Image Inpainting

Dwaipayan Haldar
BEE-IV, 002110801187
Dept. of Electrical Engineering
Jadavpur University

4th April 2025

Outline

Introduction

What is Image Inpainting?

Image inpainting originated from an ancient technique performed by artists to restore damaged paintings or photographs with small defects such as scratches, cracks, dust and spots to maintain its quality to as close to the original as possible.

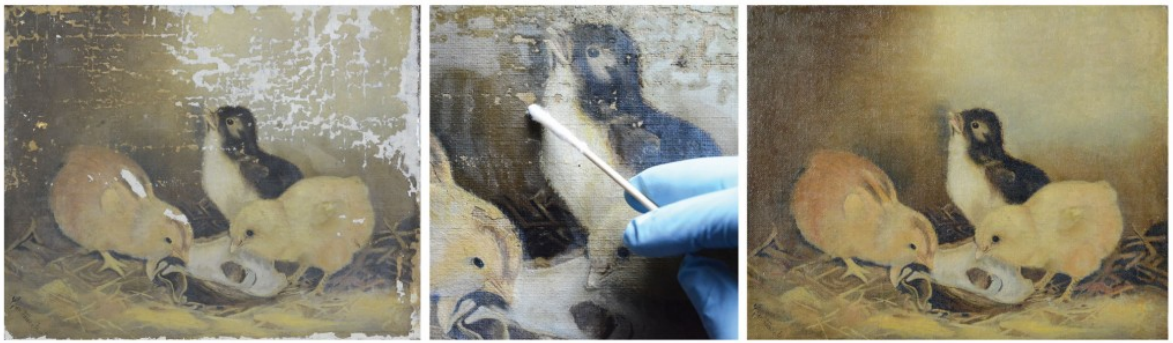


Figure 1: Hand inpainting performed by an artist. Image courtesy of Thottam (2015).

Object Removal

Image inpainting removes unwanted objects by filling the gaps with realistic textures using deep learning (GANs, diffusion models) or traditional methods, ensuring seamless restoration.

(a) Real image (b) Modified image

Figure 2: An example of a people removal method where the person has been manually selected in the real image (a), and then automatically removed in the second image (b) by filling the region concerning the person using an exemplar-based image inpainting method. Reprinted from Criminisi et al. (2004).

Example Problem

Figure 3: Photo of Dwaipayan Haldar

Example Problem

Figure 4: Photo of Dwaipayan Haldar distorted by Manas Sarkar

Example Problem

Figure 5: Ghibli of Dwaipayan Haldar

Example Problem

Figure 6: Ghibli of Dwaipayan Haldar distorted by Pramit Khamrui

Classification

Classification of different Inpainting Techniques

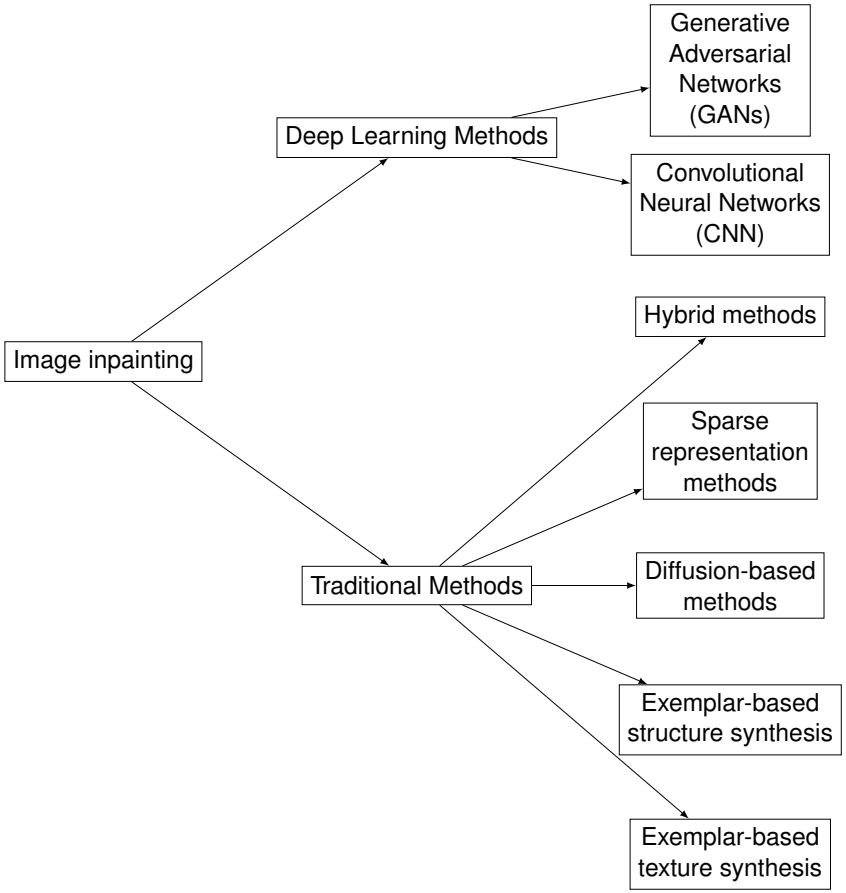


Figure 7: Hierarchical representation of image inpainting techniques in two main categories

Classification of different Inpainting Techniques

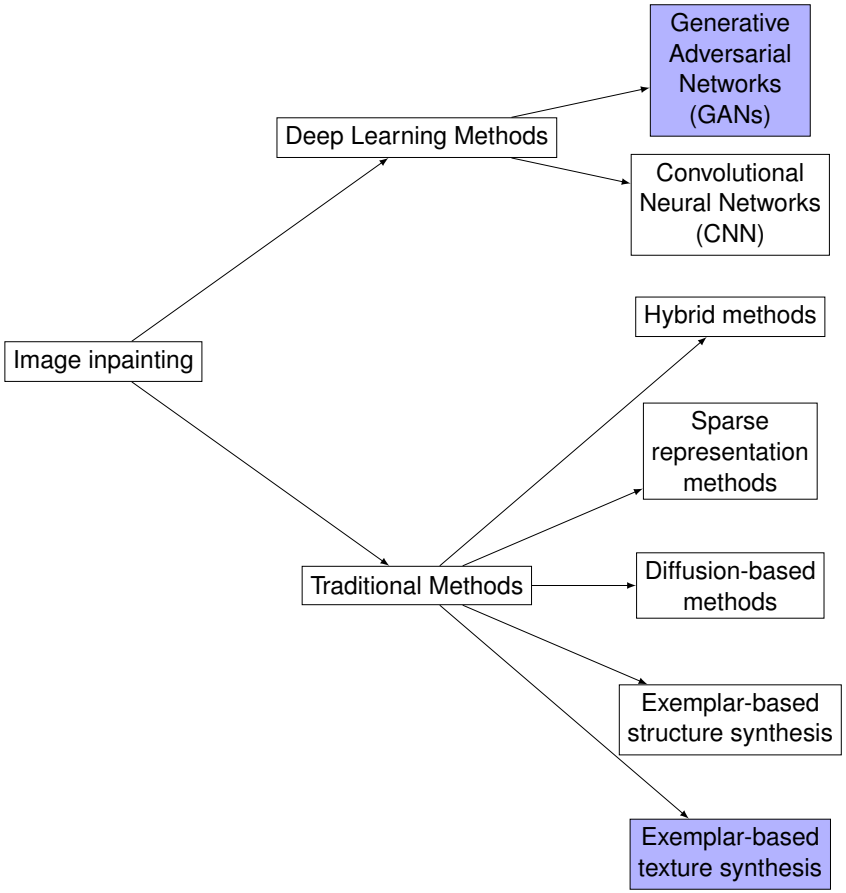


Figure 8: Hierarchical representation of image inpainting techniques in two main categories

Exemplar Based Texture Synthesis

Definitions

What is a texture?

A texture is a visual pattern on an infinite 2-D plane with a stationary distribution at some scale (Efros and Leung, 1999). This pattern refers to the feel (smooth, rough) of the image surface.

Types of textures

Textures have been traditionally classified as either regular (consisting of repeated texels) or stochastic (without explicit texels). However, almost all real-world textures lie somewhere in between these two extremes and should be captured with a single model.

Assumptions

Assumptions

- **Markov Random Field (MRF) Model:**The algorithm assumes that texture can be modeled as a Markov Random Field. This means that the probability distribution of a pixel's brightness value, given the brightness values of its spatial neighborhood, is independent of the rest of the image. In simpler terms, a pixel's value depends only on its neighbors.
- **Stationary Texture:** The usual assumption, which this paper also makes, is that the sample texture is large enough to capture the stationarity of the texture. "Stationarity" in this context means that the statistical properties of the texture are consistent across the image.

Overview of Algorithm

- Let I be an image being synthesized from a texture sample image I_{smp}, where I_{real} is the real infinite texture.
- Let $p \in I$ be a pixel, and $\omega(p) \subset I$ be a square image patch of width w centered at p.
- Let $d(\omega_1, \omega_2)$ denote some perceptual distance between two patches.
- Assume all pixels in I except p are known.
- Based on the MRF model, assume that p is independent of $I \setminus \omega(p)$ given $\omega(p)$.

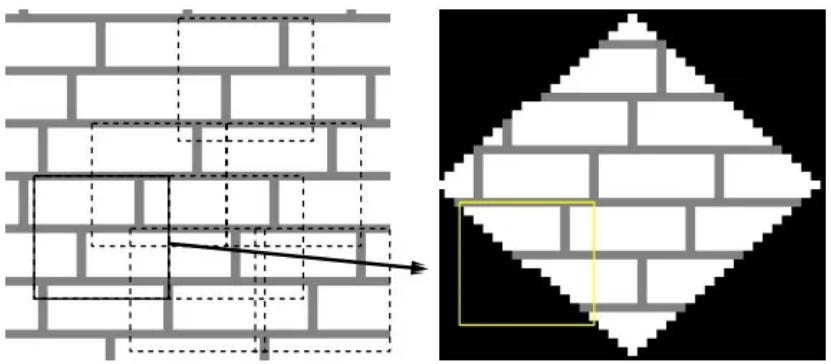


Figure 9: Algorithm Overview

Overview of the algorithm

- Define the set:

$$S(p) = \{\omega' \subset I_{real} : d(\omega', \omega(p)) = 0\}$$

  containing all occurrences of $\omega(p)$ in I_{real}.
- Estimate the conditional probability density function (pdf) of p using a histogram of all center pixel values in $S(p)$.
- Since only a finite sample I_{smp} is available, there may not be exact matches for $\omega(p)$.
- Use a heuristic to find a plausible subset $S_0(p) \subset S(p)$ for sampling.
- Implement a nearest neighbor technique: find the best match $\omega_{best} = \arg\min_{\omega} d(\omega(p), \omega)$ in I_{smp}.

Overview of the algorithm

- Include all patches ω satisfying:

$$d(\omega(p), \omega) < (1 + \epsilon) d(\omega(p), \omega_{best})$$

  where $\epsilon = 0.1$.
- Use the center pixel values of patches in $S_0(p)$ to construct a histogram for p.
- Sample from this histogram, either uniformly or weighted by d.
- Define a suitable distance metric d, such as normalized sum of squared differences (SSD) with Gaussian kernel.

$$d_{ssd} \cdot G = \frac{\sum_{i,j} G(i,j) \cdot (F_{i,j} - I_{i,j})^2}{\sqrt{\sum_{i,j} F_{i,j}^2 \cdot \sum_{i,j} I_{i,j}^2}}$$

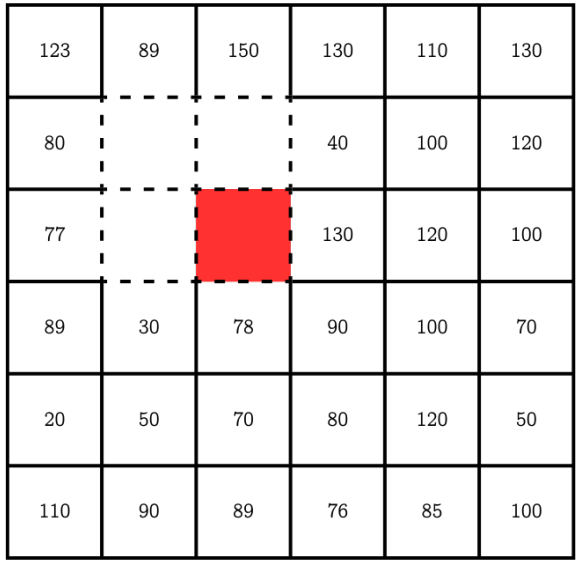
Hole filling with Texture Synthesis

123	89	150	130	110
80	90	70	40	100
77	67	59	130	120
89	30	78	90	100
20	50	70	80	120

123	89	150	130	110	130
80			40	100	120
77			130	120	100
89	30	78	90	100	70
20	50	70	80	120	50
110	90	89	76	85	100

Hole filling with Texture Synthesis

123	89	150	130	110
80	90	70	40	100
77	67	59	130	120
89	30	78	90	100
20	50	70	80	120



Hole filling with Texture Synthesis

123	89	150	130	110
80	90	70	40	100
77	67	59	130	120
89	30	78	90	100
20	50	70	80	120

123	89	150	130	110	130
80			40	100	120
77			130	120	100
89	30	78	90	100	70
20	50	70	80	120	50
110	90	89	76	85	100

Hole filling with Texture Synthesis

123	89	150	130	110
80	90	70	40	100
77	67	59	130	120
89	30	78	90	100
20	50	70	80	120

123	89	150	130	110	130
80			40	100	120
77			130	120	100
89	30	78	90	100	70
20	50	70	80	120	50
110	90	89	76	85	100

Results

Figure 10: Inpainted Image with 5*5 window

Figure 11: Inpainted Image with 13*13 window

Figure 12: Inpainted Image with 25*25 window

Results

Figure 13: Original Image

Figure 14: Inpainted Image with 25*25 window

Discussions

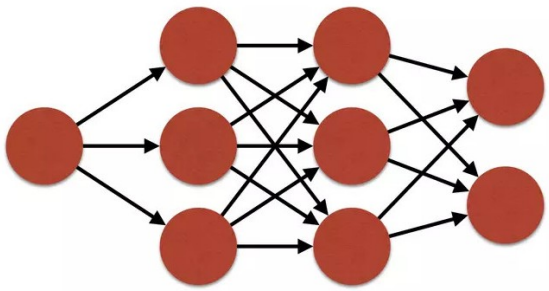
Some Key Points

- Very Simple
- Surprisingly good results
- But very slow
- Limited in terms of mask size, accuracy and sometimes efficiency

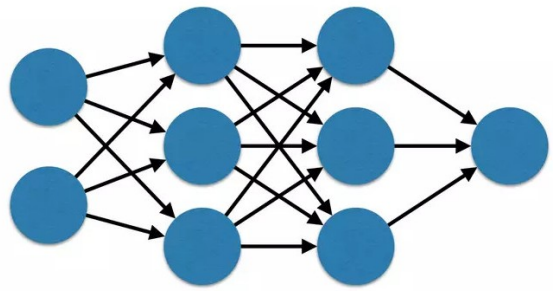
General Adversarial Networks

Introduction

GAN consists of two neural networks. One is called the generator, and the other is the discriminator.

