Generative Adversarial Networks (GANs)

Industrial AI Lab.

Iljeok Kim

Source

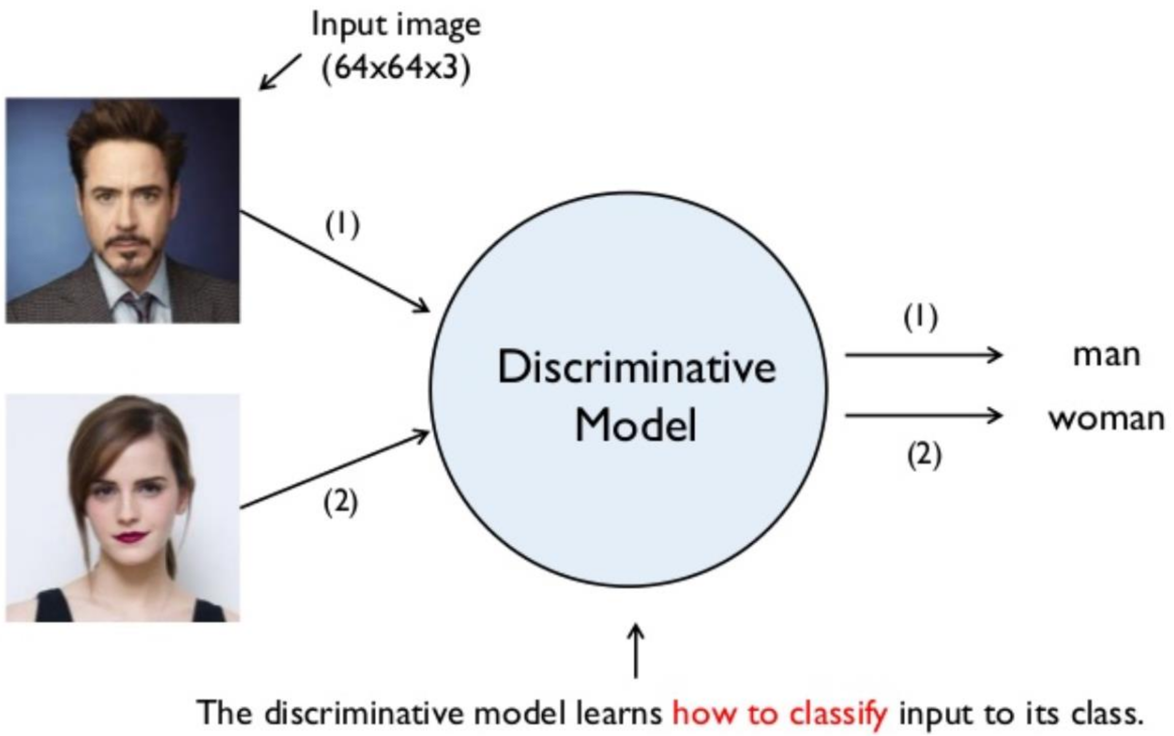
- 1시간만에 GAN (Generative Adversarial Network) 완전 정복하기
  - by 최윤제
  - YouTube: https://www.youtube.com/watch?v=odpjk7_tGY0
  - Slides: https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

- 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/

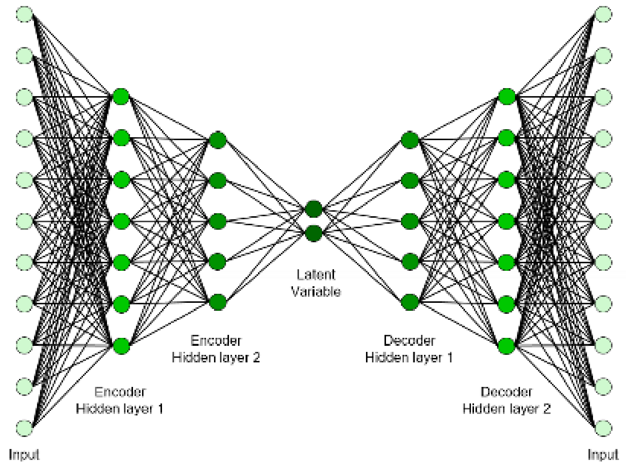
Supervised Learning

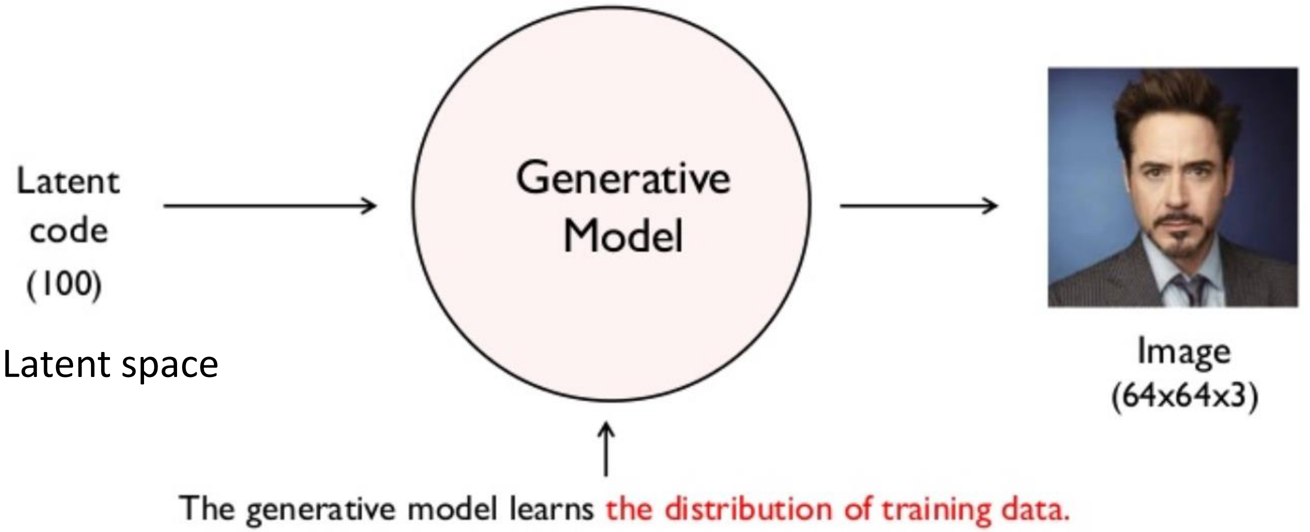
- Discriminative model



Unsupervised Learning

- Generative model





= Latent space

Model Distribution vs. Data Distribution

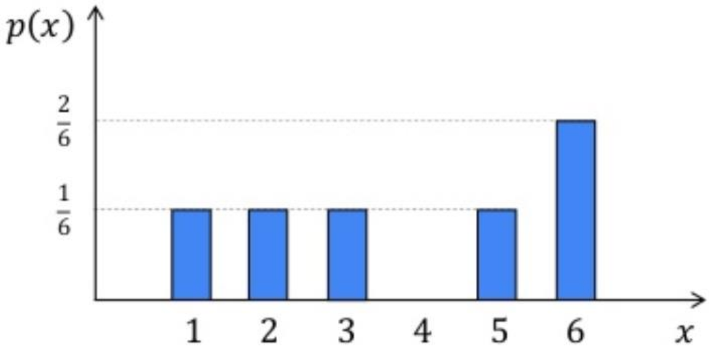
Probability Distribution

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

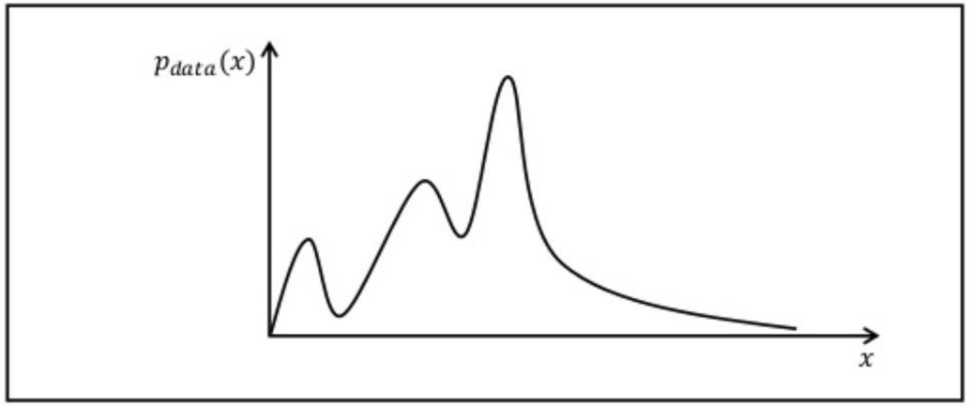
What if x is actual images in the training data?

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

Probability Distribution

Probability density function

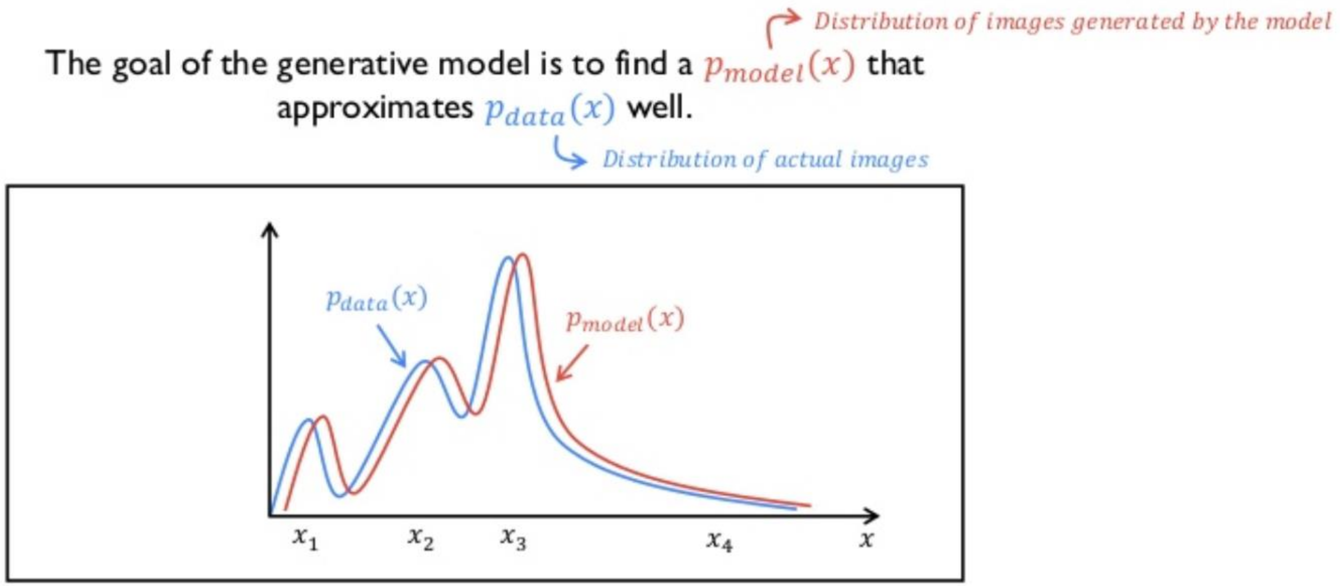
There is a $p_{data}(x)$ that represents the distribution of actual images.



