Online Joint GlueX-EIC-PANDA Machine Learning Workshop

Generative Adversarial Networks

P. Jiang

Sep. 21-25, 2020

GSI Helmholtzzentrum für Schwerionenforschung GmbH

Institute of Modern Physics, Chinese Academy of Sciences

Many thanks to K. Goetzen, R. Kliemt, F. Nerling and K. Peters

Supported by China and Germany Postdoctoral Exchange Program

Outline

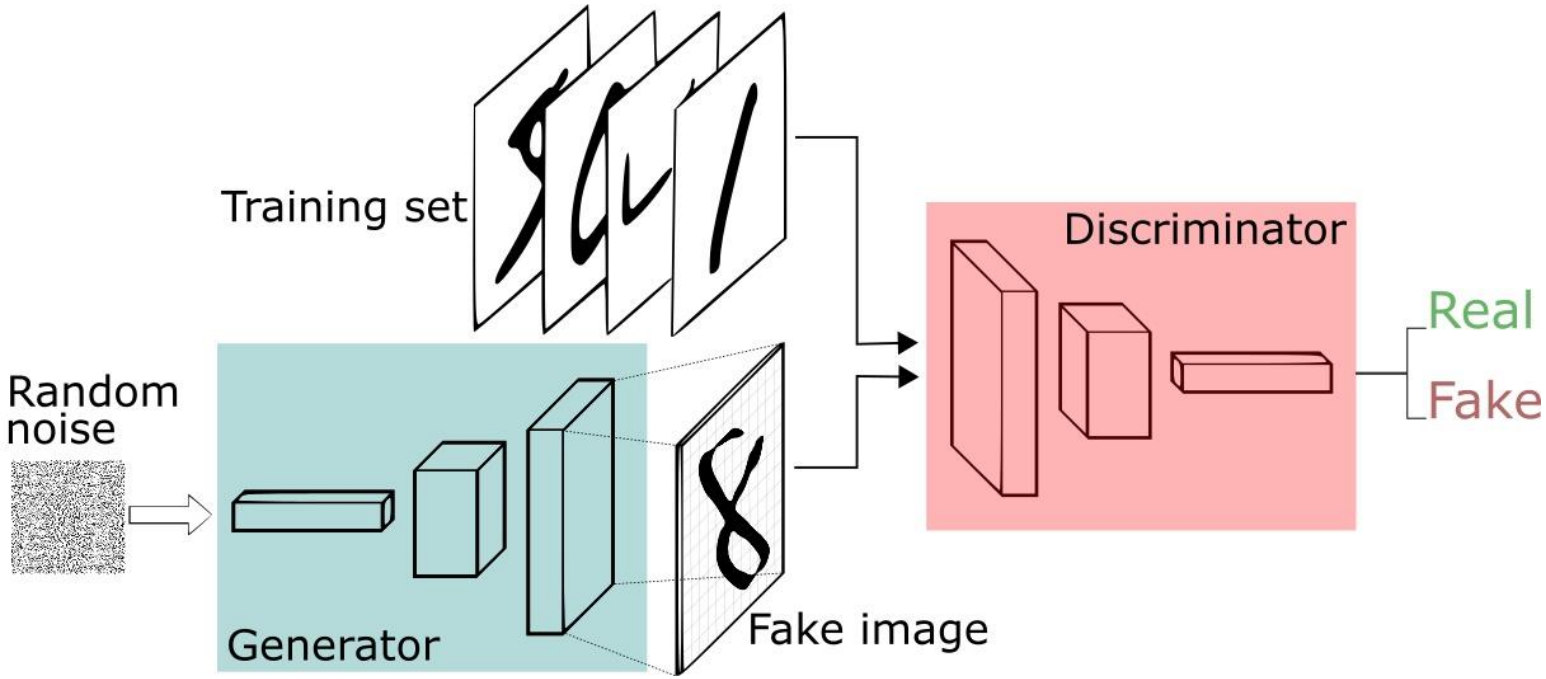
- Introduction —— What is Generative Adversarial Network (GAN)?
- Prerequisites —— What is the prior knowledge for GANs?
- Basic idea —— How does a GAN work?
- Algorithm —— How to train a GAN?
- Theory —— What is the mathematics behind a GAN?
- WGAN —— Is there a more stable GAN?
- GANs —— Are there other GANs?
- Evaluation —— How good is a GAN?
- Tricks —— What are the tricks to make a GAN work?
- Practice —— How to code the first line?

Introduction

—— What is Generative Adversarial Network (GAN)?

Introduction

- The two neural networks, the generator and the discriminator, cooperate in an adversarial way to achieve the purpose of joint training.
- The generator generates fake samples while the discriminator discriminates the real samples and the fake samples.
- The generator is trained to “fool” the discriminator by generating the realistic looking fake samples.
- The discriminator is trained to discriminate the real samples and the fake samples.



arXiv:1406.2661

Applications

- Image processing and computer vision
  - Super-resolution
  - Image synthesis and manipulation
  - Texture synthesis
  - Object detection
  - Video applications
  - Other image and vision applications
- Sequential data
  - Natural language processing
  - Music
  - Speech and Audio
- Other applications
  - Medical field
  - Data science

arXiv:1802.07228

arXiv:2001.06937

arXiv 1703.10593

Prerequisites

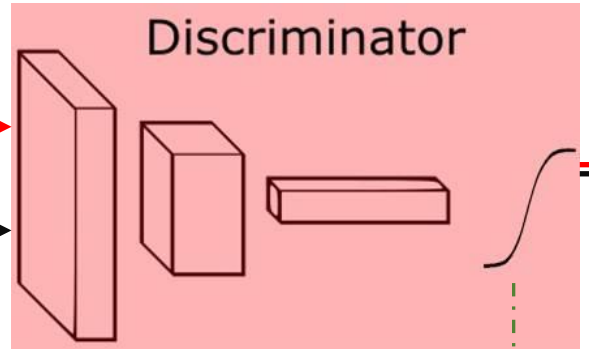
—— What is the prior knowledge for GANs?

Binary classification

Real samples from labeled dataset

label=1

BCE loss=$-\sum (y \log(\hat{y}(\theta))+(1-y) \log(1-\hat{y}(\theta)))$

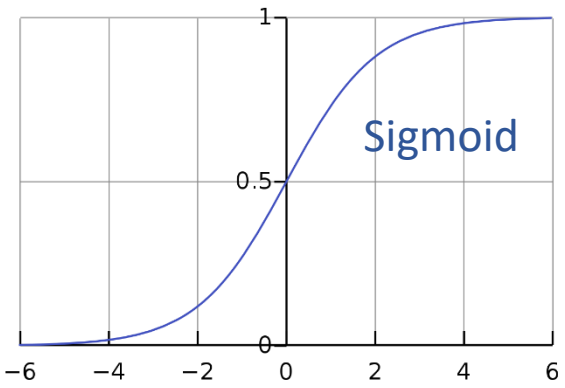


Real

Fake

Fake samples from labeled dataset

label=0



Binary cross entropy

- Given a sample, it can only be classified as 0 or 1. It follows the Bernoulli distribution.
- If the probability of the sample being classified as 1 is $\hat{y}(\theta)$， the probability of being classified as 0 is 1- $\hat{y}(\theta)$， where θ are neural network parameters.
- The probability mass function is $f(y;\hat{y}(\theta))=\begin{cases}\hat{y}(\theta) & \text{if } y=1,\\ 1-\hat{y}(\theta) & \text{if } y=0.\end{cases} = \hat{y}(\theta)^{y}\,(1-\hat{y}(\theta))^{(1-y)} \qquad y \in \{0,1\}$
- When we have multiple samples, we can construct the likelihood function $L(\theta) = \prod \hat{y}(\theta)^{y}\,(1-\hat{y}(\theta))^{(1-y)}$
- By maximizing $L(\theta)$, update $\theta \leftarrow \underset{\theta}{\operatorname{argmax}}\, L(\theta)$
- Maximizing likelihood function is equivalent to minimizing binary cross entropy

$$\underset{\theta}{\operatorname{argmax}}\, L(\theta) = \underset{\theta}{\operatorname{argmax}} \prod \hat{y}(\theta)^{y}\,(1-\hat{y}(\theta))^{(1-y)} = \underset{\theta}{\operatorname{argmax}} \log\left(\prod \hat{y}(\theta)^{y}\,(1-\hat{y}(\theta))^{(1-y)}\right)$$

$$= \underset{\theta}{\operatorname{argmax}} \sum (y\log(\hat{y}(\theta))+(1-y)\log(1-\hat{y}(\theta))) = \underset{\theta}{\operatorname{argmin}}\left[-\sum (y\log(\hat{y}(\theta))+(1-y)\log(1-\hat{y}(\theta)))\right]$$

$$= \underset{\theta}{\operatorname{argmin}}\ \mathrm{BCE}(y;\hat{y}(\theta))$$

- **BCE loss=$-\sum (y\log(\hat{y}(\theta))+(1-y)\log(1-\hat{y}(\theta)))$**

Basic idea

—— How does a GAN work?

From binary classification to GAN

Real samples from labeled dataset

label=1

Fake samples ~~from labeled dataset~~ from generator

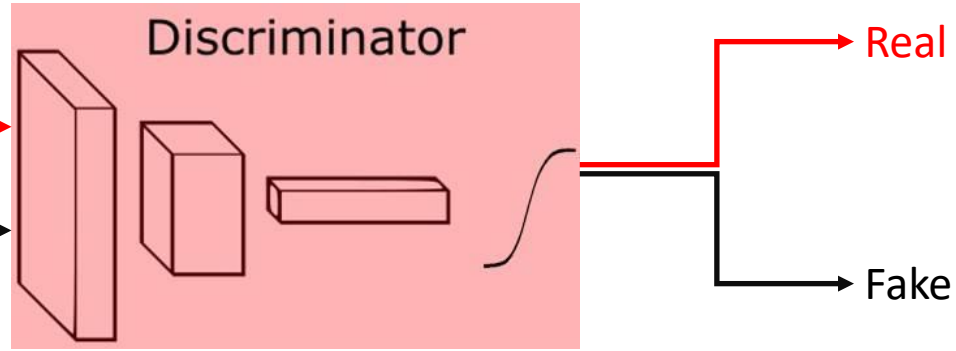
label=0

$$\text{BCE loss} = -\sum (y \log(\hat{y}(\theta)) + (1-y) \log(1-\hat{y}(\theta)))$$

$$\text{BCE loss} = -\sum (y_{real} \log(\hat{y}_{real}) + (1-y_{fake}) \log(1-\hat{y}_{fake}))$$

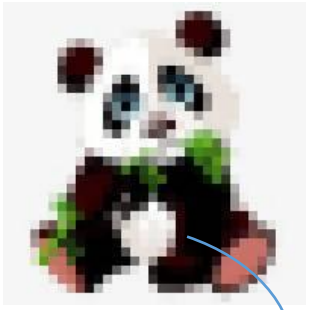
$$\text{BCE loss} = -\sum (\log(\hat{y}_{real}) + \log(1-\hat{y}_{fake}))$$

$$\text{BCE loss} = -\sum (\log(D(x_{real})) + \log(1-D(x_{fake})))$$



Encoder-Decoder

$$\begin{bmatrix} mouth = & 0.49 \\ ear & = -0.67 \\ & \vdots \\ eyes & = 0.44 \\ nose & = 1.80 \end{bmatrix}$$



