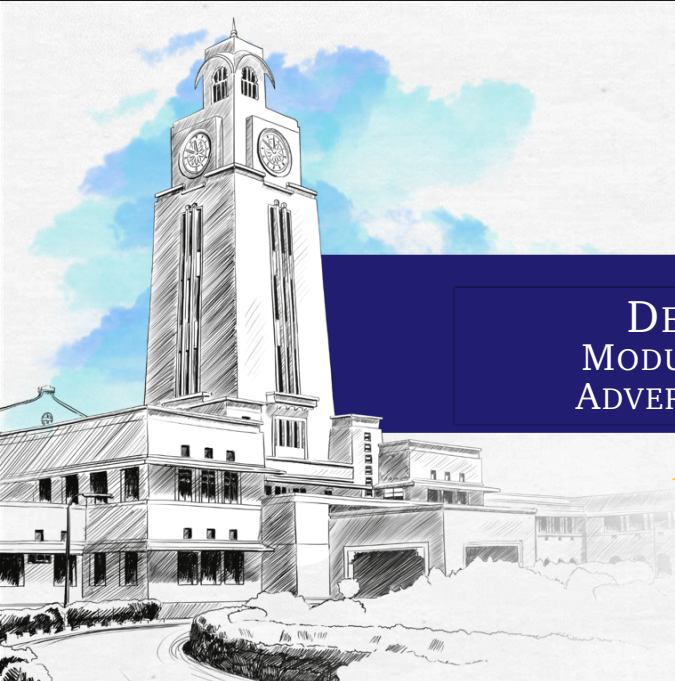


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Learning
Module 8: Generative Adversarial Networks

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

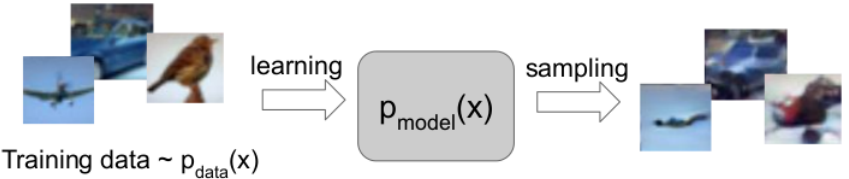
Machine Learning Algorithms

- Discriminate between categories.
- Also known as classifiers.
- Set of features X.
- Set of categories Y.
- Mapping from features to categories i.e. $X \rightarrow Y$
- Model categories Y given X i.e. $P(Y \mid X)$.

- Learn realistic representation of a class.
- Random input or Noise ξ and a set of class Y.
- Generate a set of features X that will realistically represent the class Y.
- Mapping from categories to features i.e. $\xi, Y \rightarrow X$
- Model features X given Y i.e. $P(X \mid Y)$.

Generative Models

Given training data, generate new samples from same distribution

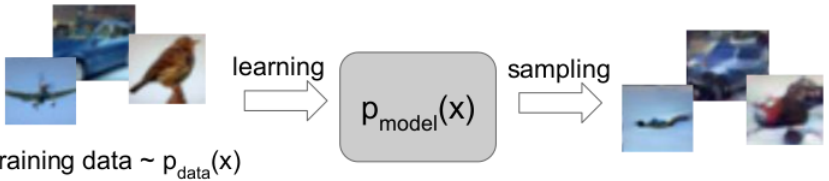


Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. **Sampling new x from $p_{model}(x)$**

Generative Models

Given training data, generate new samples from same distribution



Formulate as density estimation problems:

- **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.**

Generative Models

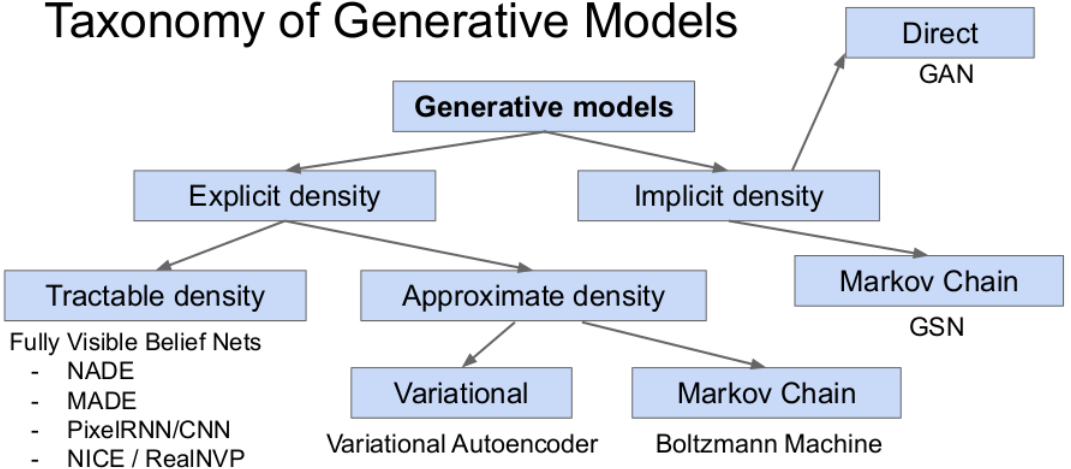


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Models

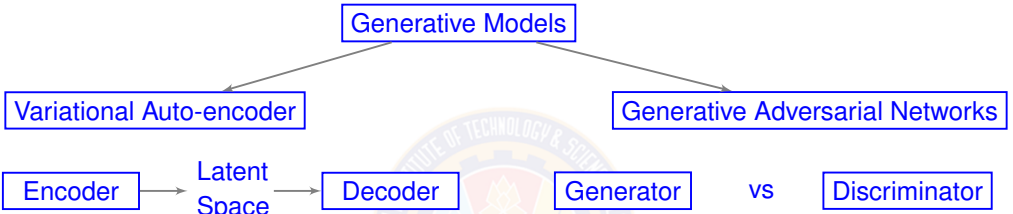
Explicit density

- Explicit generative models specify the form of the data distribution and have tractable likelihoods.
- use an explicit density function
- assume some prior distribution about the data.

Implicit density

- Implicit generative models specify a stochastic process to simulate data.
- do not use an explicit density function

Generative Models



- Encoder and Decoder work together.
- Once training is over, remove the encoder.
- Decoder will reproduce a realistic image corresponding to the selected random point.

- Competition between Generator and Discriminator. So called Adversarial.
- Once training is over, remove the Discriminator.
- Generator will reproduce a realistic image.

Generative Models

Variational Auto-encoder (VAE)

- Train Encoder and Decoder such that the latent space (distribution) is learned.
- Once training is over, remove the encoder.
- Sample / Pick random point in latent space (distribution).
- Decoder will reproduce a realistic image corresponding to the selected random point.

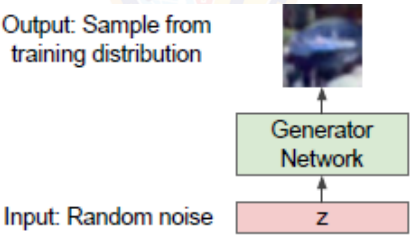
Generative Adversarial Networks (GAN)

- Generator is similar to the Decoder of VAE.
- Input to the generator is a noise vector.
- Discriminator will determine which image is real and fake.
- Competition between Generator and Discriminator. So called Adversarial.
- After training, the Generator will output realistic images.

In This Segment

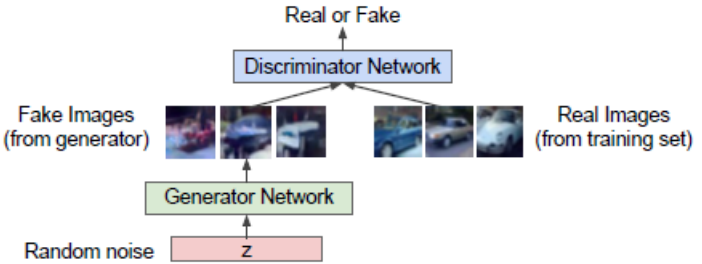
Generative Adversarial Networks (GAN)

- A generative model
- GANs are composed of two models - Generator and Discriminator.
- Generator and Discriminator compete with each other in order to produce realistic images.
- Learn to generate from training distribution through 2-player game theory technique.



