Generative Adversarial Networks

”The most exciting idea in Deep Learning in decades.”

Ben Zhang

Waterloo Data Science Club

Table of contents

Introduction

Generative Models

The Problem: Instead of performing classification or regression, we want to **generate** data from a distribution.

- Density Estimation: Learn the underlying density function of the sample data.
- Sample Generation: Sample points from the underlying density function.

Discriminative vs Generative

Discriminative Model

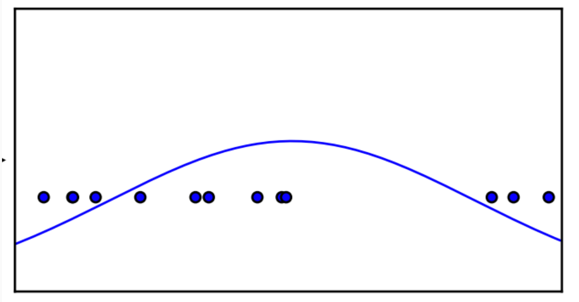
- Find $P(y|x)$.
- Given a data point x, what is the most likely value of y?
- Easy to model.
- Eg. Logistic Regression, SVMs, etc.

Generative model

- Find $P(x, y)$.
- What is the most likely pair (x, y) that we can observe?
- Needs lots of data to get a good approximation.
- Eg. Naive Bayes Classifier, Gaussian Mixture Models

Consider a true distribution that draws uniformly from $(0, 0), (0, 0), (1, 0), (1, 1)$. What is $P(x, y)$ and $P(y|x)$?

Generative Models



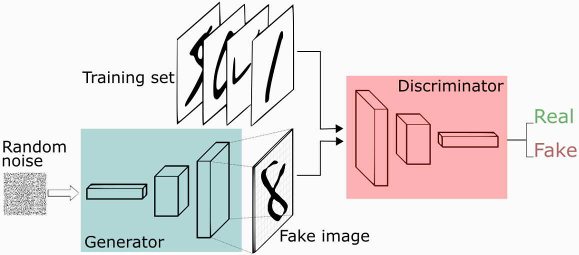
Adversarial Training

A widely used phrase to describe many many things.

> *"Training a model in a worst-case scenario, with inputs chosen by an adversary" - Ian Goodfellow, 2016*

GANs

Overview


Training set
Discriminator
Real
Fake
Random noise
Generator
Fake image

Generative Adversarial Networks

Two players (Neural Networks) that are adversarial to each other.

- The Discriminator, D: A classifier which takes a data point, and determines whether it is generated by G, or drawn from the true distribution.
- The Generator, G: generates adversarial examples from the sample distribution against the discriminator.

Example: Counterfeit Money vs Police

Consider this analogy in the real world.

- Counterfeiters create fake coins that try to look as real as possible.
- Police try to determine whether coins are real or fake.
- The two entities learn from each other, producing better coins, and better detectors.

What does this model converge to?

Discriminator

$D : X \to \{0, 1\}$, where is a neural network which takes in a data point from the domain X and determines whether it is real or not.

D is trained such that it maximizes the probability of labelling the inputs correctly. Its objective is to maximize:

$$\mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Equivalently, we can set the loss function to be:

$$J^{(D)} = -\mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] - \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

This is the *Jenson-Shannon divergence* of the true distribution and $G(z)$.

Generator

$G : Z \to X$, where Z is the latent domain, for example $Z := [0, 1]^d$.

G is the generator network, which takes some random noise $z \in Z$ and produced $G(z) \in X$, a "fake data point". If X is the domain of pictures then $G(z)$ would be a generated picture.

During training, we want G to minimize D's accuracy when it comes to generated samples. Its objective function is:

$$J^{(G)} = \log(1 - D(G(z)))$$

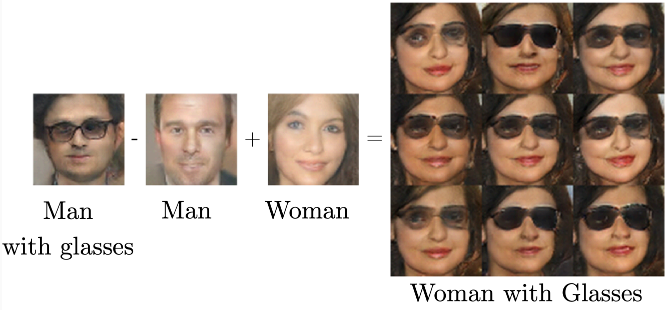
There are a lot of variations that produce similar optimal values, with stronger gradients, eg.

