Generative Adversarial Networks

Mohammad Rahmdel
Amirkabir University of Technology

Amirkabir University of Technology
(Tehran Polytechnic)

AAISS
Amirkabir Artificial Intelligence Summer Summit

August 2020

Overview

Generative Models
Applications
Taxonomy

GANs
How does it work
Generator
Discriminator

Training process
Cost Functions

Tips and Tricks
Challenges

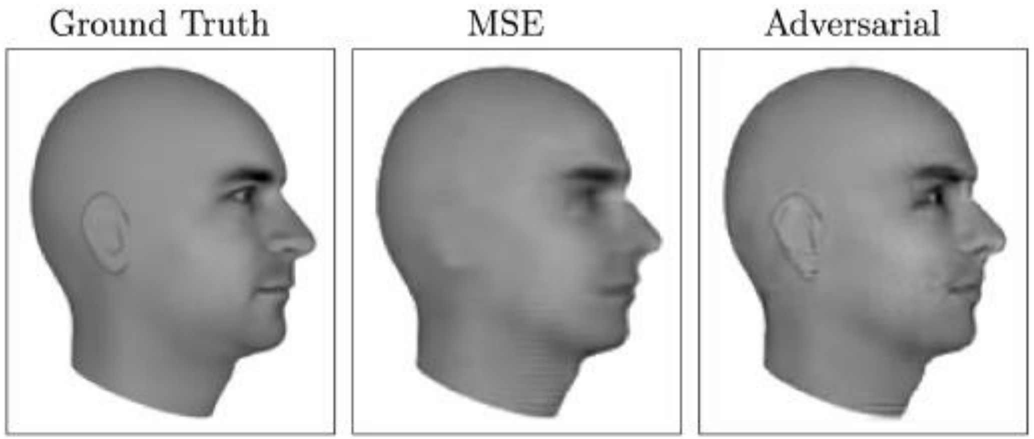
Generative Models

Any model that takes a training set, consisting of samples drawn from a distribution p_{data}, and learns to represent an estimate of that distribution. The result is a probability distribution p_{model}

Why Generative Modeling?

- Represent and manipulate high-dimensional probability distributions
- Incorporated into reinforcement learning
- Train and prediction in missing data(semi-supervised learning)
- Enables machine learning to work with multi-modal outputs
- Realistic samples required in many tasks

Multi-modal



Sample generation

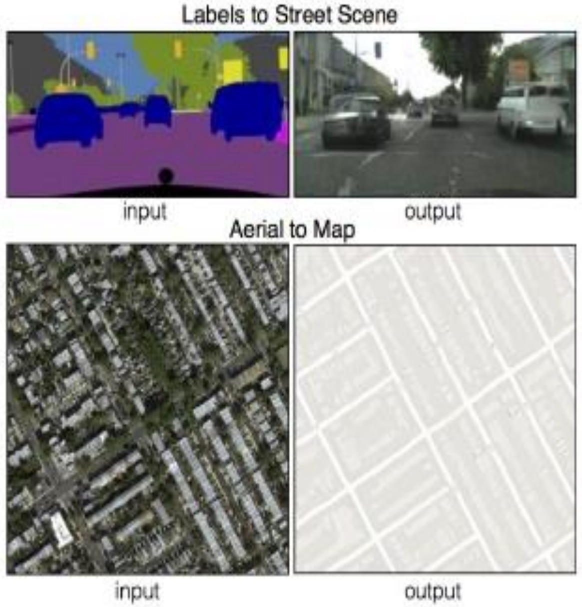
Generative Models Applications

Image super-resolution

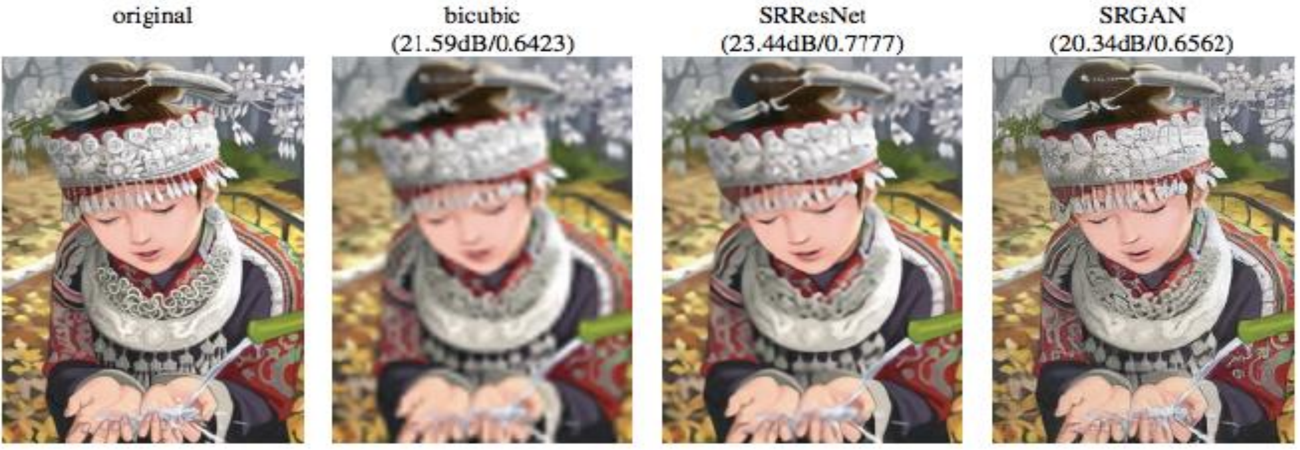
Creating realistic images

Image-to-image translation

Image inpainting



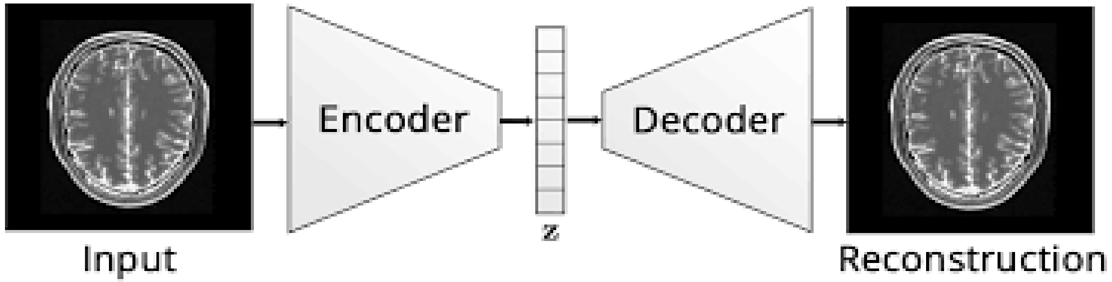
colorization

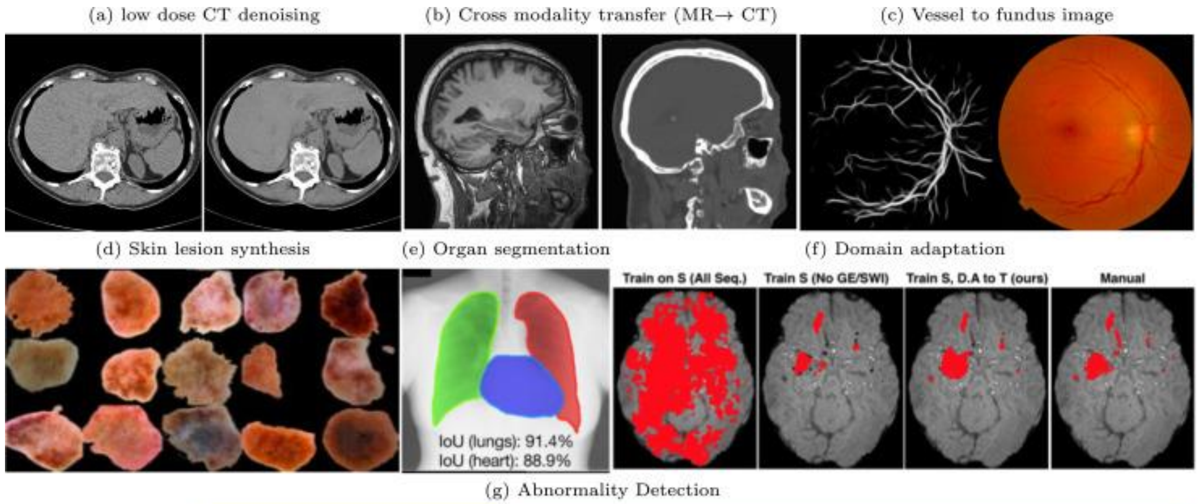


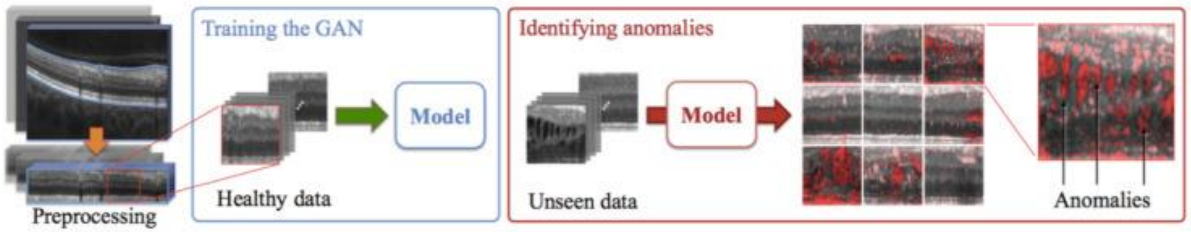
Super-resolution

Generative Models Applications

- Anomaly Detection
- Medical Imaging
- Segmentation
- NLP







A taxonomy of Deep Generative Models

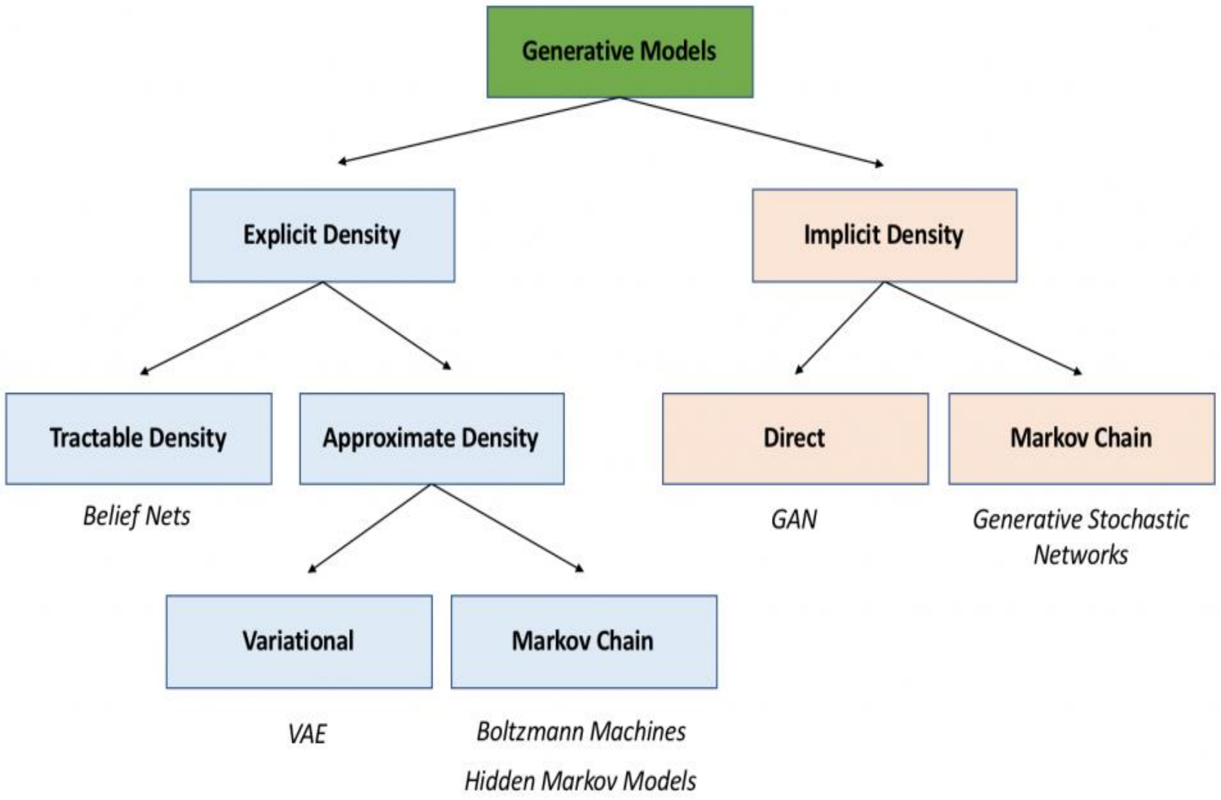
