URBAN EDGE

Module V

Autoencoders, variational autoencoders

Generative Adversarial Networks (GAN): Discriminative and generative models, GAN discriminator, GAN generator, upsampling, GAN Training, GAN Challenges, loss fns, cross entropy, minimax loss, Wasserstein loss.

Autoencoders:-

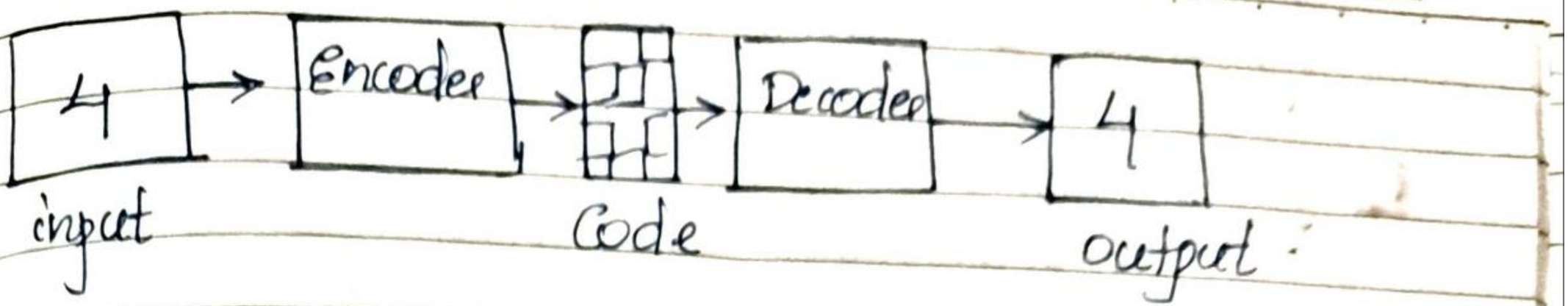
* Autoencoders are a specific type of feedforward Neural networks where the input is the same as the output.

* They compress the input into a lower dimensional 'code' and then reconstruct the output from this representation.

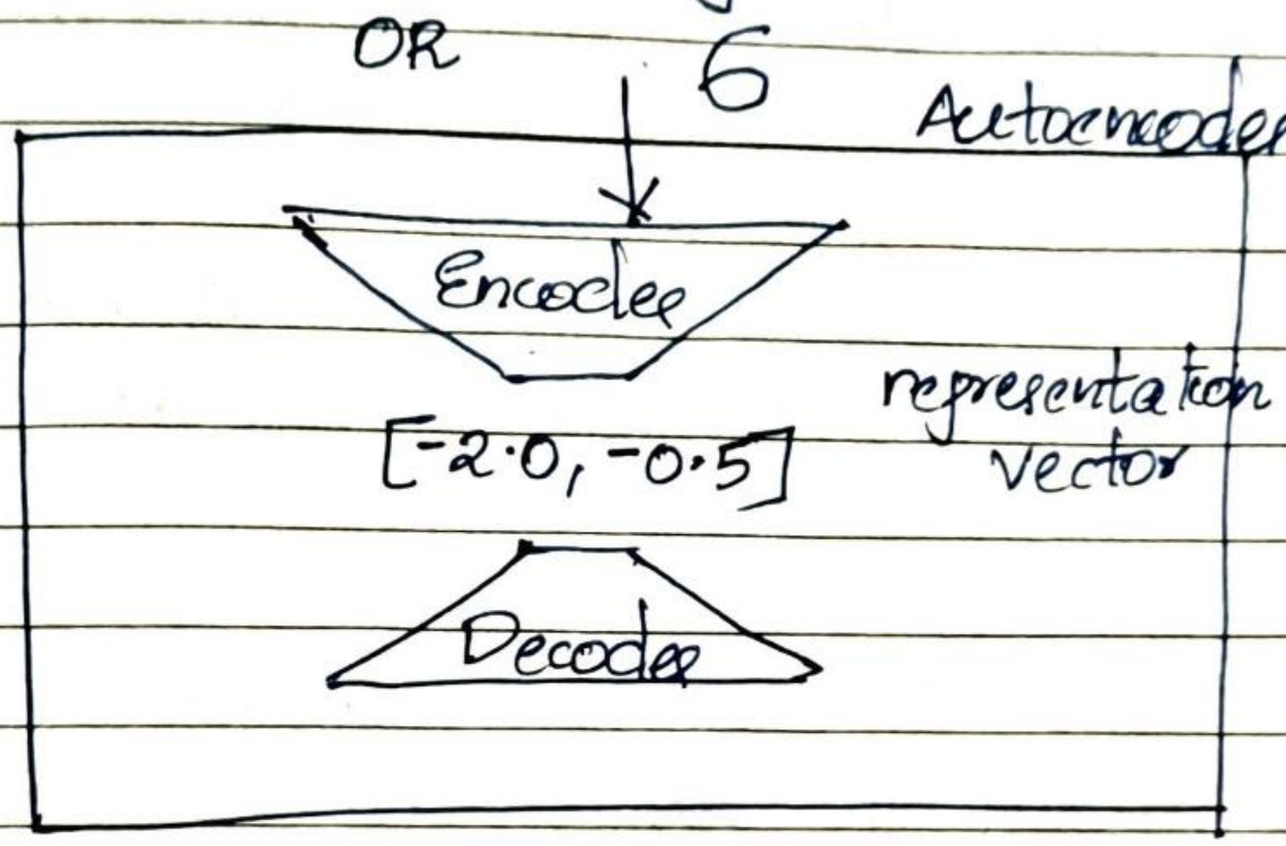
* 'code' is a compact "summary" or "compression" of the i/p, also called the latent-space representation

* The n/w is trained to find weights for the encoder and decoder that minimizes the loss b/w the original i/p and the reconstruction of the i/p after it has passed through the encoder and decoder.

Scanned by TapScanner



* To build an encoder we need 3 things:
  ↳ Encoding method ↳ decoding method
  ↳ Loss function to compare the output with the target



↳ The representation vector is a compression of the original image into a lower dimensional latent space. By choosing any point in the latent space, we should be able to generate novel images by passing this point through the decoder, since the decoder has learned how to convert points in the latent space into viable images.

* Autoencoders can also be used to clean noisy images, since the encoder learns that it is not useful to capture the position of the random noise inside the latent space.

Scanned by TapScanner

URBAN EDGE

Properties of Autoencoders :-

- **Data-specific :-**
  - ↳ Autocoders are only able to meaningfully compress data similar to what they have been trained on.
  - ↳ They only learn features specific for the ~~giving~~ given training data.. so they are different than a standard data compression algorithm.
  - ↳ eg:- we can't expect an autoencoder trained on handwritten digits to compress landscape photos.

- **Lossy :-**
  The o/p of the autoencoders will not be exactly the same as the input, it will be close but degraded representation

- **Unsupervised :**
  Autoencoders are considered as unsupervised learning algorithms/technique ~~is~~ because they don't need explicit labels to train on ie they are self-supervised because they generate their own labels from the training data.