Generative Adversarial Networks (GAN)

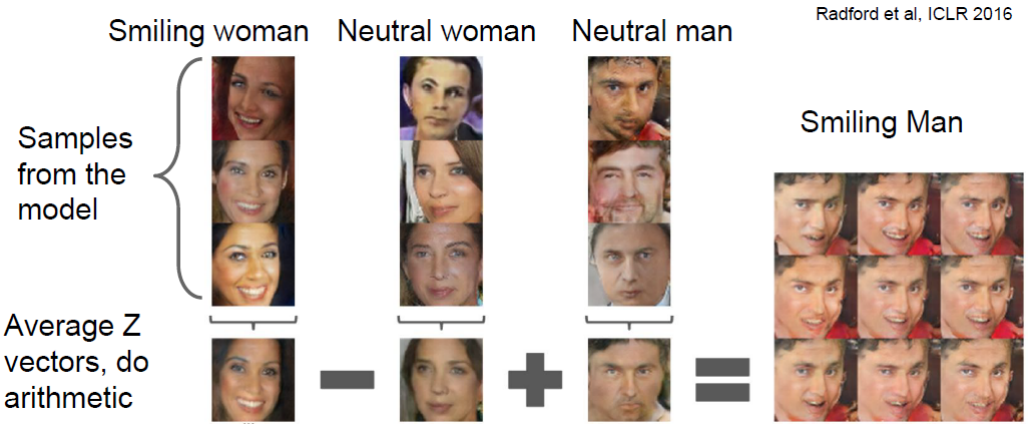
- Generator network
  - Try to fool the discriminator by generating real-looking images.
- Discriminator network
  - Try to distinguish between real and fake images.
- Both are trained simultaneously in minimax game.



Fake and real images copyright Emily Denton et al. 2015.

In This Segment

GAN - Interpretable Vector Math



GAN - Interpretable Vector Math

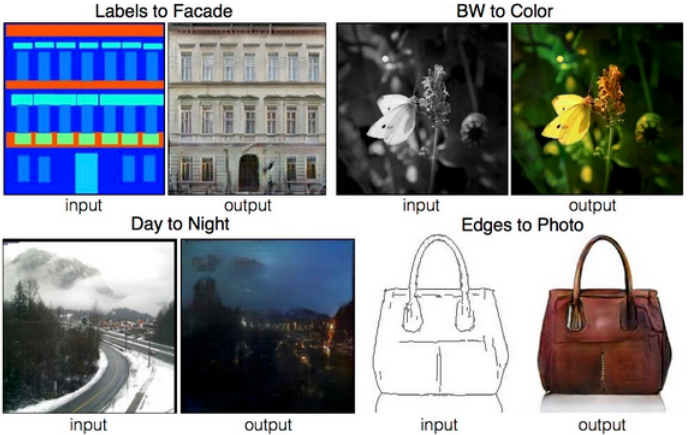
GAN - Image-to-Image Translation

Background masking

Kaihu Chen performed a number of interesting experiments using our method, including getting it to mask out the background of a portrait as shown above.

phillipi.github.io/pix2pix/

GAN - Image-to-Image Translation



phillipi.github.io/pix2pix/

GAN - Text-to-Image Translation

this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.

the flower has petals that are bright pinkish purple with white stigma

this white and yellow flower have thin white petals and a round yellow stamen

Reed et al. 2017.

GAN - Source-Target Domain Transfer

CycleGAN. Zhu et al. 2017.

Real Life GANs

- StyleGAN2 – Face generation
  `thispersondoesnotexist.com`
  `thesecatsdonotexist.com`
- CycleGAN – Image translation
- GauGAN – Photo realistic drawing
- 3DGAN – 3D objects

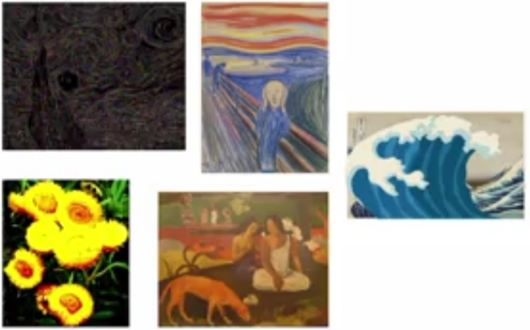
In This Segment

Generator vs Discriminator

- Generator learns to make **fakes** that look like real.
- Discriminator learns to distinguish **real** from fake.

