Theory of Generative Adversarial Nets

Jianping Fan
Department of Computer Science
UNC-Charlotte

Course Website:
http://webpages.uncc.edu/jfan/itcs5152.html
Probabilistic Generative Models

Real world → observations

Model → synthetic observations

Density Estimation

\[ Pr(\text{observation}) = Pr(\text{synthetic obs.}) \]
Synthesizing Examples From Probabilistic Generative Model

\[ Pr(\text{obs.}) \]

Goodfellow NIPS tutorial and accompanying paper (arXiv:1701.00160v4 [cs.LG]) provided some figures.
Maximum Likelihood Estimation

\[ \theta^* = \arg \max_{\theta} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(x \mid \theta) \]
Density function $Pr_{\text{model}}(x|\theta)$

Explicit and analytical
- e.g., Gaussian
- can sample directly from model

Explicit and approximate
- e.g., Boltzmann machine
- can estimate probability by running Markov chain monte carlo

Implicit
- GAN
- can’t estimate probability but can draw from distribution with given probability
Adversarial Networks

Generative Model → Real world → Discriminative Model
real or fake?
Generative Model

How to make it generate different samples each time it is run?

- input to model is noise

Generative model as a neural network

- computes $x = G(z|\theta)$
- differentiable
- does not have to be invertible
- $z$ typically has very high dimensionality (higher than $x$)
Think of it as a critic

- a good critic can tell real from fake

Discriminative model as a neural net

- differentiable

- computes $D(x)$, with value 1 if real, 0 if fake
Training Procedure: Basic Idea

G tries to fool D

D tries not to be fooled

Models are trained simultaneously

- As G gets better, D has a more challenging task
- As D gets better, G has a more challenging task

Ultimately, we don’t care about the D

- Its role is to force G to work harder
Loss Functions

Loss function for D
- maximize the likelihood that model says ‘real’ to samples from the world and ‘fake’ to generated samples

\[ \mathcal{L}_D = - \frac{1}{2} \mathbb{E}_{x \sim \text{world}} \ln D(x) - \frac{1}{2} \mathbb{E}_z \ln (1 - D(G(z))) \]

What should the loss function be for G?
- \[ \mathcal{L}_G = -\mathcal{L}_D \]

But because first term doesn’t matter for G (why?)

\[ \mathcal{L}_D = \frac{1}{2} \mathbb{E}_z \ln (1 - D(G(z))) \]

Known as a minimax procedure
Training Procedure

Train both models simultaneously via stochastic gradient descent using minibatches consisting of

- some generated samples
- some real-world samples

Training of D is straightforward

Error for G comes via back propagation through D

- Two ways to think about training
  1. freeze D weights and propagate $\mathcal{L}_G$ through D to determine $\frac{\partial \mathcal{L}_G}{\partial x}$
  2. Compute $\frac{\partial \mathcal{L}_D}{\partial x}$ and then $\frac{\partial \mathcal{L}_G}{\partial x} = -\frac{\partial \mathcal{L}_D}{\partial x}$

D can be trained without altering G, and vice versa

- May want multiple training epochs of just D so it can stay ahead
- May want multiple training epochs of just G because it has a harder task
The Discriminator Has a Straightforward Task

D has learned when

\[ D(x) = Pr(\text{real}|x) = \frac{Pr(x|\text{real})}{Pr(x|\text{real}) + Pr(x|\text{synthesized})} \]
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.
All losses seem to produce sharp samples
Deconvolutional GANs (DCGAN) (Radford et al., 2015)

Batch normalization important here, apparently
Using Labels Can Improve Generated Samples

Denton et al. (2015)

Generative Model

noise ($z$)

class ($y$)

Salimans et al. (2016)

Discriminative Model

real fake class 1

real class $n$
Using Labels Can Improve Generated Samples
(Denton et al., 2015)

