Generative Adversarial Networks

11785 Deep Learning
Spring 2024

Chetan Chilkunda **Quentin Auster**

Recap and Learning Objectives

- VAEs
- Flow Models
- Diffusion Models

- Today: GANs

Learning Objectives

- Generative vs Discriminative models
- Explicit vs Implicit models
- The insufficiency of Maximum Likelihood Estimation for learning GANs
  - Using a Discriminator network for losses
- How GANs train
- Benefits and challenges of GANs
- Learning paradigms (learning through comparison)
  - Comparison by Ratios and the emergence of the Jensen Shannon Divergence
  - Comparison by Differences and the use of Wasserstein distance
  - Zero-sum vs Non-zero-sum
- Variants of GANs

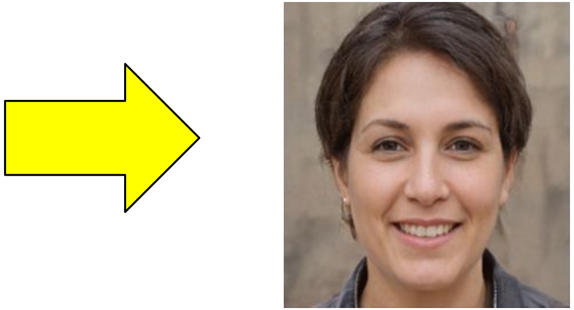
The Problem

From a large collection of images of faces, can a network learn to *generate* new portrait?

Generate samples from the distribution of “face” images

How do we even characterize this distribution?

The Problem



From a large collection of images of faces, can a network learn to *generate* new portrait?

Generate samples from the **distribution** of “face” images

How do we even characterize this distribution?

What are GANs

Generative Adversarial Networks

What are GANs

Generative Adversarial Networks

Generative Model which generates data similar to training data (like VAEs)

Discriminative vs Generative Models

Discriminative

- Learn the conditional distribution $P(Y \mid X)$.
- Learns the decision boundary.
- Limited scope. Used for classification tasks.
- E.g., logistic regression, SVM, etc.

Generative

- Learns joint distribution $P(X, Y)$
    - Can also condition on covariates
- Learns the actual probability distribution of the data.
    - *This is a tougher problem, since it requires a deeper "understanding" of the distribution.*
- Capable of both generative and discriminative tasks.
- E.g., Naïve Bayes, Gaussian Mixture Models, VAE, Diffusion, GANs.

Generative Models

Goals and Tasks

- Generation
- Density Estimation
- Missing Value Imputation
- Structure Discovery
- Latent Space Interpolation + Arithmetic
- … and more

Evaluation

- Sample quality
- Sample diversity
- Generalization

Generative Models

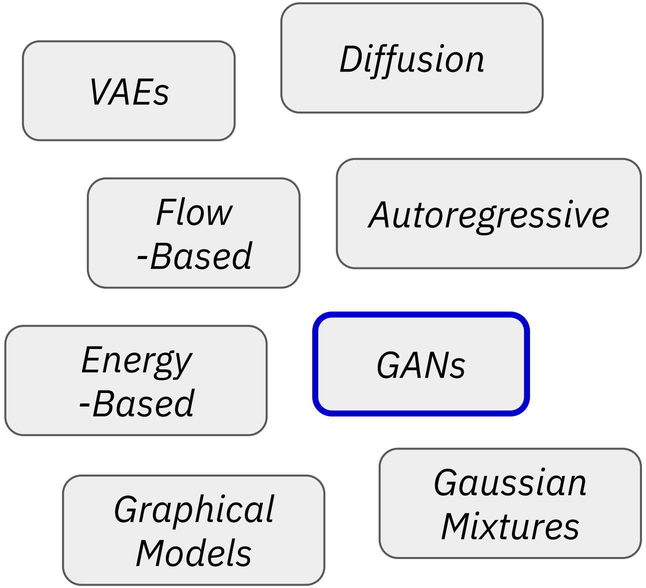
Goals and Tasks

- Generation
- Density Estimation
- Missing Value Imputation
- Structure Discovery
- Latent Space Interpolation + Arithmetic
- … and more

Evaluation

- Sample quality
- Sample diversity
- Generalization

Generative Models



A lot...

How can we start to distinguish between model types?

- Can we evaluate a **probability density function**?
- Can we **sample** from them (quickly)?
- What **training method** can we use?
- Does it rely on a **latent variable** for generation?
- What **architecture** should we use?

Generative Models

VAEs

Diffusion

Flow-Based

Autoregressive

Energy-Based

GANs

Graphical Models

Gaussian Mixtures

A lot...

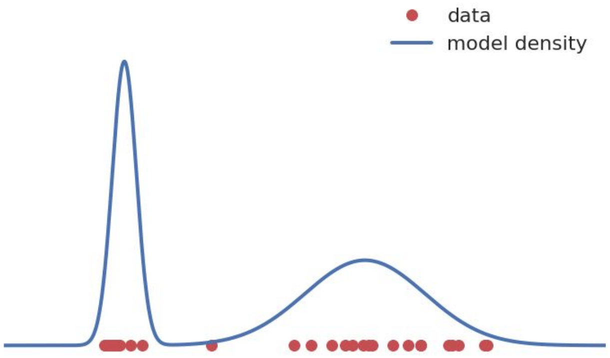
How can we start to distinguish between model types?

- Can we evaluate a **probability density function**?
- Can we **sample** from them (quickly)?
- What **training method** can we use?
- Does it rely on a **latent variable** for generation?
- What **architecture** should we use?

Explicit vs Implicit Models

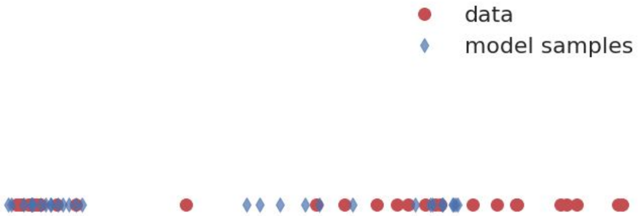
Explicit

- Direct access to probability density function for the distribution.
- Can compute the exact probability of samples.



Implicit

- Ability to sample from distribution, but no access to the density function.

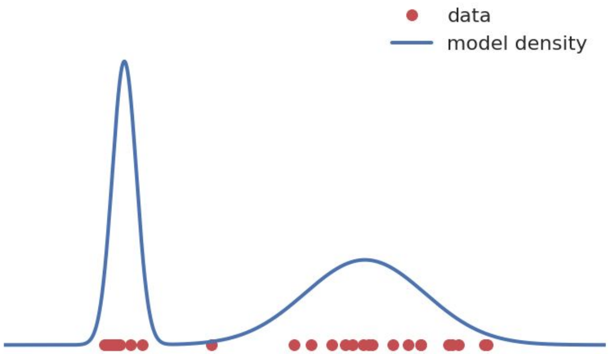


Figures from Murphy (2023), Fig. 26.1, with code available at https://github.com/probml/pyprobml/blob/master/notebooks/book2/26/genmo_types_implicit_explicit.ipynb

Explicit vs Implicit Models

Explicit

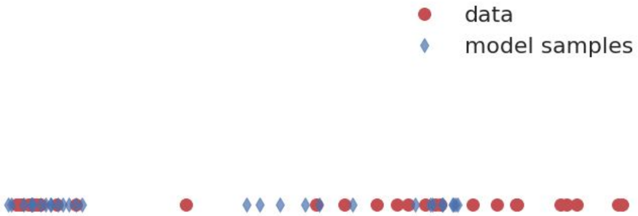
- Direct access to probability density function for the distribution.
- Can compute the exact probability of samples.



Implicit

- Ability to sample from distribution, but no access to the density function.

VAEs and GANs are implicit generative models



Figures from Murphy (2023), Fig. 26.1, with code available at https://github.com/probml/pyprobml/blob/master/notebooks/book2/26/genmo_types_implicit_explicit.ipynb

Poll 1

Q1: What is the difference between Discriminative models vs. Generative models?

- Discriminative models model the decision boundary between classes, whereas Generative models model class distributions
- Generative models model the decision boundary between classes, whereas Discriminative models model class distributions

Q2: What is the difference between Explicit and Implicit Generative models?

- Implicit models compute the probability of samples, whereas Explicit models only let you draw samples from the distribution
- Explicit models compute the probability of samples, whereas Implicit models only let you draw samples from the distribution

Poll 1

Q1: What is the difference between Discriminative models vs. Generative models?

- **Discriminative models model the decision boundary between classes, whereas Generative models model class distributions**
- Generative models model the decision boundary between classes, whereas Discriminative models model class distributions

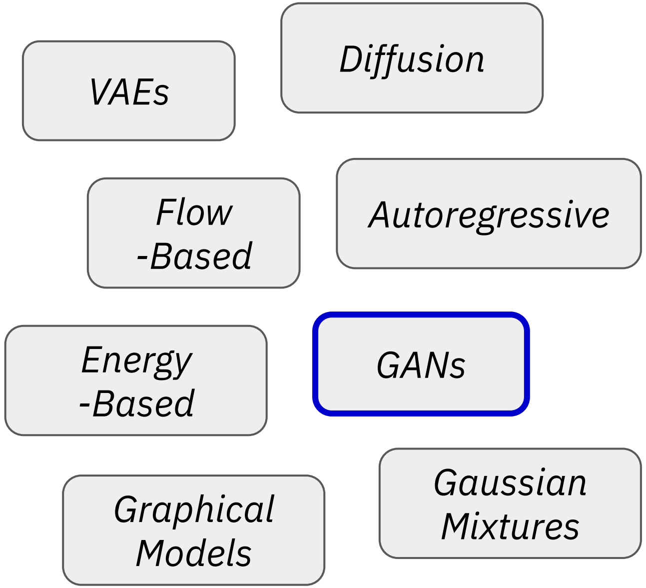
Q2: What is the difference between Explicit and Implicit Generative models?

- Implicit models compute the probability of samples, whereas Explicit models only let you draw samples from the distribution
- **Explicit models compute the probability of samples, whereas Implicit models only let you draw samples from the distribution**

Learning Objectives

- ✓ Generative vs Discriminative models
- ✓ Explicit vs Implicit models
- ❑ The insufficiency of Maximum Likelihood Estimation for learning GANs
  - ❑ Using a Discriminator network for losses
- ❑ How GANs train
- ❑ Benefits and challenges of GANs
- ❑ Learning paradigms (learning through comparison)
  - ❑ Comparison by Ratios and the emergence of the Jensen Shannon Divergence
  - ❑ Comparison by Differences and the use of Wasserstein distance
  - ❑ Zero-sum vs Non-zero-sum
- ❑ Variants of GANs

Generative Models

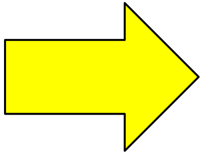


A lot...

How can we start to distinguish between model types?

- Can we evaluate a **probability density function**?
- Can we **sample** from them (quickly)?
- What **training method** can we use?
- Does it rely on a **latent variable** for generation?
- What **architecture** should we use?

The Problem

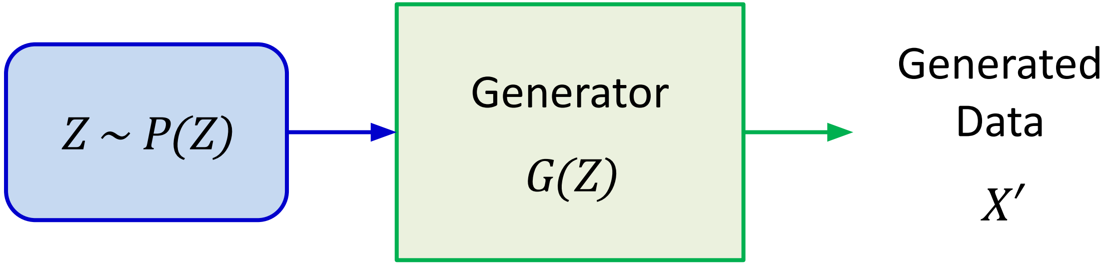


From a large collection of images of faces, can a network learn to *generate* new portrait?

Generate samples from the **distribution** of “face” images

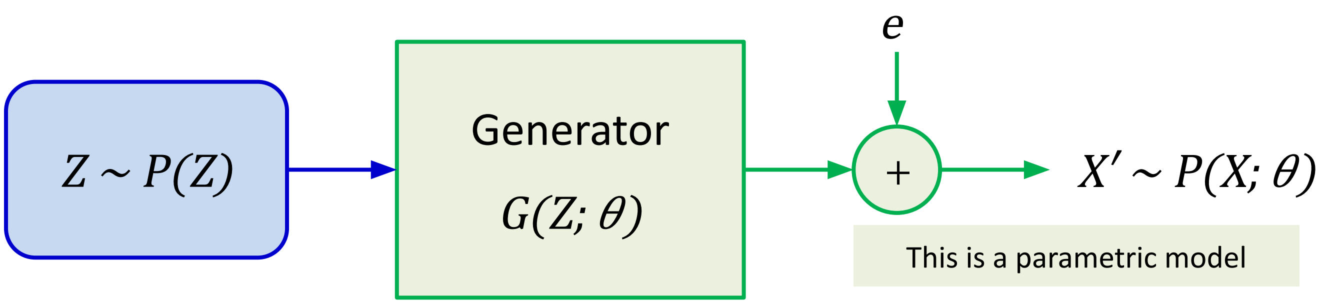
How do we even characterize this distribution?

What we have seen: VAE



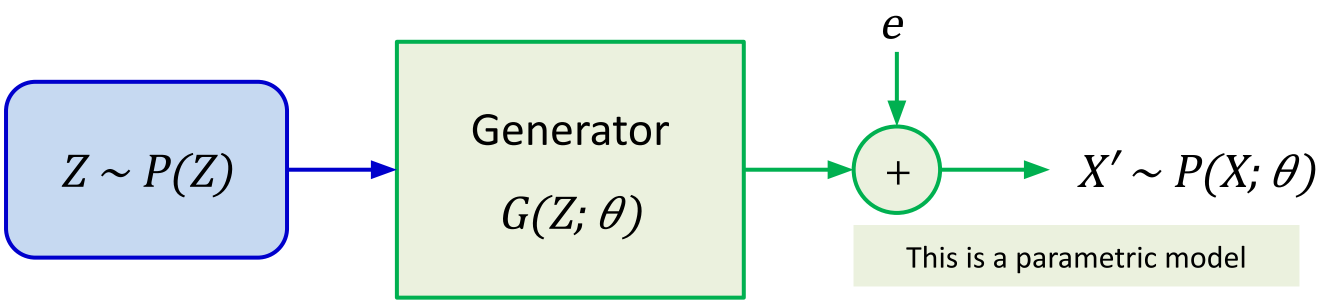
Generator is a decoder of a VAE… how did we train this?

