GENERATIVE ADVERSARIAL NETWORKS

Deep Neural Networks

Session 23

Pramod Sharma
pramod.sharma@prasami.com

2

Agenda

- PixelRNN / CNN
- Variational Autoencoder
- GAN

6/1/2024

pra-sâmi

3

Technologies with Conflicting Goals

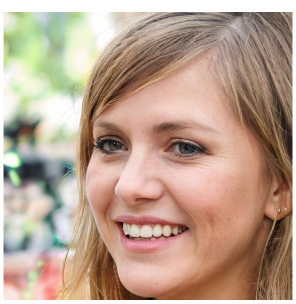
Image based CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a type of **security measure** known as challenge-response authentication

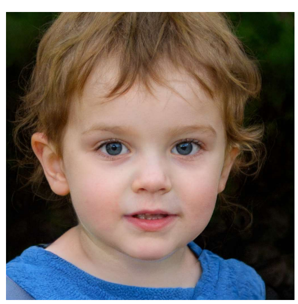
Generative Adversarial Networks

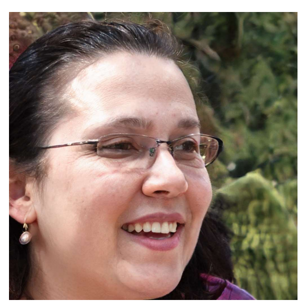
4

Unbelievably Real

- This person does not exist : thispersondoesnotexist.com







5

Overview

- Developed by Ian Goodfellow
- In generative modeling, we'd like to train a network that models a distribution,
  - Such as a distribution over images.
- One way to judge the quality of the model is to sample from it
- Active area of research with rapid progress

2009

2015

2018

pra-sâmi

6

Take a Step Back

Supervised Learning

- Data: (x, y)
  - x is data, y is label
- Goal: Learn a function to map x -> y
- Examples:
  - Classification,
  - Regression,
  - Object detection,
  - Semantic segmentation,
  - Image captioning,
  - …

Unsupervised Learning

- Data: x
  - Just data, no labels!
- Goal: Learn some underlying hidden structure of the data
- Examples:
  - Clustering,
  - Dimensionality reduction,
  - Feature learning,
  - Density estimation,
  - …

Makes Training data cheap!

Holy grail:
Solve unsupervised learning
➔ Understand structure of visual world

pra-sâmi

7

Generative Models

- Given the training data, generate new samples from same distribution

Training data ~ $p_{data}(x)$

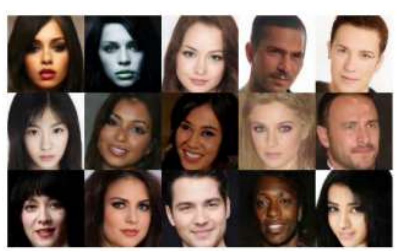
Generated samples ~ $p_{model}(x)$

- Want to learn $p_{model}(x)$ similar to $p_{data}(x)$
- Addresses density estimation, a core problem in unsupervised learning
- Several flavors:
  - Explicit density estimation: explicitly define and solve for $p_{model}(x)$
  - Implicit density estimation: learn model that can sample from $p_{model}(x)$ without explicitly defining it

8

Why Generative Models?

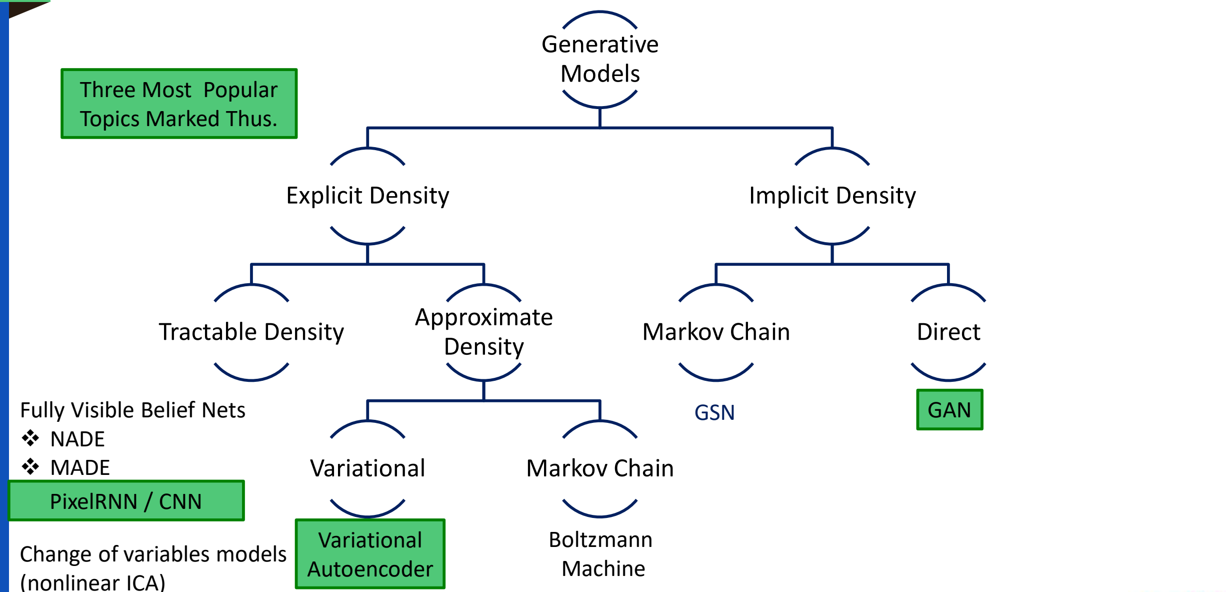
- Realistic samples for artwork, super-resolution, colorization, etc.



- Generative models of time-series data can be used for simulation and planning
  - Such as reinforcement learning applications!
- Training generative models can also enable inference of latent representations that can be useful as general features

9

Taxonomy of Generative Models



6/1/2024

Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

pra-sâmi

10

Fully Visible Belief Network

- Explicit Density Model
- Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, x_2, x_3 \ldots x_{i-1})$$

Likelihood of image x

Probability of ith pixel value given all previous pixels

- Then maximize likelihood of training data
- Note:
  - Will need to define ordering of "previous pixels"
  - Complex distribution over pixel values ➔ Express using a neural network!

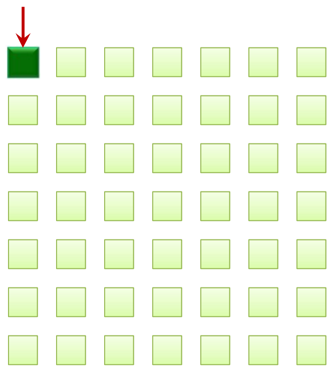
6/1/2024

pra-sâmi

11

PixelRNN [van der Oord et al. 2016]

- Generate image pixels starting from corner
- Dependency on previous pixels modeled using an RNN (LSTM)

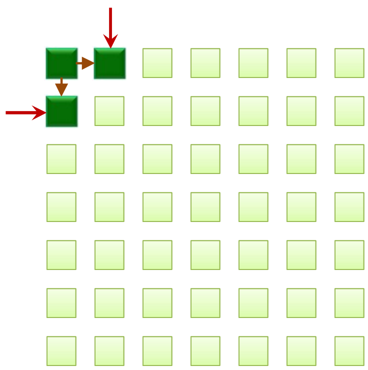


pra-sâmi

12

PixelRNN [van der Oord et al. 2016]

- Generate image pixels starting from corner
- Dependency on previous pixels modeled using an RNN (LSTM)

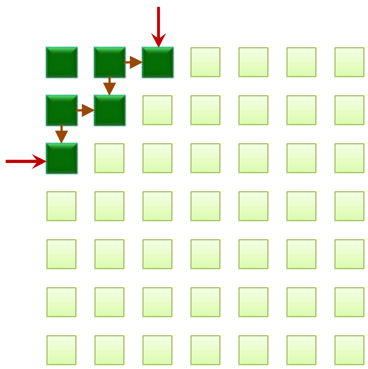


pra-sâmi

13

PixelRNN [van der Oord et al. 2016]

- Generate image pixels starting from corner
- Dependency on previous pixels modeled using an RNN (LSTM)

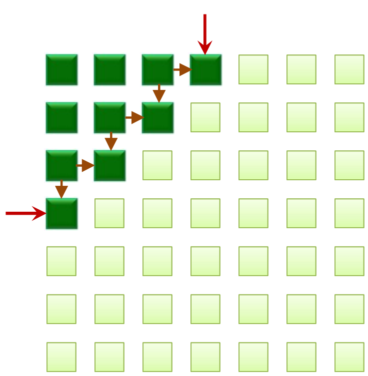


pra-sâmi

14

PixelRNN [van der Oord et al. 2016]

- Generate image pixels starting from corner
- Dependency on previous pixels modeled using an RNN (LSTM)

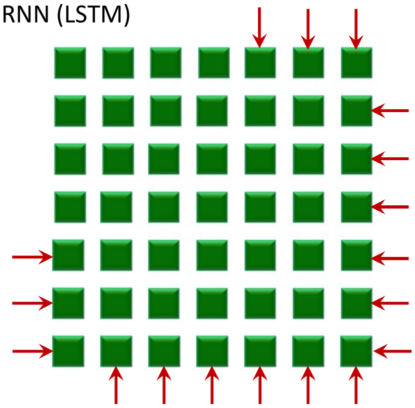


pra-sâmi

PixelRNN [van der Oord et al. 2016]

15

- Generate image pixels starting from corner
- Dependency on previous pixels modeled using an RNN (LSTM)



- Drawback:
  - Very sequential generation, very slow!

6/1/2024

pra-sâmi

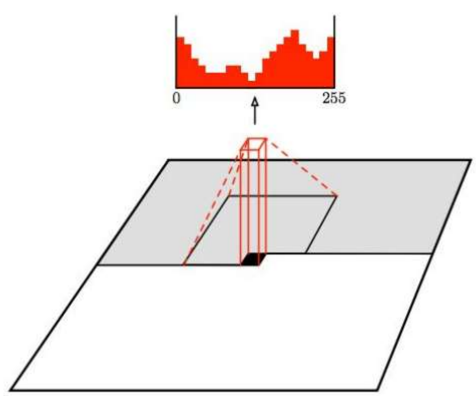
PixelCNN [van der Oord et al. 2016]

16

- PixelCNN also generates image pixels starting from corner,
- Dependency on previous pixels now modeled using a CNN over context region
- Training: maximize likelihood of training images

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, x_2, x_3 \ldots x_{i-1})$$

- Training is faster than PixelRNN
  - can parallelize convolutions since context region values known from training images
- Generation must still proceed sequentially ➔ still slow!



6/1/2024

pra-sâmi

17

Generation Samples

32x32 CIFAR-10

32x32 ImageNet

pra-sâmi

18

PixelRNN and PixelCNN

Pros:

- Can explicitly compute likelihood p(x)
- Explicit likelihood of training data gives good evaluation metric
- Good samples

Con:

- Sequential generation ➔ slow

Reference
- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc…

pra-sâmi

19

Overview

- Four modern approaches to generative modeling:
  - **Generative adversarial networks**
  - Reversible architectures
  - Autoregressive models
  - **Variational autoencoders**
- All four approaches have different pros and cons
- In this session we will focus on
  - Variational autoencoders i.e. VAEs
  - Generative Adversarial Networks i.e. GANs

pra-sâmi

20

Variational Autoencoders (VAE)

pra-sâmi

Difference between PixelCNN and VAE

- PixelCNNs define **tractable** density function, optimize likelihood of training data:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, x_2, x_3 \ldots x_{i-1})$$

- VAEs define **intractable** density function with latent z:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- Cannot optimize directly,
  - So we derive and optimize lower bound on likelihood instead
- Too lengthy, remained theoretical discussions…
- What if we give up on explicitly modeling density, and just want ability to sample?

Background: Autoencoders

- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
- How to learn these features
  - Train such that features can be used to reconstruct original data
  - "Autoencoding" – encoding itself.

z (dimensions) usually smaller than x (dimensionality reduction)

Z Features

Encoders

X Input Data

Originally: Linear + sigmoid
Thereafter: MLP
Now : ReLU CNN

23

Background: Autoencoders

- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
- How to learn these features
  - Train such that features can be used to reconstruct original data
  - “Autoencoding” – encoding itself.

Doesn’t use labels!

L2 Loss function :
$\|x - \hat{x}\|^2$

$\hat{X}$ Reconstructed Input Data

Decoders

Originally: Linear + sigmoid
Thereafter: MLP
Now : ReLU CNN (upConv)

Z Features

Encoders

X Input Data

Encoder: 4 Conv Layers
Decoder: 4 Upconv Layers

6/1/2024

pra-sâmi

24

Background: Autoencoders

- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data
- How to learn these features
  - Train such that features can be used to reconstruct original data
  - “Autoencoding” – encoding itself.

L2 Loss function :
$\|x - \hat{x}\|^2$

Reduce size in Conv
Bring it back to original in upConv

$\hat{X}$ Reconstructed Input Data

Decoders

After first round of training,
we can throw away the decoders

Z Features

Encoders

X Input Data

Decoders n-layer upConv

Z

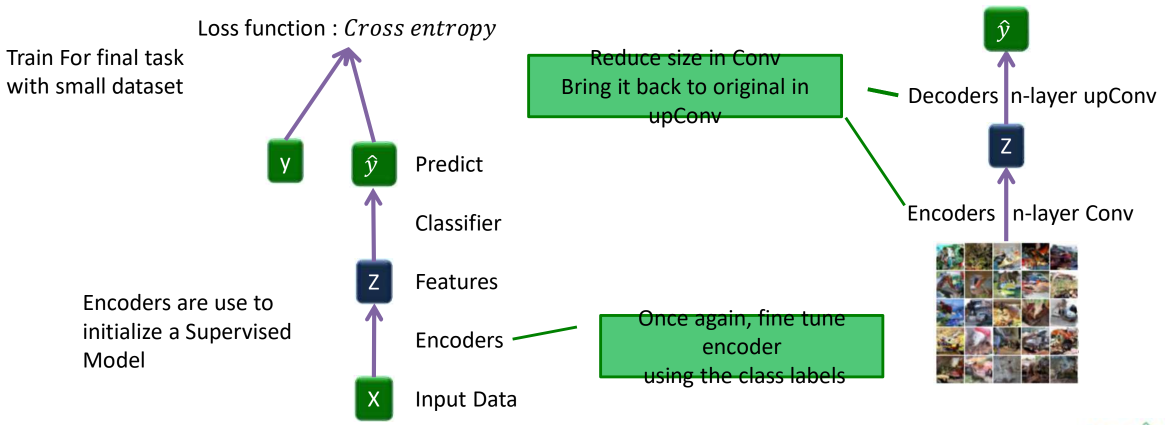
Encoders n-layer Conv

6/1/2024

pra-sâmi

25 Autoencoders

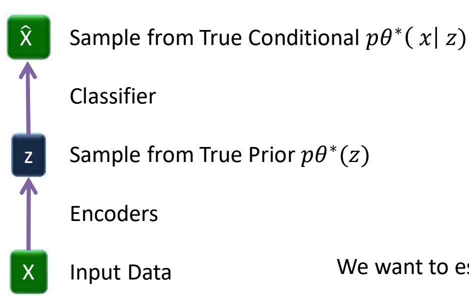
- Now job of decoder is done and we have optimized weights of the Encoder
- Use this encoder for your analysis



pra-sâmi

26 Variational Autoencoders

- Probabilistic spin on auto encoders
  - Will let us sample from the generated data
- Assume that Training data is generated from some underlying unobserved (latent) representation z



Intuition

- X is an image, z is latent factors use to generate x
- Attributes: pose, smile, pupil dilation, angle of eyebrow etc.

We want to estimate the true parameters θ^* of this generative model!

pra-sâmi

27

Variational Autoencoders

- We want to estimate the true parameters θ^* of this generative model!

- How should we represent this model?

$\hat{X}$ Sample from True Conditional $p\theta^*(x|z)$

Conditional $p\theta^*(x|z)$ is complex → represent with Neural Network

Decoder Network

z Sample from True Prior $p\theta^*(z)$

Choose Prior $p\theta^*(z)$ to be simple say Gaussian

Encoders

X Input Data

pra-sâmi

28

Variational Autoencoders

- Straightforward way is to maximize likely hood of data model
  - $p_\theta(x) = \int p_\theta(z) * p_\theta(x|z) * dz$
  - We need to integrate as we are looking at all possible values of x
  - Hence it is not tractable.

- In details: data likelihood $p_\theta(x) = \int p_\theta(z) * p_\theta(x|z) * dz$
  - $p_\theta(z)$ → ok, we can use Gaussian Prior probabilities
  - $p_\theta(x|z)$ → Ok too as we can use a decode Neural Network
  - Integration is a problem, as we need to look at all possible values of z

- It turns out that posterior $p_\theta(x|z)$ is also intractable (difficult to converge)
  - $p_\theta(z|x) = p_\theta(x|z) * \frac{p_\theta(z)}{p_\theta(x)}$ Intractable

- Solution:
  - Decoder model for $p_\theta(x|z)$ and a separate encoder model $q_\theta(z|x)$

pra-sâmi

31

Explicit Density Models

- PixelCNNs define tractable density function, optimize likelihood of training data:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, x_2, x_3 \ldots x_{i-1})$$

- VAEs define intractable density function with latent z:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- Cannot optimize directly, derive and optimize lower bound on likelihood instead
- Too lengthy, remained theoretical discussions…
- What if we give up on explicitly modeling density, and just want ability to sample?
- GANs: don't work with any explicit density function! Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

6/1/2024

pra-sâmi

32

Generative Adversarial Networks

6/1/2024

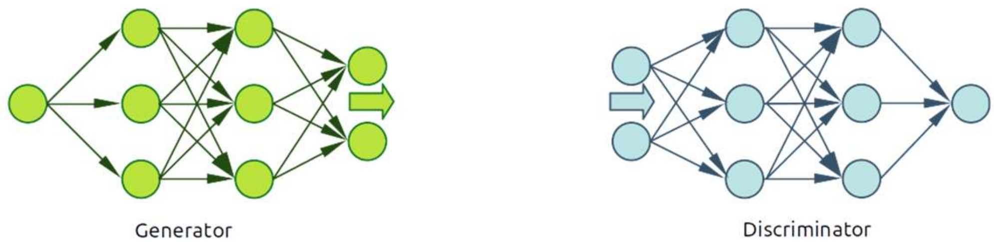
pra-sâmi

33

Generative Adversarial Networks

- The idea behind Generative Adversarial Networks (GANs): train two different networks
  - The generator network tries to produce realistic-looking samples
  - The discriminator network tries to figure out whether an image came from the training set or the generator network

- The generator network tries to fool the discriminator network

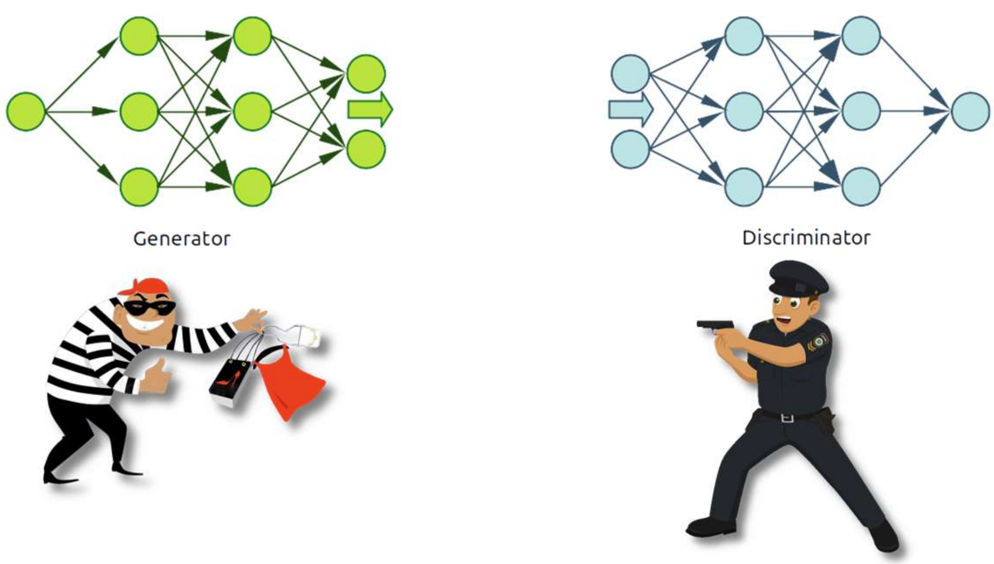


6/1/2024

pra-sâmi

34

Generative Adversarial Network

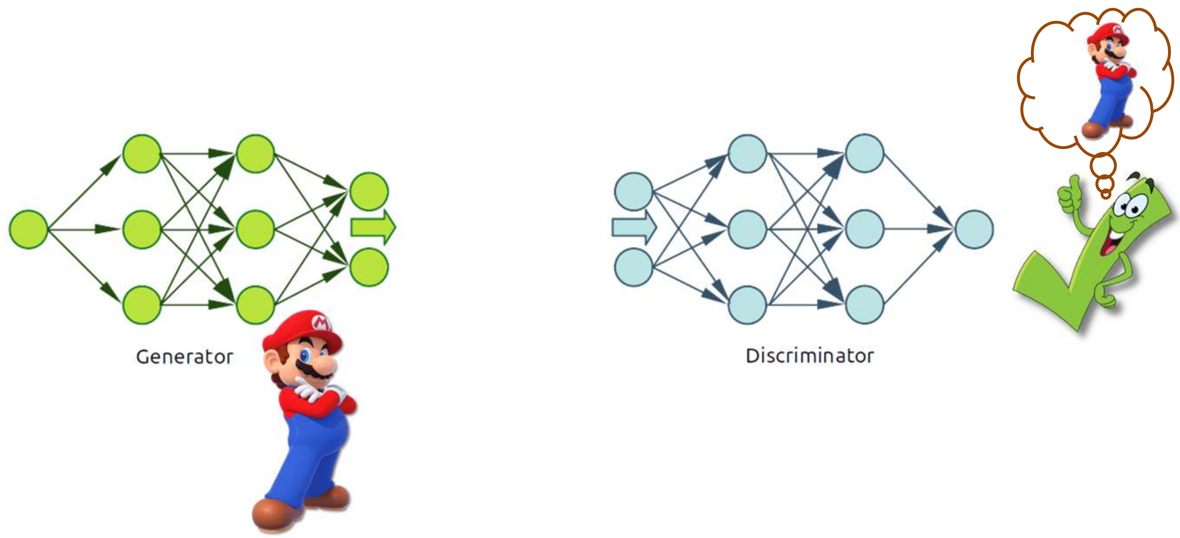


6/1/2024

pra-sâmi

35

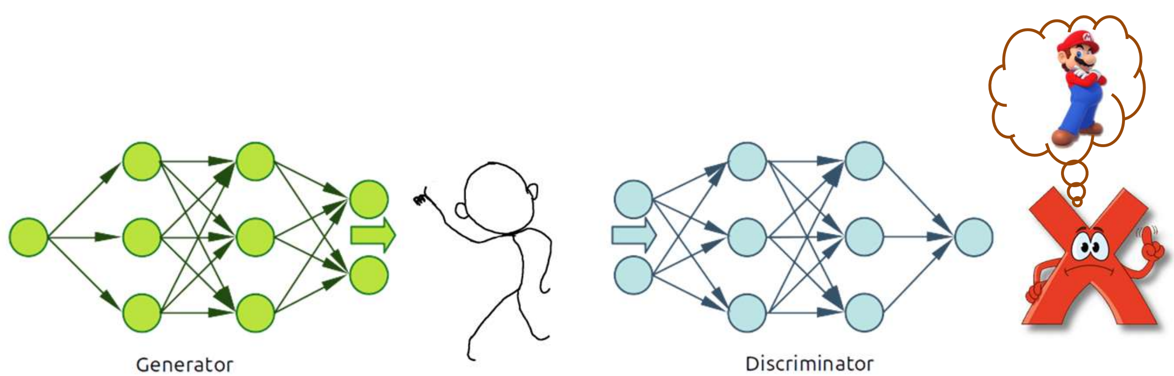
Generative Adversarial Network



6/1/2024

36

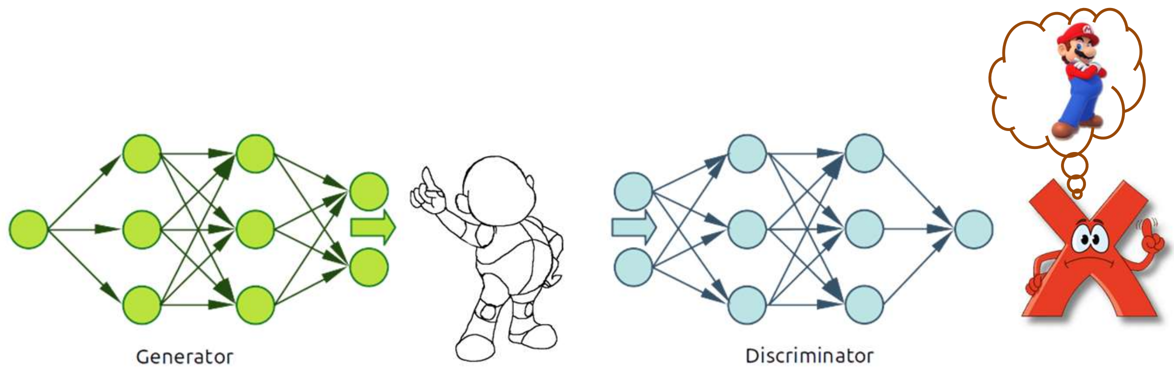
Generative Adversarial Network



6/1/2024

37

Generative Adversarial Network

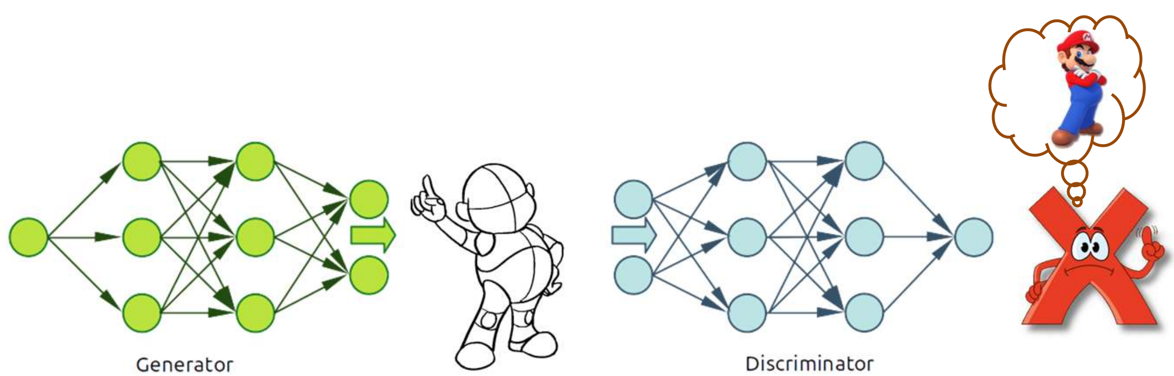


6/1/2024

pra-sâmi

38

Generative Adversarial Network

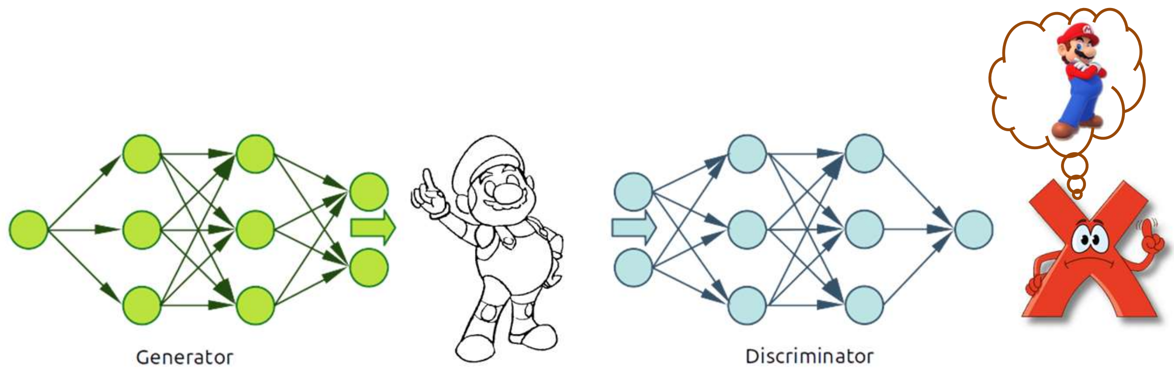


6/1/2024

pra-sâmi

39

Generative Adversarial Network

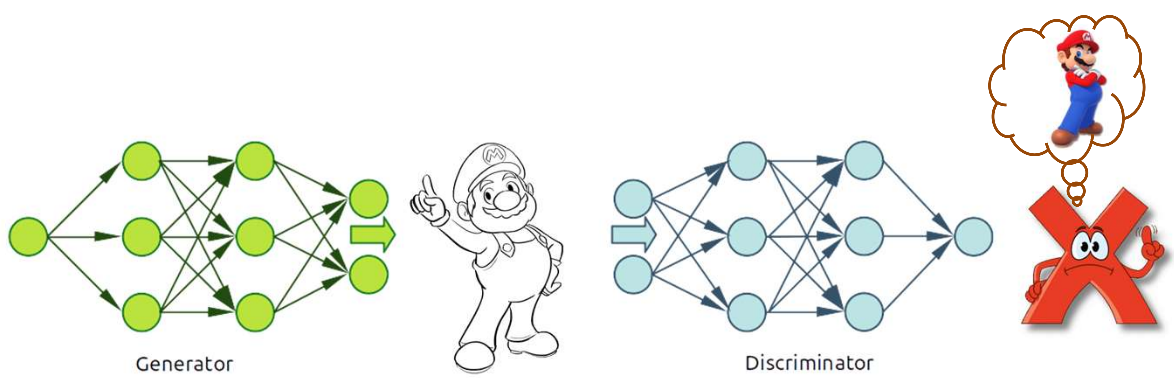


6/1/2024

pra-sâmi

40

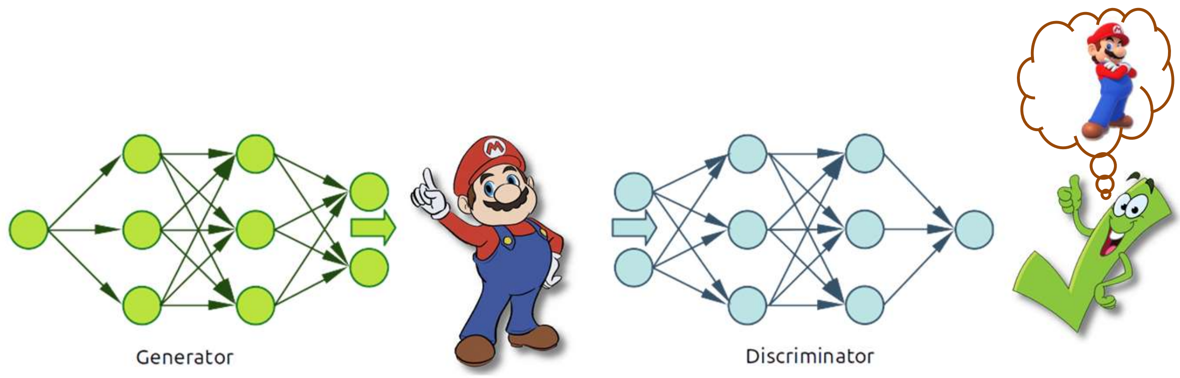
Generative Adversarial Network



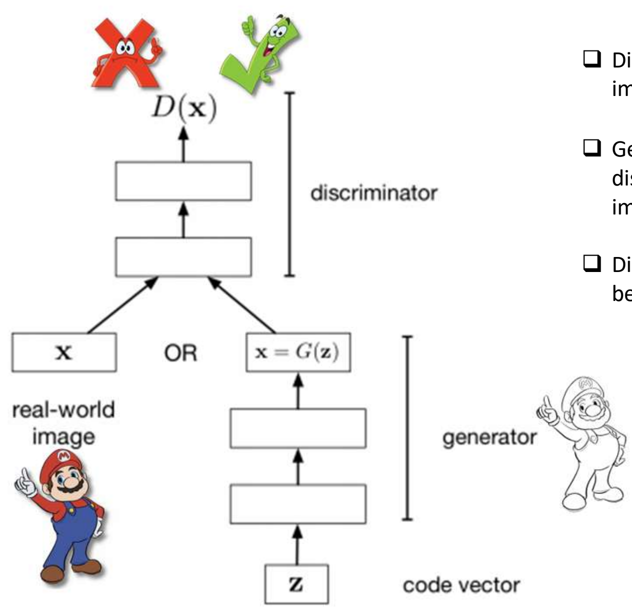
6/1/2024

pra-sâmi

Generative Adversarial Network



Generative Adversarial Networks

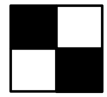


- ❑ Discriminator network is trained on real images as well as generated images
- ❑ Generator network: try to fool the discriminator by generating real-looking images
- ❑ Discriminator network: try to distinguish between real and fake images

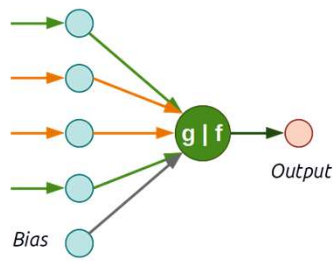
43

Coding Gan

- Imagine a simplest 2 x 2 images
  - Depending upon features these shades may vary



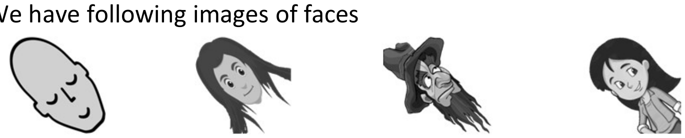
- Let's also consider our simplest neural network



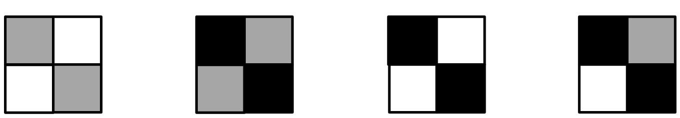
44

GAN

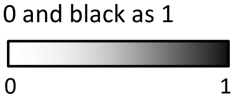
- Imagine all images are slanted backward by 45°
- We have following images of faces



- The corresponding pixel on the images will be as follows:



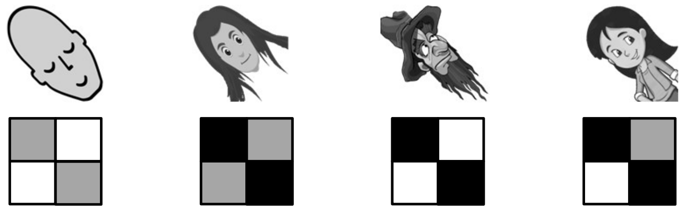
- For argument sake let's take white pixel as 0 and black as 1
  - Gray shades will be somewhere in between



45

GAN

- We have following images of faces



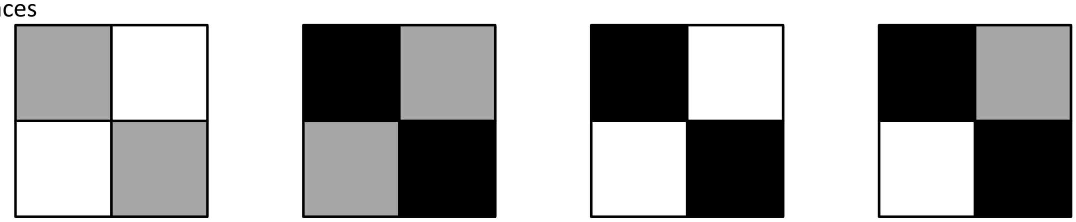
- Images containing no face will appear as follows:

pra-sâmi

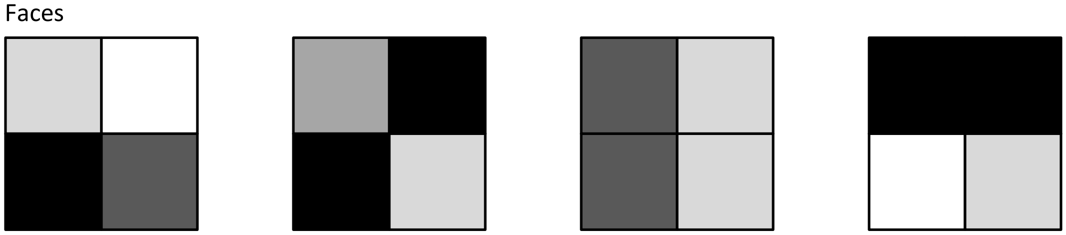
46

What Agent will see

- Faces



- No Faces



pra-sâmi

47

What Agent will see

- Faces

0.75	0
0	0.75

1	0.5
0.5	1

1	0
0	1

1	0.25
	1

- No Faces

0.25	0
1	0.75

0.75	1
1	0.25

0.75	0.25
0.75	0.25

1	1
0	0.25

48

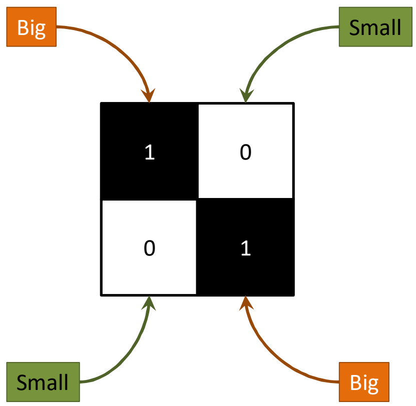
Discriminator

49

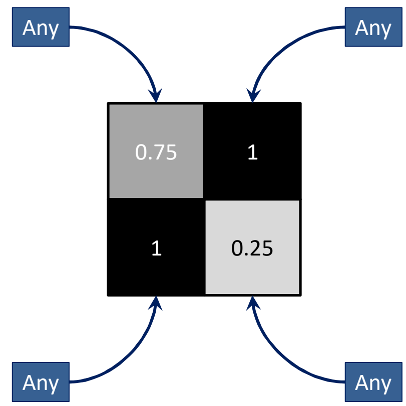
Discriminator

How to differentiate between Face and Noise!

- Faces



- Noise

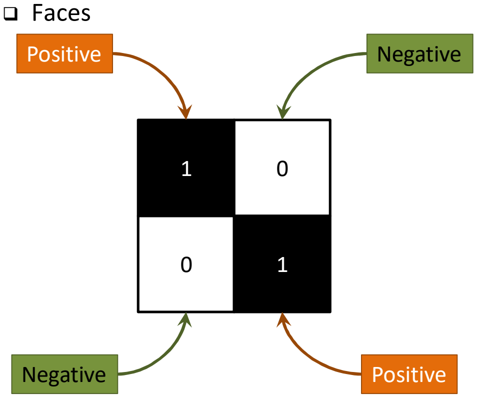


50

Discriminator

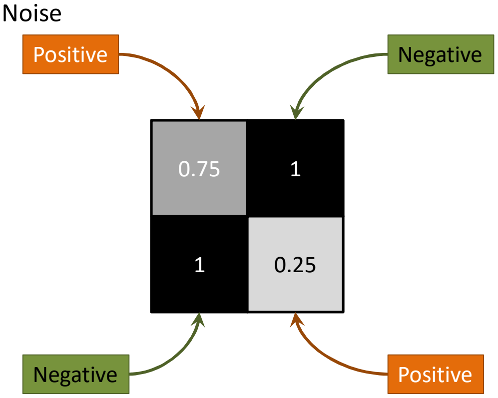
How to differentiate between Face and Noise!

- Faces



- 1 * 1 + 1 * 1 + 0 * (-1) + 0* (-1) = 2.0

- Noise



- 0.75 * 1 + 0.25 * 1 + 1 * (-1) + 1 * (-1) = -1.0

Considering threshold as 0.0 (arbitrary), we can safely assume:
- Positive value → Face
- Negative value → No Face

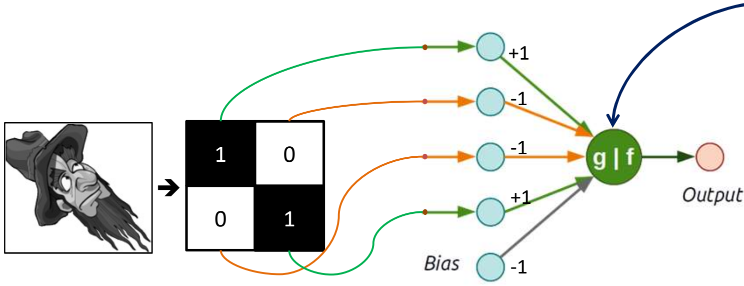
51

Discriminator

In Neuron Network Language

g = + 1 * 1 + 0 * (-1) + 0 * (-1) + 1 * 1 - 1
= 1

f = σ (g) ➔ output = 0.73

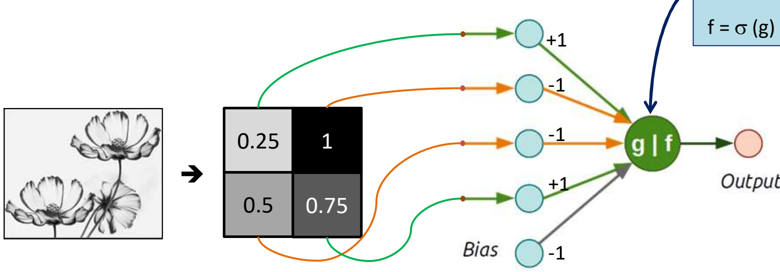


52

Discriminator

g = + 0.25 * 1 + 1 * (-1) + 0.5 * (-1) + 0.75* 1 - 1
= -1.5

f = σ (g) ➔ output = 0.18



- Thus
  - for f > 0.5 it is face
  - for f < 0.5 it is not a face

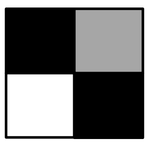
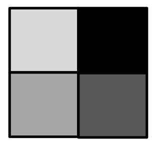
53

Discriminator

- All images are slanted backward by 45°

0.5	0	1	0.5	1	0	1	0.5	0.25	1
0	0.5	0.5	1	0	1	0	1	0.5	0.75

- So the discriminator knows that following:

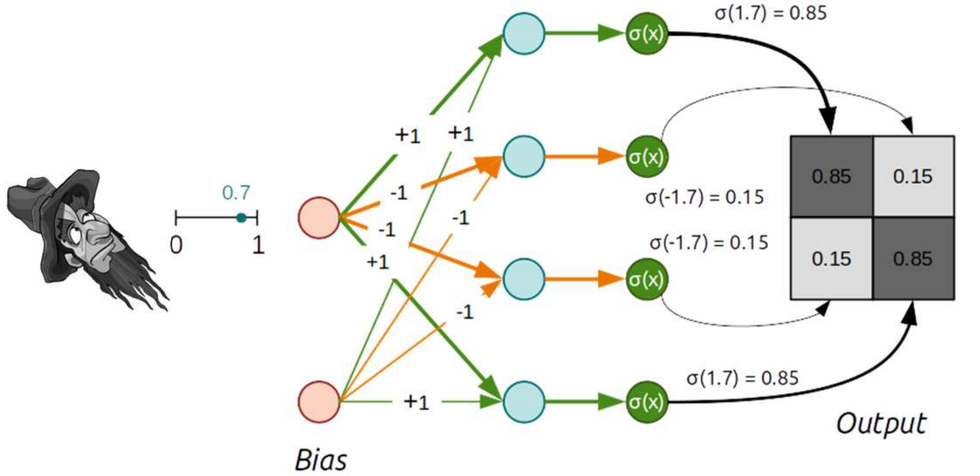
- Face  Not a Face 

54

Generator

55

Generator

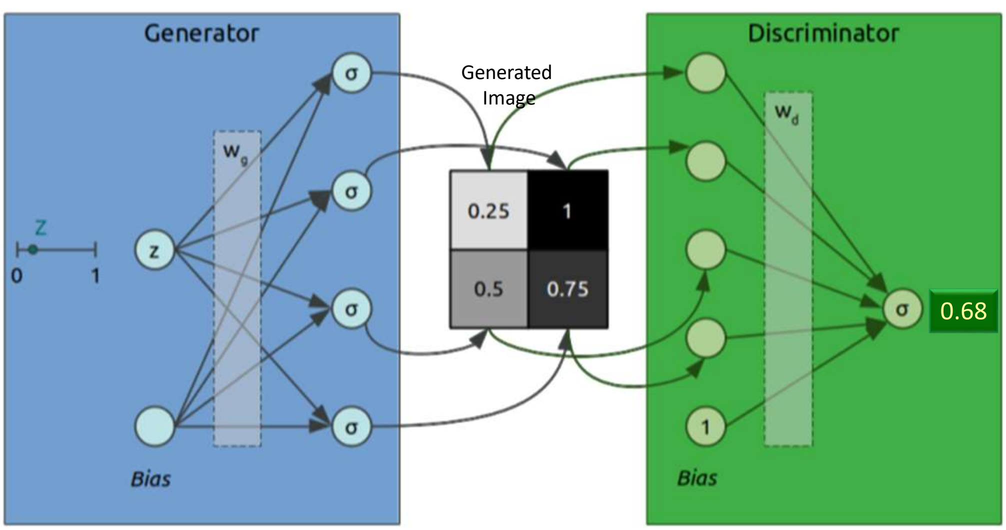


6/1/2024

pra-sâmi

56

Error Functions



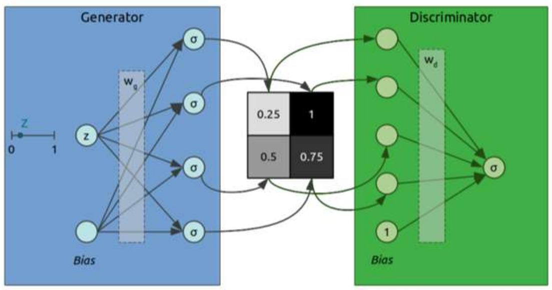
Generator is Max and Discriminator is Min

6/1/2024

pra-sâmi

57

Error Functions



- Generator and Discriminators are working against each other
- Discriminator tries to generate label as close to 0 as possible (Claiming it is fake)
  - Error function = -log(1-p)
- Generator tries to generate labels as close to 1 as possible (Claiming it to be an image)
  - Error function = -log(p)

58

Error Functions

Discriminator

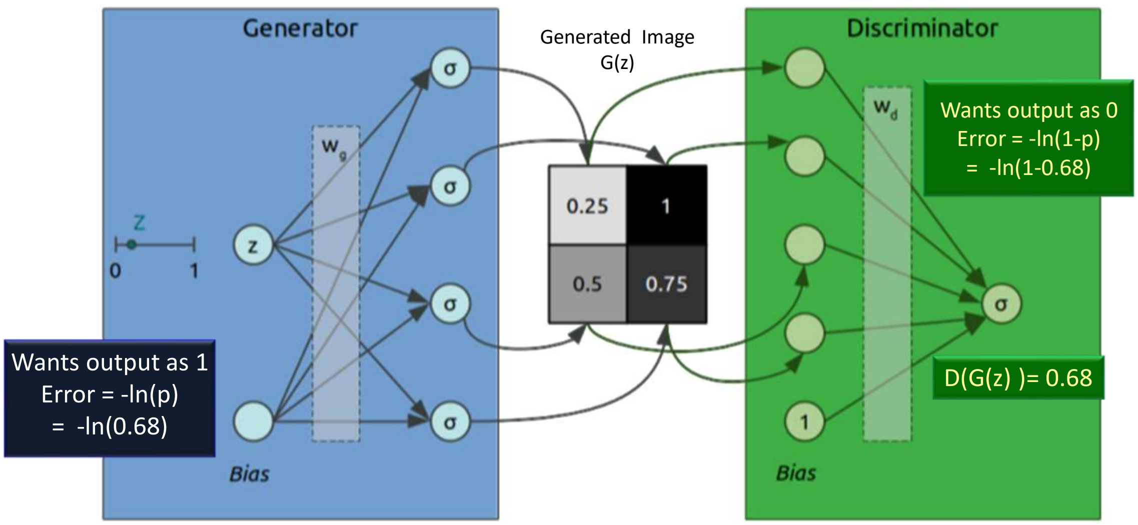
- If our value 0 and prediction is 0.1 ➔ error is small
- If our value is 0 and prediction is 0.9 ➔ error is large
- Consider negative log error
  - For pred = 0.1; error = - ln (1- 0.1) = 0.11
  - For pred = 0.9 error = - ln (1- 0.9) = 2.30
- Thus our error function is:
  - - ln (1-pred)

Generator

- If our value 1 and prediction is 0.1➔ error is large
- If our value is 1 and prediction is 0.9 ➔ error is small
- Consider negative log error
  - For pred = 0.1; error = - ln (0.1) = 2.30
  - For pred = 0.9 error = - ln (0.9) = 0.1
- Thus our error function is:
  - - ln (pred)

59

Error Functions



Error Function = -ln(D(G(z))

Error Function = -ln(1-D(G(z))

6/1/2024

pra-sâmi

60

Three Reasons that it’s a Miracle GANs Work

- G has a reinforcement learning task
  - It knows when it does good (i.e., fools D) but it is not given a supervised signal
  - Reinforcement learning is hard
  - Back prop through D provides G with a supervised signal; the better D is, the better this signal will be
- Can’t describe optimum via a single loss
  - Will there be an equilibrium?
- D is seldom fooled
  - But G still learns because it gets a gradient telling it how to change in order to do better the next round.

6/1/2024

pra-sâmi

61

Training GANs: Two-player game

- Generator network: try to fool the discriminator by generating real-looking images
- Discriminator network: try to distinguish between real and fake images
- Train jointly in MiniMax game
- MiniMax objective function:

$$\min_{\theta_g}\max_{\theta_d}\left[E_{x\sim pdata}\log D_{\theta_d}(x) + \ E_{z\sim p(z)}\log(1 - D_{\theta_d}((G_{\theta_d}(z)))\right]$$

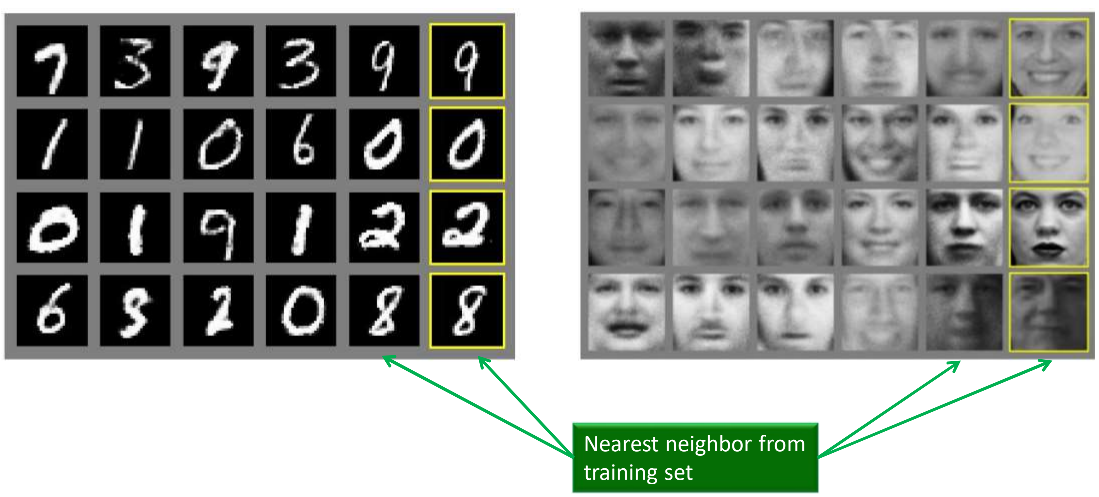
- Discriminator outputs likelihood in (0,1) of real image
- Discriminator (θ_d) wants to maximize objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to minimize objective such that D(G(z)) is close to 1
  - Discriminator is fooled into thinking generated G(z) is real

6/1/2024

pra-sâmi

62

Generative Adversarial Nets



6/1/2024

pra-sâmi

64

Reflect…

- Which of the following are key components of a Generative Adversarial Network (GAN)?
  a. Generator
  b. Discriminator
  c. Classifier
  d. Loss function
- Answer : a, b, d

- Select the statements that correctly describe the training process of a GAN.
  a. The generator aims to produce data that is indistinguishable from real data.
  b. The discriminator provides feedback to the generator about the generated samples.
  c. GANs are trained using supervised learning techniques.
  d. The loss function for GANs involves both a generator loss and a discriminator loss.
- Answer: a, b, d

- Which applications can benefit from the use of Generative Adversarial Networks?
  a. Image generation
  b. Style transfer
  c. Text summarization
  d. Speech recognition
- Answer : a, b, d

- What is the purpose of the generator in a GAN?
  a. To discriminate between real and fake data.
  b. To generate synthetic data.
  c. To evaluate the quality of generated samples.
  d. To provide feedback to the discriminator.
- Answer : b

6/1/2024

pra-sâmi

65

Reflect…

- Choose the correct statements regarding the mode collapse phenomenon in GANs.
  a. Mode collapse occurs when the generator produces diverse samples covering the entire data distribution.
  b. Mode collapse happens when the generator focuses on generating only a limited set of samples.
  c. Mode collapse is a desired behavior in GAN training.
  d. Mode collapse is related to the overfitting of the discriminator.
- Answer : b

- Which regularization techniques are commonly used to stabilize GAN training?
  a. Dropout
  b. Batch normalization
  c. L1 regularization
  d. Gradient clipping
- Answer : a, b, d

- Select the statements that correctly describe the challenges associated with training Generative Adversarial Networks.
  a. GANs may suffer from mode collapse.
  b. Training GANs can be unstable.
  c. GANs always converge to a globally optimal solution.
  d. GANs require a large amount of labeled training data.
- Answer: a, b

- What is the role of the discriminator in a GAN?
  a. To generate synthetic data.
  b. To evaluate the quality of generated samples.
  c. To provide feedback to the generator.
  d. To discriminate between real and fake data.
- Answer: d

6/1/2024

pra-sâmi

67

6/1/2024

pra-sâmi