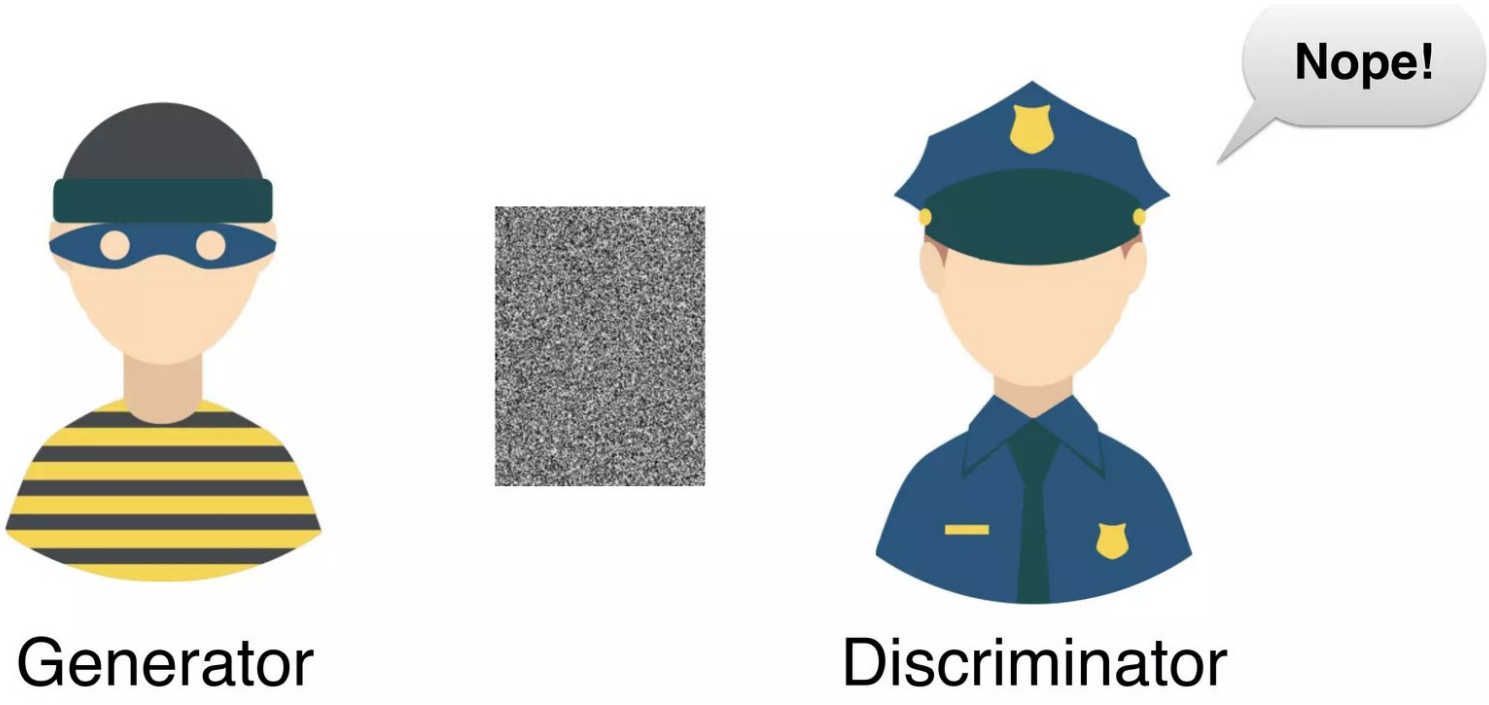


Generator

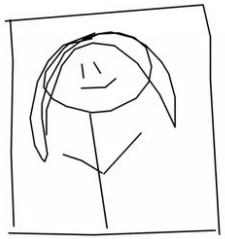


Discriminator

Introduction



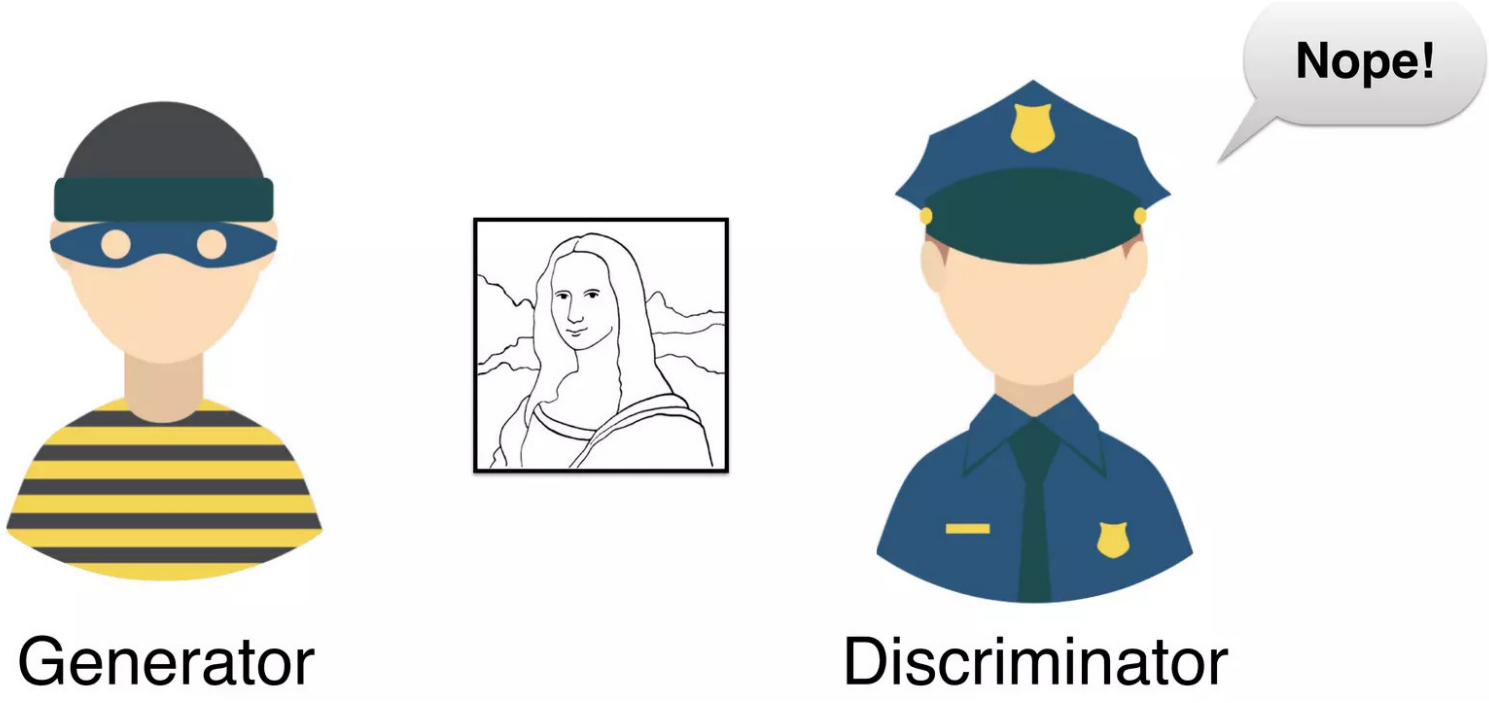
Introduction



Generator

Discriminator

Introduction



Introduction

GAN Block Architecture

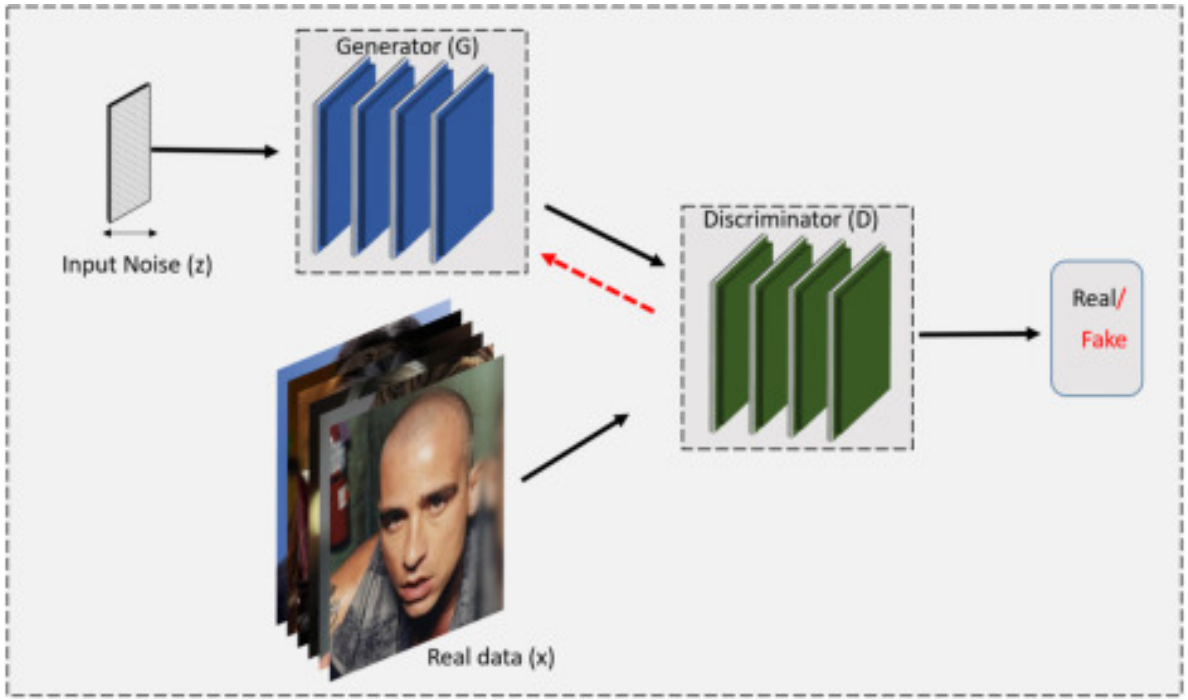
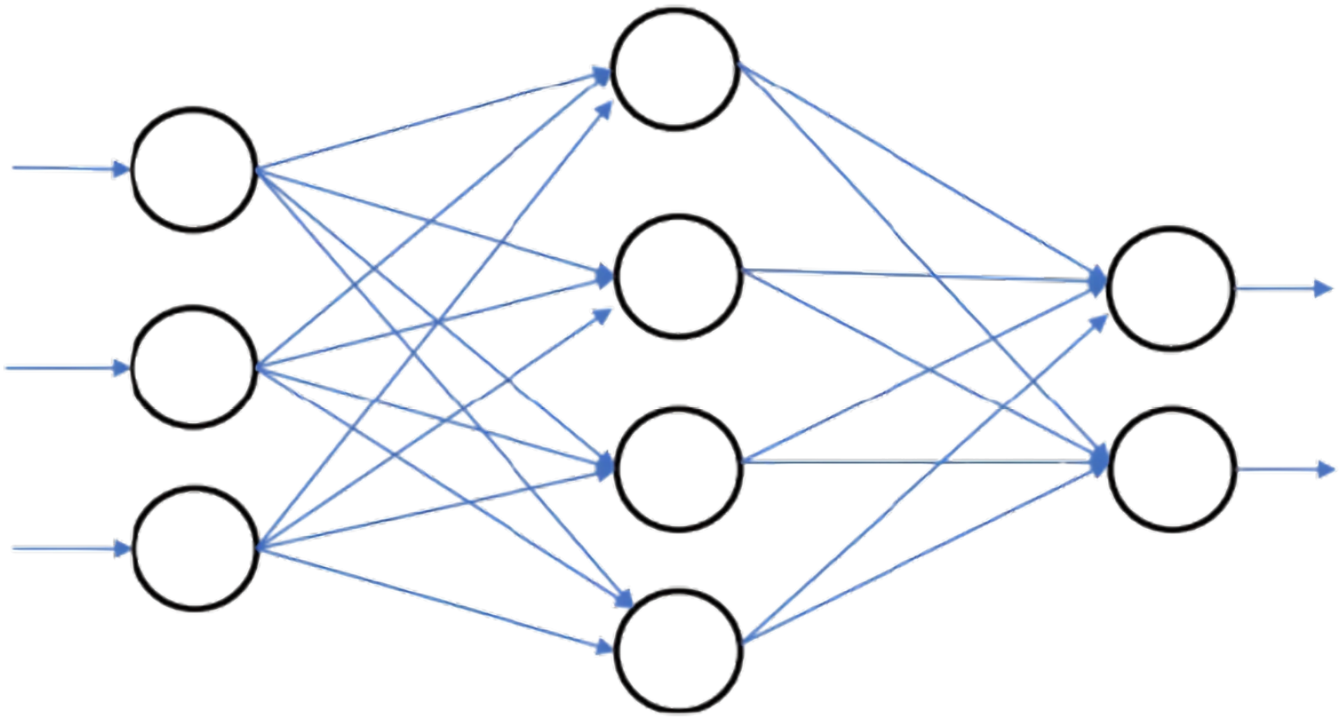
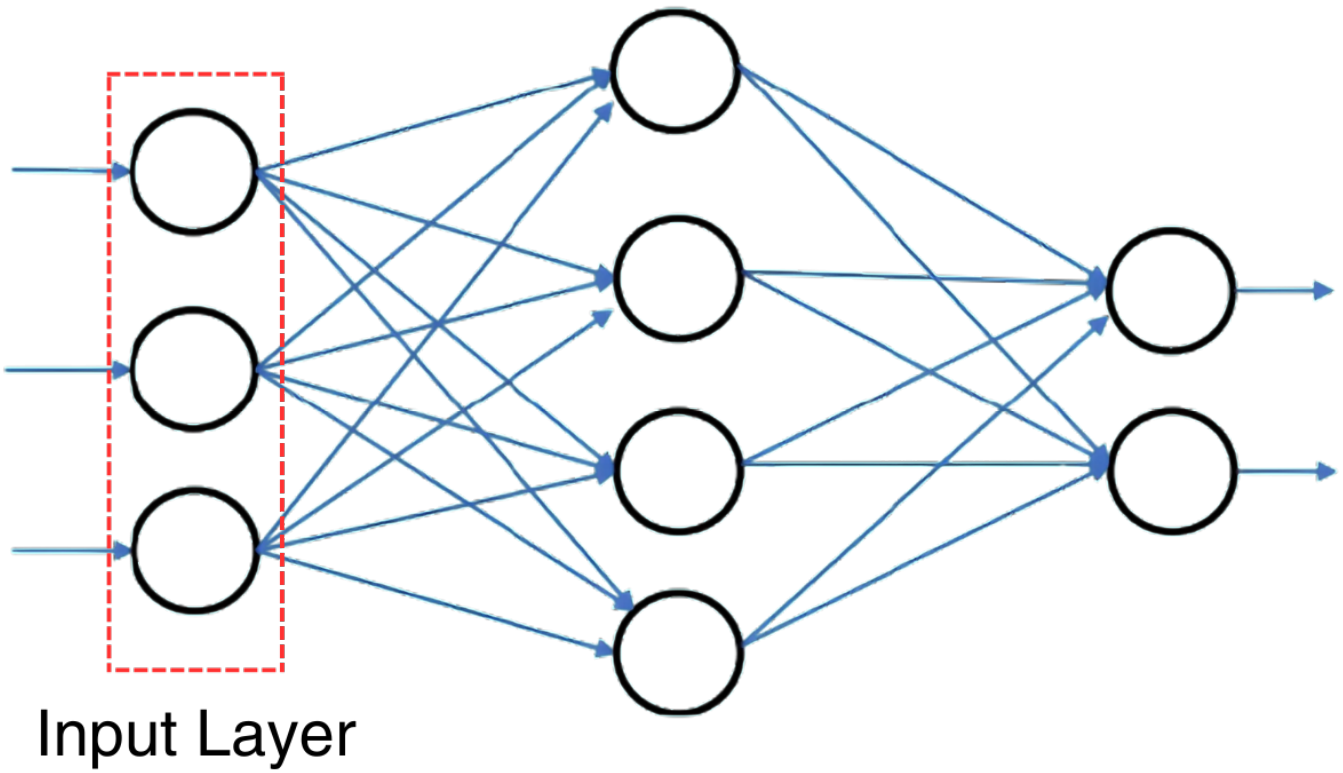


Figure 15: An example GAN block. The input of the generator (z) is sampled from a random noise vector and the input of the discriminator (x) is sampled from real data distribution.

