
# TensorFlow Implementation



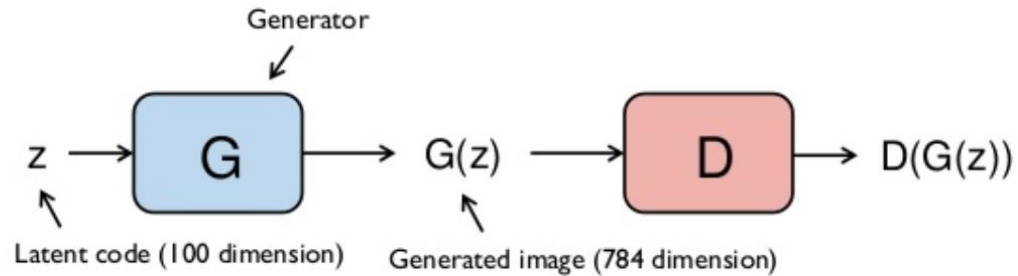
Define the discriminator

```
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
```

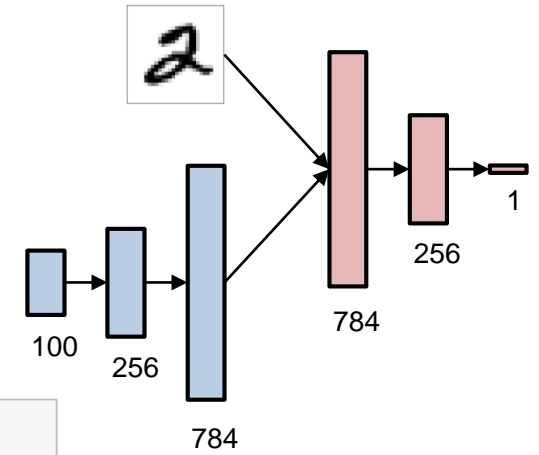


# TensorFlow Implementation



Define the generator

```
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
```





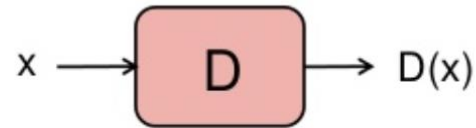
# TensorFlow Implementation

Step 1: Fix  $G$  and perform a gradient step to

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

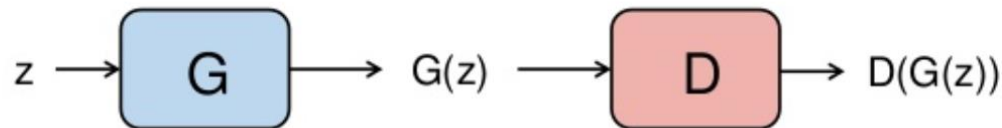
Step 2: Fix  $D$  and perform a gradient step to

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



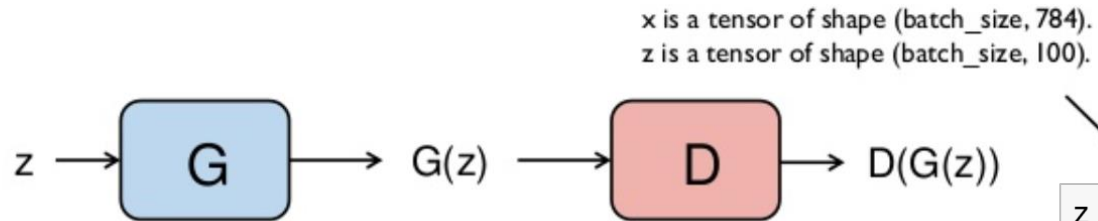
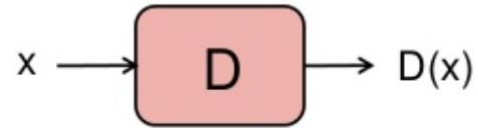
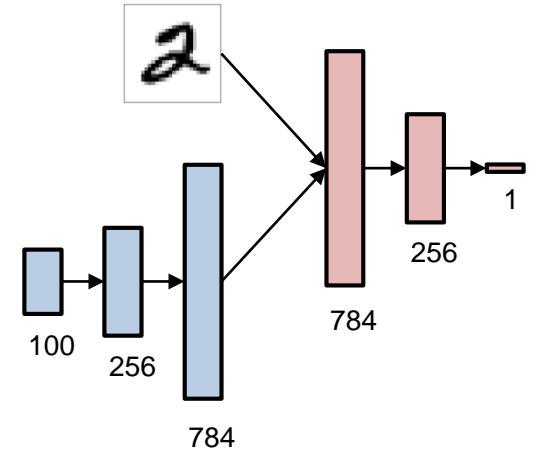
```
D_loss = tf.reduce_mean(- tf.log(D_real) - tf.log(1 - D_fake))  
G_loss = tf.reduce_mean(- tf.log(D_fake))
```

```
D_var_list = [weights['D1'], biases['D1'], weights['D2'], biases['D2']]  
G_var_list = [weights['G1'], biases['G1'], weights['G2'], biases['G2']]
```