LAPGAN: multiresolution deconvolutional pyramid
CC-LAPGAN: class conditional LAPGAN
GAN: original Goodfellow model
Beyond Labels: Providing Images as Input to Generator: Next Video Frame Prediction (Lotter et al., 2016)

MSE tends to produce blurry images on any task

- when you can’t predict well, predict the expectation
Beyond Labels: Providing Images as Input to Generator: Image Super-Resolution (Ledig et al., 2016)
Visually-Aware Fashion Recommendation and Design With GANs (Kang et al., 2017)

Recommender systems predict how much a particular user will like a particular item

- Can predict based on features of item (e.g., movie director, dress length)
- Can also predict directly from images

Twist here is that instead of predicting from a predefined set, *generate images* that would be liked.
Visually-Aware Fashion Recommendation and Design With GANs (Kang et al., 2017)

Use class-conditional generator and discriminator

(a) Generator $G(z,c)$

(b) Discriminator $D(x,c)$
Visually-Aware Fashion Recommendation and Design With GANs (Kang et al., 2017)

Optimize with GAN

- find latent representation $z$ that obtains the highest recommendation score
- gradient ascent search
Cycle GANs
(Zhu et al., 2017; arXiv:1703.10593v2 [cs.CV])

Given two image collections

- algorithm learns to translate an image from one collection to the other
- does not require correspondence between images
Photos to paintings
Paintings to photos
Star GAN

High resolution image synthesis
Components of GANs

Real World

Noise

Generator

Discriminator

True/False
Overview of GANs

Source: https://ishmaelbelghazi.github.io/ALI
Discriminative Models

A discriminative model learns a function that maps the input data \((x)\) to some desired output class label \((y)\).

In probabilistic terms, they directly learn the conditional distribution \(P(y/x)\).
Generative Models

A generative model tries to learn the joint probability of the input data and labels simultaneously i.e. $P(x,y)$.

Potential to understand and explain the underlying structure of the input data even when there are no labels.
How GANs are being used?

Applied for modelling natural images.

Performance is fairly good in comparison to other generative models.

Useful for unsupervised learning tasks.
Why GANs?

- Use a latent code.
- Asymptotically consistent (unlike variational methods).
- No Markov chains needed.
- Often regarded as producing the best samples.
How to train GANs?

Objective of generative network - increase the error rate of the discriminative network.

Objective of discriminative network – decrease binary classification loss.

Discriminator training - backprop from a binary classification loss.

Generator training - backprop the negation of the binary classification loss of the discriminator.
Loss Functions

\[ \mathcal{L}(\hat{x}) = \min_{x \in \text{data}} (x - \hat{x})^2 \]

\[ D^*_G(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \]

Generator

Discriminator
One-sided Label smoothing - replaces the 0 and 1 targets for a classifier with smoothed values, like .9 or .1 to reduce the vulnerability of neural networks to adversarial examples.

Virtual batch Normalization - each example $x$ is normalized based on the statistics collected on a reference batch of examples that are chosen once and fixed at the start of training, and on $x$ itself.
Original CIFAR-10 vs. Generated CIFAR-10 samples

Several new concepts built on top of GANs have been introduced –

- **InfoGAN** – Approximate the data distribution and learn interpretable, useful vector representations of data.

- **Conditional GANs** - Able to generate samples taking into account external information (class label, text, another image). Force $G$ to generate a particular type of output.
Major Difficulties

Networks are difficult to converge.

Ideal goal – Generator and discriminator to reach some desired equilibrium but this is rare.

GANs are yet to converge on large problems (E.g. Imagenet).
Common Failure Cases

The discriminator becomes too strong too quickly and the generator ends up not learning anything.

The generator only learns very specific weaknesses of the discriminator.

The generator learns only a very small subset of the true data distribution.
So what can we do?