What we have seen: VAE



Generator is a decoder of a VAE… how did we train this?

What we have seen: VAE

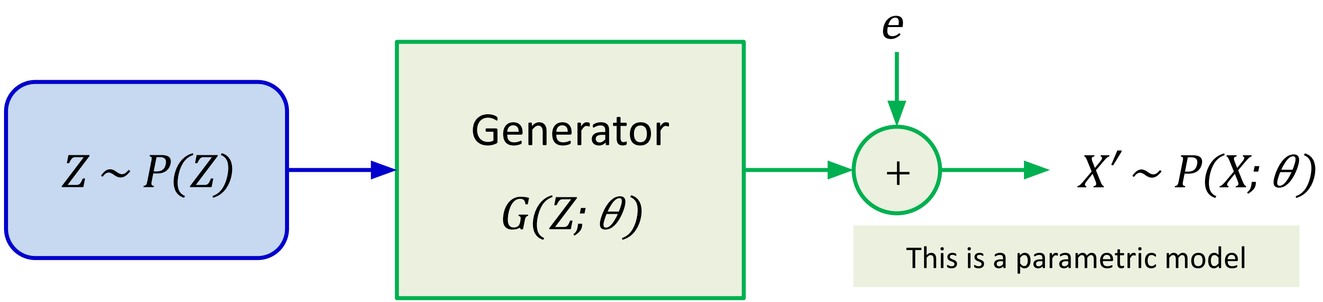


Generator is a decoder of a VAE… how did we train this?

By maximizing the *likelihood* of the data (MLE)

$$\theta^* = \text{argmax}_\theta \ \ \log P(X;\theta)$$

What we have seen: VAE

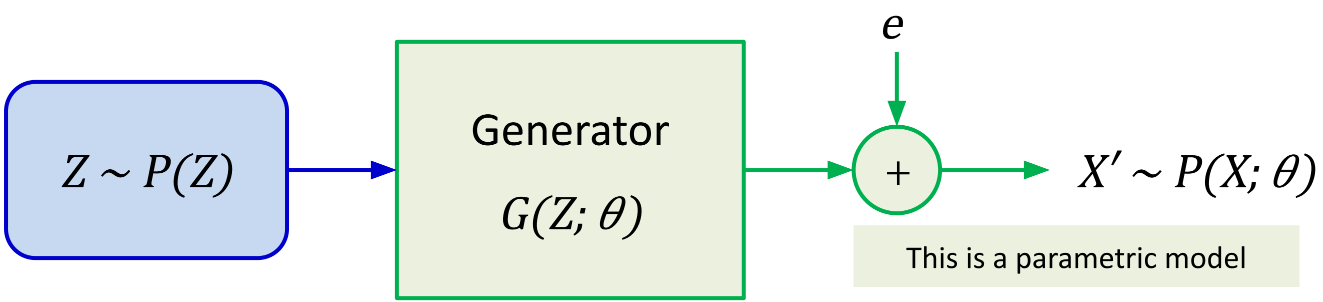


Generator is a decoder of a VAE… how did we train this?

By maximizing the *likelihood* of the data (MLE)

$$\theta^* = \text{argmin}_\theta \quad -\log P(X;\theta)$$

What we have seen: VAE



Generator is a decoder of a VAE… how did we train this?

By maximizing the *likelihood* of the data (MLE)

$$\theta^* = \text{argmin}_\theta \quad -\log P(X; \theta)$$

Any issues here?

Issues with Maximum Likelihood Estimation

- Likelihood can be difficult to compute
  - VAEs and GANs are implicit generative models, so we don’t directly have the likelihood
  - With VAEs, we were able to compute bounds on the log likelihood.
- **Likelihood is not related to perceptual sample quality**

Issues with Maximum Likelihood Estimation

- Likelihood can be difficult to compute
  - VAEs and GANs are implicit generative models, so we don't directly have the likelihood
  - With VAEs, we were able to compute bounds on the log likelihood.
- **Likelihood is not related to perceptual sample quality**
  - **High Likelihood, Bad Samples**
    - Consider a composite model: 0.01(Great Model) + 0.99 (Noise)
    - For high dimensional (D) data, the log likelihood of the composite model will be similar to that of the "Great Model," but 99% of the samples will be noise.

$$q_2(x) = 0.01 q_0(x) + 0.99 q_1(x)$$

$$\log q_2(x) = \log[0.01 q_0(x) + 0.99 q_1(x)] \geq \log[0.01 q_0(x)] = \log q_0(x) - 100$$

$$|\log q_0(x)| \sim D \gg 100 \implies \log q_2(x) \approx \log q_0(x)$$

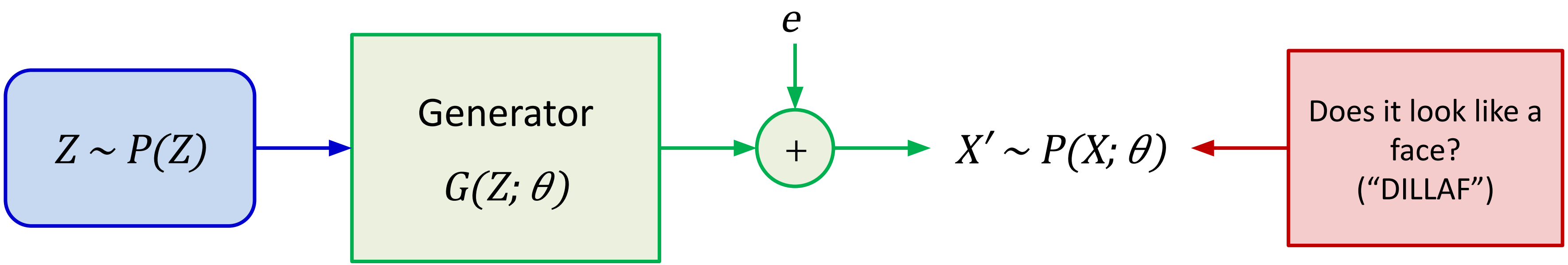
Example from Murphy (2023), Section 20.4.1.3 and Theis et al., "A Note on the Evaluation of Generative Models."

Issues with Maximum Likelihood Estimation

- Likelihood can be difficult to compute
  - VAEs and GANs are implicit generative models, so we don't directly have the likelihood
  - With VAEs, we were able to compute bounds on the log likelihood.
- **Likelihood is not related to perceptual sample quality**
  - **Low Likelihood, Good Samples**
    - Consider a Gaussian Mixture Model centered on training images
    - There may be low noise, meaning the samples will look good, however the model may overfit to the training data and have a poor likelihood on the test set

Example from Murphy (2023), Section 20.4.1.3 and Theis et al., "A Note on the Evaluation of Generative Models."

Replace the negative log likelihood with a more relevant loss



Poll 2

Q1: VAEs are implicit Generative models, True or False

- True
- False

Q2: Why would likelihood maximization not result in a model that produces more face-like outputs (for a face-generating VAE)?

- The model can maximize the likelihood of training data without any assurance about what other (non-training) samples look like
- The model is more likely to run into poor local optima
- The model only captures the mode of the distribution of faces, whereas most face-like images are in the tail of the distribution

Q3: The face-generating model is more likely to generate face-like images if it were trained with a differentiable loss function that explicitly evaluates if the outputs look like faces or note, True or False

- True
- False

Poll 2

Q1: VAEs are implicit Generative models, True or False

- **True**
- False

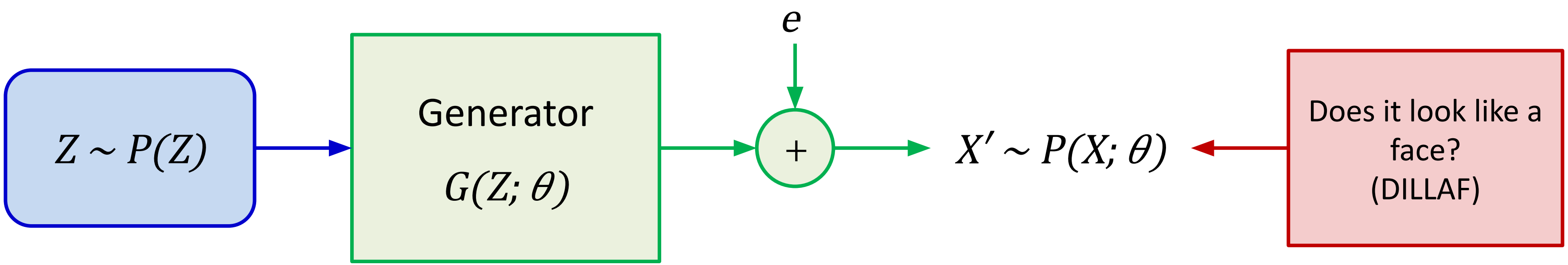
Q2: Why would likelihood maximization not result in a model that produces more face-like outputs (for a face-generating VAE)?

- **The model can maximize the likelihood of training data without any assurance about what other (non-training) samples look like**
- The model is more likely to run into poor local optima
- The model only captures the mode of the distribution of faces, whereas most face-like images are in the tail of the distribution

Q3: The face-generating model is more likely to generate face-like images if it were trained with a differentiable loss function that explicitly evaluates if the outputs look like faces or not, True or False

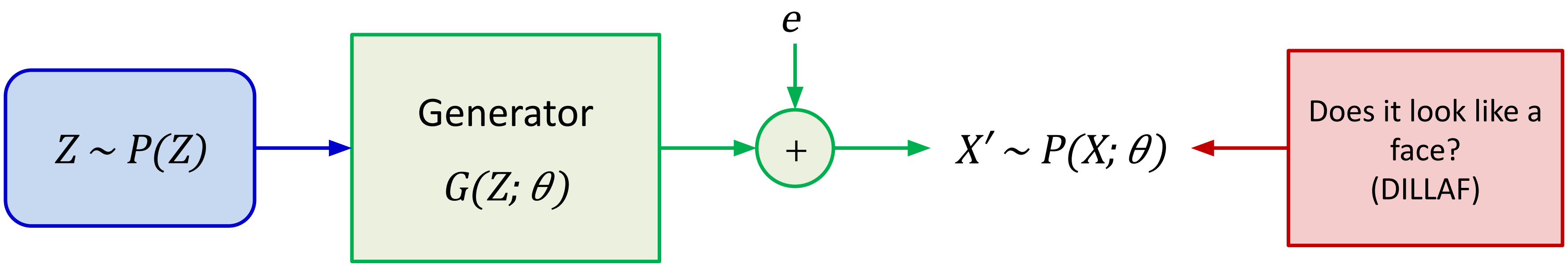
- **True**
- False

Replace the negative log likelihood with a more relevant loss



What is a good “DILLAF” loss?

Replace the negative log likelihood with a more relevant loss



What is a good “DILLAF” loss?

Enter: **GANs**

What are GANs

Generative Adversarial Networks

Generative Model which generates data similar to training data (like VAEs for example)

What are GANs

Generative **Adversarial** Networks

Generative Model which generates data similar to training data (like VAEs for example)

Adversarial Training using two competing (adversarial) networks that are trying to beat each other

What are GANs

Generative Adversarial Networks

Generative Model which generates data similar to training data (like VAEs for example)

Adversarial Training using two competing (adversarial) networks that are trying to beat each other

Deep Neural **Networks**

What are GANs

Generative Adversarial Networks

Generative Model which generates data similar to training data (like VAEs for example)

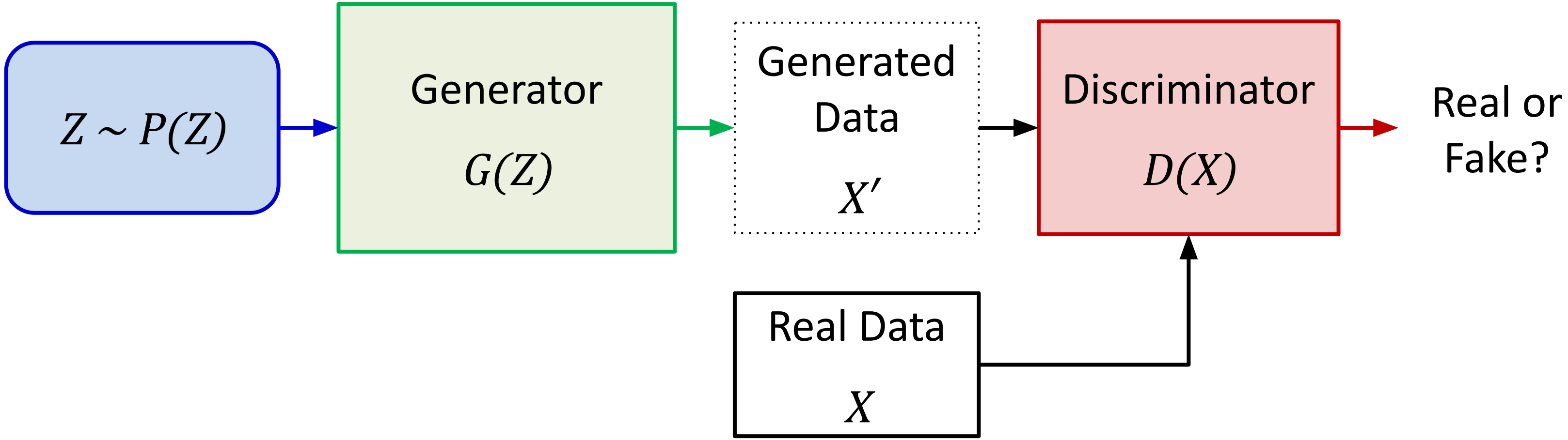
Goal is to **model the training data distribution** $P(X)$ so we can generate new samples