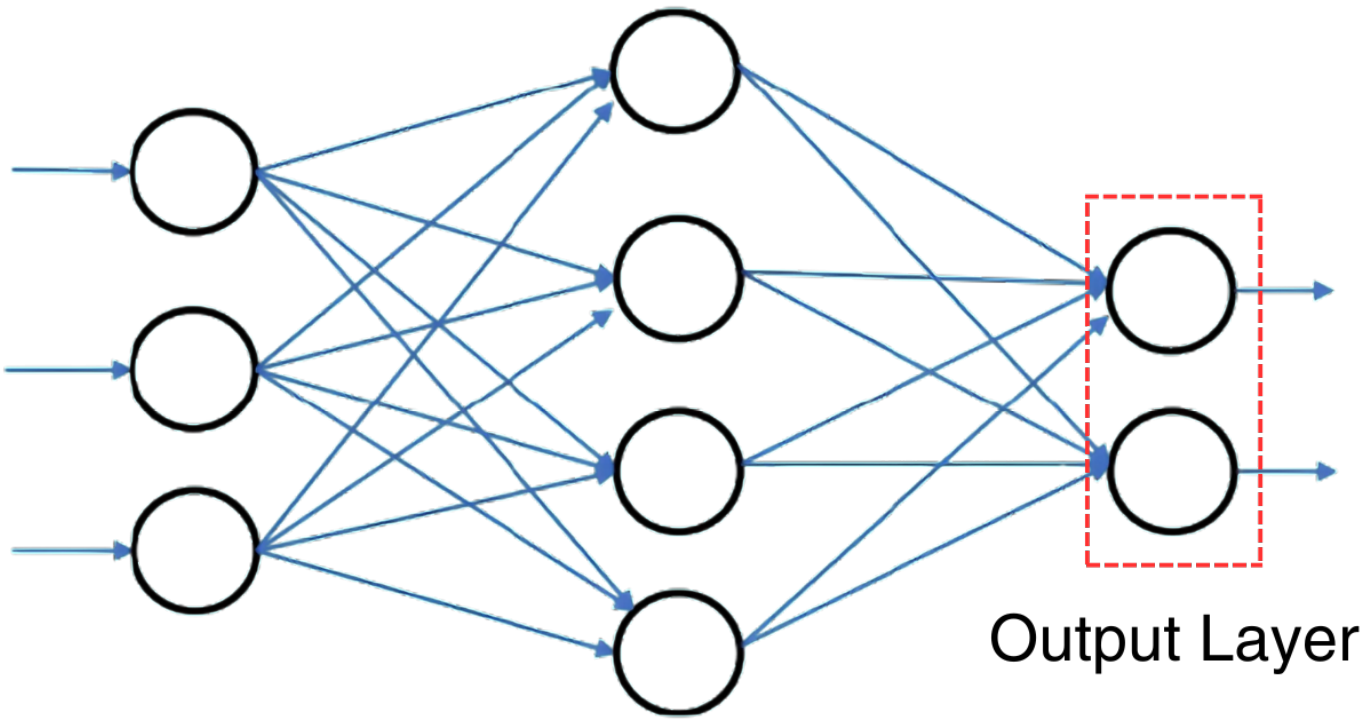
Neural Networks



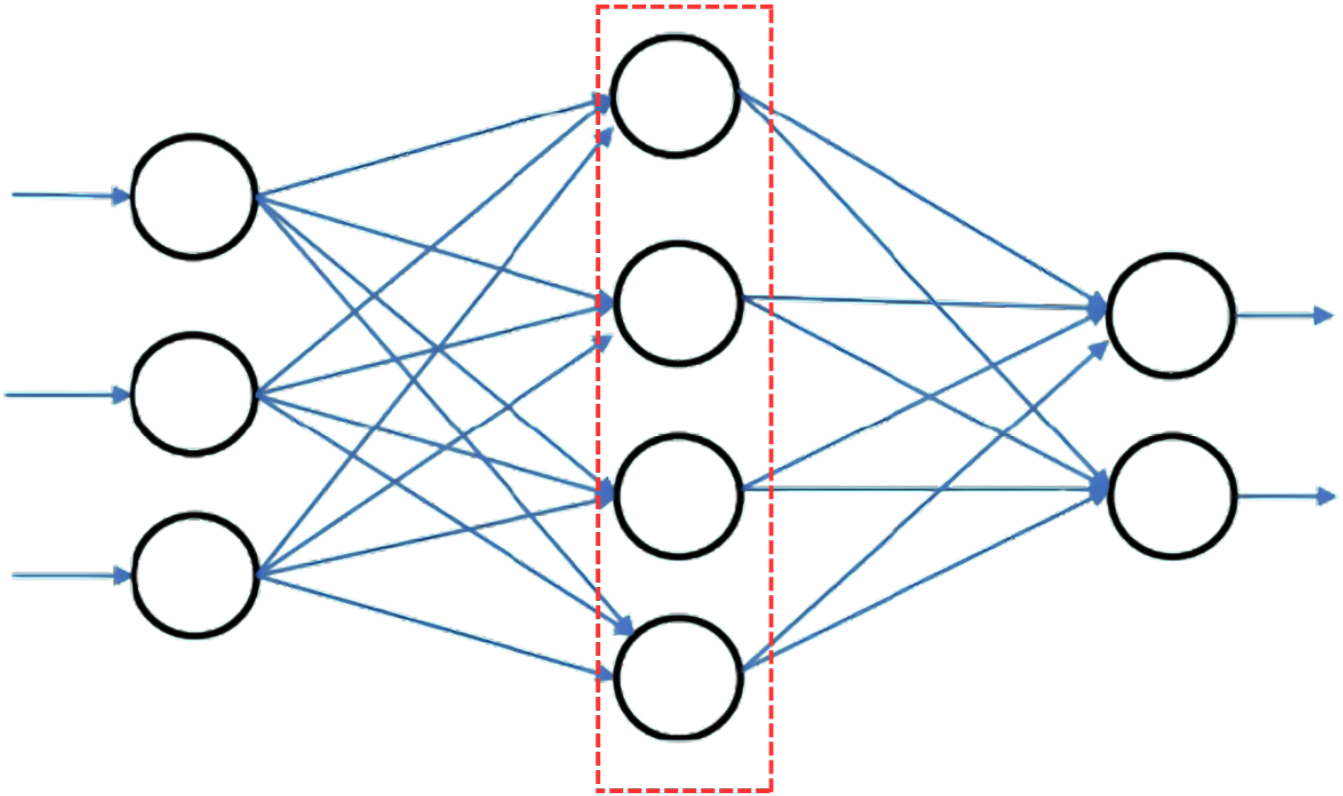
Neural Networks



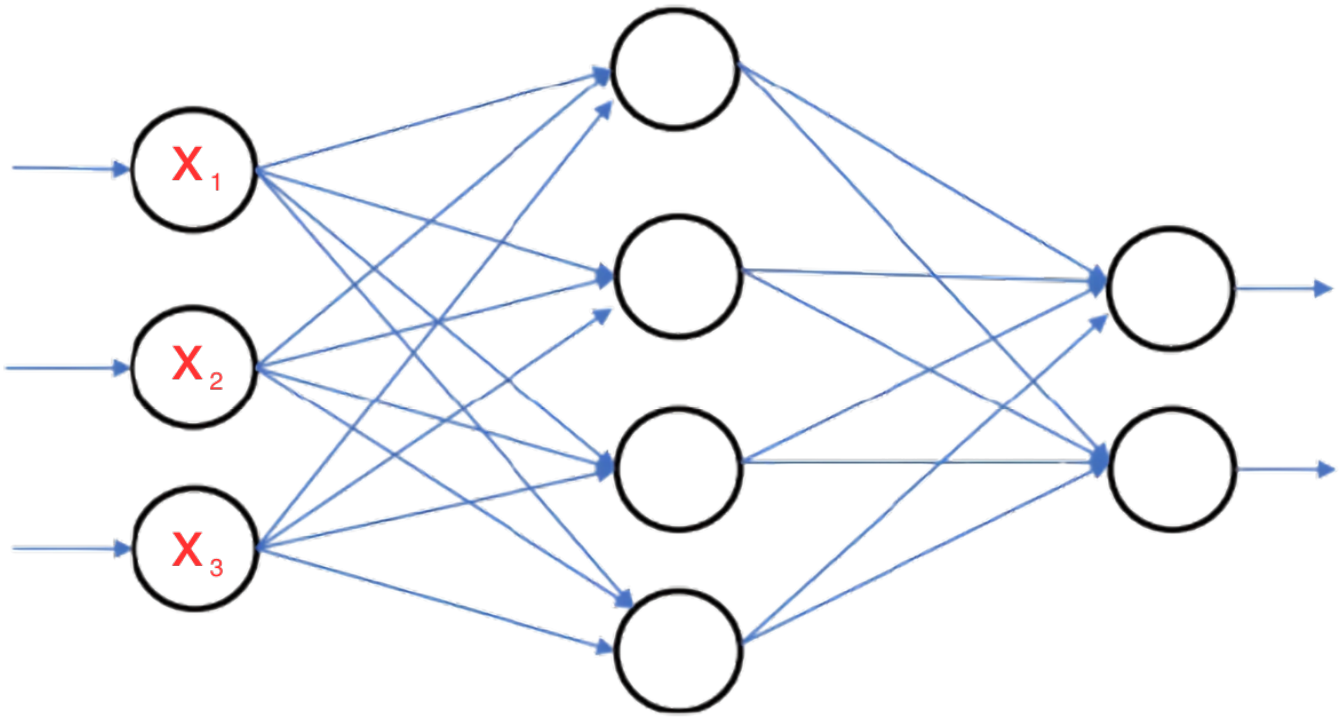
Neural Networks



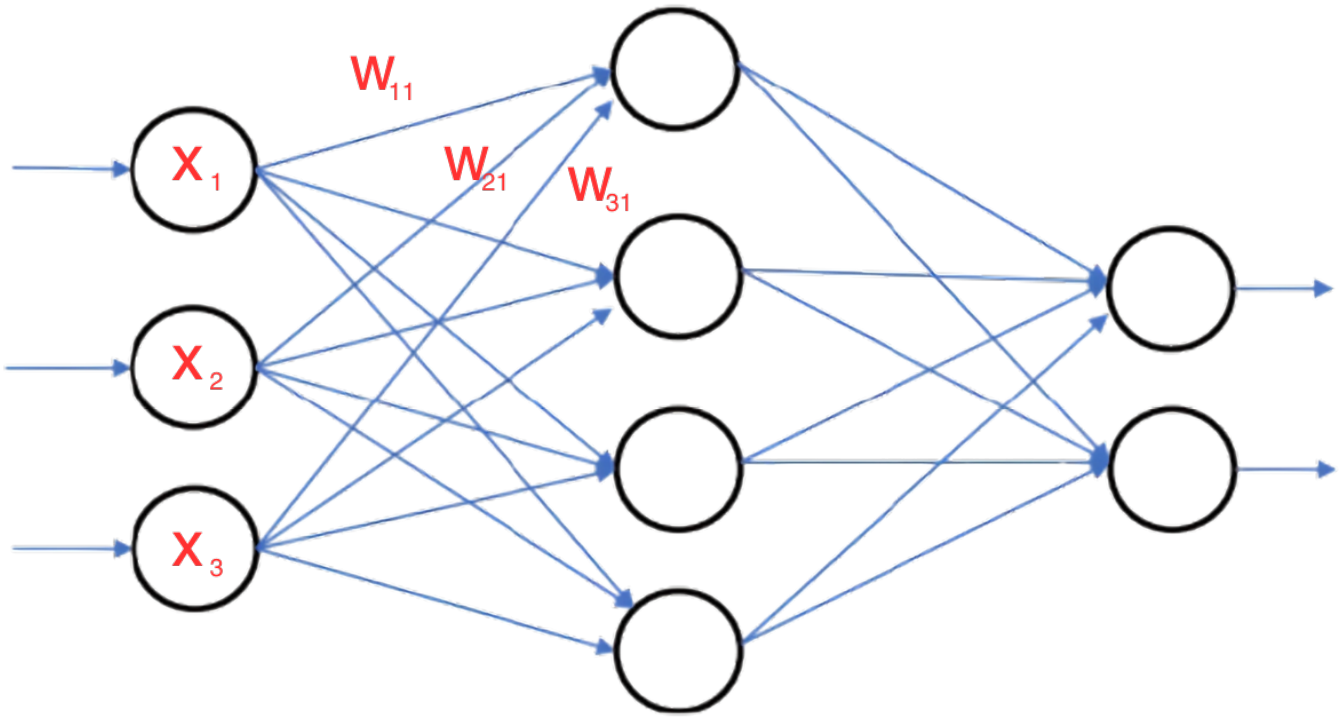
Neural Networks



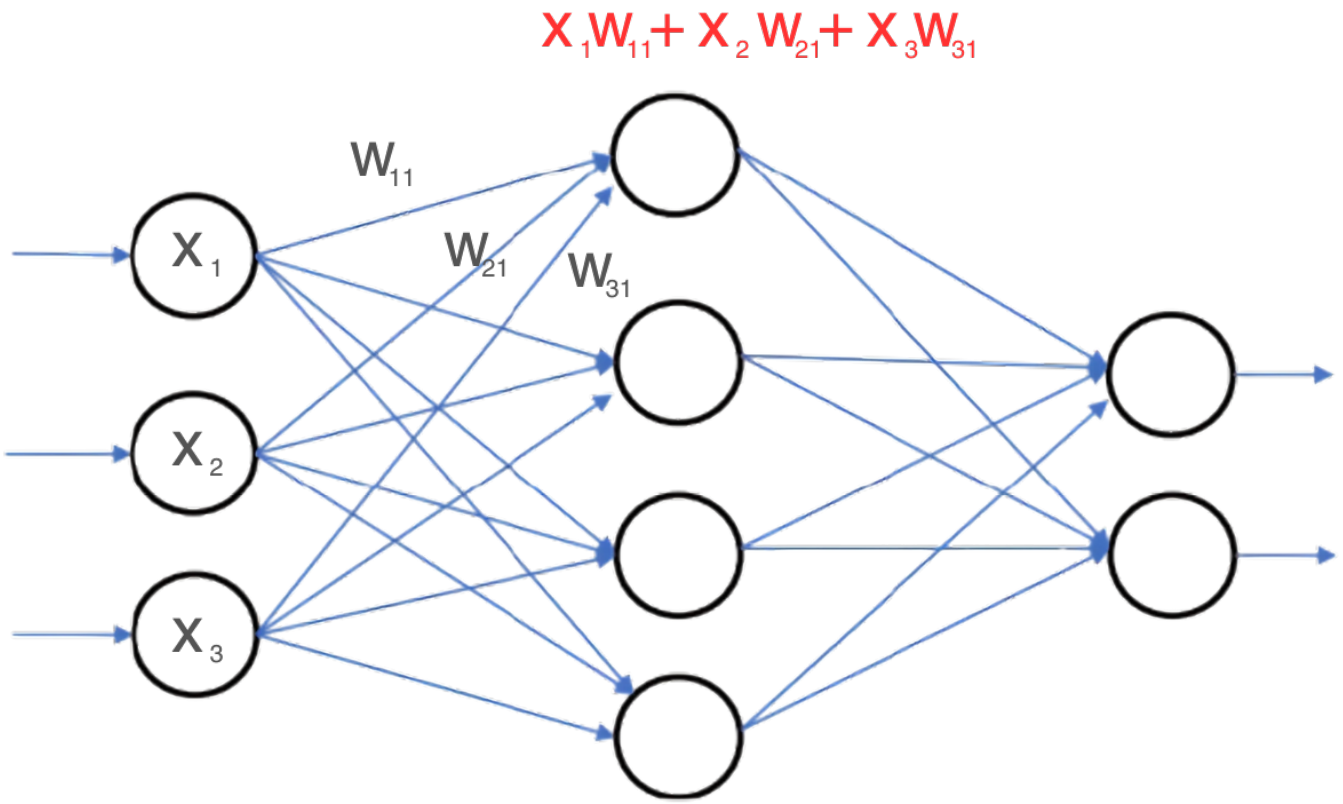
Neural Networks



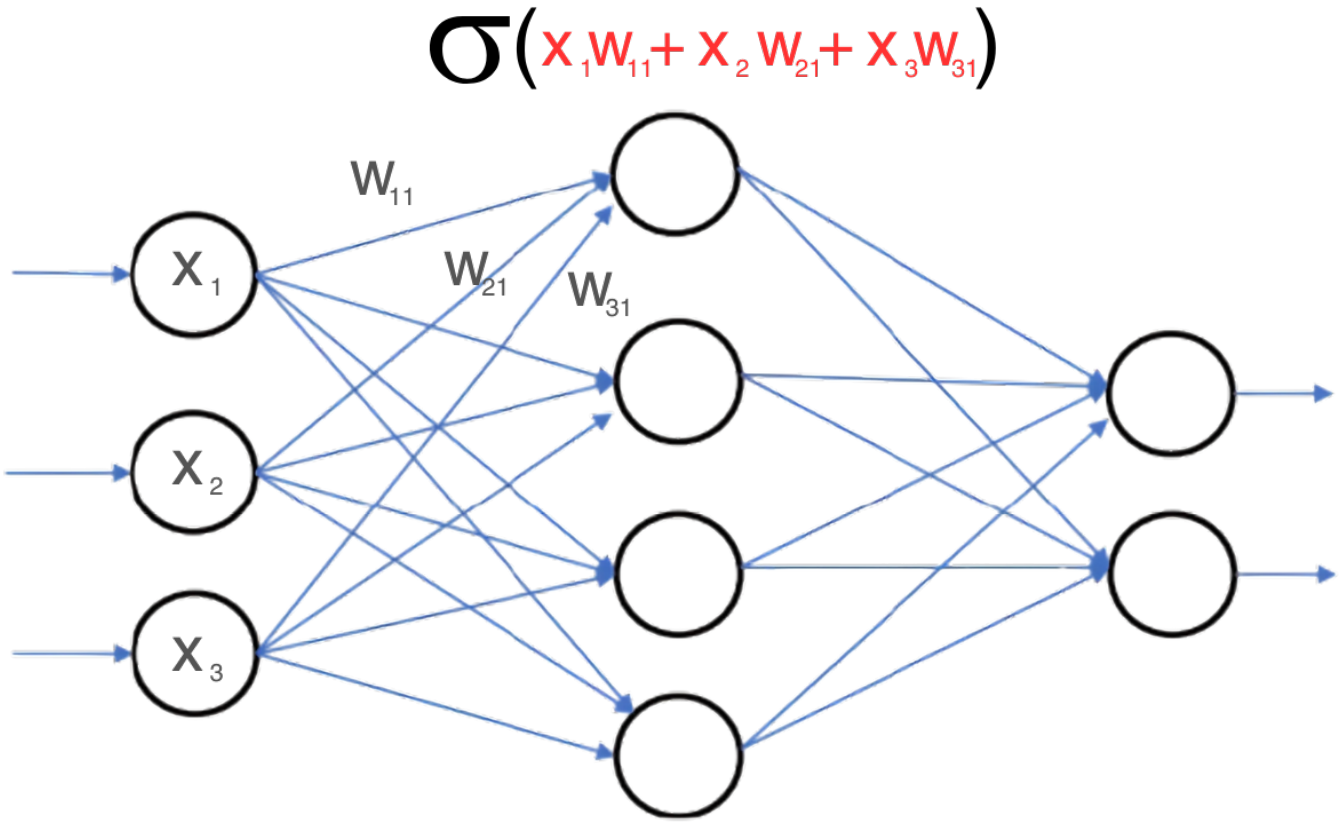
Neural Networks



Neural Networks

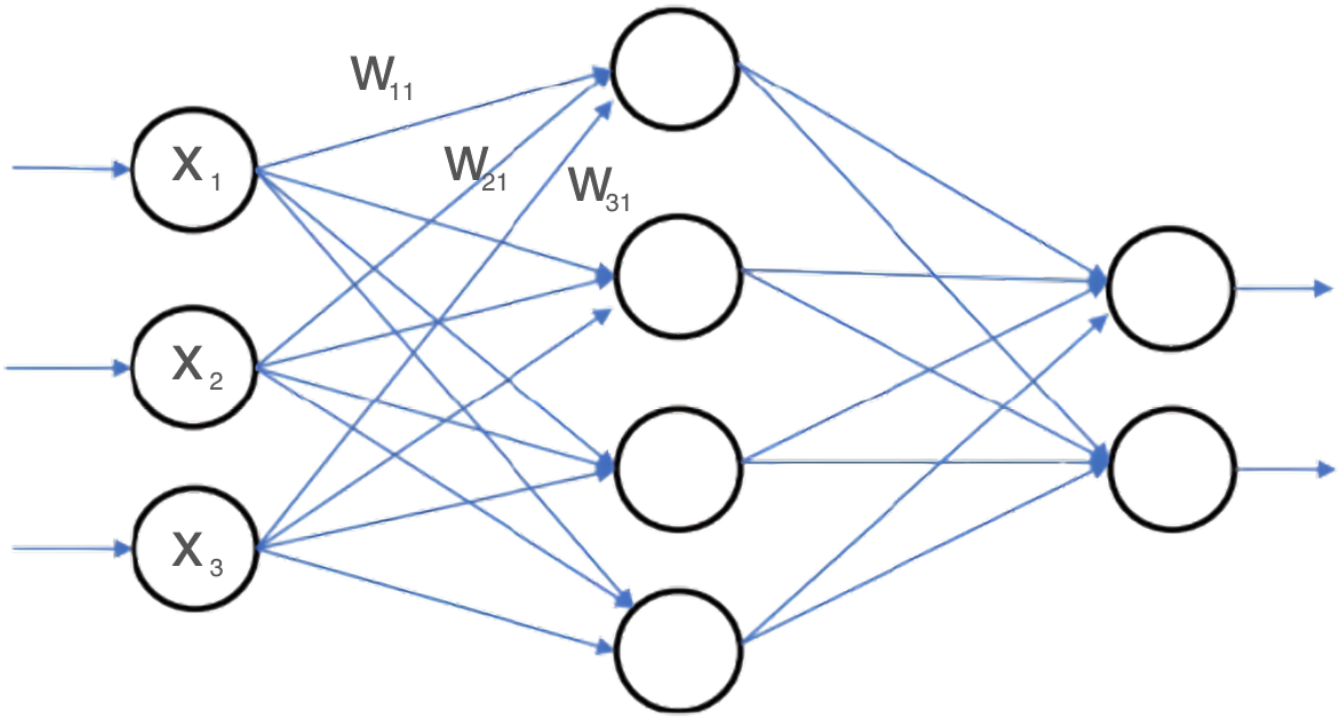


Neural Networks



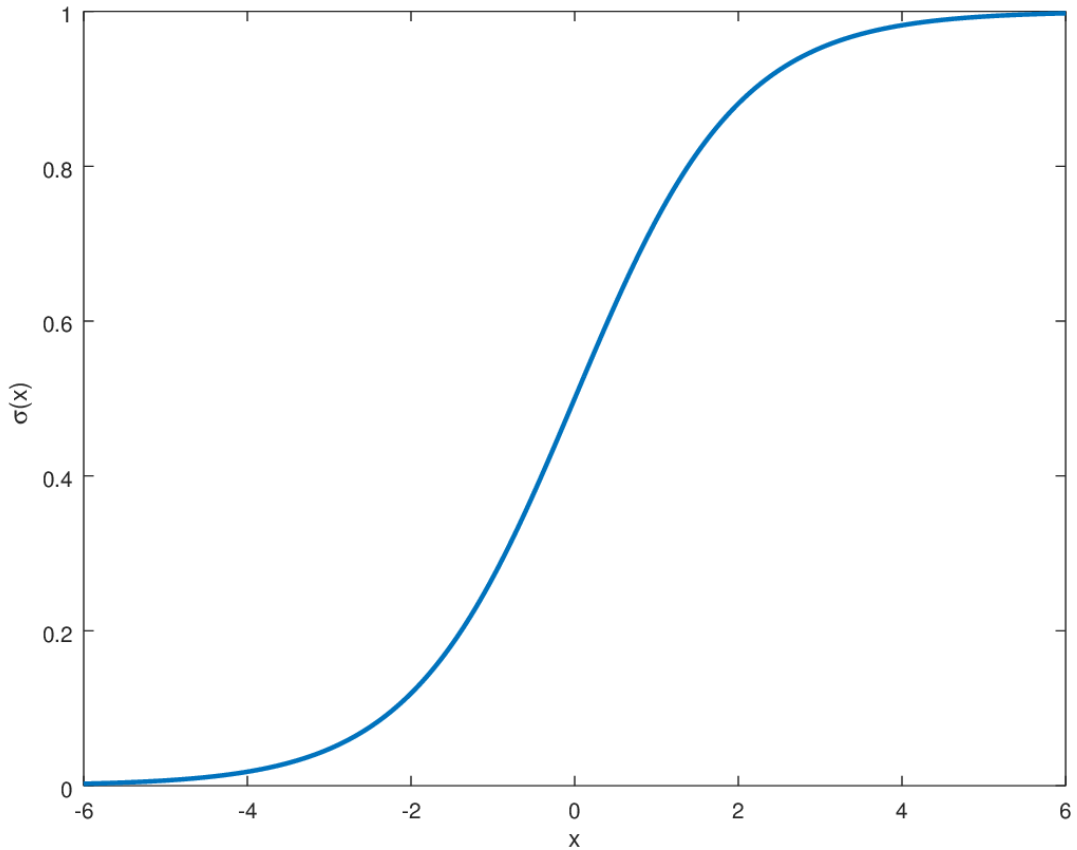
Neural Networks

$$\sigma(x_1 w_{11} + x_2 w_{21} + x_3 w_{31}) = y_1$$

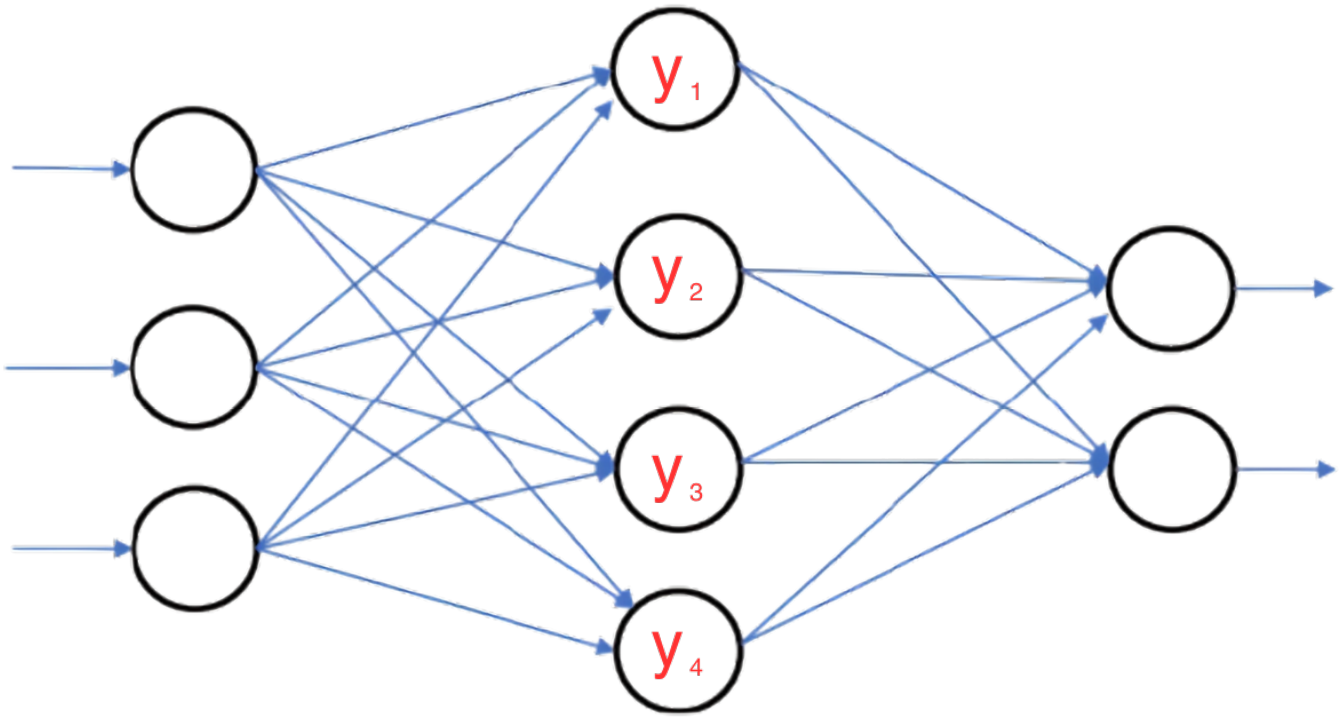


Neural Networks-Sigmoid Function

$$\sigma(w_{11}x_1 + w_{21}x_2 + w_{31}x_3) = \frac{1}{1 + e^{-(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)}}$$



Neural Networks



Neural Network Training

Neural Network Backpropagation

This algorithm computes gradients of the loss function with respect to each weight by applying the chain rule. For a neuron output $a = \sigma(z)$, where $z = \sum_j w_j y_j + b$, the gradient is:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial y} \cdot \frac{\partial y}{\partial w_i}$$

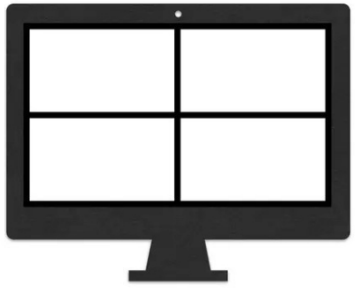
Once the gradients are calculated, the weights are updated using:

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w_i}$$

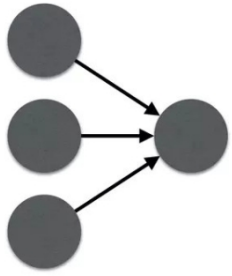
where η is the learning rate.