- Normalize the inputs
- A modified loss function
- Use a spherical Z
- BatchNorm
- Avoid Sparse Gradients: ReLU, MaxPool
- Use Soft and Noisy Labels
- DCGAN / Hybrid Models
Autoencoder

As close as possible

Randomly generate a vector as code

Image ?
Autoencoder with 3 fully connected layers

Training: model.fit(X,X)
Cost function: $\Sigma_{k=1..N} (x_k - x'_k)^2$
Auto-encoder

- NN Decoder
- 2D code
- NN Decoder
- Image
Auto-encoder
Auto-encoder

Input $\rightarrow$ NN Encoder $\rightarrow$ NN Decoder $\rightarrow$ Output

Minimize reconstruction error

VAE

Input $\rightarrow$ NN Encoder $\rightarrow$ code $\rightarrow$ NN Decoder $\rightarrow$ Output

$c_i = \exp(\sigma_i)e_i + m_i$

$\sum_{i=1..3} [\exp(\sigma_i) - (1+\sigma_i) + (m_i)^2]$

This constrains $\sigma_i$ approaching 0 is good

Problems of VAE

It does not really try to simulate real images

- code
- NN Decoder
- Output
- As close as possible
- One pixel difference to the target
- Realistic
- Fake
- Also one pixel difference to the target
- VAE treats these the same
Gradual and step-wise generation

NN Generator v1 → Disriminator v1 → Real images: 5 0 4 1

NN Generator v2 → Disriminator v2 → Generated images:

NN Generator v3 → Disriminator v3 → Generated images:

These are Binary classifiers
GAN – Learn a discriminator

Randomly sample a vector

Something like Decoder in VAE

Real images Sampled from DB:

1/0 (real or fake)
GAN – Learn a generator

Updating the parameters of generator

The output be classified as “real” (as close to 1 as possible)

Generator + Discriminator = a network

Using gradient descent to update the parameters in the generator, but fix the discriminator

They have Opposite objectives

Randomly sample a vector
Generating 2nd element figures

You can use the following to start a project (but this is in Chinese):

Source of images: https://zhuanlan.zhihu.com/p/24767059
From Dr. HY Lee’s notes.

DCGAN: https://github.com/carpedm20/DCGAN-tensorflow
GAN – generating $2^{nd}$ element figures

100 rounds

This is fast, I think you can use your CPU
GAN – generating 2\textsuperscript{nd} element figures

1000 rounds
GAN – generating 2\textsuperscript{nd} element figures

2000 rounds
GAN – generating 2\textsuperscript{nd} element figures

5000 rounds
GAN – generating 2\textsuperscript{nd} element figures

10,000 rounds
GAN – generating 2\textsuperscript{nd} element figures

20,000 rounds
GAN – generating 2\textsuperscript{nd} element figures

50,000 rounds
Next Video Frame Prediction

Ground Truth  MSE  Adversarial

Traditional mean-squared Error, averaged, blurry

(Lotter et al 2016)
Single Image Super-Resolution

original
bicubic (21.59dB/0.6423)
SRResNet (23.44dB/0.7777)
SRGAN (20.34dB/0.6562)

(Ledig et al 2016)

Last 2 are by deep learning approaches.
Image to Image Translation

(Isola et al 2016)
DCGANs for LSUN Bedrooms

(Radford et al 2015)
Similar to word embedding (DCGAN paper)

Vector Space Arithmetic

Man with glasses - Man + Woman = Woman with Glasses

(Radford et al, 2015)
256x256 high resolution pictures by Plug and Play generative network

PPGN Samples

(Nguyen et al 2016)
From natural language to pictures

PPGN for caption to image

oranges on a table next to a liquor bottle

(Nguyen et al 2016)
Deriving GAN

During the rest of this lecture, we will go thru the original ideas and derive GAN.

I will avoid the continuous case and stick to simple explanations.
Maximum Likelihood Estimation

Give a data distribution $P_{\text{data}}(x)$

We use a distribution $P_G(x; \theta)$ parameterized by $\theta$ to approximate it

- E.g. $P_G(x; \theta)$ is a Gaussian Mixture Model, where $\theta$ contains means and variances of the Gaussians.