My daddy has a big mouth…

About 0.49

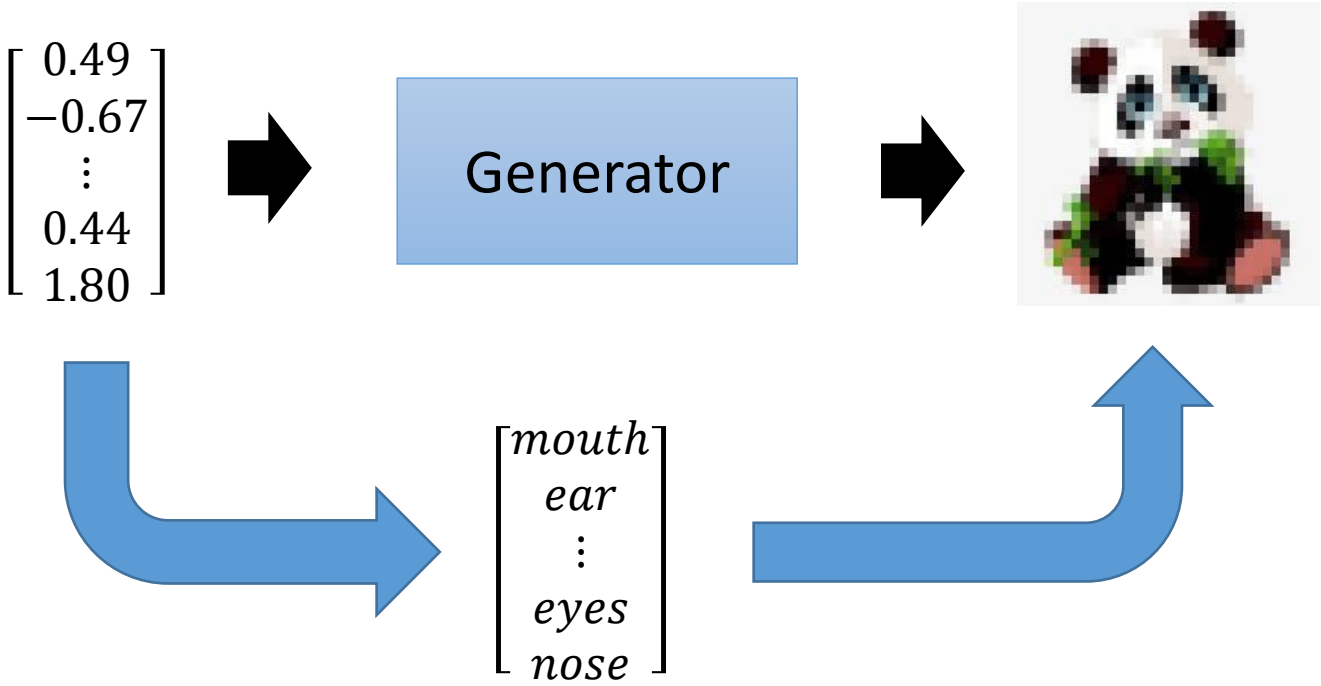
Encoder

How big?

Wow! Huge…

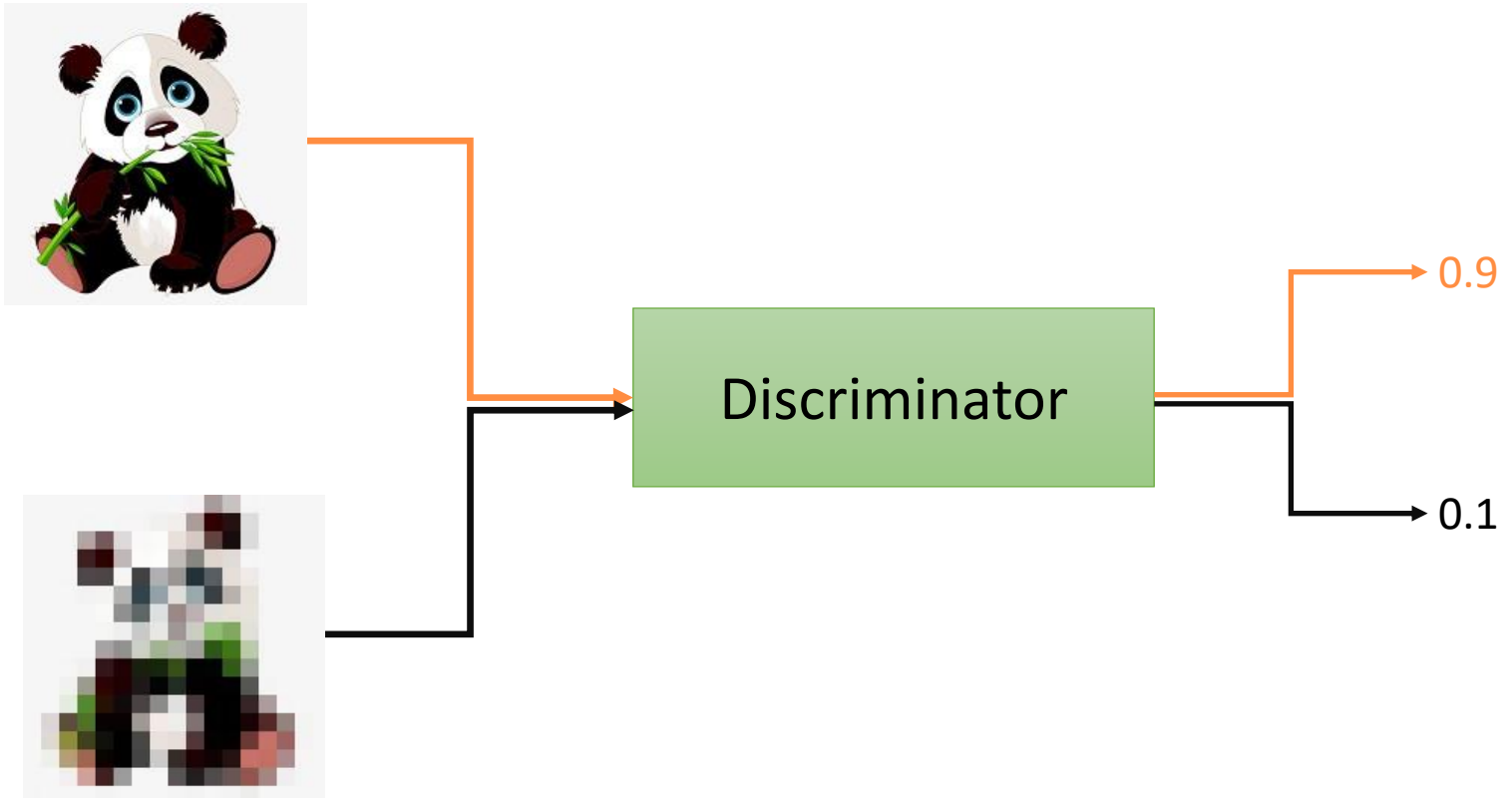
Decoder

Generator



Each dimension of input vector represents some characteristics.

Discriminator



Contest

Real money

Discriminator

Generator

Our slogan is to make the most realistic looking fake money (counterfeit money)…

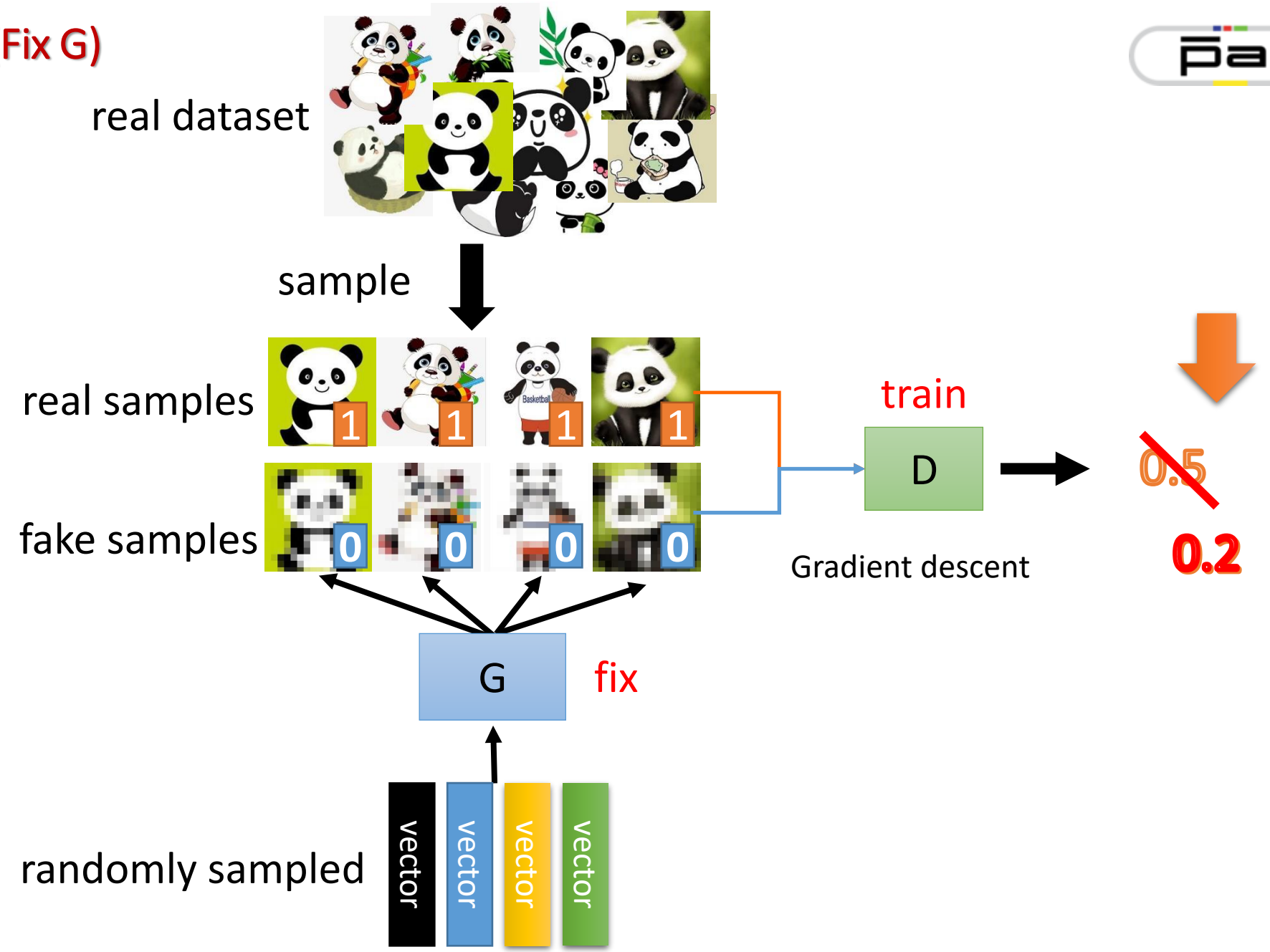
Fake money

Currency detector

Algorithm

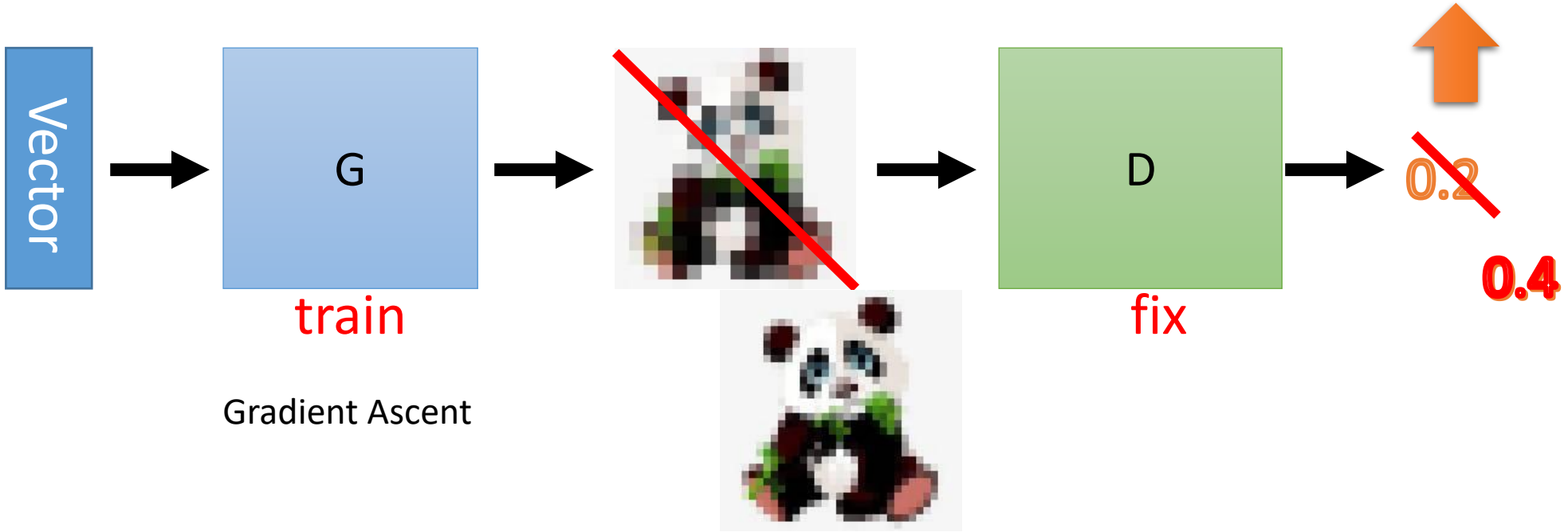
—— How to train a GAN?

