GANs: Generative Adversarial Networks

Biswarup Bhattacharya
University of Southern California
bbhattac@usc.edu

18-Aug-2017
$(0x3F)_{16}$

About Me

- Bachelors in EE (minor in CS) from IIT Kharagpur, India.
- Currently a 1st year PhD CS student @ USC.
- Bachelors thesis: Applying AI to solve national-level power grid problems.
- Research Intern @ **Adobe Research** – Virtual Assistant for Enterprises.
- Software Engineering @ **National Digital Library** – Member of the first team to build the core backend of India's largest public digital library (Government project). http://ndl.iitkgp.ac.in
- Worked in various projects on deep learning, machine learning, networks (electrical and social), power & control systems and IoT among others.
- https://biswarupb.github.io

Outline

- Part 1: Definition of GANs
- Part 1: Why GANs?
- Part 1: GAN details
- Part 1: Training GANs
- Part 1: Limitations of GANs
- Part 1: DCGAN
- Part 2: Coding vanilla GANs
- Part 3: AAAI 2017 work: Hand-GAN
- Part 3: NIPS 2016 work: SAD-GAN
- Other work that I have done
- References

These indicate tips and tricks which can be used for actually training GANs.

Part 0: Prerequisites

Basic assumptions of this talk

Prerequisites

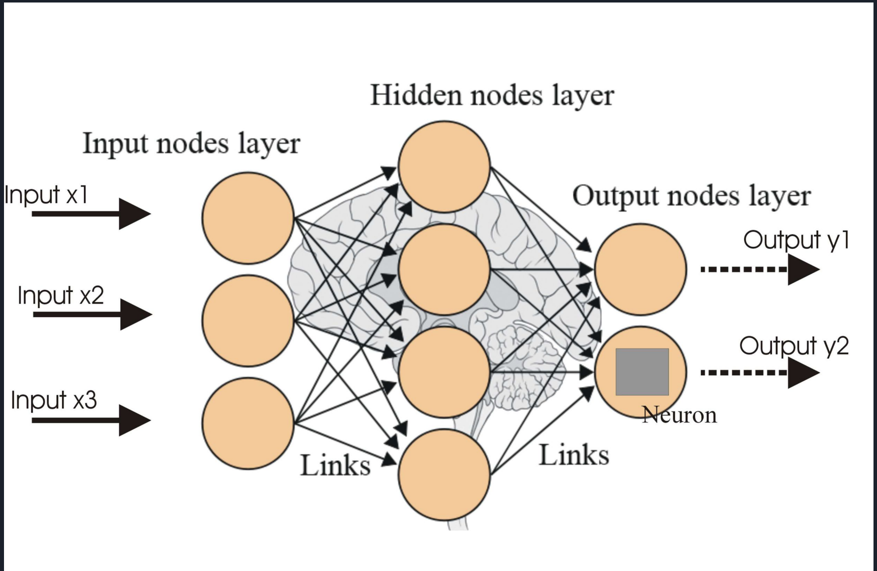
- What is supervised and unsupervised machine learning?
- General idea about deep learning and NNs
- What is a statistical distribution?
- What is a zero sum game?

Machine Learning

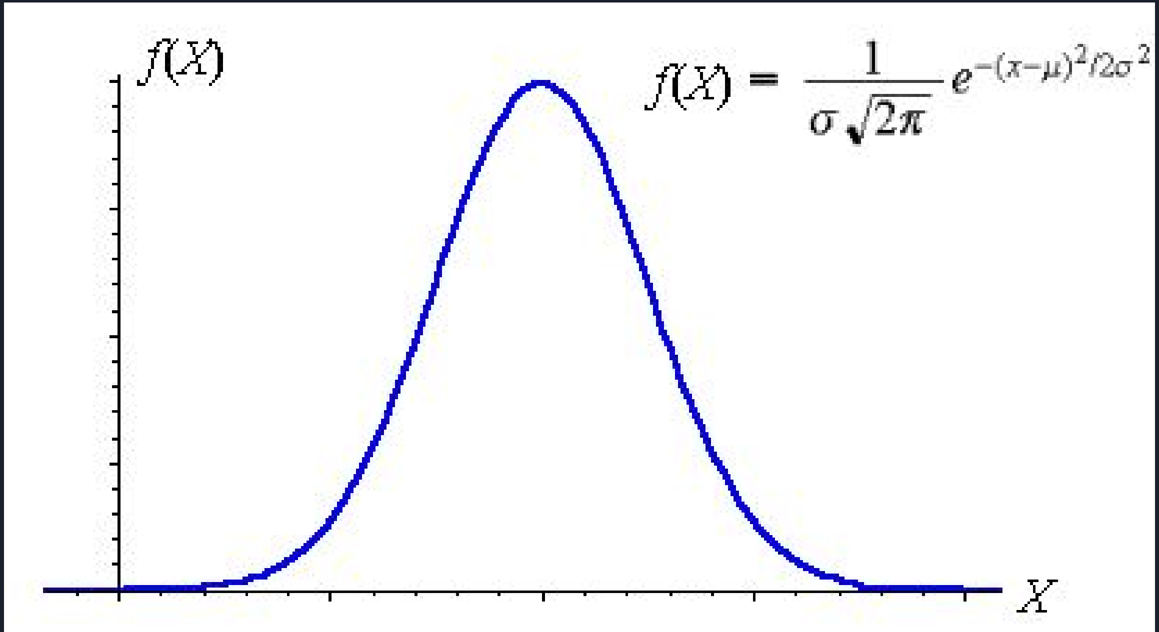
- Supervised: Task of inferring a function from labeled training data. Labelled and a teacher exists. Eg. SVM, decision trees, random forests.
  - Classification
  - Regression
- Unsupervised: Model the underlying structure or distribution in the data in order to learn more about the data. No labels and no teacher.
  - Clustering
  - Association
- Semi-supervised: Problems where you have a large amount of input data (X) and only some of the data is labeled (Y).

Deep Learning

Application of artificial neural networks (ANNs) to learning tasks that contain more than one hidden layer.



Statistical Distribution



Normal Distribution

Zero-sum game

- If one gains, another loses.
- Rock, paper, scissors is an example of a zero-sum game without perfect information. No matter what a person decides, the mathematical probability of winning, drawing, or losing is exactly the same.

