Generative Adversarial Network(GANs)

Azka Saddiqa

PHDAIF22M003

Outline

- ✓ What is GAN?
- ✓ Training of GAN
- ✓ Loss function of GAN
- ✓ Optimization
- ✓ Training & Back Propagation via SGD
  Problems in GAN
- ✓ Conditional GAN (C-GAN)
- ✓ Info GAN
- ✓ Cycle GAN

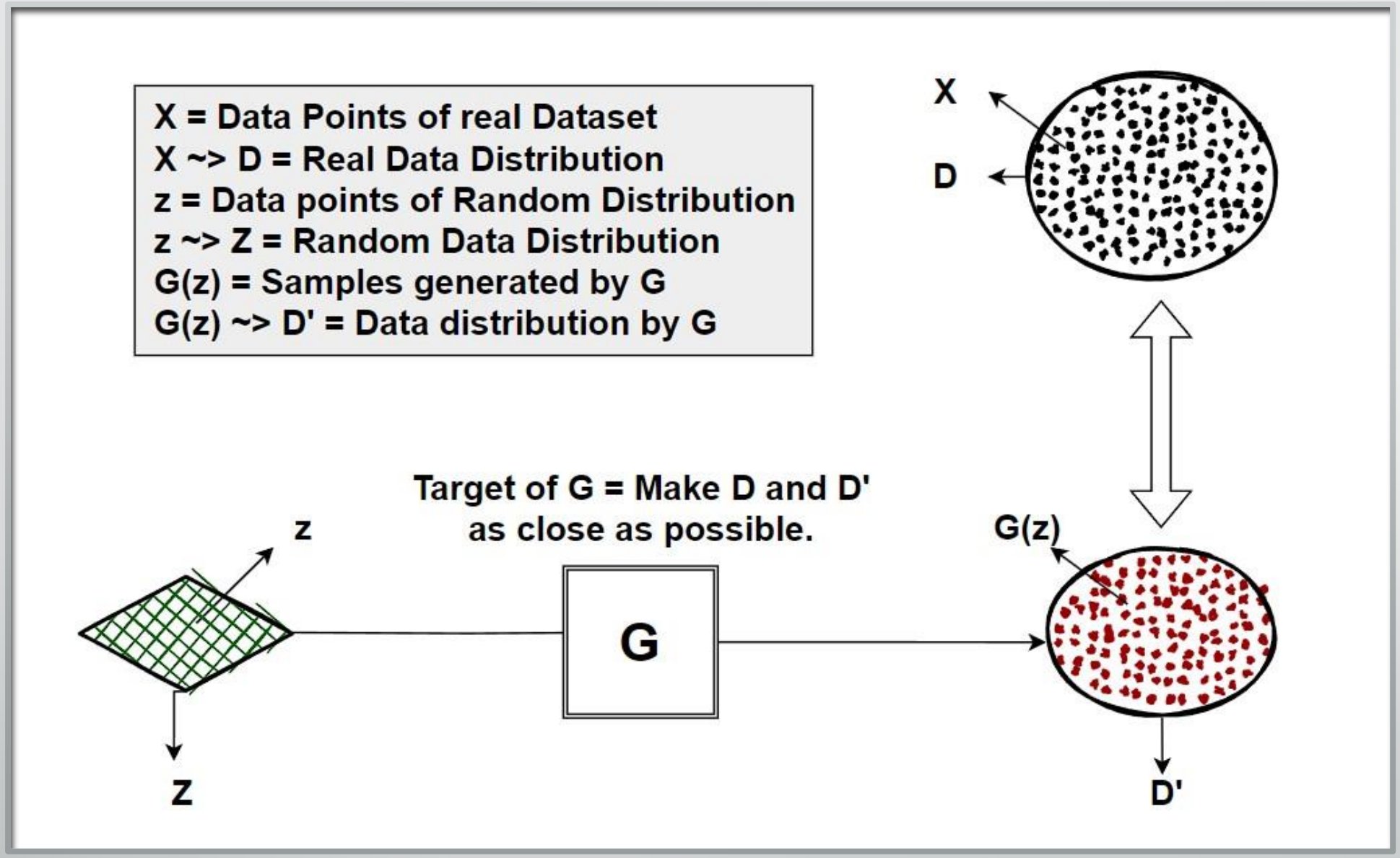
What is GAN?

- GAN stands for Generative Adversarial Network
  - Generative means we generate a probability distribution that should mimics the original data distribution.
  - Adversarial means conflict/opposition.
- GAN's are the deep neural network comprised of two neural networks, competing against each other.
- These two neural networks are called Generator and Discriminator.
- Our aim is to build a deep neural network that are trained in an adversarial manner and learns how to generate the data close to some data distribution.

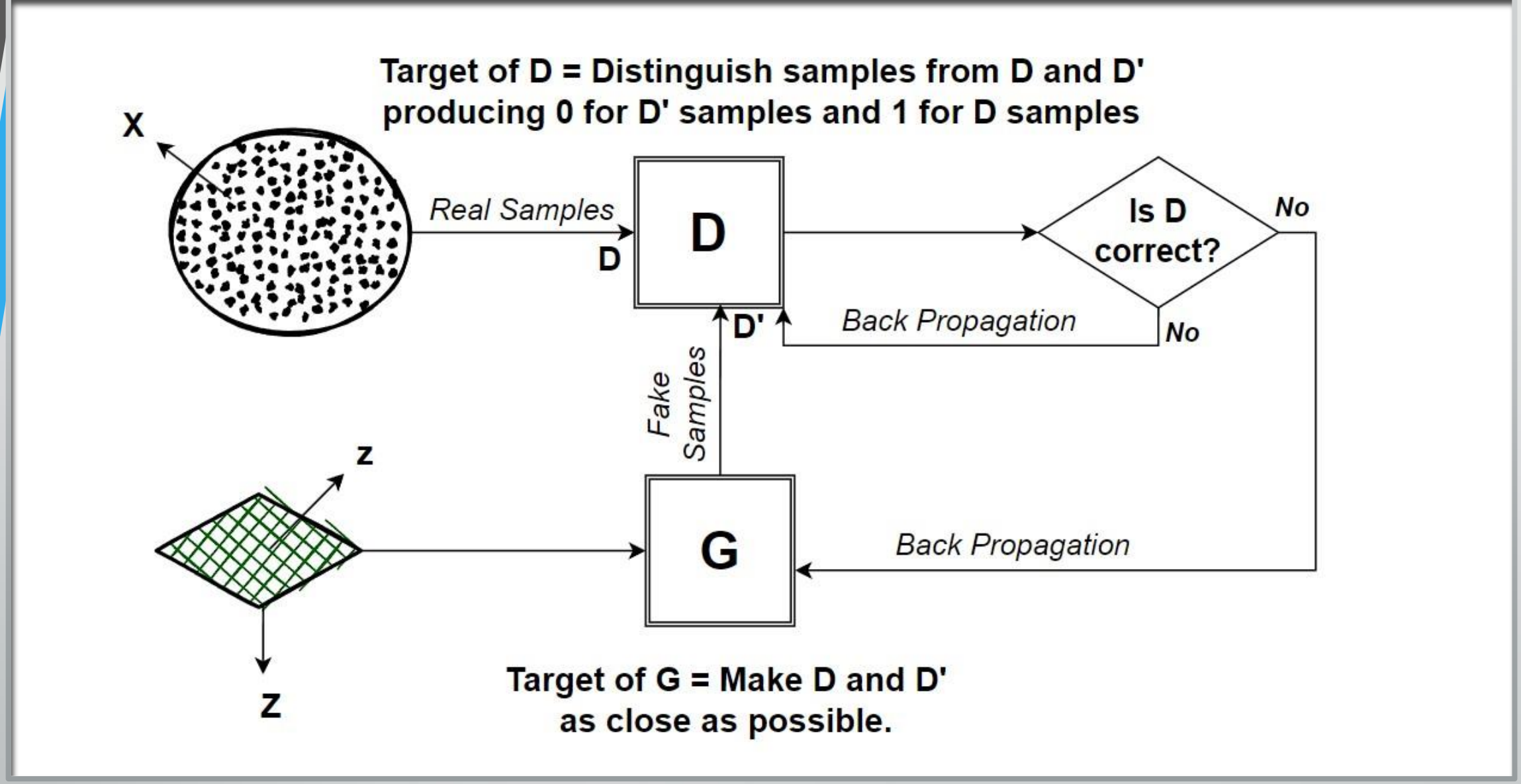
What is GAN?

There exists two classes of models in Machine Learning. Both of them are used by GANs.

- Discriminative Model discriminates between two classes of data. For instance, the image is fake or not. If fake, it produces one(1) else produces zero(0). All the classification problems in Machine Learning is done using these models.
- Generative Model is a model that is trained on the training data X, sampled from true distribution D, given some random distribution Z, produces a distribution D' which is close to D according to the closeness metric.



The **objective** of a **Generator Network** is represented in the figure. D and D′ follows same distribution but contains different samples. X and G(z) are different but D and D′ must be same.



Overall GAN: The Discriminator and generator works as explained below. G produces fake samples and D has to decide whether it is fake or not. If G isn't able to fool D, we need to do back propagation to update weights so that G can do better job next time.

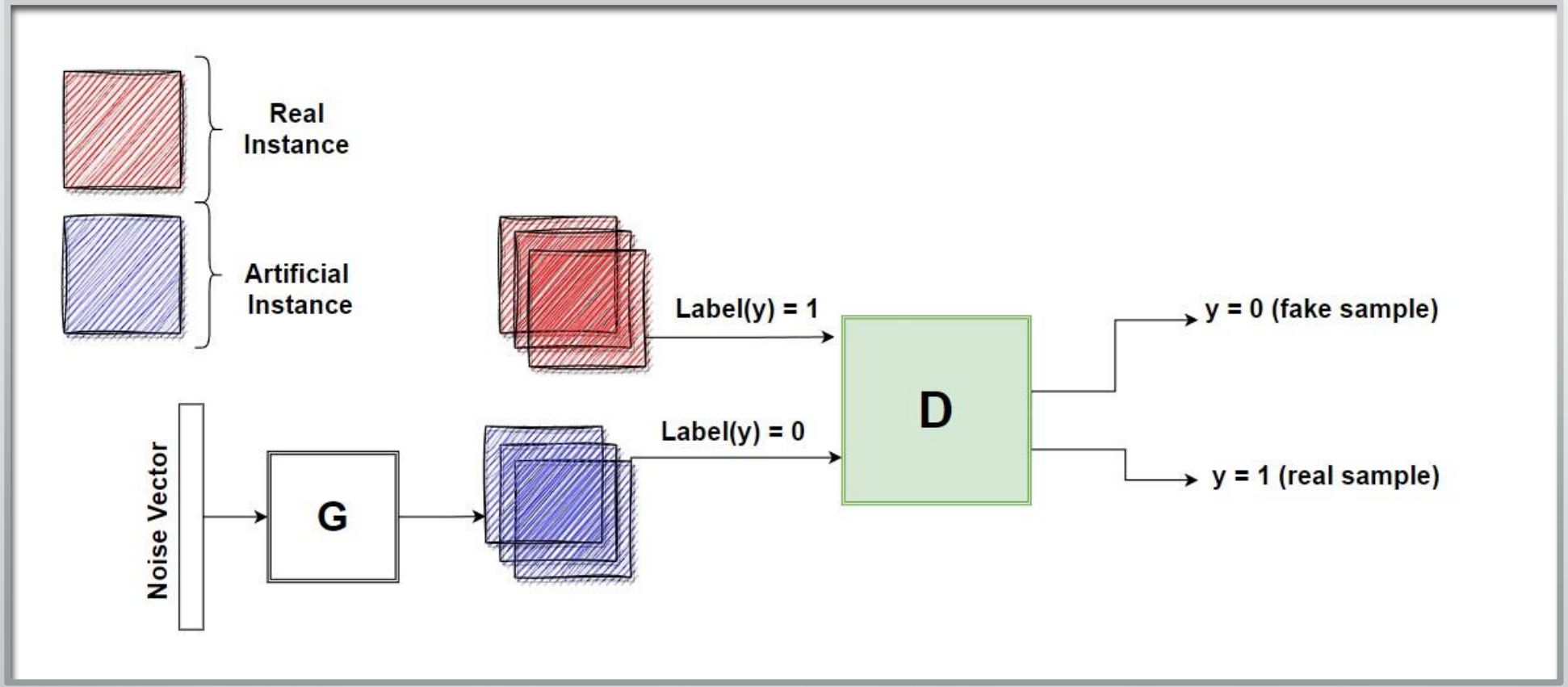
What is GAN?