- We wish to find $\theta$ s.t. $P_G(x; \theta)$ is close to $P_{\text{data}}(x)$

In order to do this, we can sample

$\{x^1, x^2, \ldots, x^m\}$ from $P_{\text{data}}(x)$

The likelihood of generating these $x^i$’s under $P_G$ is

$L = \prod_{i=1}^{m} P_G(x^i; \theta)$

Then we can find $\theta^*$ maximizing the $L$. 
KL (Kullback-Leibler) divergence

Discrete:

\[ D_{KL}(P \mid Q) = \sum_i P(i) \log \left[ \frac{P(i)}{Q(i)} \right] \]

Continuous:

\[ D_{KL}(P \mid Q) = \int p(x) \log \left[ \frac{p(x)}{q(x)} \right] dx \]

Explanations:

Entropy: \(- \sum P(i) \log P(i)\) - expected code length (also optimal)

Cross Entropy: \(- \sum P(i) \log Q(i)\) - expected coding

\[ D_{KL} = \sum P(i) \log \left[ \frac{P(i)}{Q(i)} \right] = \sum P(i) \left[ \log P(i) - \log Q(i) \right], \text{ extra bits} \]

JSD(P \mid Q) = \frac{1}{2} D_{KL}(P \mid M) + \frac{1}{2} D_{KL}(Q \mid M), \text{ M= \frac{1}{2} (P+Q), symmetric KL} \]

* JSD = Jensen-Shannon Divergency
Maximum Likelihood Estimation

$$\theta^* = \arg \max_\theta \prod_{i=1..m} P_G(x^i; \theta) \rightarrow$$

$$\arg \max_\theta \log \prod_{i=1..m} P_G(x^i; \theta)$$

$$= \arg \max_\theta \sum_{i=1..m} \log P_G(x^i; \theta), \{x^1, ..., x^m\} \text{ sampled from } P_{data}(x)$$

$$= \arg \max_\theta \sum_{i=1..m} P_{data}(x^i) \log P_G(x^i; \theta) \quad \text{--- this is cross entropy}$$

$$= \arg \max_\theta \sum_{i=1..m} P_{data}(x^i) \log P_G(x^i; \theta) - \sum_{i=1..m} P_{data}(x^i) \log P_{data}(x^i)$$

$$= \arg \min_\theta \text{KL } (P_{data}(x) \mid \mid P_G(x; \theta)) \quad \text{--- this is KL divergence}$$

Note: $P_G$ is Gaussian mixture model, finding best $\theta$ will still be Gaussians, this only can generate a few blubs. Thus this above maximum likelihood approach does not work well.

Next we will introduce GAN that will change $P_G$, not just estimating $P_G$ is parameters We will find best $P_G$, which is more complicated and structured, to approximate $P_{data}$. 
Thus let’s use an NN as $P_G(x; \theta)$

$P_G(x, \theta) = \int \left[ P_{\text{prior}}(z) I_{[G(z) = x]} \right] dz$

https://blog.openai.com/generative-models/
Basic Idea of GAN

Generator $G$
- $G$ is a function, input $z$, output $x$
- Given a prior distribution $P_{\text{prior}}(z)$, a probability distribution $P_G(x)$ is defined by function $G$

Discriminator $D$
- $D$ is a function, input $x$, output scalar
- Evaluate the “difference” between $P_G(x)$ and $P_{\text{data}}(x)$

In order for $D$ to find difference between $P_{\text{data}}$ from $P_G$, we need a cost function $V(G,D)$:

$$G^* = \arg \min_G \max_D V(G,D)$$

Hard to learn $P_G$ by maximum likelihood
Basic Idea

\[ G^* = \text{arg min}_G \max_D V(G,D) \]