Conv2DTranspose -

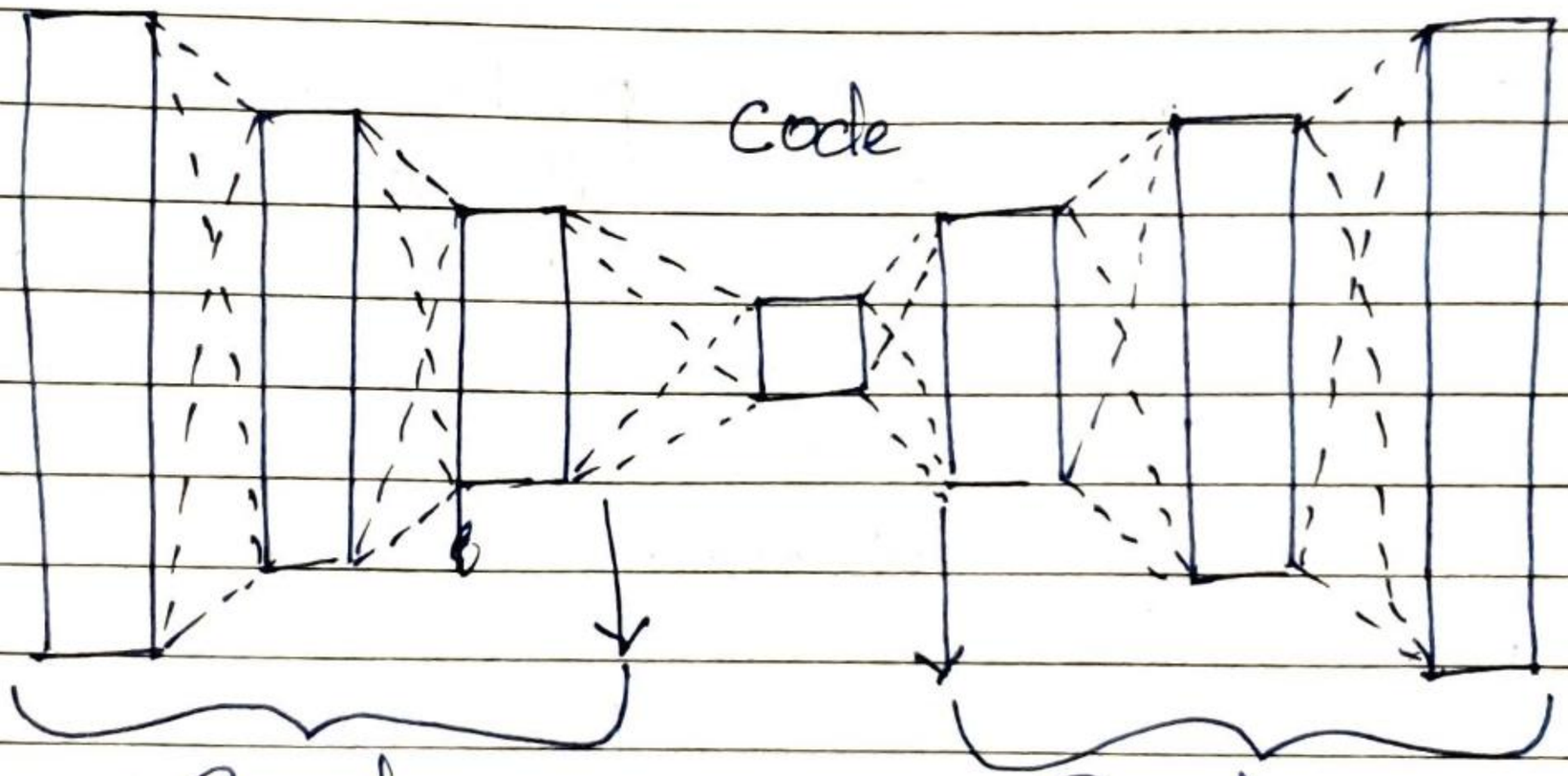
[Convolutional transpose layers uses the same principle as a standard convolutional layer, but is different in ~~that~~ setting strides = 2 doubles the size of i/p tensor in both height and width]

Scanned by TapScanner

Architecture:

* Encoder and decoder are fully-connected feedforward neural networks

* Code is a single layer of an ANN with the dimensionality of our choice.
  ↳ The no: of nodes in the code layer is a hyperparameter that we set before training the autoencoder.



Uses Conv2D { * I/p passes through the encoder, which is a fully-connected ANN, to produce the code

Uses Conv2D Transpose { * The decoder, which is also a fully-connected layer produces the o/p by using the code

* The aim is to produce the o/p of same dimensionality as of input.

Scanned by TapScanner

URBAN EDGE

⇒* 4 hyperparameters that we need to set before training an encoder

1)* ↳ Code size: Number of nodes in the code / middle layer. Smaller the size, more will be the compression.

2)* ↳ Number of Layers:
: layers to be added in the encoder and decoder.

3)* ↳ Number of nodes per layer.
: the no. of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder.

4)* ↳ Loss Function:
Use either mean-squared-error or binary cross entropy.

Types of Autoencoders:-

* Denoising Autoencoder:-

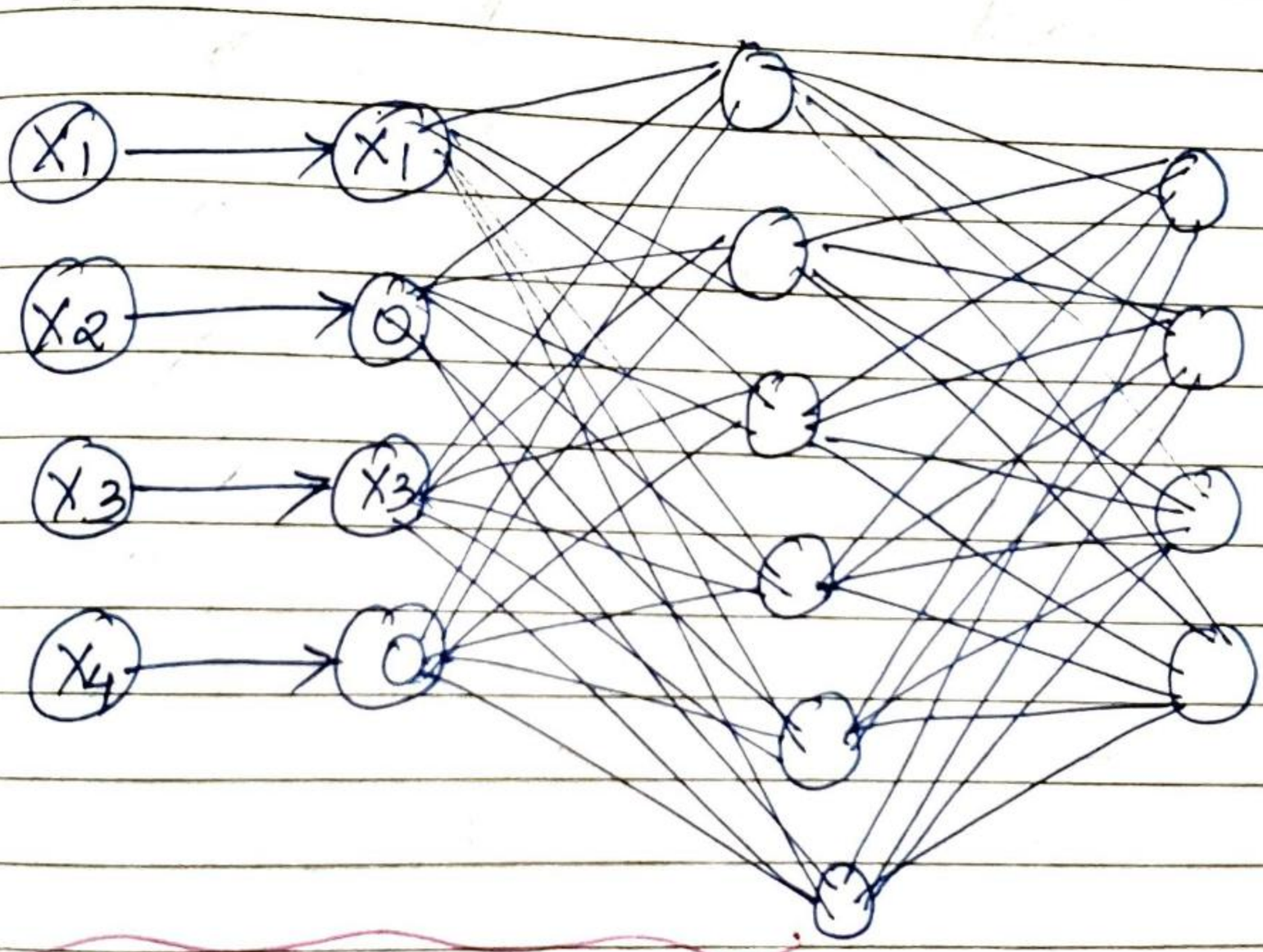
Autoencoders are Neural n/ws which are commonly used for feature selection and extraction. When there are more nodes in the hiddenlayer than input layer, the n/w is risking to learn the so called "Identity Function" / also called "Null Function"

Scanned by TapScanner

meaning that the output equals the input, making the autoencoder useless.