- When the generator is able to fool the discriminator, i.e. it produces samples that will not be discriminated by discriminator. In that case, discriminator produces an output which is neither zero nor one but **0.5**
- At that time, the discriminator is said to be Optimal Discriminator**(D*)**. Thus, the target of building GAN is achieved.

Training Process of GAN

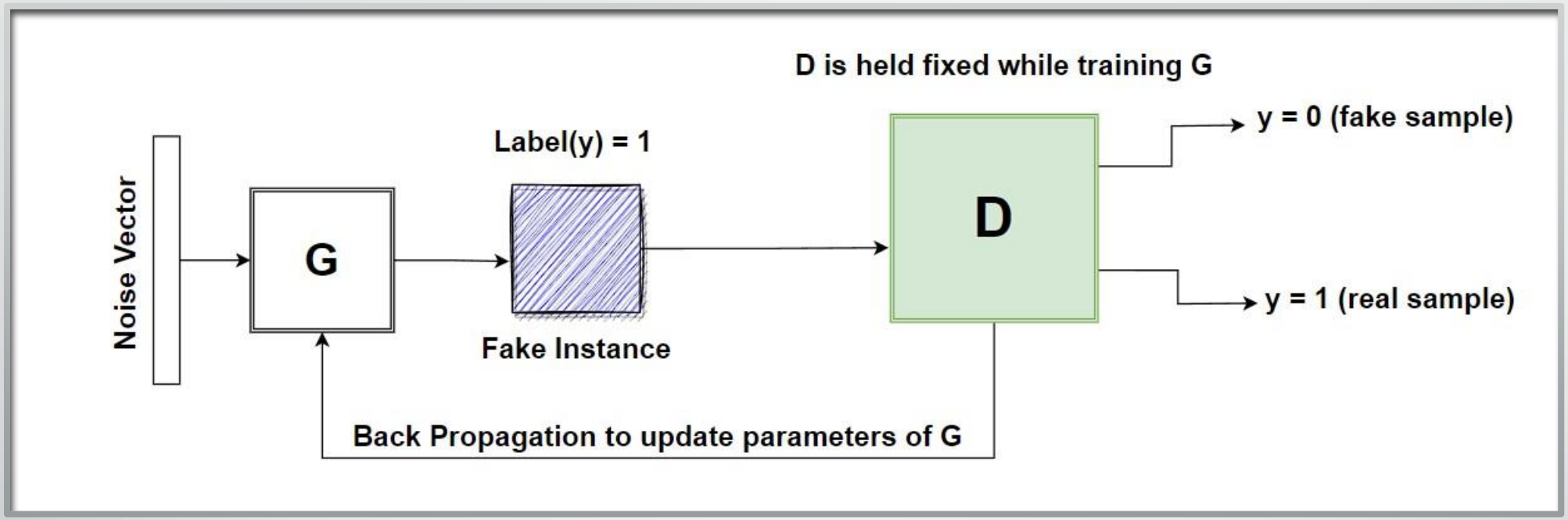
- **Training of GAN:**
  - Training of GAN consists of training of Discriminator and Generator.
  - Each of these networks(G and D) are held fixed while training the other.
  - While training Generator(G), you'll make Discriminator(D) constant, i.e. no weights and biases of D will update.
  - Training of Discriminator is relatively easier. We will thus, alternate between the D and G while training.

Training Process of GAN



a)Training of Discriminator is shown in the figure. All the artificial instances (blue) from generator given the noise vector are labelled as zero(0) while real instances(red) are labelled as one(1). Given these two instances to the model, it classifies the data whether it is fake or real.

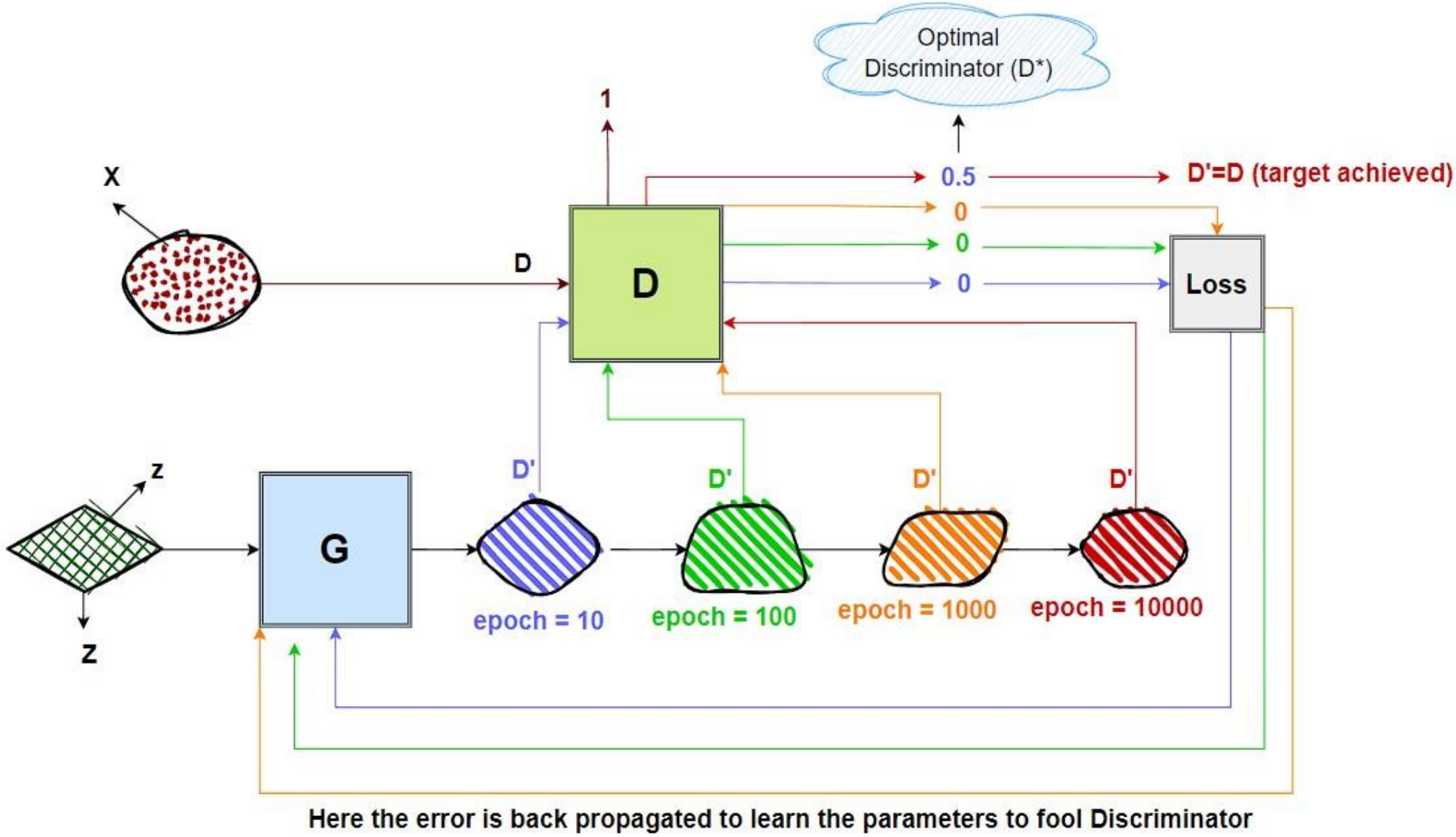
Training Process of GAN



b)Training of Generator is shown in the figure. The goal of G is to fool the discriminator, therefore, given the noise vector, G produces a fake sample which is assigned a label of 1(as real). If D recognizes the fake sample, perform backpropagation and update the parameters of G so that it becomes smarter and smarter to fool D which then classifies fake sample as real one.

Training Process of GAN

- **Training of GAN:**
  - The training of D is similar to simple classification problem in machine learning. You can stop training when D is trained enough to discriminate well.
  - The training of G, However, should stop when the target of G is achieved. In other words, if G produces output similar to the real instance and D outputs the value of 0.5 ideally, then it means G is now become intelligent to fool D.

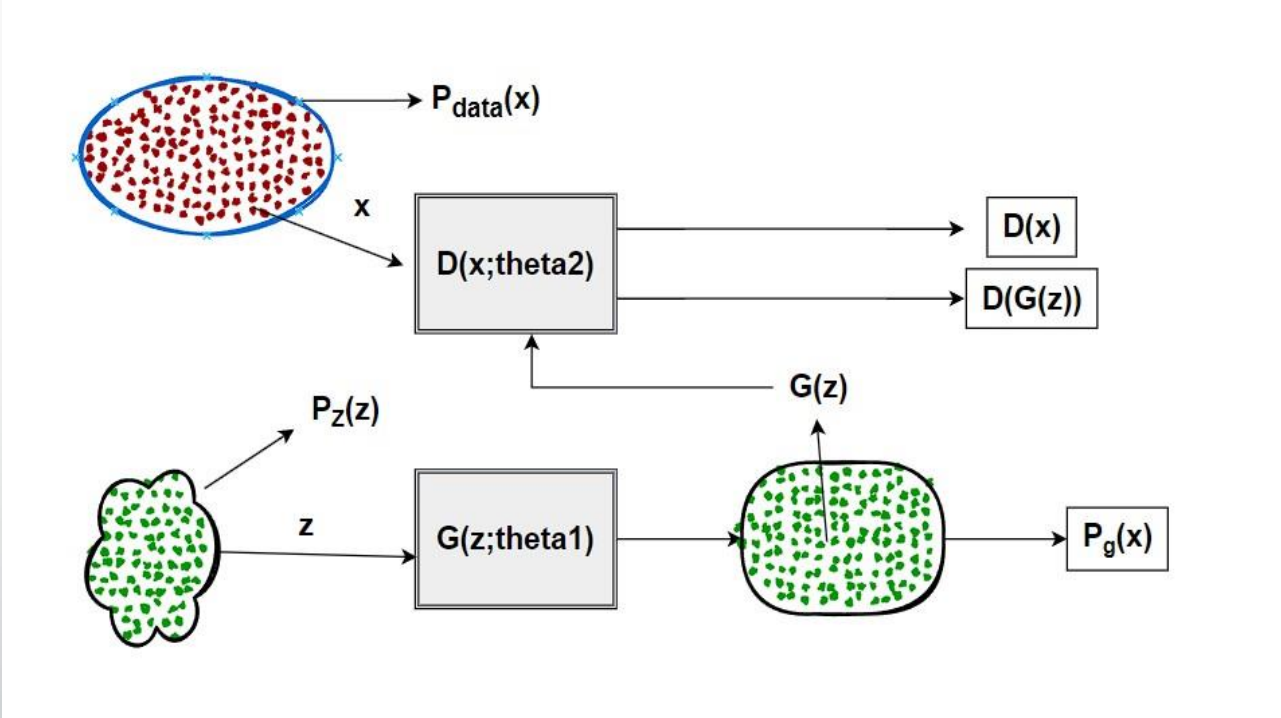


Overall Training and Back propagation of Generator over multiple iterations. The similarity between D and D' is calculated using L2 norm. (Note that datapoints X and the datapoints inside D' are different)

Loss Function of GAN

Basic conventions in understanding loss function

- **X=x** is the Instance of Random variable X of original data
- **$P_{data}(x)$** = Unknown Probability distribution that will be approximated by generator.
- **D(x,theta2)** = Discriminator Network
- **theta2** = Discriminator Parameters
- **D(x)** = transformed variable by D which indicates the probability distribution of original data x.
- **z** = random noise **$P_Z(z)$** = Probability distribution generated by random noise
- **G(z;theta1)** = Generator Network
- **theta1** = Generator Parameters
- **G(z)** = the samples from generated distribution.
- **$P_g(x)$** = Map the $P_z(z)$ to another distribution close to the $P_{data}(x)$
- **D(G(z))** = transformed variable by D which indicates the probability distribution of generator's distribution G(z).



Loss Function of GAN

Basic conventions in understanding loss function

- $\mathbf{P_{data}(x)}$ is the probability distribution sampled from the original dataset having datapoints **x.** However, $\mathbf{P_g(x)}$ is the probability distribution that is produced by the generator. Since, the objective of G is to generate a distribution close to the original one, i.e. $\mathbf{P_g(x) \sim P_{data}(x)}$ where x is G(z) points of new distribution in $P_g(x)$.
- The probability denoted as **D(G(z))** and **D(x)** ranges from [0,1].

Loss Function of GAN

Basic conventions in understanding loss function

- **$P_{data}(x)$** is the probability distribution sampled from the original dataset having datapoints **x**
- However, **$P_g(x)$** is the probability distribution that is produced by the generator. Since, the objective of G is to generate a distribution close to the original one, i.e. **$P_g(x) \sim P_{data}(x)$** where x is G(z) points of new distribution in $P_g(x)$.
- When G(z) from $P_g(x)$ is passed to D(x,theta), it produces a probability denoted as **D(G(z))** ranges from [0,1].

Loss Function of GAN

$$V(D(x), D(G(z) =$$
$$E_{x \sim pdata}(x)\{\log(D(x)\} + E_{z \sim pz}(z)\{\log(1 - D(G(z)))\}$$

$$\mathbf{V(D, G) = \min_G \max_D (V(D,G))}$$

Optimization

- ✓ Optimal maximum value of Discriminator is:

$$D_G^*(x) = \frac{p_{data(x)}}{p_{data(x)} + p_{g(x)}}$$

- ✓ Optimal maximum value of Generator is:

$$G_{D^*}^*(x) = -\log 4$$

**Proof is in the document.

Training & Back Propagation using SGD

- Training of GAN will occur alternatively. In the beginning we start training Discriminator only making Generator sit idle for some epochs. After that, we train Generator by making Discriminator sit idle for some epochs. Hence, this will train the GAN sequentially.
- We are using stochastic gradient descent algorithm to train GAN.

Training & Back Propagation using SGD

Steps:

- 1) Take the m number of images sampled from real dataset ($p_{data}(x)$) and m images sampled from random distribution ($p_z(z)$).
- 2) Train Discriminator and update the weights of discriminator by SGD as we ascend since we are concerning about the maxima of loss function.
- 3) Train generator and update the weights as we descend since we are concerning about minimizing loss.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Problems in GAN

1) Vanishing Gradient Problem:

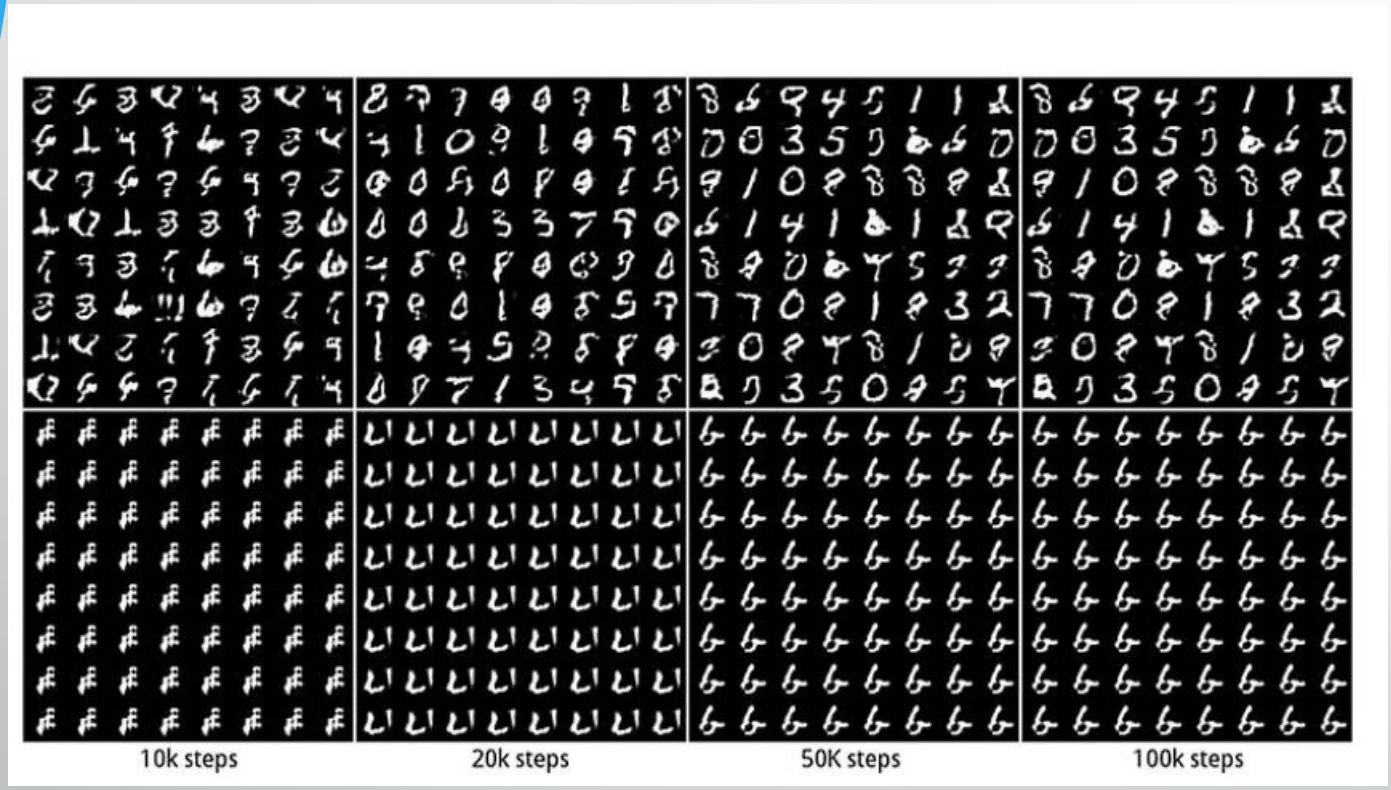
- In the start, the generator produces bad outcomes and discriminator performs well because it has been trained first. The gradients concerning weights and biases become close to zero. The generator doesn't learn to produce good fake samples. To avoid this situation, here's little modification in generator's loss function. Instead of minimizing log(1-D(G(z))), a max of log(D(G(z)) will avoid the above situation.

Problems in GAN

2) Mode Collapse:

- During training, the generator may collapse to the setting where it always generate same output. This is called Mode Collapse. This usually happens because the real data distribution is actually a complex multi-model distribution.

Problems in GAN



For example, in MNIST, there are 10 major **modes** from digit ‘0’ to digit ‘9’. The samples below are generated by two different GANs. The top row produces all 10 modes while the second row creates a single mode only (the digit “6”). This problem is called **mode collapse** when only a few modes of data are generated.