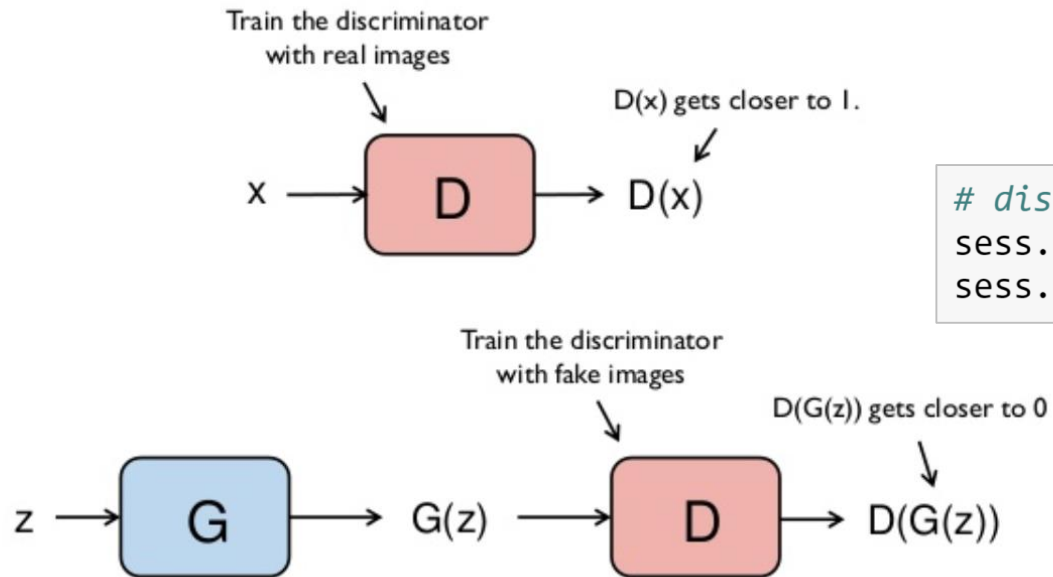
```
LR = 0.0002  
D_optm = tf.train.AdamOptimizer(LR).minimize(D_loss, var_list = D_var_list)  
G_optm = tf.train.AdamOptimizer(LR).minimize(G_loss, var_list = G_var_list)
```

# TensorFlow Implementation

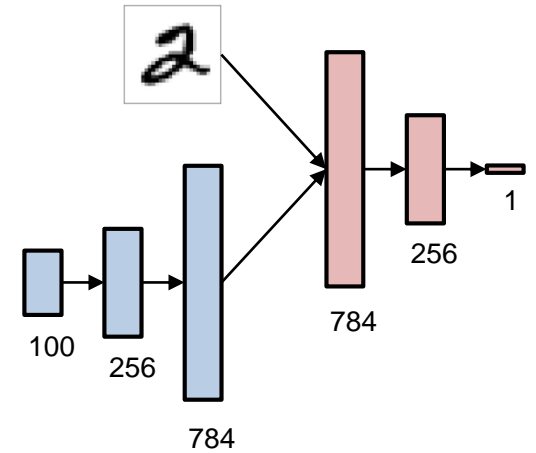


```
z = tf.placeholder(tf.float32, [None, n_G_input])  
x = tf.placeholder(tf.float32, [None, n_D_input])
```