Generator vs Discriminator

Generator learns to make *fakes* that look real

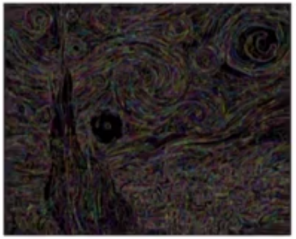
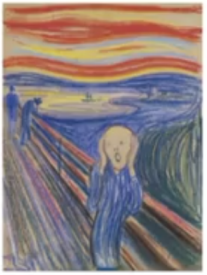


Discriminator learns to distinguish real from *fake*

Generator

- Generator does not see real images.
- Generator is input with some random noise.
- Generator generates some elementary sketches.

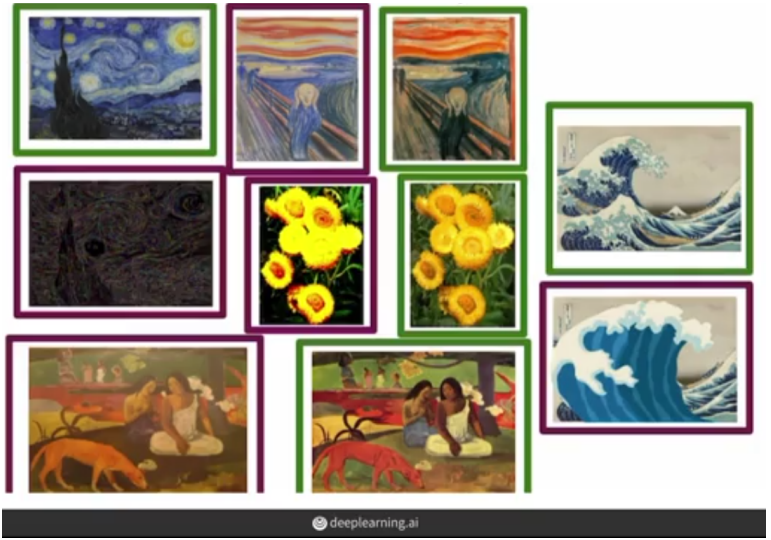
Generator

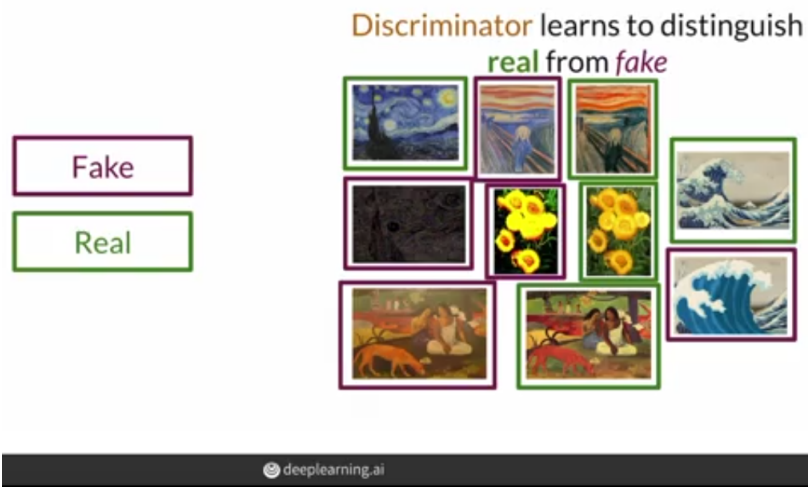

Discriminator

- Discriminator is allowed to see real images.
- It does not which images are real or which are fake.
- Discriminator is able to differentiate sketches from real images.