Adversarial Training using two competing (adversarial) networks that are trying to beat each other

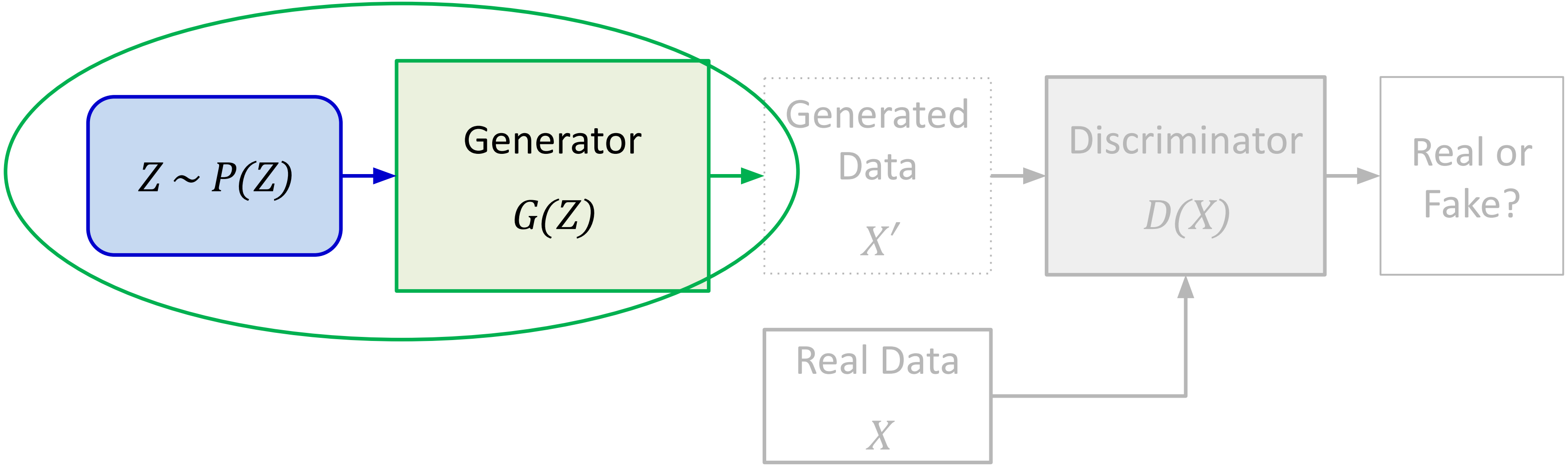
We use a **"Generator"** and a **"Discriminator"** to train (where the Discriminator is our "DILLAF" loss!)

Deep Neural **Networks**

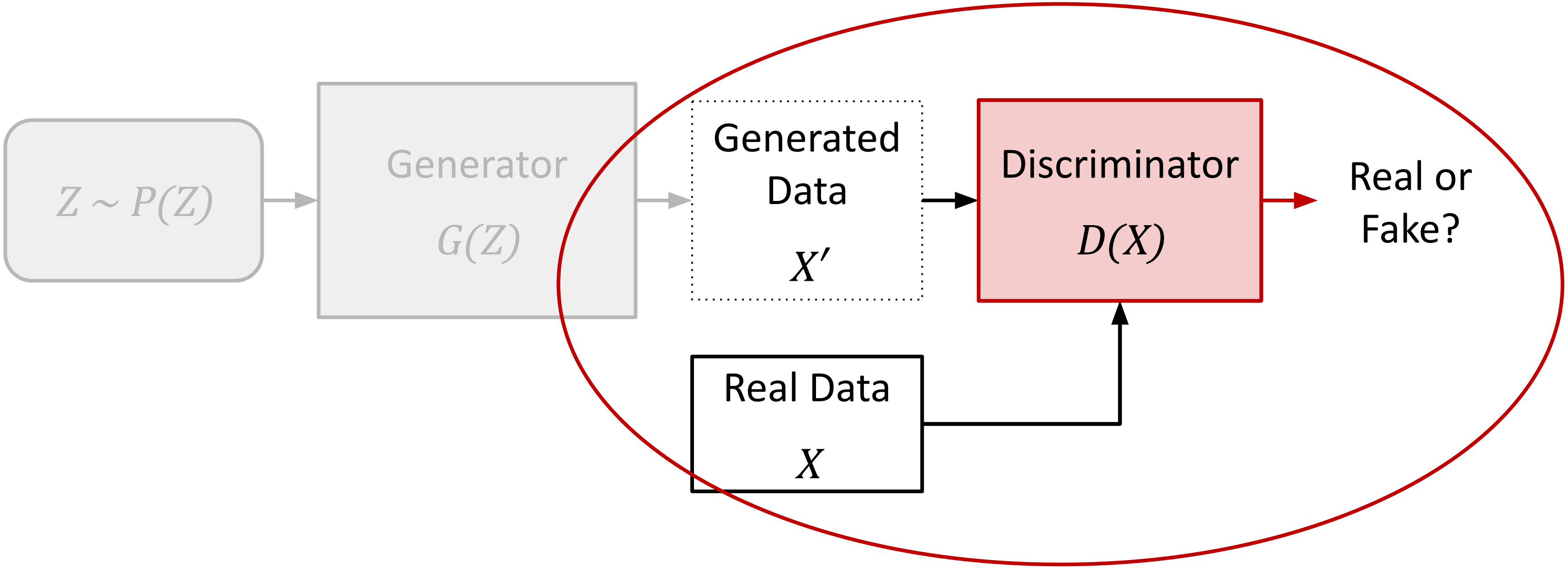
How GANs work



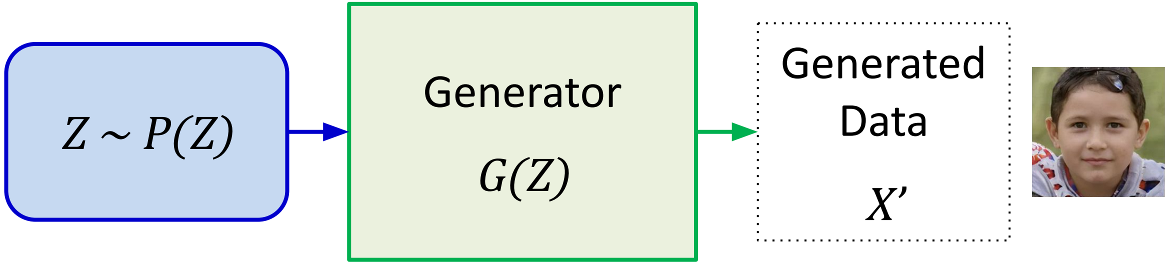
How GANs work



How GANs work

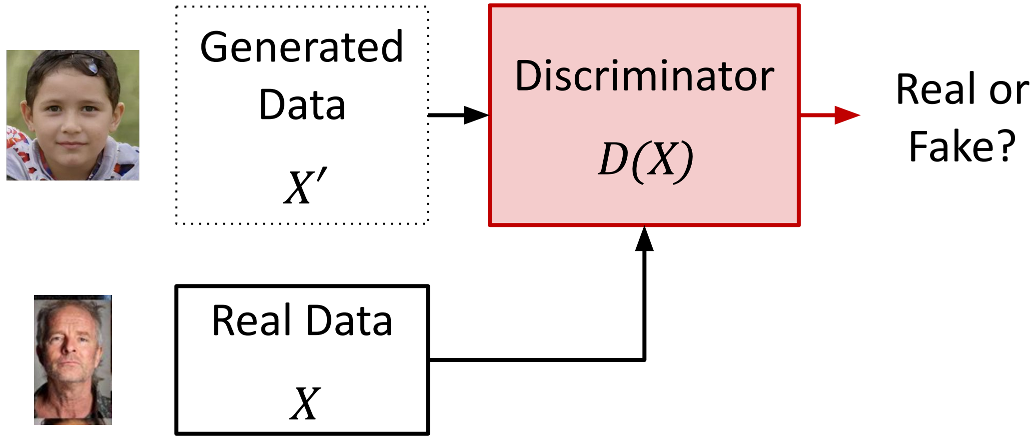


The Generator



- The generator produces realistic looking $X' = G(z)$ from the latent vector Z
    - Generator input X can be sampled from a known prior (e.g., a standard Gaussian)
- Goal: We want the generated distribution $P_G(X)$ to match the true data distribution $P_X(X)$
    - $P_G(X)$ is just easier notation for $P_{X'}(X)$, which is the probability that a generated sample takes on the value X

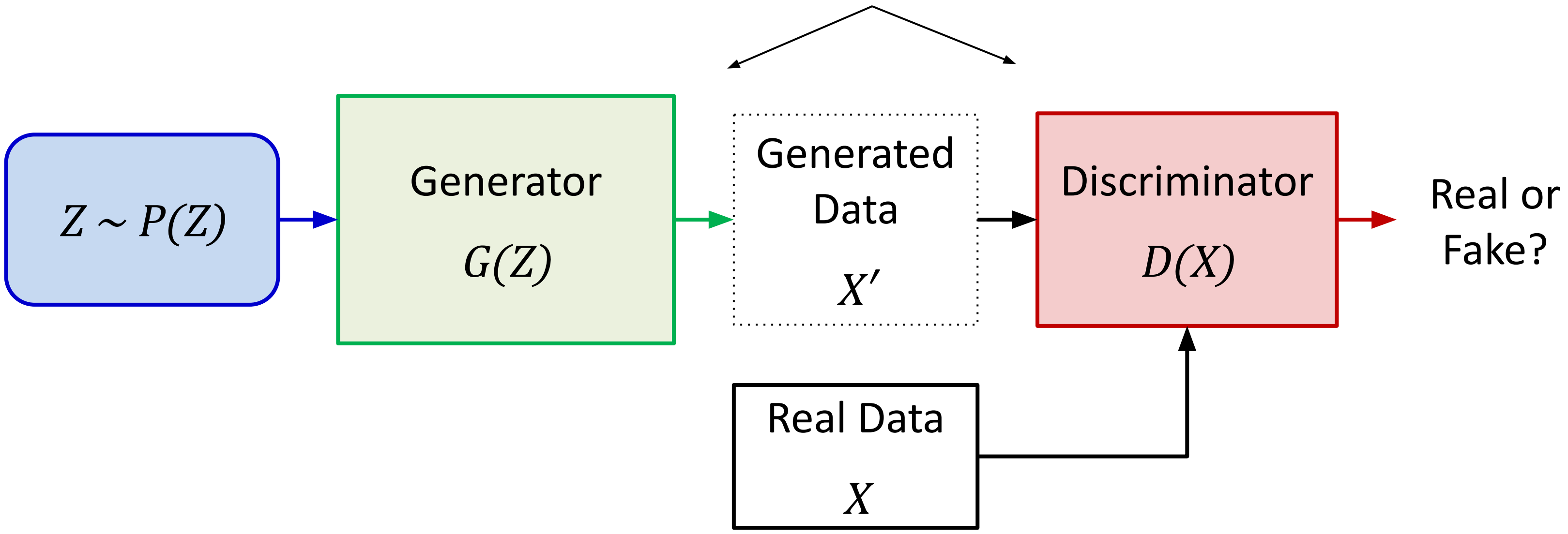
The Discriminator



- The Discriminator D(X) is trained to distinguish between the real and generated (fake) data
  - Specifically, data produced by the generator
  - If a perfect discriminator is fooled, the real and generated data cannot be distinguished

Training GANs

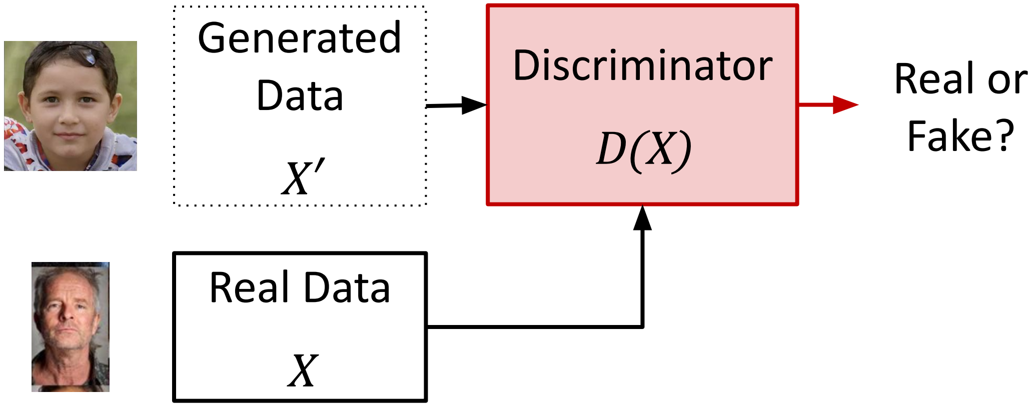
Both Generator and Discriminator need to be trained together



But first, some notation

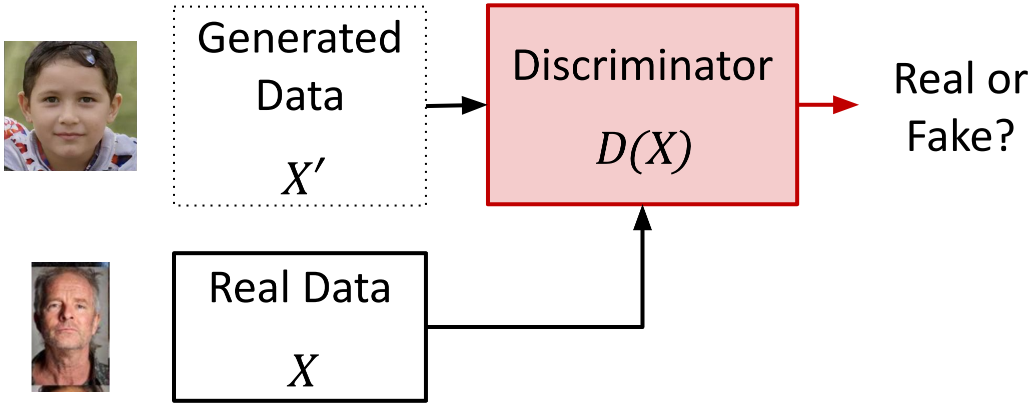
x	Data sample
z	Latent input noise vector
P_X	Distribution of real data
P_G	Distribution of generated data
P_Z	Distribution of latent input noise vector
$G(z; \theta_G)$	Generator (the function itself)
$D(x; \theta_D)$	Discriminator (the function itself)
$G(z)$ or x'	Generator output
$D(x)$ or $D(G(z))$	Discriminator output