Step 1: Train D (Fix G)



Step 2: Train G (Fix D)

Generator learns to “fool” the Discriminator.



Algorithm

- Initialize θ_d for D and θ_g for G
- In each training iteration:

Train D

- Sample m real samples $\{x^1, x^2, \dots, x^m\}$ from dataset
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Generate m fake samples $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to **minimize**
  - $\overline{V_d} = -[\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)]$
  - $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} \overline{V_d}(\theta_d)$

Train G

- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Update generator parameters θ_g to **maximize**
  - $\overline{V_g} = -[\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)] = -[\frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)]$
  - $\overline{V_g} = -\frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D\left(G(z^i)\right)\right)$
  - $\theta_g \leftarrow \theta_g + \eta \nabla_{\theta_g} \overline{V_g}(\theta_g)$

Algorithm of GAN

- Initialize θ_d for D and θ_g for G
- In each training iteration:

Train D

- Sample m real samples $\{x^1, x^2, \dots, x^m\}$ from dataset
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Generate m fake samples $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to minimize
  - $\overline{V_d} = -\left(\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)\right)$
  - $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} \overline{V_d}(\theta_d)$

- Sample m noise samples$\{z^1, z^2, \dots, z^m\}$ from a distribution
- Update generator parameters θ_g to minimize
  - $V_g = \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D\left(G(z^i)\right)\right)$
  - $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V_g(\theta_g)$

Train G

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

Algorithm of GAN

- Initialize θ_d for D and θ_g for G
- In each training iteration:

- Sample m real samples $\{x^1, x^2, \dots, x^m\}$ from dataset
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Generate m fake samples $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to minimize
  - $\overline{V_d} = \text{BCELoss}$
  - $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} \overline{V_d}(\theta_d)$

Train D

- Sample m noise samples$\{z^1, z^2, \dots, z^m\}$ from a distribution
- Update generator parameters θ_g to minimize
  - $V_g = -\text{BCELoss}$
  - $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V_g(\theta_g)$

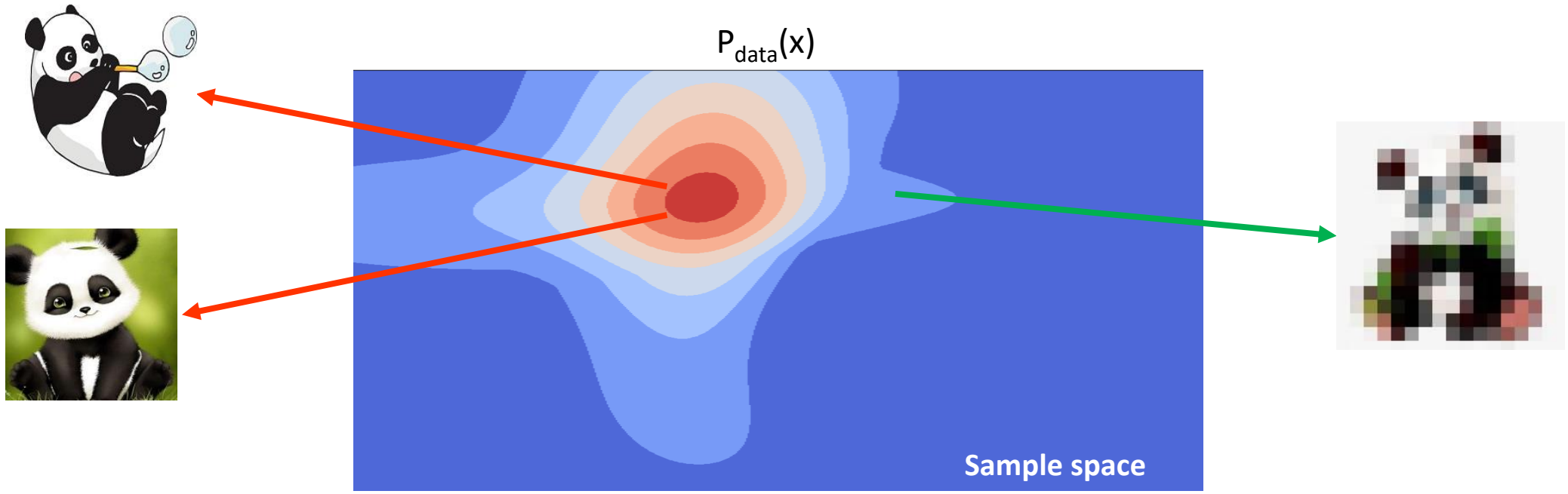
Train G

Theory

—— What is the mathematics behind a GAN?

Probability distribution - $P_{data}(x)$

- Probability distribution of given data
  - $P_{data}(x)$
  - x: samples



Find $P_{data}(x)$ - Maximum likelihood estimation

- **Given** a data probability distribution $P_{data}(x)$
- Assume a distribution $P_G(x;\theta)$ parameterized by θ
- Aim: Find θ such that $P_G(x;\theta)$ close to $P_{data}(x)$

- Assume $P_G(x;\theta)$ is a model parameterized by θ

  (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 $P_{data}(x)$
- Compute $P_G(x^i;\theta)$
- Likelihood function of the sampled samples

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

- Find θ^* by maximizing the likelihood function

Maximizing likelihood function is equivalent to minimizing Kullback-Leibler Divergence

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

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

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

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

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

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

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

$$= \arg\min_\theta [-\int_x P_{data}(x)\log P_G(x;\theta)dx + \int_x P_{data}(x)\log P_{data}(x)dx]$$

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

Maximizing likelihood function is equivalent to minimizing KLD

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

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

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

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

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

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

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

$$= \arg\min_\theta [-\int_x P_{data}(x)\log P_G(x;\theta)dx + \int_x P_{data}(x)\log P_{data}(x)dx]$$

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

Cross entropy
$H(P_{data}(x), P_G(x;\theta))$

Entropy
$-H(P_{data}(x))$

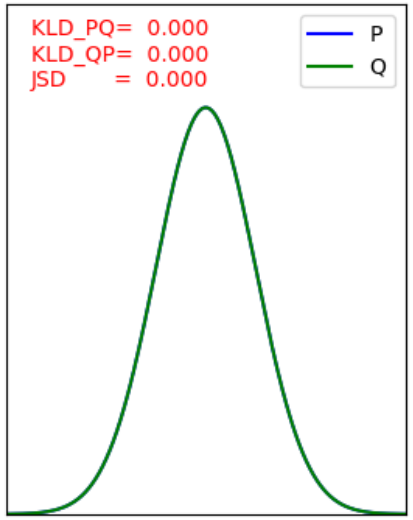
Distance

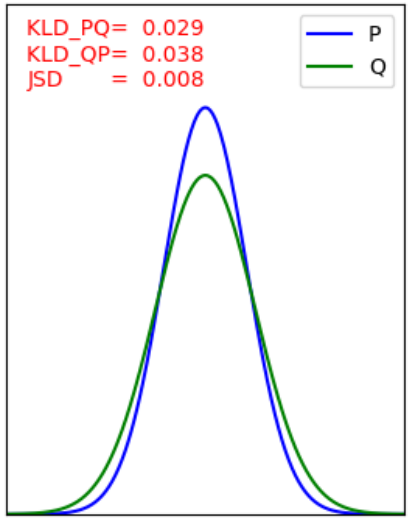
- Point to Point
  - $\mathrm{p-norm\ distance} = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$
- Distribution to Distribution
  - Kullback–Leibler divergence
    - $D_{KL}(P||Q) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)})$
    - $D_{KL}(P||Q) = \int_{-\infty}^{\infty} P(x) \log(\frac{P(x)}{Q(x)})\, dx$
    - $D_{KL}(P||Q) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)})$

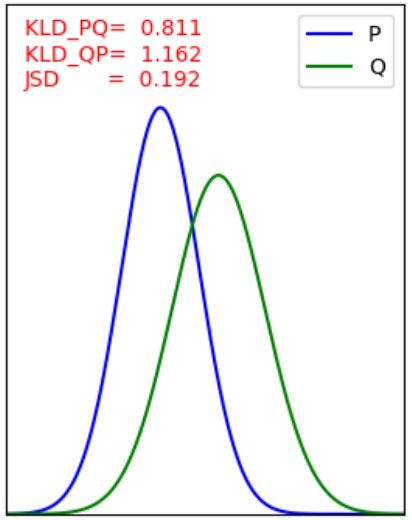
      $= -\sum_{x \in X} P(x) \log(Q(x)) + \sum_{x \in X} P(x) \log(P(x))$

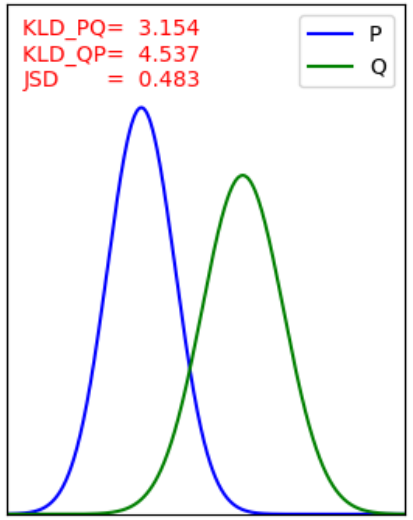
      $= H(P, Q) - H(P)$
    - $D_{KL}(P||Q) \neq D_{KL}(Q||P)$
  - Jensen–Shannon divergence
    - $JSD(P||Q) = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M)$, where $M = \frac{1}{2}(P + Q)$
    - $JSD(P||Q) = JSD(Q||P)$

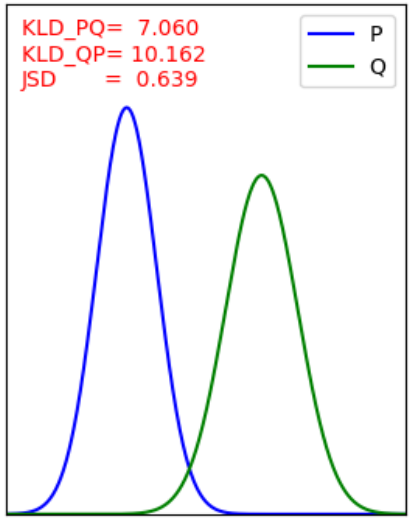
KLD & JSD

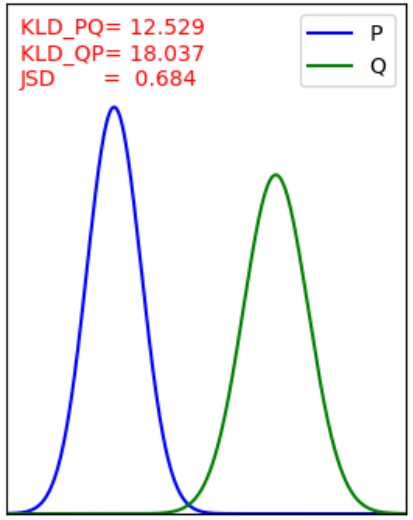

KLD_PQ= 0.000
KLD_QP= 0.000
JSD = 0.000
P
Q

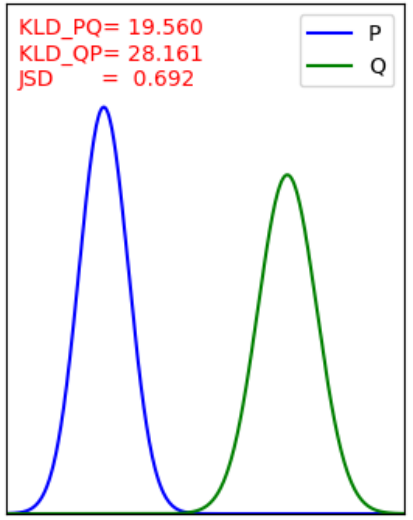

KLD_PQ= 0.029
KLD_QP= 0.038
JSD = 0.008
P
Q

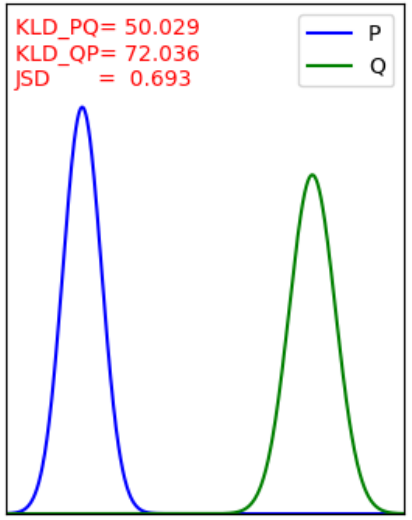

KLD_PQ= 0.811
KLD_QP= 1.162
JSD = 0.192
P
Q

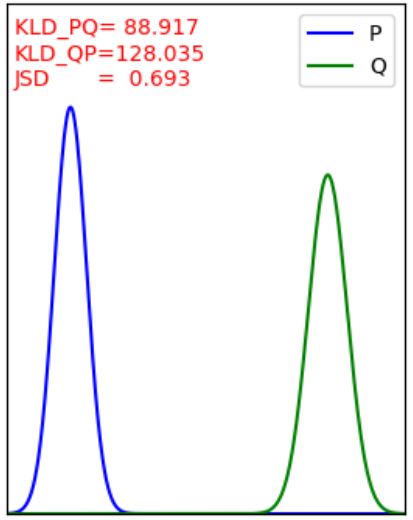

KLD_PQ= 3.154
KLD_QP= 4.537
JSD = 0.483
P
Q

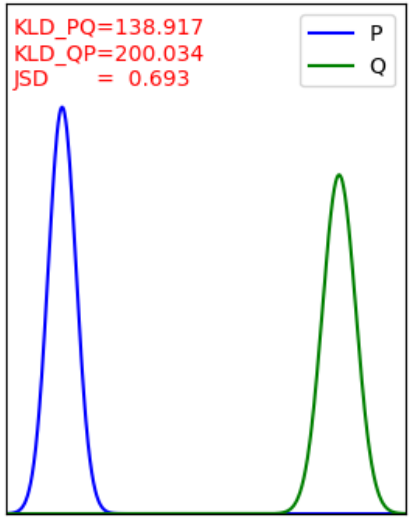

KLD_PQ= 7.060
KLD_QP= 10.162
JSD = 0.639
P
Q


KLD_PQ= 12.529
KLD_QP= 18.037
JSD = 0.684
P
Q


KLD_PQ= 19.560
KLD_QP= 28.161
JSD = 0.692
P
Q


KLD_PQ= 50.029
KLD_QP= 72.036
JSD = 0.693
P
Q


KLD_PQ= 88.917
KLD_QP=128.035
JSD = 0.693
P
Q


KLD_PQ=138.917
KLD_QP=200.034
JSD = 0.693
P
Q

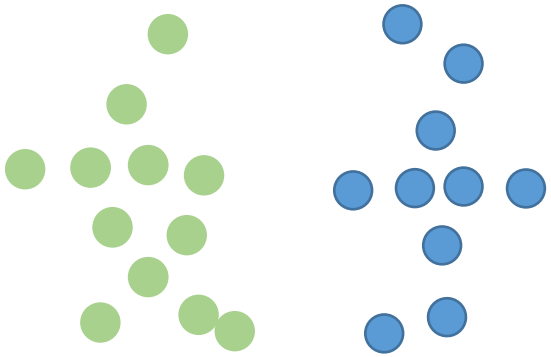
Divergence and discriminator

- : data sampled from P_{data}
- : data sampled from P_G

small divergence

Discriminator

hard to discriminate



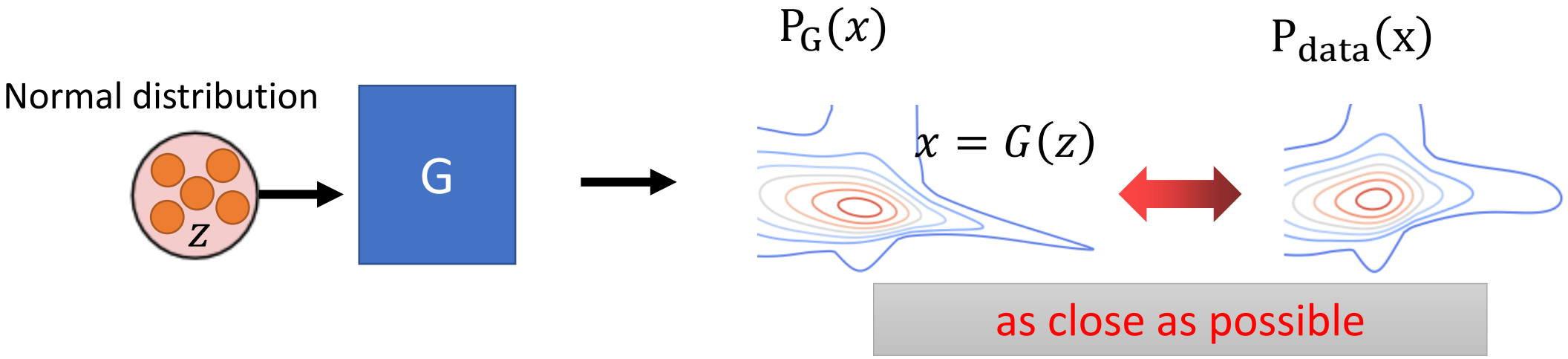
large divergence

Discriminator

easy to discriminate

Generator

- A generator G is a network which defines a probability distribution P_G

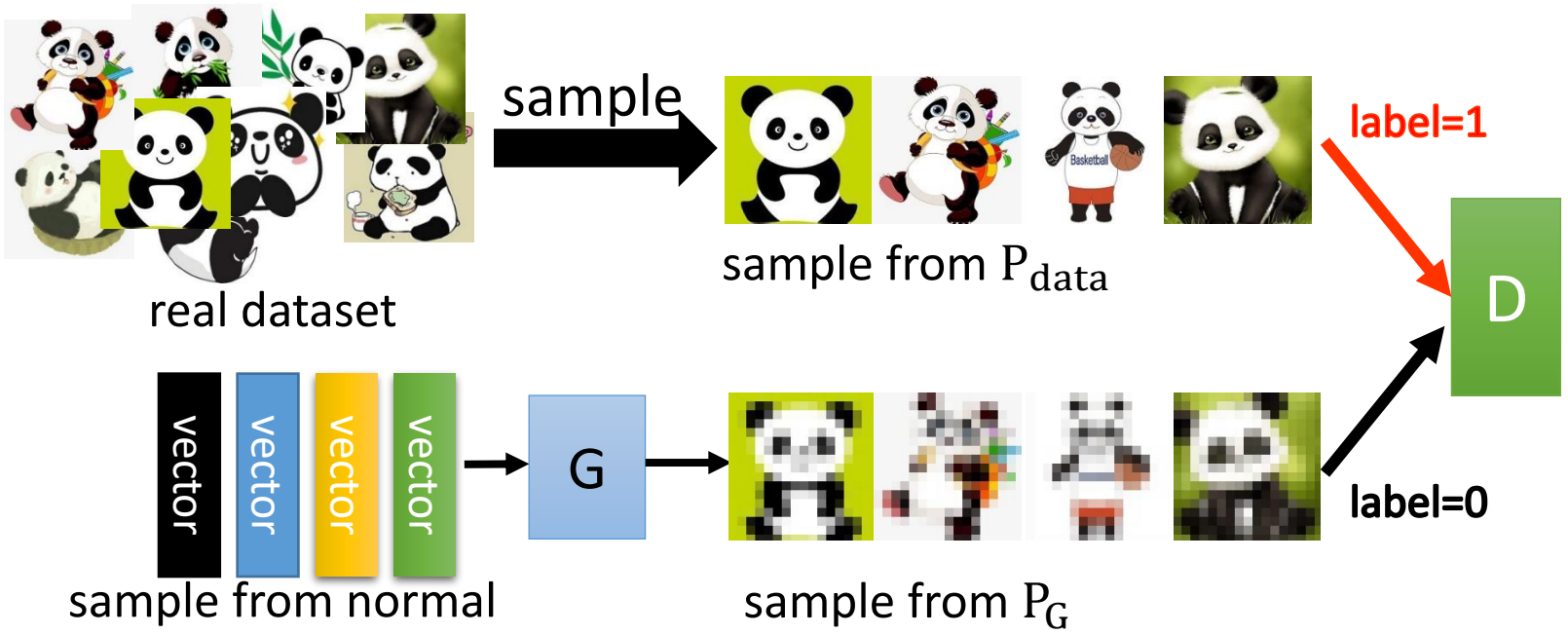


Train G: $G^* = \arg\min_G \mathrm{Div}(P_G, P_{data})$

Divergence between distributions P_G and P_{data}

Discriminator

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



$$\overline{V_d} = -\left(\frac{1}{m}\sum_{i=1}^{m}\log D(x^i) + \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D(\tilde{x}^i)\right)\right)$$

- Objective Function for D

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

↑ $V(D,G)$; $D(x)$ → 1; $D(\tilde{x})$ → 0; G is fixed

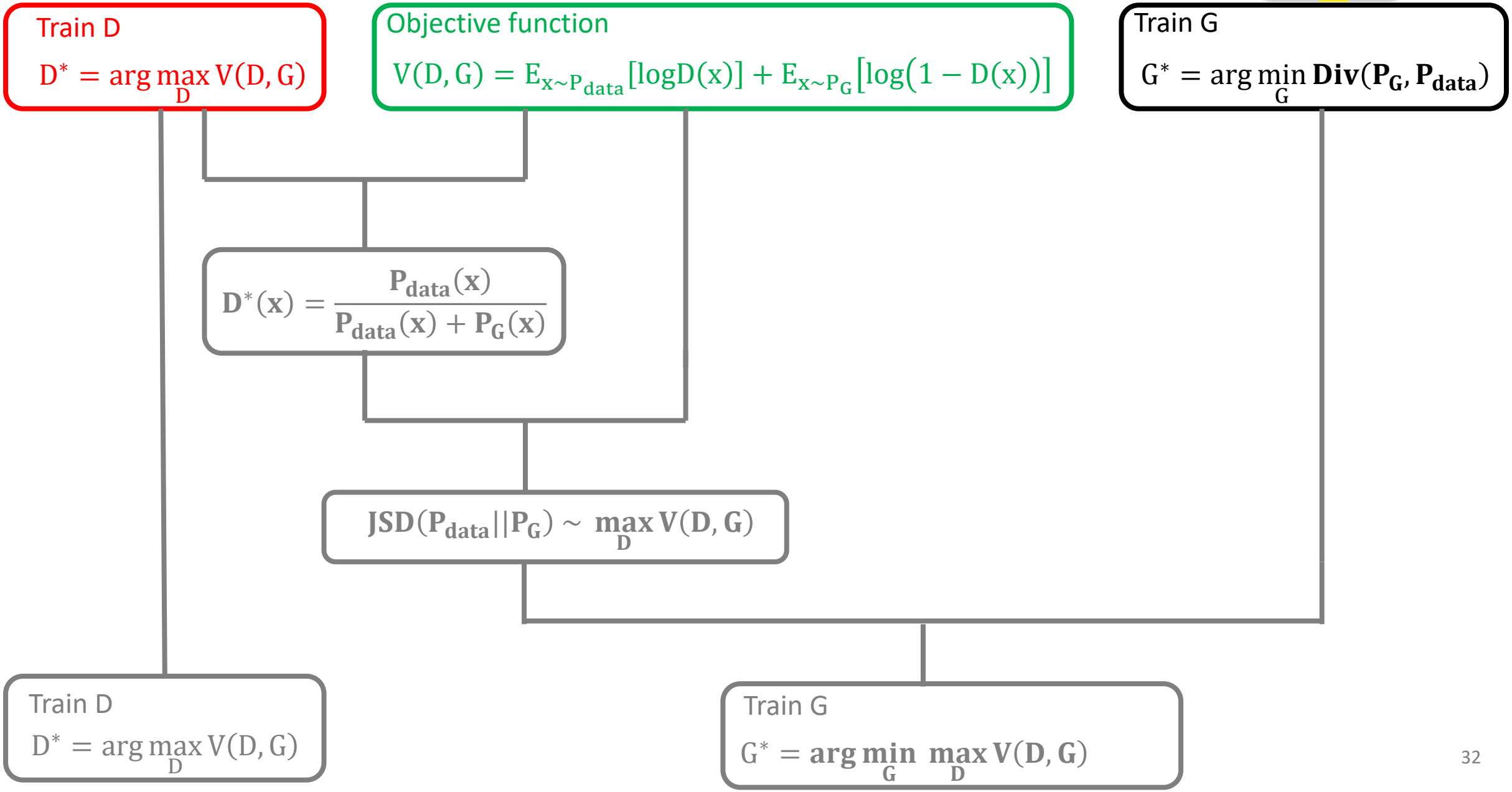
- Train D: $D^* = \arg\max_D V(D,G)$

Goodfellow, et al., NIPS, 2014

G^* & D^*

- $G^* = \arg\min_G \mathrm{Div}(P_G, P_{data})$
- $D^* = \arg\max_D V(D, G)$
- $V(D, G) = E_{x \sim P_{data}}[\log D(x)] + E_{x \sim P_G}[\log(1 - D(x))]$

Family tree


Train D
$D^* = \arg\max_D V(D,G)$
Objective function
$V(D,G) = E_{x\sim P_{data}}[\log D(x)] + E_{x\sim P_G}[\log(1-D(x))]$
Train G
$G^* = \arg\min_G \mathbf{Div}(P_G, P_{data})$
$D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$
$JSD(P_{data}||P_G) \sim \max_D V(D,G)$
Train D
$D^* = \arg\max_D V(D,G)$
Train G
$G^* = \arg\min_G \max_D V(D,G)$

D*

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

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

  $= \int_x [P_{data}(x)\log D(x) + P_G(x)\log(1-D(x))]\,dx$

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

- Given G， assuming that $D(x)$ can be any function, the optimal D^* maximize

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

- $f(D) = a\log(D) + b\log(1-D)$
  - $\frac{df(D)}{dD} = 0 \quad \Rightarrow D^* = \frac{a}{a+b}$

- $D^*(x) = \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}$ $\quad D^*(x) \in (0,1)$

$\max_D V(D, G)$

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

$$D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

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

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

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

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

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

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

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

G^* & D^* - Div → $\max_D V(D, G)$

$D^* = \arg\max_D V(D, G)$ $\quad$ $G^* = \arg\min_G \mathbf{Div}(P_G, P_{data})$