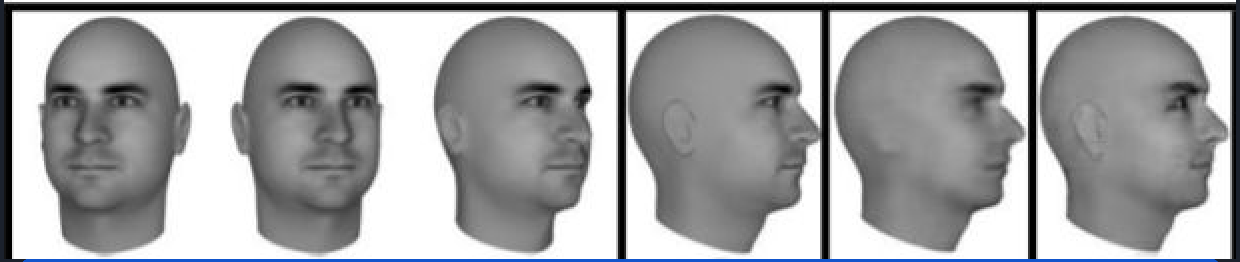
		Player 2		
		Rock	Paper	Scissors
Player 1	Rock	0 0	1 -1	-1 1
	Paper	-1 1	0 0	1 -1
	Scissors	1 -1	-1 1	0 0

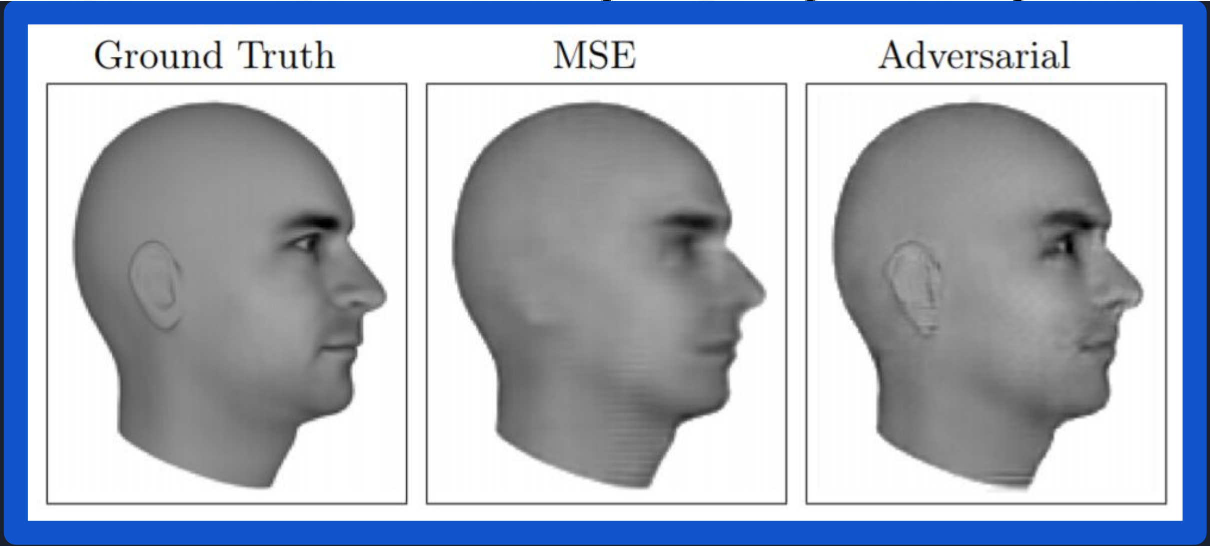
Part 1: GANs

Definition

- GANs are a class of **artificial intelligence algorithms**
- used in **unsupervised machine learning**
- implemented by a system of **two neural networks contesting** with each other
- in a **zero-sum game framework**.
- Minimax game based on Nash equilibrium.
- Introduced by Ian Goodfellow (currently at Tesla) et al. in 2014.

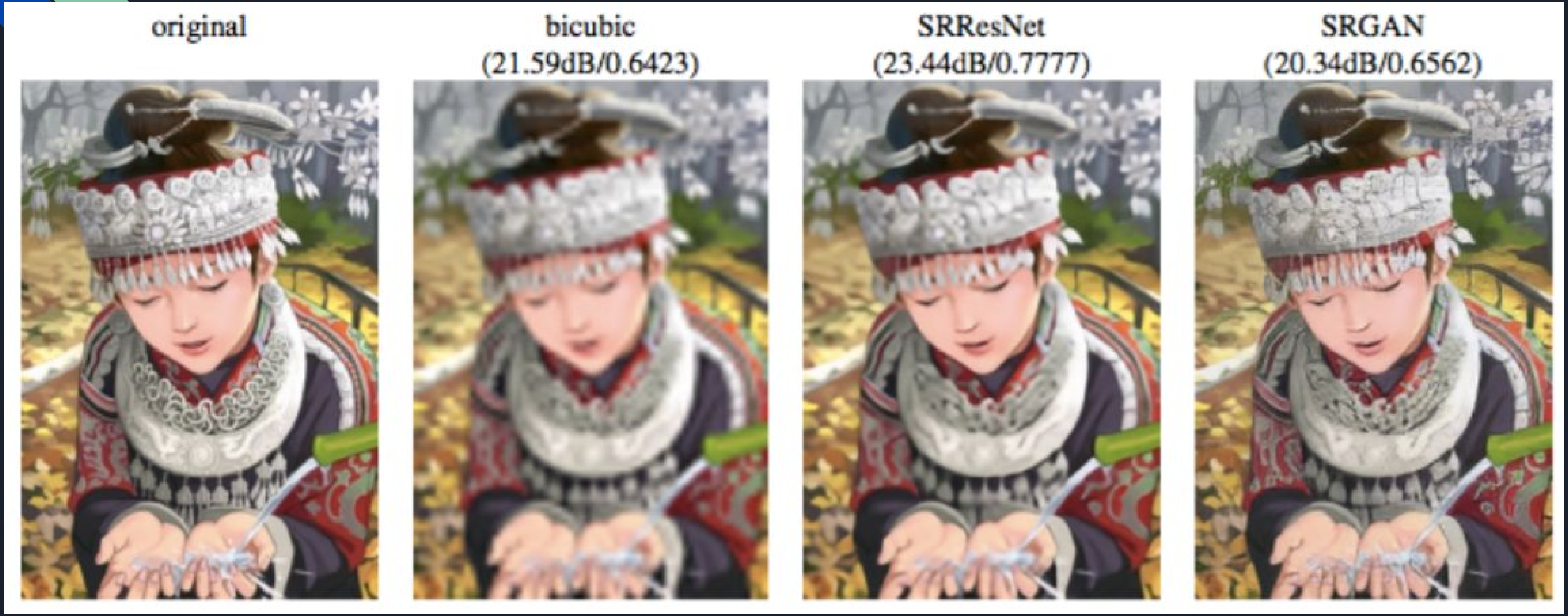
Next frame video prediction





Lotter et. al., 2016

Single image super resolution



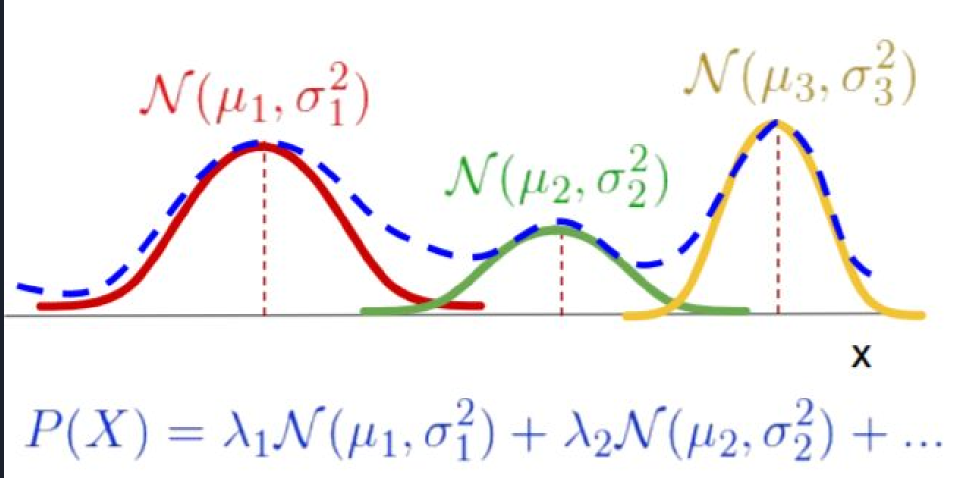
Ledig et. al., 2016

Why GANs?