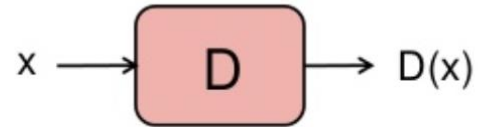
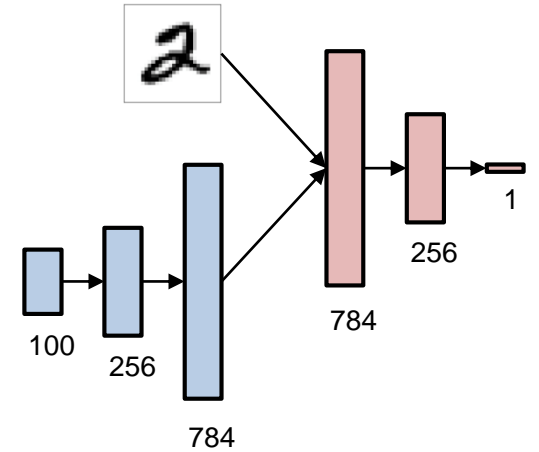
# TensorFlow Implementation



```
# discriminator and generator are separately trained  
sess.run(D_optm, feed_dict = {x: train_x, z: noise})  
sess.run(G_optm, feed_dict = {z: noise})
```

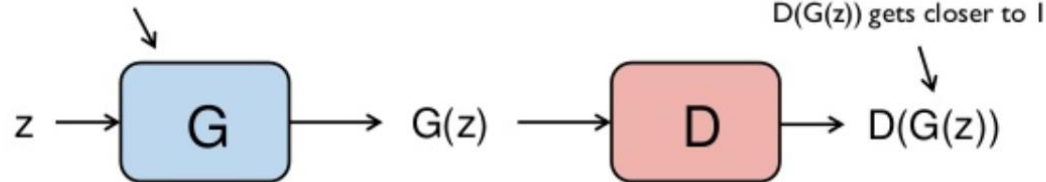


# TensorFlow Implementation



```
# discriminator and generator are separately trained  
sess.run(D_optm, feed_dict = {x: train_x, z: noise})  
sess.run(G_optm, feed_dict = {z: noise})
```

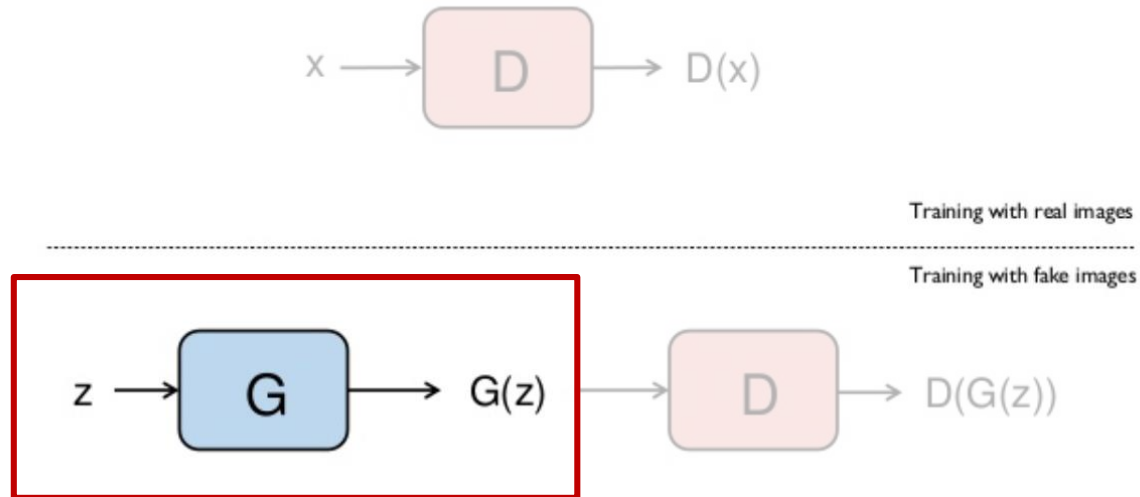
Train the generator  
to deceive the discriminator