$\max_D V(D, G) = V(D^*, G) = -2\log 2 + 2JSD(P_{data}||P_G)$

$\mathbf{JSD}(P_{data}||P_G) \sim \max_D V(D, G)$

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

Family tree

Train D

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

Objective function

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

Train G

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

$$\mathbf{D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}}$$

$$\mathbf{JSD(P_{data} || P_G) \sim \max_D V(D, G)}$$

Train D

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

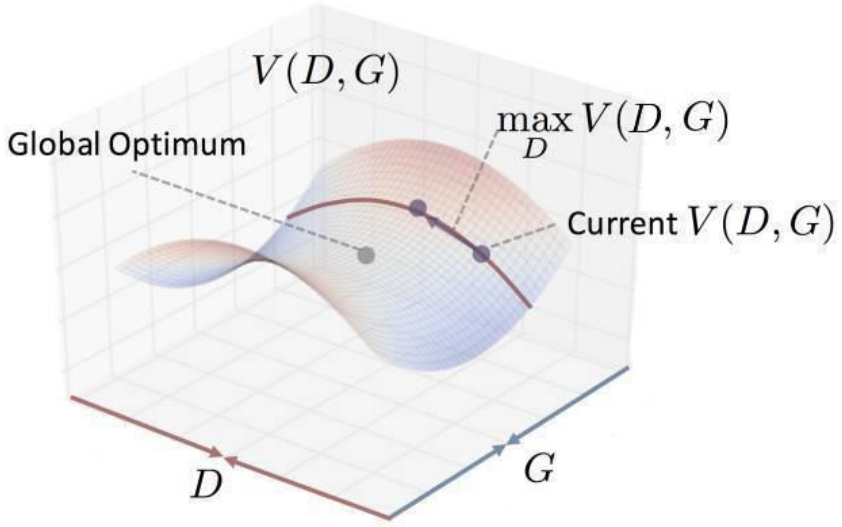
Train G

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

D^* & G^* - Algorithm

Step 1. Train D

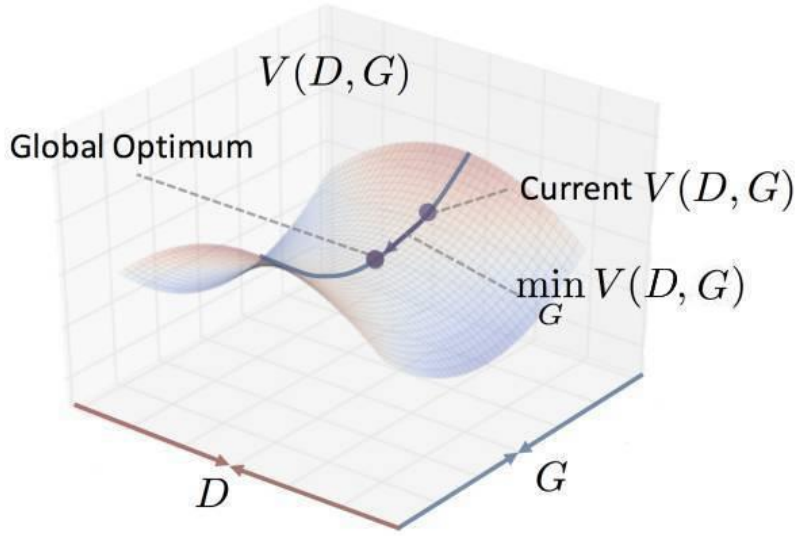
- $D^* = \arg\max_D V(D, G)$



- Fix G, and update D

Step 2. Train G

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

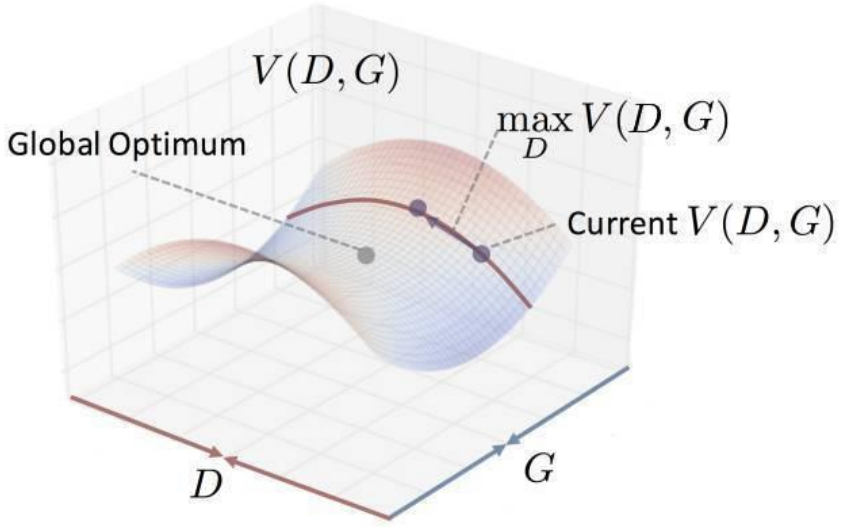


- Fix D, and update G

D^* & G^* - Algorithm

Step 1. Train D

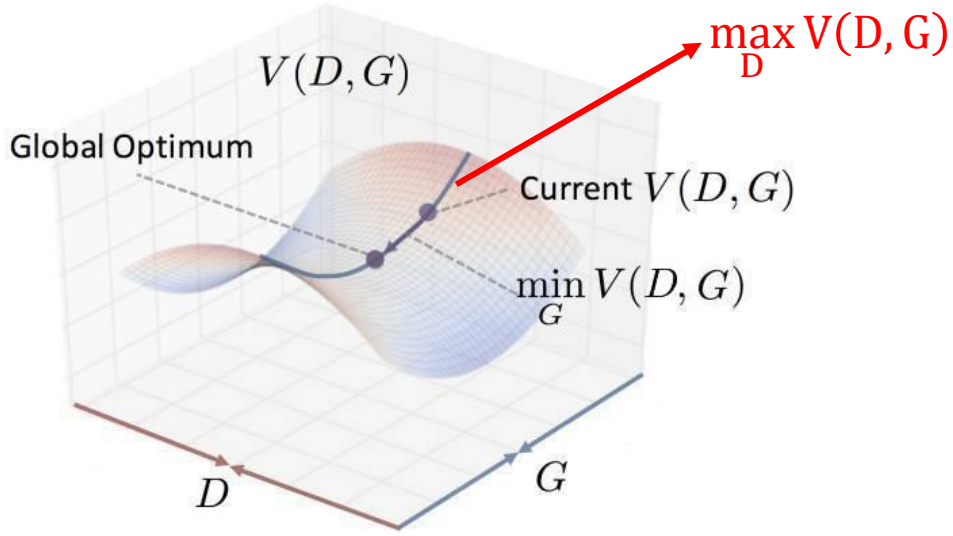
- $D^* = \arg\max_D V(D, G)$



- Fix G, and update D

Step 2. Train G

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

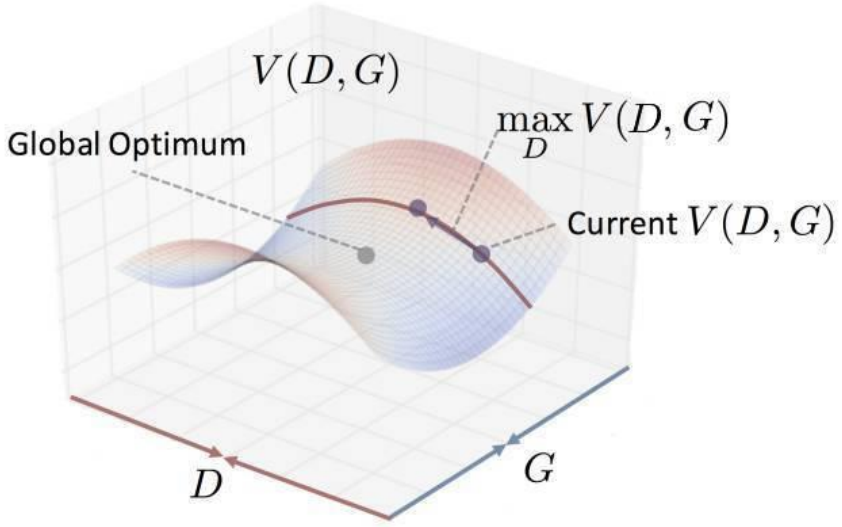


- Fix D, and update G

D^* & G^* - Algorithm

Step 1. Train D

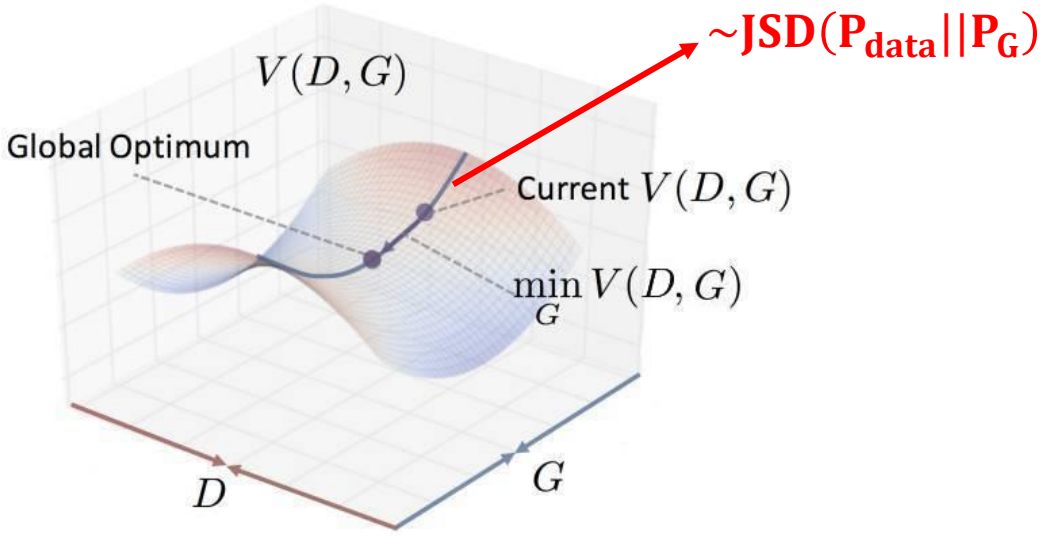
- $D^* = \arg\max_D V(D, G)$



- Fix G, and update D

Step 2. Train G

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



- Fix D, and update G

Family tree

Train D

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

Objective function

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

Train G

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

$$\mathbf{D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}}$$

$$\mathbf{JSD(P_{data} || P_G) \sim \max_D V(D, G)}$$

Train D

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

Train G

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

Family tree

Train D

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

Objective function

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

Train G

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

$$D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

$$\mathbf{JSD}(P_{data} || P_G) \sim \max_D V(D, G)$$

Train D

$$D^* = D \mid \mathbf{JSD}(P_{data} || P_G)$$

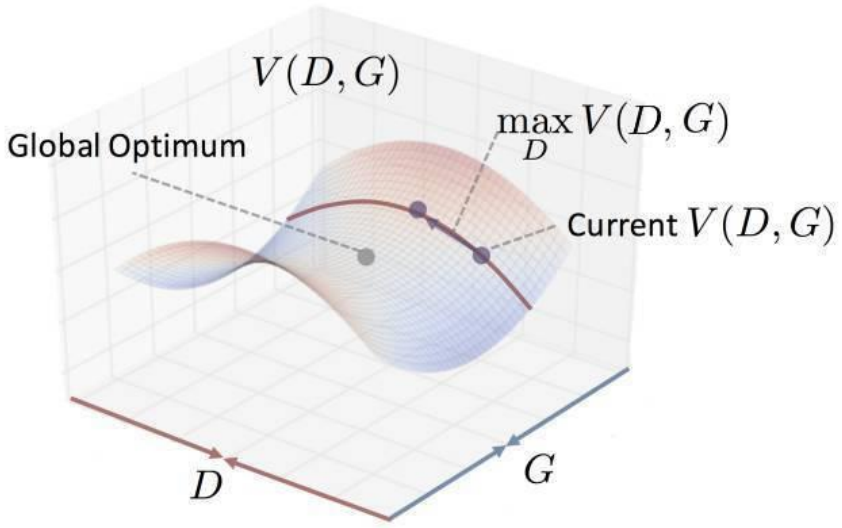
Train G

$$G^* = \arg\min_G \mathbf{JSD}(P_{data} || P_G)$$

D^* & G^* - Algorithm

Step 1. Train D

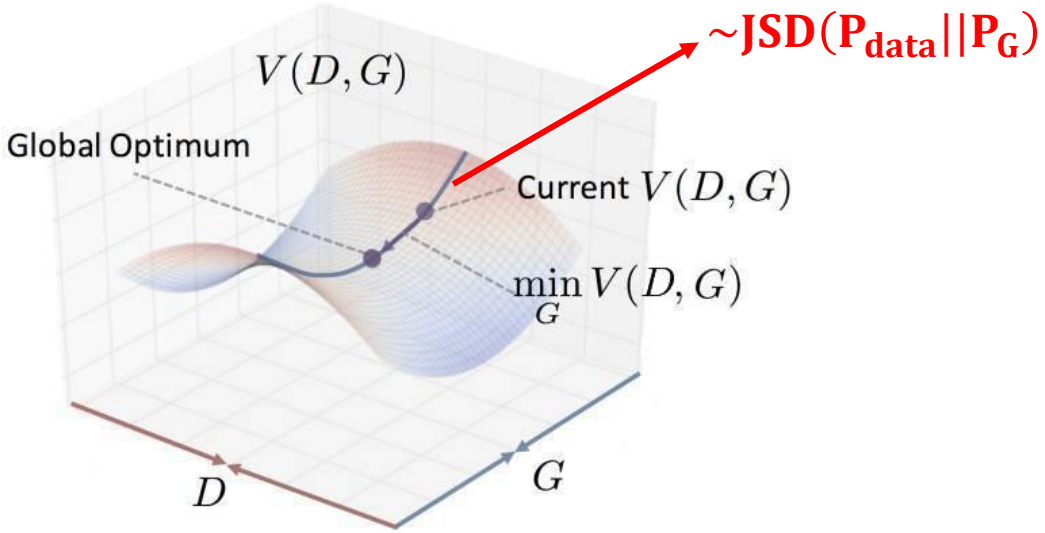
- $D^* = D \mid JSD(P_{data}||P_G)$
- **Evaluate $JSD(P_{data}||P_G)$**



- Fix G, and update D

Step 2. Train G

- $G^* = \arg\min_G JSD(P_{data}||P_G)$
- **Minimize $JSD(P_{data}||P_G)$**



- Fix D, and update G

Weights update

$\mathbf{D^*}$ - Evaluate $\mathbf{JSD(P_{data}||P_G)}$

- $D^* = D \mid JSD(P_{data}||P_G)$
- $D^* = \arg\max_D V(D, G)$
- $\theta_d \leftarrow \theta_d + \eta \nabla_{\theta_d} V(D, G)$

- Repeat n times to evaluate $JSD(P_{data}||P_G)$ correctly

$\mathbf{G^*}$ - Minimize $\mathbf{JSD(P_{data}||P_G)}$

- $G^* = \arg\min_G JSD(P_{data}||P_G)$
- $G^* = \arg\min_G \max_D V(D, G)$

  $= \arg\min_G V(D^*, G)$
- $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V(D, G)$

- Run once to change P_G slightly

Weights update

D^* - Evaluate $JSD(P_{data}||P_G)$

- $D^* = D \mid JSD(P_{data}||P_G)$
- $D^* = \arg\max_D V(D,G)$
- $\theta_d \leftarrow \theta_d + \eta\nabla_{\theta_d} V(D,G)$

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

- $V_d = \frac{1}{m}\sum_{i=1}^m \log D(x^i) + \frac{1}{m}\sum_{i=1}^m \log\left(1-D(\tilde{x}^i)\right)$

G^* - Minimize $JSD(P_{data}||P_G)$

- $G^* = \arg\min_G JSD(P_{data}||P_G)$
- $G^* = \arg\min_G \max_D V(D,G)$

  $= \arg\min_G V(D^*, G)$
- $\theta_g \leftarrow \theta_g - \eta\nabla_{\theta_g} V(D,G)$
- $V = E_{x\sim P_{data}}[\log D(x)] + E_{x\sim P_G}[\log(1-D(x))]$
- ~~Sample $\{x^1, x^2, \dots, x^m\}$ from $P_{data}(x)$~~
- $\{z^1, z^2, \dots, z^m\}$ from the prior $P_{prior}(z)$
- $V_g = \frac{1}{m}\sum_{i=1}^m \log D(x^i) + \frac{1}{m}\sum_{i=1}^m \log(1-D(G(z)))$
- $V_g = \frac{1}{m}\sum_{i=1}^m \log(1-D(G(z)))$

Weights update

D^* - Evaluate $JSD(P_{data}||P_G)$

- $D^* = D \mid JSD(P_{data}||P_G)$
- $D^* = \arg\max_D V(D, G)$

- $\theta_d \leftarrow \theta_d + \eta \nabla_{\theta_d} V(D, G)$
- $V_d = \frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)$

- $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} V(D, G)$
- $\overline{V_d} = -\left[\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)\right]$
- $\overline{V_d}$ = BCE Loss

G^* - Minimize $JSD(P_{data}||P_G)$

- $G^* = \arg\min_G JSD(P_{data}||P_G)$
- $G^* = \arg\min_G \max_D V(D, G)$

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

- $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V(D, G)$
- $V_g = \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(G(z))\right)$

- $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V(D, G)$
- $V_g = \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(G(z))\right)$
- V_g= - BCE Loss

Algorithm of GAN

- Initialize θ_d for D and θ_g for G
- In each training iteration:
  - Sample m real samples $\{x^1, x^2, \dots, x^m\}$ from dataset
  - Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
  - Generate m fake samples $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
  - Update discriminator parameters θ_d to minimize
    - $\overline{V_d} = -\left(\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)\right)$

      $\overline{V_d}$=BCE Loss
    - $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} \overline{V_d}(\theta_d)$