Discriminator



Discriminator



Generator vs Discriminator

Discriminator

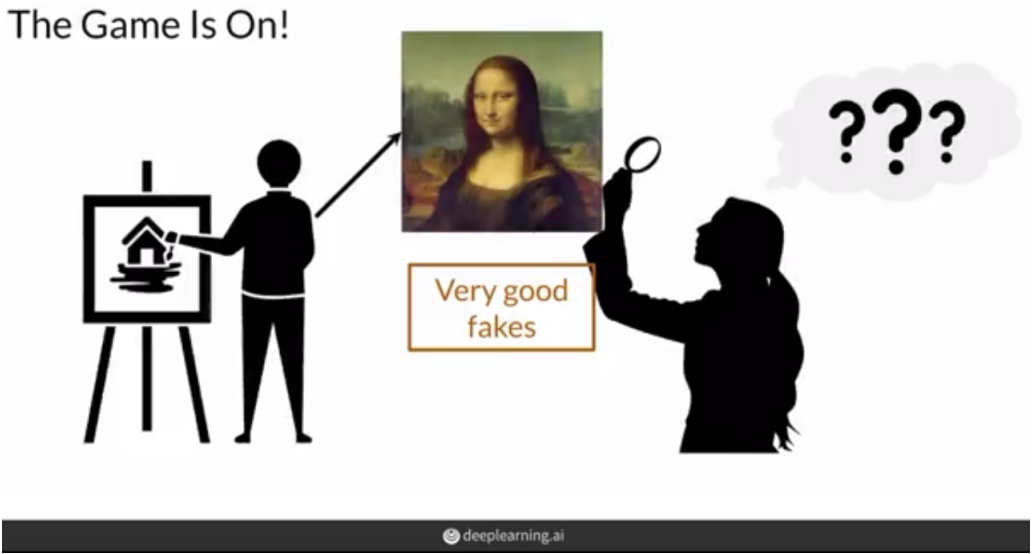
- eg: Image created by generator.
- Discriminator says that it is 60% real.
- You say it is wrong. It is fake.
- Continue training.

Generator vs Discriminator

Generator vs Discriminator

- After many rounds of training, the generator starts to produce images that are harder and harder to distinguish.
- Generator can produce awesome fake images (realistic images).
- The game ends when it becomes very difficult for the discriminator to distinguish real from fake.

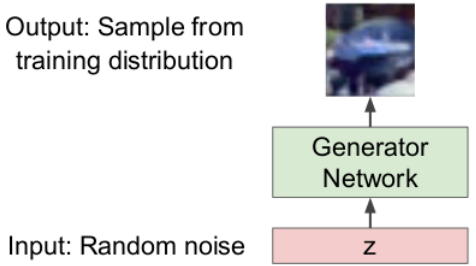
Generator vs Discriminator



GAN

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

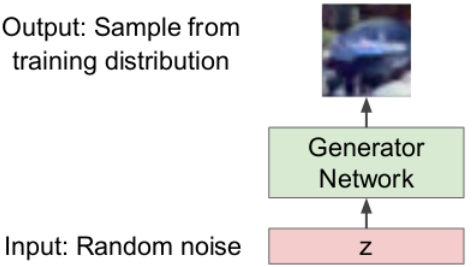


GAN

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images



GAN

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution

Objective: generated images should look "real"