- To augment data. Eg. generate new images from the existing ImageNet dataset.
- Seems to produce better samples faster.
- GANs are able to infer frames, upscale images better than existing techniques.
- Photorealistic images/reconstruct 3D models. In the extreme case, create photos/movies by itself!
- No Monte Carlo (MCMC) approximations required to train.
- The GAN framework can train any kind of generator net.
- No need to design the model need to obey any kind of factorization.
- Easier to use discrete latent variables.
- Goodfellow has discussed more specific advantages in his Quora answer.

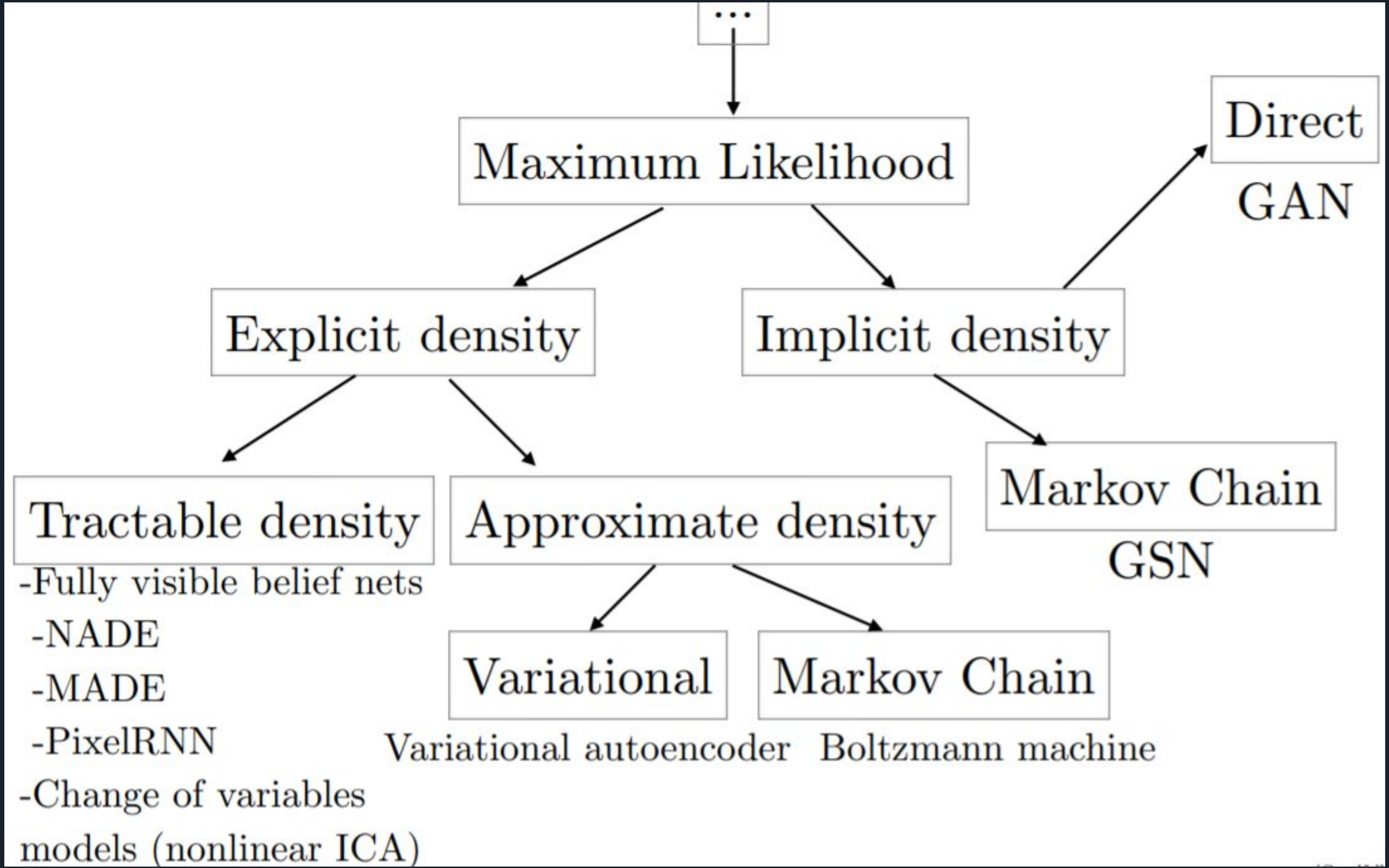
Generative model

- A model P(X; ϴ) from which we can draw samples.
- E.g. Gaussian Mixture Model (GMM)



- But GMMs are not complex enough to draw samples of images from it.

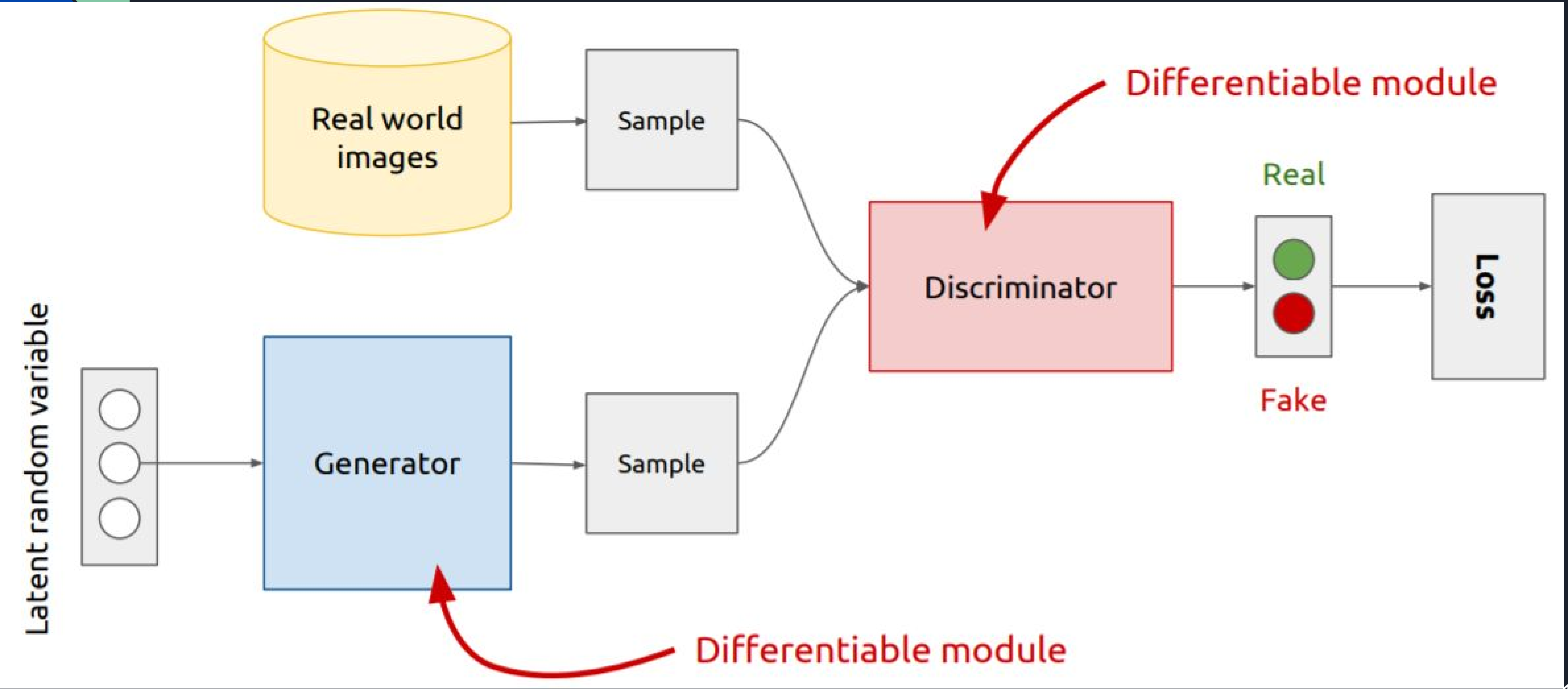
Taxonomy of Generative Models



Adversarial nets framework

- Generator (G): The **counterfeiter** who is trying to produce real data.
- Discriminator (D): The **cop** who is trying to identify which is the fake data.
- More technically, G tries to "trick" D by generating samples that are hard for D to distinguish from the real data.
- G: trained to maximize the probability of D making a mistake.
- D: trained to estimate the probability that a sample came from data distribution rather than G.

Conceptual diagram



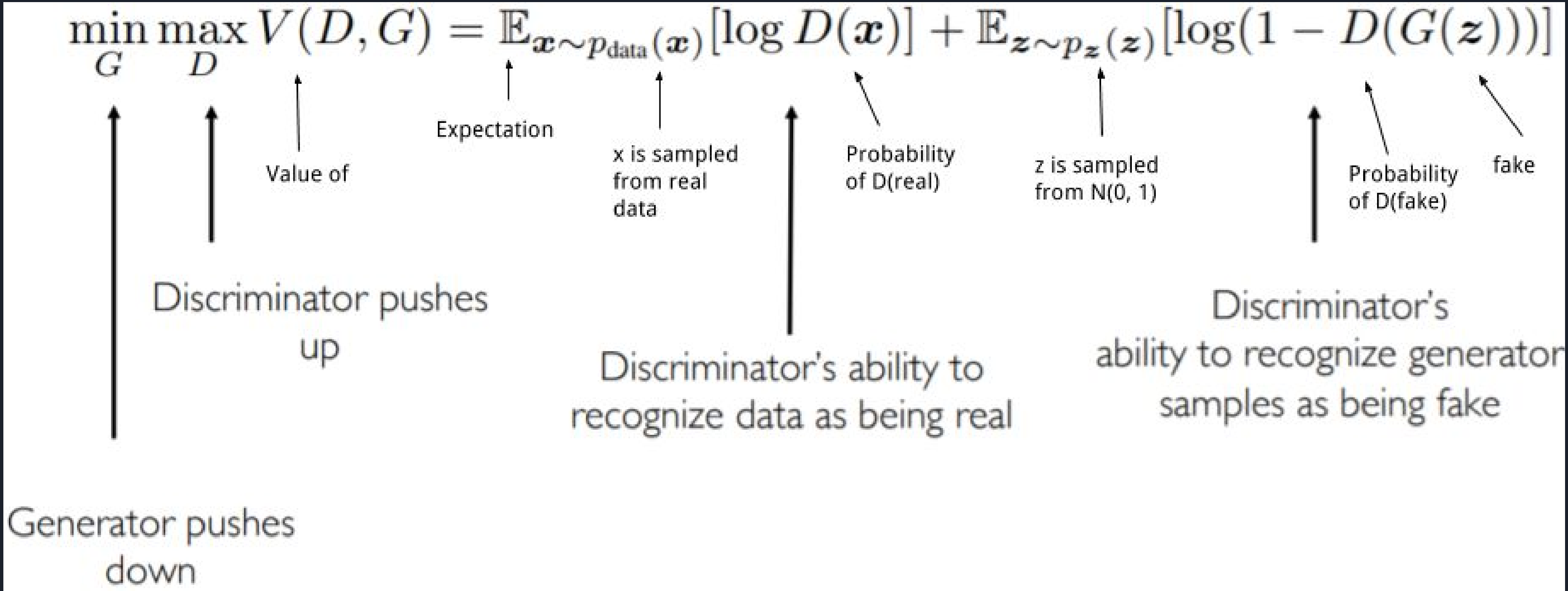
The Generator (G)

- Deterministic mapping from a latent random vector to sample from q(x) ~ p(x).
- Usually a deep neural network (DCGAN).

The Discriminator (D)

- Parameterised function that tries to distinguish between samples from real images p(x) and generated ones q(x).
- Usually a deep **convolutional** neural network (DCGAN).

The Objective Function



The Objective Function

- Change the objective from min log(1-D(G(z))) to max log(D(G(z))) to avoid saturating gradients early on when G is terrible.
- $p_z(z)$: Random noise injected to produce stochasticity in a physical system; typically a fixed uniform or normal distribution with some latent dimensionality.
- For G fixed, the optimal discriminator D is:

$$D^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

  Because, the function *a log(y) + b log(1-y)* achieves its maximum in [0, 1] at a/(a+b)

Refer original Goodfellow paper for all original GAN theory with derivations: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (NIPS 2014)

The Algorithm

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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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.

GAN Training

1. Fix generator weights, draw samples from both real world and generated images.
2. Train discriminator to distinguish between real world and generated images.
3. Fix discriminator weights.
4. Sample from generator.
5. Backprop error through discriminator to update generator weights.

- Iterate until convergence. It is our hope that the generator gets so good that it is impossible for the discriminator to tell the difference between real and generated images.

GAN Training

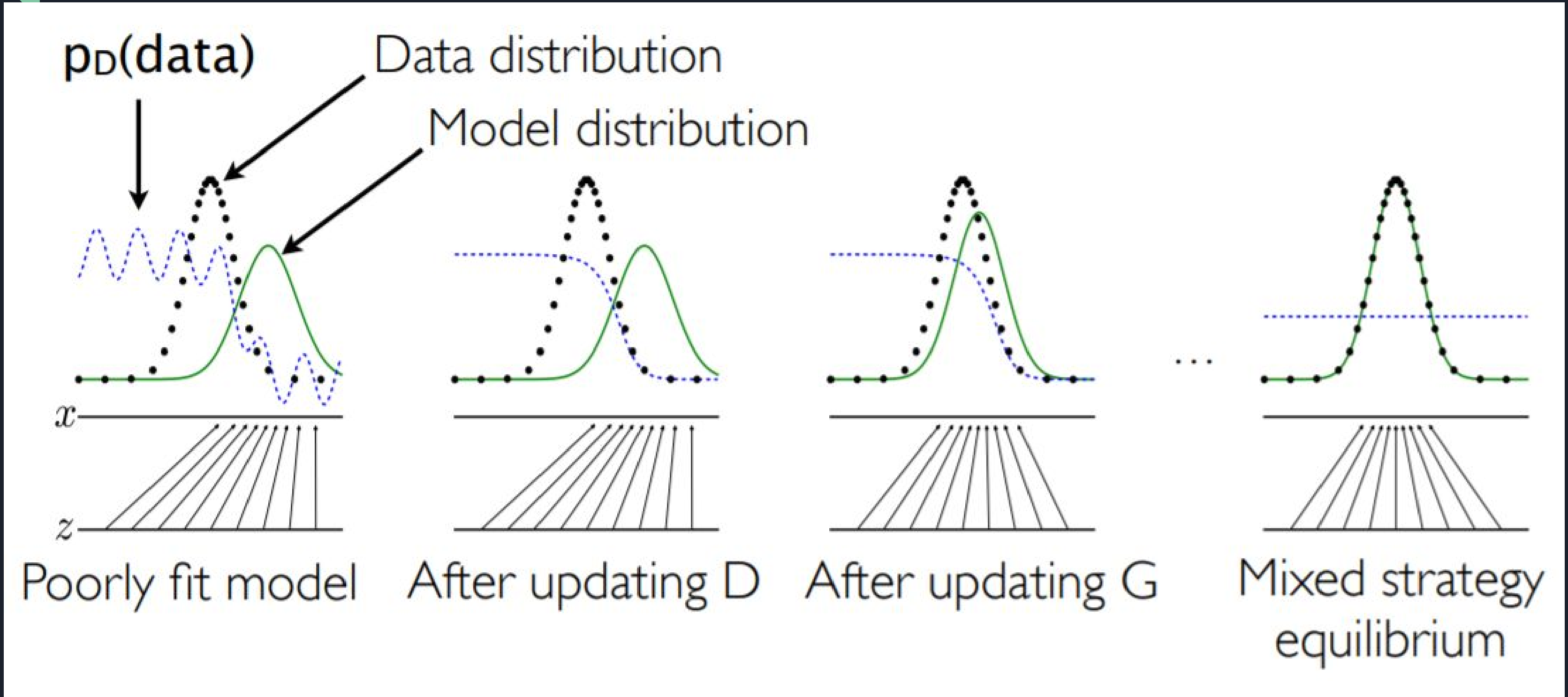
- Training GAN is equivalent to minimizing the Jensen-Shannon divergence (symmetrized and smoothed version of the Kullback–Leibler divergence) between generator and data distributions.

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

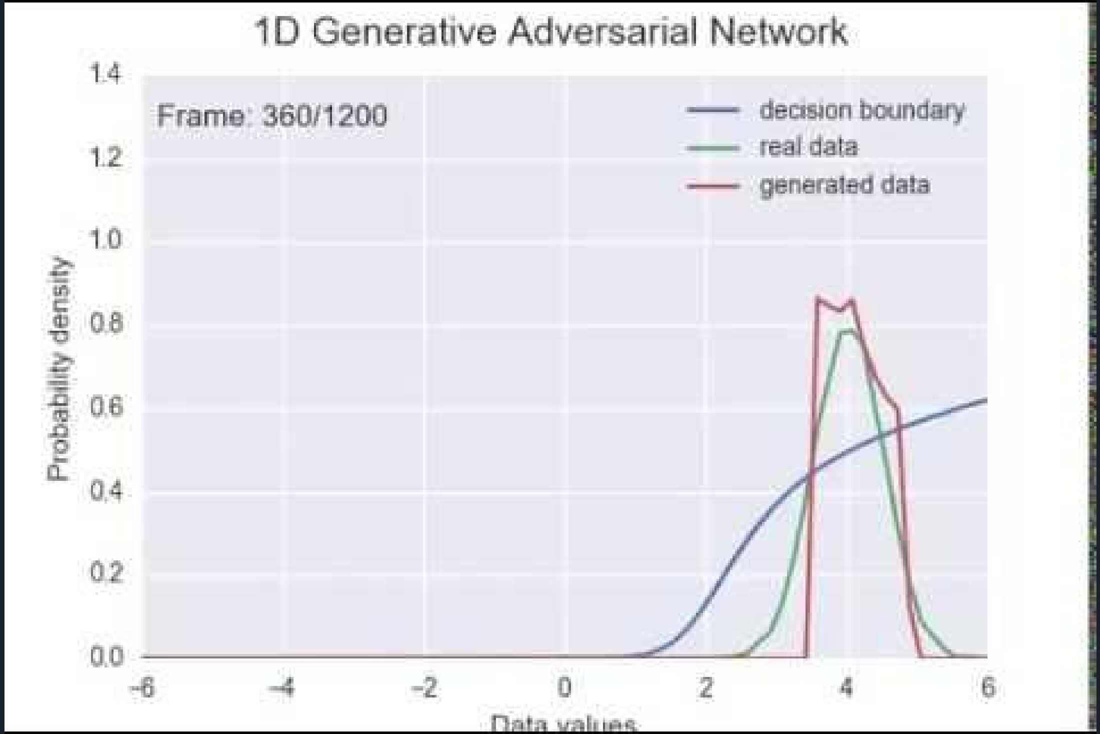
where $M = \frac{1}{2}(P + Q)$

- Updating the discriminator should make it better at **discriminating** between real images and generated ones.
- Updating the generator makes it better at **fooling** the current discriminator.

The Learning Process



The Learning Process

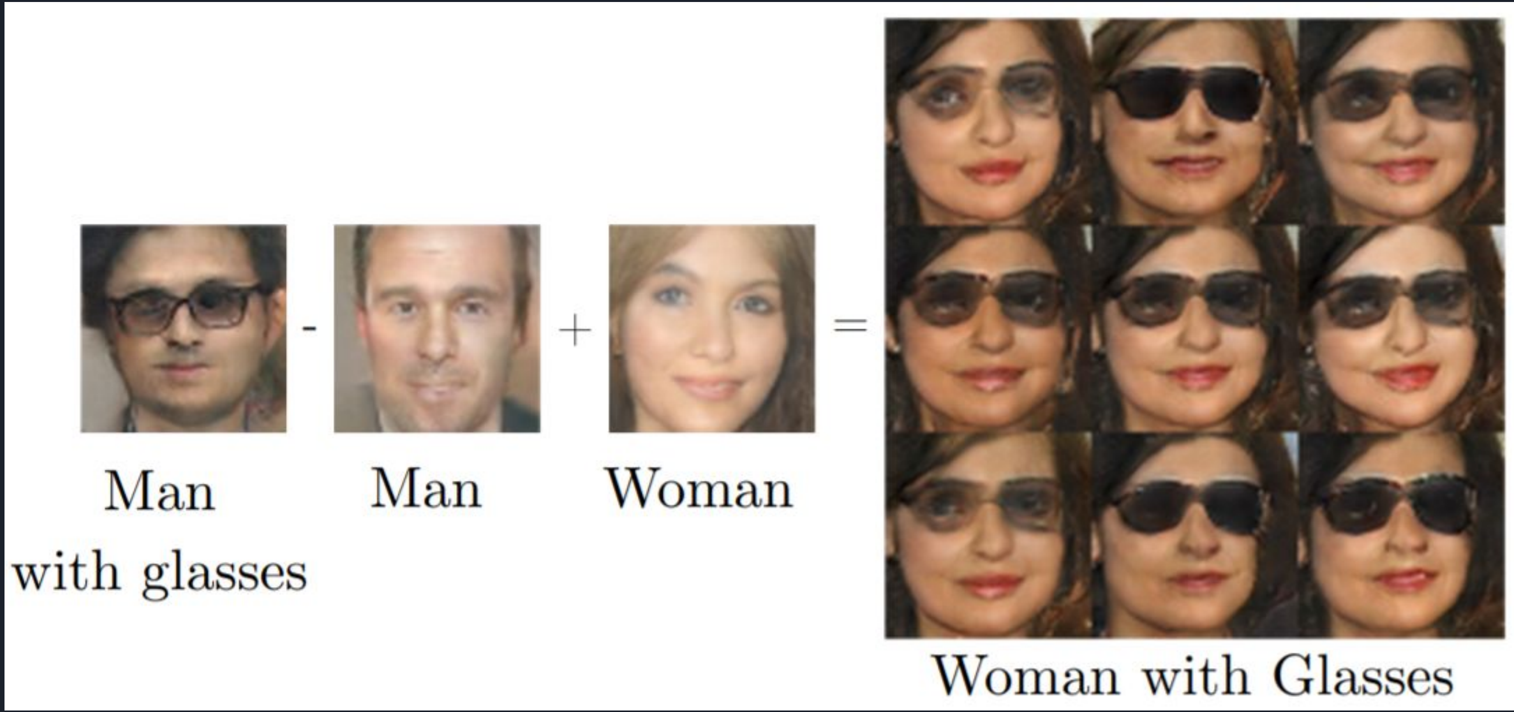


So what do we get after training?

- D: Trained as an unsupervised “**density estimator**”, i.e. a contrast function that gives us a low value for data and higher output for everything else.
  - D develops a good internal representation of the data.
  - Can be used as a **feature extractor for a classifier**, for example.

- G: **Parametrizes the complicated surface** of real data.
  - Arithmetic on faces in the Z vector space: [man with glasses] - [man without glasses] + [woman without glasses] = [woman with glasses].

Face Arithmetic



Radford et. al., 2015

Issues

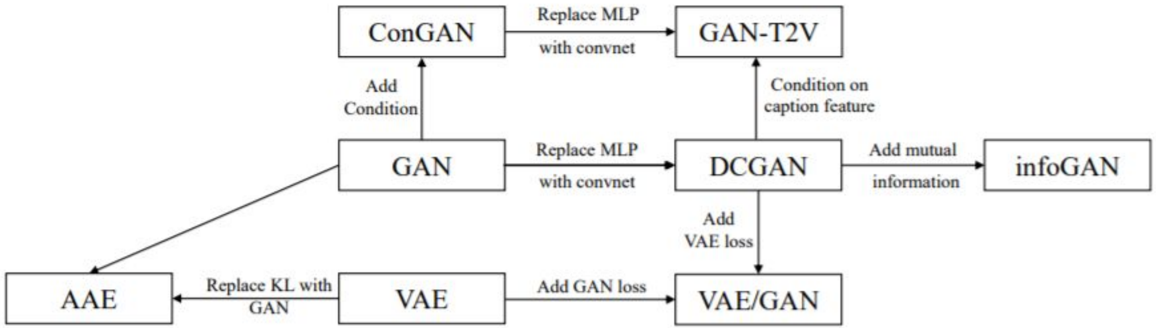
- Hard to train (immature tools for minimax optimization).
- Unstable dynamics: hard to keep generator and discriminator in balance. Generator can collapse. Need to babysit during training.
- Optimization can oscillate between solutions. Easy to get trapped in local optima that memorize training data.
- Unclear stopping criteria.
- No explicit representation of $p_g(x)$.
- No evaluation metric so hard to compare with other models.
- Hard to invert generative model to get back latent z from generated x.

Do read this if you wish to train a GAN: https://github.com/soumith/ganhacks

Convergence issues

- GANs don’t always converge. Especially difficult for large problems.
- A Game Theory paper titled “*Characterization and Computation of Local Nash Equilibria in Continuous Games*” by Ratliff et. al. gives some conditions under which simultaneous gradient descent on two player’s costs will converge but GANs never satisfy those conditions because the **Hessian of the generators costs is all zeros at equilibrium**.
- The conditions mentioned however are not *necessary* conditions, thus GANs can converge sometimes.

GAN Variations



GAN, DCGAN: $\mathcal{L}_{GAN} = \mathbb{E}_{x\sim P_{data}}(\log D(x)) + \mathbb{E}_{z\sim p(z)}(\log D(1 - G(z)))$

ConGAN, GAN-T2I: $\mathcal{L} = \mathbb{E}_{x\sim P_{data}}(\log D(x|y)) + \mathbb{E}_{z\sim p(z)}(\log(1 - D(G(z|y)|y))$

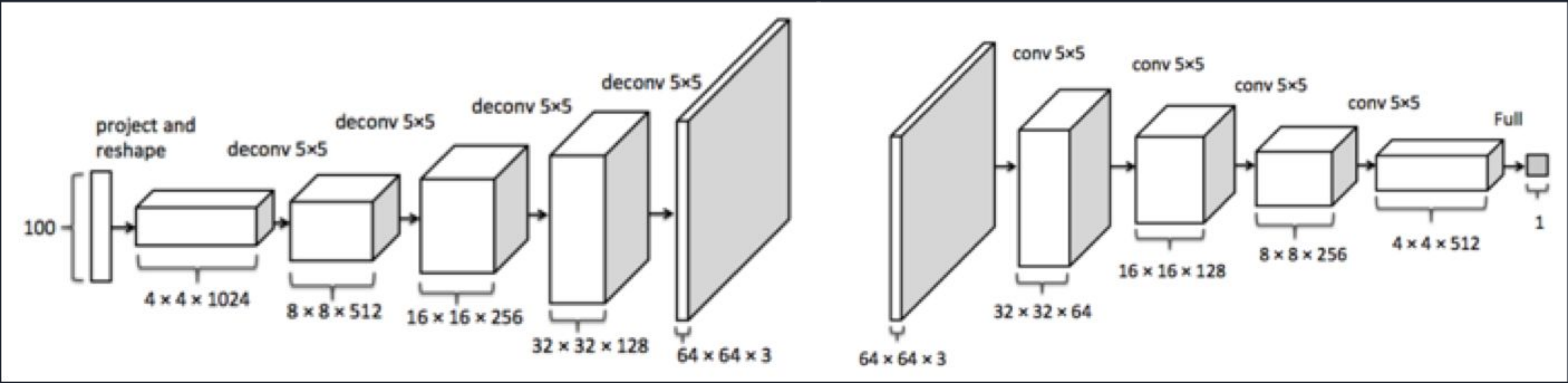
Info GAN: $\mathcal{L} = \mathcal{L}_{GAN} +$ Latent code reconstruction

VAE/GAN: $\mathcal{L} = \mathcal{L}_{GAN} +$ Representation reconstruction

AAE: $\mathcal{L} =$ Reconstruction $+\mathcal{L}_{GAN}$

DCGAN

- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Network (DCGAN).
- Most GANs today are at least loosely based on the DCGAN architecture (Radford et al., 2015).



DCGAN layers

- Use deep CNN for generator and discriminator instead of MLP.
    - Replace any pooling layers with strided convolution.
    - Use batchnorm in both the generator and the discriminator.
    - Remove fully connected hidden layers for deeper architectures.
    - Uses Tanh for the output (and sigmoid).
    - Use **Leaky ReLU** in the discriminator and ReLU in the generator.
- Use the trained discriminators for image classification tasks.

Part 2: Coding vanilla GANs (using tf)

https://github.com/wiseodd/generative-models/blob/master/GAN/vanilla_gan/gan_tensorflow.py

Imports

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist
import input_data
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import os
```

Defining the discriminator weights and biases

```
X = tf.placeholder(tf.float32, shape=[None, 784])

D_W1 = tf.Variable(xavier_init([784, 128]))

D_b1 = tf.Variable(tf.zeros(shape=[128]))

D_W2 = tf.Variable(xavier_init([128, 1]))

D_b2 = tf.Variable(tf.zeros(shape=[1]))

theta_D = [D_W1, D_W2, D_b1, D_b2]
```

Defining the generator weights and biases

```
Z = tf.placeholder(tf.float32, shape=[None, 100])

G_W1 = tf.Variable(xavier_init([100, 128]))

G_b1 = tf.Variable(tf.zeros(shape=[128]))

G_W2 = tf.Variable(xavier_init([128, 784]))

G_b2 = tf.Variable(tf.zeros(shape=[784]))

theta_G = [G_W1, G_W2, G_b1, G_b2]
```

What is Xavier Initialization?

$$\mathrm{Var}(W) = \frac{1}{n_{\text{in}}}$$

- Initialize the weights in the network by drawing them from a distribution with zero mean and a specific variance (shown above)
  - n_{in} is the number of neurons feeding into the neuron
  - n_{out} is the number of neurons the result is fed to.
- Glorot & Bengio's paper originally recommended using $$\mathrm{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$
- But we don't usually use the original recommendation as:
  - Preserving the forward-propagated signal is much more important.
  - Difficult to find out how many neurons in the next layer consume the output of the current one.

What is Xavier Initialization?

- It helps signals reach deep into the network.
- If the weights in a network start too small, then the signal shrinks as it passes through each layer until it’s too tiny to be useful.
- If the weights in a network start too large, then the signal grows as it passes through each layer until it’s too massive to be useful.

Z

```
def sample_Z(m, n):
    return np.random.uniform(-1., 1., size=[m, n])
```

D

```
def discriminator(x):
    D_h1 = tf.nn.relu(tf.matmul(x, D_W1) + D_b1)
    D_logit = tf.matmul(D_h1, D_W2) + D_b2
    D_prob = tf.nn.sigmoid(D_logit)
    return D_prob, D_logit
```

G

```
def generator(z):
    G_h1 = tf.nn.relu(tf.matmul(z, G_W1) + G_b1)
    G_log_prob = tf.matmul(G_h1, G_W2) + G_b2
    G_prob = tf.nn.sigmoid(G_log_prob)
    return G_prob
```

Value definition

```
G_sample = generator(Z)

D_real, D_logit_real = discriminator(X)

D_fake, D_logit_fake = discriminator(G_sample)


# D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))

# G_loss = -tf.reduce_mean(tf.log(D_fake))
```

Discriminator Loss functions

```
D_loss_real =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_logit
_real, labels=tf.ones_like(D_logit_real)))

D_loss_fake =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D_logit
_fake, labels=tf.zeros_like(D_logit_fake)))

D_loss = D_loss_real + D_loss_fake
```

Generator Loss function

```
G_loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits
=D_logit_fake, labels=tf.ones_like(D_logit_fake)))
```

Adam Optimizer

```
D_solver = tf.train.AdamOptimizer().minimize(D_loss,
var_list=theta_D)

G_solver = tf.train.AdamOptimizer().minimize(G_loss,
var_list=theta_G)
```

What is Adam Optimizer?

- Best optimizer currently present. Best replacement for SGD.
- Estimates 1st-order moment (the gradient mean) and 2nd-order moment (element-wise squared gradient) of the gradient using exponential moving average, and corrects its bias.
- Combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.
- Relatively easy to configure.

Reference: https://arxiv.org/pdf/1412.6980.pdf (ICLR 2015, Kingma & Ba)

Define sizes & take input

```
mb_size = 128

Z_dim = 100

mnist = input_data.read_data_sets('../MNIST_data',
one_hot=True)
```

Initialize session

```
sess = tf.Session()

sess.run(tf.global_variables_initializer())
```

Iterations

```
for it in range(1000000):
    if it % 1000 == 0:
        samples = sess.run(G_sample, feed_dict={Z: sample_Z(16, Z_dim)})
        fig = plot(samples)

    X_mb, _ = mnist.train.next_batch(mb_size)

    _, D_loss_curr = sess.run([D_solver, D_loss], feed_dict={X:
    X_mb, Z: sample_Z(mb_size, Z_dim)})

    _, G_loss_curr = sess.run([G_solver, G_loss], feed_dict={Z:
    sample_Z(mb_size, Z_dim)})
```

Results

Part 3: My AAAI+NIPS work

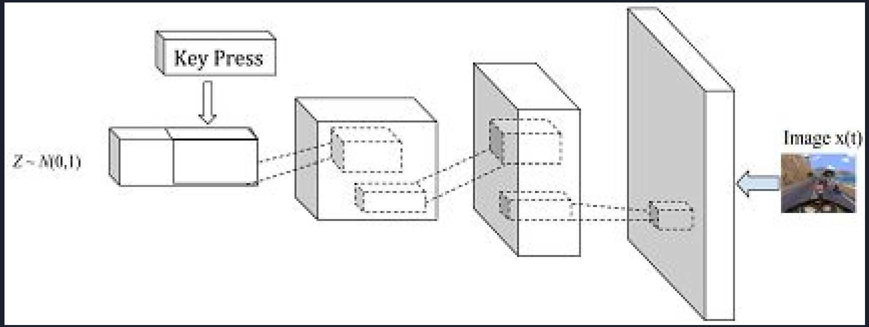
AAAI 2017: Handwriting Profiling using GANs (Hand-GAN)

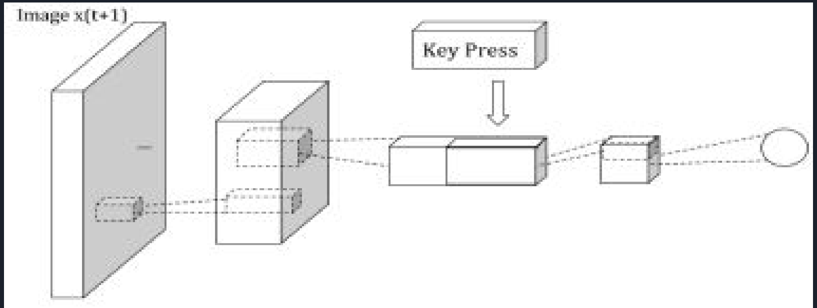
- System tries to learn the handwriting of an entity.
- Generate letter strokes which were not previously seen before.
- Used a modified architecture of DCGAN (Radford, Metz, and Chintala 2015).
- Used Reinforcement Learning to learn spacing, strokes and inflections – rewards and penalties to make the generator learn.
- Data: MNIST and survey handwriting.
- Useful for Identification of forged documents, signature verification, computer generated art, digitization of documents.

NIPS 2016: Synthetic Autonomous Driving using GANs (SAD-GAN)

- To make a controller trainer network using images plus key press data to mimic how a human learns driving.
- Used DCGAN.
- Built a keylogger software to automatically generate datasets by playing RoadRash races. Each race generated around 500 usable images. Played around 200 races!
- Trained the model on one video game (RoadRash) and compared the accuracy by running the model on other maps (GTA etc.) to determine the extent of learning.

NIPS 2016: Synthetic Autonomous Driving using GANs (SAD-GAN)





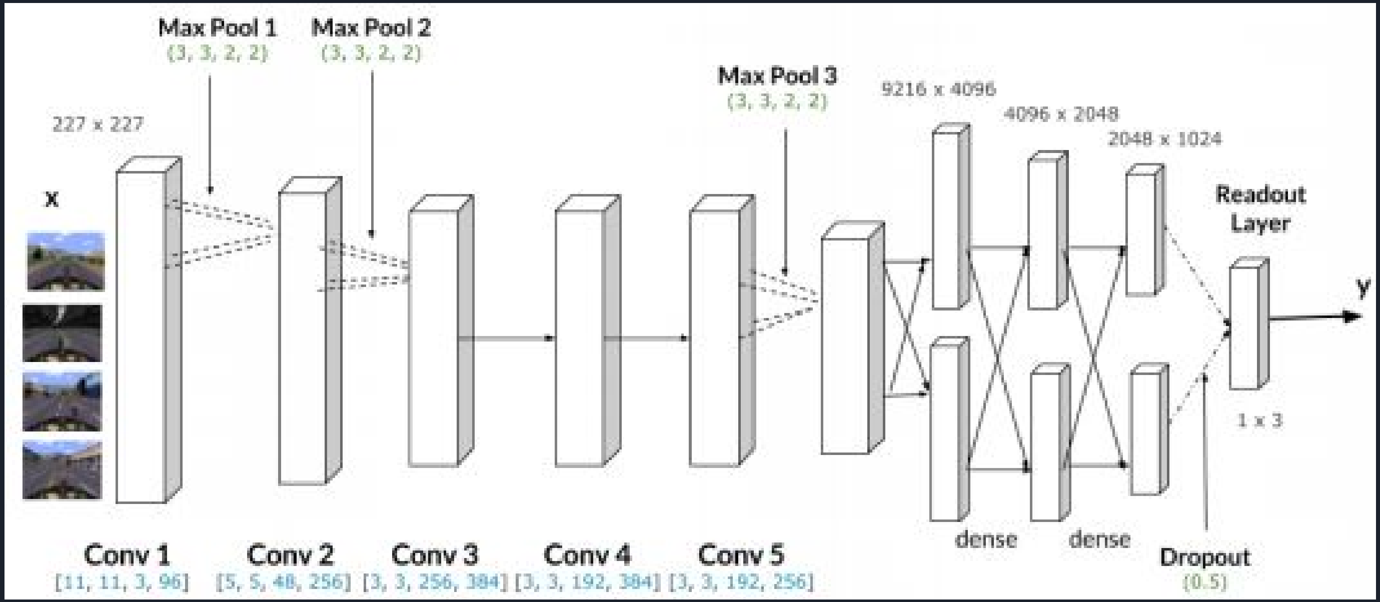
Generator

- Receives Keypress + Input image at time t + Noise
- Tries to estimate image at t+1

Discriminator

- Receives generator image and actual t+1 image
- Tries to guess the correct image
- During training, it has created a feature map (classifier)

NIPS 2016: Synthetic Autonomous Driving using GANs (SAD-GAN)



AlexNet - Inputs: actual images at t and t+1, Output: Key pressed by a human who is expected to drive safely

NIPS 2016: Synthetic Autonomous Driving using GANs (SAD-GAN)

- G: Train to predict next image given current image and key press.
- D: Distinguish between dataset images and images generated by G.
- After reasonable efficiency is achieved in G, it is used to predict all three images (pressing left, right and up arrow keys) from the given image.
- The three images are classified as “safe” and “unsafe” (by the AlexNet). If “safe”, go down the game tree. If “unsafe”, choose another option (image).
- The metric for reinforcement learning is set as the maximum number of levels down the game tree the decision yields a safe scene.

NIPS 2016: Synthetic Autonomous Driving using GANs (SAD-GAN)

Other interesting GAN research

- **EBGAN**: Energy input instead of probability distributions (2016, Zhao et. al.)
- **Generative Adversarial Metric (GAM)**: Compare performance by judging each generator under the opponent's discriminator. (2016, Im et. al.)
- **GMAN: Generative Multi-Adversarial Networks**. Modifies GAM to evaluate multiple adversaries. (ICLR 2017, Durugkar et. al.)
- So much more interesting stuff done, requires more talks!

Some fun GAN articles

- Learn GANs with Spongebob! (DCGAN with Tensorflow code): https://medium.com/@awjuliani/generative-adversarial-networks-explained-with-a-classic-spongebob-squarepants-episode-54deab2fce39
- Abuse GANs to make 8-bit pixel art: https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

GAN Conclusion

- Not a magic solution to everything! (yet)
- Concept is relatively easy to understand, but training is a challenge.
- Open questions: Does an equilibrium exist where G wins (D loses)? Is the p_g really close to p_{real} (meaningful generation or simply memorization)?
- GANs are useful mainly for image datasets. It has been especially successful in text to image transformation and synthetic driving applications.
- Game theoretical aspect of GANs has not been explored adequately. https://arxiv.org/pdf/1703.00573.pdf (Paper from Princeton Theory group, Sanjeev Arora et. al., revised August 2017, ICML 2017)

Other Selected Research

- **Contextual Analytics Personal Assistant**: Used Adobe Analytics usage data to model user intent and behavior to generate recommendations and work as a personal assistant to users. 1 US patent pending + 1 research paper.

- **AI for electrical power grids**: Fault analysis and subset selection for optimal economic dispatch using AI (deep learning) techniques. 2 research papers.

- **Location Optimization of ATM networks**: Where to place an ATM given demographic information so as to maximize profit and usability. 1 research paper.

(Above papers are in submitted/drafting phase)

References

1. Robin Ricard: http://www.rricard.me/machine/learning/generative/adversarial/networks/2017/04/05/gans-part1.html
2. The NIPS 2014 paper: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
3. Goodfellow talk: http://www.cs.toronto.edu/~dtarlow/pos14/talks/goodfellow.pdf
4. NIPS 2016 tutorial: https://arxiv.org/pdf/1701.00160.pdf
5. Kevin McGuinness: http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D4L1-adversarial.pdf
6. GAN and its variations: http://people.ee.duke.edu/~lcarin/Yunchen9.30.2016.pdf
7. GAN Foundations: https://www.cs.toronto.edu/~duvenaud/courses/csc2541/slides/gan-foundations.pdf

Thanks!

Contact me:

SAL 300
USC

bbhattac@usc.edu
https://biswarupb.github.io