Train D
Repeat n times

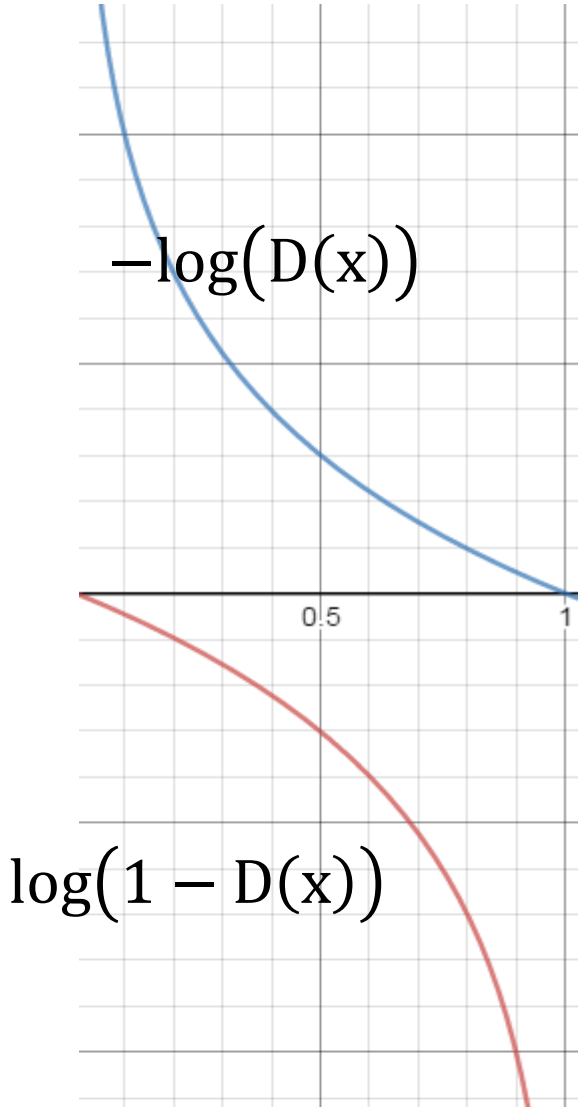
- Sample m noise samples$\{z^1, z^2, \dots, z^m\}$ from a distribution
- Update generator parameters θ_g to minimize
  - $V_g = \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z)))$

    V_g= - BCE Loss
  - $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V_g(\theta_g)$

Train G
Run once

MMGAN & NSGAN

- MMGAN(Minimax Gan)
  - $V_g = E_{x\sim P_G}\left[\log\left(1 - D(x)\right)\right]$
- NSGAN(Non-saturating GAN)
  - $V_g = E_{x\sim P_G}\left[-\log\left(D(x)\right)\right]$



Algorithm of NSGAN

- Initialize θ_d for D and θ_g for G
- In each training iteration:

  - Sample m real samples $\{x^1, x^2, \dots, x^m\}$ from dataset
  - Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
  - Generate m fake samples $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
  - Update discriminator parameters θ_d to minimize
    - $\overline{V_d} = -\left(\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)\right)$
    - $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} \overline{V_d}(\theta_d)$

  Train D
  Repeat n times

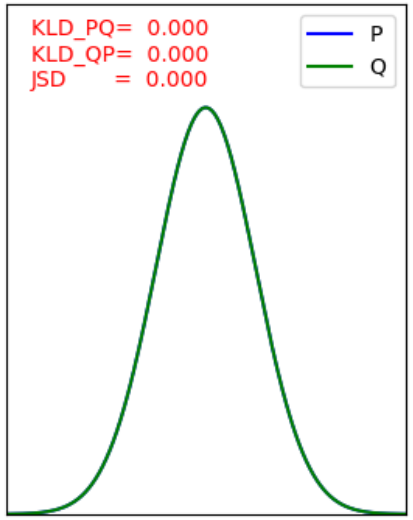
  - Sample m noise samples$\{z^1, z^2, \dots, z^m\}$ from a distribution
  - Update generator parameters θ_g to minimize
    - $V_g = \frac{1}{m}\sum_{i=1}^{m}\left[-\log\left(D(G(z))\right)\right]$
    - $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V_g(\theta_g)$

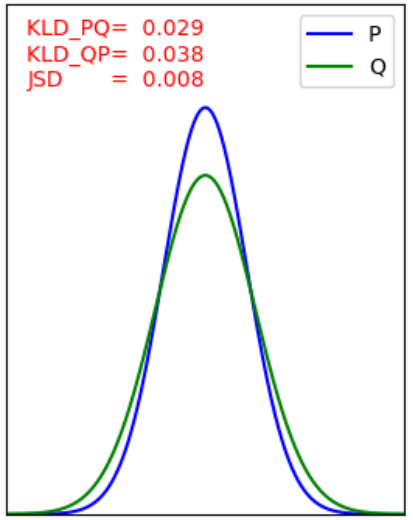
  Train G
  Run once

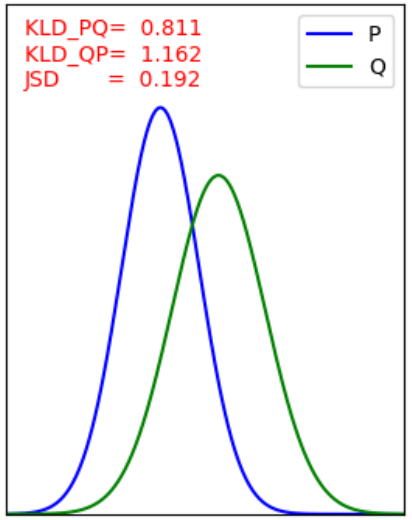
WGAN

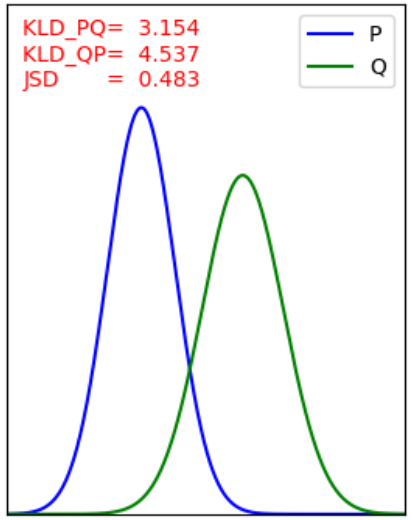
—— Is there a more stable GAN?