Training the Discriminator



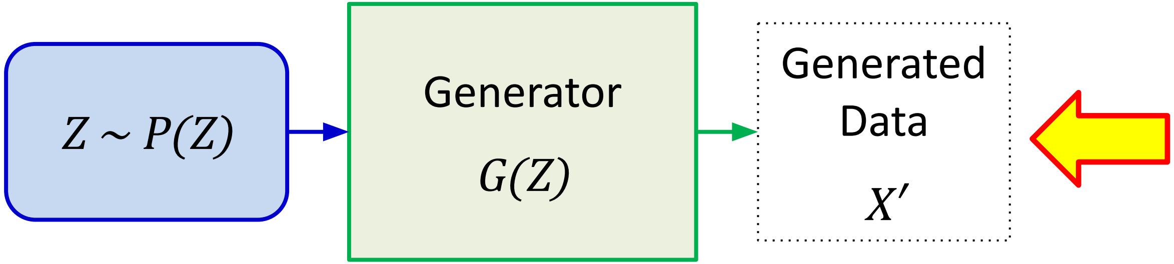
- Fed real and synthetic examples
- Aims to minimize classification loss → Minimize error between actual and predicted
- $D(x) = 1$ for real faces, $D(x) = 0$ for synthetic faces

Training the Discriminator



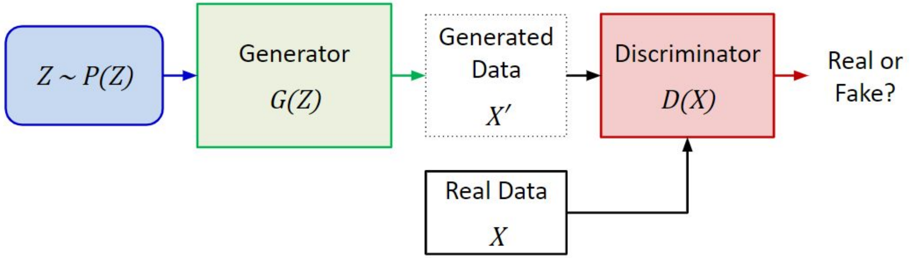
- Fed real and synthetic examples
- Aims to minimize classification loss → Minimize error between actual and predicted
- $D(x) = 1$ for real faces, $D(x) = 0$ for synthetic faces
  - Maximize $log\ (D(X))$ for real faces
  - Maximize $log\ (1 - D(X'))$ for synthetic faces

Training the Generator



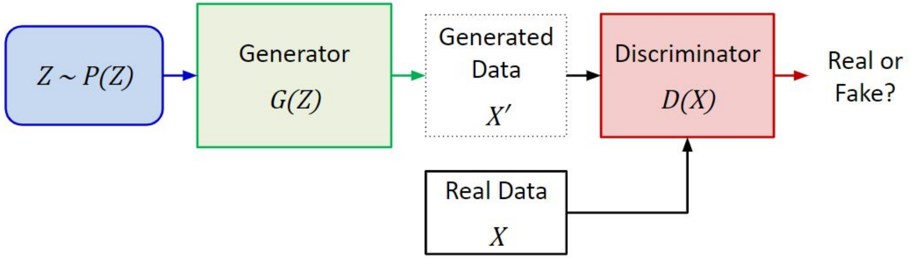
- The discriminator loss is propagated back to the generator
- Aims to *maximize* the discriminator loss (we want to “fool” the discriminator)
- Trained such that $D(G(Z)) = 1$ (i.e., $1 - D(G(Z)) = 0$)
  - Minimize $log\ (1 - D(G(Z))$

The GAN formulation



- Discriminator
  - For real data X, maximize $log\ (D(X))$
  - For synthetic data, maximize $log\ (1 - D(X'))$
- Generator
  - Minimize $log\ (1 - D(X'))$

The GAN formulation

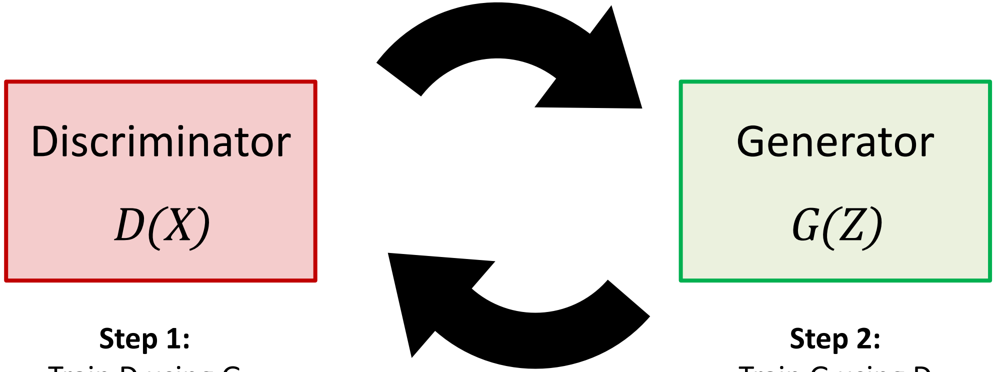


- The original GAN formulation is therefore a min-max optimization

Optimize: $$\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$$

- Objectives
  - D: $D(X) = 1$ and $D(G(Z)) = 0$
  - G: $D(G(Z)) = 1$

Training GANs



Step 1:
Train D using G

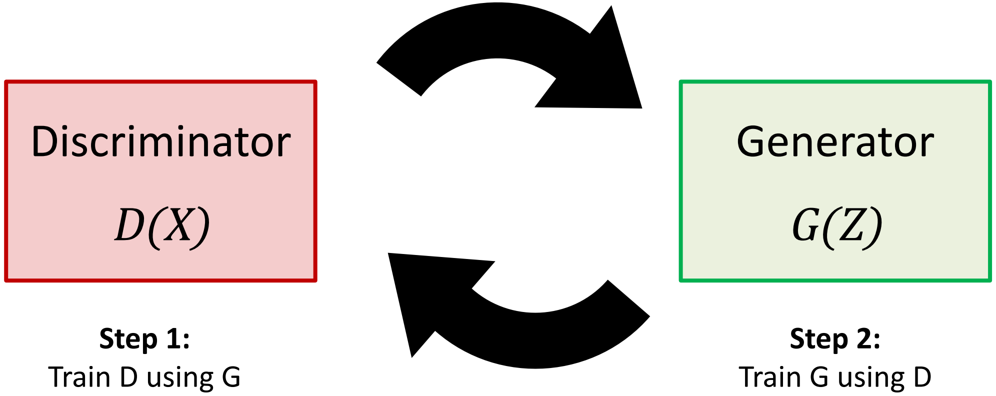
Step 2:
Train G using D

Optimize: $$\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$$

If the discriminator is undertrained, it provides sub-optimal feedback to the generator

If discriminator is overtrained, there is no local feedback for marginal improvements

Training GANs



Optimize: $$\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$$

The discriminator is not needed after convergence

Training GANs

```
for num_epochs do:

        for k_steps do:
                {z(1)… z(m)} ~ P_Z (Sample m noise vectors)
                {x(1)… x(m)} ~ P_X (Sample m data points)
```

$$L_D \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$

$$g_{\theta_D} \leftarrow \nabla_{\theta_D} L_D$$

$$\theta_D \leftarrow \theta_D + \alpha \cdot g_{\theta_D}$$

```
        end for

        {z(1)… z(m)} ~ P_Z (Sample m noise vectors)
```

$$L_G \leftarrow \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

$$g_{\theta_G} \leftarrow \nabla_{\theta_G} L_G$$

$$\theta_G \leftarrow \theta_G - \alpha \cdot g_{\theta_G}$$

```
end for
```

Hyperparameter.
Goodfellow et al. use k = 1

In practice, this saturates early in training. We can instead maximize log (D(G(z))) for better gradients.

Goodfellow et al. (2014), Generative Adversarial Networks.

Poll 3

Q1: When training a GAN, which component must you train first

- The discriminator
- The generator

Q2: Which component is updated more frequently

- The discriminator
- The generator

Poll 3

Q1: When training a GAN, which component must you train first

- **The discriminator**
- The generator

Q2: Which component is updated more frequently

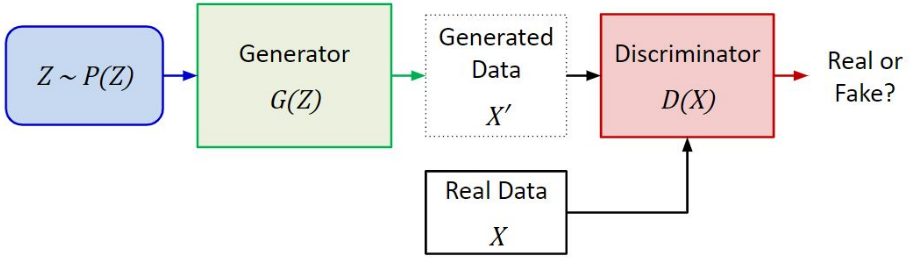
- **The discriminator**
- The generator

The discriminator is the "DILLAF" loss. Training the loss is more important, since this is what ***guides*** the training!

Learning Objectives

- ✓ Generative vs Discriminative models
- ✓ Explicit vs Implicit models
- ✓ The insufficiency of Maximum Likelihood Estimation for learning GANs
  - ✓ Using a Discriminator network for losses
- ❑ How GANs train
- ❑ Benefits and challenges of GANs
- ❑ Learning paradigms (learning through comparison)
  - ❑ Comparison by Ratios and the emergence of the Jensen Shannon Divergence
  - ❑ Comparison by Differences and the use of Wasserstein distance
  - ❑ Zero-sum vs Non-zero-sum
- ❑ Variants of GANs

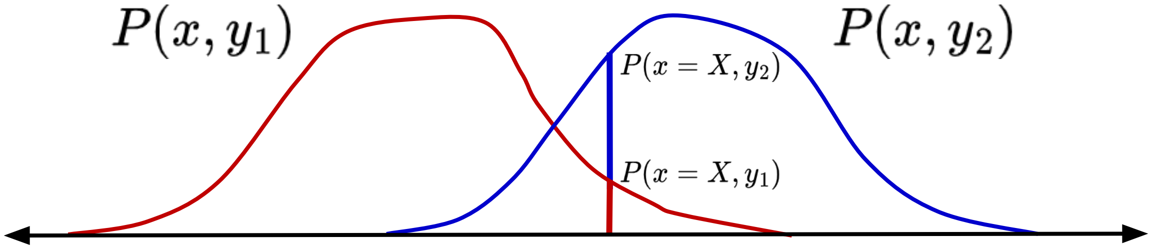
The GAN formulation



- How does this work when each piece is optimized?
  - We will consider the optimal Discriminator first…
  - Then the optimal Generator

The optimal discriminator (binary classification)

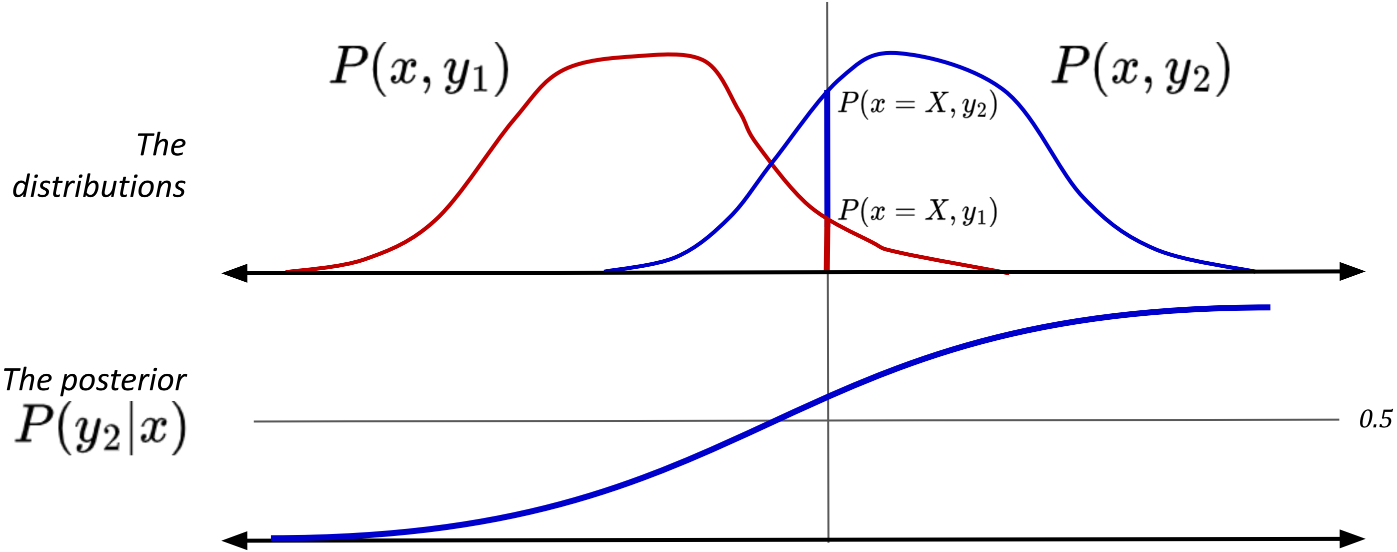
The distributions



The posterior probability of the classes for any instance $x = X$ is:

$$P(y_i|X) = \frac{P(X, y_i)}{P(X, y_1) + P(X, y_2)}$$

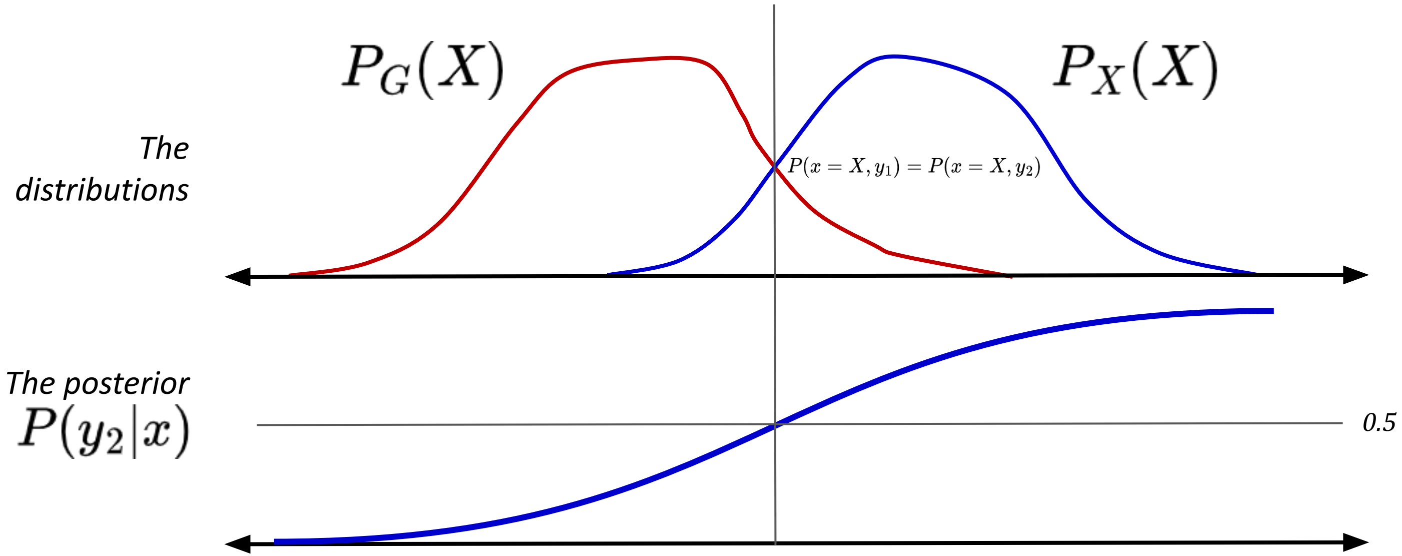
The optimal discriminator (binary classification)



The posterior probability of the classes for any instance $x = X$ is:

$$P(y_i|X) = \frac{P(X,y_i)}{P(X,y_1)+P(X,y_2)}$$

The optimal discriminator (binary classification)

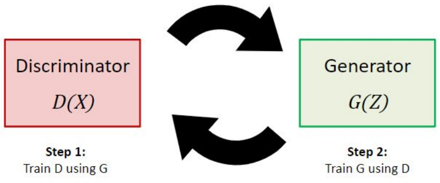


Assuming a uniform prior, the optimal discriminator in our case will be a Bayesian Classifier

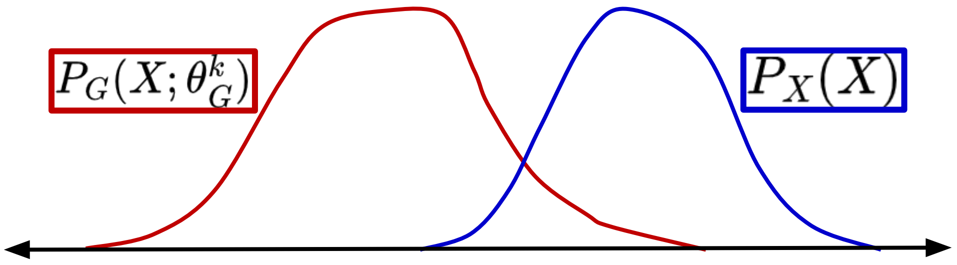
$$D(X) = \frac{P_X(X)}{P_X(X)+P_G(X)}$$

Iterative Training

Recall our training procedure:



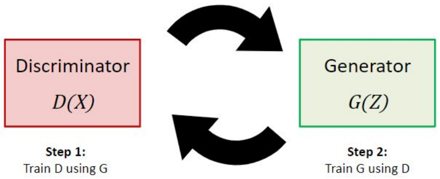
Optimize: $\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$



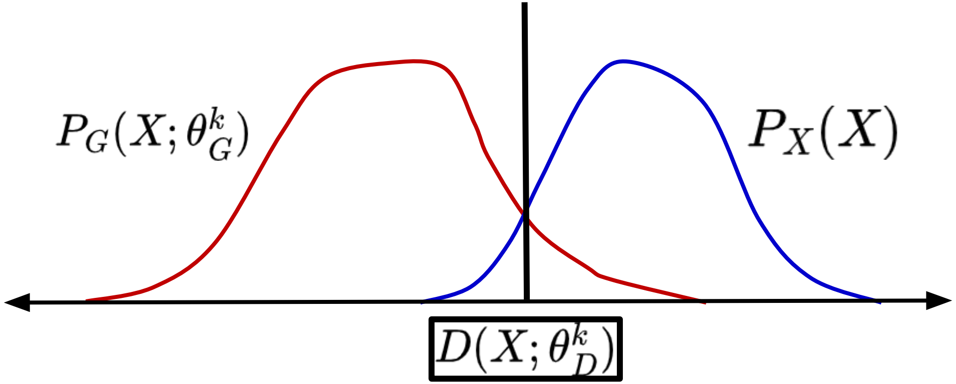
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



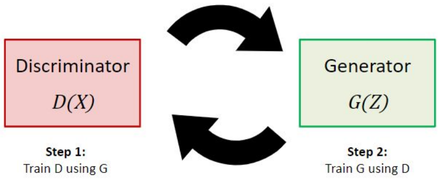
Optimize: $\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$



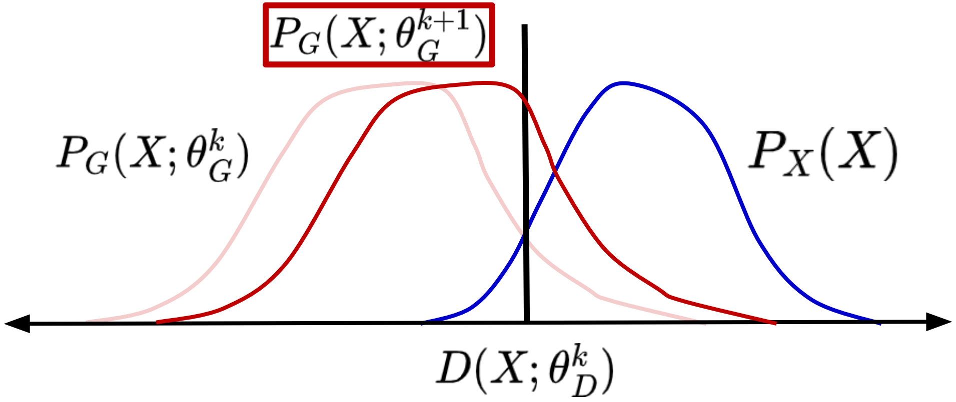
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



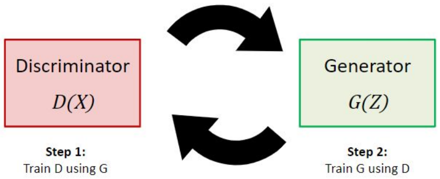
Optimize: $\min_G \max_D \mathbb{E}_{x\sim P_X} \log D(X) + \mathbb{E}_{z\sim P_Z} \log(1 - D(G(z)))$



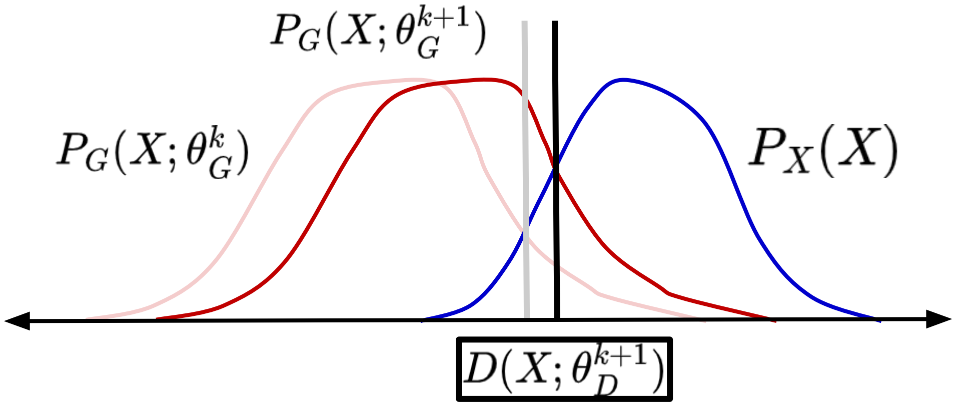
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



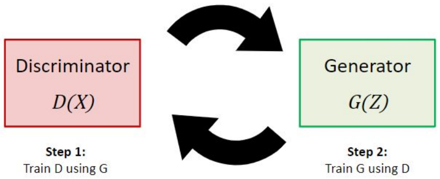
Optimize: $\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$



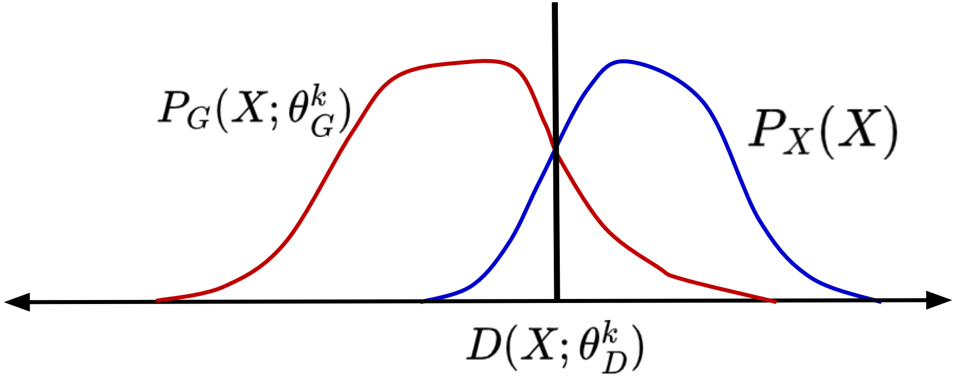
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



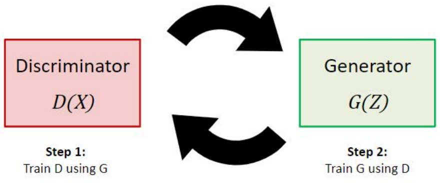
Optimize: $\min_G \max_D \mathbb{E}_{x\sim P_X} \log D(X) + \mathbb{E}_{z\sim P_Z} \log(1 - D(G(z)))$



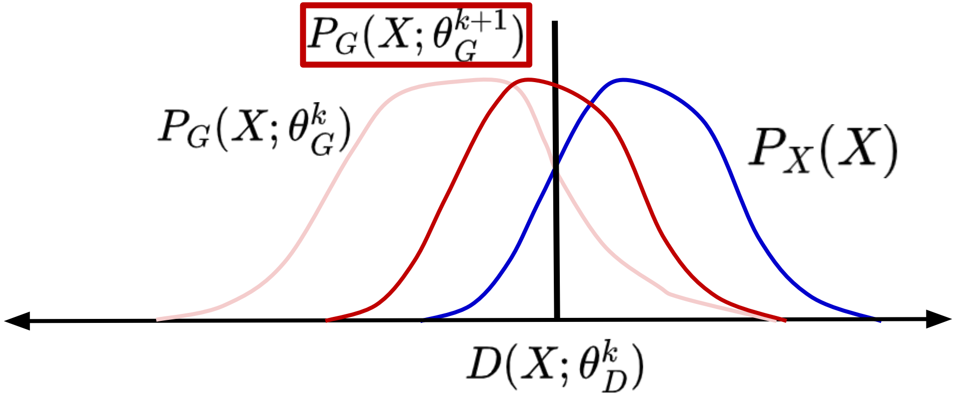
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



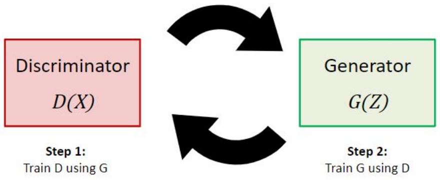
Optimize: $\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$



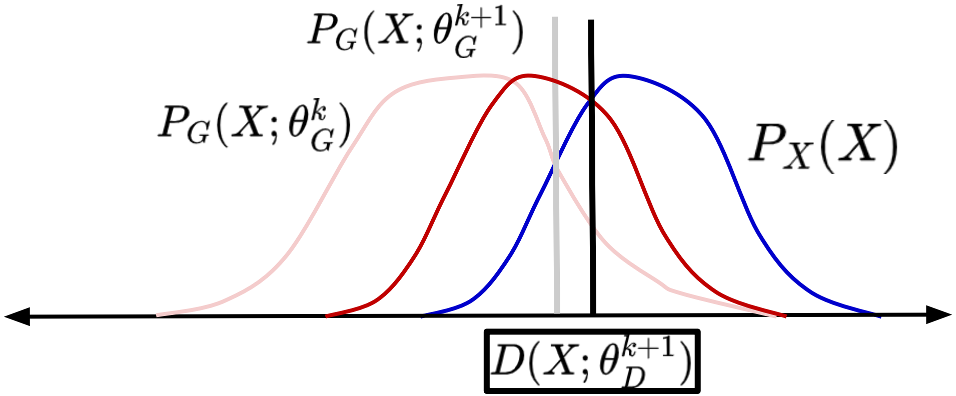
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



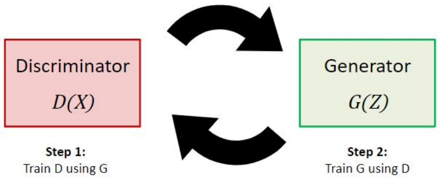
Optimize: $\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$