GAN Example - Slanted Land

Slanted people

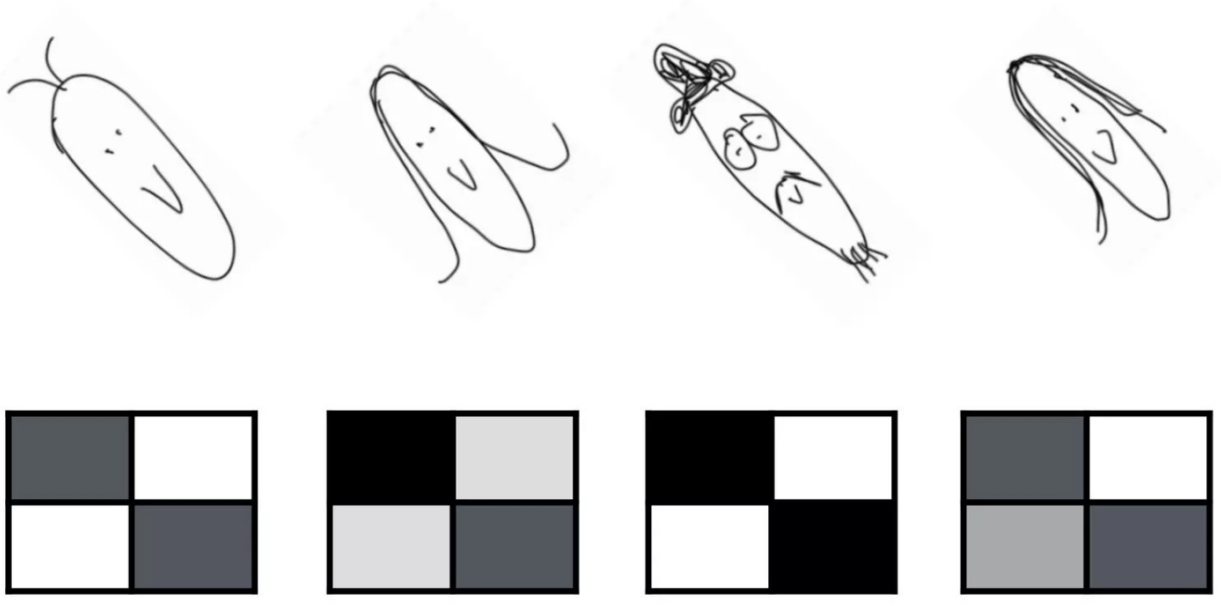


2x2 screens

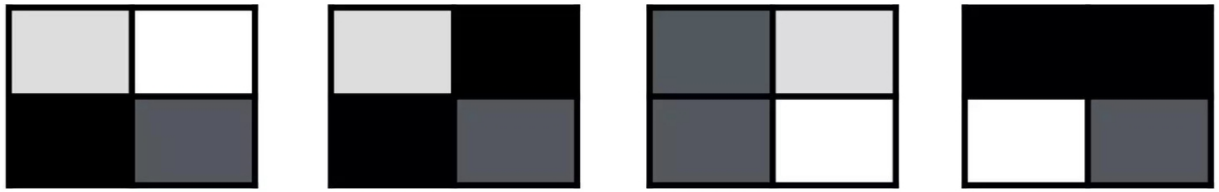


1-layer
Neural networks

Slanted Land - Faces



Slanted Land - Noises



Slanted Land - Faces and Noises

0 1

Faces

0.75	0
0	0.75

1	0.25
0.25	0.75

1	0
0	1

0.75	0
0.25	0.75

Noise

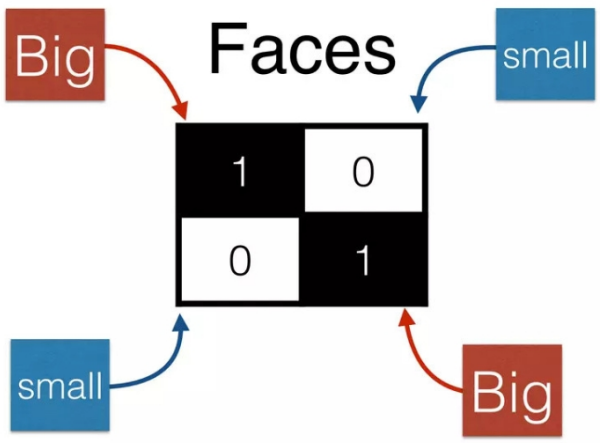
0.25	0
1	0.75

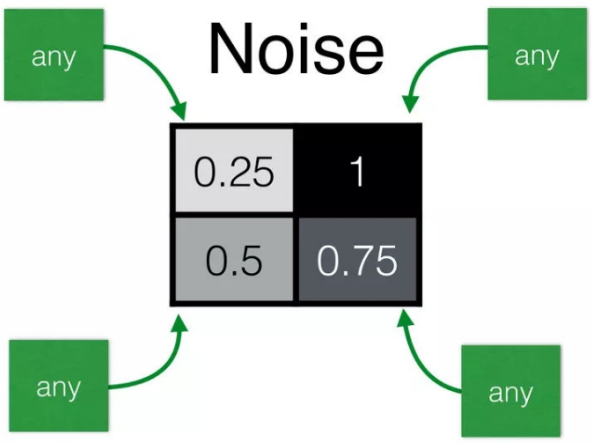
0.25	1
0.5	0.75

0.75	0.5
0.75	0

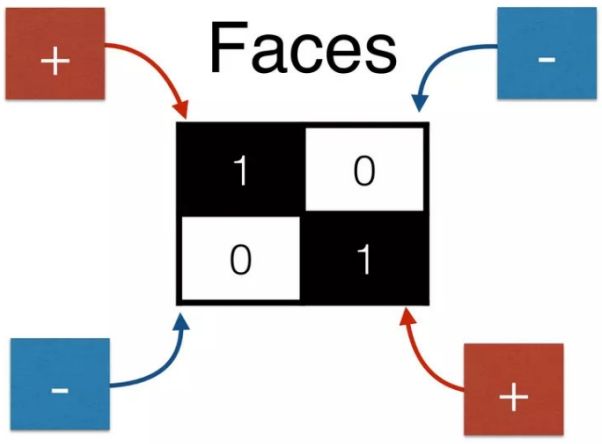
1	1
0	0.75

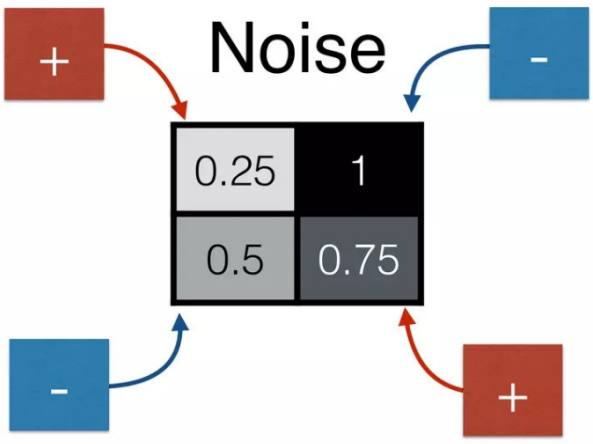
Slanted Land - Faces and Noises





Building Discriminator





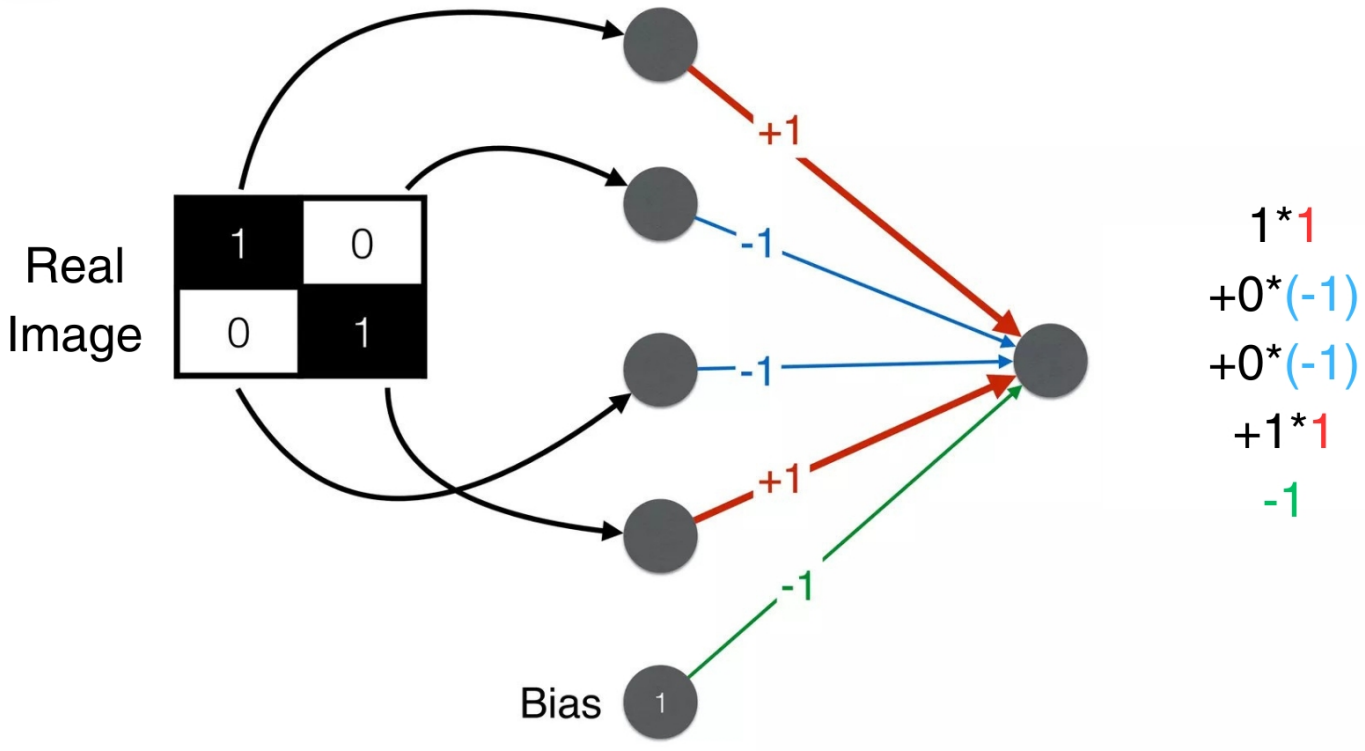
$1*1 + 0*(-1) + 0*(-1) + 1*1 = 2$

$0.25*1 + 1*(-1) + 0.5*(-1) + 0.75*1 = -0.5$

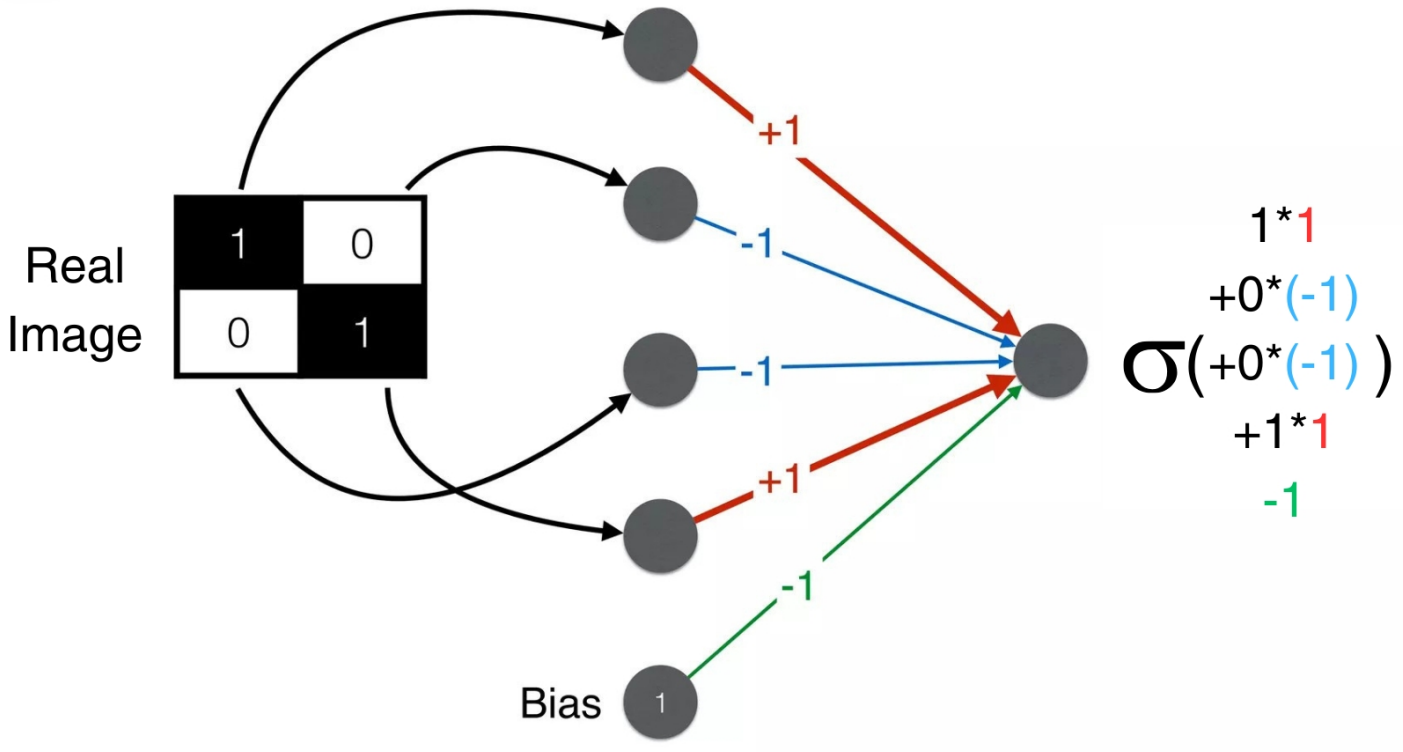
Threshold = 1

More than 1: face. Less than 1: no face

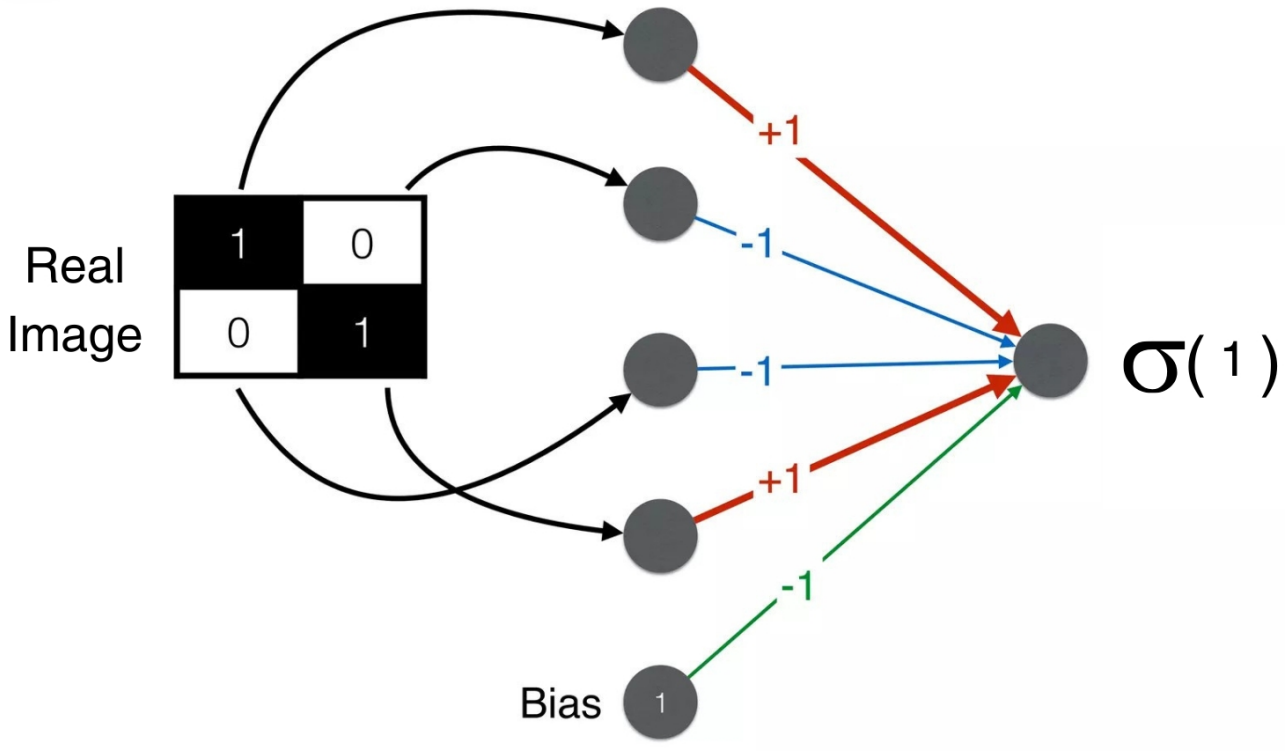
Building Discriminator



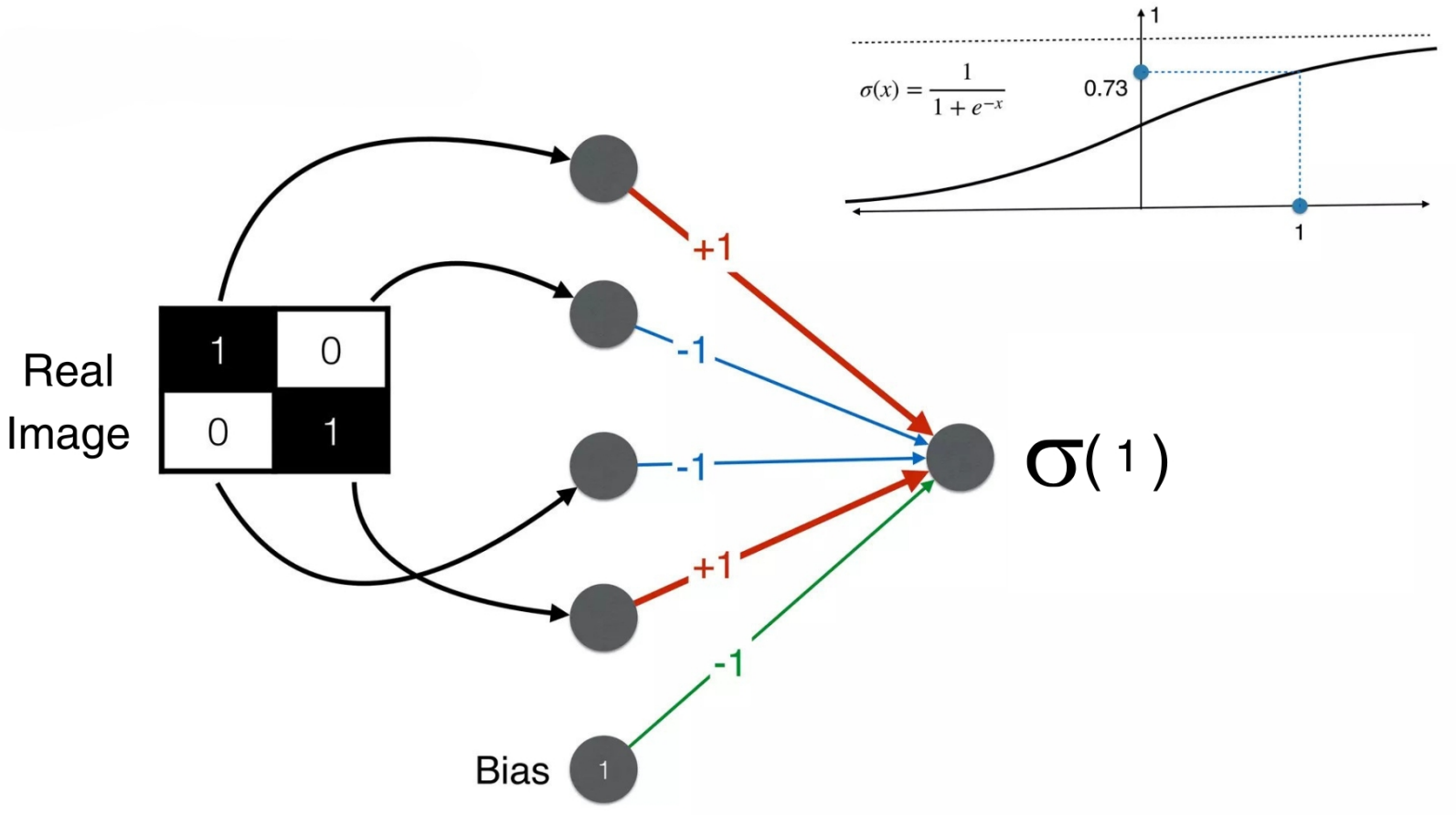
Building Discriminator



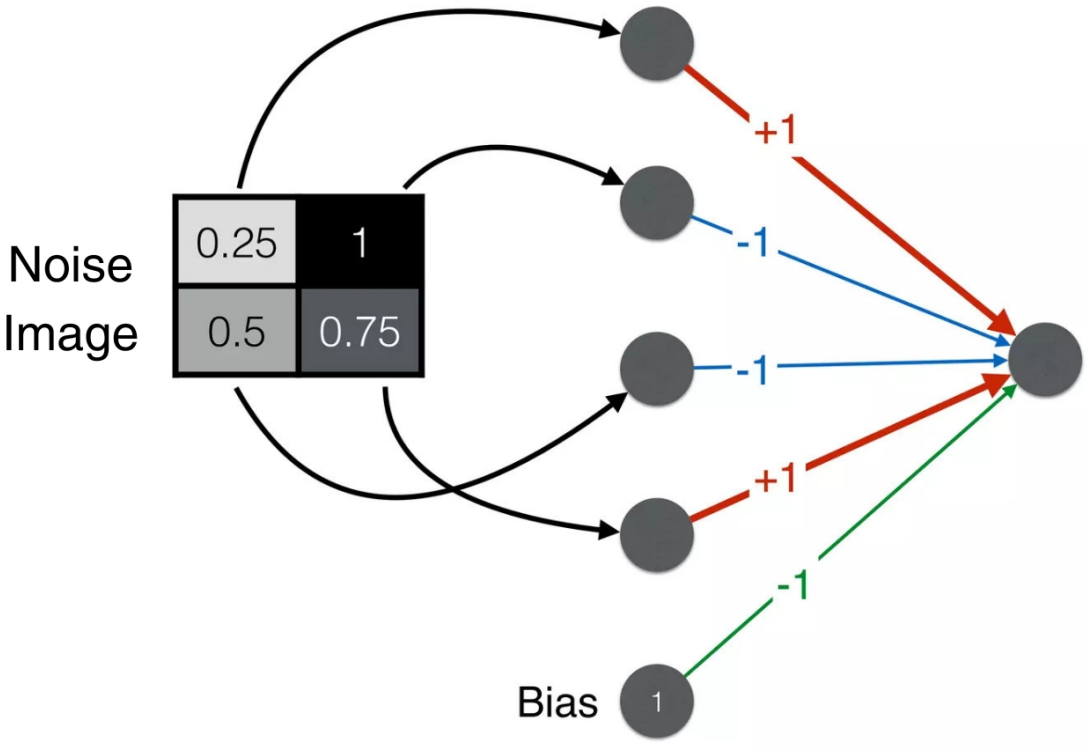
Building Discriminator



Building Discriminator