Pick JSD function: \( V = E_{x \sim P_{\text{data}}} [\log D(x)] + E_{x \sim P_G} [\log(1-D(x))] \)

Given a generator \( G \), \( \max_D V(G,D) \) evaluates the “difference” between \( P_G \) and \( P_{\text{data}} \)

Pick the \( G \) s.t. \( P_G \) is most similar to \( P_{\text{data}} \)
Given G, what is the optimal D* maximizing

\[ V = \mathbb{E}_{x \sim P_{\text{data}}} [\log D(x)] + \mathbb{E}_{x \sim P_{\text{G}}} [\log(1-D(x))] \]

\[ = \sum [ P_{\text{data}}(x) \log D(x) + P_{\text{G}}(x) \log(1-D(x)] \]

Thus: \(D^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_{\text{G}}(x)}\)

Assuming \(D(x)\) can have any value here

Given \(x\), the optimal \(D^*\) maximizing is:

\[ f(D) = a \log D + b \log(1-D) \Rightarrow D^* = \frac{a}{a+b} \]
\[ \max_D V(G,D), \quad G^* = \arg\min_G \max_D V(G,D) \]

\[ D_1^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_{G_1}(x)} \]

\[ D_2^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_{G_2}(x)} \]

"difference" between \( P_{G_1} \) and \( P_{\text{data}} \)
\[
\begin{align*}
\max_D V(G,D) &= V(G,D^*)\quad \text{where} \quad D^*(x) = \frac{P_{\text{data}}}{P_{\text{data}} + P_G}, \text{ and} \\
1-D^*(x) &= \frac{P_G}{P_{\text{data}} + P_G} \\
&= E_{x \sim P_{\text{data}}} \log D^*(x) + E_{x \sim P_G} \log (1-D^*(x)) \\
&\approx \Sigma [P_{\text{data}}(x) \log D^*(x) + P_G(x) \log (1-D^*(x))] \\
&= -2\log 2 + 2 \text{ JSD}(P_{\text{data}} \parallel P_G),
\end{align*}
\]

JSD(P||Q) = Jensen-Shannon divergence

\[
= \frac{1}{2} D_{\text{KL}}(P||M) + \frac{1}{2} D_{\text{KL}}(Q||M)
\]

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

\[
D_{\text{KL}}(P||Q) = \Sigma P(x) \log P(x) / Q(x)
\]
Generator G, Discriminator D

Looking for $G^*$ such that

$$V = \mathbb{E}_{x \sim P_{\text{data}}} \left[ \log D(x) \right] + \mathbb{E}_{x \sim P_G} \left[ \log (1 - D(x)) \right]$$

Given $G$, $\max_D V(G,D)$

$$= -2 \log 2 + 2 \text{JSD}(P_{\text{data}}(x) || P_G(x))$$

What is the optimal $G$? It is $G$ that makes JSD smallest = 0:

$$P_G(x) = P_{\text{data}}(x)$$
Algorithm

\[ G^* = \arg \min_G \max_D V(G,D) \]

To find the best \( G \) minimizing the loss function \( L(G) \):

\[ \theta_G \leftarrow \theta_G = -\eta \frac{\partial L(G)}{\partial \theta_G} \]

\( \theta_G, \theta_G \) defines \( G \)

Solved by gradient descent. Having max ok. Consider simple case:

\[ f(x) = \max\{D_1(x), D_2(x), D_3(x)\} \]

If \( D_i(x) \) is the Max in that region, then do \( dD_i(x)/dx \)

\[ \frac{dD_1(x)}{dx}, \frac{dD_2(x)}{dx}, \frac{dD_3(x)}{dx} \]
Given $G_0$

Find $D^*_0$ maximizing $V(G_0,D)$

$$V(G_0,D^*_0)$$ is the JS divergence between $P_{data}(x)$ and $P_{G_0}(x)$

$$\theta_G \leftarrow \theta_G - \eta \frac{\Delta V(G,D^*_0)}{\theta_G} \quad \Rightarrow \quad \text{Obtaining } G_1 \text{ (decrease JSD)}$$