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

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 $P_{model}(x)$

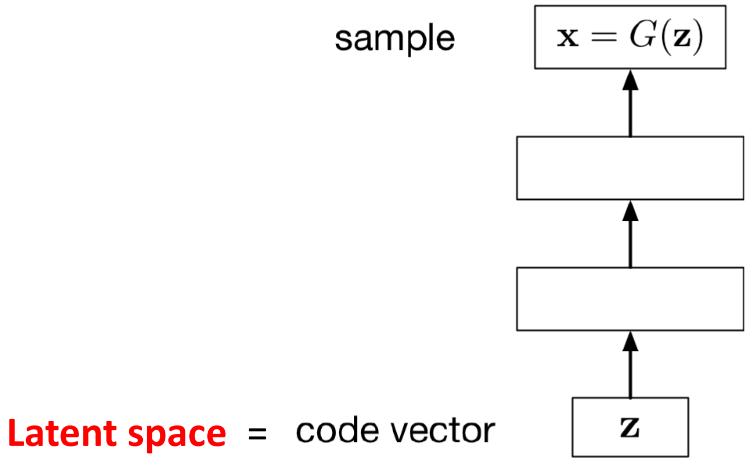


https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Generative Models from Lower Dimension

- Learn transformation via a neural network

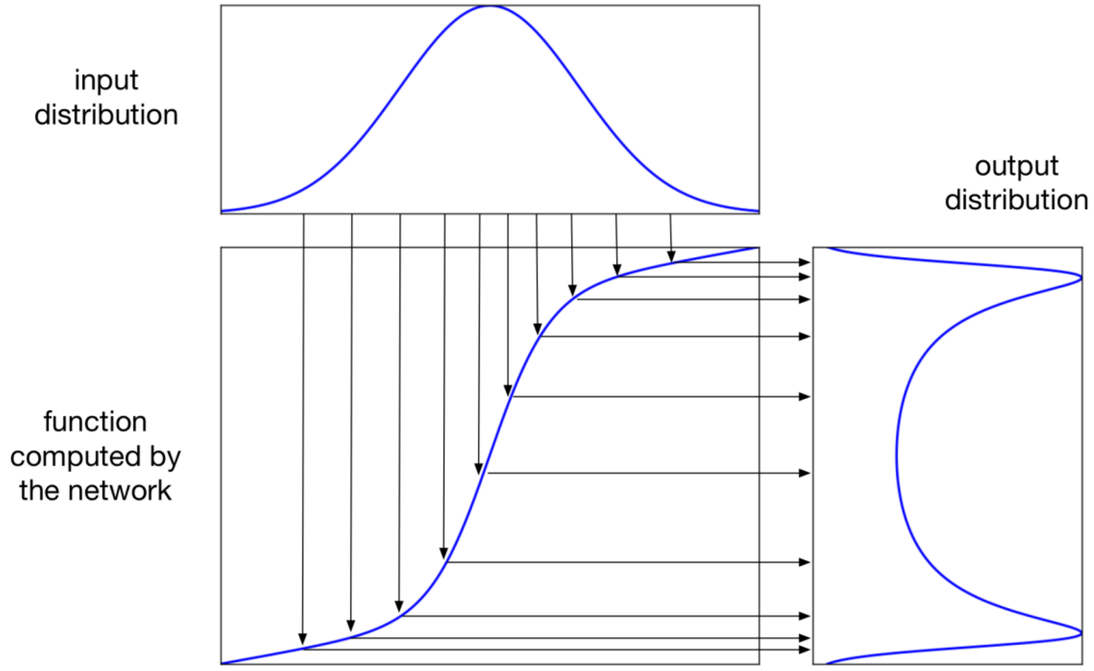
- Start by sampling the code vector z from a fixed, simple distribution (e.g. uniform distribution or Gaussian distribution)
- Then this code vector is passed as input to a deterministic generator network G, which produces an output sample $x = G(z)$



Source: Prof. Roger Grosse at U of Toronto

Deterministic Transformation (by Network)

- 1-dimensional example:

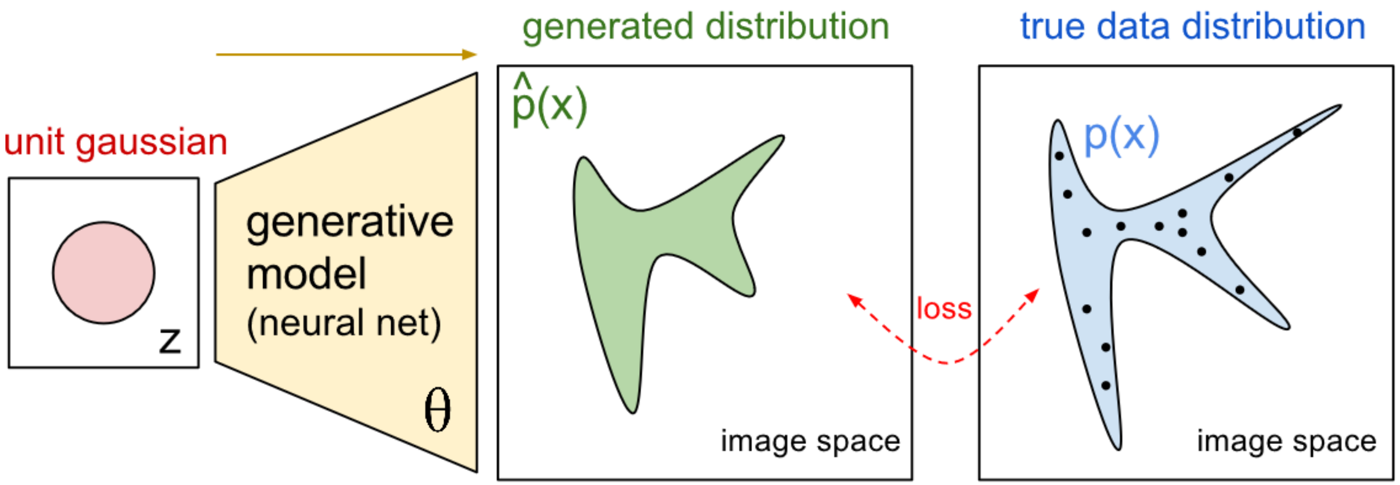


- Remember
  - Network does not generate distribution, but
  - It maps known distribution to target distribution

Source: Prof. Roger Grosse at U of Toronto

Deterministic Transformation (by Network)

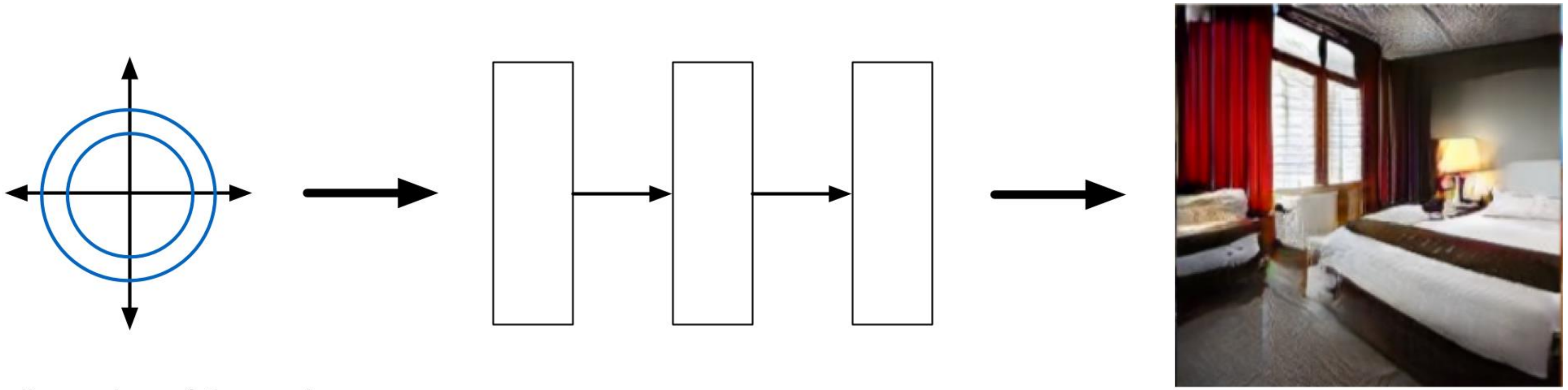
- High dimensional example:



Source: Prof. Roger Grosse at U of Toronto

Prob. Density Function by Deep Learning

- Generative model of image



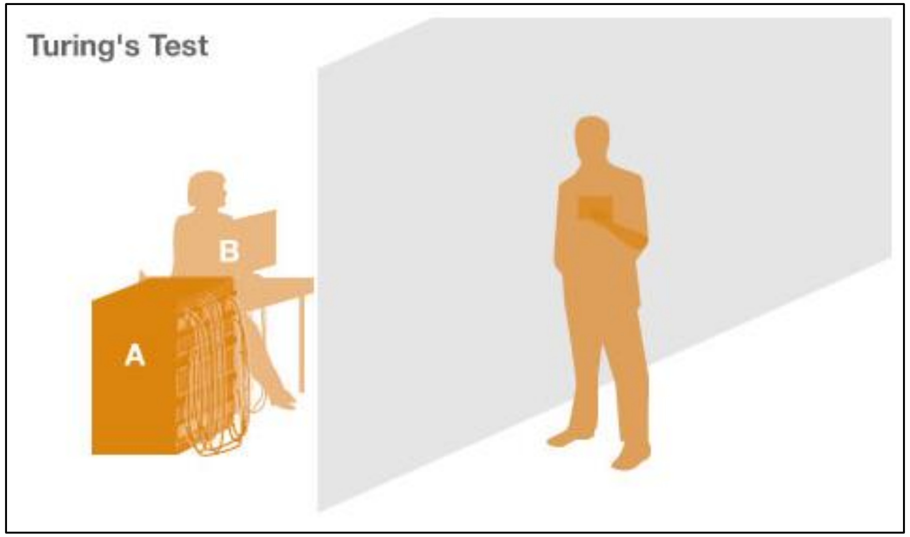
Each dimension of the code vector is sampled independently from a simple distribution, e.g. Gaussian or uniform.

This is fed to a (deterministic) generator network.

The network outputs an image.

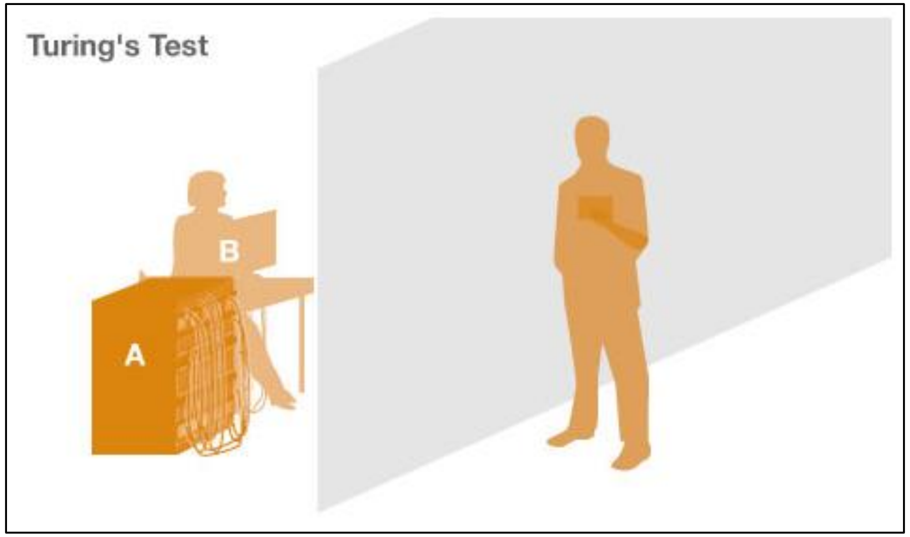
Generative Adversarial Networks (GANs)

- 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



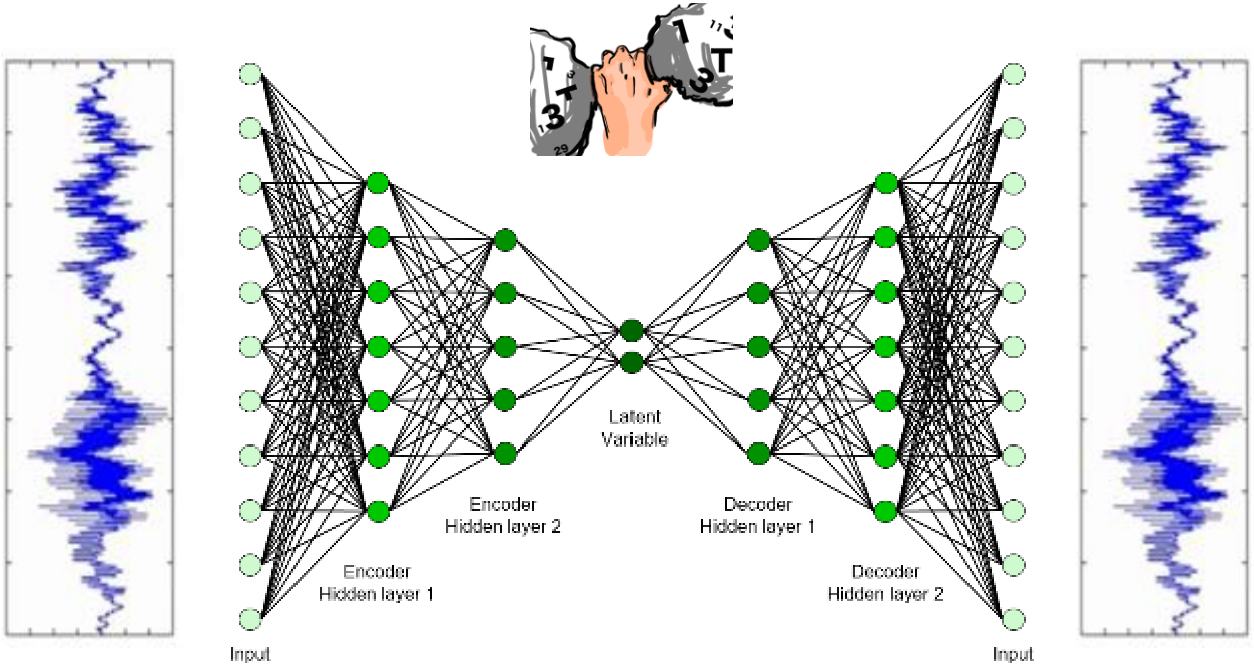
Generative Adversarial Networks (GAN)