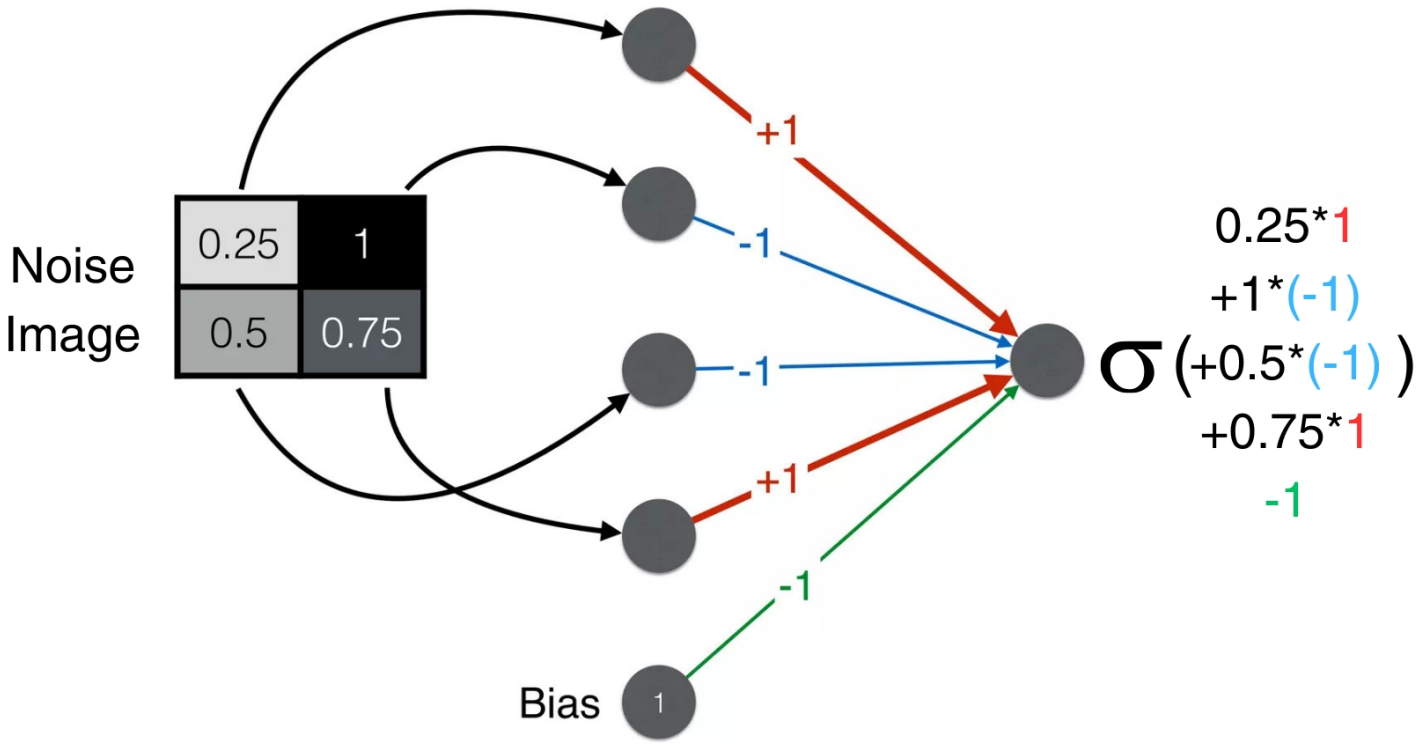


Building Discriminator

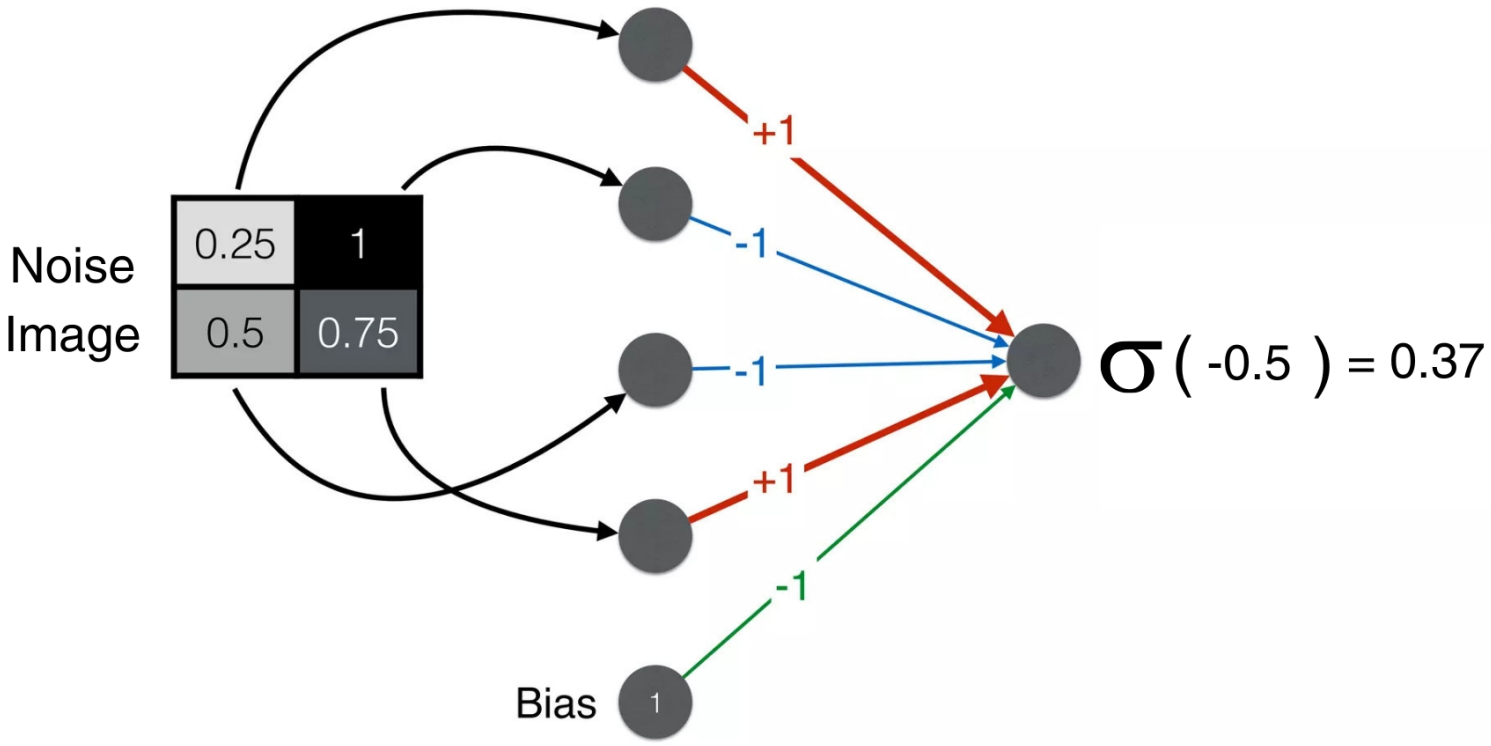


0.25*1
+1*(-1)
+0.5*(-1)
+0.75*1
-1

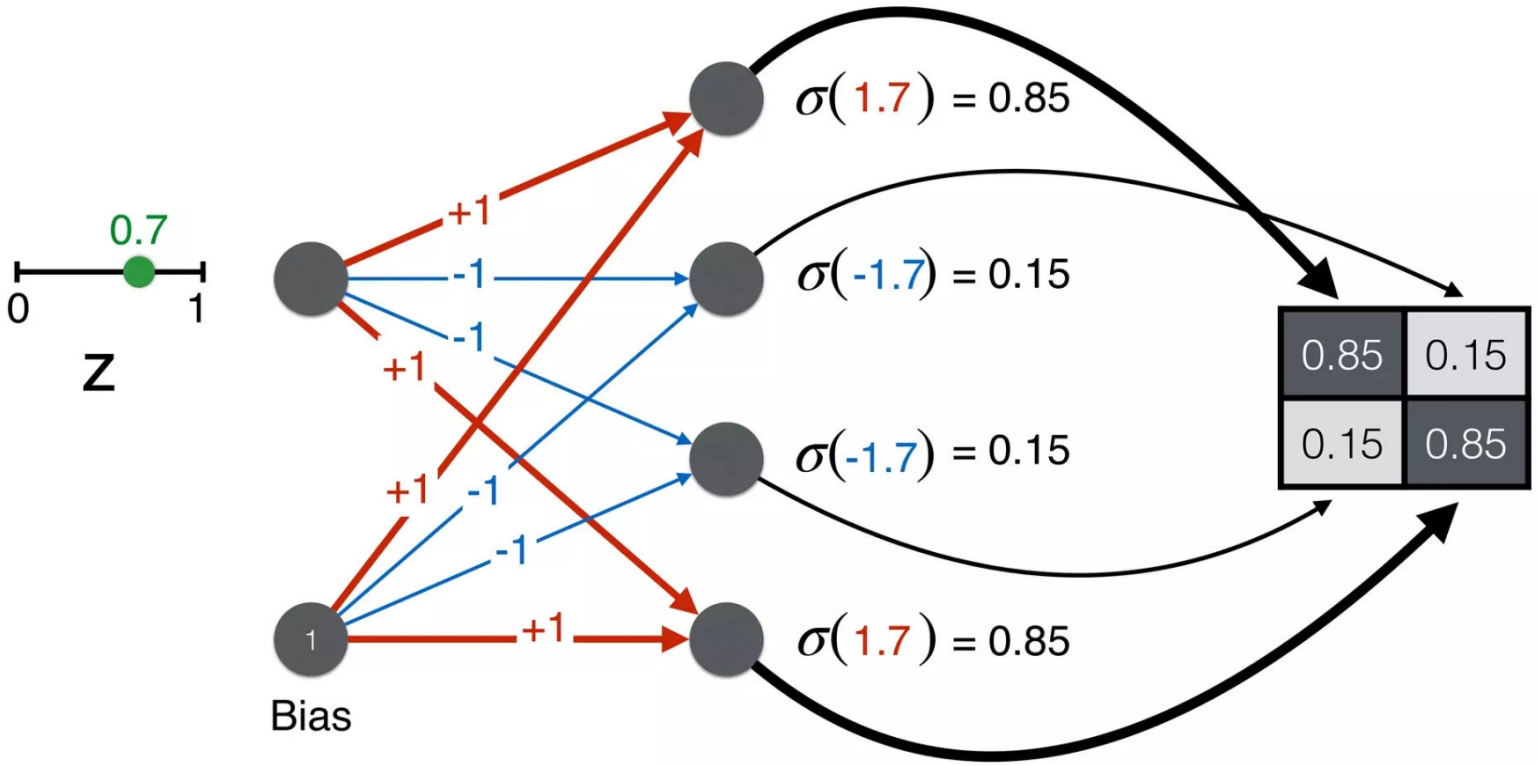
Building Discriminator



Building Discriminator



Building Generator



Log Loss Error

Label: 1
Prediction: 0.1

Log Loss Error

Label: 1

Prediction: 0.1

Error: large

Log Loss Error

Label: 1
Prediction: 0.1

Error: large

Label: 1
Prediction: 0.9

Log Loss Error

$$\text{Error} = -\ln(\text{prediction})$$

Label: 1
Prediction: 0.1 Error: large

Label: 1
Prediction: 0.9 Error: small

Log Loss Error

$$\text{Error} = -\ln(\text{prediction})$$

Label: 1
Prediction: 0.1 Error: large $-\ln(0.1) = 2.3$

Label: 1
Prediction: 0.9 Error: small $-\ln(0.9) = 0.1$

Log Loss Error

Label: 0
Prediction: 0.1

Log Loss Error

Label: 0

Prediction: 0.1

Error: small

Log Loss Error

Label: 0
Prediction: 0.1

Error: small

Label: 0
Prediction: 0.9

Log Loss Error

$$\text{Error} = -\ln(1 - \text{prediction})$$

Label: 0
Prediction: 0.1 Error: small

Label: 0