Find $D_1^*$ maximizing $V(G_1,D)$

$$V(G_1,D_1^*)$$ is the JS divergence between $P_{data}(x)$ and $P_{G_1}(x)$

$$\theta_G \leftarrow \theta_G - \eta \frac{\Delta V(G,D_1^*)}{\theta_G} \quad \Rightarrow \quad \text{Obtaining } G_2 \text{ (decrease JSD)}$$

And so on ...
In practice ...

\[
V = \mathbb{E}_{x \sim P_{\text{data}}} [\log D(x)] + \mathbb{E}_{x \sim P_{G}} [\log(1-D(x))]
\]

Given \( G \), how to compute \( \max_D V(G,D) \)?

- Sample \( \{x^1, \ldots, x^m\} \) from \( P_{\text{data}} \)
- Sample \( \{x^{*1}, \ldots, x^{*m}\} \) from generator \( P_G \)

Maximize:

\[
V' = \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log(1-D(x^*_i))
\]

This is what a Binary Classifier do

Output is \( D(x) \)

- Minimize Cross-entropy

If \( x \) is a positive example

Minimize \( -\log D(x) \)

If \( x \) is a negative example

Minimize \( -\log(1-D(x)) \)

Positive example
D must accept

Negative example
D must reject
D is a binary classifier (can be deep) with parameters $\theta_d$

$$\{x^1, x^2, \ldots x^m\} \text{ from } P_{data}(x) \quad \text{Positive examples}$$

$$\{x^*1, x^*2, \ldots x^*m\} \text{ from } P_{G}(x) \quad \text{Negative examples}$$

Minimize $$L = -V'$$

or

Maximize $$V' = \sum_{i=1..m} \log D(x^i) + \frac{1}{m} \sum_{i=1..m} \log(1-D(x^i))$$
Algorithm

Initialize $\theta_d$ for D and $\theta_g$ for G

In each training iteration

- Sample $m$ examples $\{x^1, x^2, \ldots, x^m\}$ from data distribution $P_{data}(x)$
- Sample $m$ noise samples $\{z^1, \ldots, z^m\}$ from a simple prior $P_{prior}(z)$
- Obtain generated data $\{x^{*1}, \ldots, x^{*m}\}$, $x^{*i} = G(z^i)$

Learning D

Update discriminator parameters $\theta_d$ to maximize

$V' = 1/m \Sigma_{i=1..m} \log D(x^i) + 1/m \Sigma_{i=1..m} \log(1-D(x^{*i}))$

$\theta_d \leftarrow \theta_d + \eta \Delta V'(\theta_d)$ (gradient ascent)

- Sample another $m$ noise samples $\{z^1, z^2, \ldots, z^m\}$ from the prior $P_{prior}(z)$, $G(z^i) = x^{*i}$
- Update generator parameters $\theta_g$ to minimize

$V' = 1/m \Sigma_{i=1..m} \log D(x^i) + 1/m \Sigma_{i=1..m} \log(1-D(x^{*i}))$

$\theta_g \leftarrow \theta_g - \eta \Delta V'(\theta_g)$ (gradient descent)

Can only find lower bound of JSD or $\max_{\theta} V(G,D)$

Ian Goodfellow comment: this is also done once
Objective Function for Generator in Real Implementation

\[ V = E_{x \sim P_{\text{data}}} [\log D(x)] + E_{x \sim P_G} [\log (1-D(x))] \]

Training slow at the beginning

\[ V = E_{x \sim P_G} [- \log (D(x))] \]

Real implementation: label x from \( P_G \) as positive
Some issues in training GAN

Evaluating JS divergence

Discriminator is too strong: for all three Generators, JSD = 0

Evaluating JS divergence

JS divergence estimated by discriminator telling little information

Weak Generator

Strong Generator
Reason 1. Approximate by sampling

Weaken your discriminator?