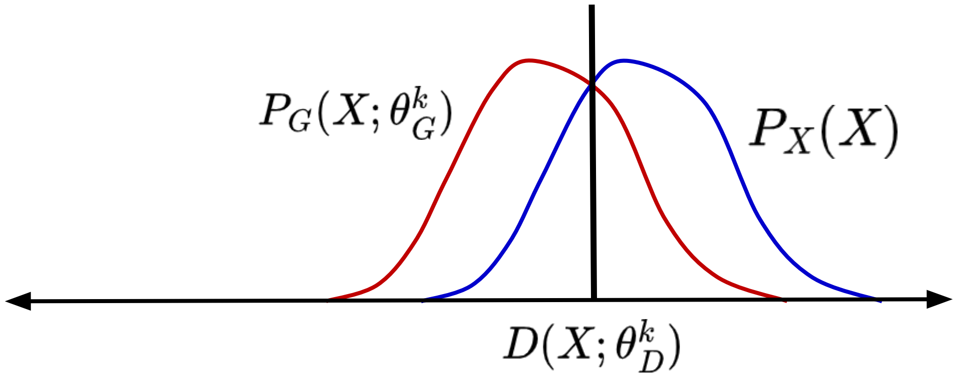
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



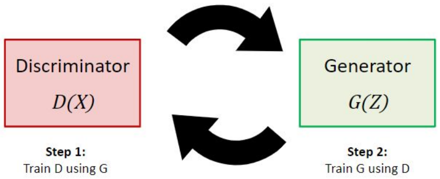
Optimize: $\min_G \max_D \mathbb{E}_{x\sim P_X} \log D(X) + \mathbb{E}_{z\sim P_Z} \log(1 - D(G(z)))$



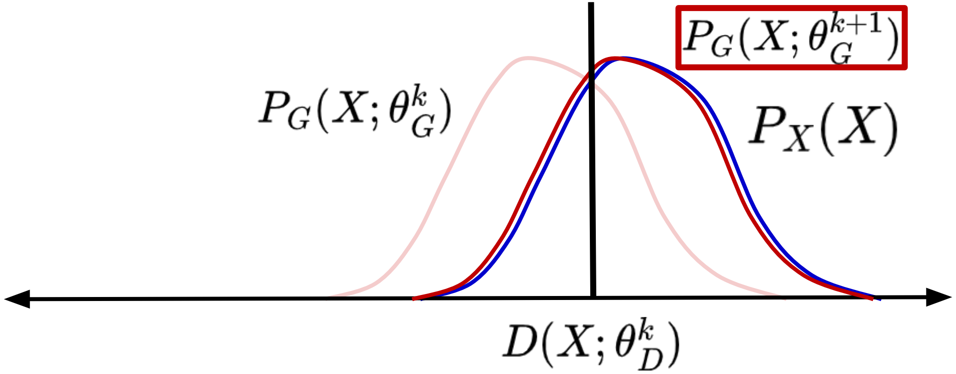
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



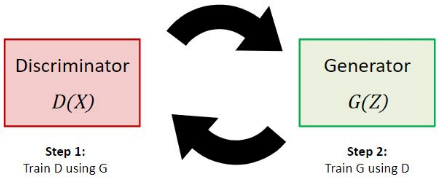
Optimize: $\min_G \max_D \mathbb{E}_{x \sim P_X} \log D(X) + \mathbb{E}_{z \sim P_Z} \log(1 - D(G(z)))$



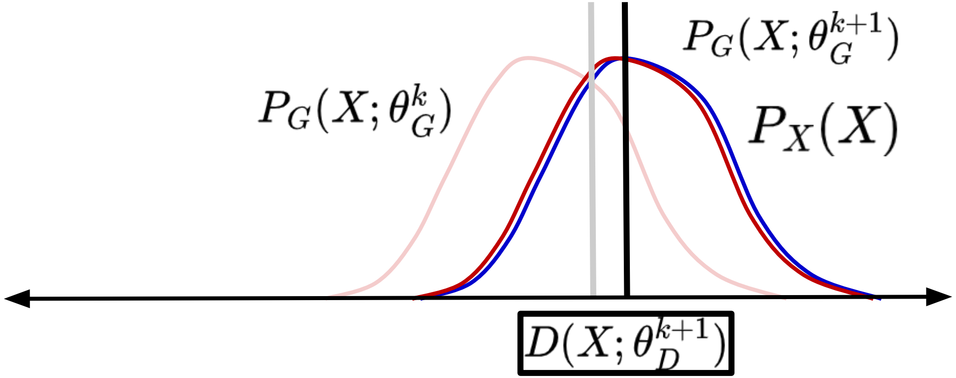
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



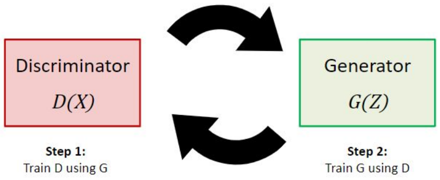
Optimize: $\min_G \max_D \mathbb{E}_{x\sim P_X} \log D(X) + \mathbb{E}_{z\sim P_Z} \log(1 - D(G(z)))$



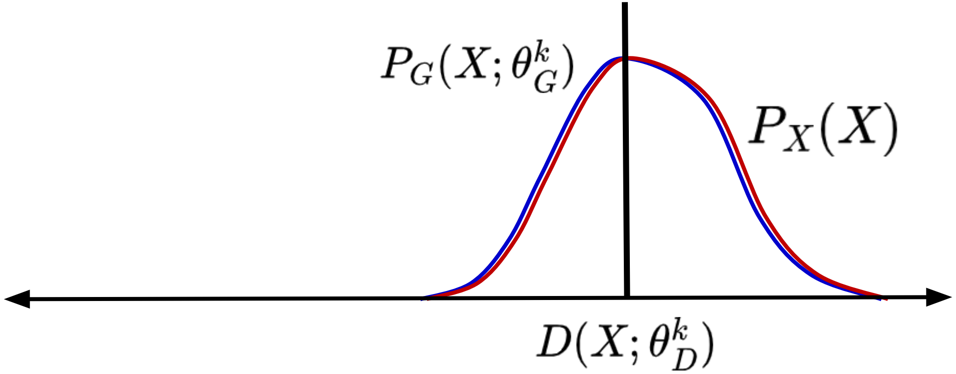
- Start with a training distribution and a generator distribution that is untrained
- Fit a discriminator
- Update the generator to “fool” the discriminator

Iterative Training

Recall our training procedure:



Optimize: $\min_G \max_D \mathbb{E}_{x\sim P_X} \log D(X) + \mathbb{E}_{z\sim P_Z} \log(1 - D(G(z)))$



- In the limit, the Generator's distribution will sit perfectly on the true distribution, and the Discriminator will be random.
- The derivative of D(X) wrt X will be zero → No further updates

Min-Max Stationary Point

- There exists a stationary point…
  - If the generated data exactly matches the real data (discriminator outputs 0.5 for all inputs)
  - If the discriminator outputs 0.5, the gradients for the generator are flat, so the generator does not learn
  - This is true of a perfect discriminator paired with a very good generator. **However, it is also true of a random discriminator.**
- Stationary points need not be stable.
  - Depends on the exact GAN formulation
  - The generator may overshoot or oscillate around the optimum
  - A discriminator with unlimited capacity can still assign an arbitrarily large distance to 2 similar distributions.

Benefits and Challenges

- GANs produce clear crisp results for many problems
- However, they have stability issues and are difficult to train
  - Mode Collapse or Mode Hopping
    - Improvements can be made by using larger batch sizes, increasing discriminator expressivity, regularizing the discriminator and generator, and other optimization methods.
  - Low variability/diversity in outputs
  - Poor gradients

Benefits and Challenges

- GANs produce clear crisp results for many problems
- However, they have stability issues and are difficult to train
  - Mode Collapse or Mode Hopping
    - Improvements can be made by using larger batch sizes, increasing discriminator expressivity, regularizing the discriminator and generator, and other optimization methods.
  - Low variability/diversity in outputs
  - Poor gradients

Benefits and Challenges

- GANs produce clear crisp results for many problems
- However, they have stability issues and are difficult to train
  - **Mode Collapse** or Mode Hopping
    - Improvements can be made by using larger batch sizes, increasing discriminator expressivity, regularizing the discriminator and generator, and other optimization methods.
  - Low variability/diversity in outputs
  - Poor gradients

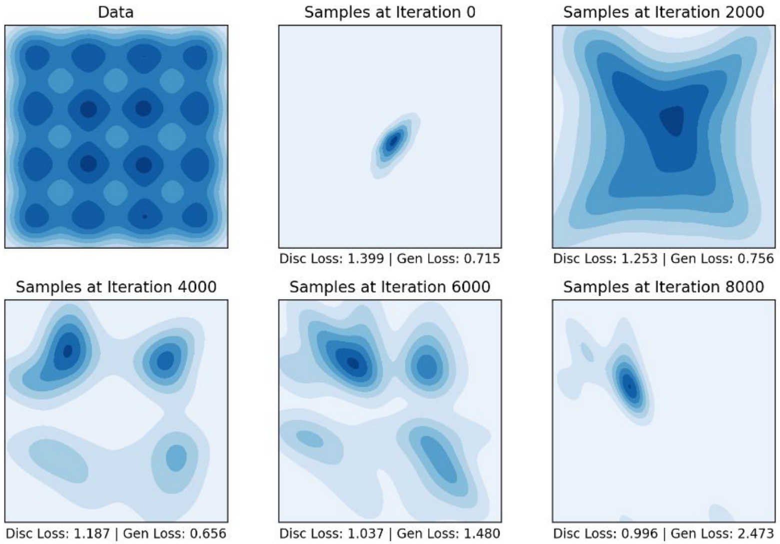
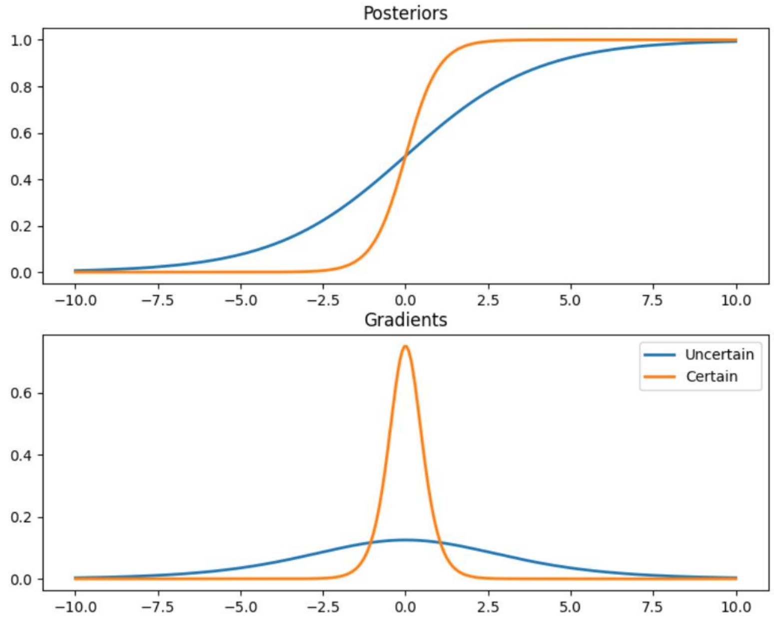


Illustration of Mode Collapse from Murphy (2023), Fig. 26.6, with code available at https://github.com/probml/pyprobml/blob/master/notebooks/book2/26/gan_mixture_of_gaussians.ipynb.

Benefits and Challenges

- GANs produce clear crisp results for many problems
- However, they have stability issues and are difficult to train
    - Mode Collapse or Mode Hopping
        - Improvements can be made by using larger batch sizes, increasing discriminator expressivity, regularizing the discriminator and generator, and other optimization methods.
    - Low variability/diversity in outputs
    - **Poor gradients as Discriminator gets better**



Poll 4

Identify potential reasons a GAN could fail

- Generator always generates the same face that fools the discriminator
- The divergence may have poor derivatives preventing the model from learning
- The discriminator may be random resulting in no derivatives
- The discriminator may be too certain, resulting in no derivatives

Poll 4

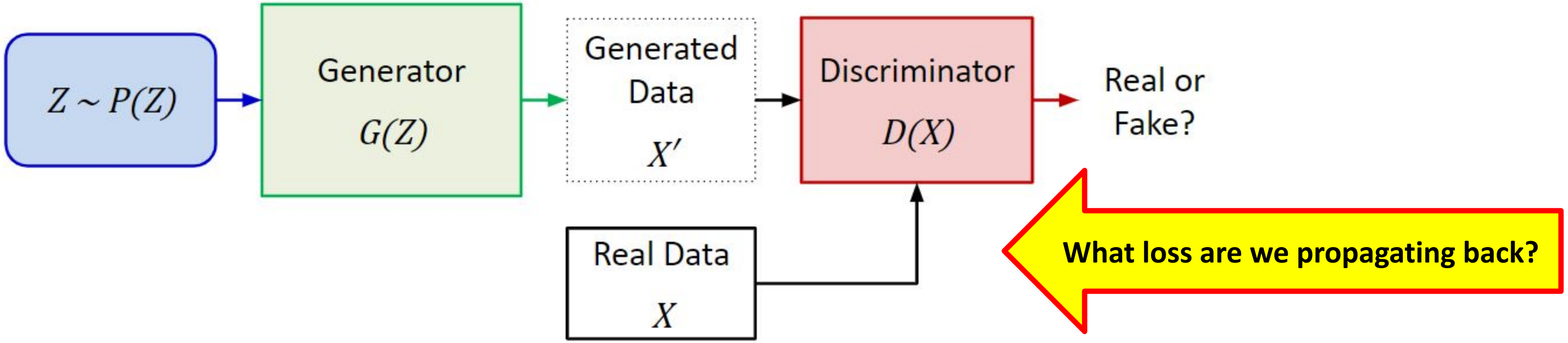
Identify potential reasons a GAN could fail

- **Generator always generates the same face that fools the discriminator**
- **The divergence may have poor derivatives preventing the model from learning**
- **The discriminator may be random resulting in no derivatives**
- **The discriminator may be too certain, resulting in no derivatives**

Learning Objectives

- ✓ Generative vs Discriminative models
- ✓ Explicit vs Implicit models
- ✓ The insufficiency of Maximum Likelihood Estimation for learning GANs
  - ✓ Using a Discriminator network for losses
- ✓ How GANs train
- ✓ Benefits and challenges of GANs
- ❑ Learning paradigms (learning through comparison)
  - ❑ Comparison by Ratios and the emergence of the Jensen Shannon Divergence
  - ❑ Comparison by Differences and the use of Wasserstein distance
  - ❑ Zero-sum vs Non-zero-sum
- ❑ Variants of GANs

What is the learning paradigm?