- 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

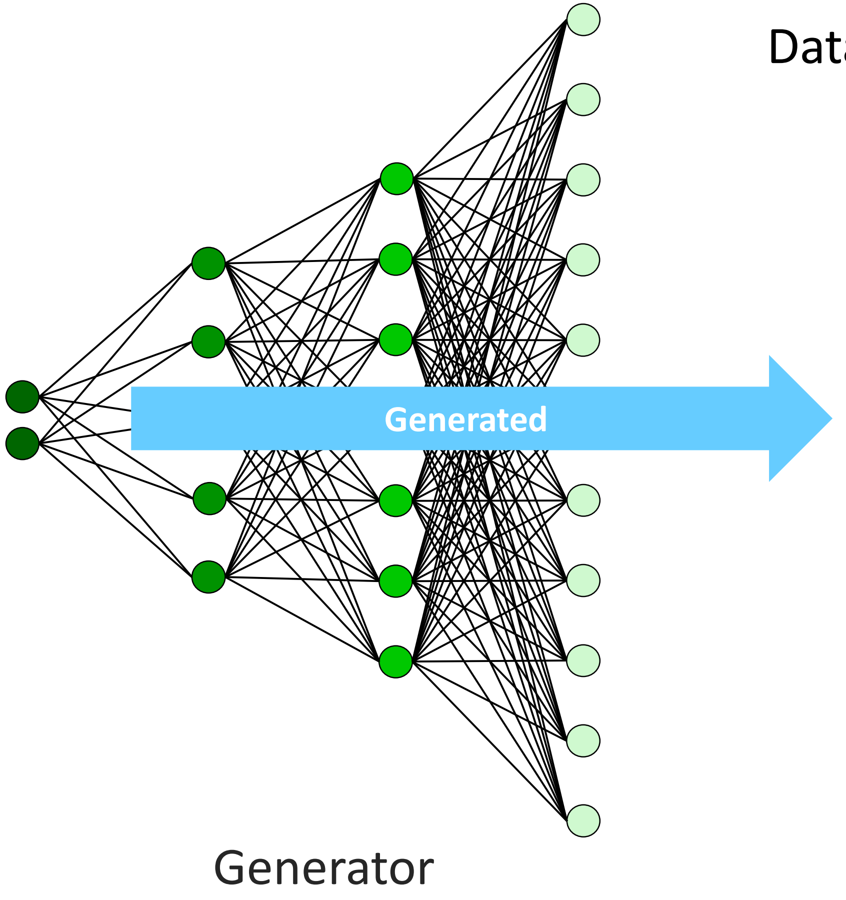
Autoencoder

- 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



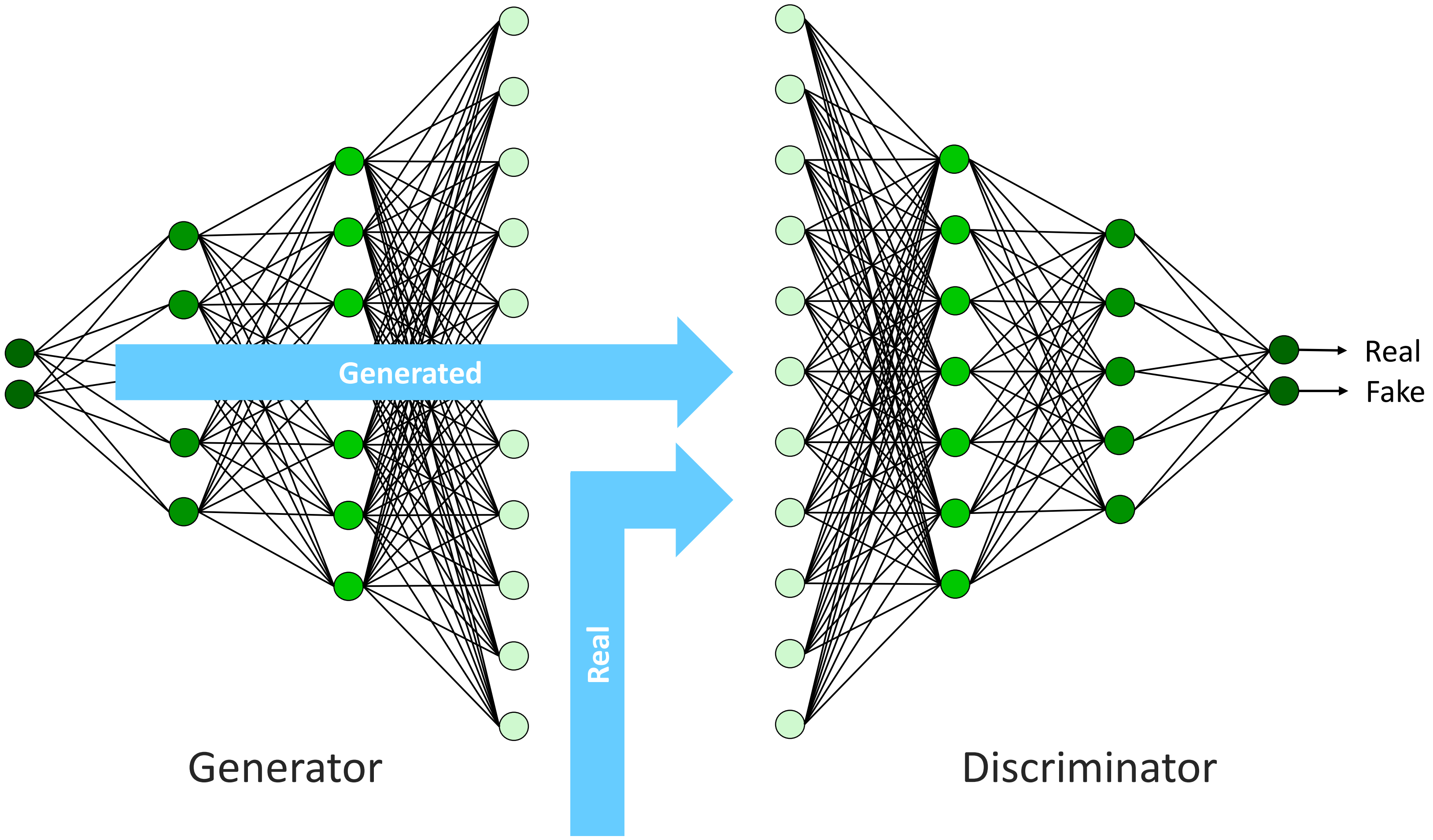
Generative Adversarial Networks (GAN)

- Analogous to Turing Test

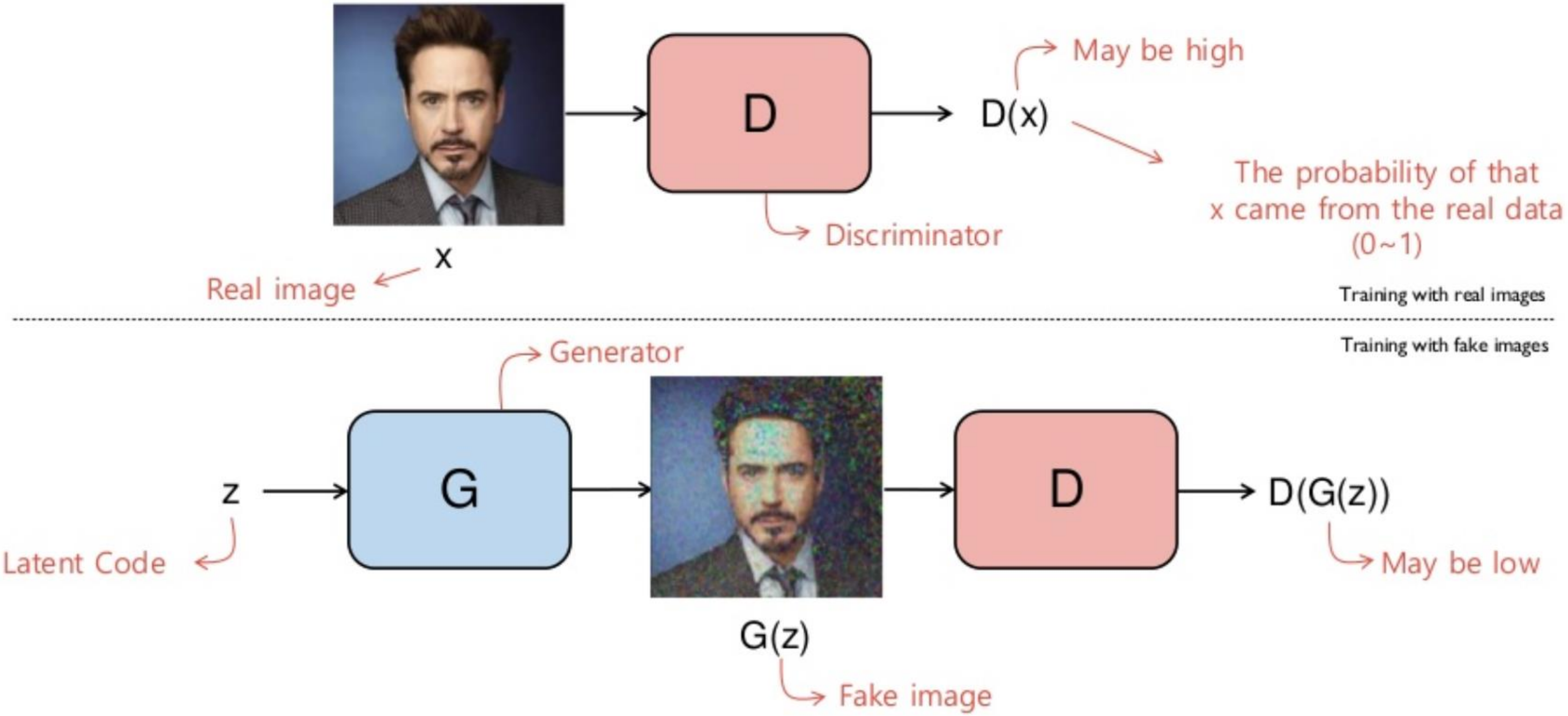


Generative Adversarial Networks (GAN)