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

Z-space

While many examples of GANs treat the z vector that is inputted into G as a random vector, we can consider it as a latent space.

Specifically, we can perform arithmetic on the latent space vectors. Thus, we can see Z as a low dimensional latent representation of X where important features are preserved on linear relations.



(Radford et al, 2015)

Minimax

Then, training the two models together, we have the following minimax value function:

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

Note that the order matters!

$$\min_G \max_D V(D,G) \neq \max_D \min_G V(D,G)$$

The latter suffers from mode collapse: all mass converges to the most likely point.

A Diagram

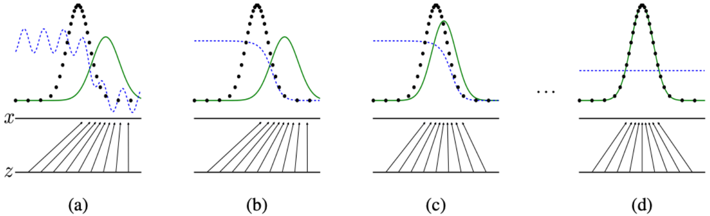


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

The Original 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.

The Optimal Discriminator

For a fixed G, the optimal discriminator is:

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

An intuitive result, but proven in the paper.

Then, the optimal point for our GAN is achieved when minimizing $C(G) =$

$$\mathbb{E}_{x \sim p_{data}(x)} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)} \right] + \mathbb{E}_{z \sim p_z(z)} \left[\log \frac{p_{model}(x)}{p_{data}(x) + p_{model}(x)} \right]$$

Equivalence of $C(G)$ with Jenson-Shannon divergence

With a bit of algebra, we can show:

$$
\begin{aligned}
C(G) = \mathbb{E}_{x \sim p_{data}(x)} &\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)} \right] \\
&+ \mathbb{E}_{z \sim p_z(z)} \left[\log \frac{p_{model}(x)}{p_{data}(x) + p_{model}(x)} \right] \\
&= -\log(4) + KL\left(p_{data} \mid\mid \frac{p_{model} + p_{data}}{2} \right) + KL\left(p_{model} \mid\mid \frac{p_{model} + p_{data}}{2} \right) \\
&= -\log(4) + 2 * JS(p_{data} \mid\mid p_{model})
\end{aligned}
$$

Sicne the Jenson-Shannon divergence is always non-negative, and only zero of $p_{data} = p_{model}$, it follows that the optimal value exists only when $p_{data} = p_{model}$.

This is an elegant proof of this fact: at the optimal value, assuming perfect training, the GAN will converge to a perfect generator that is indistinguishable from the true distribution.

Entropy

Entropy

An abstract idea from information theory introduced by Claude Shannon. What is the smallest number of bits, on average, needed to encode the given distribution?

Consider sending a message from A to B through a channel, while trying to encode 4 possible messages.

What if the possible messages occurs with different probabilities?

Entropy Formula

The entropy of probability distribution P, which we also denote as the density function, is given by

$$H(P) = \mathbb{E}_{x \sim P}[-\log P(x)]$$

So, if P is a a distribution over a continuous variable, then:

$$H(P) = -\int P(x) \log P(x) dx$$

And if discrete then:

$$H(P) = -\sum_{x} P(x) \log P(x)$$

Example: Entropy of a Fair Die

- What is the entropy of a fair 6 sided die?

Example: Entropy of a Fair Die

- What is the entropy of a fair 6 sided die?
- $$H(P) = -\sum_{i=1}^{6} \frac{1}{6} \log_2 \frac{1}{6} = \log_2(6) \approx 2.585$$

Cross Entropy

Familiar concept in statistics, eg. for calculating binary classification loss.

Given an approximate distribution Q, how many bits of information do we need to encode P?

$$H(P, Q) = \mathbb{E}_{x \sim P}[-\log Q(x)]$$

Note: Cross Entropy $\geq$ Entropy, with equality if and only if $P = Q$.