What loss are we actually using?

- KL Divergence?

$$KL(P,Q) = \sum_X P(X) log\left(\frac{P(X)}{Q(X)}\right)$$

$$KL(Q,P) = \sum_X Q(X) log\left(\frac{Q(X)}{P(X)}\right)$$

- Are there any problems with this?

What loss are we actually using?

- KL Divergence?

$$KL(P,Q) = \sum_X P(X) log\left(\frac{P(X)}{Q(X)}\right)$$

$$KL(Q,P) = \sum_X Q(X) log\left(\frac{Q(X)}{P(X)}\right)$$

- Are there any problems with this?

(1) KL is not symmetric
  (a) One sacrifices image quality
  (b) One sacrifices image diversity
(2) We run into issues if either P or Q become zero

Jensen Shannon Divergence

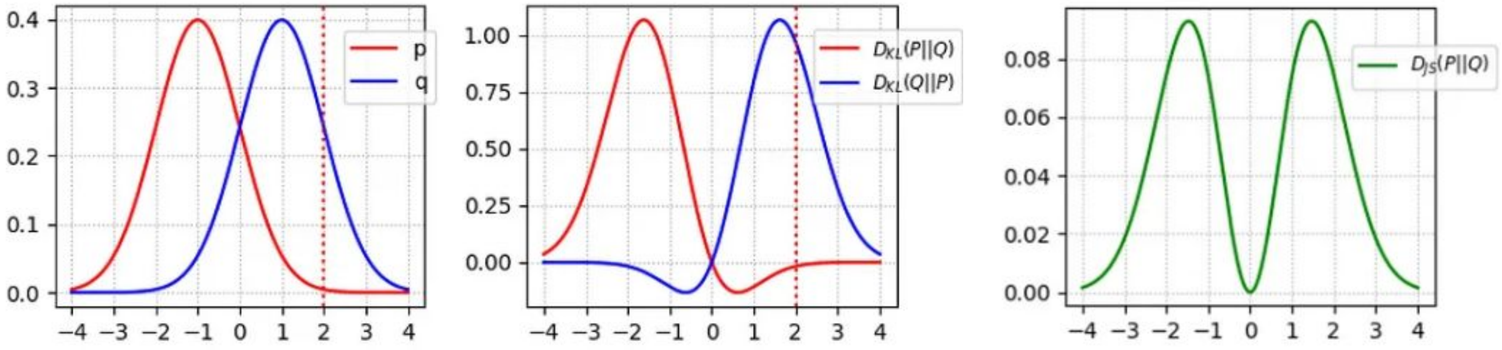
- Symmetric alternate to KL Divergence that removes issues with P or Q of 0.
- Does not exaggerate instances where one of the distributions assigns 0 probability

$$JSD(P,Q) = \frac{1}{2}KL(P, \frac{P+Q}{2}) + \frac{1}{2}KL(Q, \frac{P+Q}{2})$$

Jensen Shannon Divergence

- Symmetric alternate to KL Divergence that removes issues with P or Q of 0.
- Does not exaggerate instances where one of the distributions assigns 0 probability

$$JSD(P,Q) = \frac{1}{2}KL(P, \frac{P+Q}{2}) + \frac{1}{2}KL(Q, \frac{P+Q}{2})$$



Distributions KL Divergence JS Divergence

Jonathan Hui, "GAN — Why it is so hard to train Generative Adversarial Networks!" https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Jensen Shannon Divergence

- Symmetric alternate to KL Divergence that removes issues with P or Q of 0.
- Does not exaggerate instances where one of the distributions assigns 0 probability

$$JSD(P,Q) = \frac{1}{2}KL(P, \frac{P+Q}{2}) + \frac{1}{2}KL(Q, \frac{P+Q}{2})$$

- This isn't simply a convenience we take. It emerges as a natural consequence if we want to compare the distributions of our generator and our true data using the *ratio* of their density functions!

Jensen Shannon Divergence

- Symmetric alternate to KL Divergence that removes issues with P or Q of 0.
- Does not exaggerate instances where one of the distributions assigns 0 probability

$$JSD(P,Q) = \frac{1}{2}KL(P, \frac{P+Q}{2}) + \frac{1}{2}KL(Q, \frac{P+Q}{2})$$

- This isn't simply a convenience we take. It emerges as a natural consequence if we want to compare the distributions of our generator and our true data using the *ratio* of their density functions!

Jensen Shannon Divergence

- Recall we converted the ratio of density functions into a binary classification problem

$$\frac{P_X(X)}{P_G(X)} = \frac{D(X)}{1 - D(X)} \quad \Rightarrow \quad D^*(x) = \frac{P_X(X)}{P_X(X) + P_G(X)}$$

Jensen Shannon Divergence

- Recall we converted the ratio of density functions into a binary classification problem

$$\frac{P_X(X)}{P_G(X)} = \frac{D(X)}{1-D(X)} \quad \Rightarrow \quad D^*(x) = \frac{P_X(X)}{P_X(X)+P_G(X)}$$

- Using a binary cross entropy loss for the parameterized discriminator, we have

$$\begin{aligned} Div &= \mathbb{E}\left[y \log D(X;\theta_D) + (1-y)\log(1-D(X;\theta_D))\right] \\ &= \frac{1}{2}\mathbb{E}_{P_X}\left[\log D(X;\theta_D)\right] + \frac{1}{2}\mathbb{E}_{P_G}\left[\log(1-D(X;\theta_D))\right] \end{aligned}$$

Jensen Shannon Divergence

- Recall we converted the ratio of density functions into a binary classification problem

$$\frac{P_X(X)}{P_G(X)} = \frac{D(X)}{1-D(X)} \quad \Rightarrow \quad D^*(x) = \frac{P_X(X)}{P_X(X) + P_G(X)}$$

- Using a binary cross entropy loss for the parameterized discriminator, we have

$$\begin{aligned} Div &= \mathbb{E}\left[y \log D(X;\theta_D) + (1-y)\log(1 - D(X;\theta_D))\right] \\ &= \frac{1}{2}\mathbb{E}_{P_X}\left[\log D(X;\theta_D)\right] + \frac{1}{2}\mathbb{E}_{P_G}\left[\log(1 - D(X;\theta_D))\right] \end{aligned}$$

- Substituting in the optimal discriminator, we get an objective with the JSD in it!

$$Div = 2JSD(P_X, P_G) - \log 4$$

Jensen Shannon Divergence

- Recall we converted the ratio of density functions into a binary classification problem

$$\frac{P_X(X)}{P_G(X)} = \frac{D(X)}{1-D(X)} \quad \Rightarrow \quad D^*(x) = \frac{P_X(X)}{P_X(X)+P_G(X)}$$

- Using a binary cross entropy loss for the parameterized discriminator, we have

$$\begin{aligned} Div &= \mathbb{E}\left[y \log D(X;\theta_D) + (1-y)\log(1-D(X;\theta_D))\right] \\ &= \frac{1}{2}\mathbb{E}_{P_X}\left[\log D(X;\theta_D)\right] + \frac{1}{2}\mathbb{E}_{P_G}\left[\log(1-D(X;\theta_D))\right] \end{aligned}$$

- Substituting in the optimal discriminator, we get an objective with the JSD in it!

$$Div = 2JSD(P_X, P_G) - \log 4$$

This is a consequence of making a comparison of the ***ratios*** between distributions

Let’s take a quick step back

What are the “decisions” we are making here vs what are the “inherent” elements?

- Learning by comparison (pretty inherent). However…
    - We have focused on a comparison through *ratios*
    - What about a comparison through *differences*?
- We have assumed a “zero-sum” adversarial structure…
    - This need not be the case (and we have actually already seen a variation)

Let’s take a quick step back

What are the “decisions” we are making here vs what are the “inherent” elements?

- Learning by comparison (pretty inherent). However…
  - We have focused on a comparison through *ratios*
  - What about a comparison through *differences*?
- We have assumed a “zero-sum” adversarial structure…
  - This need not be the case (and we have actually already seen a variation)

Learning by Comparison

Ratios

- E.g., density ratios (we just did this) or f-divergence

Learning by Comparison

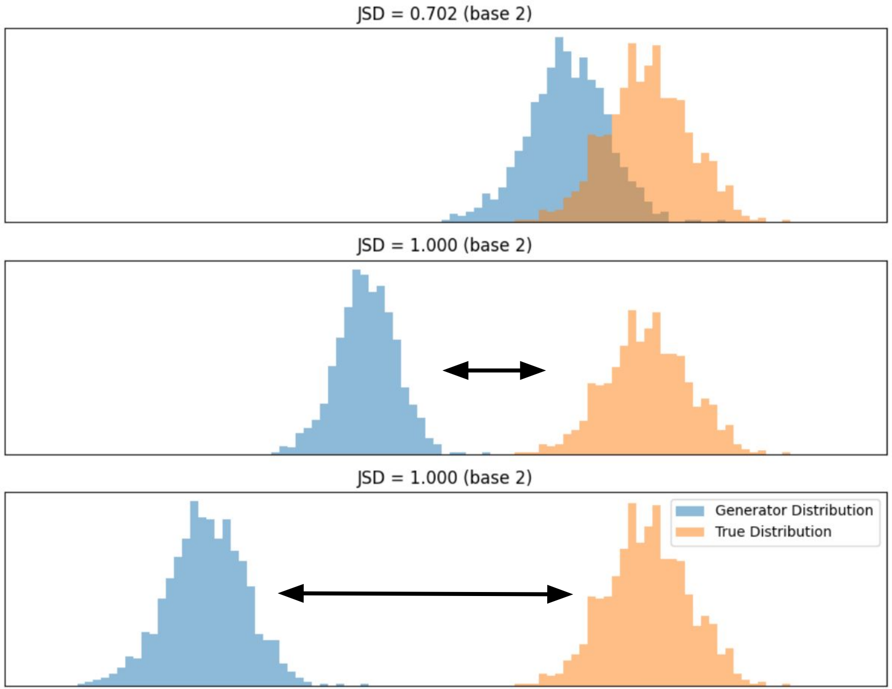
Ratios

- E.g., density ratios (we just did this) or f-divergence
- Poor behavior when the generator distribution and true distribution do not have overlapping support
  - When the true distribution assigns a non-zero probability but the generator assigns a zero probability, the ratios in KL blow up, and the JSD becomes log 2 no matter what.

Learning by Comparison

Ratios

- E.g., density ratios (we just did this) or f-divergence
- Poor behavior when the generator distribution and true distribution do not have overlapping support
    - When the true distribution assigns a non-zero probability but the generator assigns a zero probability, the ratios in KL blow up, and the **JSD becomes log 2 no matter what**.



Learning by Comparison

Ratios

- E.g., density ratios (we just did this) or f-divergence
- Poor behavior when the generator distribution and true distribution do not have overlapping support
    - When the true distribution assigns a non-zero probability but the generator assigns a zero probability, the ratios in KL blow up, and the JSD becomes log 2 no matter what.

Learning by Comparison

Ratios

- E.g., density ratios (we just did this) or f-divergence
- Poor behavior when the generator distribution and true distribution do not have overlapping support
  - When the true distribution assigns a non-zero probability but the generator assigns a zero probability, the ratios in KL blow up, and the JSD becomes log 2 no matter what.

Differences

- E.g., integral probability metrics (IPM) or moment matching
- Wasserstein GANs (example of IPM)
- Introduces “smoothness”

Wasserstein GAN

Earth Mover's Distance or **Optimal Transport**

How much distance you need to cover to move all the parts of one distribution to the other?

Wasserstein GAN

Earth Mover's Distance or **Optimal Transport**

How much distance you need to cover to move all the parts of one distribution to the other?

Inf (Topic for real analysis):
Infimum, the closest thing to a lower bound you can get. (alt to a **min**)

$$W(P_X, P_G) = \inf_{\gamma \in \Gamma(P_X, P_G)} \mathbb{E}_{(x,y)\sim\gamma(x,y)}[||x - y||]$$

Mapping of which x goes to which y

Distance between the points

Wasserstein GAN

Earth Mover's Distance or **Optimal Transport**

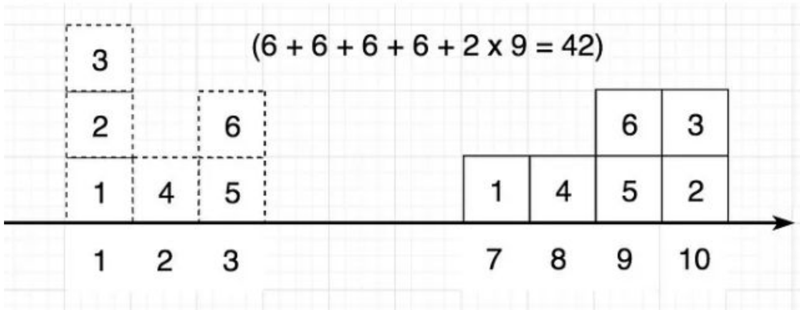
How much distance you need to cover to move all the parts of one distribution to the other?

Inf (Topic for real analysis):
Infimum, the closest thing to a lower bound you can get. (alt to a **min**)

$$W(P_X, P_G) = \inf_{\gamma \in \Gamma(P_X, P_G)} \mathbb{E}_{(x,y)\sim \gamma(x,y)}[||x - y||]$$

Mapping of which x goes to which y

Distance between the points



γ_1

	7	8	9	10
1	1	0	0	2
2	0	1	0	0
3	0	0	2	0

Wasserstein GAN

Earth Mover's Distance or **Optimal Transport**

How much distance you need to cover to move all the parts of one distribution to the other?