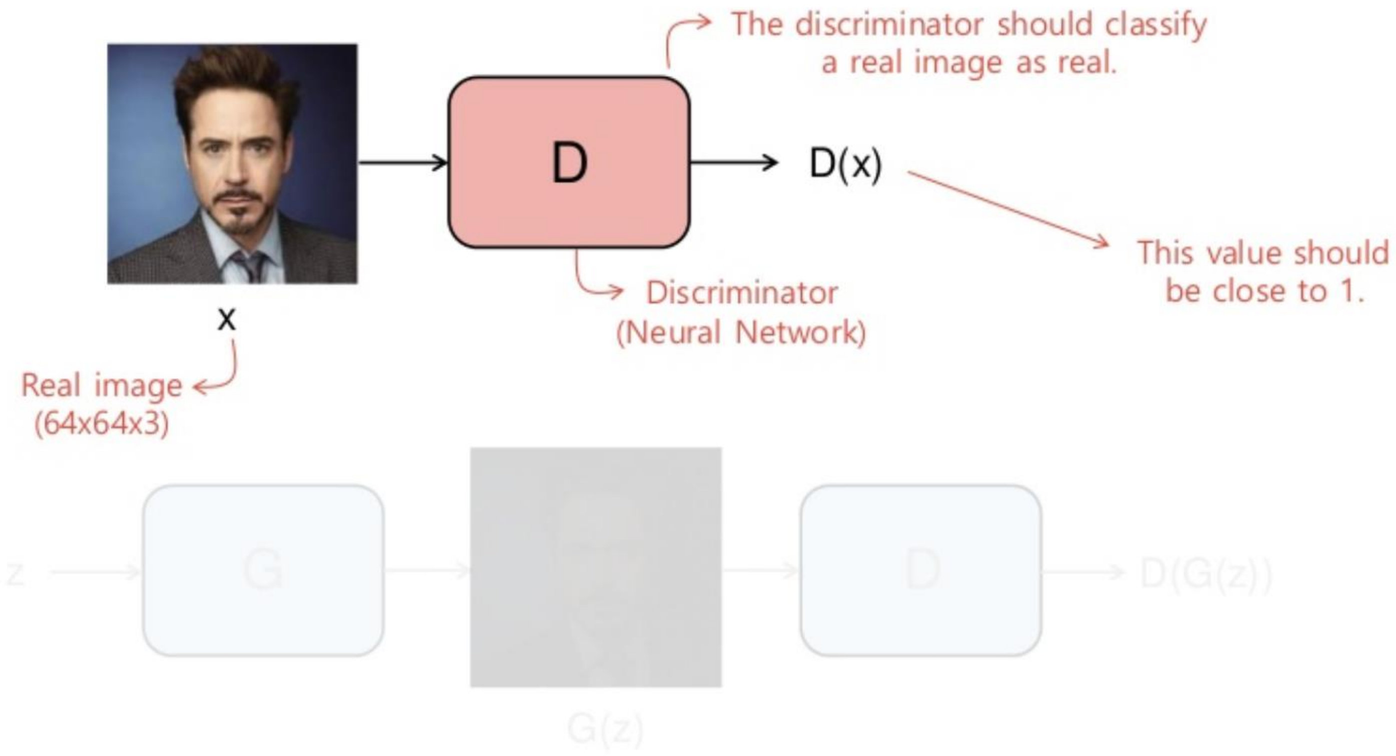
- Analogous to Turing Test



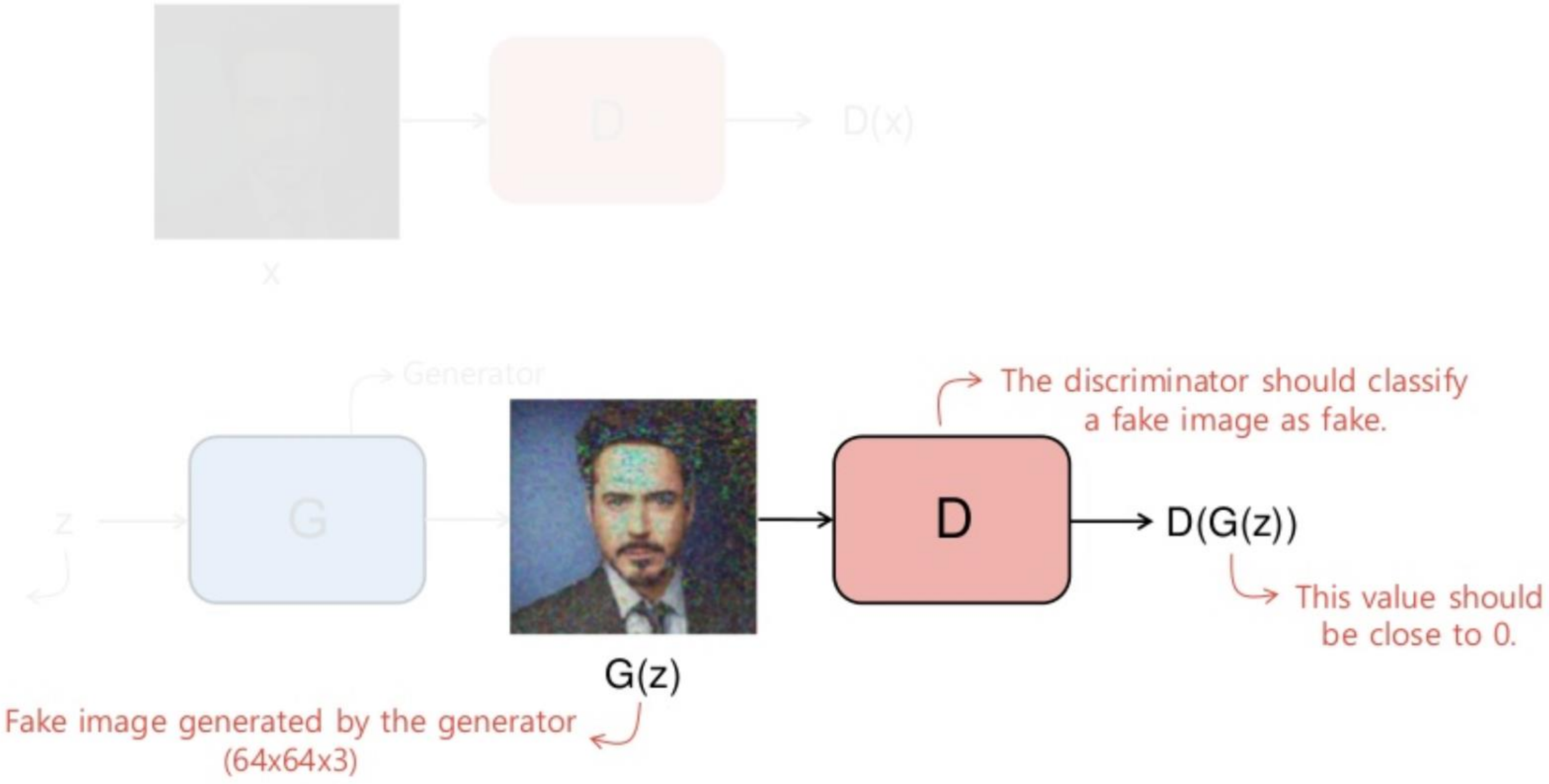
Intuition for GAN



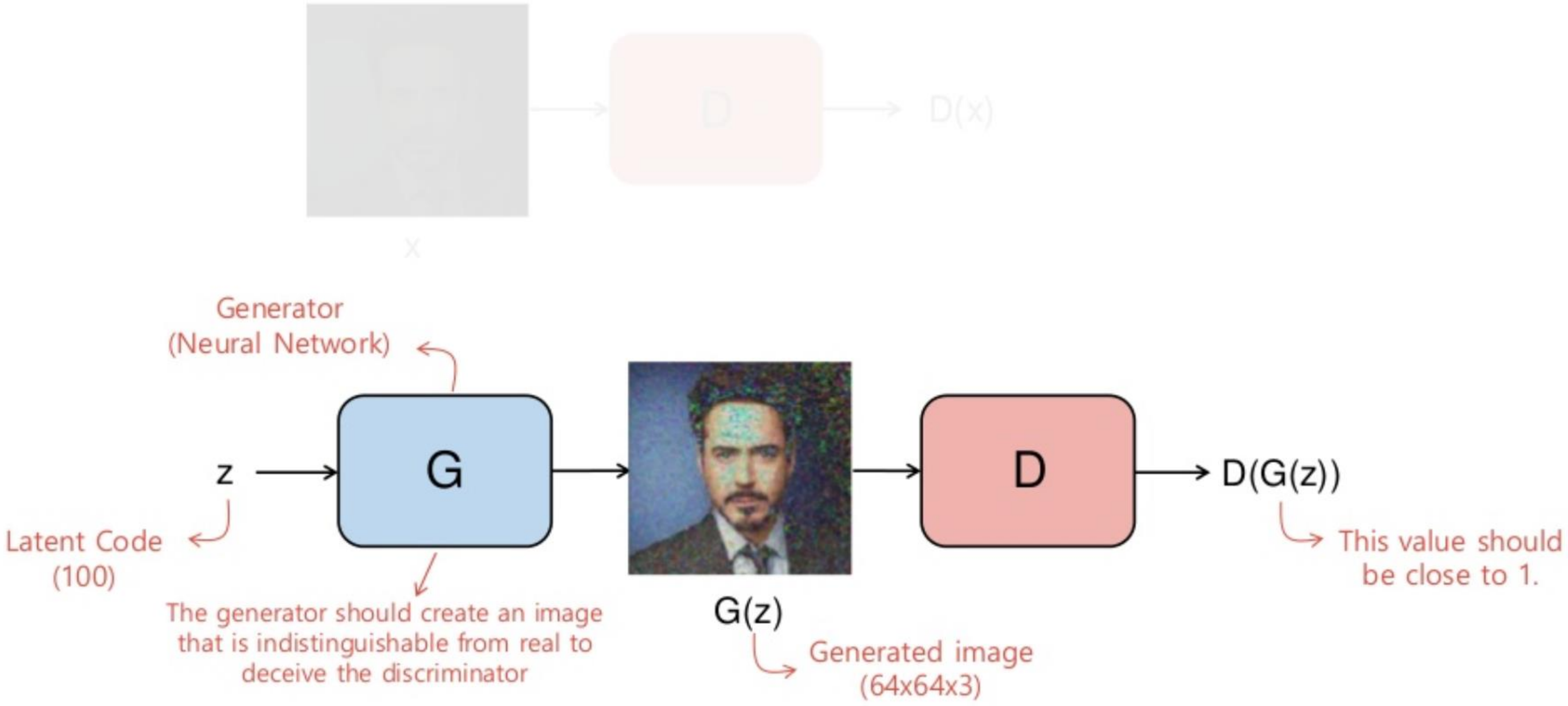
Discriminator Perspective (1/2)



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

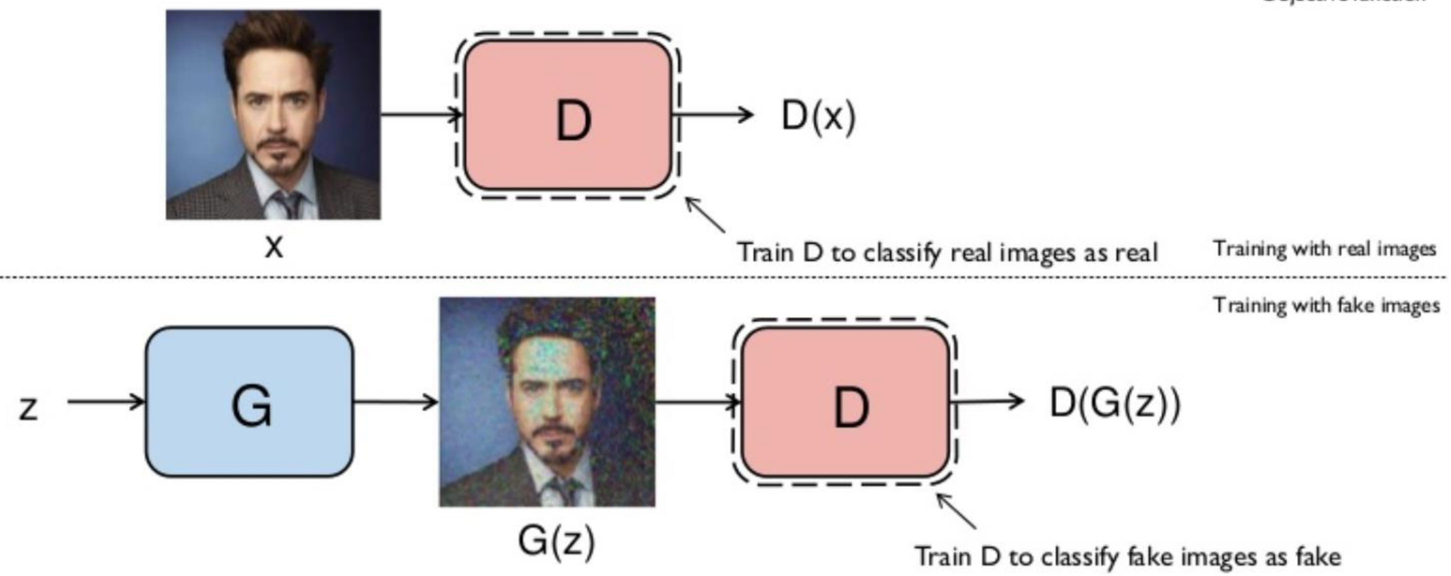
$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