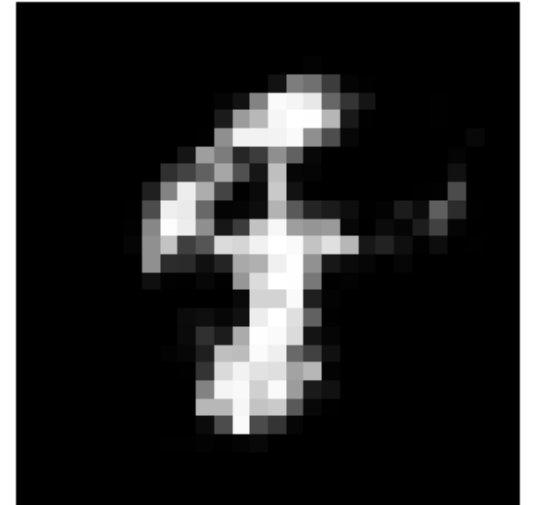
$D(G(z))$  gets closer to 1

# After Training

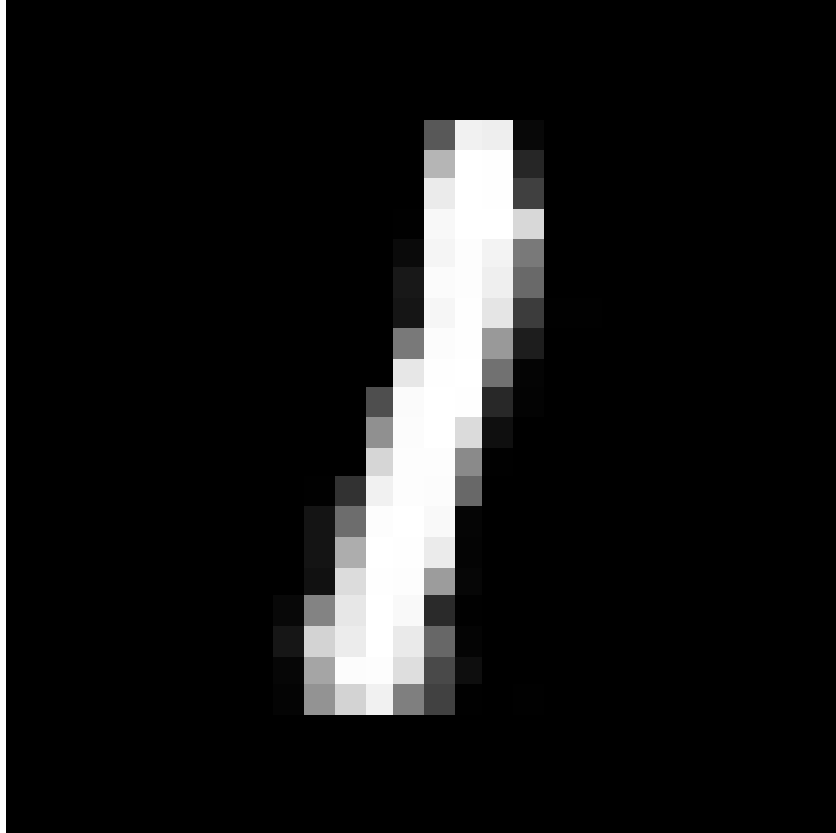
- After training, use generator network to generate new data



```
noise = make_noise(n_batch, n_G_input)
G_img = sess.run(G_output, feed_dict = {z: noise})
```



# GAN Samples



CelebA-HQ  
1024 × 1024

Latent space interpolations