Example: Cross Entropy of a Fair Die

- Assume that we did not have a fair die, but we had a fair coin instead. We model the fair die by flipping the coin 3 times. Of the 8 outcomes, we assign 2 outcomes to 1 and 2, and only 1 outcome to $3, 4, 5, 6$. What is the cross entropy of this approximate distribution compared to the true distribution?

Example: Cross Entropy of a Fair Die

- Assume that we did not have a fair die, but we had a fair coin instead. We model the fair die by flipping the coin 3 times. Of the 8 outcomes, we assign 2 outcomes to 1 and 2, and only 1 outcome to $3, 4, 5, 6$. What is the cross entropy of this approximate distribution compared to the true distribution?
- $$\begin{aligned} H(P, Q) &= -\sum_{i=1}^{2} \frac{1}{6} \log_2 \frac{1}{4} - \sum_{i=3}^{6} \frac{1}{6} \log_2 \frac{1}{8} \\ &= \frac{1}{3} \log_2(4) + \frac{2}{3} \log_2(8) = \frac{8}{3} \approx 2.667 \end{aligned}$$

Kullback–Leibler Divergence

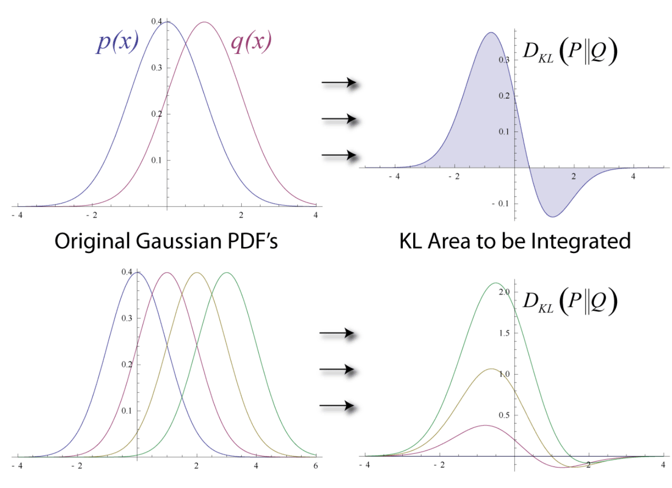
Given distributions P and Q, we want a measure how good Q is at approximating P.

- Useful as a measure of how good our generator G is at approximating the sample distribution.
- Compute this by comparing the cross entropy of P and Q against just the entropy of P.

$$KL(P \mid\mid Q) = H(P, Q) - H(P) = \mathbb{E}_{X \sim P}\left[\frac{\log(P)}{\log(Q)}\right]$$

KL Divergence Explained

The KL divergence tells us how well the probability distribution Q approximates the probability distribution P by calculating the cross-entropy minus the entropy.



Example: KL Divergence of 3 coin tosses vs. a die

The KL divergence of using the 3 coins to approximate a fair die is

$$KL(P \mid\mid Q) = H(P, Q) - H(P) \approx 2.667 - 2.585 = 0.082$$

Problems with KL Divergence

- Non-symmetric: $KL(P \,||\, Q) \neq KL(Q \,||\, P)$.
- Pointwise calculation, blind to closeness in x.

Jenson-Shannon Divergence

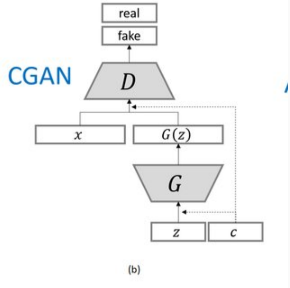
Addresses the non-symmetry of KL divergence.

$$JS(P \mid\mid Q) = \frac{1}{2}KL\left(P \mid\mid \frac{P+Q}{2}\right) + \frac{1}{2}KL\left(Q \mid\mid \frac{P+Q}{2}\right)$$

Note that $JS(P \mid\mid Q) \geq 0$ with equality if and only if $P = Q$.

Variations and Applications

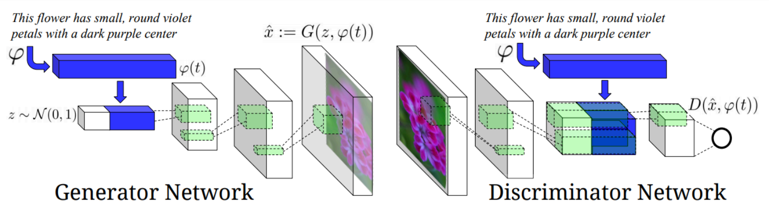
Conditional GANs



- Instead of only z, we also pass in an auxiliary information into G and D.
- For example, we can pass in the class label.

Text to Image Generation with Conditional GANs

From *Generative Adversarial Text to Image Synthesis*, Reed et. Al 2016.



Text to Image Generation with Conditional GANs

From *Generative Adversarial Text to Image Synthesis*, Reed et. Al 2016.

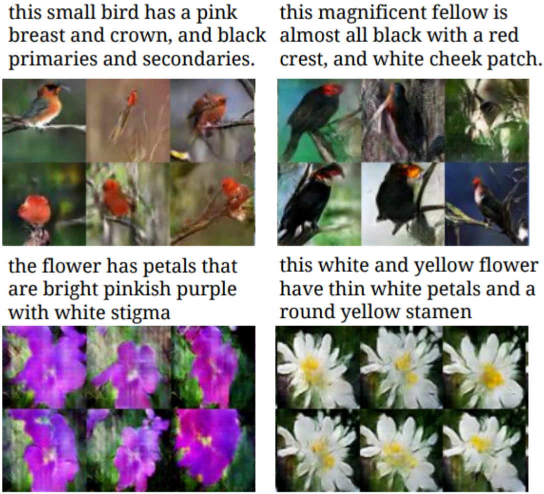


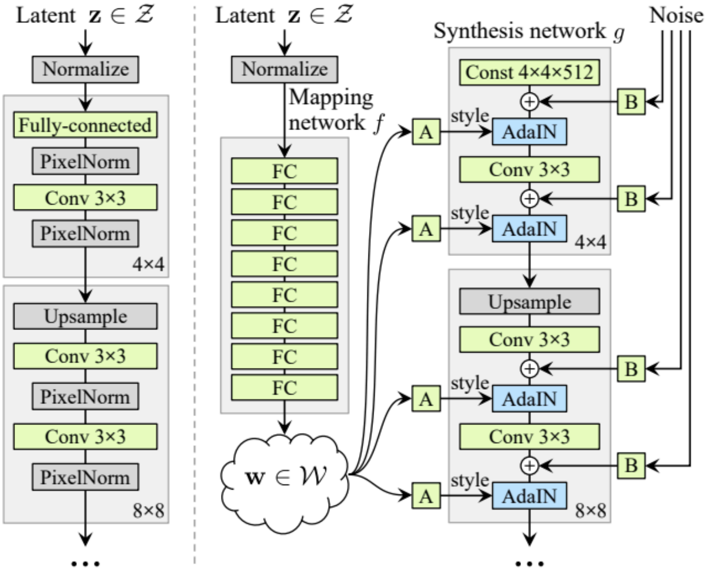
Figure 1. Examples of generated images from text descriptions. Left: captions are from zero-shot (held out) categories, unseen

Image to Image with Conditional GANs

Generating Faces with DC-GANs, NVIDIA 2018

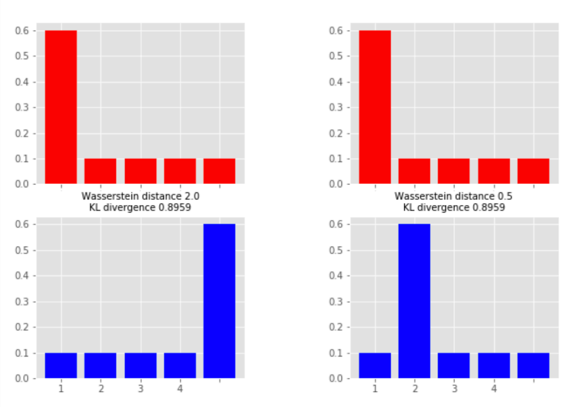
One of the exciting recent results from 2018 came from NVIDIA.

Generating Faces with DC-GANs, NVIDIA 2018



Wasserstein GANs

The Wasserstein Metric addresses the point-wise calculation problem of KL divergence.



The Wasserstein Metric is the solution to the optimal transport problem.

Wasserstein GANs

- Introduced in 2018.
- By using the Wasserstein Metric as the loss function instead of JS divergence, the loss curve is much smoother for non-differentiable density functions, so learning is more stable.