Can weak discriminator compute JS divergence?

Discriminator

\[ V = E_{x \sim P_{\text{data}}} \left[ \log D(x) \right] + E_{x \sim P_{\tilde{G}}} \left[ \log(1 - D(x)) \right] \]

\[ = \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(x^i)) \]

\[ \max_D V(G, D) = -2 \log 2 + 2 \text{ JSD}(P_{\text{data}} \parallel P_{\tilde{G}}) = 0 \]

\log 2 \text{ when } P_{\text{data}} \text{ and } P_{\tilde{G}} \text{ differ completely}
Discriminator

\[ V = \mathbb{E}_{x \sim P_{\text{data}}} \left[ \log D(x) \right] + \mathbb{E}_{x \sim P_{\text{G}}} \left[ \log(1-D(x)) \right] \]

\[ = \frac{1}{m} \sum_{i=1..m} \log D(x^i) + \frac{1}{m} \sum_{i=1..m} \log(1-D(x^i)) \approx 0 \]

\[ \max_D V(G,D) = -2 \log 2 + 2 \text{JSD}(P_{\text{data}} \parallel P_{\text{G}}) = 0 \]

Reason 2. the nature of data

\( P_{\text{data}}(x) \) and \( P_G(x) \) have very little overlap in high dimensional space
Evolution
http://www.guokr.com/post/773890/

Better
Evolution needs to be smooth:

\[ JSD(P_{G_0} \parallel P_{\text{data}}) = \log_2 \]

\[ JSD(P_{G_{50}} \parallel P_{\text{data}}) = \log_2 \]

\[ JSD(P_{G_{100}} \parallel P_{\text{data}}) = 0 \]
One simple solution: add noise

Add some artificial noise to the inputs of discriminator

Make the labels noisy for the discriminator
  Discriminator cannot perfectly separate real and generated data

\[ P_{\text{data}}(x) \text{ and } P_{G}(x) \text{ have some overlap} \]

Noises need to decay over time
Mode Collapse

Generated Distribution

Data Distribution

Converge to same faces

Sometimes, this is hard to tell since one sees only what’s generated, but not what’s missed.
Mode Collapse Example

8 Gaussian distributions:

What we want …

In reality …
### Text to Image, by conditional GAN

<table>
<thead>
<tr>
<th>Caption</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>a pitcher is about to throw the ball to the batter</td>
<td><img src="image1.png" alt="Images" /></td>
</tr>
<tr>
<td>a group of people on skis stand in the snow</td>
<td><img src="image2.png" alt="Images" /></td>
</tr>
<tr>
<td>a man in a wet suit riding a surfboard on a wave</td>
<td><img src="image3.png" alt="Images" /></td>
</tr>
</tbody>
</table>
## Text to Image - Results
From CY Lee lecture

<table>
<thead>
<tr>
<th>Caption</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>this flower has white petals and a yellow stamen</td>
<td>![Images of flowers with white petals and yellow stamen]</td>
</tr>
<tr>
<td>the center is yellow surrounded by wavy dark purple petals</td>
<td>![Images of flowers with purple petals and yellow centers]</td>
</tr>
<tr>
<td>this flower has lots of small round pink petals</td>
<td>![Images of flowers with pink petals]</td>
</tr>
</tbody>
</table>

**Project topic:** Code and data are all on web, many possibilities!
**Algorithm WGAN**

In each training iteration

- Sample m examples \( \{x^1, x^2, \ldots, x^m\} \) from data distribution \( P_{data}(x) \)
- Sample m noise samples \( \{z^1, \ldots, z^m\} \) from a simple prior \( P_{prior}(z) \)
- Obtain generated data \( \{x^*_1, \ldots, x^*_m\} \), \( x^*_i = G(z^i) \)

