

A Brief Introduction to Generative Adversarial Networks

Content

- **Generative model & discriminative model**
- **What's GANs?**
- **Math behind GANs?**
- **Characteristics of GANs**
- **Applications of GANs**

Generative model & discriminative model

- Given an **observable variable** X and a **target variable** Y , a **generative model** is a **statistical model** of the **joint probability distribution** on $X \times Y$, $P(X, Y)$;[1]
- A **discriminative model** is a model of the **conditional probability** of the target Y , given an observation x , symbolically, $P(Y|X = x)$; and
- Classifiers computed without using a probability model are also referred to loosely as "discriminative".

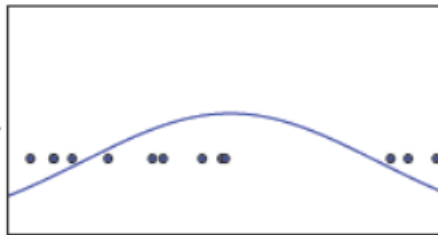
Generative model & discriminative model

- Discriminative models learn the boundary between classes
- Generative models model the distribution of the sample data

Density estimation



Observed instances
from an unknown distribution
 $P_{data}(x)$

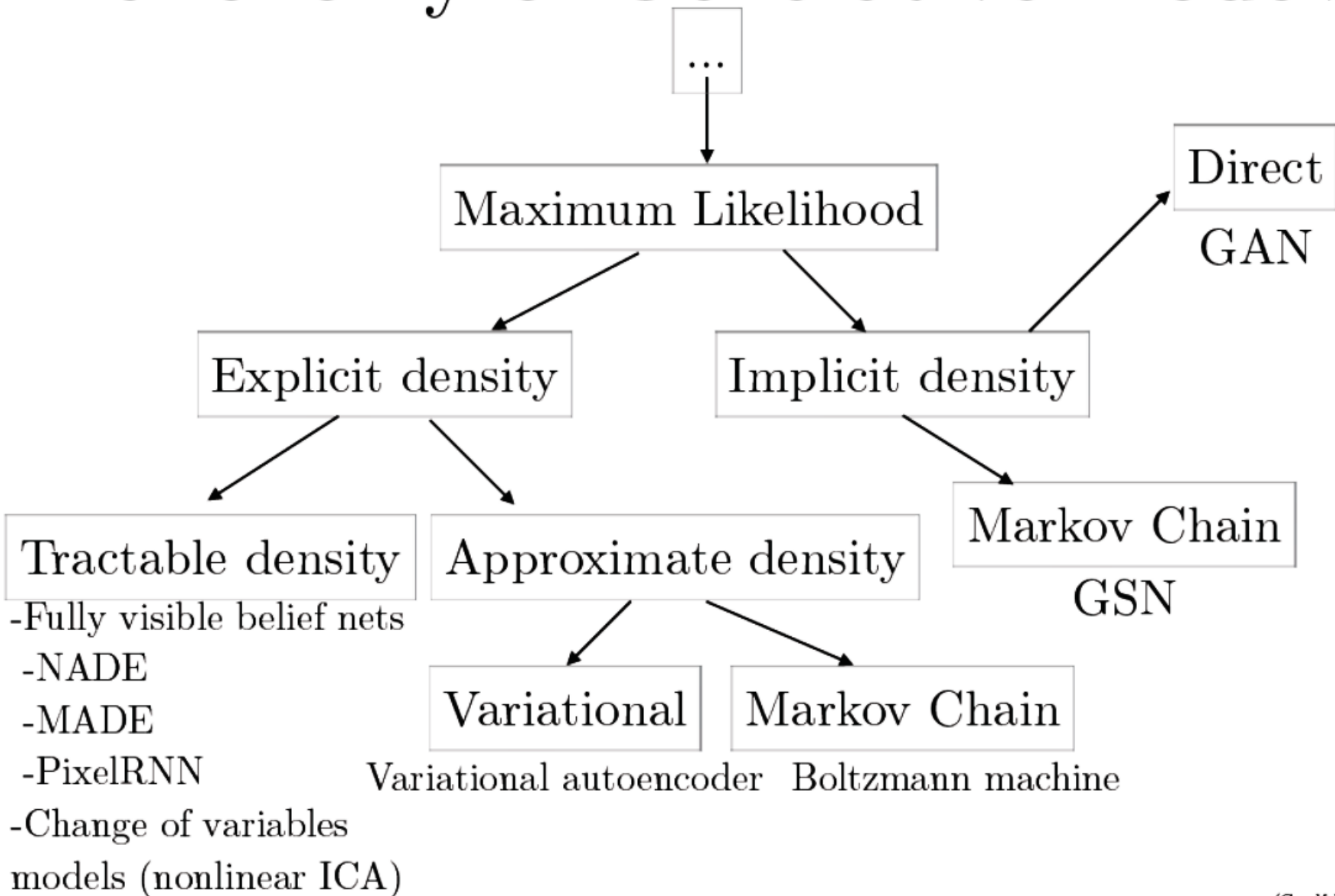


Approximate distribution
 $P_G(x)$ to the real distribution
 $P_{data}(x)$



New instance

Taxonomy of Generative Models



Generative models

Types of generative models are:

- Gaussian mixture model (and other types of mixture model)
- Hidden Markov model
- Probabilistic context-free grammar
- Bayesian network (e.g. Naive bayes, Autoregressive model)
- Averaged one-dependence estimators
- Latent Dirichlet allocation
- Boltzmann machine (e.g. Restricted Boltzmann machine, Deep belief network)
- Variational autoencoder
- Generative adversarial network
- Flow-based generative model

Discriminative models

- k-nearest neighbors algorithm
- Logistic regression
- Support Vector Machines
- Maximum-entropy Markov models
- Conditional random fields
- Neural networks

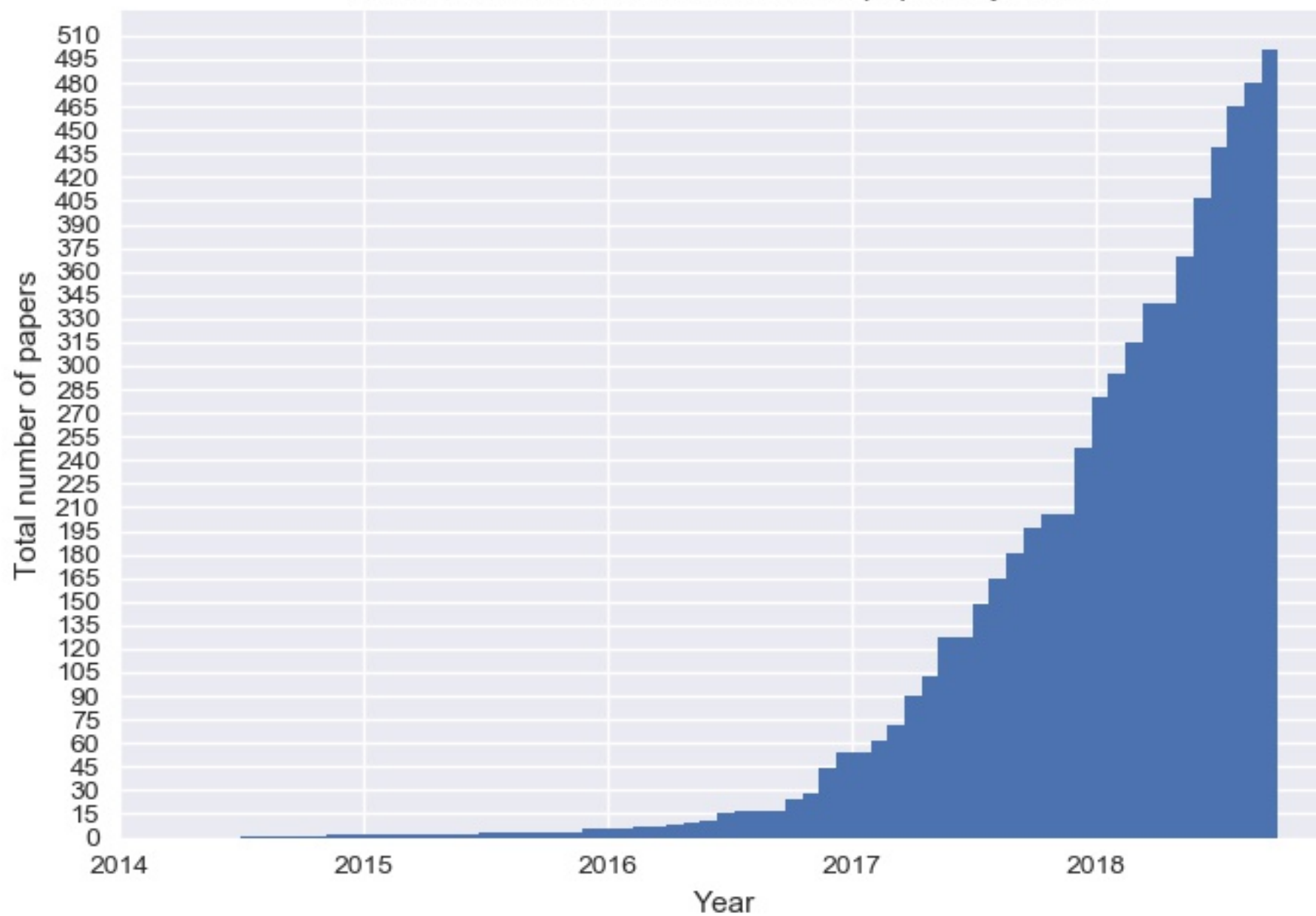
What is GAN?

- GANs are generative models devised by [Goodfellow et al.](#) in 2014.

Goodfellow I J, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[C]// International Conference on Neural Information Processing Systems. 2014

- Facebook's AI research director Yann LeCun [called adversarial training](#) “the most interesting idea in the last 10 years in ML.”
- GANs' potential is huge, because they can learn to mimic any distribution of data (such as images, music, speech, prose).

Cumulative number of named GAN papers by month



Generation problem

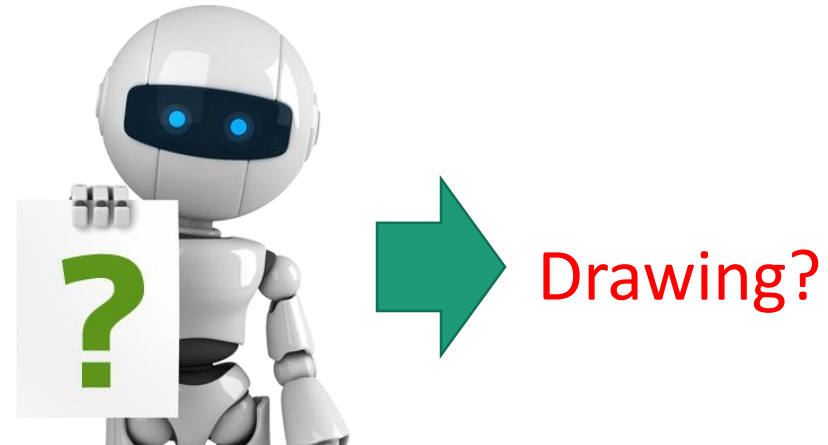
- Given a sample of x



What's the distribution of these images?

$P_{data}(x)$ (unknown)

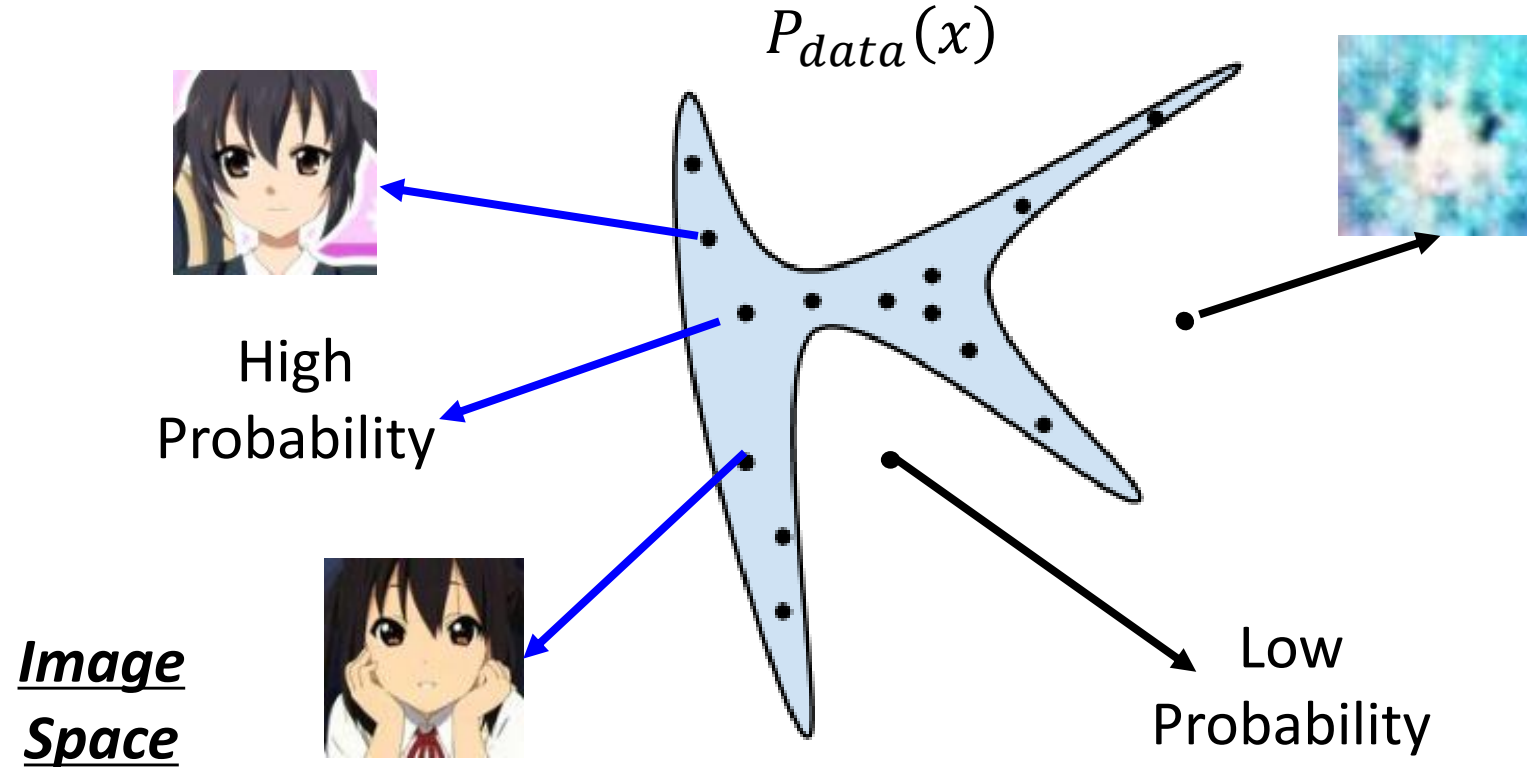
How to sample a new image from its distribution?



Generation

x : an image (a high-dimensional vector)

- We want to find data distribution $P_{data}(x)$



Maximum Likelihood Estimation

- There is a data distribution $P_{data}(x)$ (We don't know, but can sample from it.)
- We have a distribution $P_G(x; \theta)$ parameterized by θ
 - We want to find θ such that $P_G(x; \theta)$ close to $P_{data}(x)$
 - E.g. $P_G(x; \theta)$ is a Gaussian Mixture Model, θ are means and variances of the Gaussians

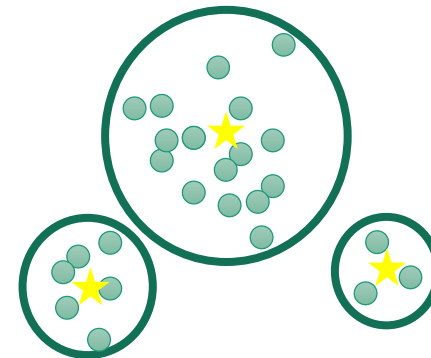
Sample $\{x^1, x^2, \dots, x^m\}$ from $P_{data}(x)$

We can compute $P_G(x^i; \theta)$

Likelihood of generating the samples

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

Find θ^* maximizing the likelihood



Maximum Likelihood Estimation = Minimize KL Divergence

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

$$= \arg \max_{\theta} \sum_{i=1}^m \log P_G(x^i; \theta) \quad \{x^1, x^2, \dots, x^m\} \text{ from } P_{data}(x)$$

$$\approx \arg \max_{\theta} E_{x \sim P_{data}}[\log P_G(x; \theta)]$$

$$= \arg \max_{\theta} \int_x P_{data}(x) \log P_G(x; \theta) dx - \int_x P_{data}(x) \log P_{data}(x) dx$$

$$= \arg \min_{\theta} KL(P_{data} || P_G)$$

How to define a general P_G ?

KL divergence & JS Divergence

(1) KL (Kullback–Leibler) divergence measures how one probability distribution p diverges from a second expected probability distribution q .

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

(2) Jensen–Shannon Divergence is another measure of similarity between two probability distributions, bounded by $[0,1][0,1]$. JS divergence is symmetric and more smooth.

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

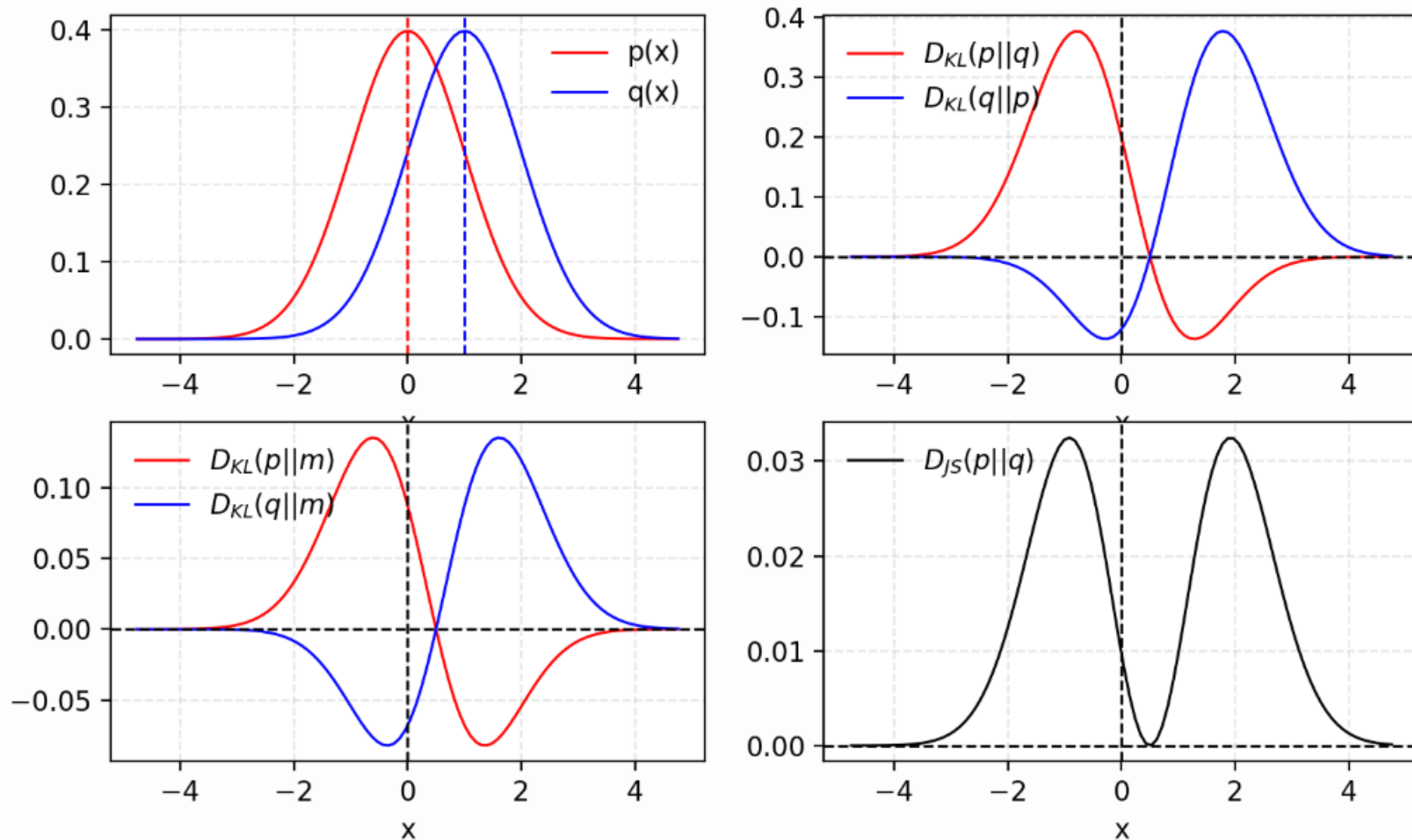


Fig. 1. Given two Gaussian distribution, p with mean=0 and std=1 and q with mean=1 and std=1. The average of two distributions is labelled as $m = (p + q)/2$. KL divergence D_{KL} is asymmetric but JS divergence D_{JS} is symmetric.

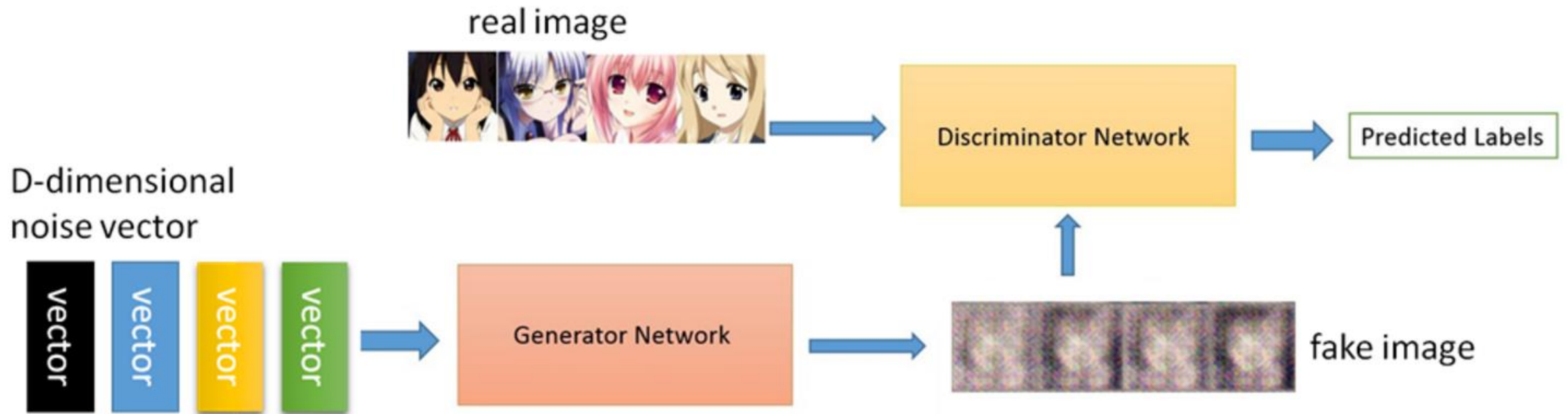
JS Divergence

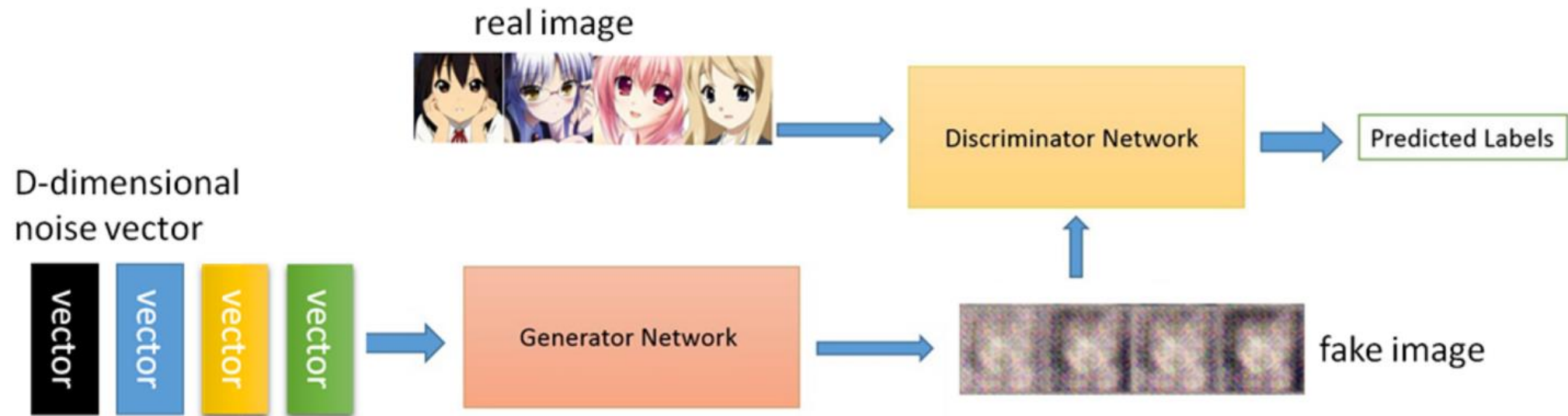
- Also called as the symmetric KL divergence

$$D_{JSD}[p, q] = \frac{1}{2} \left(D_{KL} \left[p, \frac{p+q}{2} \right] + D_{KL} \left[q, \frac{p+q}{2} \right] \right)$$

- Properties
 - $D_{JSD}[p, q] \geq 0$
 - $D_{JSD}[p, q] = 0$ iff $p = q$
 - $D_{JSD}[p, q] = D_{JSD}[q, p]$
 - $\sqrt{D_{JSD}[p, q]}$ satisfies triangle inequality \rightarrow Jensen-Shannon Distance

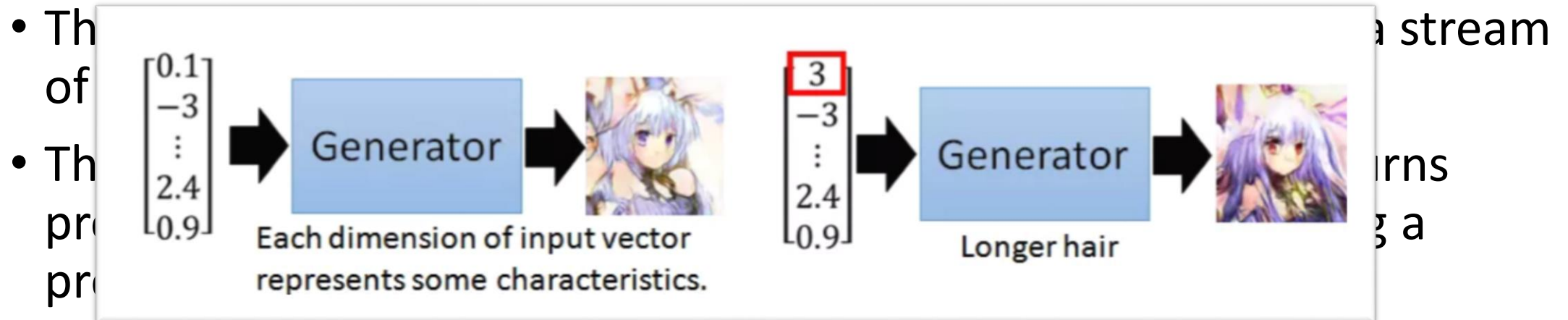
A basic framework for GAN



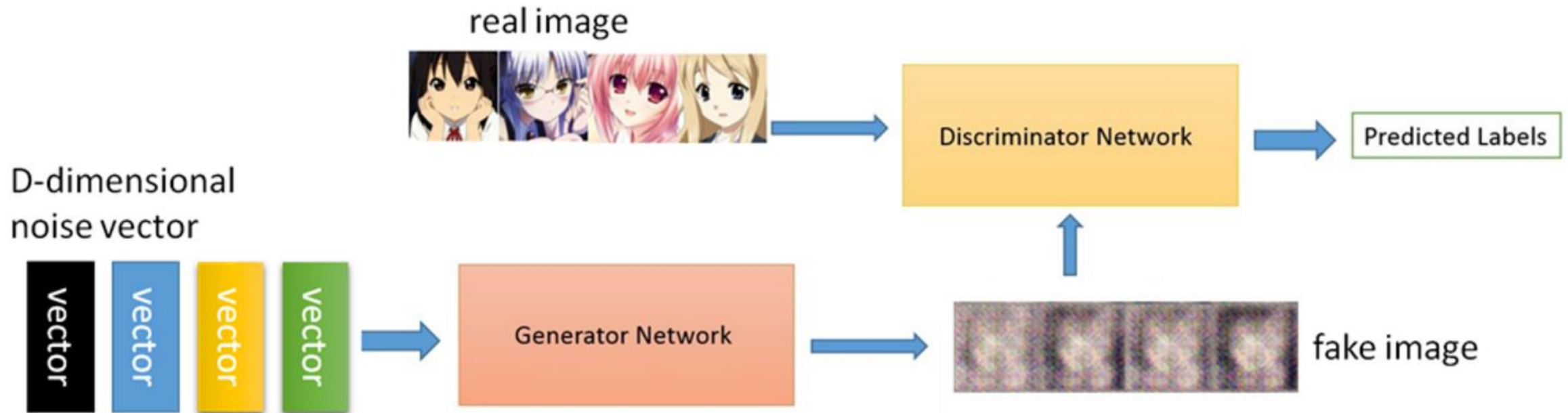


Here are the steps a GAN takes:

- The generator takes in random numbers and returns an image.

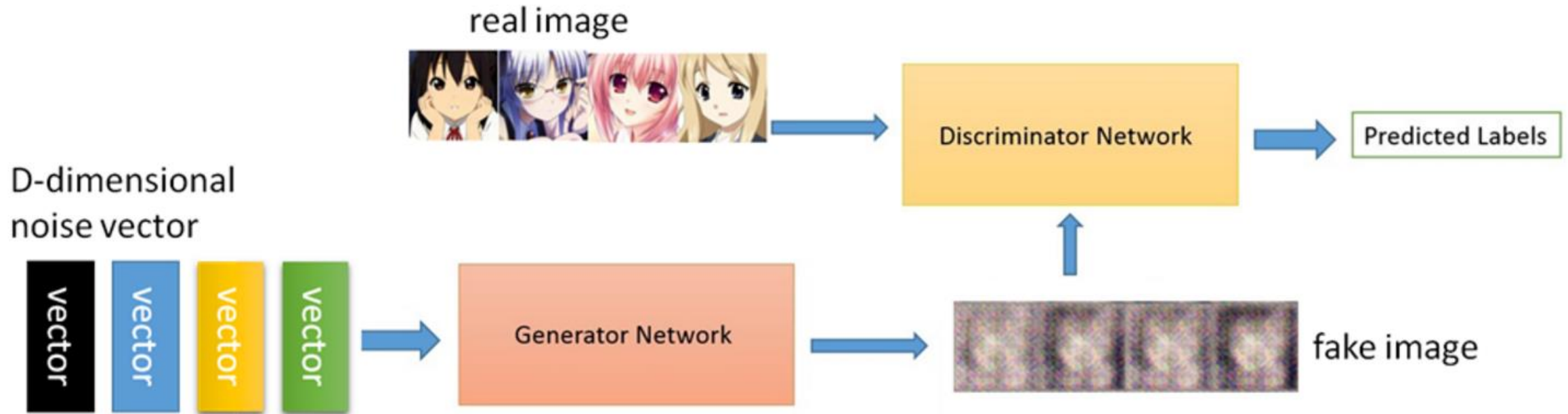


A basic framework for GAN



Both nets are trying to optimize a different and opposing objective function, or loss function, in a zero-sum game.

A basic framework for GAN



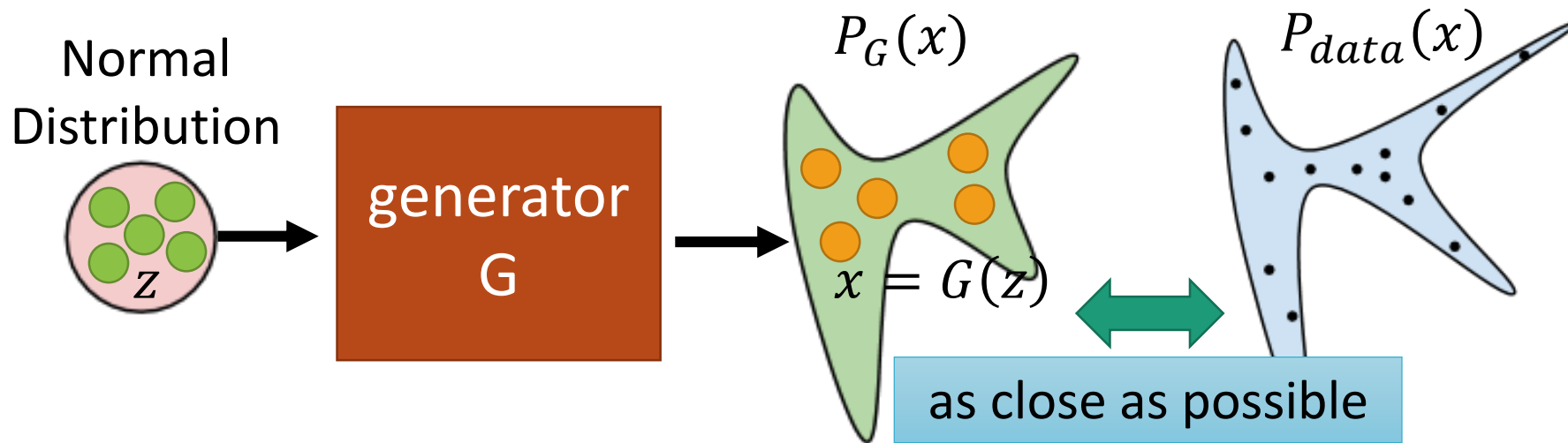
- The game follows with:
 - The generator trying to maximize the probability of making the discriminator mistakes its inputs as real.
 - And the discriminator guiding the generator to produce more realistic images.

Generator

x : an image (a high-dimensional vector)

$$x = G(z)$$

- A generator G is a network. The network defines a probability distribution P_G



$$G^* = \arg \min_G \underline{Div}(P_G, P_{data})$$

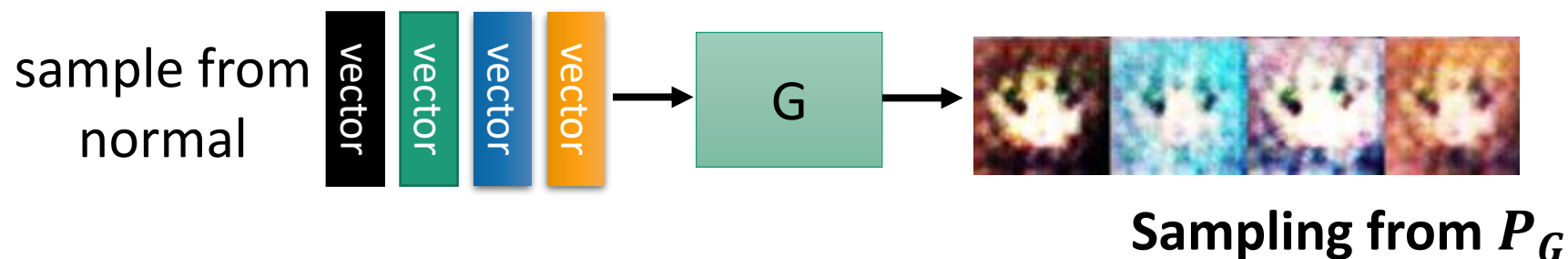
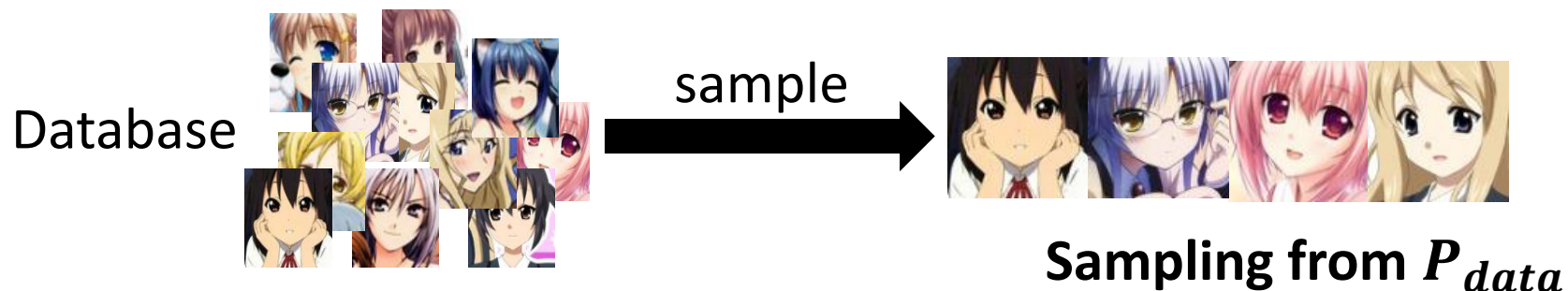
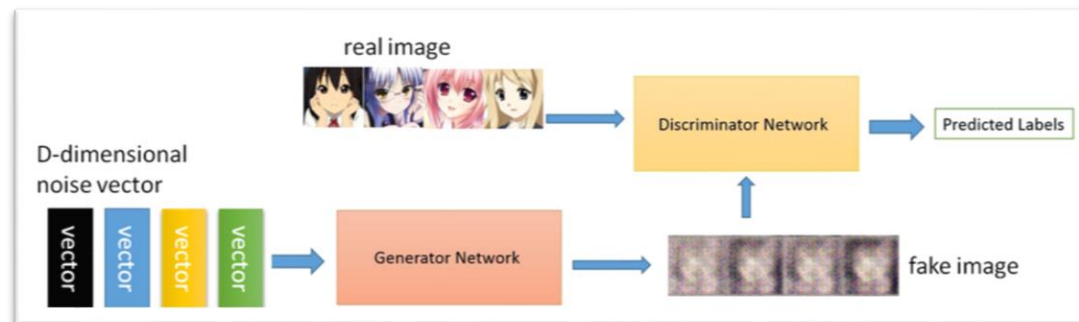
Divergence between distributions P_G and P_{data}

How to compute the divergence?

Discriminator

$$G^* = \arg \min_G \text{Div}(P_G, P_{data})$$

Although we do not know the distributions of P_G and P_{data} , we can sample from them.



Discriminator

$$G^* = \arg \min_G \text{Div}(P_G, P_{data})$$

★ : data sampled from P_{data}

★ : data sampled from P_G

Using the example objective function is exactly the same as training a binary classifier.



Discriminator

Sigmoid Output

Example Objective Function for D

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

(G is fixed)

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

The maximum objective value is related to JS divergence.

Discriminator

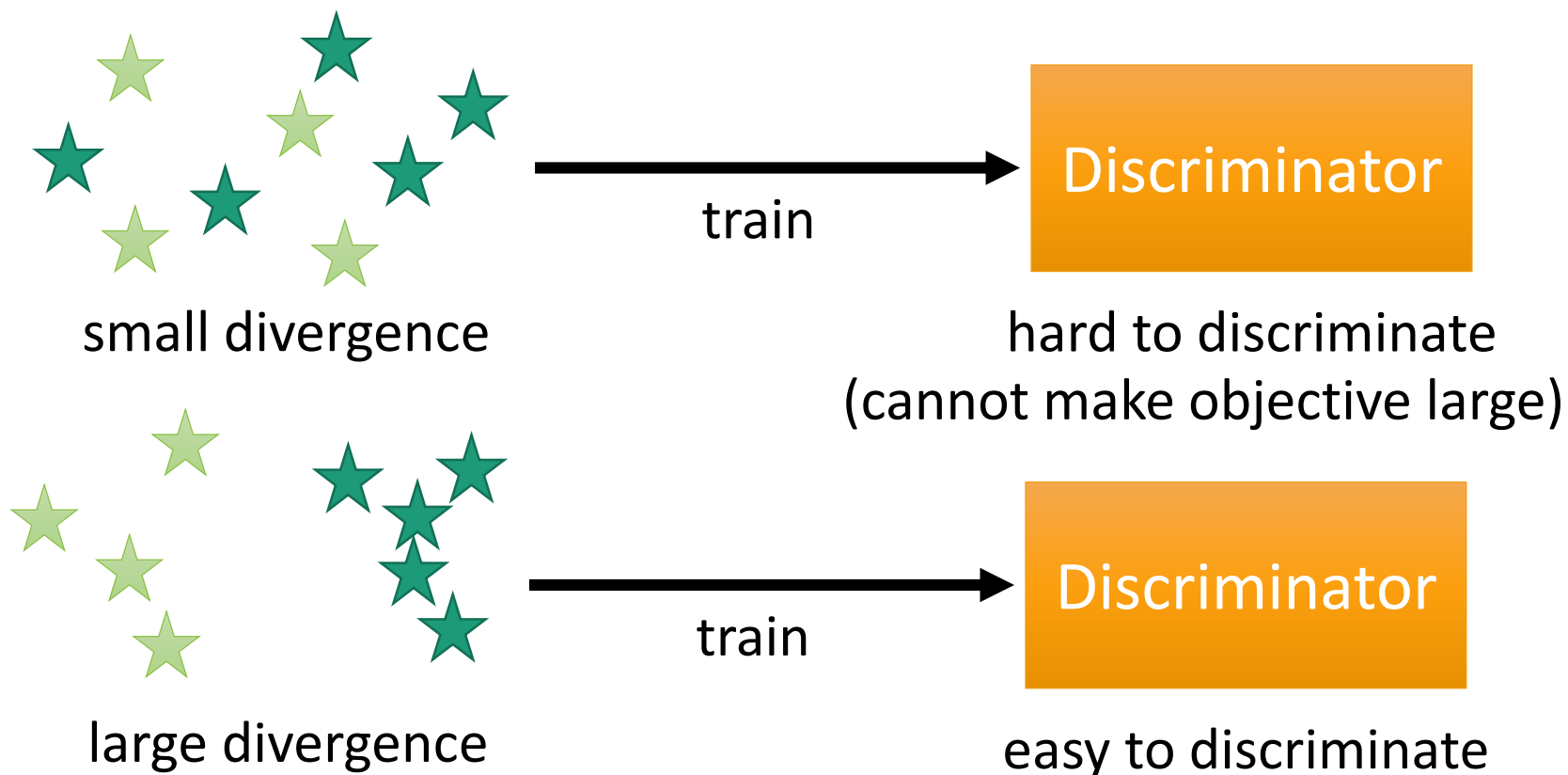
$$G^* = \arg \min_G \text{Div}(P_G, P_{data})$$

★ : data sampled from P_{data}

★ : data sampled from P_G

Training:

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



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

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

- Given G, what is the optimal D^* maximizing

$$\begin{aligned} V &= E_{x \sim P_{data}} [\log D(x)] + E_{x \sim P_G} [\log(1 - D(x))] \\ &= \int_x P_{data}(x) \log D(x) dx + \int_x P_G(x) \log(1 - D(x)) dx \\ &= \int_x [P_{data}(x) \log D(x) + P_G(x) \log(1 - D(x))] dx \end{aligned}$$

Assume that $D(x)$ can be any function

- Given x, the optimal D^* maximizing

$$P_{data}(x) \log D(x) + P_G(x) \log(1 - D(x))$$

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

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

- Given x , the optimal D^* maximizing

$$\underbrace{P_{data}(x)}_a \log \underbrace{D(x)}_D + \underbrace{P_G(x)}_b \log \underbrace{(1 - D(x))}_D$$

- Find D^* maximizing:

$$f(D) = a \log(D) + b \log(1 - D)$$

$$\frac{df(D)}{dD} = a \times \frac{1}{D} + b \times \frac{1}{1 - D} \times (-1) = 0$$

$$a \times \frac{1}{D^*} = b \times \frac{1}{1 - D^*} \quad a \times (1 - D^*) = b \times D^* \\ a - aD^* = bD^* \quad a = (a + b)D^*$$

$$D^* = \frac{a}{a + b}$$



$$\underbrace{0}_{\text{red}} < D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} < \underbrace{1}_{\text{red}}$$

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

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

$$\max_D V(G, D) = V(G, D^*) \quad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

$$= E_{x \sim P_{data}} \left[\log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} \right] \\ + E_{x \sim P_G} \left[\log \frac{P_G(x)}{P_{data}(x) + P_G(x)} \right]$$

$$= \int_x P_{data}(x) \log \frac{\frac{1}{2} P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx \\ + 2 \log \frac{1}{2} - 2 \log 2 + \int_x P_G(x) \log \frac{\frac{1}{2} P_G(x)}{\frac{P_{data}(x) + P_G(x)}{2}} dx$$

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

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

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

$$\max_D V(G, D) = V(G, D^*) \quad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

$$= -2\log 2 + \int_x P_{data}(x) \log \frac{P_{data}(x)}{(P_{data}(x) + P_G(x))/2} dx$$

$$+ \int_x P_G(x) \log \frac{P_G(x)}{(P_{data}(x) + P_G(x))/2} dx$$

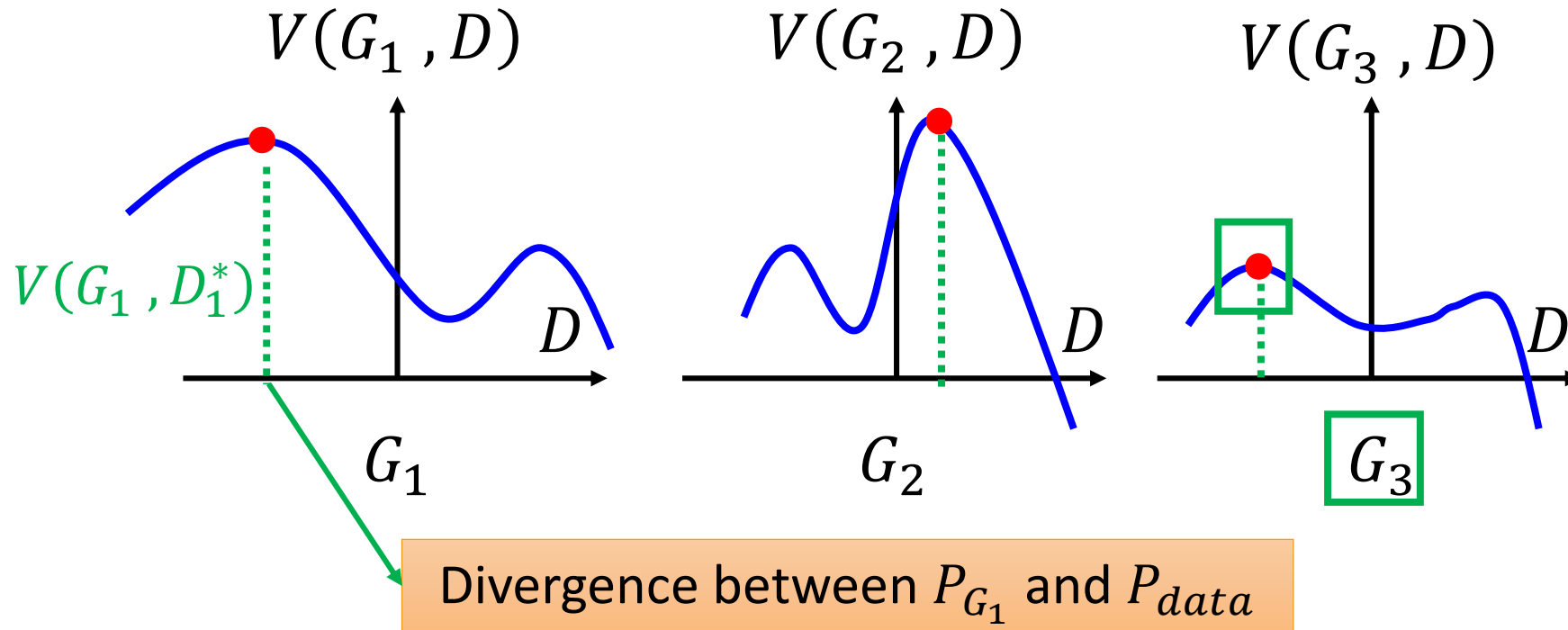
$$= -2\log 2 + \text{KL}\left(P_{data} \parallel \frac{P_{data} + P_G}{2}\right) + \text{KL}\left(P_G \parallel \frac{P_{data} + P_G}{2}\right)$$

$$= -2\log 2 + 2\text{JSD}(P_{data} \parallel P_G) \quad \text{Jensen-Shannon divergence}$$

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

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

The maximum objective value is related to JS divergence.



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

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

The maximum objective value is related to JS divergence.

- Initialize generator and discriminator
- In each training iteration:

Step 1: Fix generator G , and update discriminator D

Step 2: Fix discriminator D , and update generator G

Algorithm

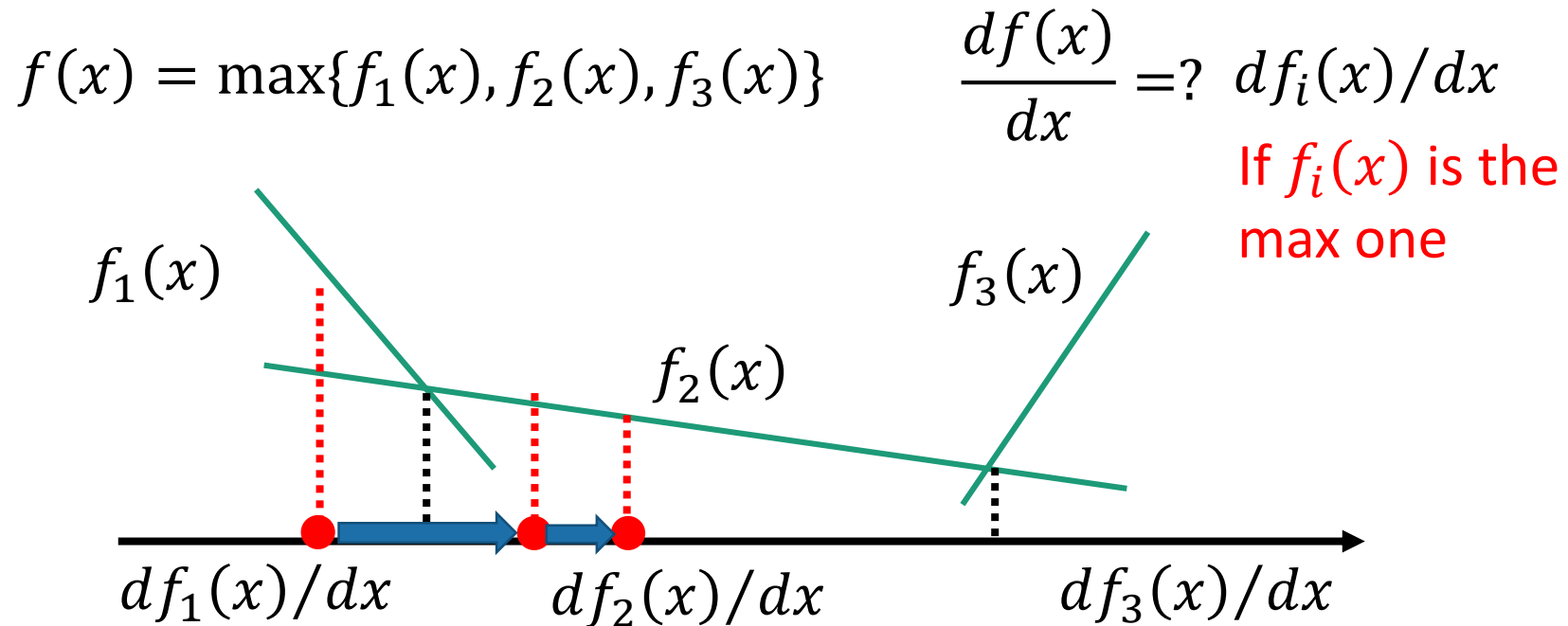
$$G^* = \arg \min_G \max_D V(G, D)$$

$L(G)$

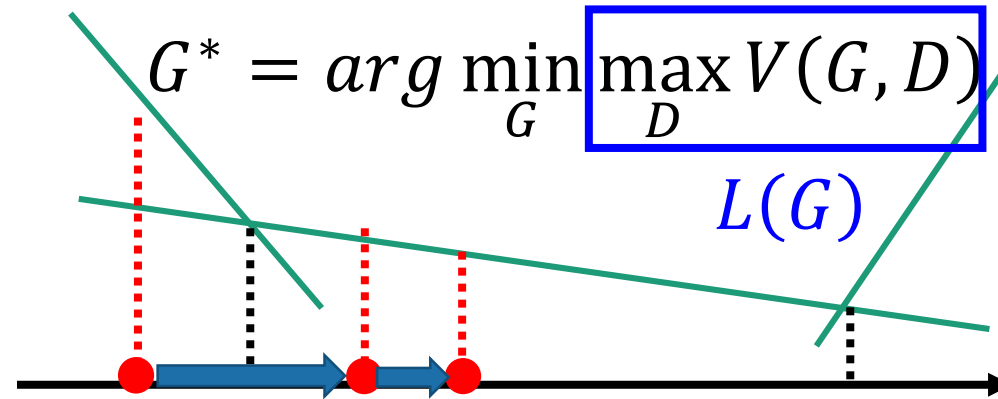
- To find the best G minimizing the loss function $L(G)$,

$$\theta_G \leftarrow \theta_G - \eta \partial L(G) / \partial \theta_G$$

θ_G defines G



Algorithm



- Given G_0

- Find D_0^* maximizing $V(G_0, D)$ **Using Gradient Ascent**

$V(G_0, D_0^*)$ is the JS divergence between $P_{data}(x)$ and $P_{G_0}(x)$

- $\theta_G \leftarrow \theta_G - \eta \partial V(G, D_0^*) / \partial \theta_G \longrightarrow$ Obtain G_1 **Decrease JS divergence(?)**
- Find D_1^* maximizing $V(G_1, D)$

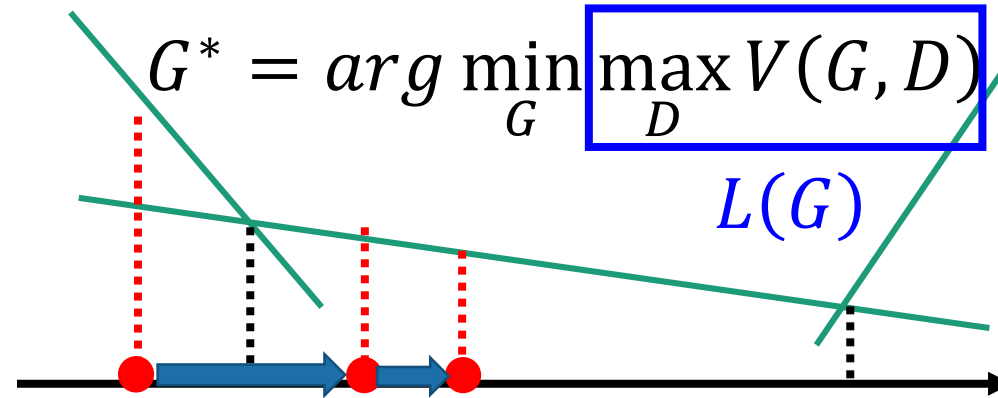
$V(G_1, D_1^*)$ is the JS divergence between $P_{data}(x)$ and $P_{G_1}(x)$

- $\theta_G \leftarrow \theta_G - \eta \partial V(G, D_1^*) / \partial \theta_G \longrightarrow$ Obtain G_2 **Decrease JS divergence(?)**
-

Algorithm

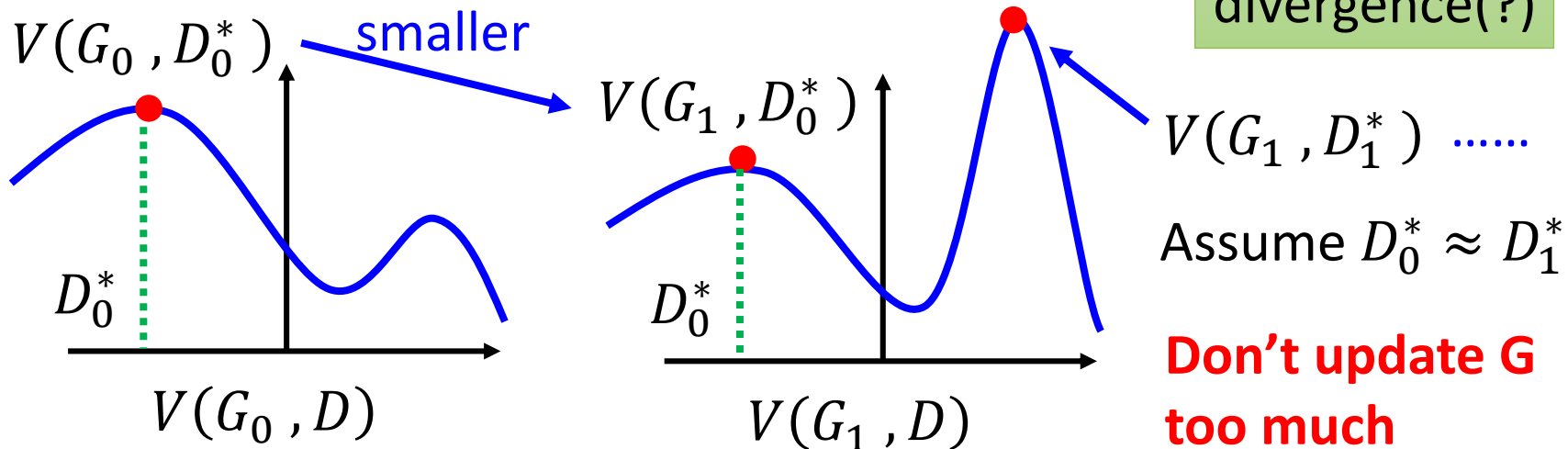
- Given G_0
- Find D_0^* maximizing $V(G_0, D)$

$V(G_0, D_0^*)$ is the JS divergence between $P_{data}(x)$ and $P_{G_0}(x)$



- $\theta_G \leftarrow \theta_G - \eta \partial V(G, D_0^*) / \partial \theta_G \longrightarrow$ Obtain G_1

Decrease JS divergence(?)



In practice ...

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


- Given G , how to compute $\max_D V(G, D)$
 - Sample $\{x^1, x^2, \dots, x^m\}$ from $P_{data}(x)$, sample $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$ from generator $P_G(x)$

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

II

Binary Classifier

D is a binary classifier with sigmoid output (can be deep)

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

$\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$ from $P_G(x)$  Negative examples

Minimize Cross-entropy

Algorithm Initialize θ_d for D and θ_g for G

Can only find
lower bound of $\max_D V(G, D)$

- In each training iteration:

Learning
D

Repeat
k times

- Sample m examples $\{x^1, x^2, \dots, x^m\}$ from data distribution $P_{data}(x)$
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from the prior $P_{prior}(z)$
- Obtaining generated data $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to maximize
 - $\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D(x^i) + \frac{1}{m} \sum_{i=1}^m \log (1 - D(\tilde{x}^i))$
 - $\theta_d \leftarrow \theta_d + \eta \nabla \tilde{V}(\theta_d)$

Learning
G

Only
Once

- Sample another m noise samples $\{z^1, z^2, \dots, z^m\}$ from the prior $P_{prior}(z)$
- Update generator parameters θ_g to minimize
 - $\tilde{V} = \frac{1}{m} \sum_{i=1}^m \log D(x^i) + \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^i)))$
 - $\theta_g \leftarrow \theta_g - \eta \nabla \tilde{V}(\theta_g)$

Objective Function for Generator in Real Implementation

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

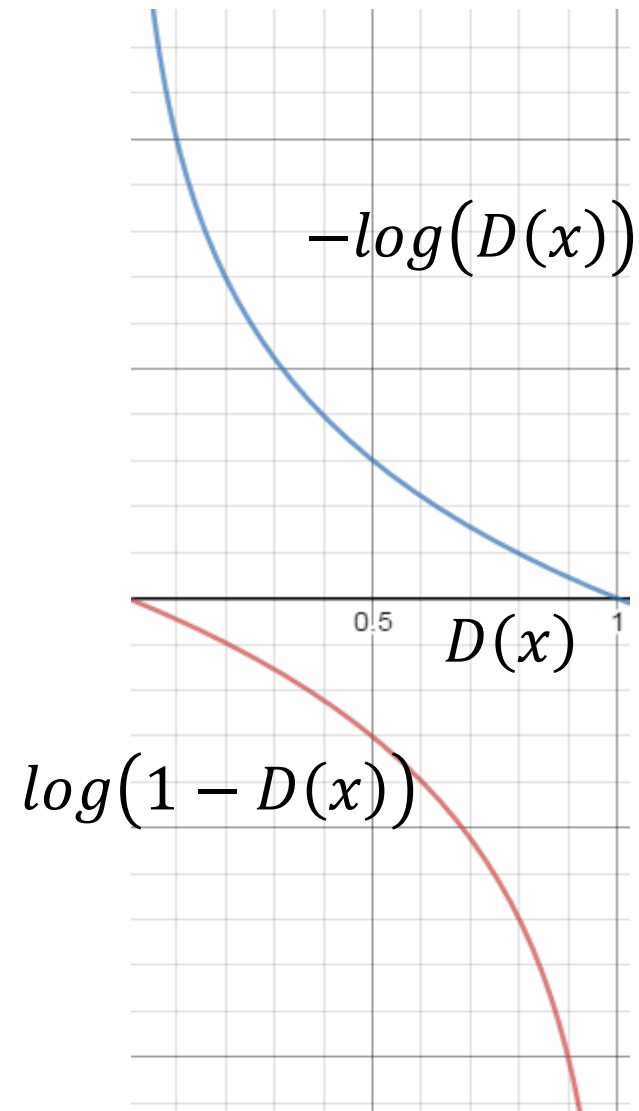
Slow at the beginning

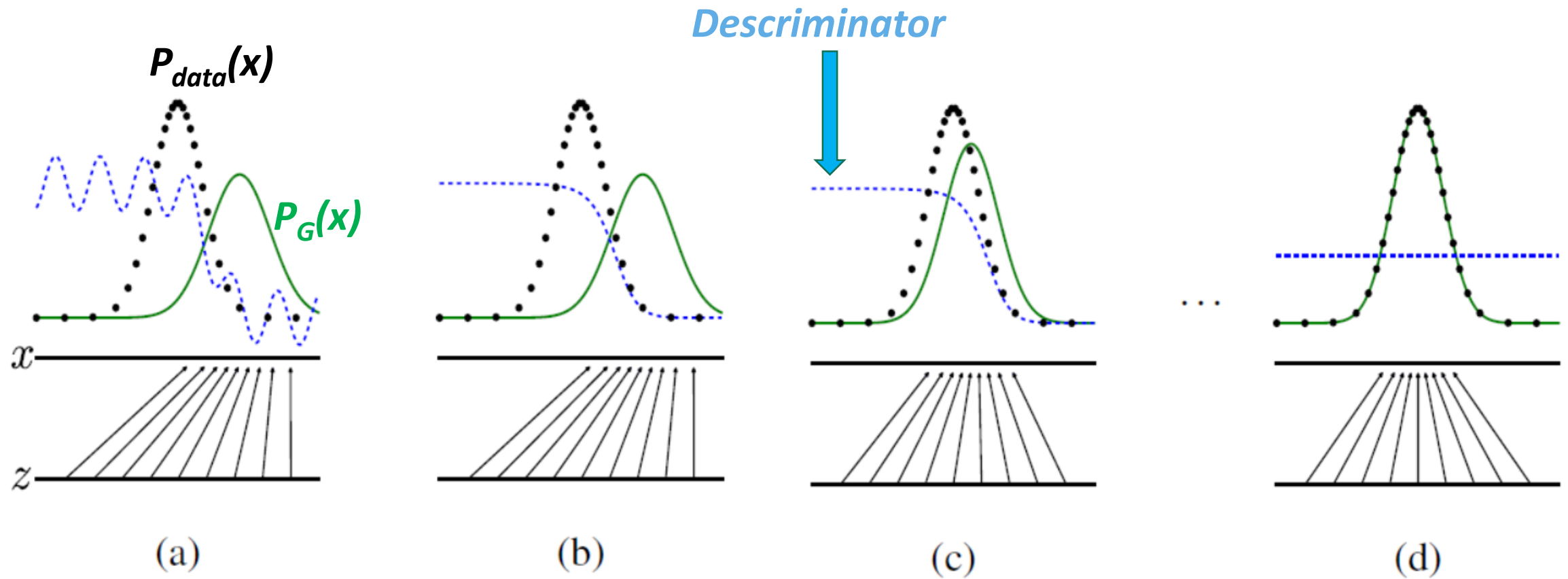
Minimax GAN (MMGAN)

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

Real implementation:
label x from P_G as positive

Non-saturating GAN (NSGAN)





- In the perfect equilibrium, the generator would capture the general training data distribution. As a result, the discriminator would be always unsure of whether its inputs are real or not.

Advantages of GANs

- Markov chain is not needed.
- In theory, as long as differentiable functions can be used to construct D and G , it can be **combined with deep learning** to learn deep production networks.
- From a statistical point of view, the parameter updating of G does not come directly from the data sample, but from the backpropagation gradient of D .
- Various loss functions can be used in the GAN model.

Disadvantages of GANs

Hard to achieve Nash equilibrium

Vanishing gradient

Mode collapse

During the training, the generator may collapse to a setting where it always produces same outputs.



Disadvantages of GANs

Lack of a proper evaluation metric

No good sign to tell when to stop; No good indicator to compare the performance of multiple models.

The distributed $P_G(x)$ of the generator is not represented.

It's difficult to train. D and G need good synchronization, such as D update K times and G update 1 time.

Tips and tricks to make GANs work

1. Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output

2: A modified loss function

In GAN papers, the loss function to optimize G is $\min (\log 1-D)$, but in practice folks practically use $\max \log D$

- because the first formulation has vanishing gradients early on
- Goodfellow et. al (2014)

In practice, works well:

- Flip labels when training generator: real = fake, fake = real

Tips and tricks to make GANs work

3: Use a spherical Z

- Don't sample from a Uniform distribution
- Sample from a gaussian distribution

4: BatchNorm

- Construct different mini-batches for real and fake, i.e. each mini-batch needs to contain only all real images or all generated images.
- when batchnorm is not an option use instance normalization (for each sample, subtract mean and divide by standard deviation).

Tips and tricks to make GANs work

5: Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers if you have sparse gradients
- LeakyReLU = good (in both G and D)
- For Downsampling, use: Average Pooling, Conv2d + stride
- For Upsampling, use: PixelShuffle, ConvTranspose2d + stride
 - PixelShuffle: <https://arxiv.org/abs/1609.05158>

6: Use Soft and Noisy Labels

- Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3 (for example).
 - Salimans et. al. 2016
- make the labels the noisy for the discriminator: occasionally flip the labels when training the discriminator

Tips and tricks to make GANs work

7: DCGAN / Hybrid Models

- Use DCGAN when you can. It works!
- if you cant use DCGANs and no model is stable, use a hybrid model : KL + GAN or VAE + GAN

8: Use stability tricks from RL

- Experience Replay
 - Keep a replay buffer of past generations and occassionally show them
 - Keep checkpoints from the past of G and D and occassionally swap them out for a few iterations
- All stability tricks that work for deep deterministic policy gradients
- See Pfau & Vinyals (2016)

Tips and tricks to make GANs work

9: Use the ADAM Optimizer

- `optim.Adam` rules!
 - See Radford et. al. 2015
- Use SGD for discriminator and ADAM for generator

11: Dont balance loss via statistics

12: If you have labels, use them

13: Add noise to inputs, decay over time

14: [notsure] Train discriminator more (sometimes)

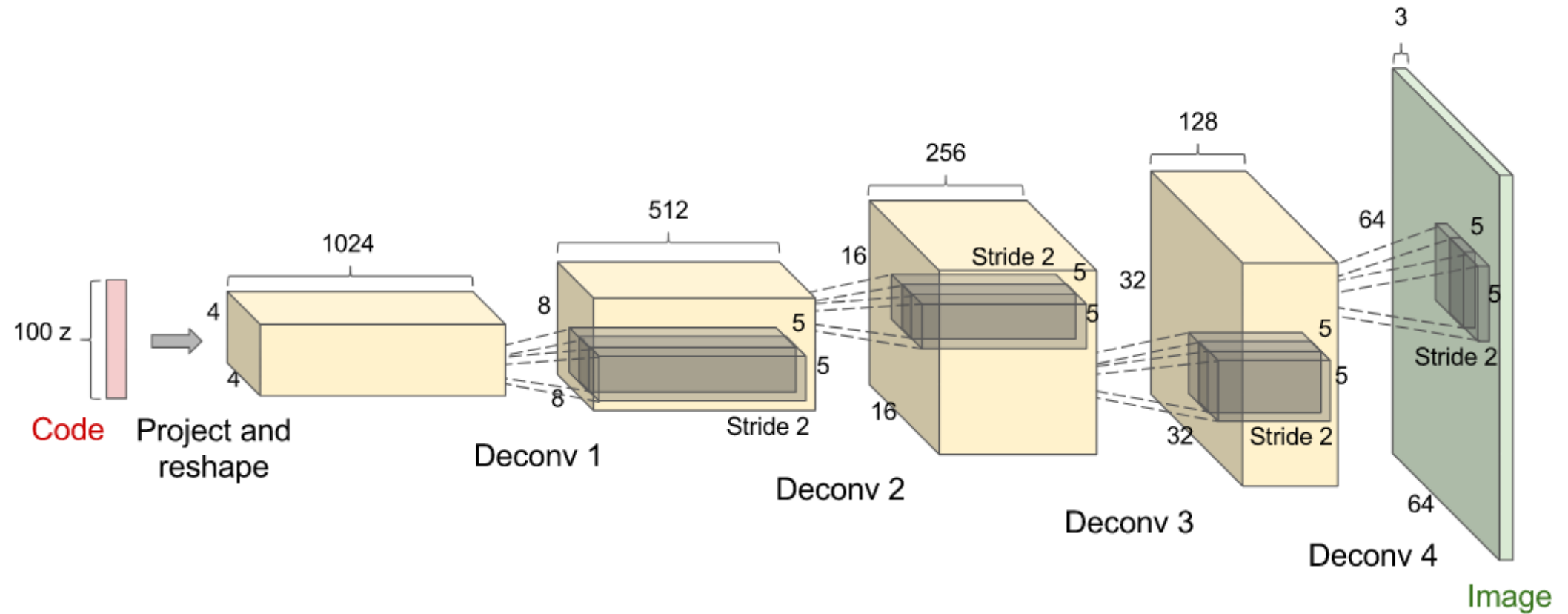
Tips and tricks to make GANs work

15: [notsure] Batch Discrimination

16: Discrete variables in Conditional GANs

17: Use Dropouts in G in both train and test phase

DCGAN



Radford A , Metz L , Chintala S . Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks[J]. Computer Science, 2015.

Wasserstein GAN (WGAN)

- Even when two distributions are located in lower dimensional manifolds without overlaps, **Wasserstein distance** can still provide a meaningful and smooth representation of the distance in-between.

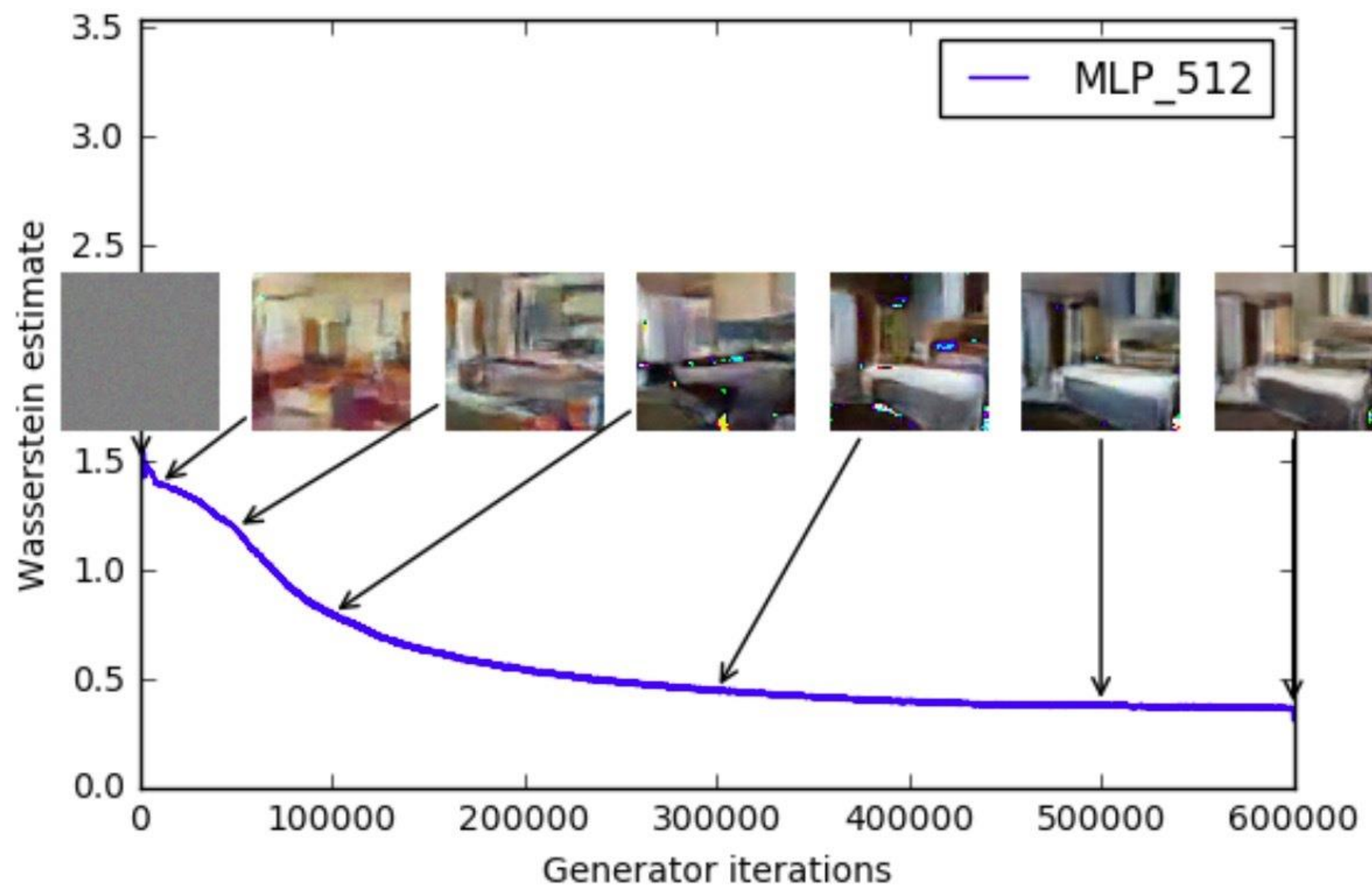
D_{KL} gives us infinity when two distributions are disjoint. The value of D_{JS} has sudden jump, not differentiable at $\theta = 0$. Only Wasserstein metric provides a smooth measure, which is super helpful for a stable learning process using gradient descents.

Wasserstein GAN (WGAN)

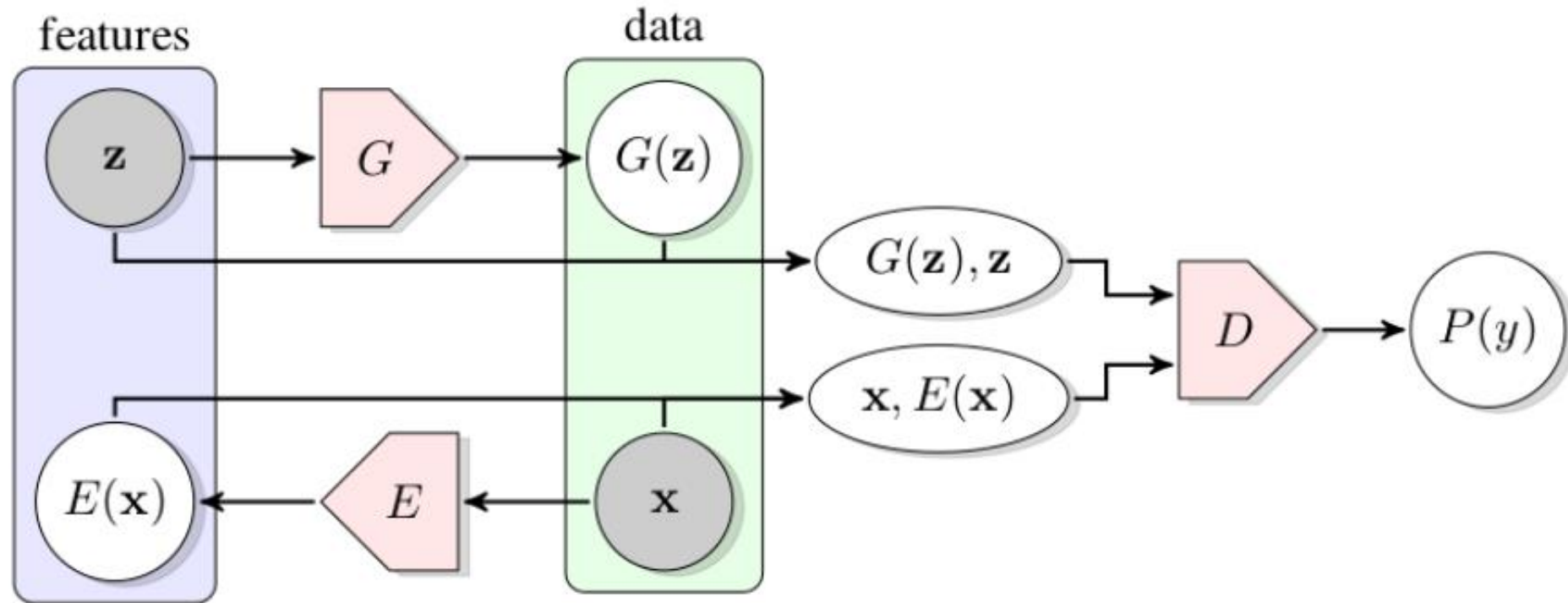
- The last layer of the discriminator removes sigmoid
- Loss of generator and discriminator does not take log
- After updating the discriminator parameters each time, truncate their absolute values to no more than a fixed constant C
- Instead of using momentum-based optimization algorithms (including momentum and Adam), recommend RMSProp and SGD.

Advantages of WGAN

- Greatly improve the problem of GAN training instability, no longer need to carefully balance the training level of generator and discriminator.
- The collapse mode problem is basically solved and the diversity of generated samples is ensured.
- At last, there is a value such as cross-entropy and accuracy to indicate the training process. The smaller the value, the better the GAN is trained and the higher the image quality produced by the generator is represented (as shown in the title chart).
- All the above benefits can be achieved without elaborate network architecture. The simplest multi-layer fully connected network can be achieved.



BiGAN



<https://arxiv.org/pdf/1605.09782.pdf>

DCGAN

- Here is the summary of DCGAN:
 - Replace all max pooling with convolutional stride
 - Use transposed convolution for upsampling.
 - Eliminate fully connected layers.
 - Use Batch normalization except the output layer for the generator and the input layer of the discriminator.
 - Use ReLU in the generator except for the output which uses tanh.
 - Use LeakyReLU in the discriminator.

Applications of GANs

- **Generating high-quality images**

- [Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks] [\[Paper\]](#)[\[Code\]](#)(Gan with convolutional networks)(ICLR)
- [Generative Adversarial Text to Image Synthesis] [\[Paper\]](#)[\[Code\]](#)[\[Code\]](#)
- [Improved Techniques for Training GANs] [\[Paper\]](#)[\[Code\]](#)(Goodfellow's paper)
- [Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space] [\[Paper\]](#)[\[Code\]](#)
- [StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks] [\[Paper\]](#)[\[Code\]](#)
- [Improved Training of Wasserstein GANs] [\[Paper\]](#)[\[Code\]](#)
- [Boundary Equilibrium Generative Adversarial Networks Implementation in Tensorflow] [\[Paper\]](#)[\[Code\]](#)
- [Progressive Growing of GANs for Improved Quality, Stability, and Variation] [\[Paper\]](#)[\[Code\]](#)

Applications of GANs

- Semi-supervised learning
 - [Adversarial Training Methods for Semi-Supervised Text Classification] [\[Paper\]](#) [\[Note\]](#)(Ian Goodfellow Paper)
 - [Improved Techniques for Training GANs] [\[Paper\]](#)[\[Code\]](#)(Goodfellow's paper)
 - [Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks] [\[Paper\]](#)(ICLR)
 - [Semi-Supervised QA with Generative Domain-Adaptive Nets] [\[Paper\]](#)(ACL 2017)

Applications of GANs

Ensembles

- [AdaGAN: Boosting Generative Models] [\[Paper\]](#)[\[\[Code\]\]](#) (Google Brain)

Clustering

- [Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks] [\[Paper\]](#)(ICLR)

Image blending

- [GP-GAN: Towards Realistic High-Resolution Image Blending] [\[Paper\]](#)[\[Code\]](#)

Applications of GANs

Image Inpainting

- [Semantic Image Inpainting with Perceptual and Contextual Losses] [\[Paper\]](#) [\[Code\]](#)(CVPR 2017)
- [Context Encoders: Feature Learning by Inpainting] [\[Paper\]](#)[\[Code\]](#)
- [Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks] [\[Paper\]](#)
- [Generative face completion] [\[Paper\]](#)[\[Code\]](#)(CVPR2017)
- [Globally and Locally Consistent Image Completion] [\[MainPAGE\]](#)(SIGGRAPH 2017)

Applications of GANs

Semantic Segmentation

- [Adversarial Deep Structural Networks for Mammographic Mass Segmentation] [\[Paper\]](#)[\[Code\]](#)
- [Semantic Segmentation using Adversarial Networks] [\[Paper\]](#) (Soumith's paper)

Object Detection

- [Perceptual generative adversarial networks for small object detection] [\[Paper\]](#) (CVPR 2017)
- [A-Fast-RCNN: Hard Positive Generation via Adversary for Object Detection] [\[Paper\]](#)[\[Code\]](#)(CVPR2017)

Applications of GANs

Conditional Adversarial Nets

- [Conditional Generative Adversarial Nets] [\[Paper\]](#)[\[Code\]](#)
- [InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets] [\[Paper\]](#)[\[Code\]](#)[\[Code\]](#)
- [Conditional Image Synthesis With Auxiliary Classifier GANs] [\[Paper\]](#)[\[Code\]](#)
(GoogleBrain ICLR 2017)
- [Pixel-Level Domain Transfer] [\[Paper\]](#)[\[Code\]](#)
- [Invertible Conditional GANs for image editing] [\[Paper\]](#)[\[Code\]](#)
- [Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space] [\[Paper\]](#)[\[Code\]](#)

Applications of GANs

Video Prediction & Generation

- [Deep multi-scale video prediction beyond mean square error] [\[Paper\]](#)[\[Code\]](#)
(Yann LeCun's paper)
- [Generating Videos with Scene Dynamics] [\[Paper\]](#)[\[Web\]](#)[\[Code\]](#)
- [MoCoGAN: Decomposing Motion and Content for Video Generation] [\[Paper\]](#)

Texture Synthesis & Style Transfer

- [Precomputed real-time texture synthesis with markovian generative adversarial networks] [\[Paper\]](#)[\[Code\]](#)(ECCV 2016)

Image Translation

- [Unsupervised cross-domain image generation] [\[Paper\]](#)[\[Code\]](#)
- [Image-to-image translation using conditional adversarial nets] [\[Paper\]](#)[\[Code\]](#)
[\[Code\]](#)
- [Learning to Discover Cross-Domain Relations with Generative Adversarial Networks] [\[Paper\]](#)[\[Code\]](#)
- [Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks] [\[Paper\]](#)[\[Code\]](#)
- [CoGAN: Coupled Generative Adversarial Networks] [\[Paper\]](#)[\[Code\]](#)(NIPS 2016)
- [Unsupervised Image-to-Image Translation with Generative Adversarial Networks] [\[Paper\]](#)
- [Unsupervised Image-to-Image Translation Networks] [\[Paper\]](#)
- [Triangle Generative Adversarial Networks] [\[Paper\]](#)

GAN Theory

- [Energy-based generative adversarial network] [\[Paper\]](#)[\[Code\]](#)(Lecun paper)
- [Improved Techniques for Training GANs] [\[Paper\]](#)[\[Code\]](#)(Goodfellow's paper)
- [Mode Regularized Generative Adversarial Networks] [\[Paper\]](#)(Yoshua Bengio , ICLR 2017)
- [Improving Generative Adversarial Networks with Denoising Feature Matching] [\[Paper\]](#)[\[Code\]](#)(Yoshua Bengio , ICLR 2017)
- [Sampling Generative Networks] [\[Paper\]](#)[\[Code\]](#)
- [How to train Gans] [\[Docu\]](#)
- [Towards Principled Methods for Training Generative Adversarial Networks] [\[Paper\]](#)(ICLR 2017)
- [Unrolled Generative Adversarial Networks] [\[Paper\]](#)[\[Code\]](#)(ICLR 2017)
- [Least Squares Generative Adversarial Networks] [\[Paper\]](#)[\[Code\]](#)(ICCV 2017)
- [Wasserstein GAN] [\[Paper\]](#)[\[Code\]](#)
- [Improved Training of Wasserstein GANs] [\[Paper\]](#)[\[Code\]](#)(The improve of wgan)
- [Towards Principled Methods for Training Generative Adversarial Networks] [\[Paper\]](#)
- [Generalization and Equilibrium in Generative Adversarial Nets] [\[Paper\]](#) (ICML 2017)

Thanks & questions?