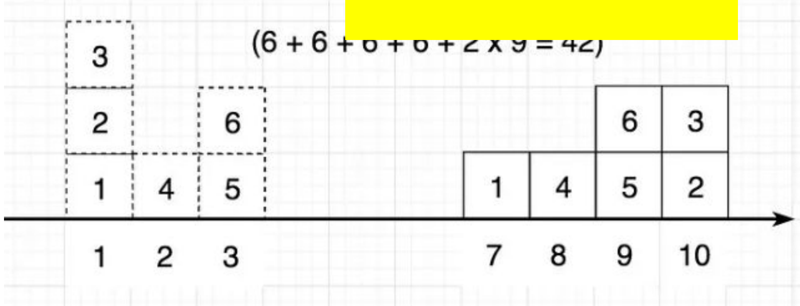
Inf (Topic for real analysis):
Infimum, the closest thing to a lower bound you can get. (alt to a **min**)

$$W(P_X, P_G) = \inf_{\gamma \in \Gamma(P_X, P_G)} \mathbb{E}_{(x,y)\sim \gamma(x,y)}[||x - y||]$$

Mapping of which x

Tends to be intractable

Distance between the points



γ_1

	7	8	9	10
1	1	0	0	2
2	0	1	0	0
3	0	0	2	0

Jonathan Hui, "GAN — Wasserstein GAN & WGAN-GP"; Arjovsky et al. (2017), "Wasserstein GAN.

Wasserstein GAN

Dealing with the intractable…

(for reference, don't worry about the details here; a nice resource is here)

- Simplify with Kantorovich-Rubinstein inequality
- Find a 1-Lipschitz function using a network similar to a Discriminator (a "Critic")
- Enforce the Lipschitz constraint
  - Authors use clipping, but acknowledge that "Weight clipping is a clearly terrible way to enforce a Lipschitz constraint"

Wasserstein GAN

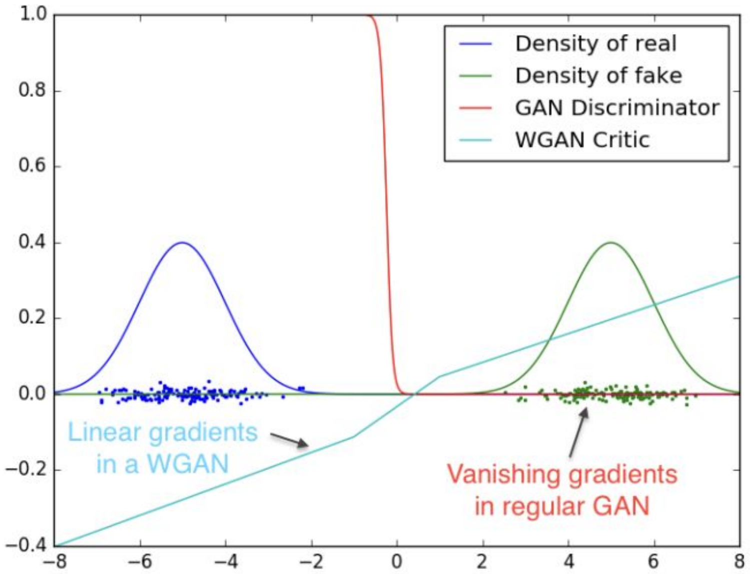


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

Arjovsky et al. (2017), "Wasserstein GAN."

Let's take a quick step back

What are the "decisions" we are making here vs what are the "inherent" elements?

- Learning by comparison (pretty inherent). However…
    - We have focused on a comparison through *ratios*
    - What about a comparison through *differences*?
- We have assumed a "zero-sum" adversarial structure…
    - This need not be the case (and we have actually already seen a variation)

Non-Zero-Sum Losses

- Recall our min-max (zero-sum) learning objective:

Optimize: $$\min_G \max_D \mathbb{E}_{x\sim P_X} \log D(X) + \mathbb{E}_{z\sim P_Z} \log(1 - D(G(z)))$$

- Rather than have the generator minimize the probability of the discriminator labeling its examples as fake, why not have it maximize the probability of the discriminator classifying its examples as real (recall note on slide 51)
    - Known as “non-saturating loss”
    - Subtle difference, but enjoys better gradients early in training (when the generator is performing poorly).
    - Can still recover the zero-sum formulation if we want

Training GANs

```
for num_epochs do:

        for k_steps do:
```

Hyperparameter.
Goodfellow et al. use k = 1

$\{z^{(1)} \ldots z^{(m)}\} \sim P_Z$ (Sample m noise vectors)

$\{x^{(1)} \ldots x^{(m)}\} \sim P_X$ (Sample m data points)

$$L_D \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$

$$g_{\theta_D} \leftarrow \nabla_{\theta_D} L_D$$

$$\theta_D \leftarrow \theta_D + \alpha \cdot g_{\theta_D}$$

```
        end for
```

$\{z^{(1)} \ldots z^{(m)}\} \sim P_Z$ (Sample m noise vectors)

$$L_G \leftarrow \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

$$g_{\theta_G} \leftarrow \nabla_{\theta_G} L_G$$

$$\theta_G \leftarrow \theta_G - \alpha \cdot g_{\theta_G}$$

```
end for
```

In practice, this saturates early in training. We can instead maximize log (D(G(z))) for better gradients.

Goodfellow et al. (2014), Generative Adversarial Networks.

Learning Objectives

- ✓ Generative vs Discriminative models
- ✓ Explicit vs Implicit models
- ✓ The insufficiency of Maximum Likelihood Estimation for learning GANs
  - ✓ Using a Discriminator network for losses
- ✓ How GANs train
- ✓ Benefits and challenges of GANs
- ✓ Learning paradigms (learning through comparison)
  - ✓ Comparison by Ratios and the emergence of the Jensen Shannon Divergence
  - ✓ Comparison by Differences and the use of Wasserstein distance
  - ✓ Zero-sum vs Non-zero-sum
- ❑ Variants of GANs

GANs in the wild

- Cycle GAN
- StarGAN
- Conditional GANs
- BiGAN
- …and many(!) more

GANs in the wild

- Cycle GAN
- StarGAN
- Conditional GANs
- BiGAN
- …and many(!) more

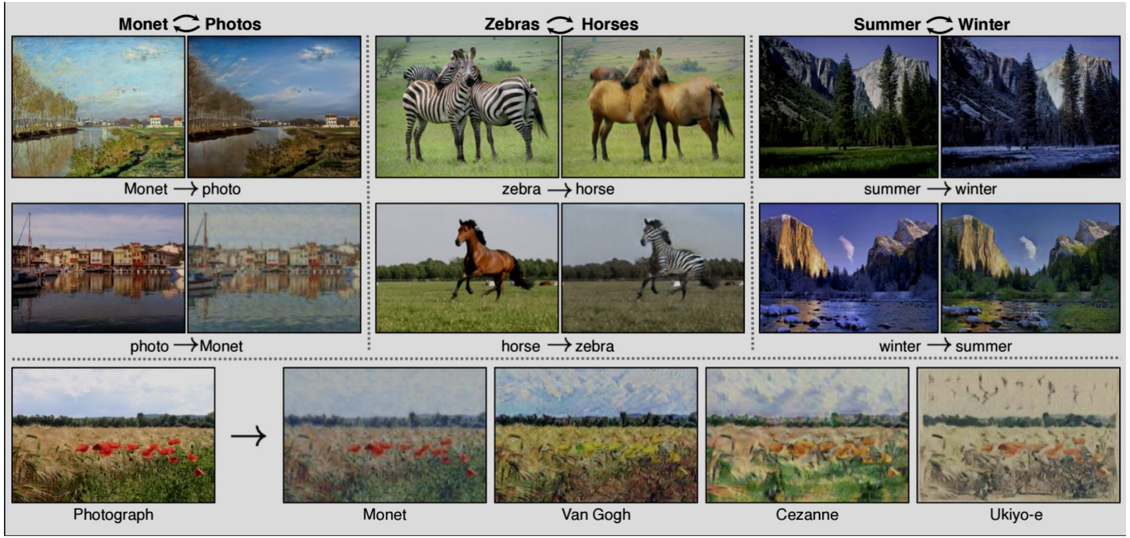


Image "translation." While a vanilla GAN will not retain information about the original image, Cycle GAN incorporates a reconstruction loss so that enough of the original is kept (such that it can be retrieved using a second generator).

Zhu et al. (2017), "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks."

GANs in the wild

- Cycle GAN
- StarGAN
- Conditional GANs
- BiGAN
- …and many(!) more

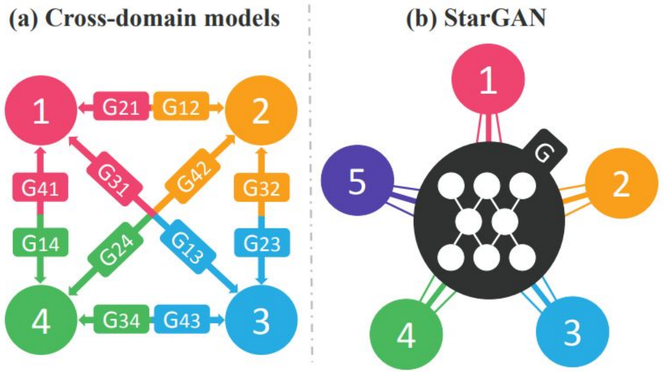


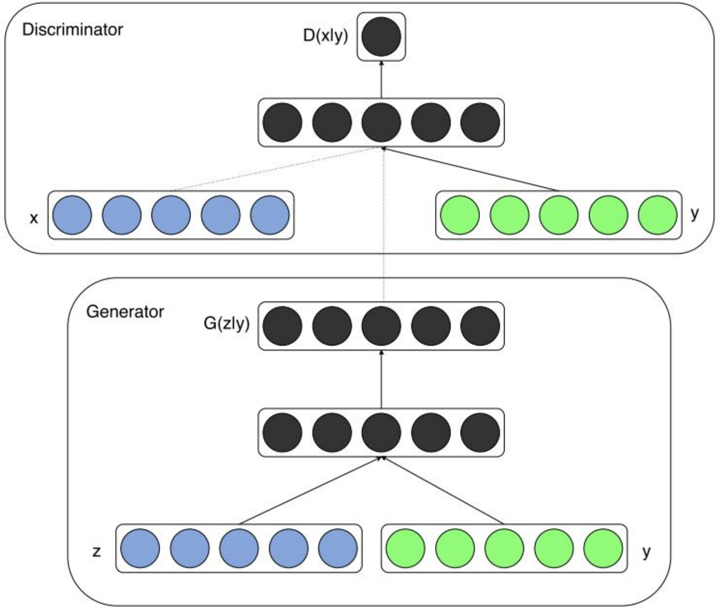
Figure 2. Comparison between cross-domain models and our proposed model, StarGAN. (a) To handle multiple domains, cross-domain models should be built for every pair of image domains. (b) StarGAN is capable of learning mappings among multiple domains using a single generator. The figure represents a star topology connecting multi-domains.

Image "translation." Ability to learn mappings across multiple domains with a single generator.

Choi et al. (2017), "StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation."

GANs in the wild

- Cycle GAN
- StarGAN
- Conditional GANs
- BiGAN
- …and many(!) more



Given paired data (x with some corresponding y, such as a class label or set of attributes), we can learn a conditional distribution.

Mirza & Osindero (2014), "Conditional Generative Adversarial Nets."

GANs in the wild

- Cycle GAN
- StarGAN
- Conditional GANs
- BiGAN
- …and many(!) more

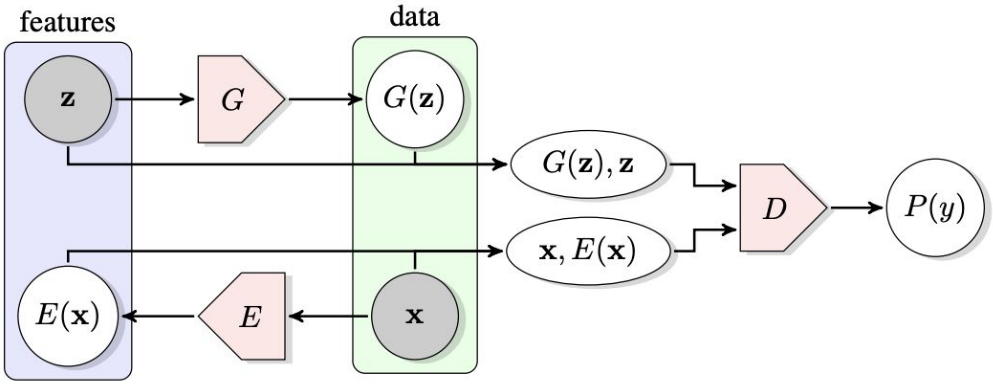


Figure 1: The structure of Bidirectional Generative Adversarial Networks (BiGAN).

Instead of mapping from latent space to feature space, we can map from the feature space to the latent space. The authors find the learned feature representations are useful for discriminative tasks among others.

Donahue et al. (2017), "Adversarial Feature Learning."

Learning Objectives

- ✓ Generative vs Discriminative models
- ✓ Explicit vs Implicit models
- ✓ The insufficiency of Maximum Likelihood Estimation for learning GANs
  - ✓ Using a Discriminator network for losses
- ✓ How GANs train
- ✓ Benefits and challenges of GANs
- ✓ Learning paradigms (learning through comparison)
  - ✓ Comparison by Ratios and the emergence of the Jensen Shannon Divergence
  - ✓ Comparison by Differences and the use of Wasserstein distance
  - ✓ Zero-sum vs Non-zero-sum
- ✓ Variants of GANs

Sources

- Arjovsky et al. (2017), "Wasserstein GAN," *Proceedings of Machine Learning Research*.
- Choi et al. (2017), "StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation."
- Donahue et al. (2017), "Adversarial Feature Learning," *ICLR*.
- Goodfellow et al. (2014), "Generative Adversarial Nets," *NeurIPS*.
- Hui, Jonathan, "GAN — Wasserstein GAN & WGAN-GP," available at https://jonathan-hui.medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490.
- Hui, Jonathan, "GAN — Why it is so hard to train Generative Adversarial Networks," available at https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b.
- Mirza & Osindero (2014), "Conditional Generative Adversarial Nets."
- Murphy (2023), *Probabilistic Machine Learning: Advanced Topics*, MIT Press.
- Theis et al. (2016), "A Note on the Evaluation of Generative Models," *NeurIPS*.
- Zhu et al. (2017), "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks."