Generative Adversarial Networks (GANs):

Human face super-resolution on poor quality surveillance video footage

Farooq, Muhammad, et al. "Human face super-resolution on poor quality surveillance video footage." *Neural Computing and Applications* 33 (2021): 13505-13523.

Figure: Experimental results of the deep learning model (SR-CGAN) for SR reconstruction

Generative Adversarial Networks (GANs):

Resources:

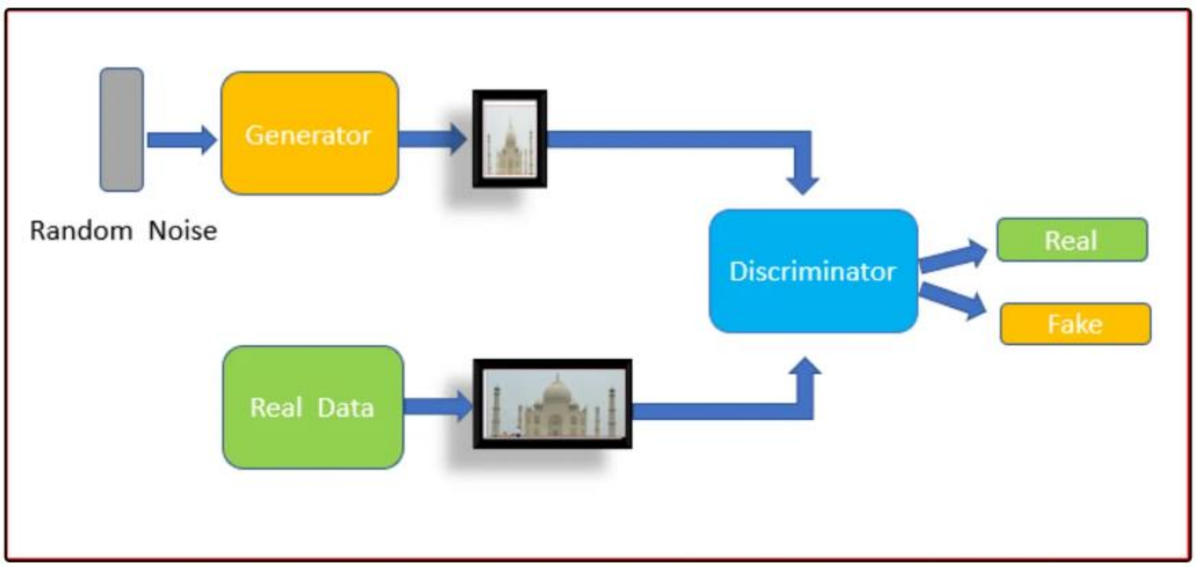
1. https://blog.devgenius.io/the-dueling-minds-a-look-into-the-adversarial-dynamics-of-gans-52002b0fe03d
2. Standford: https://www.youtube.com/watch?v=ANszao6YQuM
3. https://www.youtube.com/watch?v=RRTuumxm3CE&list=PLdxQ7SoCLQAMGgQAIAcyRevM8VvygTpCu

Generative Adversarial Networks (GAN)

- Basics of GAN
- Training (Backpropagation)
- Cost Function Derivation
- Drawbacks of GAN
- Implementation in PyTorch
- Applications of GAN

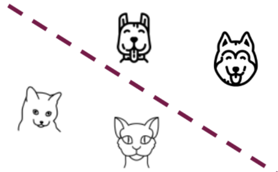
Generative Adversarial Networks (GAN)

- GANs are deep neural network architectures, comprised of two neural networks (generator and discriminator), competing one against the other. *(adversarial term is used)*
- Objective: GANs are neural networks that train in an adversarial manner to generate data mimicking some distribution (D).



Generative Models vs. Discriminative Models

Discriminative models



Features Class

$$X \rightarrow Y$$

$$P(Y|X)$$

Generative models

Noise Class Features

$$\xi, Y \rightarrow X$$

$$P(X|Y)$$

Generative Models vs. Discriminative Models

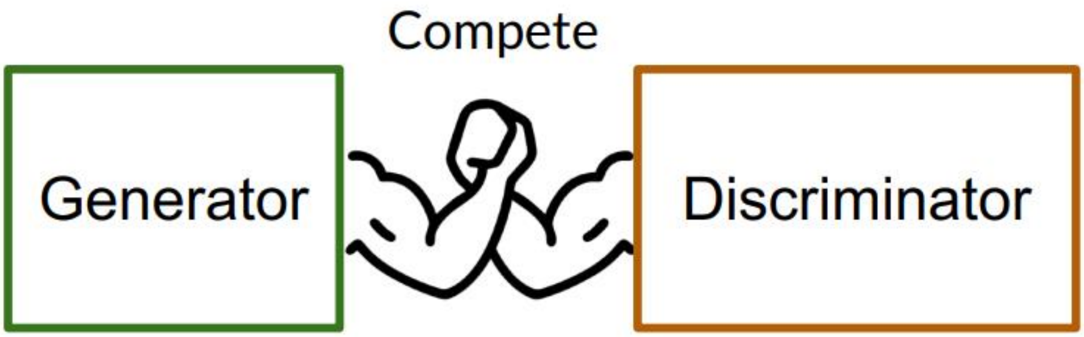
Generative models

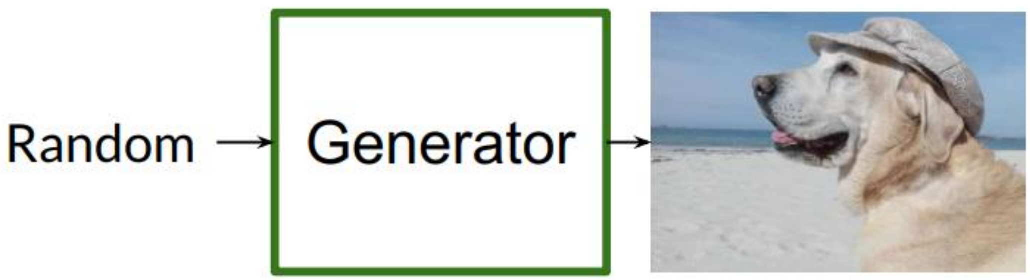
Noise Class Features

$$\xi, Y \rightarrow X$$

$$P(X|Y)$$

Generative Adversarial Networks

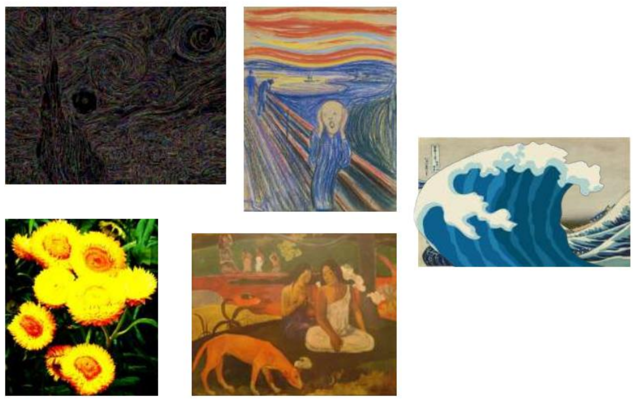




- Generative models learn to produce examples
- Discriminative models distinguish between classes

Generative Adversarial Network

Generator learns to make *fakes* that look **real**



Discriminator learns to distinguish **real** from *fake*

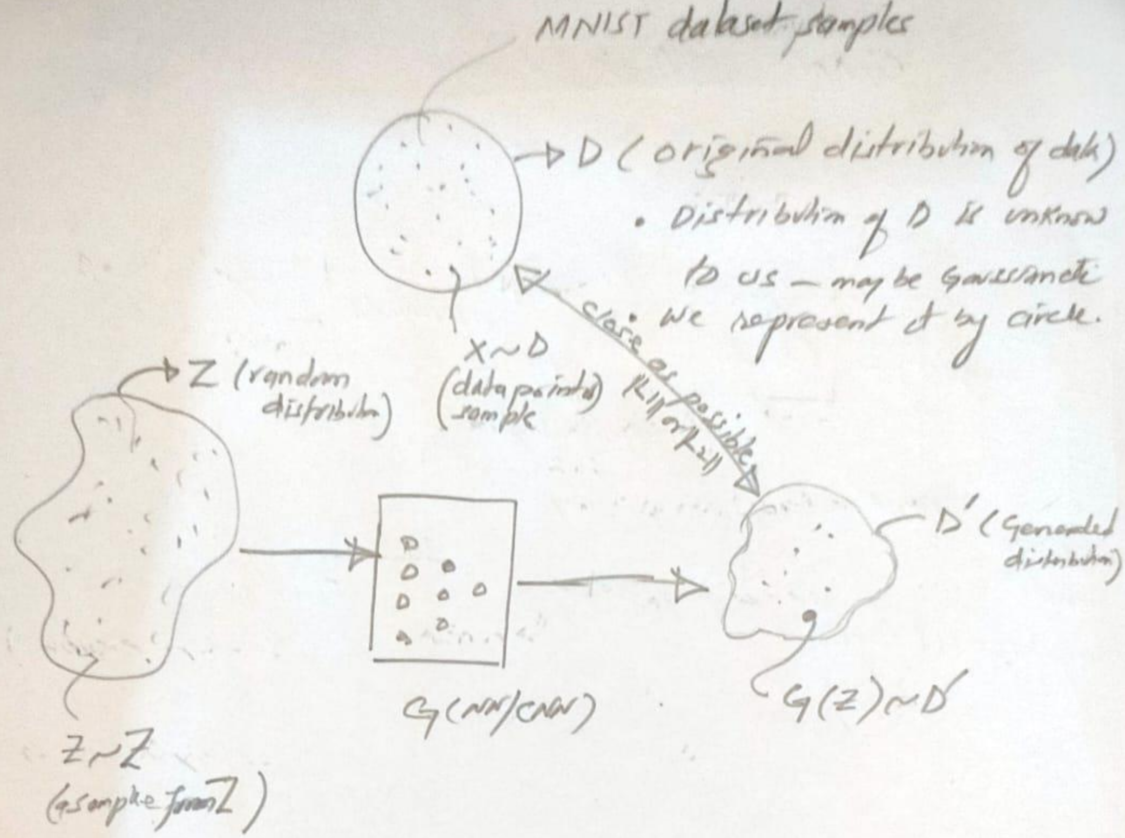
Summary

- The generator's goal is to fool the discriminator
- The discriminator's goal is to distinguish between real and fake
- They learn from the competition with each other
- At the end, *fakes* look real

Two classes of models in ML

1. **Discriminative model**:
    - It is the one that discriminates between two different classes of data.
    - Example: *Face is Fake/Real*
    - Classification problem.

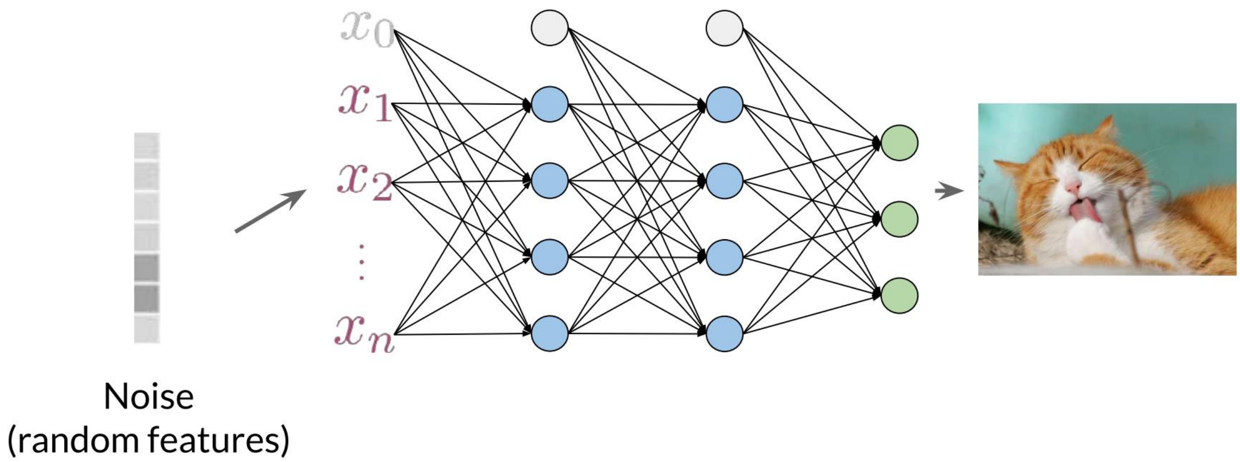
2. **Generative model**:
    - A generative model (G) is trained on training data X, sampled from some true distribution D (like MNIST dataset).
    - It is the one which, given some standard random distribution Z, produces a distribution D′ which is close to D according to some closeness metric.
        1. **Mathematically:**
           $z \in Z$ maps to sample $G(z) \sim D'$.
           $D' = D$.



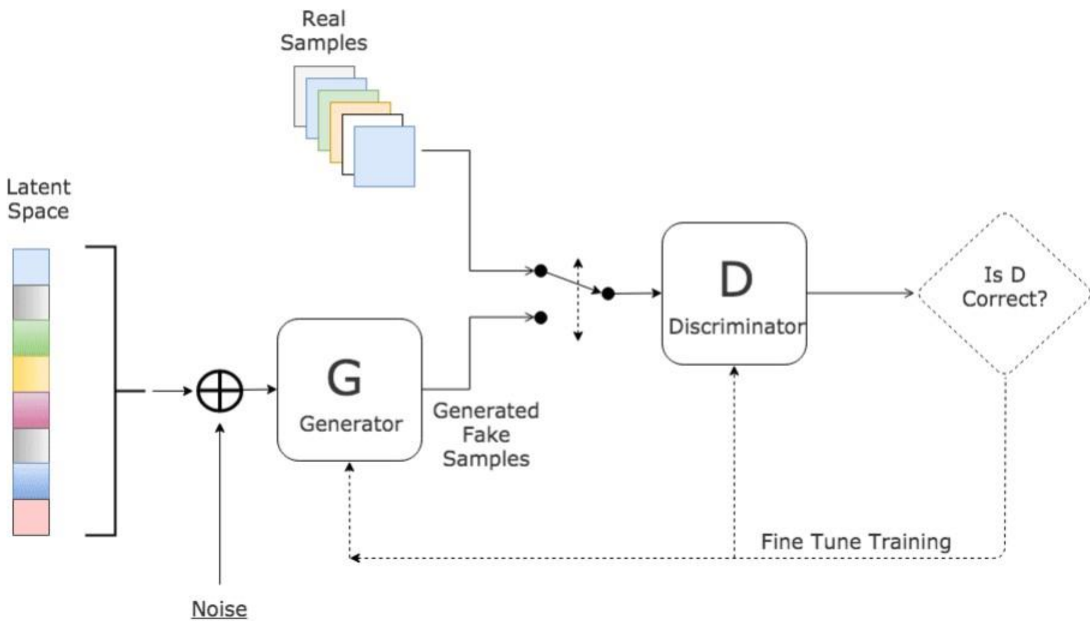
- Objective. D' as close as possible to D
- Generated distribution of data D' is as close as possible to D

• G convert random distribution Z to D'
• A sample inside Z is represented by z
• A sample inside D' is rep. by G(z)~D'
• G(z) are generated sample, which are not represented in D, but follow same distribution as D

Generator:



GANs Block diagram



Discriminator training:

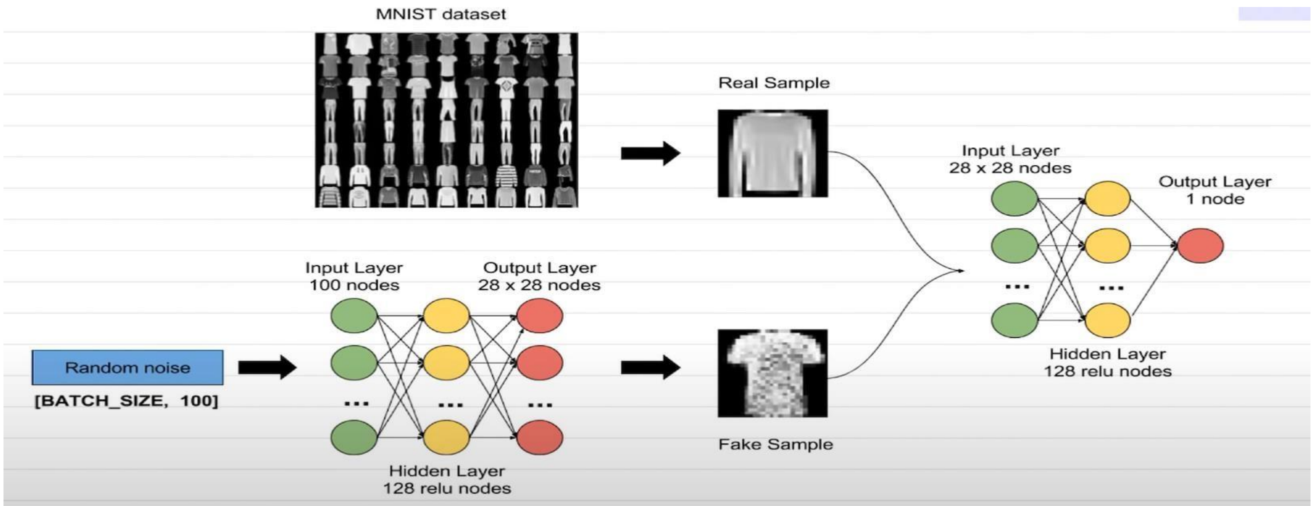
- **for real samples: label = 1**
  D(x) = should be 1 output

- **For fake samples: label = 0**
  D(G(x)) = should be 0 output

Generating training :

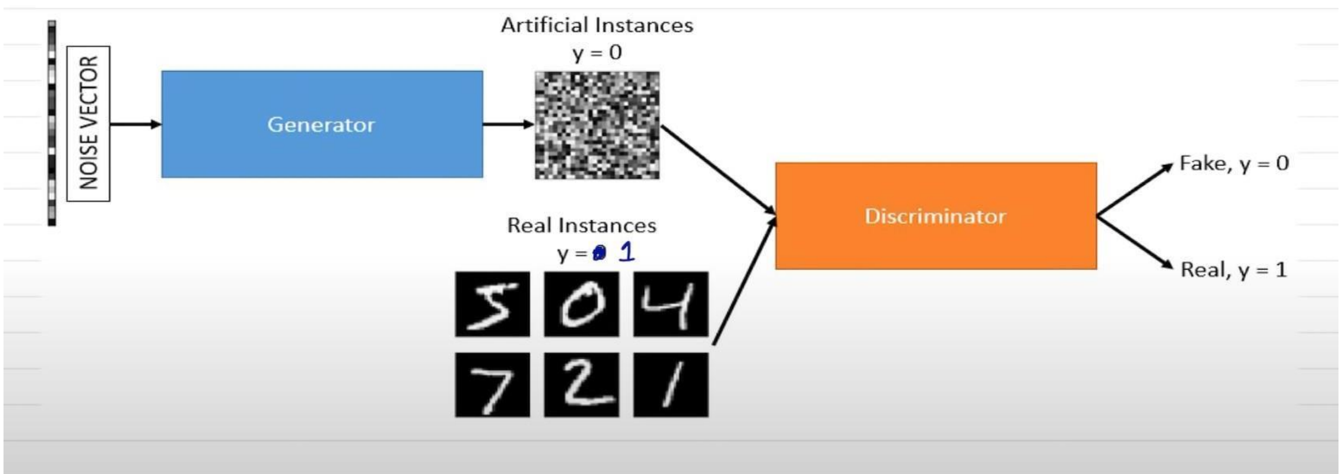
For fake samples => D(G(x)) = should be 1 , and label y = 1 (trick) to train the generator, so that it able to generate fake samples that look like real one.

Internal representation of GAN in case of Fashion MINST, at some intermediate stage of training



Learning mechanism: (W, b learning during back propagation.)

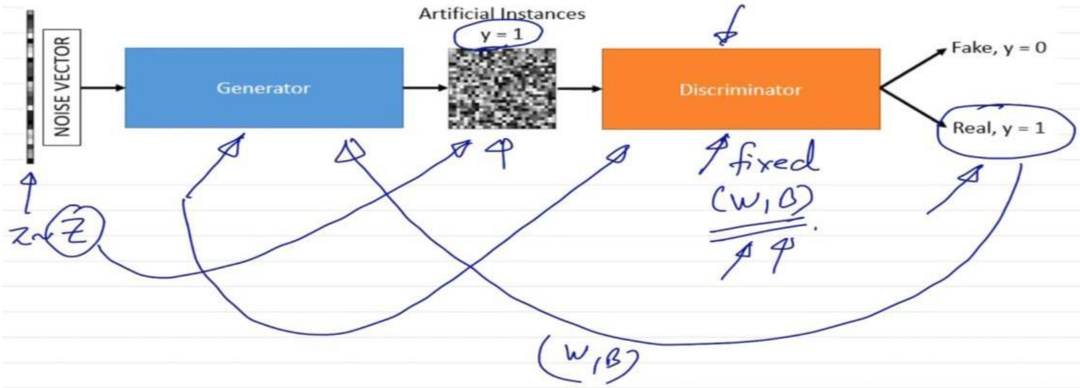
i. GAN is composed of Generator and Discriminator.
ii. Each of these is **held constant** while training the other
iii. Training a **Discriminator** is much easier; it is just like training a **binary classification**.
iv. **OBJECTIVE of the discriminator** is to distinguish between real and fake samples.
v. That's way y=0 for fake and y = 1 for real samples during training of Discriminator.



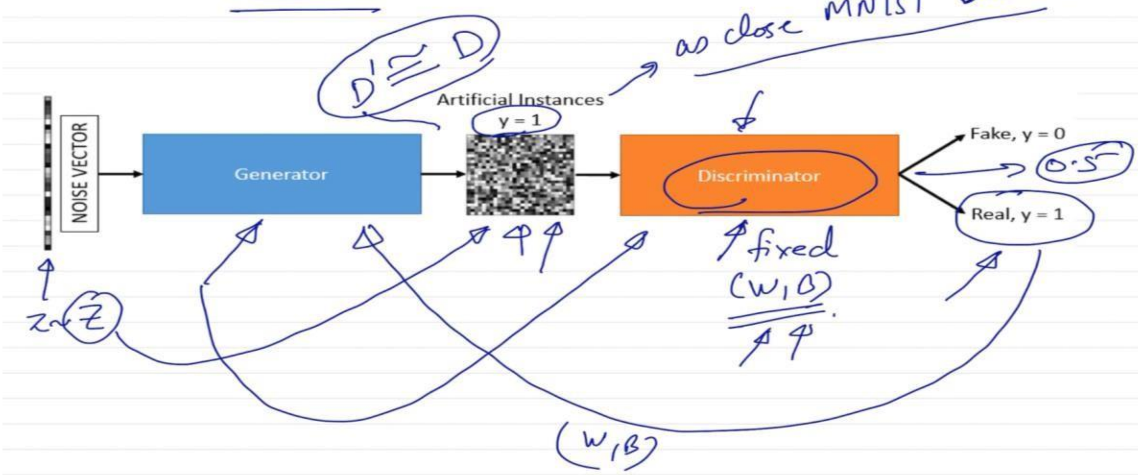
Training of Generator (G):

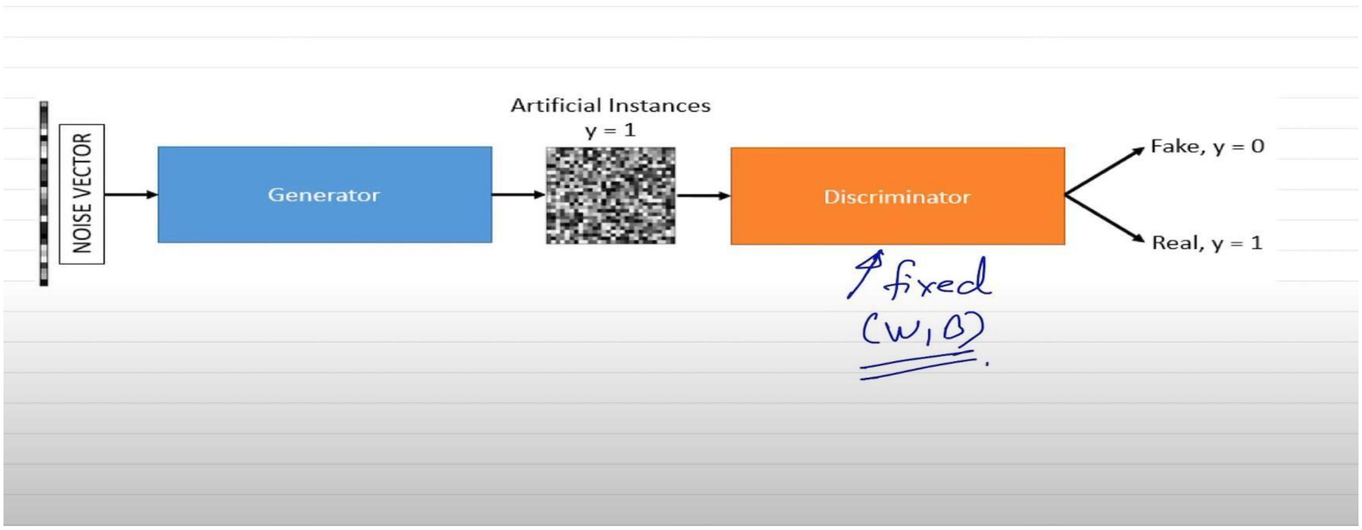
- For training of the Generator, keep the **discriminator fixed**. It means do not update the weights (W and b) of the discriminator. As we don't want to make discriminator too strong to ever beat.
- The **objective of a generator** is to fool the discriminator. The generator produces the fake sample as close as possible to the real sample (D' ≈ D) so that the Discriminator is failed to distinguish between real and fake samples and produces the output of neither 0 nor 1 for both these samples (real and fake)
- Generator objective is discriminator output the value 0.5.
    - To find the best discriminator there is a proposition:
        - D(x) = pdata(x)/ (Pdata(x)+ Pg(x))
        - (Pdata(x) ≈ Pg(x)
        - D(x) = 0.5

- During the traini**ng of G, noise instance z is created from distribution Z**
- **Feed noise instance z belongs to Z to the generator and converted to the artificial instance and we want our discriminator to get fooled by the artificial image and produce the output 1, so because of that, we label the artificial instance with 1, although it is the artificial instance.**
- **During the training of G we are trying to make it smart enough, so that it can fool the discriminator and produce the output by discriminator is 1.**
- **If G is failed to do so (fail the discriminator) then the error is back propagated, and w and b are updated to make it more intelligent- so that it produces the artificial dataset as close as possible to the real MNIST dataset.**
- **Once G is able to achieve the target then the discriminator is produced an output value is 0.5. it means the Discriminator is confused about whether the data is coming from an artificial instance or a real instance then the generator is successful in achieving the desired objective, that is D' ≈ D**



Training the Generator

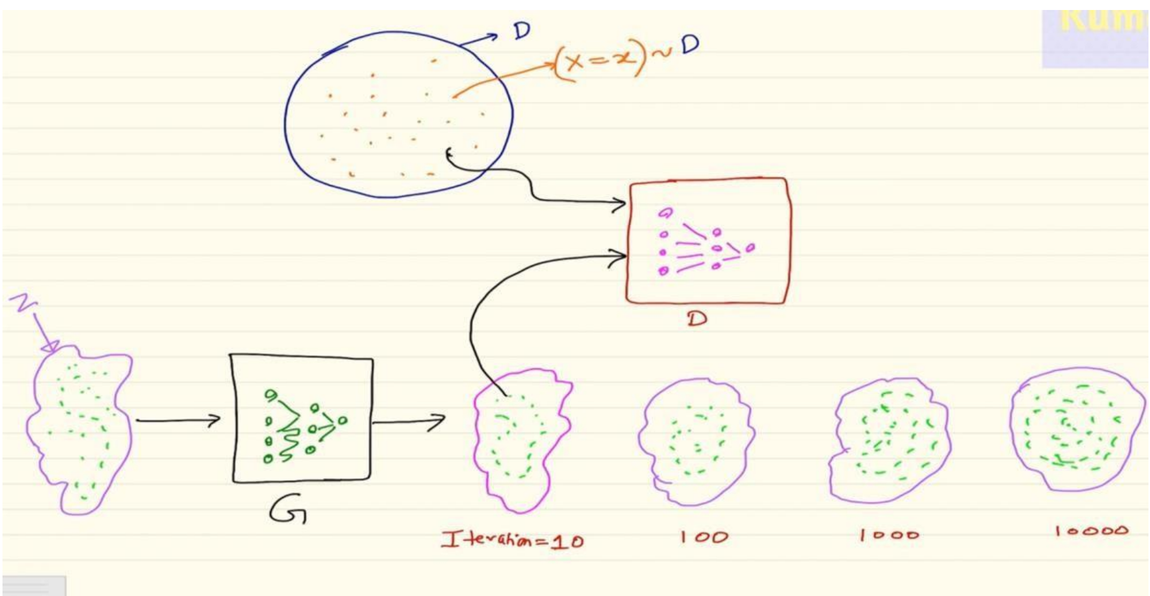




The loss function of GAN:

Discriminator: The role is to distinguish between actual data and fake data

Generator: The role is to create data in such a way that it can fool the discriminator.



- Z is a random distribution
- Black circle is dataset distirution of real dataset (MINST FASHION dataset), ogranze are the data samples (shirt, shoe etc) x sample from X and X has the some unknown distribution D.
- D is unknown distribution - not say it is a gaussian or uniform distribution.
- x is the instance of random variable X that belongs to D
- G tries to generate fake data **D'≈ D so that it can fool the discriminator. After 10 iterations D' is not close to D so Discriminator easily distinguishes between real and fake data. So, the Generator error is high, and it is back propagated to update the weights of the Generator to make it more intelligent. After 10000 iterations D'≈ D, means able to fool the discriminator**

and output it 0.5

This is happen due to the loss function which make the generated distribution close to real dataset distribution At start in first 10 iteration Generator Loss is high at the start and update gradient using the loss and weights update is high. Whereas later in after 10000 iteration loss is low and datadistribution of generated image is close to the real image.

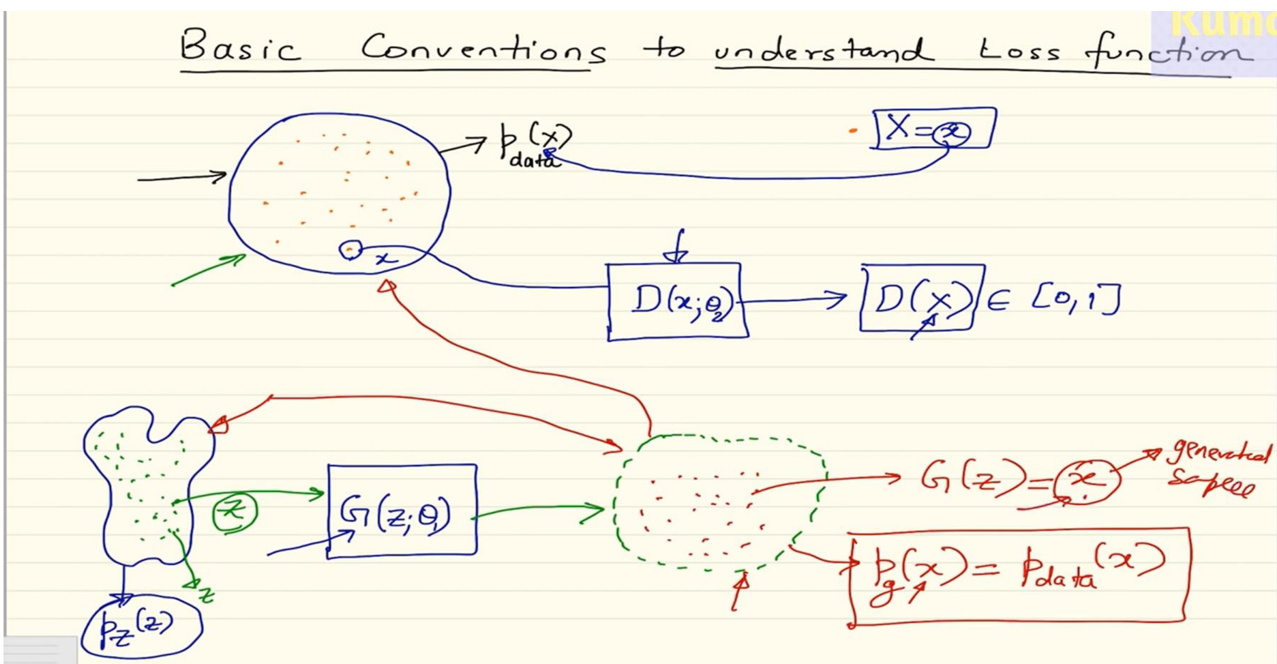
Generator is only able to fool the discriminator when the distirubtion of generated images are closer to real dataset at iteration 100000. This happen only using the loss function that ensure the generator fool the discriminator

GAN is generative model as it generates the new data.

Basic conventions used in the paper (GoodFellow et.al., 2014: Generative adversarial Nets) to understand loss function

Citation: 12969

- **X is our real dataset and x single instance that is sample/belongs to from X**
- **Distribution of X is $P_{data}(x)$** is the probability distribution of real dataset samples which was initially represented by **D**
-
- As **G(z) = x**, generated samples are close to the real sample. The distribution boundary becomes very close to the actual distribution (circle) **$p_g(x) = p_{data}(x)$. in order words D' = D**
- **D(x) is the probability output by Discriminator for x**
- **D(G(z)) is the probability output by Discriminator for input G(z)**
- **$P_{data}(x)$ is the probability distribution of real dataset**
- **$P_g(x)$ is the probability distribution of samples generated by generator from the random feature/code/random number**



Dg(x) = pdata(x)/(pdata(x)+ pg(x))   pg(x) = pdata(s), Dg(x) = 0.5

Basic conventions to understand Loss Function

(paper Goodfellow)

Boundary represent the distribution of dataset (MNIST)

$p_{data}(x)$ is prob distribution of all dots

dots are instance of X

$x \in X$

$x = X$

$Y = 1$

parameterised

$D(x; \theta_2)$ → $D(x) \in [0, 1]$

probability

$\rightarrow D(G(z)) \in [0, 1]$

- classifier

(W, b)

- prob output by D

[Suppose we have random distribution]

z

$G(z, \theta_1)$

$G(z) = x$ generated sample

→ G & D are differentiable functions

→ $\theta_1 \Rightarrow W, b$

Boundary of $p(z)$

z

Z

This distribution boundary become very close to actual distribution (circle) after long time iterations

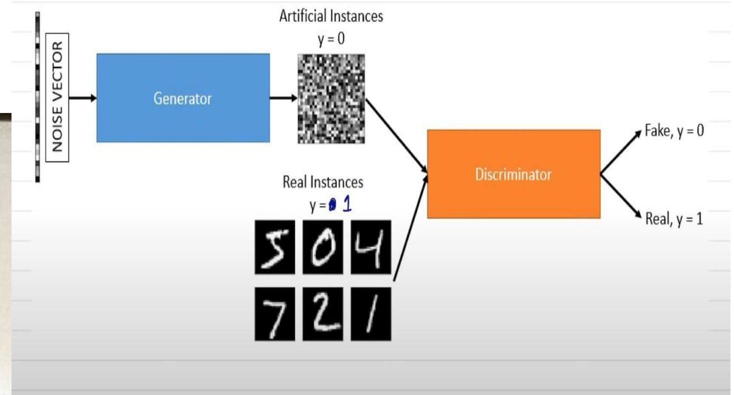
Target $\boxed{p_g(x) \simeq p_{data}(x)}$

$\boxed{D' \approx D}$

- This distribution $D' \approx D$ come after a long time.

Basic Terminology:

- $D(X)$
- $D(G(z))$
- $P_z(z)$
- $P_{data}(X)$
- $P_g(x)$



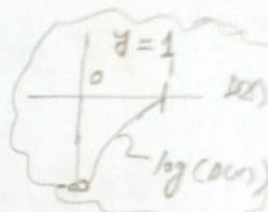
(5)

Loss Function

$$L(\hat{y}, y) = y \log \hat{y} + (1-y)\log(1-\hat{y})$$

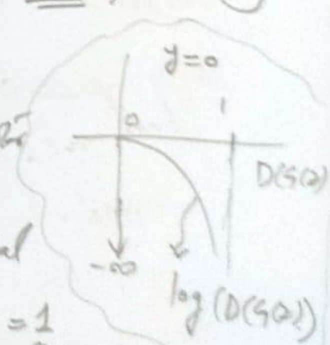
- The label for the data coming from $P_{data}(x)$ is $y = 1$ and output of discriminator $\hat{y} = D(x)$

$$\therefore L(D(x), 1) = \log(D(x)) \quad \text{— (A)}$$



- For the data coming from generator the label is $y = 0$ and output of discriminator $\hat{y} = D(G(z))$

$$\therefore L(D(G(z)), 0) = \log(1 - D(G(z))) \quad \text{— (B)}$$



- The objective of the discriminator is to correctly classify fake vs real dataset. From (A) for $y = 1$, if $D(x) = 1$ then $L(D(x), 1) = 0$, otherwise $L(D(x), 1) = -\infty$ very large.
From (B) for $y = 0$, if $D(G(z)) = 0$, then $L(D(G(z)), 0) = 0$ otherwise $L(D(G(z)), 0) = -\infty$ (very large) →
So we want to maximize both (A) & (B)

$$\max\left[\log(D(x)) + \log(1 - D(G(z)))\right]$$

Default cross entroy function uses minus sign and minimized it, if but if minus sign is added we need to maximized it

- (-) Sign CEF minimization == (+) sign CEF maximization

- Objective / Role of generator is to fool the discriminator.

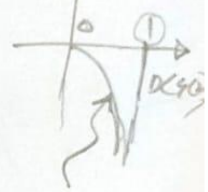
- Therefore, we want $D(G(z)) = 1$

When $y = 0$, $\hat{y} = D(G(z)) = 1$ $\frac{1}{m}\Sigma$

$$L(D(G(z)), 0) = \log(1 - D(G(z)))$$

$$L(1, 0) = \log(1-1)$$

$$= -\infty \text{ (very big -ive no)}$$



$\log(1-D(G(z)))$

we want to minimised $(\log(1-\overline{D(G(z)})$ in order to fool the discriminator & it is only possible when $\log(1-D(G(z))$ is minimum.

$$\min \left(\cancel{\log D(x)} + \log(1 - D(G(z)) \right]$$

↳ it has no role.

- Task of D is to max (+−) & task of G is to min (−+−).

$$\min_G \max_D \{ \log D(x) + \log(1 - D(G(z))) \}$$

• For whole dataset – using [GoodFellow et. al., (2014)]
representation

$$\min_{G} \max_{D} V(D, G) = E_{x \sim P_{data}(x)} [\log D(x)] +$$

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

- expectation of z sample from P_z of z
- weighted average. $(\frac{1}{m} \Sigma)$

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

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

  end for

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

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

end for

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

Goodfellow 2014