Prediction: 0.9 Error: large

Log Loss Error

$$\text{Error} = -\ln(1 - \text{prediction})$$

Label: 0
Prediction: 0.1 Error: small $-\ln(0.9) = 0.1$

Label: 0
Prediction: 0.9 Error: large $-\ln(0.1) = 2.3$

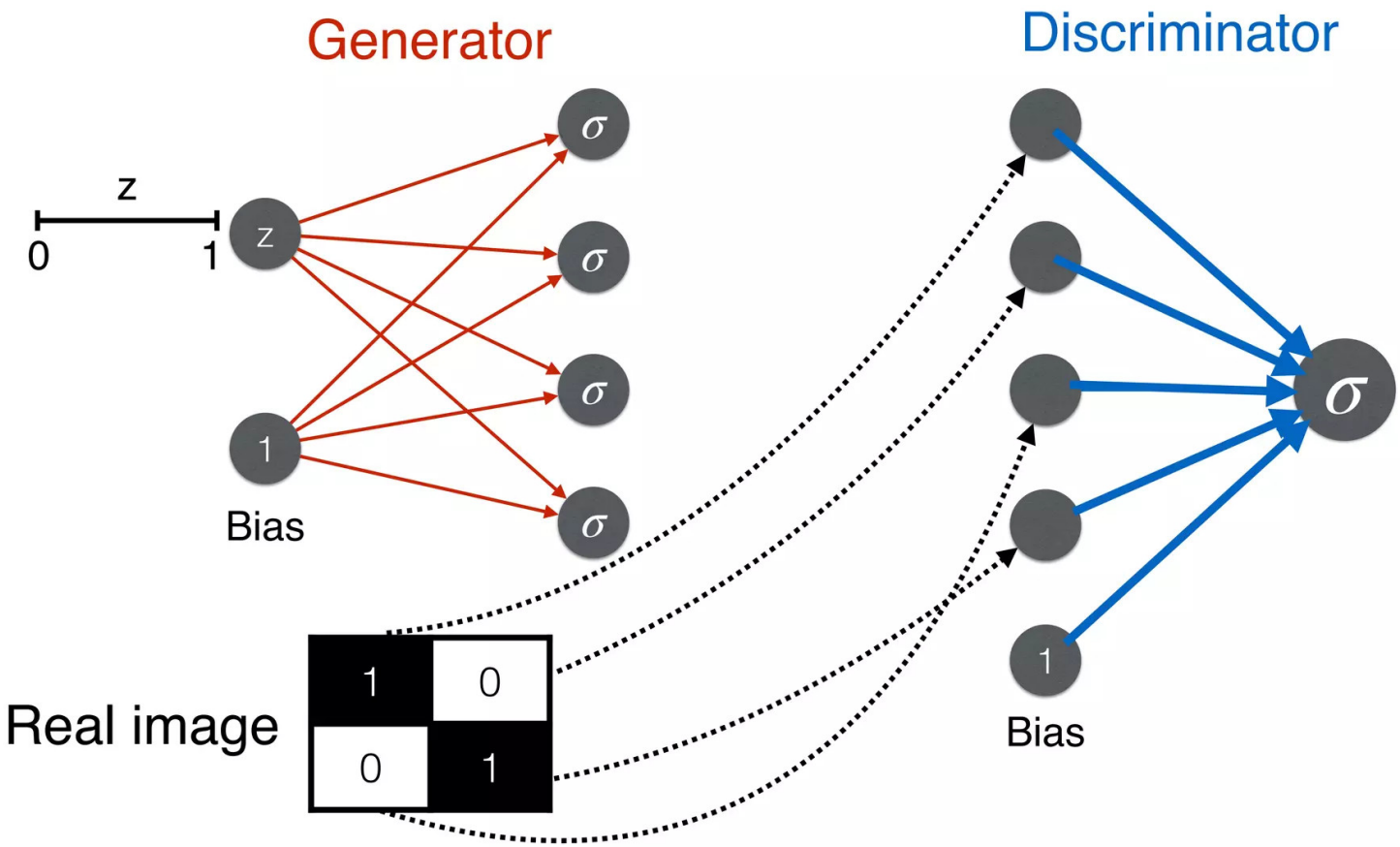
GAN Objective Function

GAN Objective Function

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))].$$

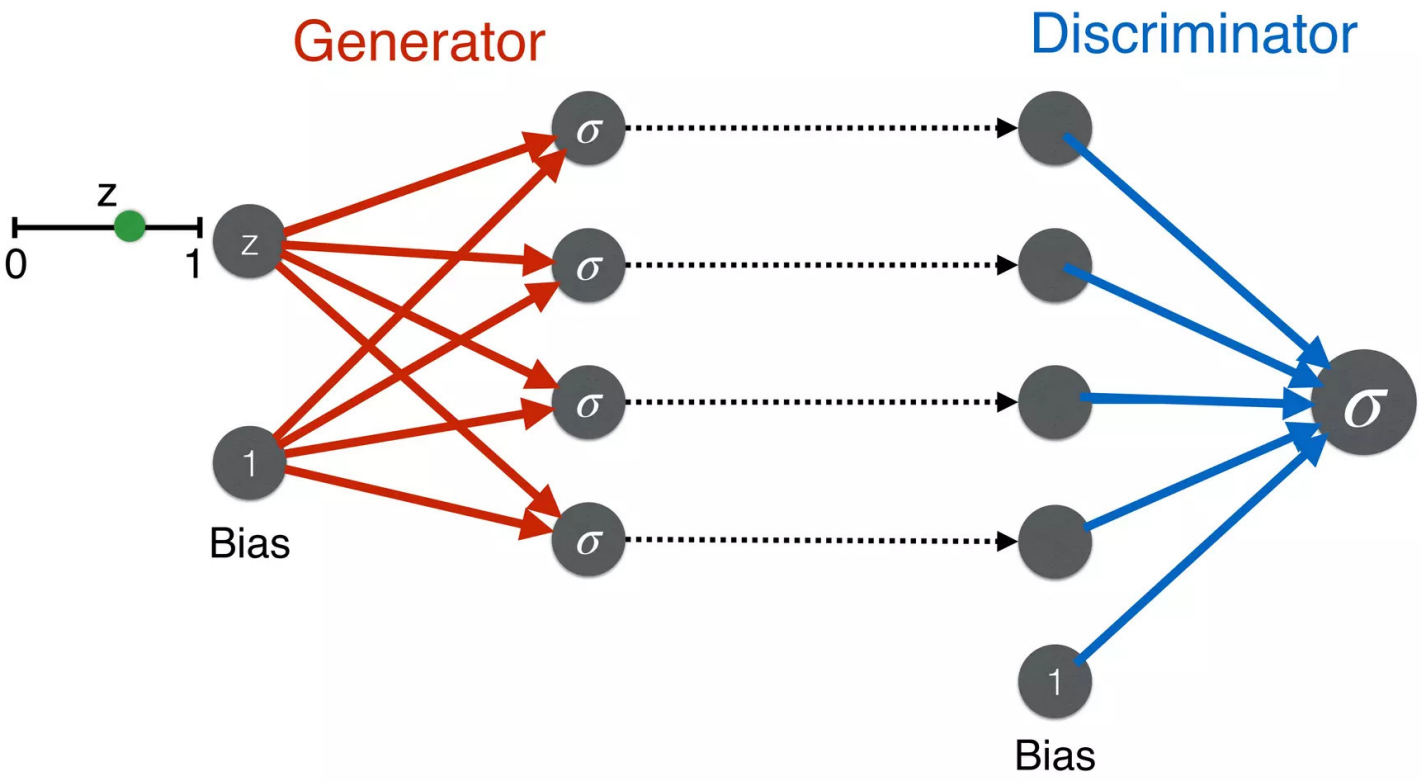
Loss Function Optimization

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})]$$



Loss Function Optimization

$$\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$



Results

Figure 16: Original Image of Dwaipayan Haldar

Figure 17: Image Distorted by Manas Sarkar

Figure 18: Output Image of Dwaipayan Haldar restored by GAN

Results

Figure 19: Original Ghibli Image of Dwaipayan Haldar

Figure 20: Ghibli Image Distorted by Pramit Khamrui

Figure 21: Output Ghibli Image of Dwaipayan Haldar restored by GAN

Conclusion

References