Problems in GAN

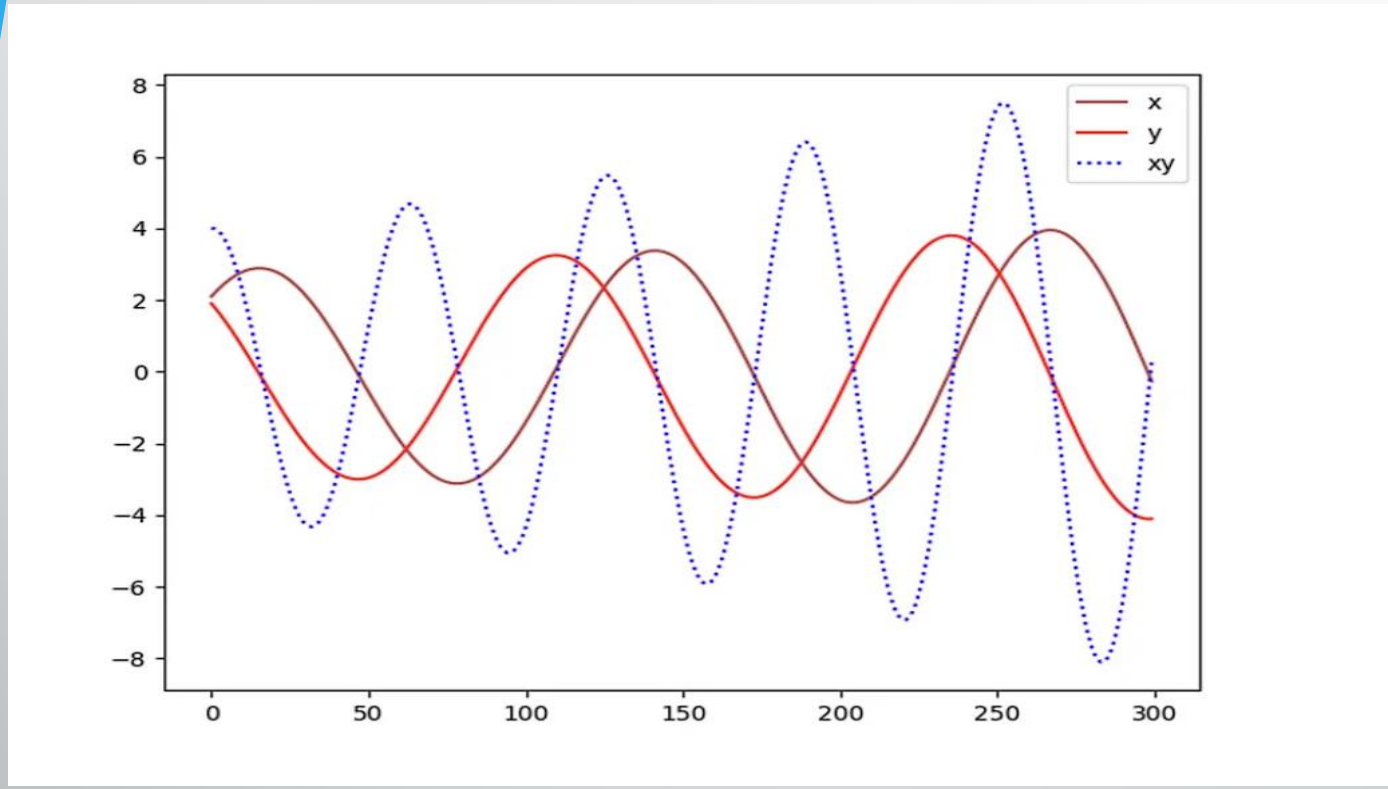
3) Difficult to achieve Nash Equilibrium:

- GAN consists of 2 networks whose objectives are opposite, i.e. just like zero-sum non-cooperative game where your opponent as a player wants to maximize its actions while your actions are to minimize them. Therefore, we want GAN model to converge i.e. both G and D reach a Nash Equilibrium.
- Nash Equilibrium is a point where “there exists no motivation for players to stray from their initial strategy alone”.
- For instance, Consider two players controlling the value of x and y. Player 1 wants to maximize the cost function and Player 2 want it to be minimized.

$$\text{Cost Function} = \min_2 \max_1 (xy)$$

- Nash Equilibrium is where x=y=0, the point where change of state of player doesn’t improve the result.

Problems in GAN



When we plot ***x***, ***y***, and ***xy*** against the training iterations, we realize our solution does not converge. This example is an excellent showcase that some cost functions will not converge with gradient descent, in particular for a non-convex game.

Create GAN from scratch (MNIST dataset)

```
def mnist_data():
    compose = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((.5, ), (.5, ))
        ])
    out_dir = './dataset'
    return datasets.MNIST(root=out_dir, train=True, transform=compose, download=True)# Load data
data = mnist_data()# Create loader with data, so that we can iterate over it
data_loader = torch.utils.data.DataLoader(data, batch_size=100, shuffle=True)
# Num batches
num_batches = len(data_loader)
```

1) Download dataset: Using torch.utils.datasets, import the MNIST data set, convert to the Tensor while normalizing and create the data loader object for iterating.

```python
class Discriminator(torch.nn.Module):
    def __init__(self):
        super(Discriminator,self).__init__()
        self.hidden0 = nn.Sequential(
            nn.Linear(784,1024),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3)
        )
        self.hidden1 = nn.Sequential(
            nn.Linear(1024,512),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3)
        )
        self.hidden2 = nn.Sequential(
            nn.Linear(512,256),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3)
        )
        self.out = nn.Sequential(
            nn.Linear(256,1),
            nn.Sigmoid()
        )

    def forward(self,x):
        x = self.hidden0(x)
        x = self.hidden1(x)
        x = self.hidden2(x)
        x = self.out(x)
        return x
discriminator  = Discriminator()
```

Create GAN from scratch (MNIST dataset)

2) Define Discriminator Network:
Define the Discriminator architecture. While defining, the input and outputs should be considered. For instance, D should receive an image of 28x28 with one channel, thus 784 dimension vector is provided to the first layer. However, the output layer should have only one neuron representing Fake(0) or Real(1) respectively.

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator,self).__init__()
        self.hidden0 = nn.Sequential(
            nn.Linear(100,256),
            nn.LeakyReLU(0.2)
        )
        self.hidden1 = nn.Sequential(
            nn.Linear(256,512),
            nn.LeakyReLU(0.2)
        )
        self.hidden2 = nn.Sequential(
            nn.Linear(512,1024),
            nn.LeakyReLU(0.2)
        )
        self.out = nn.Sequential(
            nn.Linear(1024,784),
            nn.Tanh()
        )

    def forward(self,x):
        x = self.hidden0(x)
        x = self.hidden1(x)
        x = self.hidden2(x)
        x = self.out(x)
        return x

generator = Generator()
```

Create GAN from scratch (MNIST dataset)

3) Define Generator Network:
Likewise Discriminator, define generator architecture. Considering the input and output to the generator network, in MNIST dataset, the input is the random noise vector with dimension 100. Then the vector is upscales in all the successor layers to produce the output with the same dimension as that of real data instance, i.e., 1x28x28 or 784.

Create GAN from scratch (MNIST dataset)

```
def images_to_vectors(images):
    return images.view(-1,784)

def vectors_to_images(vectors):
    return vectors.view(-1,1,28,28)

def ones_target(size):
    return Variable(torch.ones(size,1))

def zeros_target(size):
    return Variable(torch.zeros(size,1))

def noise(size):
    n = Variable(torch.randn(size,100))
    return n
```

5) Define the Helper Function:
After defining the networks, define some of the helper functions that help you in converting your input image to the 1-D vector or 1-D vector to image. Similarly, define a function to declare noise and labels for training.

Create GAN from scratch (MNIST dataset)

```
d_optim = optim.Adam(discriminator.parameters(),lr = 0.0002 )
g_optim = optim.Adam(generator.parameters(),lr = 0.0002 )
loss = nn.BCELoss()
```

6) Define Optimizer and Loss Function:

The last step before training the networks is defining the optimizer and loss function. The optimizer use for MNIST is Adam for both networks with learning rate of 0.0002. Also, the loss function is BinaryCrossEntropyLoss.

```
def train_discriminator(optimizer,real_data,fake_data):
    N = real_data.size(0)
    optimizer.zero_grad()

    prediction_real = discriminator(real_data)
    error_real = loss(prediction_real,ones_target(N))
    error_real.backward()

    prediction_fake = discriminator(fake_data)
    error_fake = loss(prediction_fake,zeros_target(N))
    error_fake.backward()

    optimizer.step()
    return error_real + error_fake, prediction_real, prediction_fake
```

```
def train_generator(optimizer,fake_data):
    N = fake_data.size(0)
    optimizer.zero_grad()
    prediction = discriminator(fake_data)
    error = loss(prediction,ones_target(N))
    error.backward()
    optimizer.step()
    return error
```

Create GAN from scratch (MNIST dataset)

7) Define function that trains the G and D networks:

The next step is to define the training process for both networks. It is to be noted that discriminator is given the real instance first and then the fake instance produced by the generator. After that loss is computed and back propagation is performed.

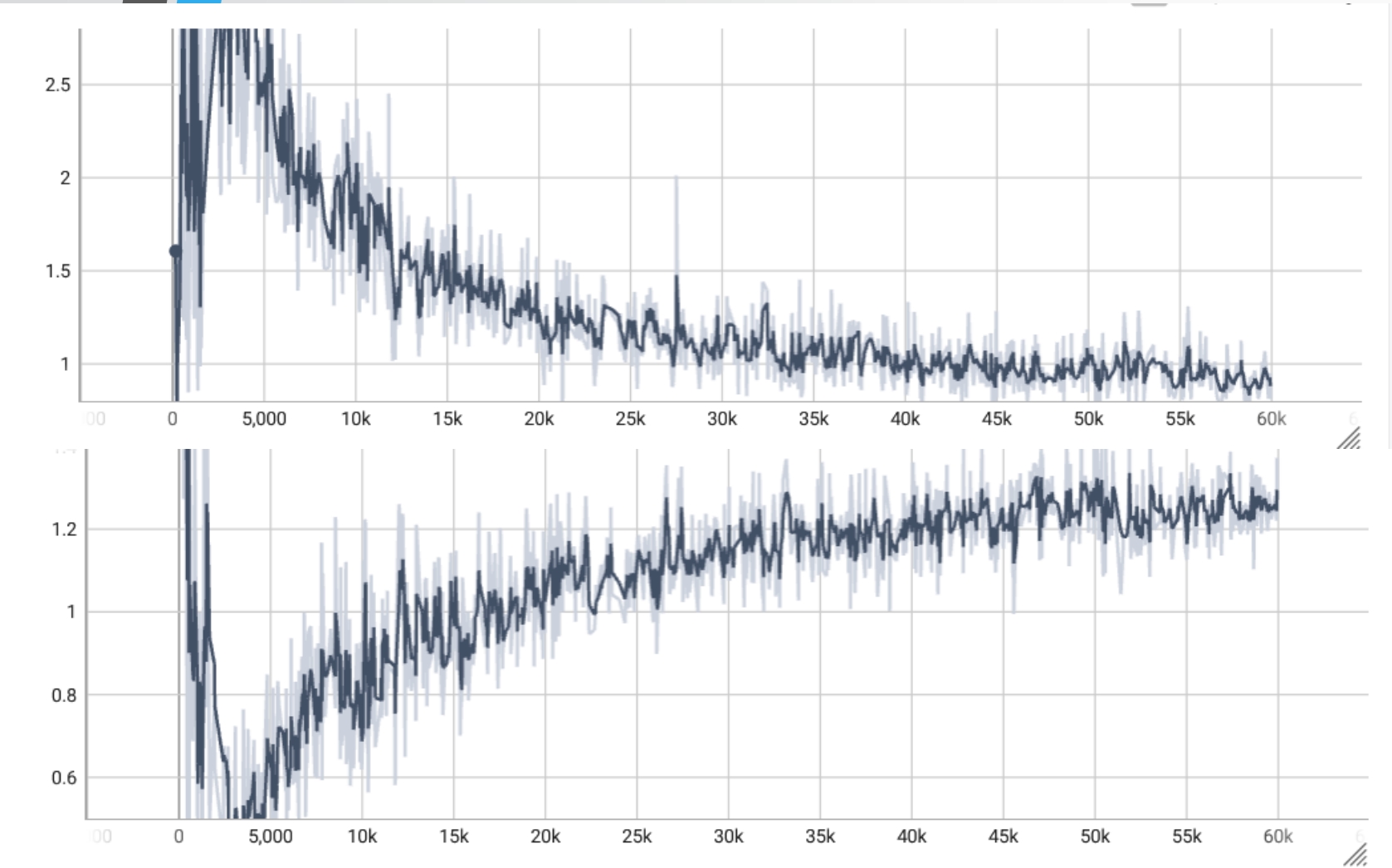
Create GAN from scratch (MNIST dataset)

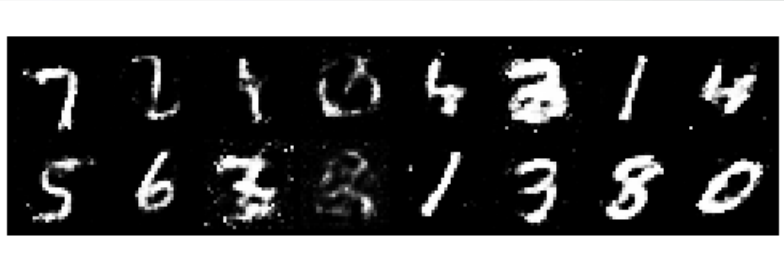
```
num_test_samples,num_epochs = 16,100
test_noise = noise(num_test_samples)
logger = Logger(model_name='GAN', data_name='MNIST')

for epoch in range(num_epochs):
    for n_batch, (real_batch,_) in enumerate(data_loader):
        N = real_batch.size(0)

        # train the discriminator
        real_data = Variable(images_to_vectors(real_batch))

        fake_data = generator(noise(N)).detach()

        d_error,d_pred_real,d_pred_fake = train_discriminator(d_optim,real_data,fake_data)

        # traing the generator
        fake_data = generator(noise(N))
        g_error = train_generator(g_optim,fake_data)

        logger.log(d_error, g_error, epoch, n_batch, num_batches)

        if (n_batch) % 300 == 0 and n_batch != 0 and epoch % 20 == 0:
            test_images = vectors_to_images(generator(test_noise))
            test_images = test_images.data

            logger.log_images(
                test_images, num_test_samples, epoch,n_batch,num_batches)

            logger.display_status(
                epoch,num_epochs,n_batch,num_batches,d_error,g_error,d_pred_real,d_pred_fake)
```

8) Overall Training of GAN:

The last step is to trigger the training of both in adversarial manner. The Discriminator is trained over the fake data produced by the G and real data. After this, keeping D fixed, train the G network. The *logger is a class created to help programmer to save the model stats, it's error and images produced by D and G.* A batch of test images are created evaluate the generator performance.

Create GAN from scratch (MNIST dataset)





Results: The top graph is the loss of the Generator while bottom graph represents the loss of Discriminator. The images produced by the generated after 100 epochs is also attached.

Loss Function of GAN:

Given y' and y, where y' is related to the generated image and y belongs to the original image of the dataset, binary cross entropy is given as:

$$\Rightarrow L(y', y) = y \log y' + (1 - y) \log(1 - y')$$

The **input coming from the $p_{data}(x)$** has label y=1. However, the generated or reconstructed image has y' equals to the D(x) which is the predicted probability distribution.

$$\Rightarrow L(y', y) = L(D(x), 1) = 1 \log(D(x)) + (1 - 1) \log(1 - D(x))$$

$$= \log(D(x)) \quad \text{---------- (A)}$$

On the other hand, **the input coming from generator**, i.e. the generated image has label y' and is equals to D(G(z)). The actual image has label y which corresponds to zero value since the objective of generator is to fool the discriminator:

$$\Rightarrow L(y', y) = L(D(G(z)), 0) = 0 + (1 - 0) \log(1 - D(G(z)))$$

$$= \log(1 - D(G(z))) \quad \text{---------- (B)}$$

The **objective of discriminator** is to classify the fake and real images correctly. It is achieved by maximizing (A) when y=1 and (B) when y=0 as represented in the Figure 1. In the left image, the log plot corresponds to the log(D(x)). We know that if data is coming from $p_{data}(x)$, then D(x) will be 1. This is the maximum value it can achieve because we are dealing with the probabilities. $D(x) = 1$ represents that discriminator is classifying correctly. The right plot corresponds to the log(1-D(G(z))).

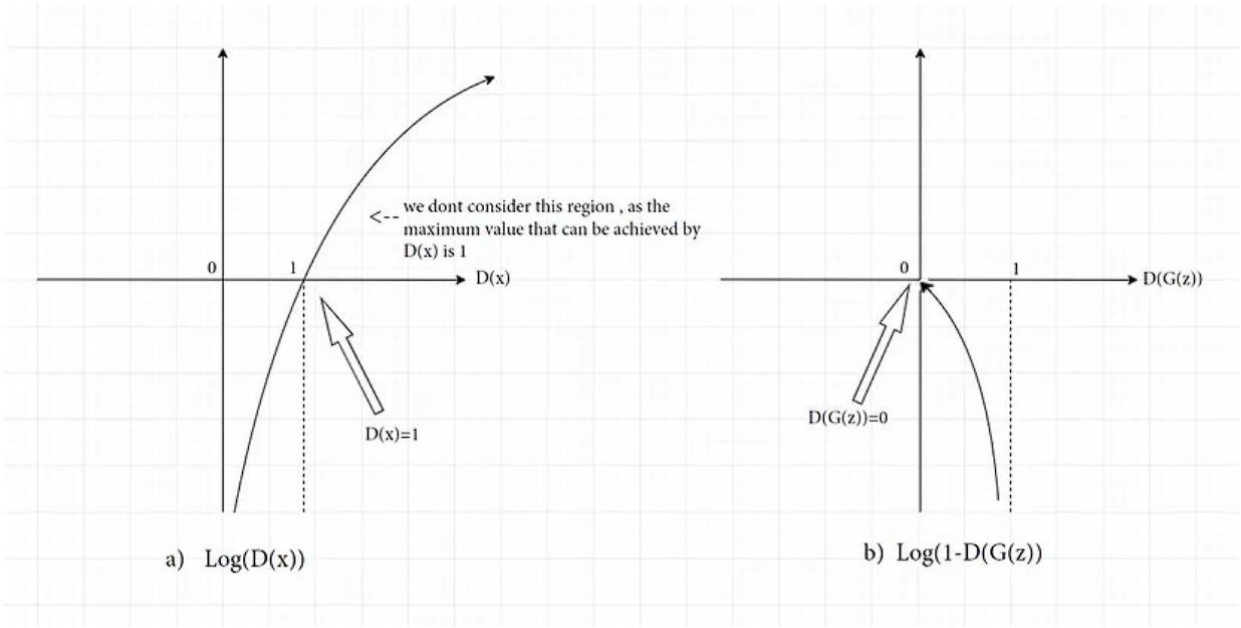


Figure 1

The input coming from G(z) passed to the D(z), it would be classified as zero. The maximum value in this plot is $D(G(z)) = 0$. Therefore, maximizing the log(1-D(G(x))) makes the Discriminator to classify correctly.

$$\Rightarrow L(y', y) = max[log(D(x) + log(1 - D(G(z)))] \text{ ---------- (C)}$$

The **objective of generator** is to fool the discriminator. That means, we want the discriminator to classify the input coming from Generator as real one. Therefore, we force the log(1-D(G(z))) term to be 1 instead of 0. Hence, we will minimize the graph where D(G(z)) produces a value of 1 to make 1-D(G(z)) to be 0. This is illustrated in the above graph.

$$\Rightarrow L(y', y) = min[log(D(x) + log(1 - D(G(z)))] \text{ --------- (D)}$$

Since, log(D(x)) has no role in generator objective, therefore, Combining both (C) and (D) together, this will make:

$$L(D(x), D(G(z)) = min_G \, max_{\,D} \{ log(D(x) + log(1 - D(G(z))) \}$$

The above loss equation is only for one sample that GAN will work on. For all sample, we will take expectation. Thus overall loss function of GAN having generator network (G) and discriminator network (D) is:

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

$$V(D, G) = minG\ maxD\ (V(D, G))$$

Optimal Value of Discriminator

To find the best discriminator among the all discriminators, the optimal value of D while making G fixed should be:

- ⇨ $D^*_G = \text{argmax}_D V(D,G)$ where argmax defines which value of D will maximize V(D,G).

According to the definition of E(x):

- ⇨ $E_{p(x)}(x) = \int x\ p_x(x)\ dx$ ---------- (1)
- ⇨ V(G,D) =

$$\int_x p_{data}\ (x) \log\big(D(x)\big)\, dx\ + \int_z p_z\,(z) \log(1 - D(G(z)))\, dz$$

Note: Given the random variable X and it's probability $p_x(x)$, we can calculate probability density function over other random variable Y in terms of X. Given Y= G(x), mathematically this concept is written as:

$$\Rightarrow P_Y(y) = p_x\,(G^{-1}(y))\ \frac{d}{dy}\ (G^{-1}(y)))$$

Utilizing above change of variable concept by replacing all the y's to the x's and all x's to z's. Thus,

$$\Rightarrow p_g(x)\ = p_z\,(G^{-1}(x))\ \frac{d}{dx}\ (G^{-1}(x)))\ \text{----------(2)}$$

Here, the x is different from the x sampled from real distribution, i.e. $x = G(z)$ and $z = G^{-1}(x)$ where G is invertible. Now, we can write V(D,G) as:

$$\int_x p_{data}\ (x) \log\big(D(x)\big)\, dx + \int_z p_z\,(G^{-1}(x)) \log\big(1 - D(x)\big)\, dG^{-1}(x)$$

Divide and multiply with dx:

$$= \int_x p_{data}\ (x) \log\big(D(x)\big)\, dx + \int_z p_z\,(G^{-1}(x)) \frac{dG^{-1}(x)}{dx} \log\Big(1\ G\big(D(x)\big)\Big)\, dx$$

Putting value of (2) in above,

$$V(D,E) = \int_x p_{data}(x) \log\big(D(x)\big)\, dx + \int_x p_g(x) \log\Big(1 - D\big(G(x)\big)\Big)\, dx \quad \textbf{---Eq.1}$$

Now, maximizing the above relation to get the optimal value of D. Making derivative of V(D, E) equals to zero as:

$$\frac{d}{d(D(x))} \int_x p_{data}(x) \log\big(D(x)\big)\, dx + \int_x p_g(x) \log\Big(1\, G\big(D(x)\big)\Big)\, dx = 0$$

$$\frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0$$

Taking LCM and multiplying terms, The final expression is :

$$D(x)\big[p_{data}(x) + p_g(x)\big] = p_{data}(x)$$

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

This is the optimal value of Discriminator. Let's check is it maximum or minimum. Take derivative again:

$$-\frac{p_{data}(x)}{\big(D(x)\big)^2} - \frac{p_g(x)}{\big(1 - D(x)\big)^2} < 0$$

Thus, the optimal value above is actually the maximum optimal value of the Discriminator given Generator.

Optimal Value of Generator

The role of generator is opposite to the discriminator. Finding the optimal value for generator means finding the minimum of the function given the optimal discriminator.

$G_{D^*}^* = argmin_G\, V(D*, E)$ where argmin defines which value of G will minimize V(D*,G). The loss function is:

$$\Rightarrow\ V(D^*, G) = argmin\{\int_x p_{data}\ (x) \log\big(D * (x)\big)\, dx$$

$$+ \int_x p_g\,(\mathrm{x}) \log\big(1 - D^*(x)\big)\, dx\}$$

Putting the value of the D*:

$$\Rightarrow argmin_G \left\{ \begin{array}{l} \int_x p_{data}\ (x) \log\left(\frac{p_{data}\,(x)}{p_{data}\,(x) + p_g(x)}\right) dx \\ + \int_x p_g\,(\mathrm{x}) \log\left(1 - \frac{p_{data}\,(x)}{p_{data}\,(x) + p_g(x)}\right) dx \end{array} \right\}$$

$$\Rightarrow argmin_G \left\{ \begin{array}{l} \int_x p_{data}\ (x) \log\left(\frac{p_{data}\,(x)}{p_{data}\,(x) + p_g(x)}\right) dx \\ + \int_x p_g\,(\mathrm{x}) \log\left(\frac{p_g\,(x)}{p_{data}\,(x) + p_g(x)}\right) dx \end{array} \right\}$$

Adding and subtracting $\log 2\ p_{data}(x)$ and $\log 2\ p_g(x)$:

$$\Rightarrow argmin_G \left\{ \begin{array}{l} \int_x [(log2 - log2) p_{data}(x) + p_{data}\ (x) \log\left(\frac{p_{data}\,(x)}{p_{data}\,(x) + p_g(x)}\right) \\ + (log2 - log2) p_g(x) + \int_x p_g\,(\mathrm{x}) \log\left(\frac{p_g\,(x)}{p_{data}\,(x) + p_g(x)}\right)] dx \end{array} \right\}$$

Rearranging terms:

$$argmin_G \left\{ \int_x [-log2\left(p_{data}(x) + p_g(x)\right) + p_{data}\ (x)[log2 + \log\left(\frac{p_{data}\,(x)}{p_{data}\,(x) + \ p_g(x)}\right)] + p_g(x)[log2 + \log\left(\frac{p_{data}\,(x)}{p_{data}\,(x) + \ p_g(x)}\right)]]dx \right\}$$

Apply properties of logarithm:

$$argmin_G \left\{ \begin{array}{c} -log2 \int_x \left(p_{data}(x) + p_g(x)\right) dx + \\ \int_x p_{data}(x)[\log\left(\frac{p_{data}\,(x)}{p_{data}\,(x) + \ p_g(x)}\right) * 2]dx + \\ \int_x p_g(x)[\log\left(\frac{p_g\,(x)}{p_{data}\,(x) + \ p_g(x)}\right) * 2]dx \end{array} \right\}$$

Apply $\int p_x dx = 1$

$$argmin_G \left\{ \begin{array}{c} -log2(1+1) + \\ \int_x p_{data}(x)[\log\left(\frac{p_{data}\,(x)}{p_{data}\,(x) + \ p_g(x)}\right)/2]dx + \\ \int_x p_g(x)[\log\left(\frac{p_g\,(x)}{p_{data}\,(x) + \ p_g(x)}\right)/2]dx \end{array} \right\}$$

Apply KL divergence here:

$$\int_x {p\_(x)}/{q(x) + p\_(x)/2)} = KL\left(p\_(x) \,||\, {p\_(x) + q(x)}/{2}\right.$$

The above relation now,

⇨ $argmin_G\{ -\log(4) + KL\,[p_{data}(x) \,||\, p_{data}(x) + p\_g/2$
$+\mathrm{KL}\,\left[p_g(x)\right|| \, p_{data}(x) + p\_g/2\ ]$

Apply Jensen Shamon Divergence:

JSD = ½ {KL(P || M) +KL (Q || M)) where M = P+Q/2

The equation now,

⇨ $argmin_G\{ -\log(4) + 2\,JSD\ (p_{data}(x) \,||\, p_g(x)\}$

Note: JSD becomes zero when p_g (x) = p_{data} (x). This is the objective of your generator. Make both these distributions as close as possible to fool discriminator. Thus it force JSD to be zero to find the minima.

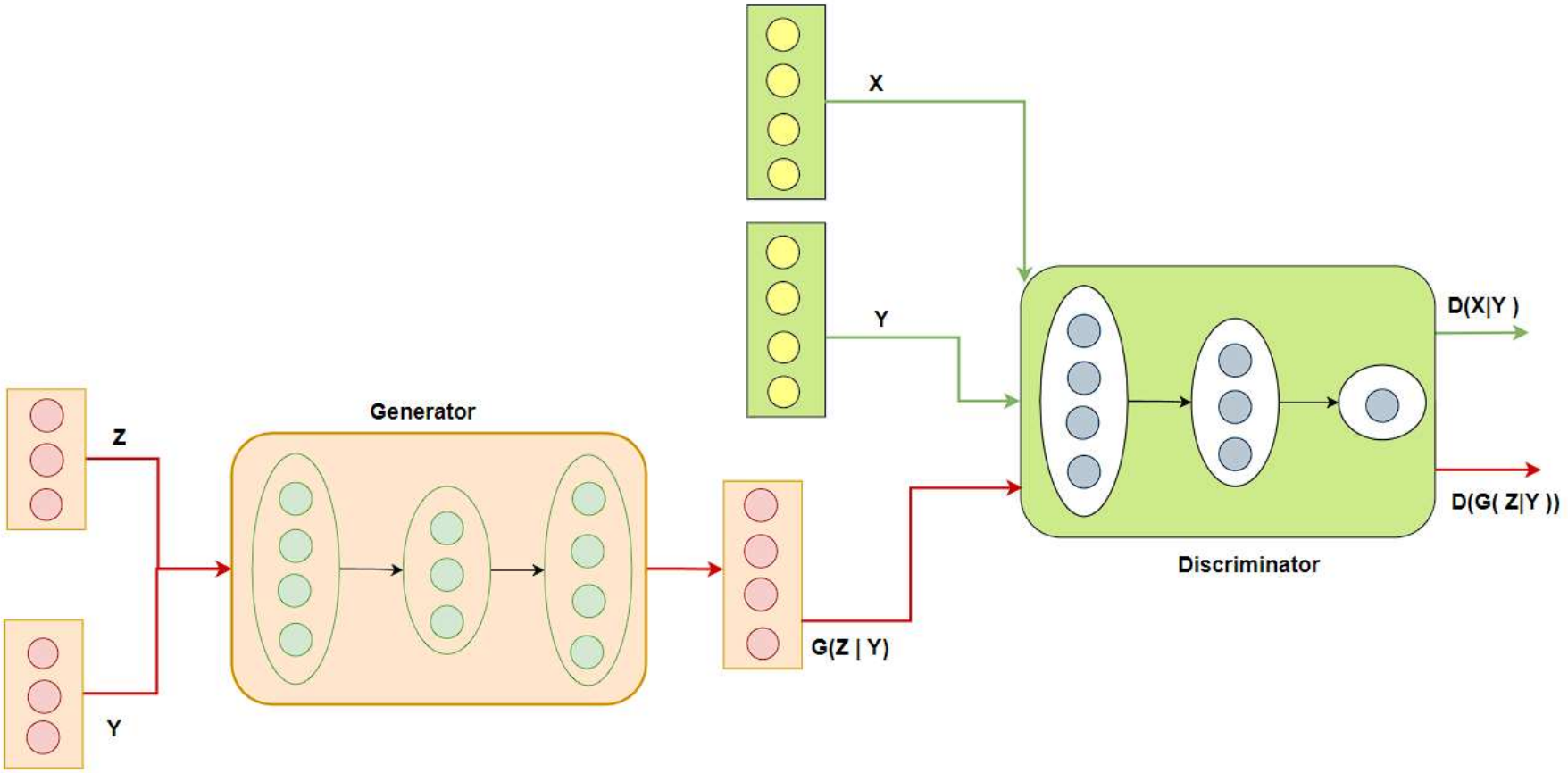
The optimal generator is now:

$$G^*_{D*} = -log4$$

Conditional GAN (C-GAN)

- In the GAN creation with MNIST dataset, the output produced consists of 10 different categories, i.e. Mixed output. This indicated there's no control over the output. This is the drawback of the simple GAN. If you want to control the output of the GAN, i.e. you want to display all shoes in case of MNIST Fashion Dataset, **give the labels to the GAN**. This version of GAN is called Condition GAN or C-GAN as explained in the next slide.

Conditional GAN (C-GAN)



Y

Conditional GAN (C-GAN)

- The loss function is now modified as :

$$E_{x \sim pdata}(x)\{\log(\boldsymbol{D}(\boldsymbol{x} \mid \boldsymbol{y})\} + E_{z \sim pz}(z)\{\log(1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}) \mid \boldsymbol{y}))\}$$

- Apply the label information into loss function, thus produced the probabilities conditioned on that label.
- The rest of the process is same as GAN.

Info GAN

- Info-GAN is actually use for information maximization of learning the disentangled representations of the images.
- It performs unsupervised learning, i.e., labels are not provided while training. It derives the labels from the data while training which is in contrast with the C-GAN.
- This type of GAN is based on **Information theory** which is a branch of applied mathematics that quantifies how much information is present in a signal in context of communication.

Info GAN

Information Theory:

- In context of machine learning, it is used to quantify the similarity between the probability distributions
- **Main Idea: Unlikely events have more information as compared to likely events .**
- For instance, “temperature is 40 degrees in summer” is not as much informative as “temperature is 5 degrees in summer” is informative.

Info GAN

Concepts in Information Theory:

1) Self Information of an event X=x is:

$$I(X = x) = -log_2 P(X = x)$$

2) Shanon Entropy: We can quantify the amount of uncertainty in entire PDF using shanon entropy:

$$H(X) = E_{x \sim P(X)}[I(x)] = -E_{x \sim P(X)}[log_2 P(X = x)]$$

3) Joint Entropy: if x and y are the discrete random variables, then,

$$H(x, y) = E_{x,y}\big[-\log\big(P(x, y)\big)\big]$$

$$= -\sum_{x,y} P(x, y) \log\big(P(x, y)\big)$$

Info GAN

Concepts in Information Theory:

4) Conditional Entropy: the conditional entropy of X given random variable Y is:

$$H(X|Y) = E_Y\big(H(X|Y)\big)$$

$$= -\sum_{y\sim P_Y(y)} p(y) \sum_{x\sim P_X(x)} p(x|y) \log\big(P(x|y)\big)$$

By simplifying the above relation,

$$\boldsymbol{H(X|Y) = H(X,Y) - H(Y)}$$

Note: If H(X|Y)=0, iff value of X is completely determined by Y

Note: If H(X|Y)=H(X), iff X and Y are independent Random variables

Info GAN

Concepts in Information Theory:

5)Mutual Information: It is the amount of information that can be obtained related to one random variable by observing the other. In other words, it is a measure of how two events X and Y are corelated.

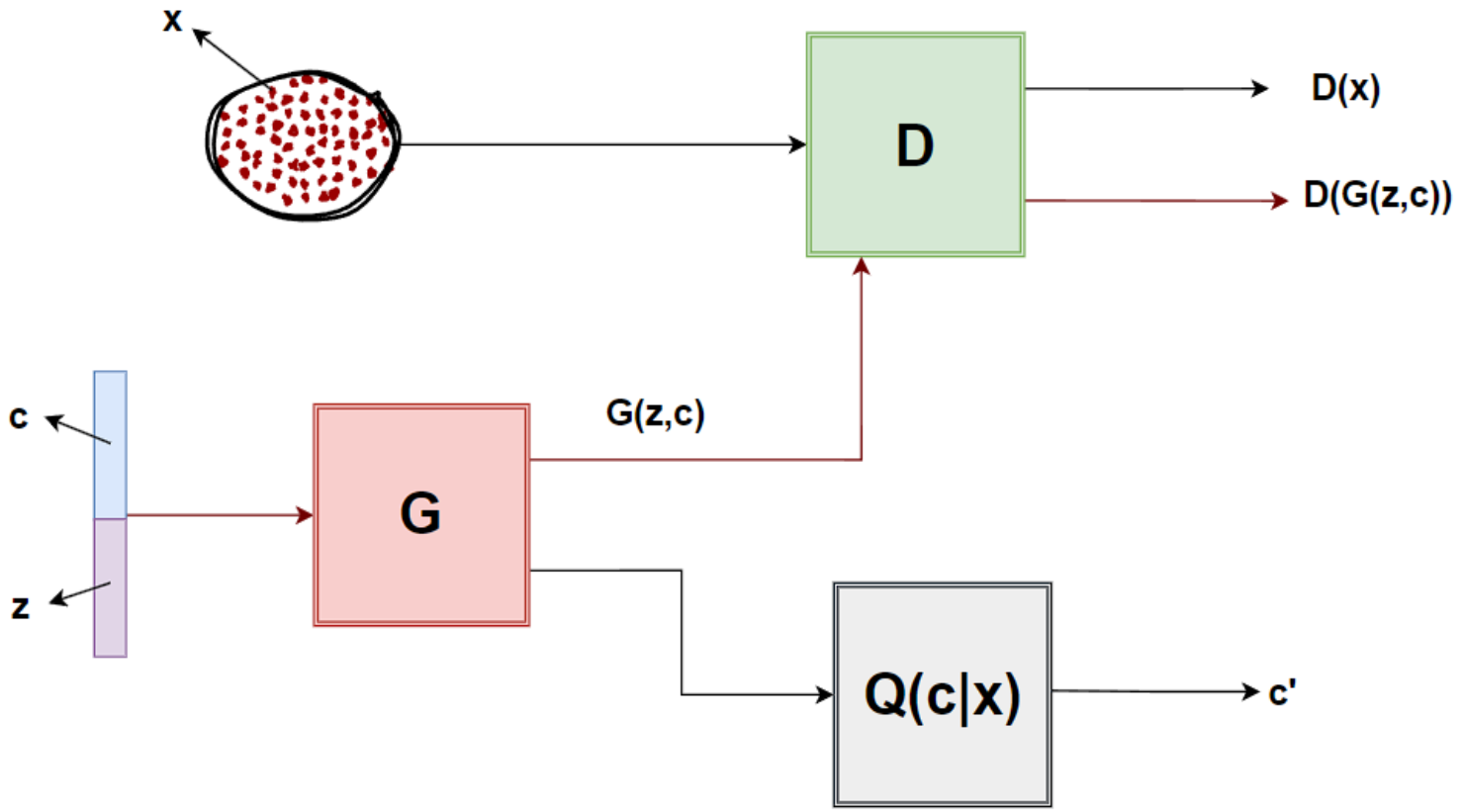
$$I(X;Y) = \sum_{x,y} P(x,y) \log\left(\frac{P(x,y)}{P(x)P(y)}\right)$$

After simplification, above expression is:

$$\boldsymbol{I(X;Y) = H(X) - H(X|Y)}$$
$$\boldsymbol{I(X;Y) = H(Y) - H(Y|X)}$$

Info GAN

Architecture of Info-GAN



Info GAN

Architecture of Info-GAN

The architecture of the info-Gan is similar to the GAN except the following modifications:

- The input noise vector to the Generator is now comprises of a standard incompressible latent vector **z** and a latent variable **c** to capture the salient semantic features of the real data samples. Thus, the output produced by generator is now G(z,c) since input vector is now the concatenation of two random variables z and c.
- It involves another neural network Q(c|x) which takes the generated image and provides an estimate of c, represented as c'.

Info GAN

Loss function

- The loss function of Info-GAN is almost same as the CGAN except the regularization term:

$$\min_{D} \max_{G} V_I(G, D) = V_G(G, D) - \lambda I\left(c, D\left(G(z, c)\right)\right)$$

Where,

$$V_G(G, D) = E_{x \sim p_{data(x)}}\{\log(D(x)\} + E_{z \sim p_z(z), c \sim p(c)} \log(1 - D(G(z, c)))$$

and the regularization parameter $\lambda < 1$.

Info GAN

Optimization of Loss function

- The optimization involves the estimation of optimal values for generator and discriminator network. Thus,
  - The optimal value for the Discriminator involves the first term only since the second term doesn't involve discriminator. Therefore, the optimal value is $\boldsymbol{D^* = \max_D V_I(G, D) = V_G(G, D)}$
  - The optimal value for the Generator involves the regularization term too. Thus, the optimal value is:

$$\boldsymbol{G^* = \min_G V_I(G, D) = E_{z\sim p_z(z), c\sim p(c)} log(1 - D\big(G(z, c)\big)} \text{ - } \boldsymbol{\lambda I\left(c, D\big(G(z, c)\big)\right)}$$

  - If you are minimizing the objective function to get G^* , then you need to maximize the mutual information because if you are min(A-B), one way is to maximize B.

C-GAN vs Info-GAN

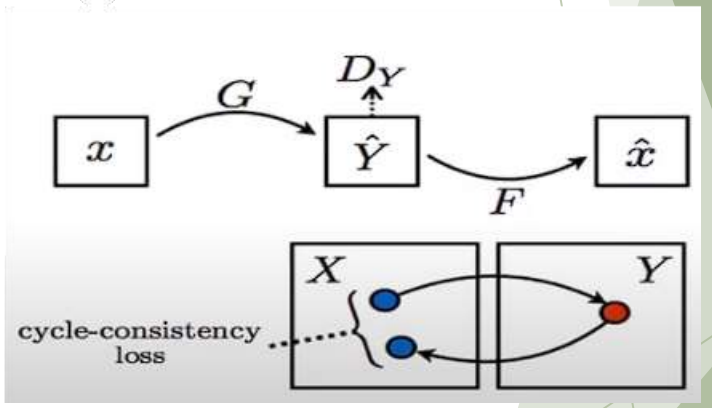
- C-GAN implies supervised method that uses both data and it's classes. It imposes a condition by adding extra information of class label to control the data generation. Moreover, it uses latent vector z which is highly entangled.
- Info-GAN uses unsupervised method. No extra information like class label is provided to it for data generation. Also, it uses disentangled feature representation. Moreover, Info-GAN also have more freedom to capture certain features of data.

Cycle GAN

- **Image to Image Translation**: conversion of one image to other keeping the attributes of first image. There exists two approaches:
  - 1) **Paired Approach(** You have two datasets X and Y and to transform one image from X to other image of Y, it is necessary to have the paired representation of the particular image in both dataset or domains.**) Example: Pix2Pix Architecture**
  - **2) Unpaired Approach(** X and Y may belong to any other domain, say X consists of natural images while Y consists of painted images. The transformation from X to Y or Y to X can be done.**) Example: Cycle GAN, Disco GAN, Dual GAN**
- **Cycle GAN will be discussed in detail.**

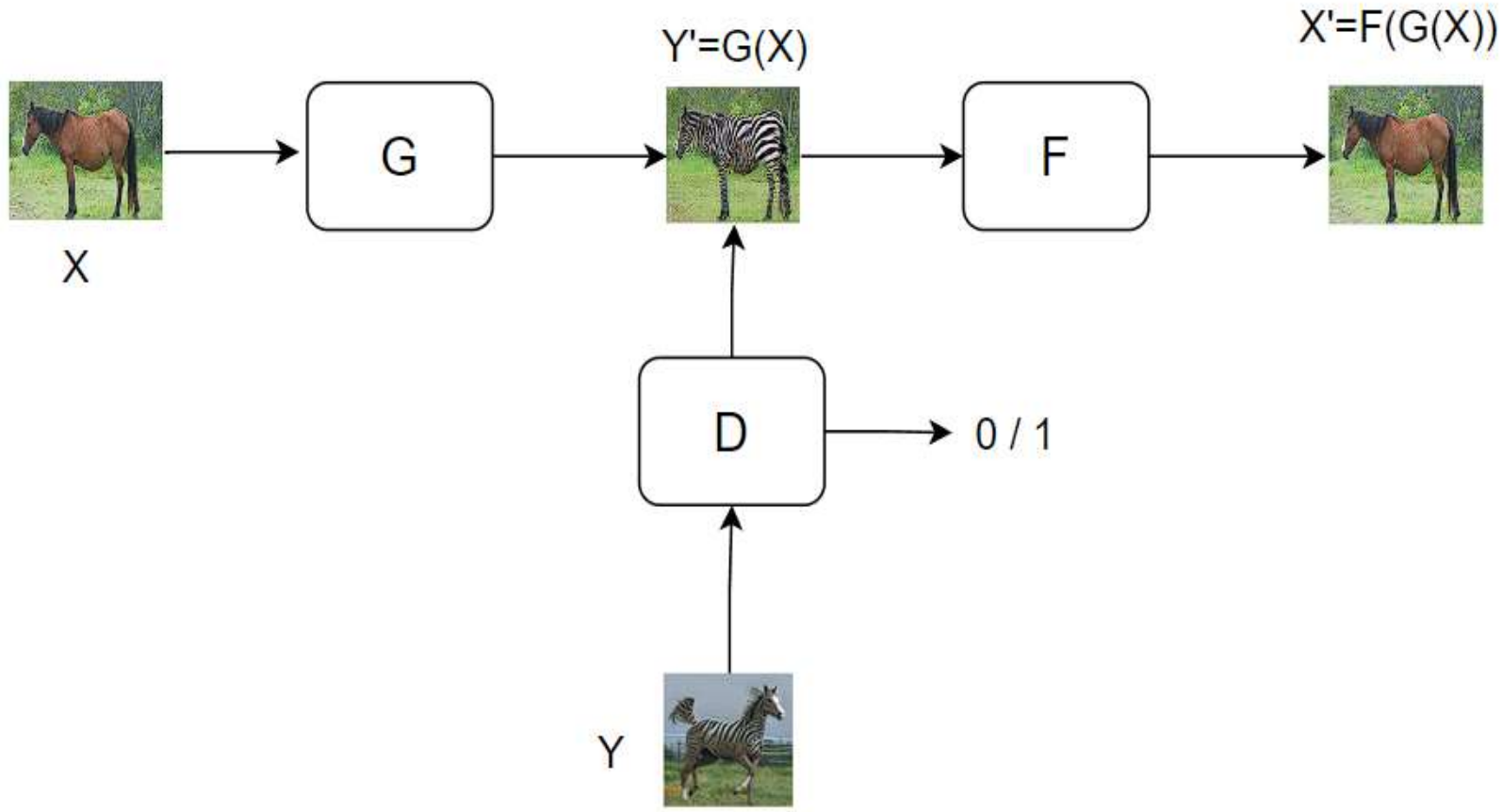
Cycle GAN Architecture

- **1) Converting X to Y and reconstruct X':** Say X contains the data from the distribution having images of Horses and Y belongs to the distribution with images of Zebra.
  - X image is given to the Generator Network G which converts it into Y' (zebra) retaining the properties of X. Y' is now given to D_Y which is assumed to trained on the images of Y. The target of D is to discriminate between the Y' and Y. Y' is then given back to the generator F which reconstructs the X as X'. The cycle-consistency loss actually determines how far these X and X' are. Pictorially, it is represented in the next slide.



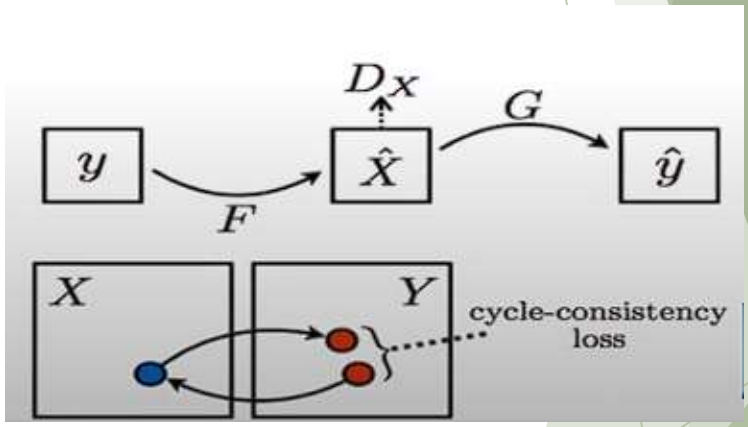
Cycle GAN Architecture

- **1) Converting X to Y and reconstruct X':**



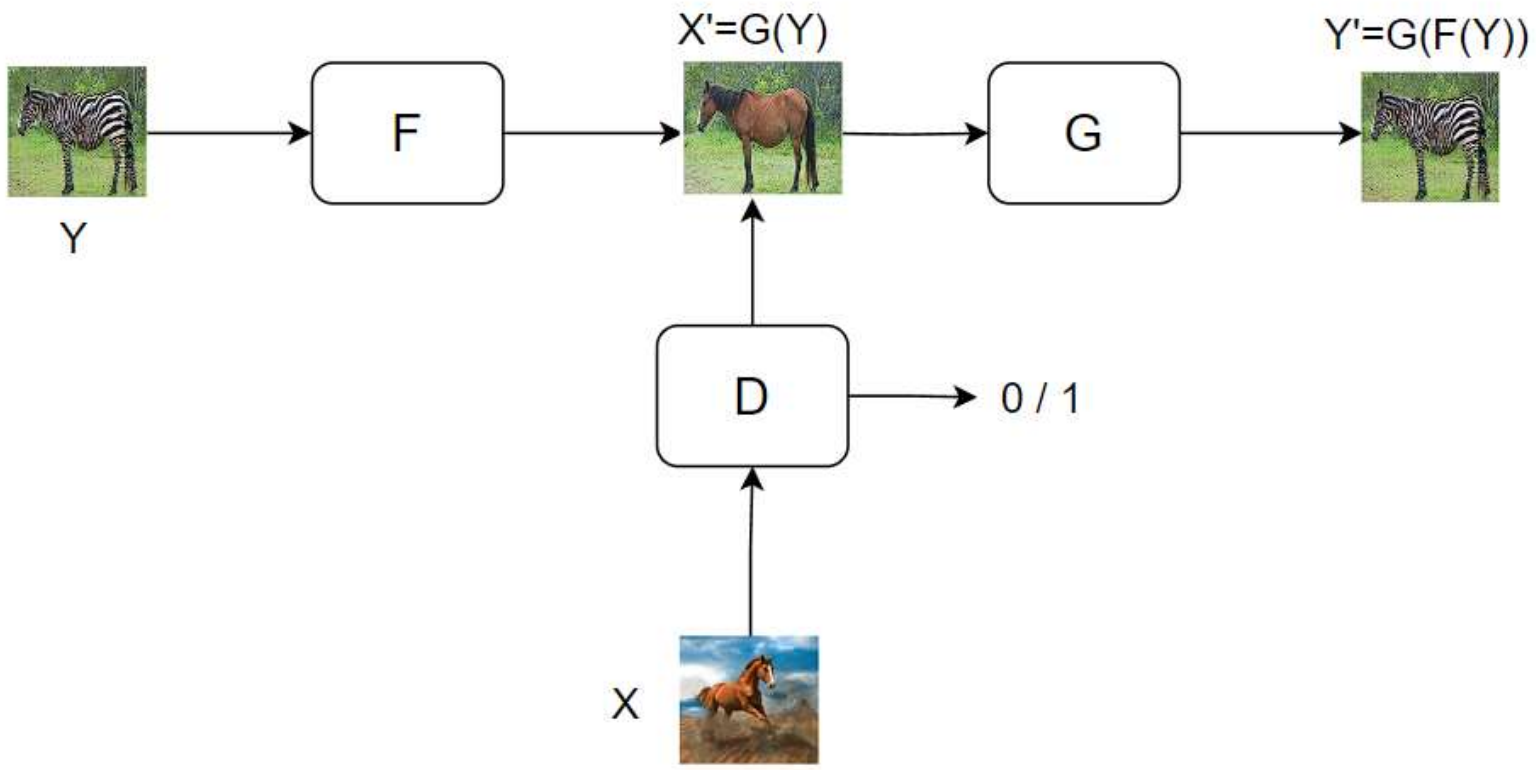
Cycle GAN Architecture

- **2) Converting Y to X and reconstruct Y':** Say X contains the data from the distribution having images of Horses and Y belongs to the distribution with images of Zebra.
  - Y image is given to the Generator Network F which converts it into X' (Horse) retaining the properties of Y. X' is now given to D_X which is assumed to trained on the images of X(horses). The target of D is to discriminate between the X' and X. X' is then given back to the generator G which reconstructs the Y as Y'. The cycle-consistency loss actually determines how far these Y and Y' are. Pictorially, it is represented in the next slide.



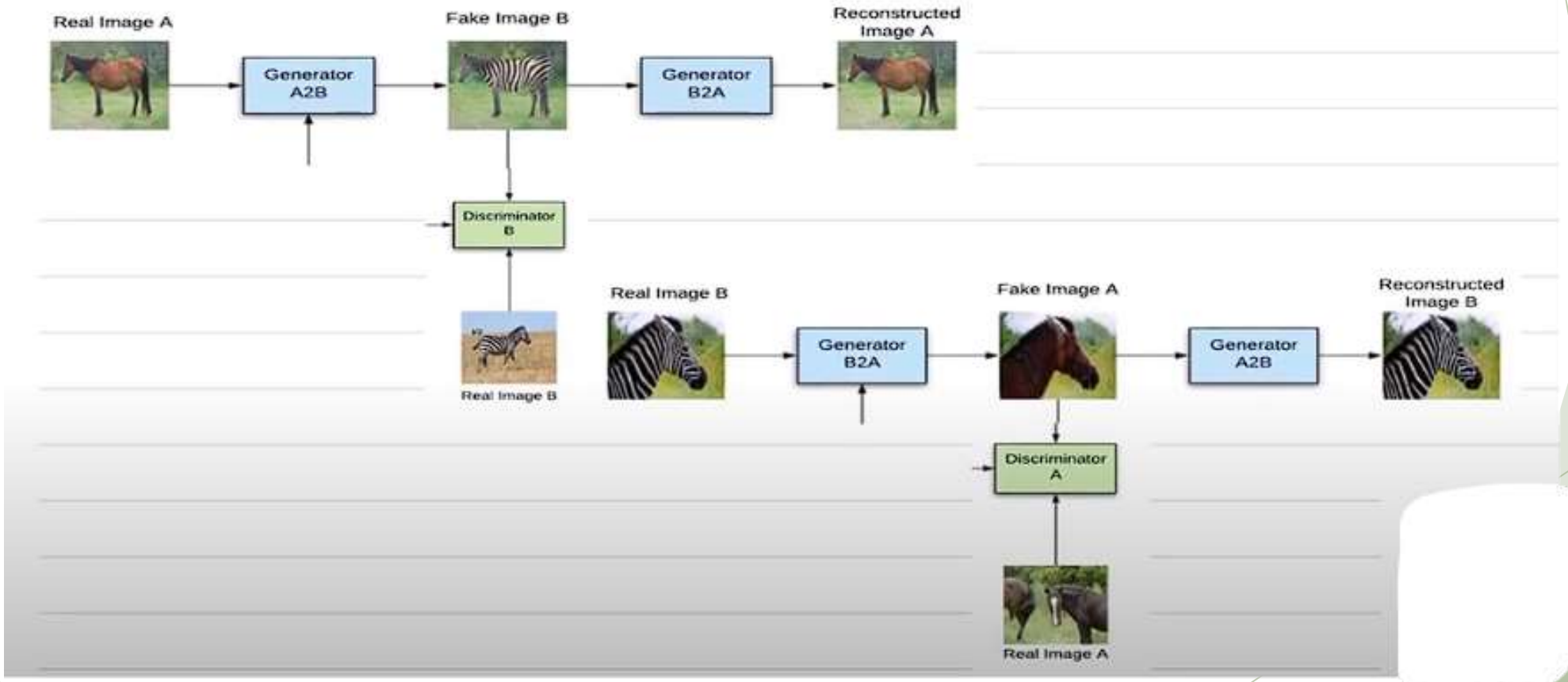
Cycle GAN Architecture

- **2) Converting Y to X and reconstruct Y':**



Cycle GAN Architecture

- Combing 1) and 2) both when X = set of images of Horses and Y = set of images of Zebras.



Cycle GAN Loss function

- **The loss function consists of two types of losses:**
  - **1) Adversarial loss**
  - **2) cycle-consistency loss**

1) Adversarial loss is given as:

$$L_{GAN}(G, D_Y, X, Y)$$

$$= E_{y \sim p_{data}(y)}\left[\log\left(D_Y(y)\right)\right] + E_{x \sim p_{data}(x)}\left[\log\left(1 - D_Y\left(G(x)\right)\right)\right]$$

2) Cycle consistency loss is given as:

$$L_{cyc}(G, F) = E_{x \sim p_{data}(x)}\left[\, ||F(G(x) - x||\right]$$

Cycle GAN Loss function

- **The combined objective function is:**

$$L(G, F, D_X, D_Y) = L_{GAN_1}(G, D_Y, X, Y) + L_{GAN_2}(F, D_X, X, Y) + L_{cyc_1}(G, F) + L_{cyc_2}(F, G)$$

- **Optimization is:**

$$G^*, F^* = argmin_{G,F}\, argmax_{D_X, D_Y}\, L(G, F, D_X, D_Y)$$