Generator Network

Input: Random noise

z

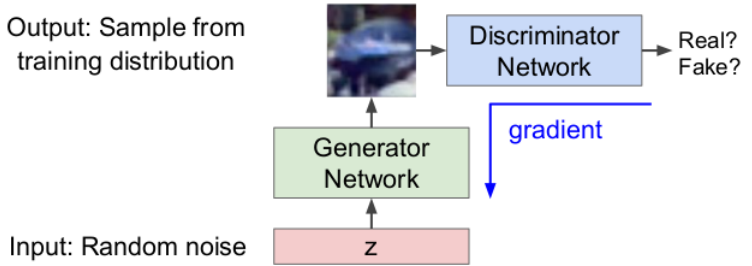
GAN

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

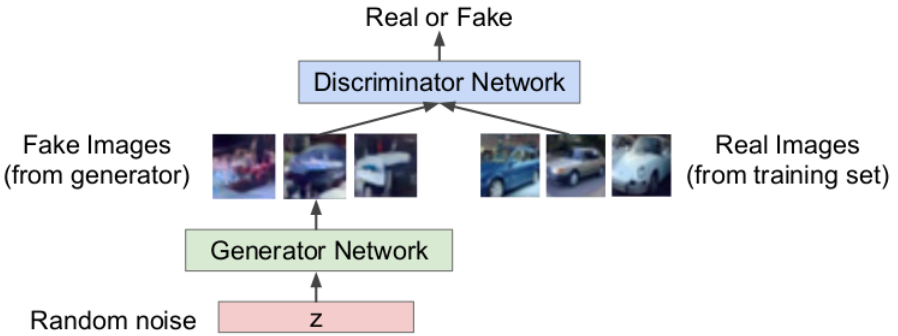
But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Solution: Use a discriminator network to tell whether the generate image is within data distribution ("real") or not



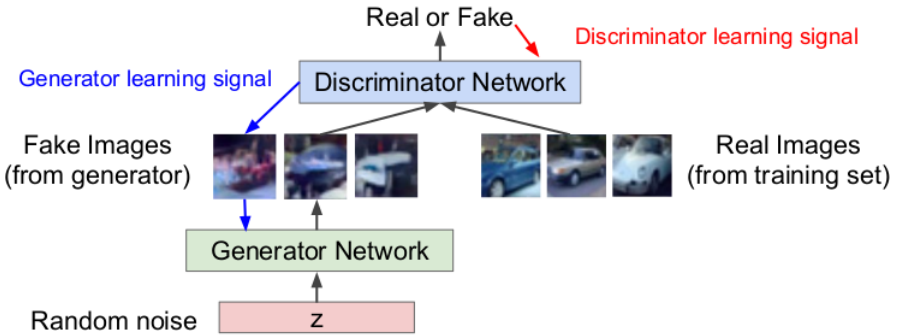
GAN

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



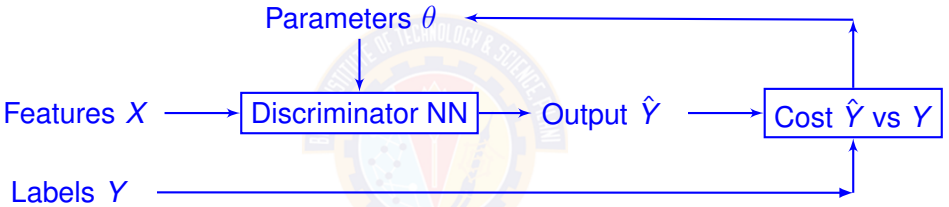
GAN

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



Discriminator

- Discriminator is Classifier.
- Implemented as a neural network.



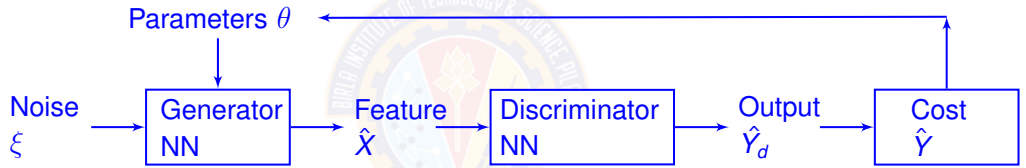
- Goal: Minimize the difference between $\hat{Y}$ and Y.
- Learns conditional probability of class Y (real or fake) given the features X.

$$\textit{capture} \quad P(Y \mid X)$$

- Probabilities are feedback to the generator.

Generator

- Generator generates examples of a class given some noise vector.
- Implemented as a neural network.



- Generate fake images.
- Learns the probability of features X.
- Model features X conditioned on the class Y.

$$\textit{capture} \quad P(X \mid Y)$$

Training GAN

- θ_g – parameters in the generator
- θ_d – parameters in the discriminator
- Discriminator output for real data $x = D_{\theta_d}(x)$
- Discriminator output for fake data $G(z) = D_{\theta_d}(G_{\theta_g}(z))$
- Discriminator wants to maximize objective such that $D(x) \approx 1$ and $D(G(z)) \approx 0$.
- Generator wants to minimize objective such that $D(G(z)) \approx 1$.
- After training, use Generator to generate new images.