* Denoising autoencoder. Solve this problem by corrupting the data on purpose by randomly turning some of the i/p values to zero. Hence, forcing the encoder to learn the original data after the noise being removed.

* Here, the autoencoder identifies the noise, removes the noise, learns the important features from the input data.



* Undercomplete Autoencoder.

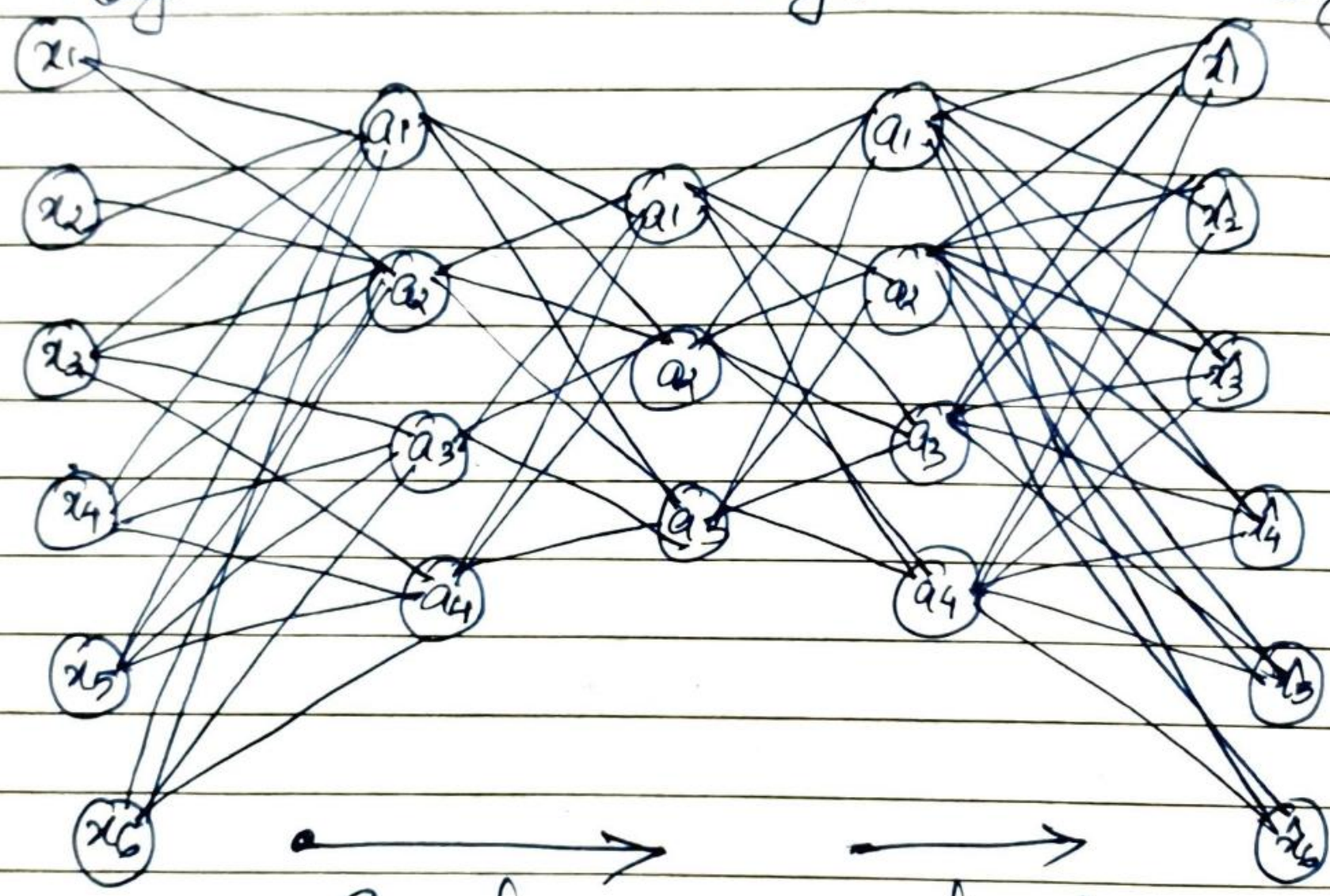
↳* An undercomplete autoencoder is one of the simplest types of autoencoders.

↳ The objective of undercomplete autoencoder is to capture the most important features present in the data.

Scanned by TapScanner

* Undercomplete autoencoders have a smaller dimension for hidden layer compared to the input layer. This helps to obtain important features from the data.

* It minimizes the loss function by penalizing the g(f(x)) for being different from the input 'x'.



Sparse AutoEncoders.

* Sparse autoencoders have hidden nodes greater than input nodes.

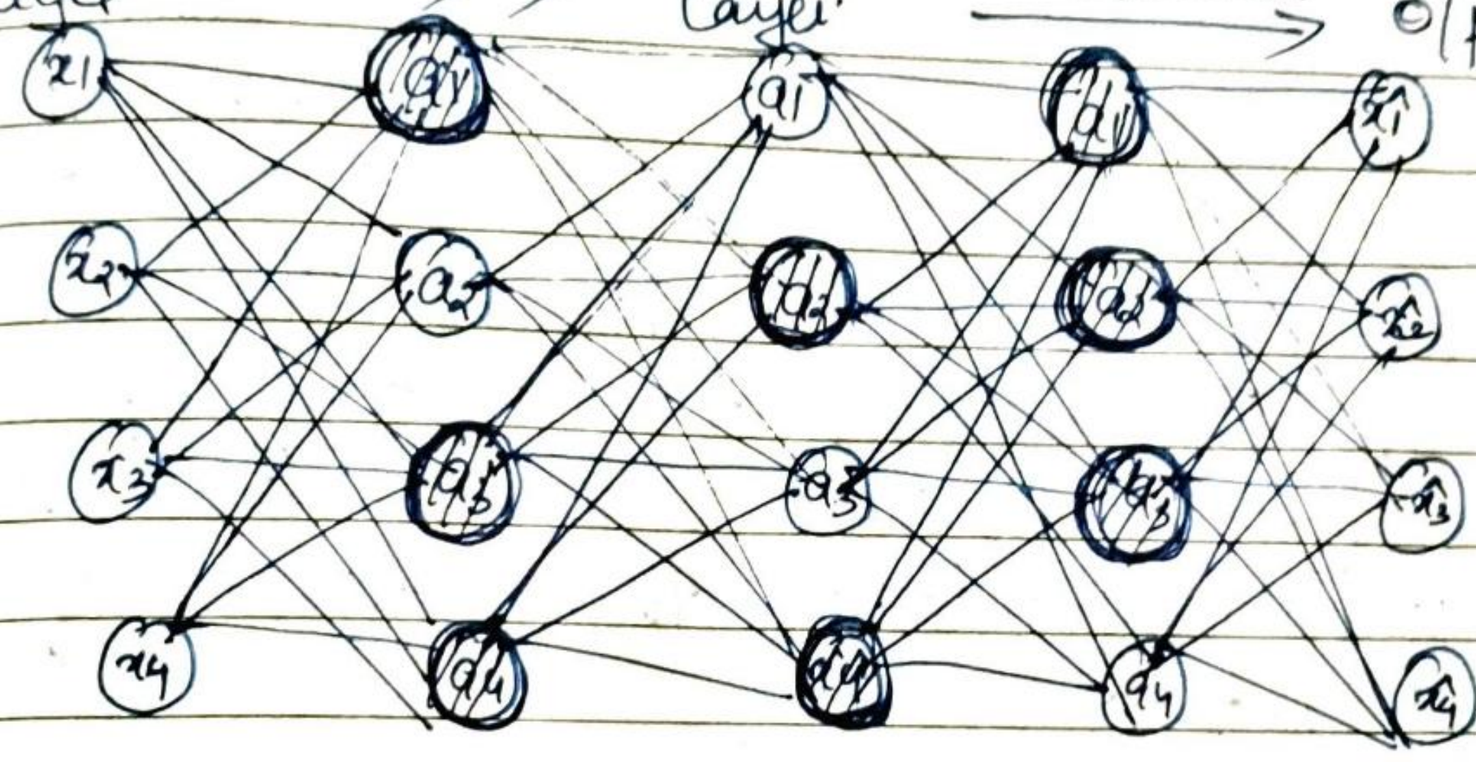
* They can still discover important features from the data.