D should maximize $V(D,G)$

Maximum when $D(x) = 1$

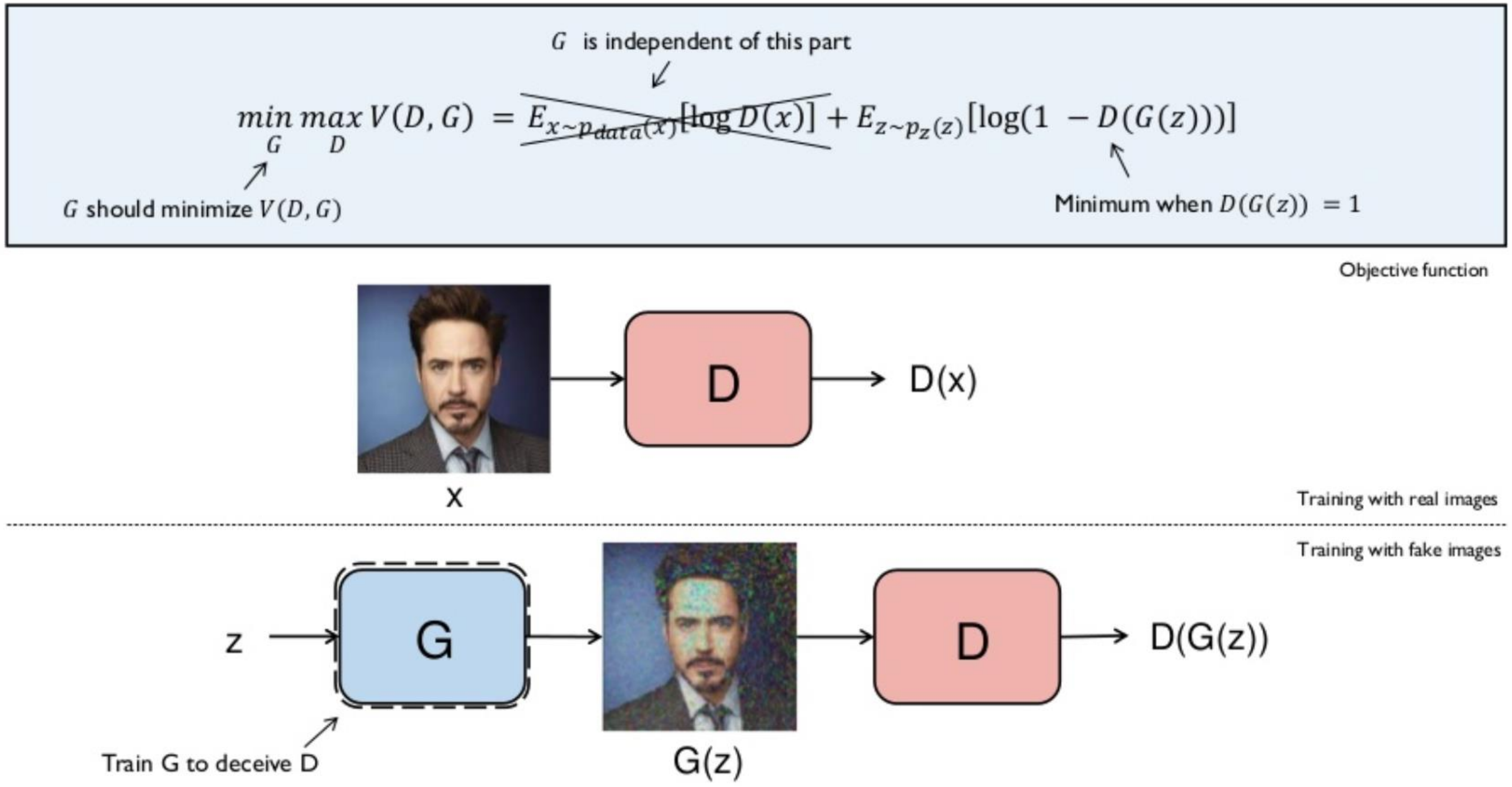
Maximum when $D(G(z)) = 0$

Objective function



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

Loss Function of Generator



Supplementary - Non-Saturating Game

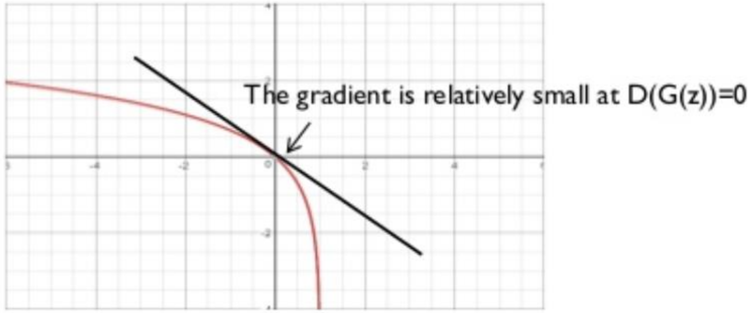
$$\min_G E_{z\sim p_z(z)}[\log(1 - D(G(z))]$$

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



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

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

Supplementary - Non-Saturating Game

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

$\downarrow$ Modification (heuristically motivated)

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

- Practical Usage

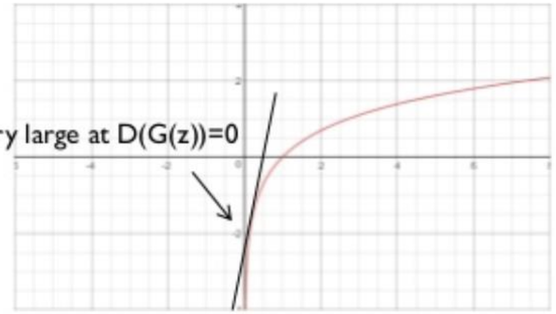
Use binary cross entropy loss function with fake label (1)

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

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

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

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

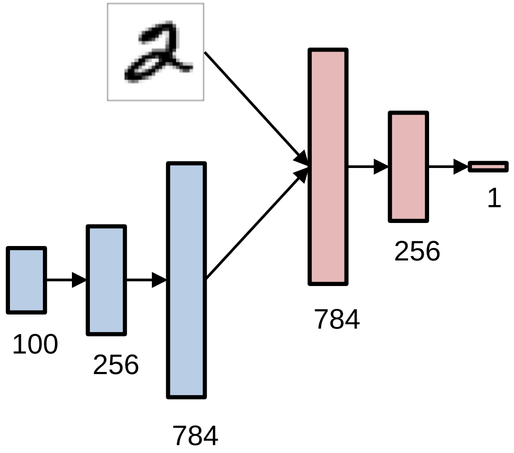
$$\min_D E_{x\sim p_{\text{data}}(x)}\left[-\log D(x)\right] + E_{x\sim p_z(z)}\left[-\log(1-D(G(z)))\right]$$

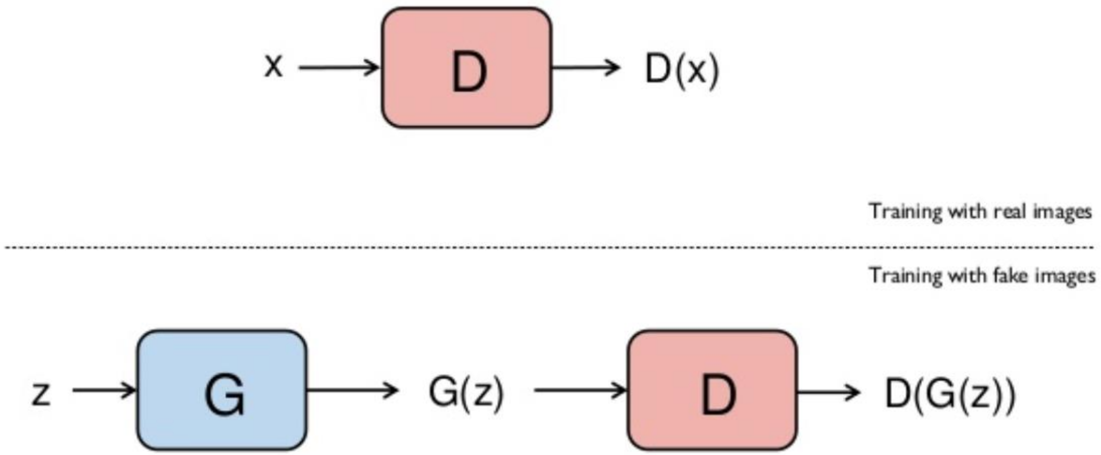
Step 2: Fix D and perform a gradient step to

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

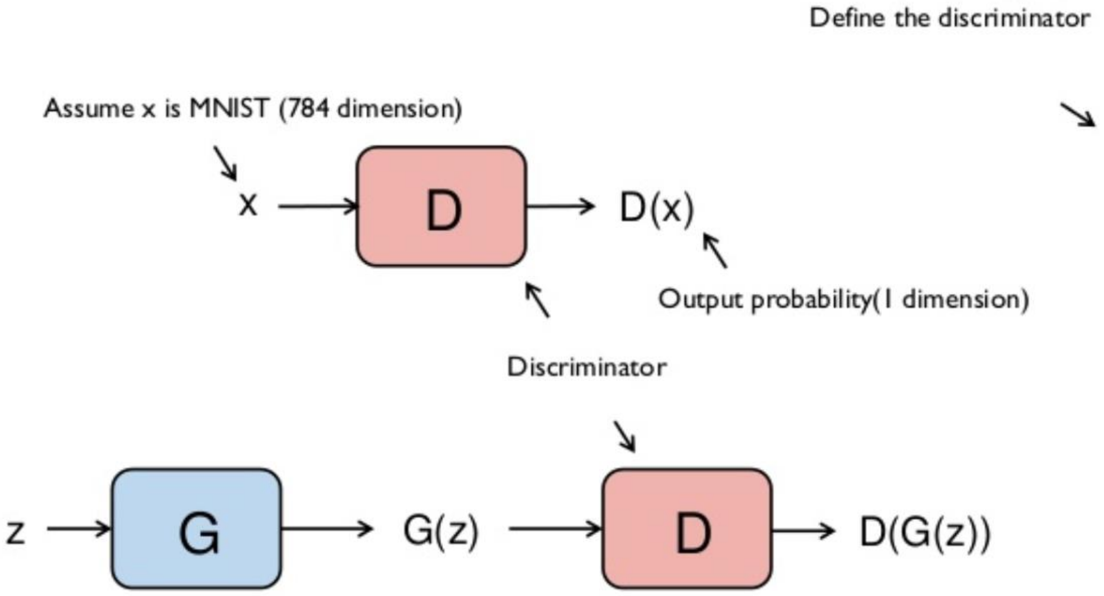
GAN Implementation in TensorFlow

TensorFlow Implementation



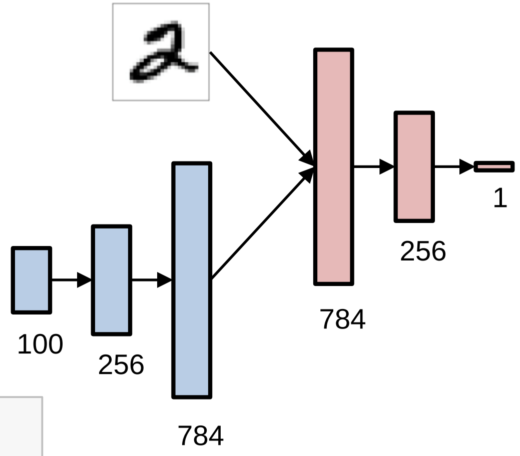


TensorFlow Implementation



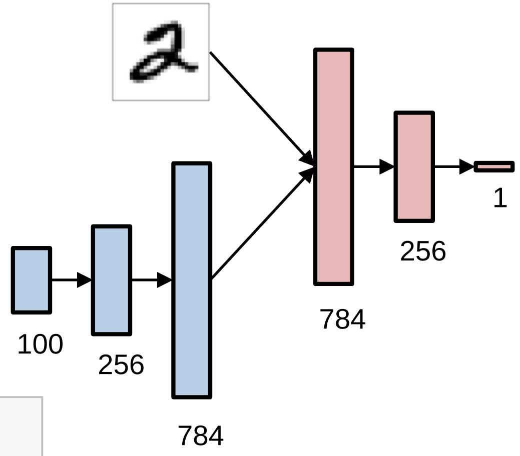

```
n_D_input = 28*28
n_D_hidden = 256
n_D_output = 1

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



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

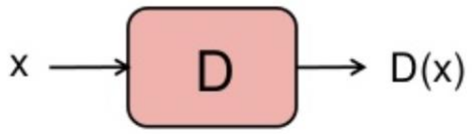
TensorFlow Implementation

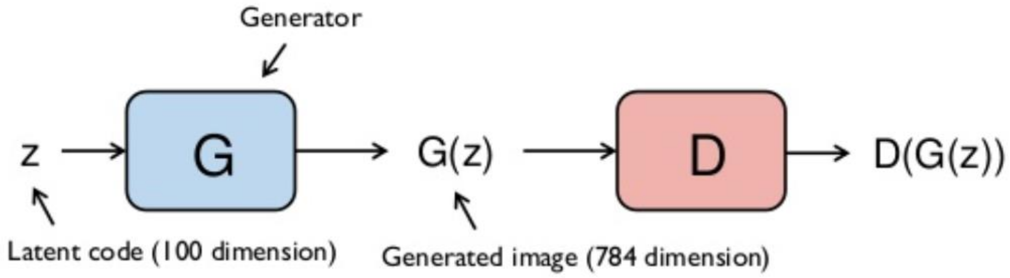


Define the generator