Training GAN

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Generator objective

Discriminator objective

Training GAN

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

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

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GAN

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

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

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GAN

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

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

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GAN

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

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

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

- 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)

Training GAN

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GAN

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

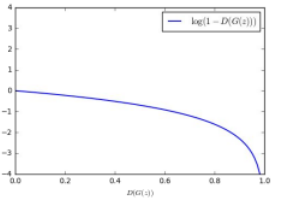
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).



Training GAN

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

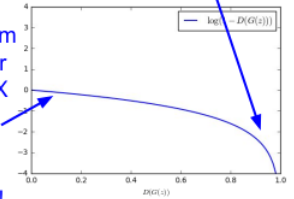
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GAN

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

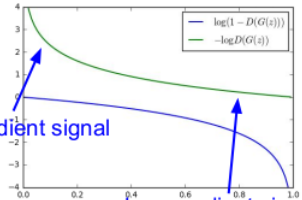
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Training GAN

Putting it together: GAN training algorithm

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Training GAN

Putting it together: GAN training algorithm

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

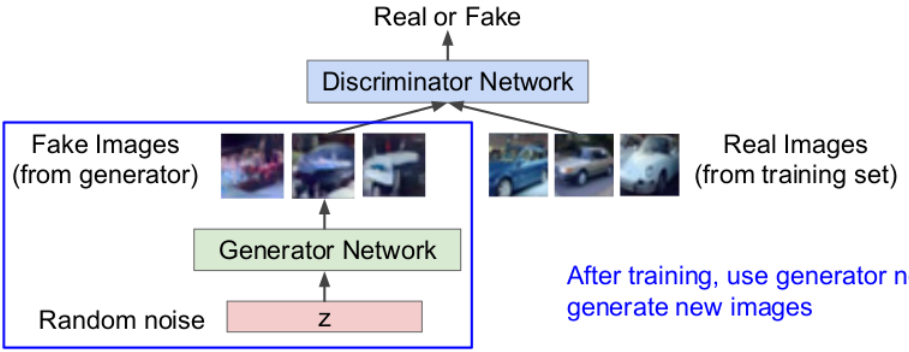
end for

Some find k=1 more stable, others use k > 1, no best rule.

Followup work (e.g. Wasserstein GAN, BEGAN) alleviates this problem, better stability!

Training GAN

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



After training, use generator network to generate new images

Training GAN

Generated samples

Nearest neighbor from training set

Training GAN – Summary

- Goal of generator is to produce fakes that look real to the discriminator.
- Goal of discriminator is to distinguish generator's fake apart from the real training examples.
- Both models learn from the competition.
- Learning continues until the examples produced by the Generator are good enough to foo the Discriminator.

GAN and CNN

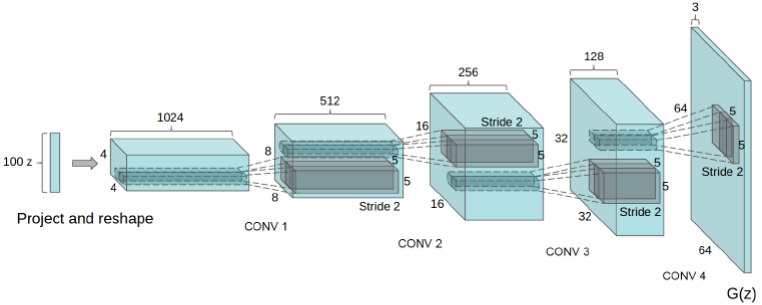


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

GAN and CNN

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

GAN – Summary

Don't work with an explicit density function
Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- Beautiful, state-of-the-art samples!

Cons:

- Trickier / more unstable to train
- Can't solve inference queries such as p(x), p(z|x)

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

References

1. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
   https://www.deeplearningbook.org/
2. Deep Learning with Python by Francois Chollet.
   https://livebook.manning.com/book/deep-learning-with-python/

Thank You!