Scanned by TapScanner

* According to the loss function, penalizes the activations within a layer.

* A generic sparse autoencoder is visualised where the obscurity of a node corresponds with the level of activation. Sparsity constraint is introduced on the hidden layer.

* This is to prevent output layer copy i/p data.



1) Stack Autoencoder
2) Denoising autoencoder ✓
3) Sparse Autoencoder ✓
4) Deep autoencoder
5) Contractive Autoencoder
6) Undercomplete autoencoder ✓
7) Convolutional "
8) Variational autoencoder

Scanned by TapScanner

Generative Adversarial Networks (GIAN)

* Generative Adversial networks are a class of machine learning techniques that consist of two simultaneously trained models
  - → Generator (trained to generate fake data)
  - → Discriminator (trained to distinguish / identify the fake data from real examples.

* The word generative indicates the overall purpose of the model ie creating new data. The data that a GIAN will learn to generate depends on the choice of the training set.
  - eg:- GAN used to generate synthesized images that looks like leonardo da Vinci's, we need to use the training dataset that contains da Vinci's artwork.

* Adversial means a game / competition between two models that constitute the GAN Framework.
  - ↓
  - Generator and Discriminator.

Scanned by TapScanner

URBAN EDGE

⇒ Generator :-

Generator's goal is to create examples that are indistinguishable from the 'real' data in the training set.

eg:- "Producing paintings that look just like da Vinci's".

⇒ Discriminator :-

* Discriminators main objective is to distinguish the fake examples produced by the generator from the real examples coming from the training data set.

eg:- Art expert assessing the authenticity of paintings believed to be da Vinci's.

* The better the Generator gets at creating convincing data, the better the Discriminator needs to be at distinguishing real examples from the fakes ones.

4)

Networks" indicates the class of machine learning models most commonly used to represent the generator and the discriminator (It can be simple FFNN, CNN etc)

Scanned by TapScanner

Working :-

* Generator's goal is to produce examples that capture the characteristics of the training dataset and produce a result that is indistinguishable from the training data.

eg :- Generator be an object recognition model, then it utilizes the "object recognition algorithms" to learn the patterns in images.

Instead of recognizing the patterns, the generator learns to create them essentially from from scratch

ie the input of the generator is a vector of random numbers.

* Generator learns through the feedback it receives from the discriminator's classification

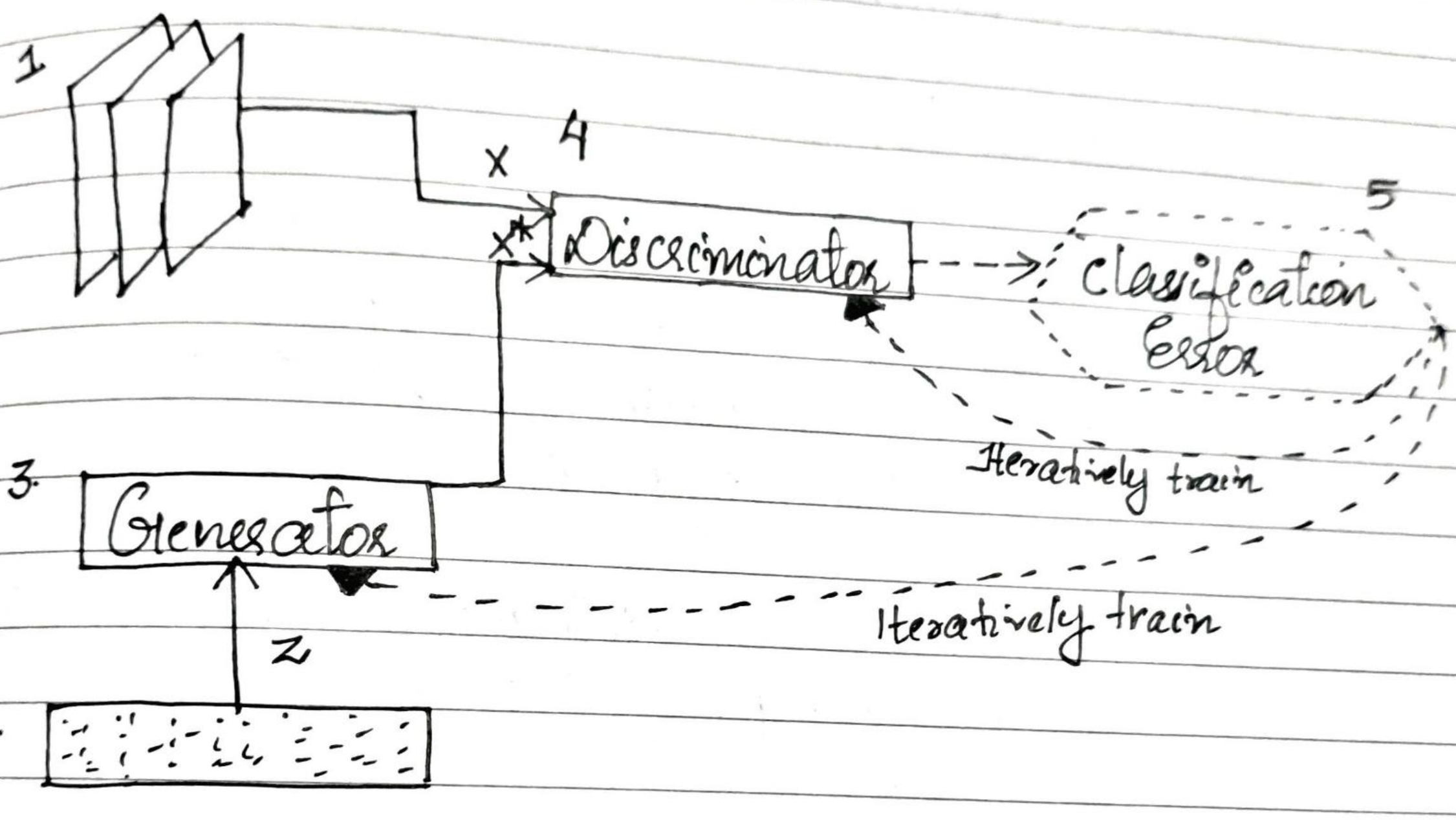
Scanned by TapScanner

	Generator	Discriminator
		The discriminator receive input two sources
Input	A vector of random numbers	• Real examples coming from the training dataset • Fake examples coming from the generator
Output	Fake example that strive to be as convincing as possible.	predicted probability that the i/p example is real
Goal	Generate fake data that is indistinguishable from members of the training dataset	Distinguish b/w the fake examples comming from the Generator and the real examples coming from the training dataset.