Learning \( D \): Update discriminator parameters \( \theta_d \) to maximize

- \( V' = \sum_{i=1..m} \log D(x^i) + \frac{1}{m} \sum_{i=1..m} \log(1 - D(x^*_i)) \)
- \( \theta_d \leftarrow \theta_d + \eta \Delta V'(\theta_d) \) (gradient ascent plus weight clipping)

Simple another m noise samples \( \{z^1, z^2, \ldots, z^m\} \) from the prior \( P_{prior}(z) \), \( G(z^i) = x^*_i \)

Learning \( G \): Update generator parameters \( \theta_g \) to minimize

- \( V = \frac{1}{m} \sum_{i=1..m} \log D(x^i) + \frac{1}{m} \sum_{i=1..m} \log(1 - D(x^*_i)) \)
- \( \theta_g \leftarrow \theta_g - \eta \Delta V'(\theta_g) \) (gradient descent)

Ian Goodfellow comment: this is also done once
Experimental Results

Approximate a mixture of Gaussians by single mixture

<table>
<thead>
<tr>
<th>train \ test</th>
<th>KL</th>
<th>KL-rev</th>
<th>JS</th>
<th>Jeffrey</th>
<th>Pearson</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL</td>
<td>0.2808</td>
<td>0.3423</td>
<td>0.1314</td>
<td>0.5447</td>
<td>0.7345</td>
</tr>
<tr>
<td>KL-rev</td>
<td>0.3518</td>
<td>0.2414</td>
<td>0.1228</td>
<td>0.5794</td>
<td>1.3974</td>
</tr>
<tr>
<td>JS</td>
<td>0.2871</td>
<td>0.2760</td>
<td>0.1210</td>
<td>0.5260</td>
<td>0.92160</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>0.2869</td>
<td>0.2975</td>
<td>0.1247</td>
<td>0.5236</td>
<td>0.8849</td>
</tr>
<tr>
<td>Pearson</td>
<td>0.2970</td>
<td>0.5466</td>
<td>0.1665</td>
<td>0.7085</td>
<td>0.648</td>
</tr>
</tbody>
</table>
We have seen that JSD does not give GAN a smooth and continuous improvement curve.

We would like to find another distance which gives that.

This is the Wasserstein Distance or earth mover’s distance.
Earth Mover’s Distance

Considering one distribution P as a pile of earth (total amount of earth is 1), and another distribution Q (another pile of earth) as the target

The “earth mover’s distance” or “Wasserstein Distance” is the average distance the earth mover has to move the earth in an optimal plan.

\[ W(P,Q) = d \]
Earth Mover’s Distance: best plan to move

P

Q
JS vs Earth Mover’s Distance

\[ W(P_{G_0}, P_{\text{data}}) = d_0 \]
\[ W(P_{G_{50}}, P_{\text{data}}) = d_{50} \]
\[ W(P_{G_{100}}, P_{\text{data}}) = 0 \]

\[ W(P_{G_0}, P_{\text{data}}) = d_0 \]
\[ W(P_{G_{50}}, P_{\text{data}}) = d_{50} \]
\[ W(P_{G_{100}}, P_{\text{data}}) = 0 \]
Explaining WGAN

Let $W$ be the Wasserstein distance.

$$W(P_{\text{data}}, P_G) = \max_{D \text{ is } 1\text{-Lipschitz}} \left[ E_{x \sim P_{\text{data}}} D(x) - E_{x \sim P_G} D(x) \right]$$

Where a function $f$ is a $k$-Lipschitz function if

$$||f(x_1) - f(x_2)|| \leq k ||x_1 - x_2||$$

How to guarantee this?

**Weight clipping:** for all parameter updates, if $w > c$

Then $w = c$, if $w < -c$, then $w = -c$.

**Blue:** $D(x)$ for original GAN

**Green:** $D(x)$ for WGAN

WGAN will provide gradient to push $P_G$ towards $P_{\text{data}}$
Earth Mover Distance Examples:

Multi-layer perceptron