- We restrict our attention to deep generative models that work by maximizing the likelihood

- The basic idea of Maximum Likelihood is to define a model that provides and estimate of a probability distribution, parametrized by θ.

- Likelihood: the probability that the model assigns to training data $\prod_{i=1}^{m} p_{model}(x^{(i)};\theta)$

- Choose θ for the model to maximize the above equation

$$\theta^* = \arg_{\theta} \max \sum_{i=1}^{m} \log p_{model}(x^{(i)};\theta)$$



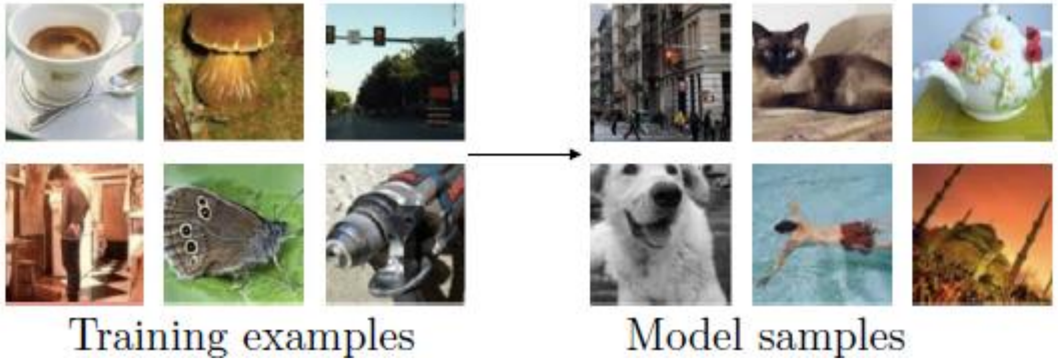
A taxonomy of Deep Generative Models

Some generative models estimates p_{model} explicitly(density estimation)

Some models only able to generate samples from p_{model} (sample generation)

Density Estimation

Sample Generation



Explicit Density Models

Tractable Density models

- Very time-consuming
- Cannot be parallelized
- Not applicable for interactive tasks

➢ highly effective, permit the use of an optimization algorithm directly on the log-likelihood of the training data

Explicit Models requiring Approximation

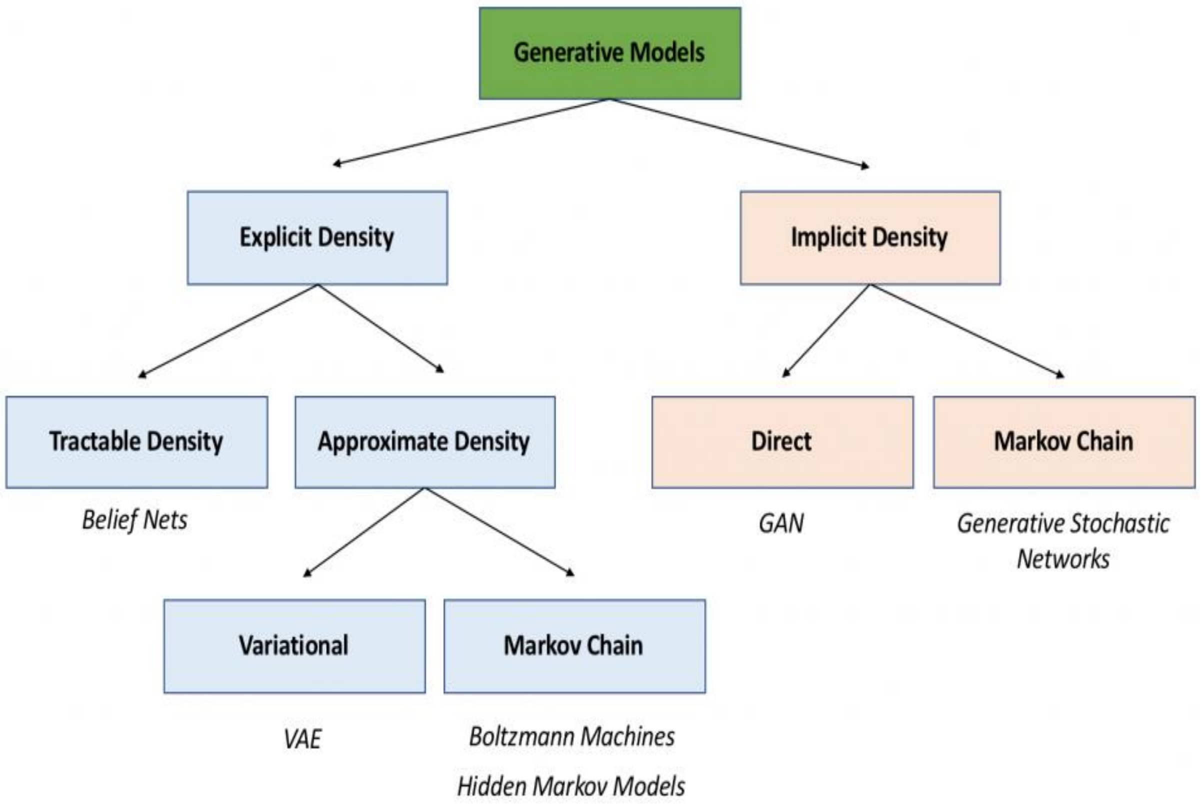
Variational

- Produce low quality samples

Markov Chain

- Convergence can be very slow
- Not scalable
- Less efficient in high-dimensional spaces

A taxonomy of Deep Generative Models



Implicit Density Models

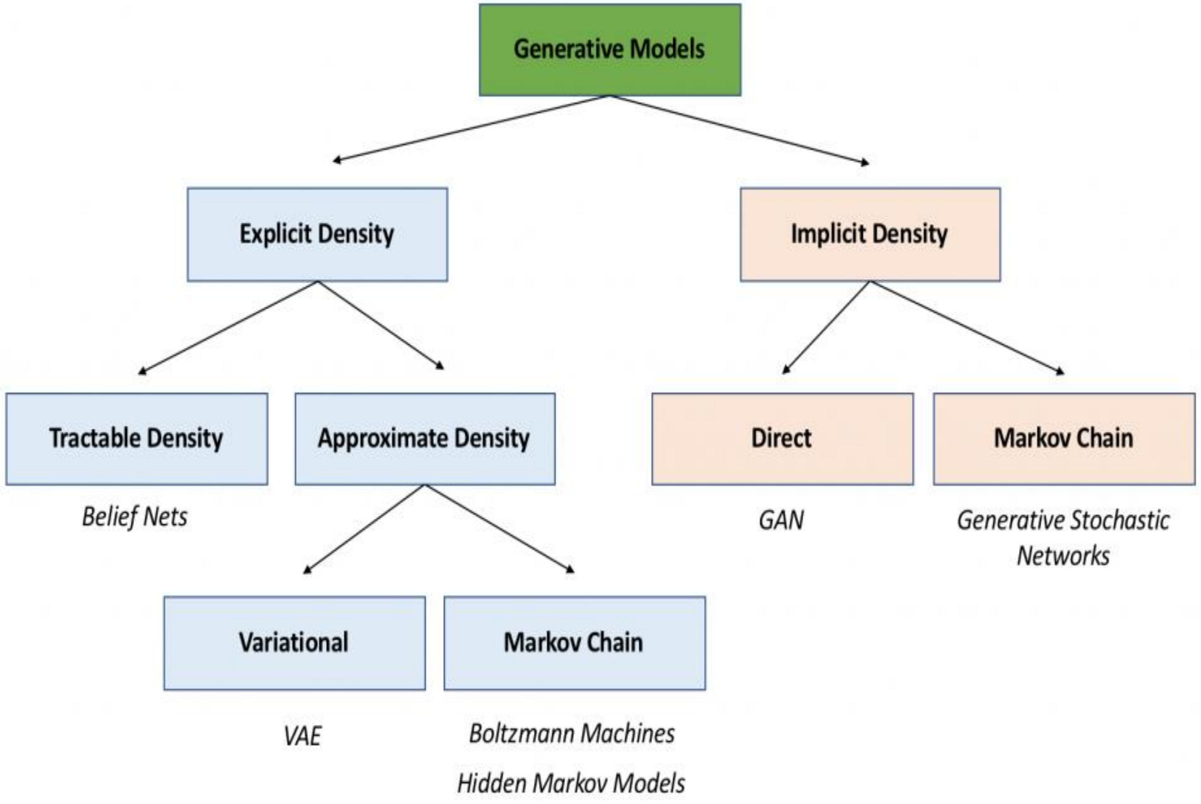
Generative Stochastive Networks

- Not scalable

Generative Adversarial Networks(GANs)

- ✓ Generate samples in parallel
- ✓ No markov chains are needed
- ✓ No variational bound is needed
- ✓ Producing better samples than other methods
- ▪ Requires finding **a Nash equilibrium**

A taxonomy of Deep Generative Models

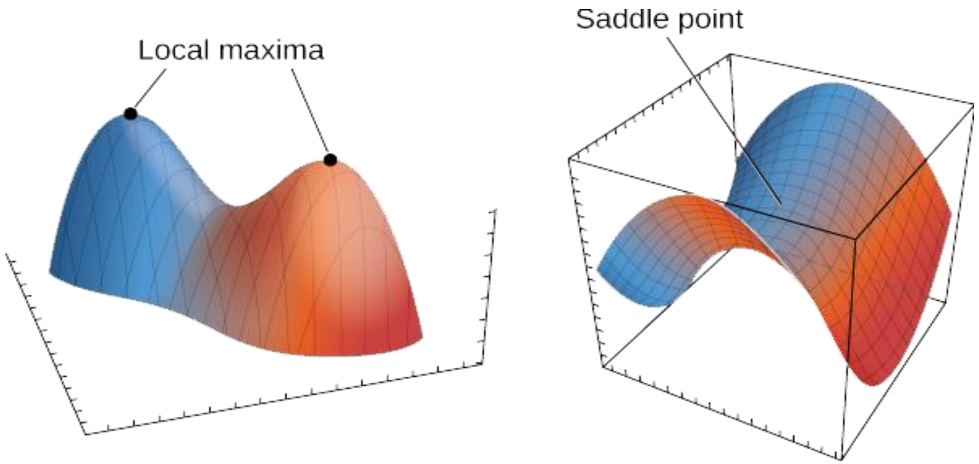


Generative Adversarial Networks

- How do GANs work?

  - ➢ A game between two players(networks); Generator and Discriminator

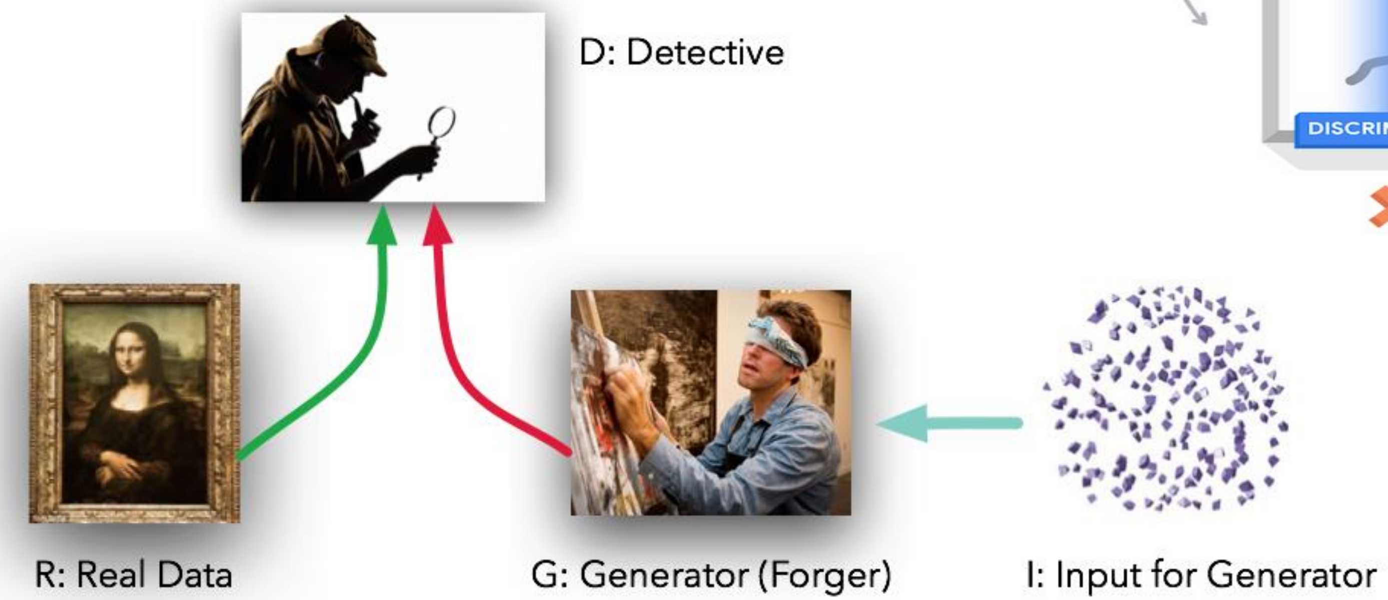
  - Generator create samples intended to come from the same distribution as the training data
  - Discriminator examines samples to determine whether they are real or fake
  - Solution is a saddle point

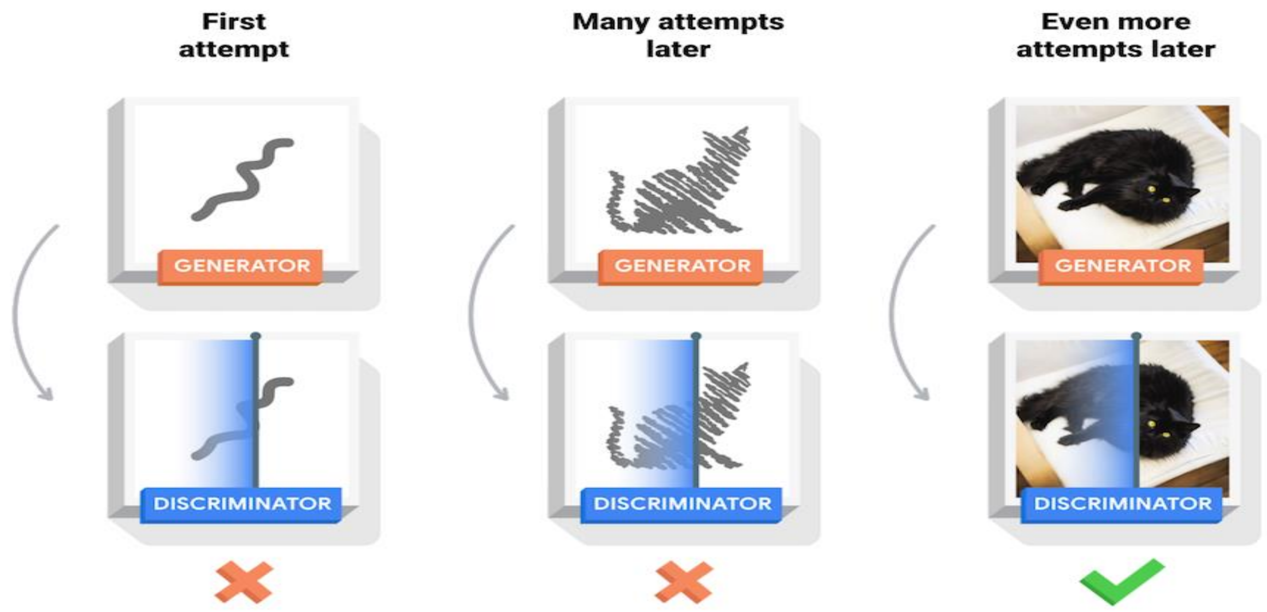


How do GANs work?

- The Discriminator learns using traditional supervised learning techniques, dividing inputs into two classes, real or fake
- The Generator is trained to fool the discriminator
- Both players have cost functions defined in terms of both players' parameters
- D wishes to minimize $J^D(\theta^{(D)}, \theta^{(G)})$ controlling only $\theta^{(D)}$ and G wishes to minimize $J^G(\theta^{(D)}, \theta^{(G)})$ controlling only θ_G
- This scenario is described as a(minimax) game rather an optimization problem
- The solution to a game is Nash equilibrium. In this context it is a tuple $(\theta^{(D)}, \theta^{(G)})$, that is a local minimum of J^D with respect to $\theta^{(D)}$ and a local minimum of J^G with respect to θ_G

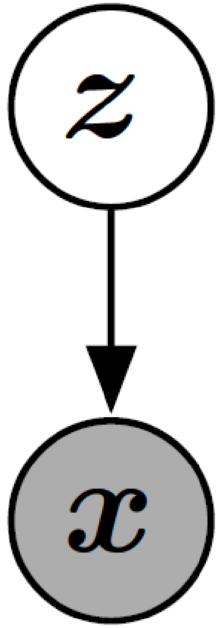
How do GANs work?





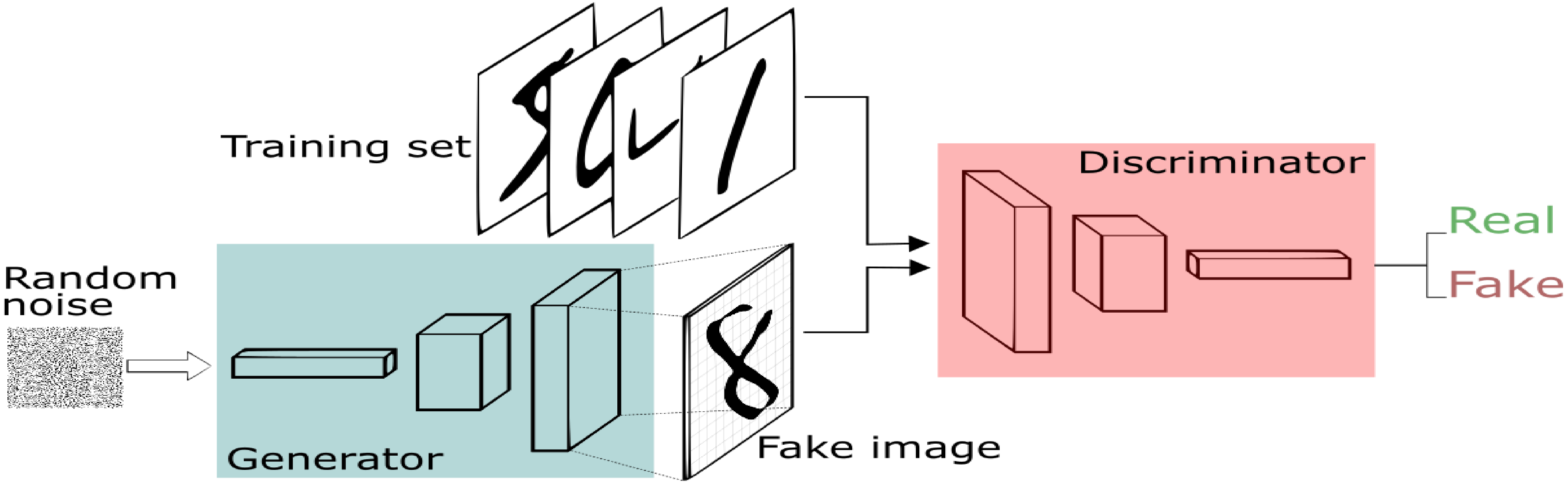
Generator

- Simply a **differentiable** function G
- is responsible for **generating** fake samples of data
- It takes as input some latent variable z (random noise) and outputs data that is of the same form as data in the real data
- If our latent variable is z and our target variable is x, we can think of the generator of network as learning a function that maps from z (the latent space) to x (hopefully, the real data distribution)
- Typically a **DNN** is used to represent G
- Inputs to the G do not need to be at the first layer of the network. It may be provided at any point throughout the network(very few restrictions on the design of the generator net)



Discriminator

- a **differentiable** function D, typically implemented as a **DNN**
- Its role is to discriminate
- It is responsible for taking in a list of samples and coming up with a prediction for whether or not a given sample is real or fake. The discriminator will output a higher probability if it believes a sample is real

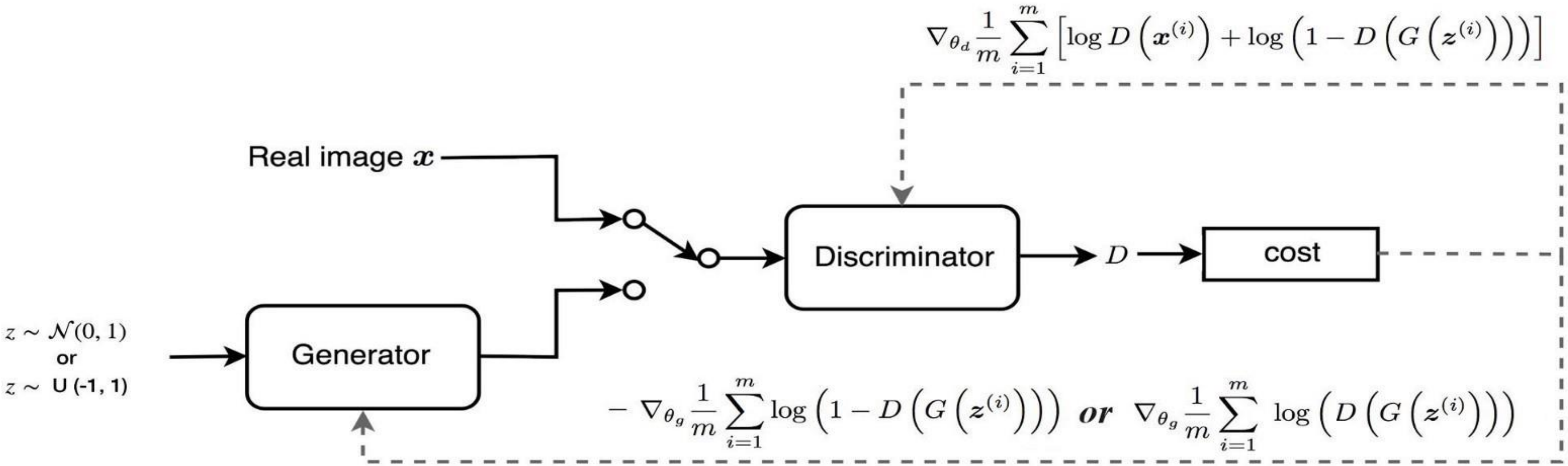


The Training Process

- Consists of simultaneous Stochastic Gradient Descent(SGD)
- On each step, two minibatches are sampled; a minibatch of x values from the dataset, a minibatch of z values drawn from the model's prior over latent variables
- Two gradient steps are made simultaneously; one updating $\theta^{(D)}$ to reduce J^D, and one updating $\theta^{(G)}$ to reduce J^G using gradient-based optimization algorithms
- Adam is usually a good choice
- Many authors recommend updating more one player than the other

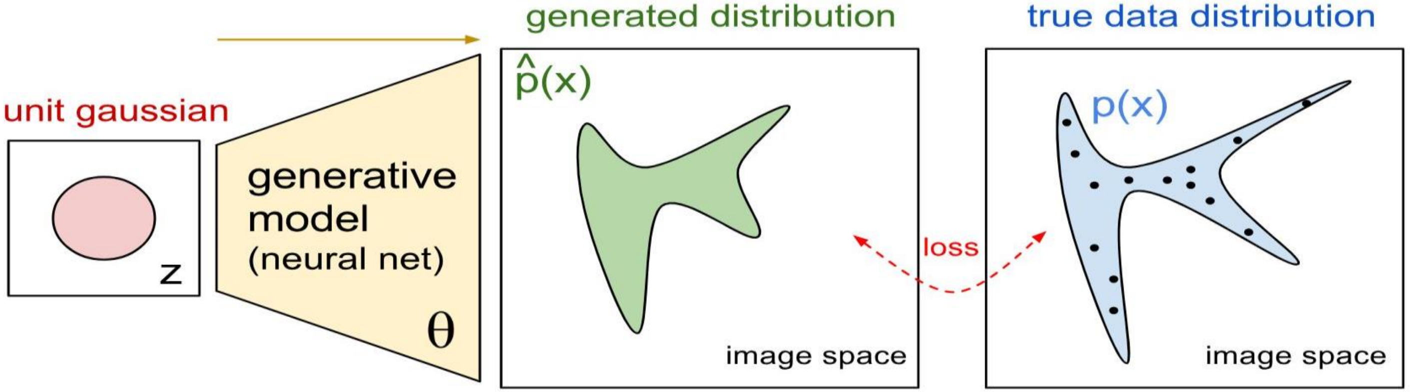
The Training Process

- D wishes to D(x) to be near to 1 and D(G(z)) approach 0
- G strives to make D(G(z)) approach 1
- If both models have sufficient capacity, then the Nash equilibrium of this game corresponds to the G(z) being drawn from the same distribution as the training data, and D(x)= 0.5 for all x



Cost Functions

- Several different cost functions may be used within the GANs framework.



- Standard cross-entropy cost

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$

Cost Functions

- The simplest version of the game is a zero-sum game

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

$$J^{(G)} = -J^{(D)}$$

- In the minimax game, D minimizes a cross-entropy and G maximizes the same cross-entropy
- G's cost is useful for theoretical analysis, but does not perform well in practice
- Problem is that the Generator's gradient vanishes

- The Generator's cost become $J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$

- Heuristically motivated!
- ➢ J(G) does not make reference to the training data directly; This makes GANs resistant to overfitting

Tips and Tricks

- Train with labels
- One-sided label smoothing
- Virtual batch normalization
- Balance G and D (Vanishing Gradients)

Challenges

Highly sensitive to Hyperparameters

Non-convergence

Mode Collapse

Other games

Evaluation

Discrete outputs

Semi-Supervised Learning

Using the code(latent)

Question?

Thanks for your attention

References

- NIPS 2016 Tutorial: GANs
- Generative Adversarial Nets 2014