```
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
```





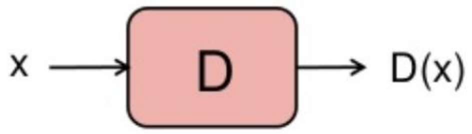
TensorFlow Implementation

Step 1: Fix G and perform a gradient step to

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

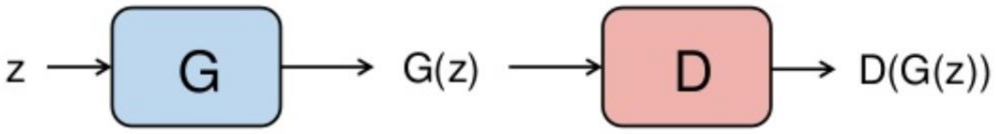
Step 2: Fix D and perform a gradient step to

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



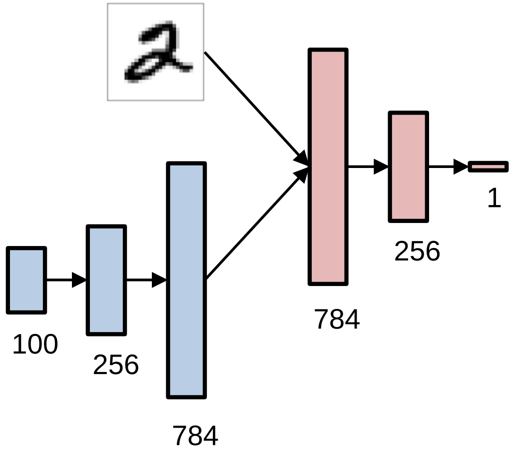

```
D_loss = tf.reduce_mean(- tf.log(D_real) - tf.log(1 - D_fake))
G_loss = tf.reduce_mean(- tf.log(D_fake))
```

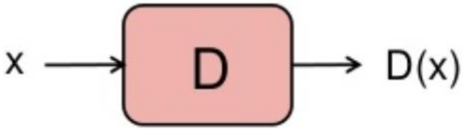
```
D_var_list = [weights['D1'], biases['D1'], weights['D2'], biases['D2']]
G_var_list = [weights['G1'], biases['G1'], weights['G2'], biases['G2']]
```



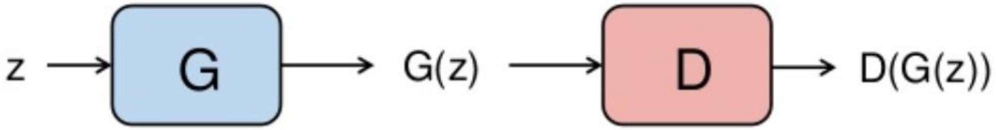

```
LR = 0.0002
D_optm = tf.train.AdamOptimizer(LR).minimize(D_loss, var_list = D_var_list)
G_optm = tf.train.AdamOptimizer(LR).minimize(G_loss, var_list = G_var_list)
```

TensorFlow Implementation



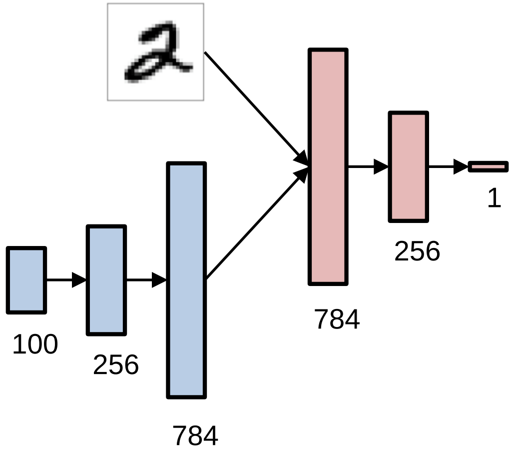


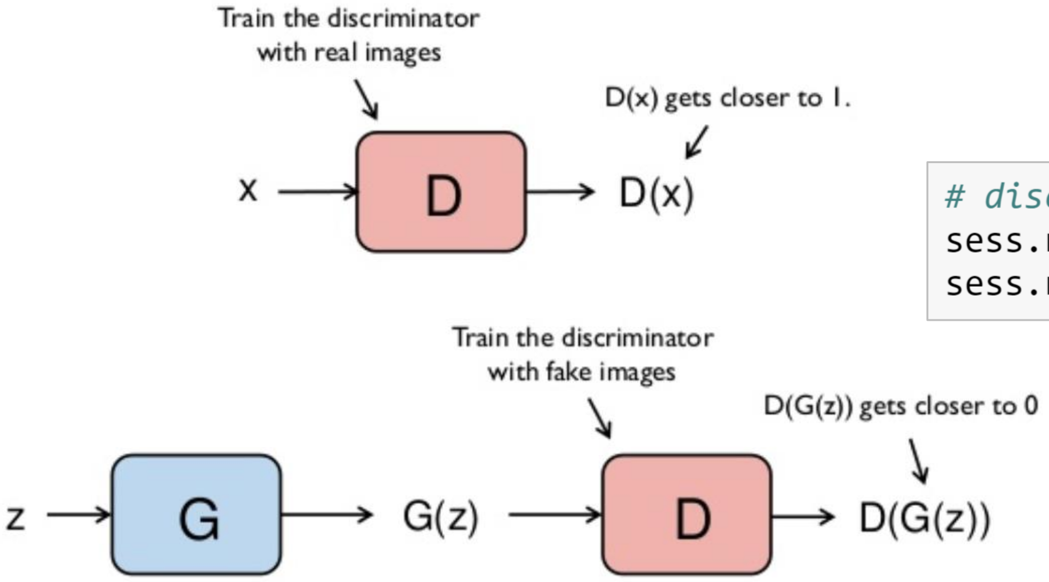
x is a tensor of shape (batch_size, 784).
z is a tensor of shape (batch_size, 100).




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

TensorFlow Implementation

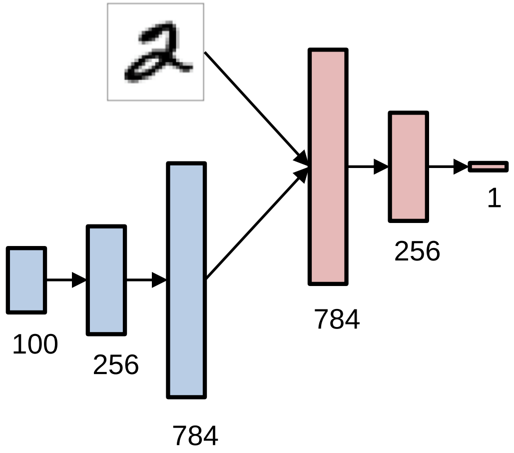


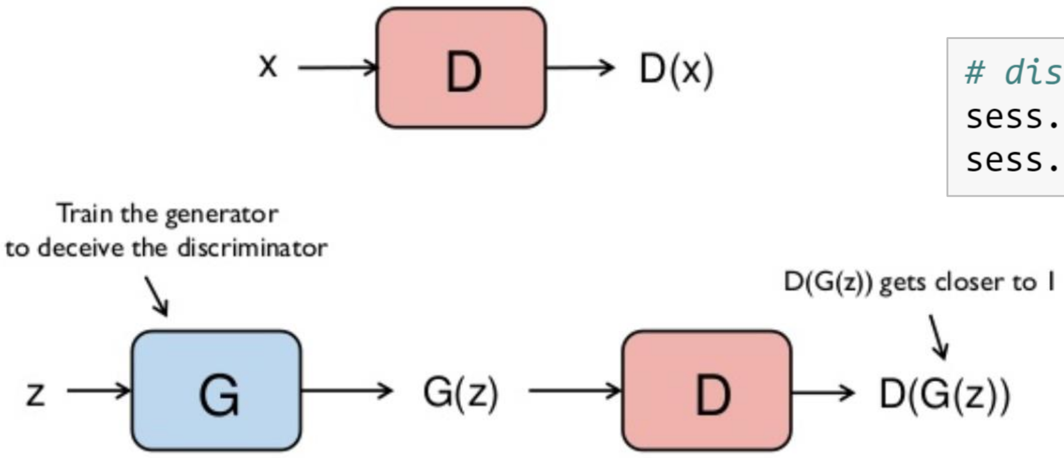



```python
# discriminator and generator are separately trained
sess.run(D_optm, feed_dict = {x: train_x, z: noise})
sess.run(G_optm, feed_dict = {z: noise})
```


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

TensorFlow Implementation

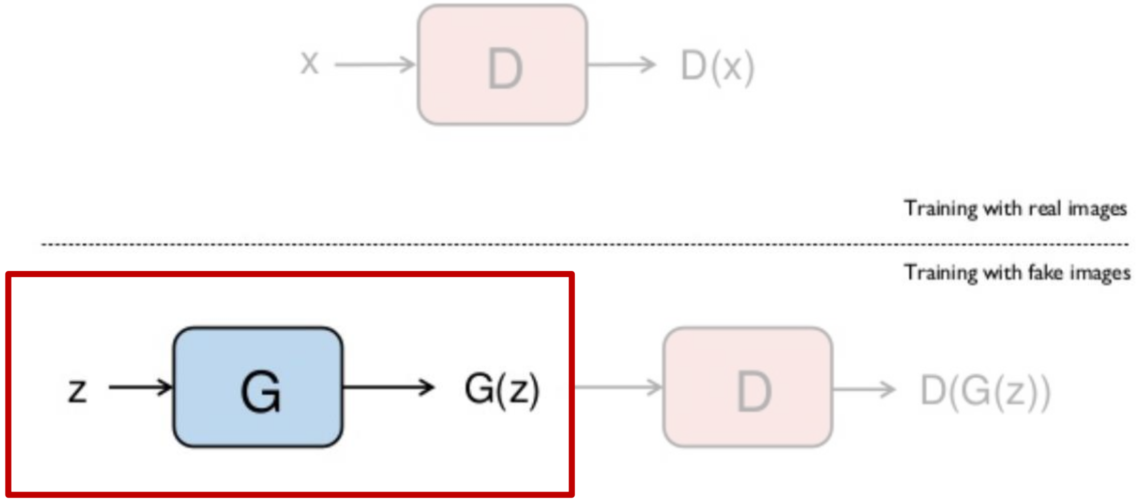


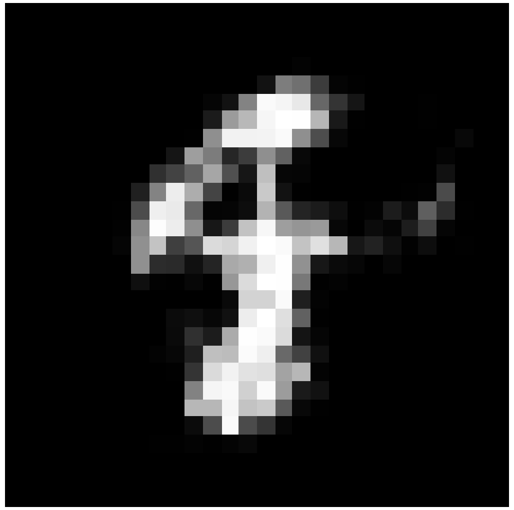



```
# discriminator and generator are separately trained
sess.run(D_optm, feed_dict = {x: train_x, z: noise})
sess.run(G_optm, feed_dict = {z: noise})
```

After Training

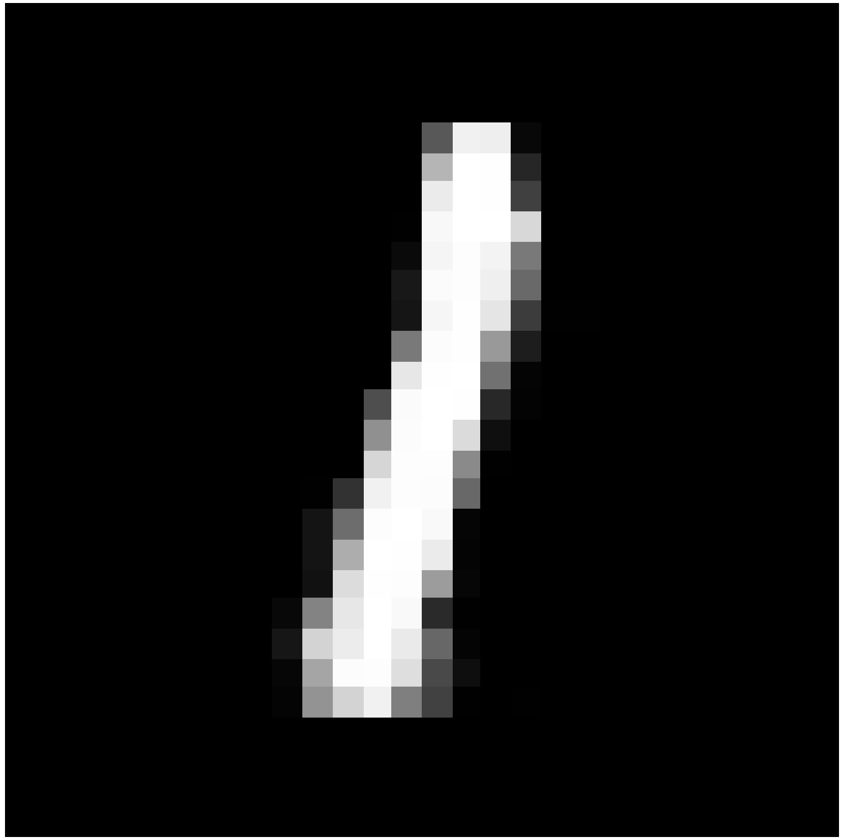
- After training, use generator network to generate new data

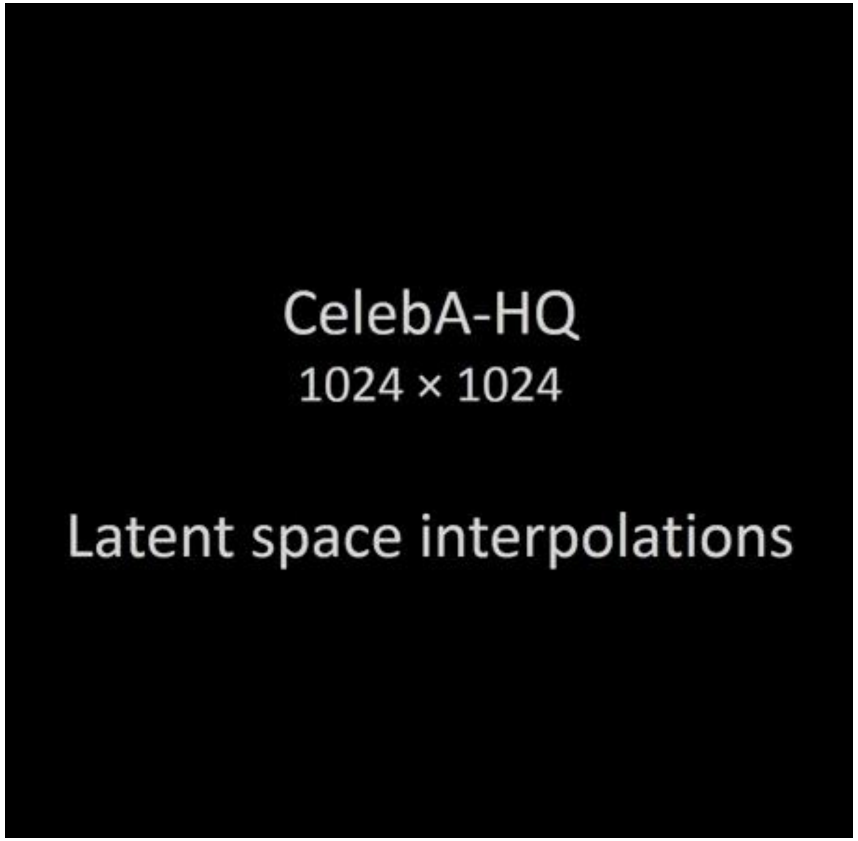




```
noise = make_noise(n_batch, n_G_input)
G_img = sess.run(G_output, feed_dict = {z: noise})
```

GAN Samples





Conditional GAN

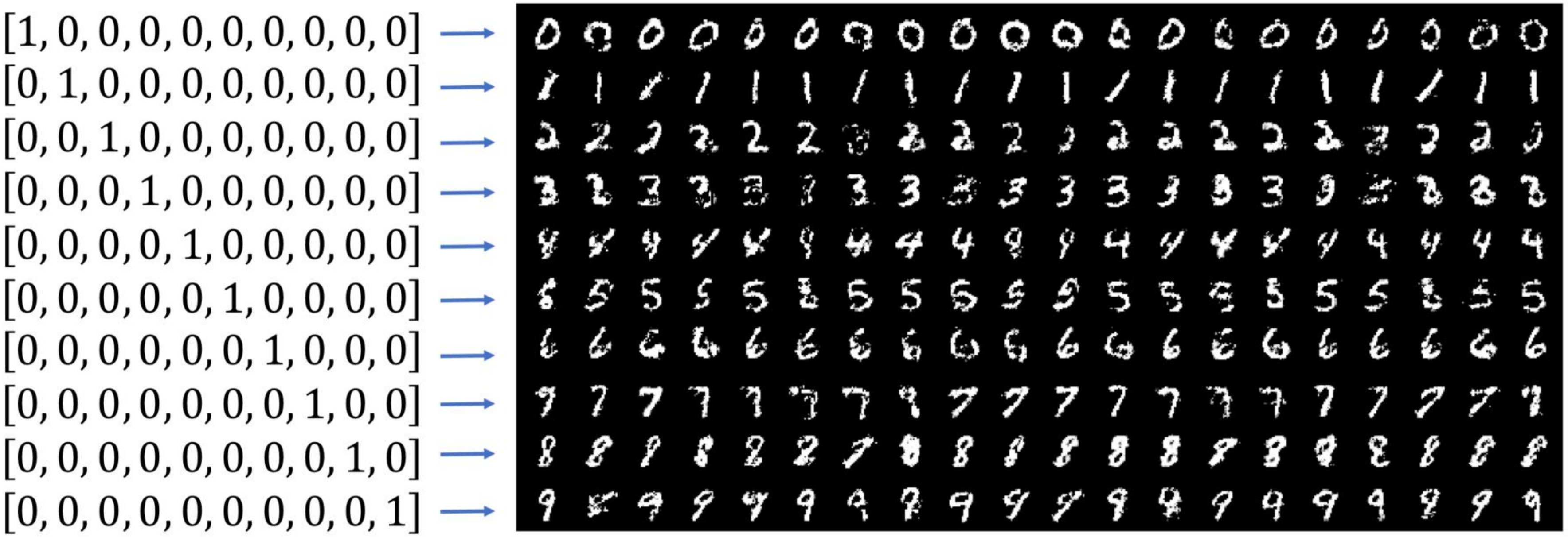
Conditional GAN

- In an unconditioned generative model, there is no control on modes of the data being generated.

- In the Conditional GAN (CGAN), the generator learns to generate a fake sample with a specific condition or characteristics (such as a label associated with an image or more detailed tag) rather than a generic sample from unknown noise distribution.

Conditional GAN

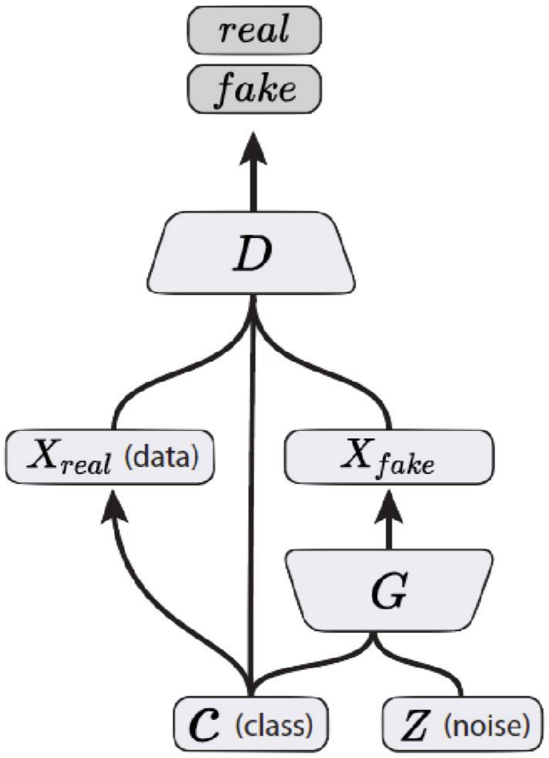
- MNIST digits generated conditioned on their class label



Conditional GAN

- Simple modification to the original GAN framework that conditions the model on additional information for better multi-modal learning

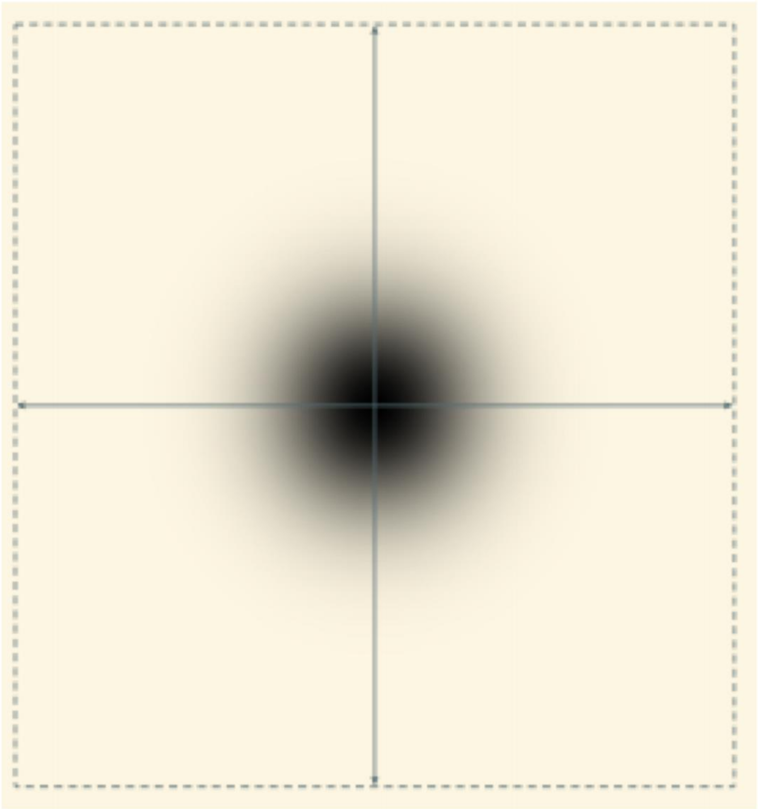
- Many practical applications of GANs when we have explicit supervision available



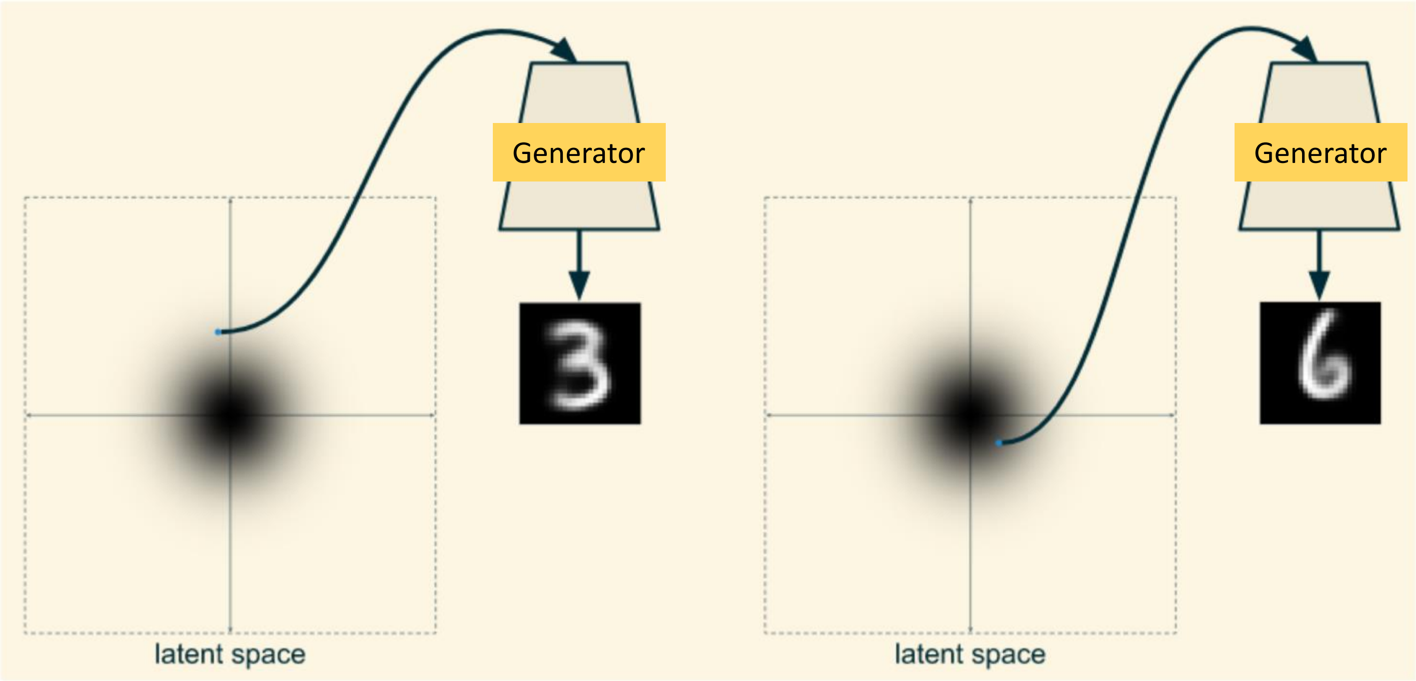
Conditional GAN
(Mirza & Osindero, 2014)

Normal Distribution of MNIST

- A standard normal distribution
- This is how we would like points corresponding to MNIST digit images to be distributed in the latent space

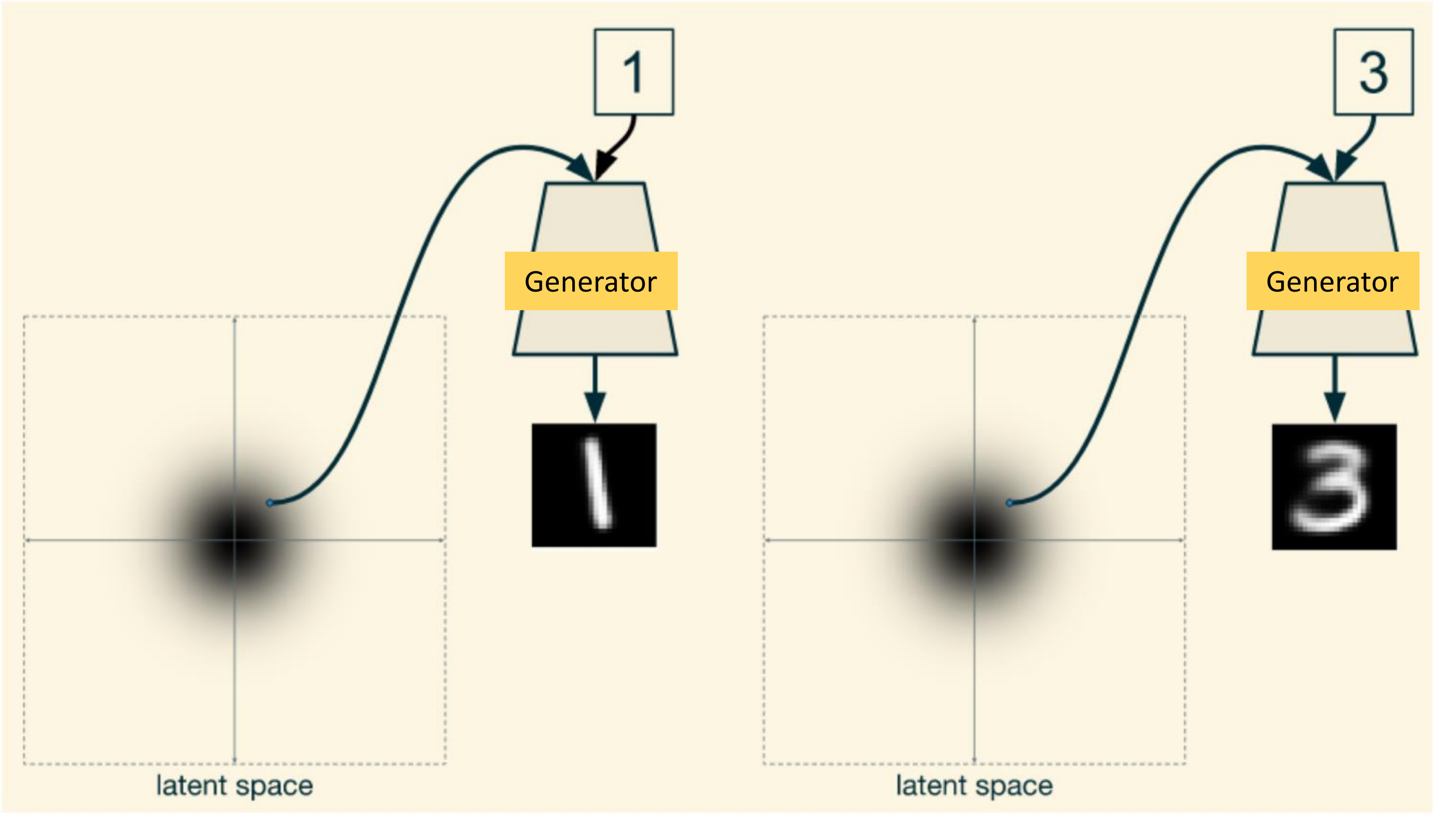


Generator at GAN



Generator at Conditional GAN

- Feed a random point in latent space and desired number.
- Even if the same latent point is used for two different numbers, the process will work correctly since the latent space only encodes features such as stroke width or angle

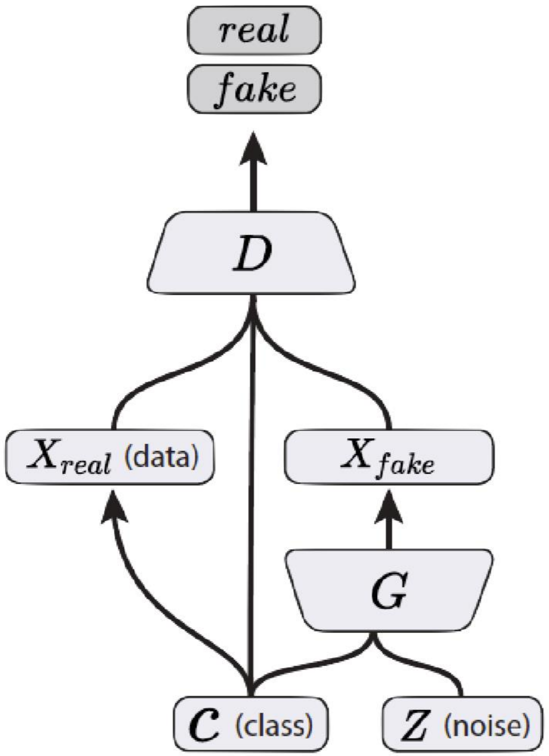


CGAN Implementation

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

n_G_input = 128
n_G_hidden = 256
n_G_output = 28*28

n_label = 10 # one-hot-encoding
```

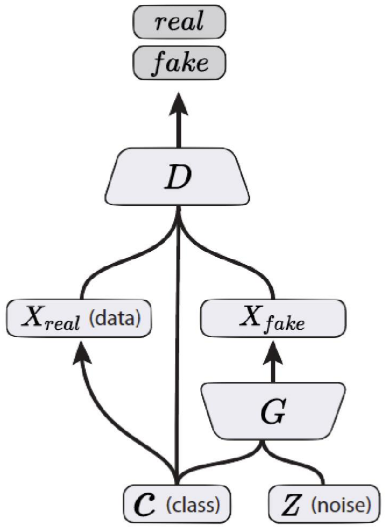


Conditional GAN
(Mirza & Osindero, 2014)

CGAN Implementation

```
weights = {
    'G1' : tf.Variable(tf.random_normal([n_G_input + n_label, n_G_hidden], stddev = 0.01)),
    'G2' : tf.Variable(tf.random_normal([n_G_hidden, n_G_output], stddev = 0.01)),
    'D1' : tf.Variable(tf.random_normal([n_D_input + n_label, n_D_hidden], stddev = 0.01)),
    'D2' : tf.Variable(tf.random_normal([n_D_hidden, n_D_output], stddev = 0.01))
}

biases = {
    'G1' : tf.Variable(tf.zeros([n_G_hidden])),
    'G2' : tf.Variable(tf.zeros([n_G_output])),
    'D1' : tf.Variable(tf.zeros([n_D_hidden])),
    'D2' : tf.Variable(tf.zeros([n_D_output]))
}
```



Conditional GAN
(Mirza & Osindero, 2014)

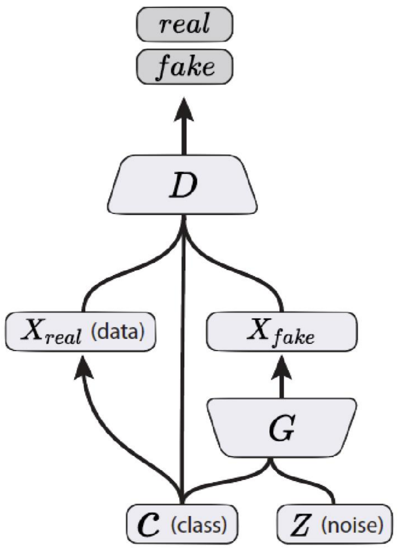
```
z = tf.placeholder(tf.float32, [None, n_G_input])
x = tf.placeholder(tf.float32, [None, n_D_input])
c = tf.placeholder(tf.float32, [None, n_label])
```

CGAN Implementation

```python
def generator(G_input, label, weights, biases):
    hidden = tf.nn.relu(tf.matmul(tf.concat([G_input, label], 1), weights['G1']) + biases['G1'])
    output = tf.nn.sigmoid(tf.matmul(hidden, weights['G2']) + biases['G2'])
    return output
```

```python
def discriminator(D_input, label, weights, biases):
    hidden = tf.nn.relu(tf.matmul(tf.concat([D_input, label], 1), weights['D1']) + biases['D1'])
    output = tf.nn.sigmoid(tf.matmul(hidden, weights['D2']) + biases['D2'])
    return output
```

```python
G_output = generator(z, c, weights, biases)
D_fake = discriminator(G_output, c, weights, biases)
D_real = discriminator(x, c, weights, biases)
```



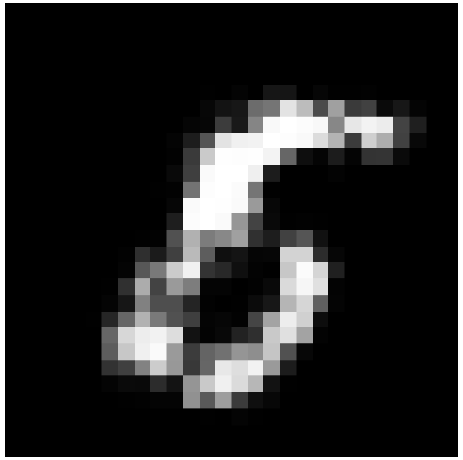
Conditional GAN
(Mirza & Osindero, 2014)

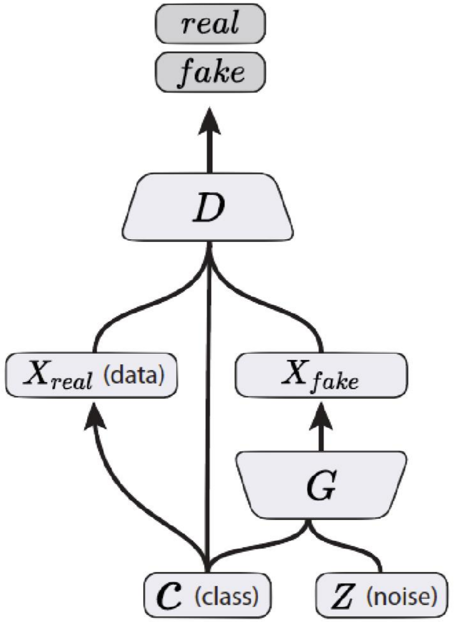
CGAN Implementation

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

- Generate fake MNIST images by CGAN

```
noise = make_noise(1, n_G_input)
G_img = sess.run(G_output, feed_dict = {z: noise, c: [[0,0,0,0,0,1,0,0,0,0]]})
```





Conditional GAN
(Mirza & Osindero, 2014)