Scanned by TapScanner

minimax loss, Wasserstein loss. ... loss functions, ... learning, cross entropy,

GAN Architecture :-



1) Training dataset :-

The dataset of real examples that we want the generator to learn to emulate with near-perfect quality.

Scanned by TapScanner

2) Random Noise Vector :-

* The raw input (z) to the generator network.

* This input is a vector of random numbers that the generator uses as a starting point for synthesizing fake examples.

3) Generator Network :-

The generator takes vector of random numbers (z) as input and outputs fake example (x^*). Goal is to make fake examples that are indistinguishable from the real examples in the training dataset.

4) Discriminator Network :-

* The Discriminator takes as input either a real example (x) coming from the training set or fake example (x^*) produced by the generator.

* Discriminator determines and outputs the probability of whether the example is real.

5) Iterative training / tuning :-

* The o/p determines how good our model is.

* Use the results/o/p to iteratively tune the

Scanned by TapScanner

Discriminator and the Generator networks through backpropagation.

↳ The discriminator's weights and biases are updated to maximize its classification accuracy (predicts 'x' as real and 'x*' as fake)

↳ The Generator's weights and biases are updated to maximize the probability that the discriminator misclassifies 'x*' as real.

GAN Training Algorithm:-

For each training iteration do

1) Train the Discriminator:

(a) Take a random mini-batch of real examples: 'x'.

(b) Take a mini-batch of random noise vectors 'z' and generate a mini-batch of fake examples $G(z) = x^*$

(c) Compute the classification losses for $D(x)$ and $D(x^*)$ and backpropagate the total error to update $\theta^{(D)}$ to minimize the classification loss.

Scanned by TapScanner

2) Train the Generator:-

(a) Take a mini-batch of random noise vectors 'z' and generate a mini-batch of fake examples $G(z) \rightarrow x^*$

(b) Compute the classification loss for $D(x^*)$, and backpropagate the loss ~~of~~ to update $\theta^{(G)}$ to maximize the classification loss

End for.

first → Cost Functions (How GANs differs from CNNs)

* (1) $J^{(G)}$ → Generator's cost function
$J^{(D)}$ → Discriminator's cost function.
$\theta^{(G)}$ → Trainable parameters of Generator ie weights and biases
$\theta^{(D)}$ → Trainable parameters of Discriminator ie weights and biases

∴ Generator's cost function $J^{(G)} \rightarrow (\theta^{(G)}, \theta^{(D)})$
Discriminator Cost function $J^{(D)} \rightarrow (\theta^{(G)}, \theta^{(D)})$

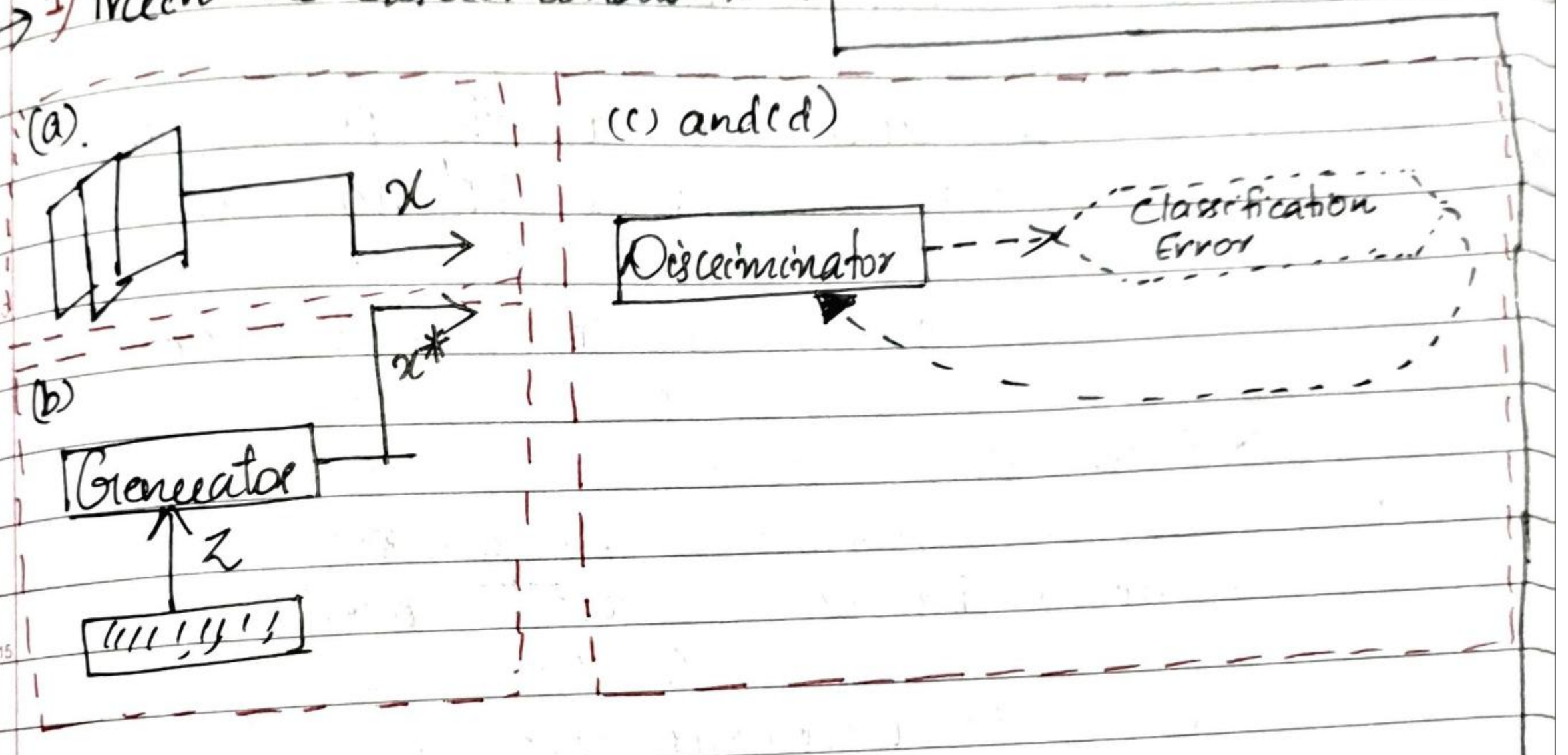
* (2) ↳ Traditional neural network can ~~tune~~ tune all its parameters 'θ' during the training process.

↳ In a GAN, each n/w can tune only its own weights and biases.

Scanned by TapScanner

The generator can learn only $O^{(G)}$ and the discriminator can learn only $O^{(D)}$ during training ie each n/w has control over only a part of what determines its loss.

→ 1) Train the discriminator :-



2) Train the Generator :-

Scanned by TapScanner

Training (continues)

* The generator and the discriminator have different training process
  GAN training proceeds in alternating periods
  1) The discriminator trains for one or more epochs
  2) The generator trains for one or more epochs
  3) Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

* Keep the generator constant during the discriminator training phase. As discriminator training tries to figure out how to distinguish real data from fake, it has to learn how to recognize the generator's flaws.

* Similarly, keep the discriminator constant during the generator training phase.

Scanned by TapScanner

When to stop GAN training
Reaching Equilebeidm !!!

* In neceal networks, stops the tracining process ie when tracining a claesifiee, we measure the clasification eeroe on the tracining and testing sets, and we stop the proces when the validation eeroe starts getting woose
* In a GAN, the two networke have competing objectives : ie when one n/w gets better, the other gets woose.
* When one network improves bcj a certain amocunt, the other network woksens bcj the same amocunt ie One network's gain is equal to other network's loss. This sitvation is called as Nash Equilibricum.
* GAN reaches Nash Equilebricum when the following conditions bee met
  - ↳ The generator produces fake examples that bee indistinguishable from the real data in the tracining dataset
  - ↳ The discriminator can at best randomly guess whether a particular example is real or fake (50/50 goes)

Scanned by TapScanner

GAN Challenges:-

(1) Mode collapse :-

* One of the main failure modes with training a generative adversarial network is called mode collapse

* The basic idea is that the generator can accidently start to produce several copies of exactly the same image.

* A good generator should make a wide variety of images that resembles the training images in all its categories

* Mode collapse happens when the discriminator can't tell the generated images are fake, so the generator keeps producing those same images to fool the discriminator.

2) Slow Convergence | Failure to Converge.

* In GAN network, during training time, the loss or accuracy of 'D' and 'G' measures individually. ie doesn't measure the GAN overall performance

* The GAN model is "good" when an equilibrium is reached b/w the generator and discriminator.

Scanned by TapScanner

3) Overgeneralization

Eg for Overgeneralization
"a cow with multiple bodies but only one head" or viceversa". This happens when the GAN overgeneralizes and learns things that should not exist based on the real data

4) Vanishing Gradients:

If discriminator is too good, then generator training can fail due to vanishing gradients. In effect, an optimal discriminator doesn't provide enough information for the generator to make progress

⇒ To overcome above challenges GAN, some remedies were formulated.

↳ Wasserstein loss
↳ Minmax loss

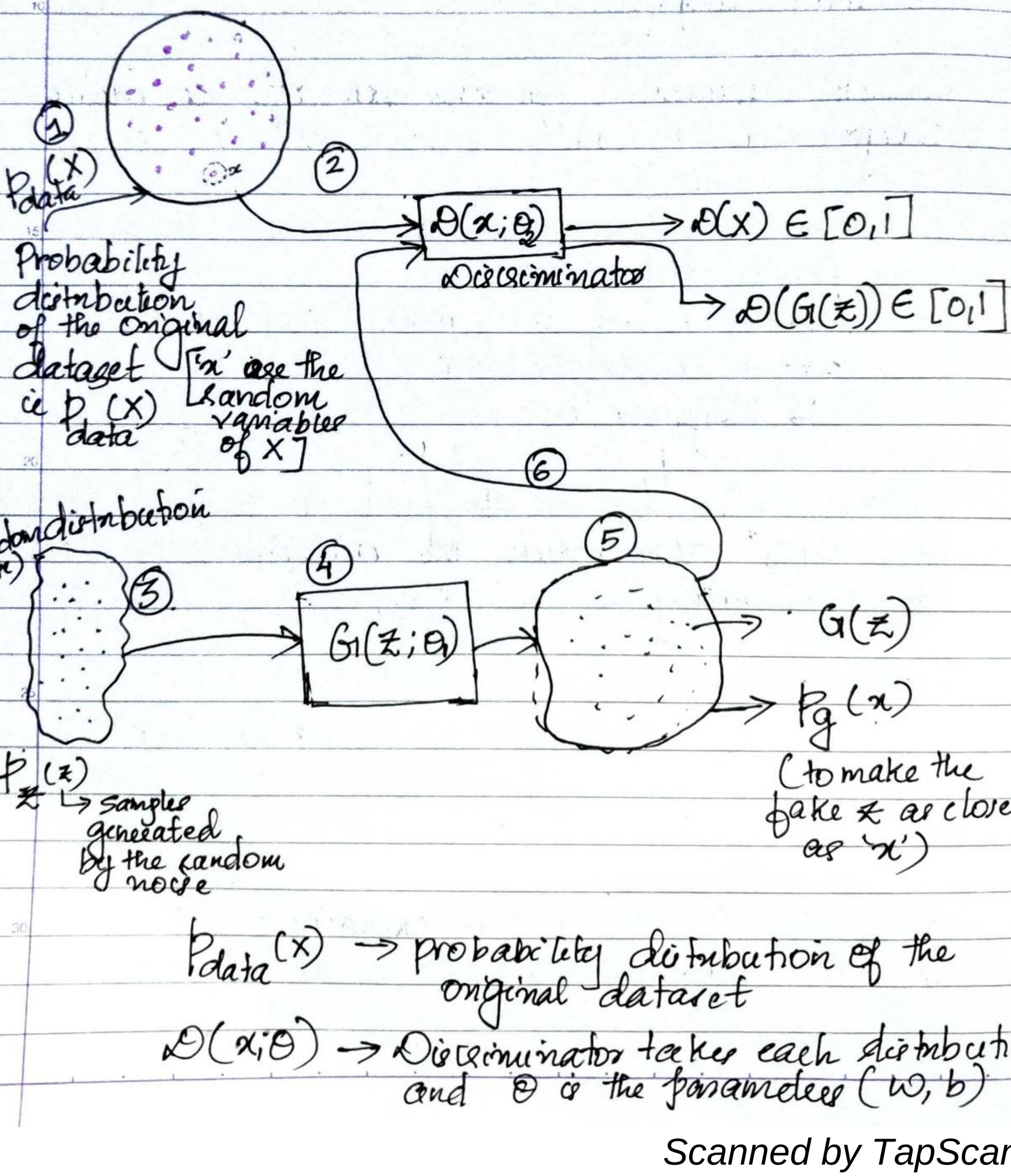
Wasserstein loss is designed to prevent vanishing gradients even when the discriminator approaches to optimality

Scanned by TapScanner

Loss function of GAN:-

* Discriminator : Role is to distinguish between actual data and fake data

* Generator : Role is to create data in such a way that it can fool the generator.



$P_{data}(x)$ → probability distribution of the original dataset

$D(x;\theta)$ → Discriminator takes each distribution and θ is the parameters (w, b)

Scanned by TapScanner

$G_1(z;\theta)$ → Generator's probability distribution from its i/p (x) and 'θ' is the learnable parameter (w, b)
w - weight ; b = bias

$P_g(x)$ → Probability distribution of the fake data estimated by the generator

Cross Entropy

Cross Entropy is a measure of the difference between two probability distributions for a given random variable.

↳ To calculate loss function, GAN used Binary Cross-entropy.

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

$\hat{y}$ = reconstructed image
y = Original image

Discriminator:-

The label for the data coming from $P_{data}(x)$ is $y = 1$

$\hat{y} = D(x)$ ⇒ reconstructed image through discriminator.

$$L(\hat{y}, y) = L(D(x), 1) = \log (D(x)) \longrightarrow (A)$$

Scanned by TapScanner

Generator:-

The label for the data coming from the generator is $y=0$ ie fake.

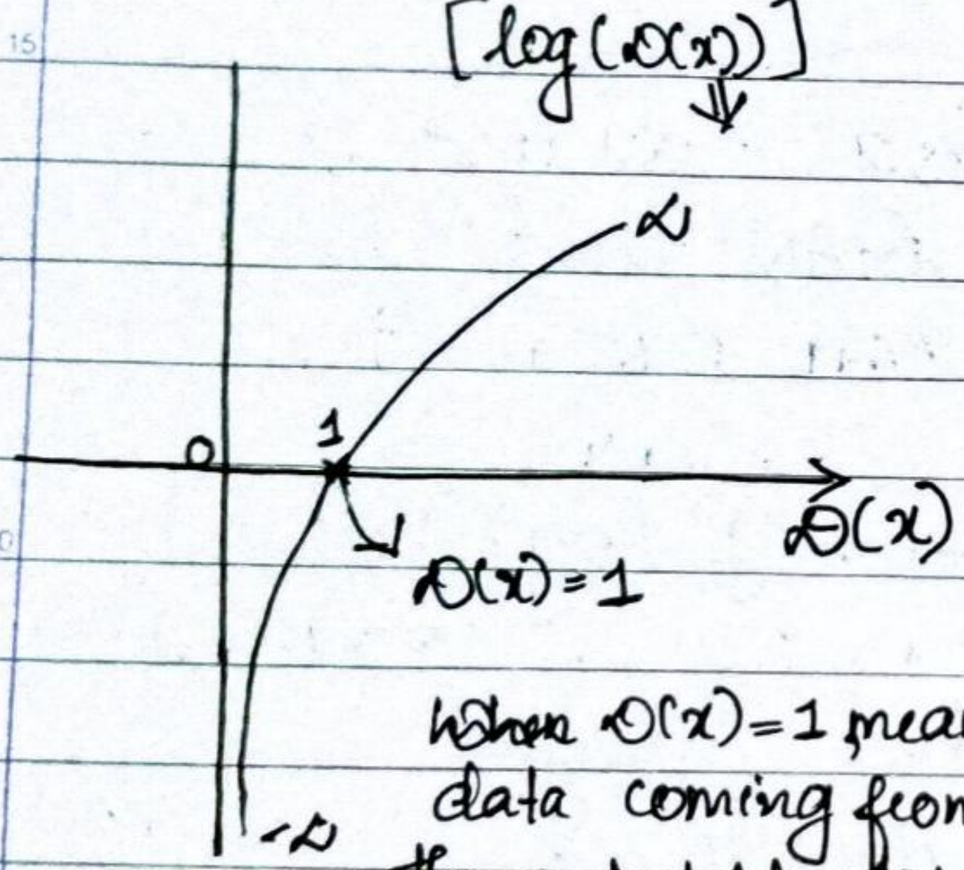
$\hat{y} = D(G(z))$

↳ when the generated image fed into the discriminator. ie called $\hat{y}$.

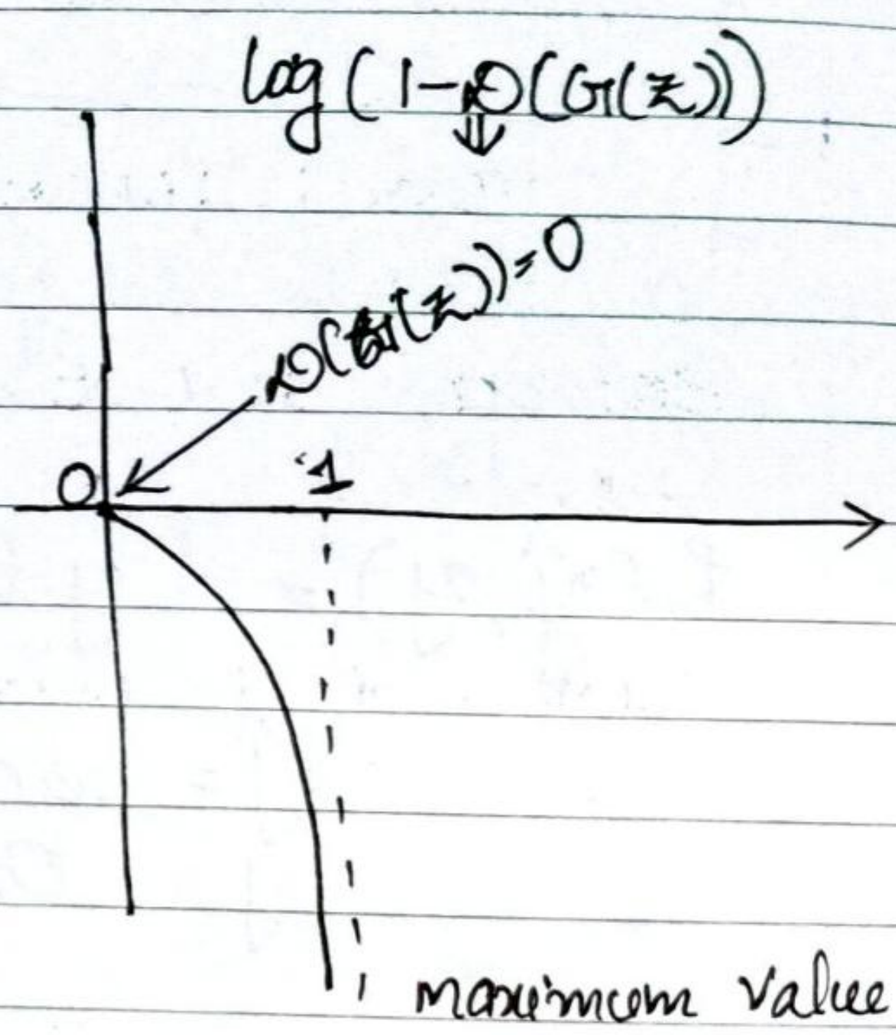
$L(\hat{y}, y) = L(D(G(z)), 0) = (1-0)(\log(1-D(G(z))))$

$= \log(1-D(G(z))) \rightarrow (B)$

→ Objective of Discriminator is to correctly classify fake Vs the real dataset. for this (A) and (B) should be maximized.



when $D(x)=1$, means data coming from the probability distribution is correctly classified as the original data



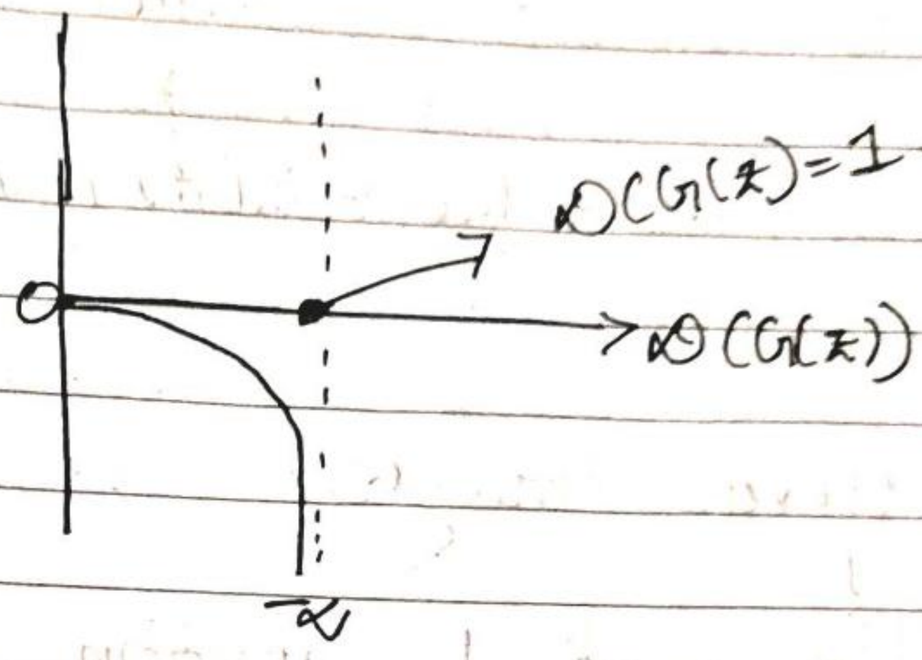
maximum value occurs at 0 so $D(G(z))$ should be 0 to correctly classify fake image as fake.

ie $\max\{\log(D(x)) + \log(1-D(G(z)))\} \rightarrow (C)$

Objective of Generator is to fool the discriminator ie Discriminator needs to produce

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

Equation (A) has no relation with generator, so neglect it.



When 'z' attains to '$-\infty$', $D(G(z))$ becomes 1; so Generator's aim is to minimize the objective function.

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

Together (B) and (C), we can rewrite the equation as

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

(Expectation)

Scanned by TapScanner

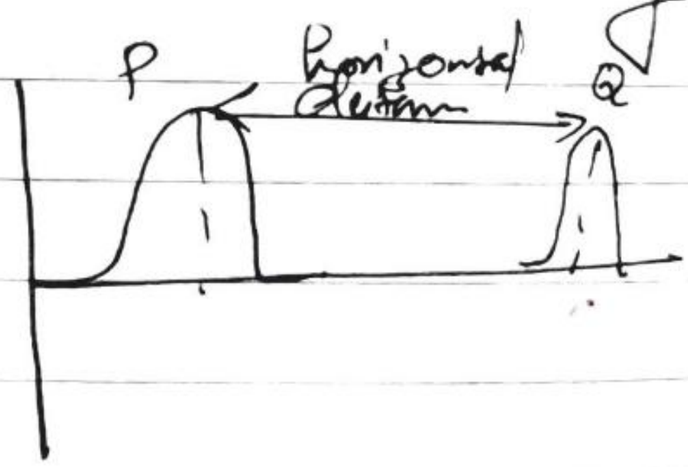
Wasserstein loss :-

VAE = KL divergence
GAN = Jensen-Shannon
WGAN - Wasserstein distance

→ This loss fn depends on the modification of the GAN scheme called WGAN

→ Instead of using a discriminator to classify or predict the probability of generated images as being real or fake, the WGAN changes or replaces the discriminator model with a critic that scores the realness or fakeness of a given image.

→ Mathematical intuition behind this is that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples. WGAN uses Earth-Mover distance or Wasserstein distance for calculating the loss to prevent problem like vanishing gradient.



→ In WGAN, discriminators are actually known as "Critic".

→ Critic loss ⇒ $D(x) - D(G(z))$

The discriminator tries to maximize this function. ie maximize the difference b/w its output on real instances and its output on fake instances

Scanned by TapScanner

→ Generator loss → $\propto O(G(z))$ → Critic's o/p for a fake instances.

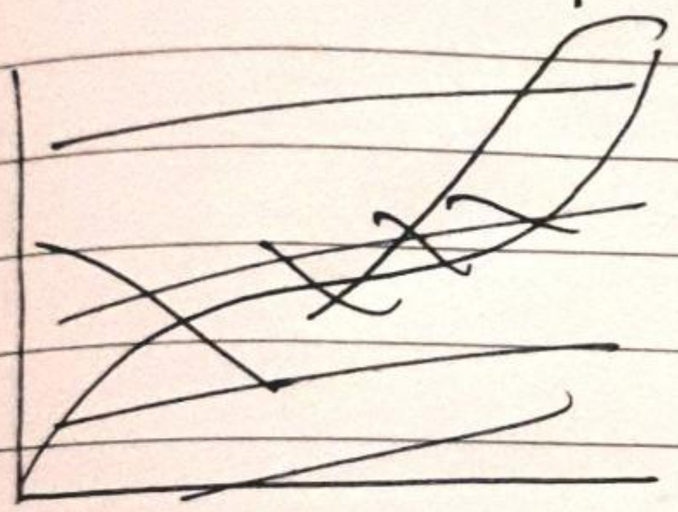
The generator tries to maximize this function.
ie It tries to maximize the Critic's output for its fake instances.

OR

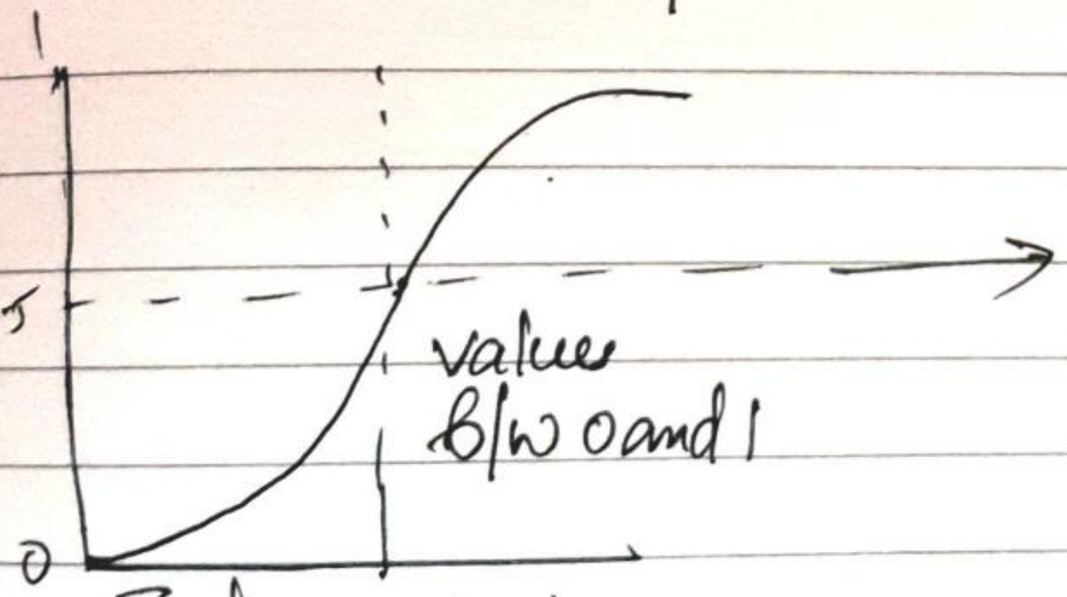
$$\min_{g} \max_{C\ (critic)} E(C(x)) - E(C(g(z)))$$

(C(x) → Critic's)

~~Discriminator o/p~~

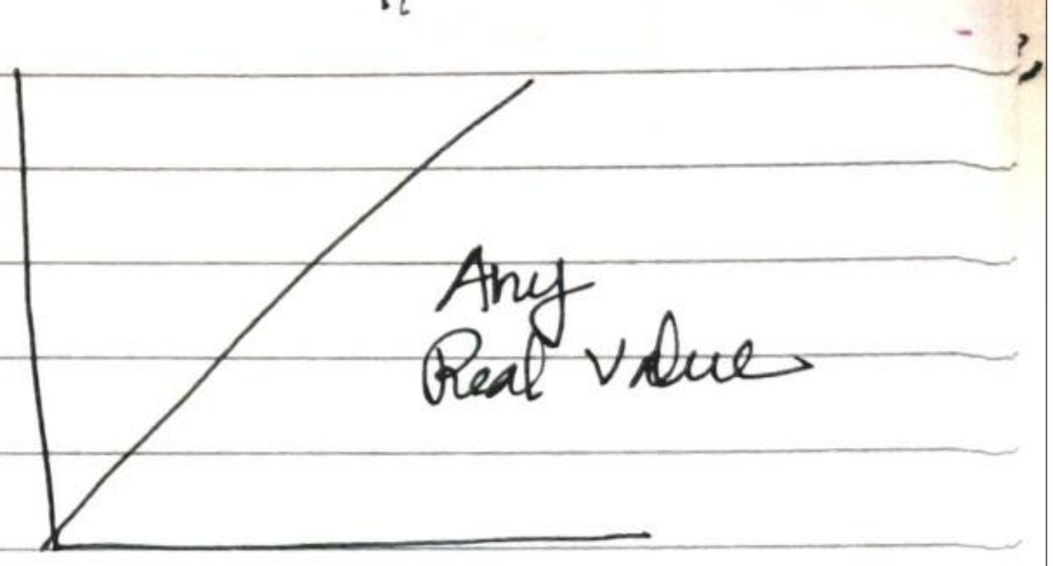


Discriminator o/p



[discriminator uses sigmoid activation in its last layer.

Critic's o/p



⇒ W-loss helps with mode collapse and vanishing gradient problems

Scanned by TapScanner