- Jireh Jam, Connah Kendrick, Kevin Walker, Vincent Drouard, Jison Gee-Sern Hsu, Moi Hoon Yap, A comprehensive review of past and present image inpainting methods, Computer Vision and Image Understanding, Volume 203, 2021.
- Efros, A.A., Leung, T.K., 1999. Texture synthesis by non-parametric sampling. In: Iccv. IEEE, p. 1033.
- Texture Synthesis by Non-parametric Sampling https://people.eecs.berkeley.edu/ efros/research/NPS/alg.html
- Thottam, I., 2015. The Cost of Conservation and Restoration. Art Business News, http://artbusinessnews.com/2015/12/the-cost-of-conservation-and-restoration/
- Liu, G., Reda, F.A., Shih, K.J., Wang, T.-C., Tao, A., Catanzaro, B., 2018a. Image inpainting for irregular holes using partial convolutions. arXiv preprint arXiv: 1804.07723.
- Serrano, L. (2019, November 11). Generative adversarial networks (GANs) [SlideShare presentation]. SlideShare. https://www.slideshare.net/slideshow/generative-adversarial-networks-gans-238846644/238846644
- Criminisi, A., Pérez, P., Toyama, K., 2004. Region filling and object removal by exemplar-based image inpainting. IEEE Trans. Image Process. 13 (9), 1200–1212.

Thank You!

Questions?