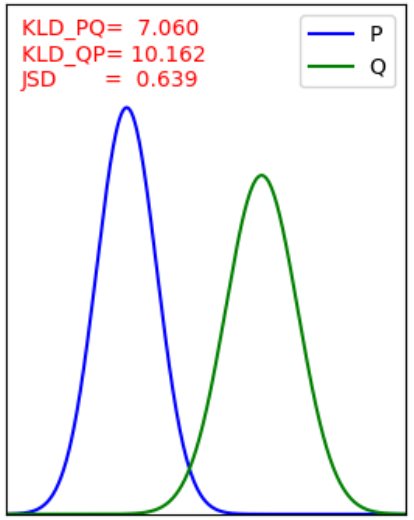
KLD & JSD

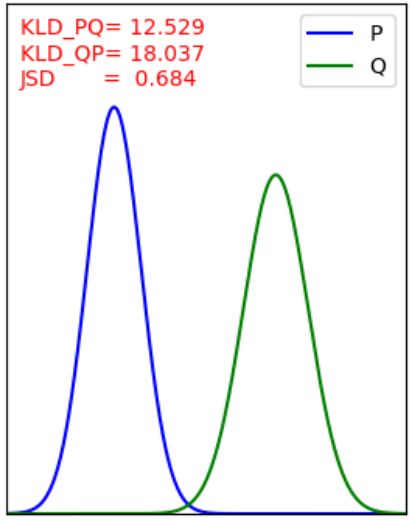

KLD_PQ= 0.000
KLD_QP= 0.000
JSD = 0.000
P
Q

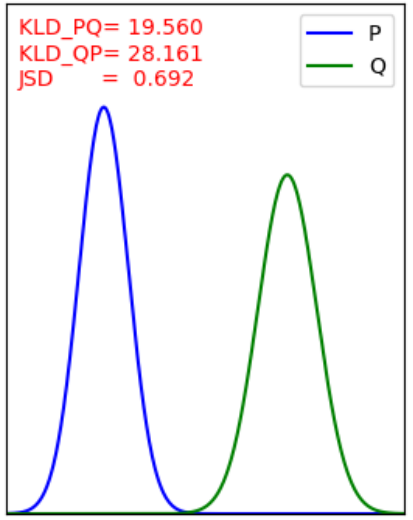

KLD_PQ= 0.029
KLD_QP= 0.038
JSD = 0.008
P
Q

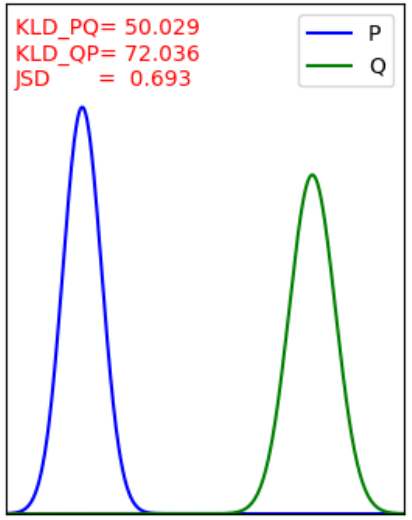

KLD_PQ= 0.811
KLD_QP= 1.162
JSD = 0.192
P
Q

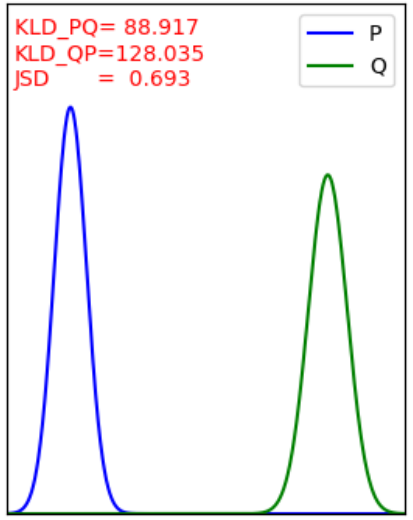

KLD_PQ= 3.154
KLD_QP= 4.537
JSD = 0.483
P
Q

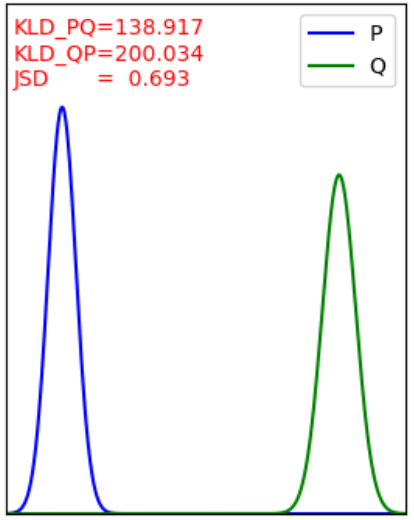

KLD_PQ= 7.060
KLD_QP= 10.162
JSD = 0.639
P
Q


KLD_PQ= 12.529
KLD_QP= 18.037
JSD = 0.684
P
Q


KLD_PQ= 19.560
KLD_QP= 28.161
JSD = 0.692
P
Q

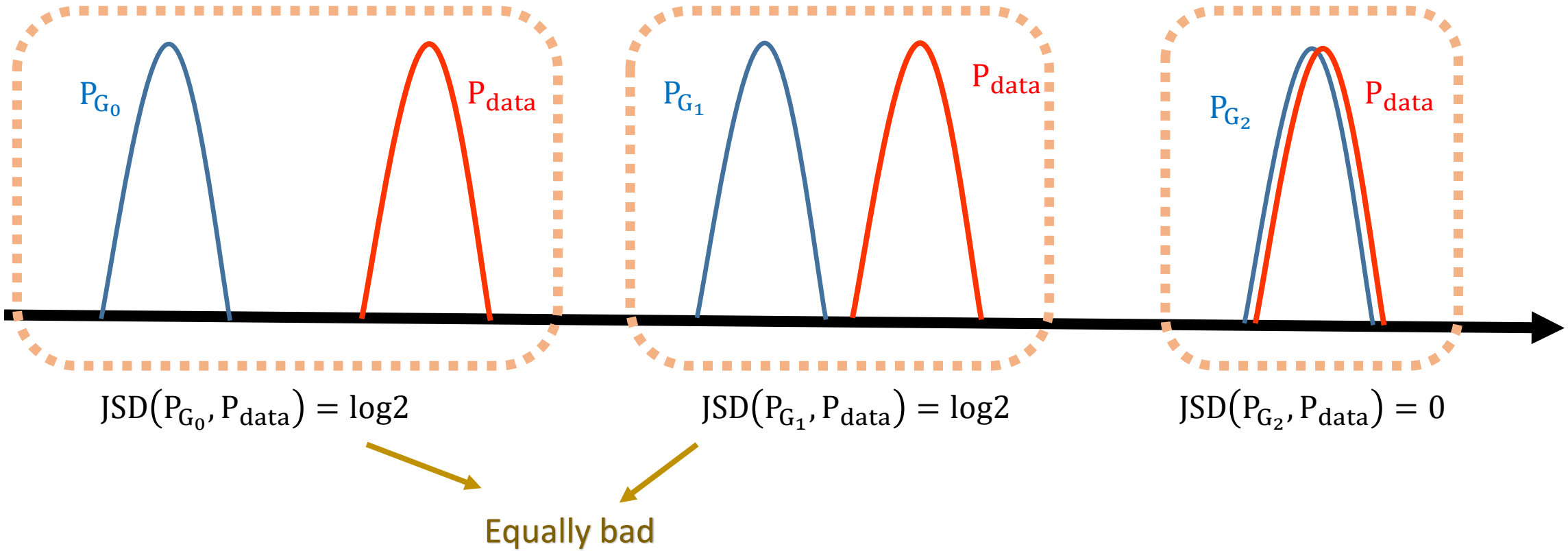

KLD_PQ= 50.029
KLD_QP= 72.036
JSD = 0.693
P
Q


KLD_PQ= 88.917
KLD_QP=128.035
JSD = 0.693
P
Q


KLD_PQ=138.917
KLD_QP=200.034
JSD = 0.693
P
Q

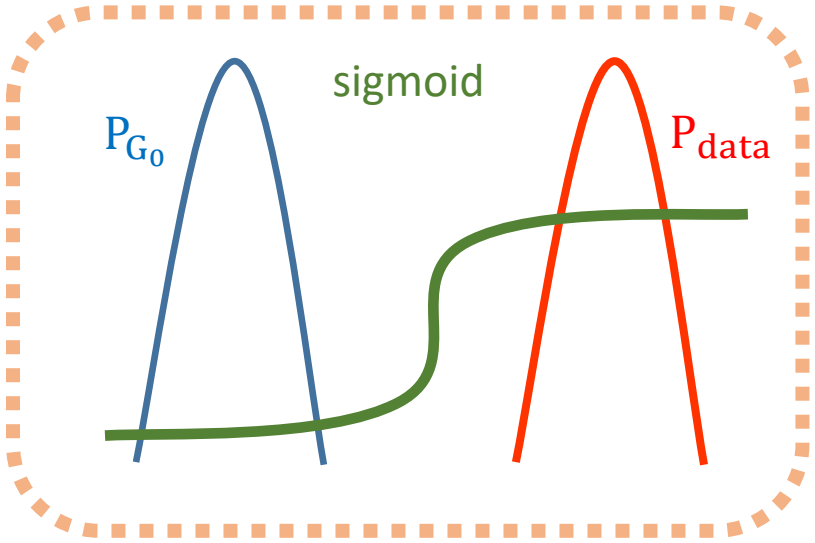
Jensen–Shannon divergence

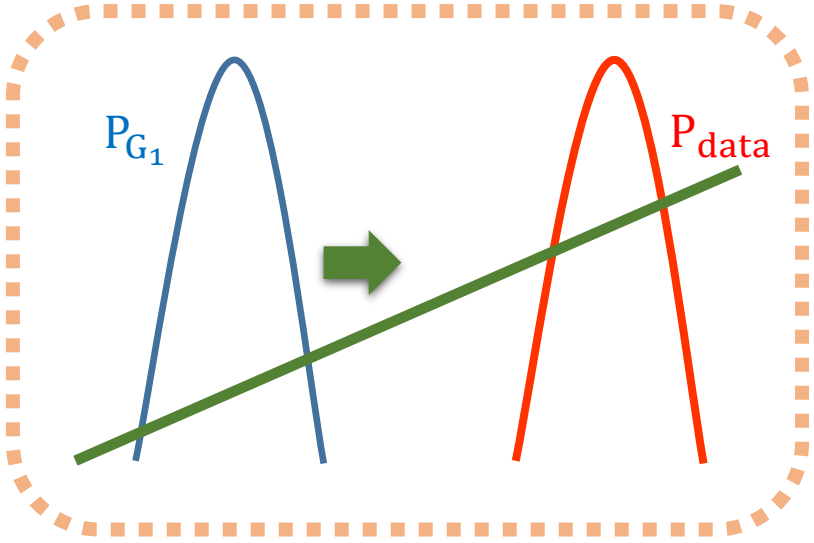
- $\mathrm{JSD}(\mathrm{P}||\mathrm{Q}) = \frac{1}{2}\mathrm{KLD}(\mathrm{P}||\mathrm{M}) + \frac{1}{2}\mathrm{KLD}(\mathrm{Q}||\mathrm{M})$, where $\mathrm{M} = \frac{1}{2}(\mathrm{P} + \mathrm{Q})$
- $\mathrm{D_{KL}}(\mathrm{P}||\mathrm{Q}) = \int_{-\infty}^{\infty} \mathrm{P(x)}\log(\frac{\mathrm{P(x)}}{\mathrm{Q(x)}})\,\mathrm{dx}$
- if $\mathrm{P_{data}}$ and $\mathrm{P_G}$ and are not overlapped, $\mathrm{JSD}(\mathrm{P}||\mathrm{Q})=\log 2$



Solution: WGAN(Wasserstein GAN)

- Replace binary classification with linear regression
- Weight clipping
  - Force the parameters w between c and –c : if w > c, w = c; if w < -c, w = -c





P_{G_0} does not move

Algorithm of GAN

- Initialize θ_d for D and θ_g for G
- In each training iteration:

- Sample m real samples $\{x^1, x^2, \dots, x^m\}$ from dataset
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Generate m fake samples $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to minimize
  - $\overline{V_d} = -\left(\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D(\tilde{x}^i)\right)\right)$
  - $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} \overline{V_d}(\theta_d)$

- Sample m noise samples$\{z^1, z^2, \dots, z^m\}$ from a prior distribution
- Update generator parameters θ_g to minimize
  - $V_g = \frac{1}{m}\sum_{i=1}^{m}\left[-\log\left(D(G(z))\right)\right]$
  - $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V_g(\theta_g)$

Algorithm of WGAN

- Initialize θ_d for D and θ_g for G
- In each training iteration:

No sigmoid for the output of D

Weight clipping

- Sample m real samples $\{x^1, x^2, \dots, x^m\}$ from dataset
- Sample m noise samples $\{z^1, z^2, \dots, z^m\}$ from a distribution
- Generate m fake samples $\{\tilde{x}^1, \tilde{x}^2, \dots, \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters θ_d to minimize
  - $\overline{V_d} = -\left[\frac{1}{m}\sum_{i=1}^{m} D(x^i) - \frac{1}{m}\sum_{i=1}^{m} D(\tilde{x}^i)\right]$
  - $\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} \overline{V_d}(\theta_d)$

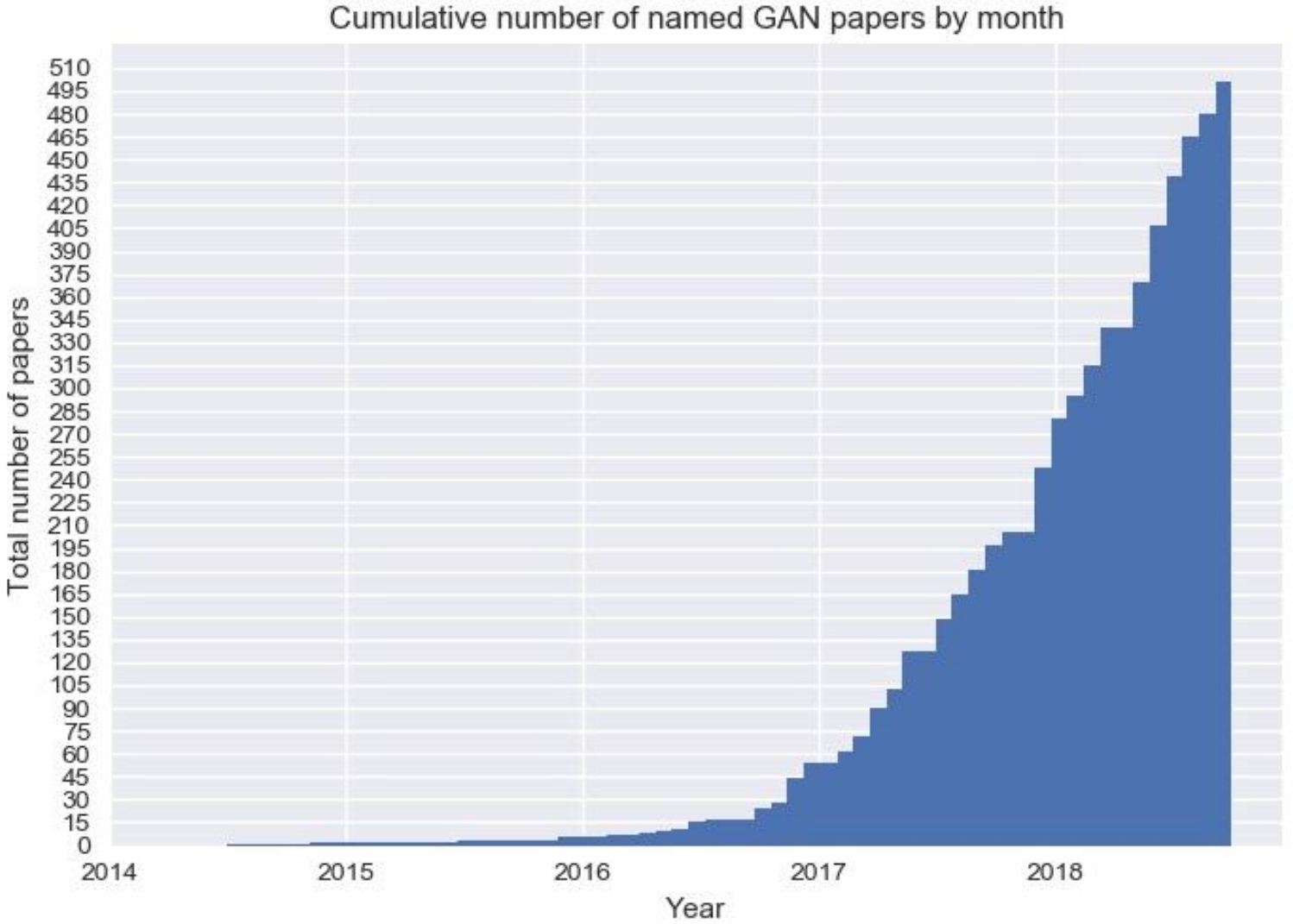
- Sample m noise samples$\{z^1, z^2, \dots, z^m\}$ from a prior distribution
- Update generator parameters θ_g to minimize
  - $V_g = \frac{1}{m}\sum_{i=1}^{m} -D\left(G(z^i)\right)$
  - $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} V_g(\theta_g)$

GANs

—— Are there other GANs?

The GAN Zoo

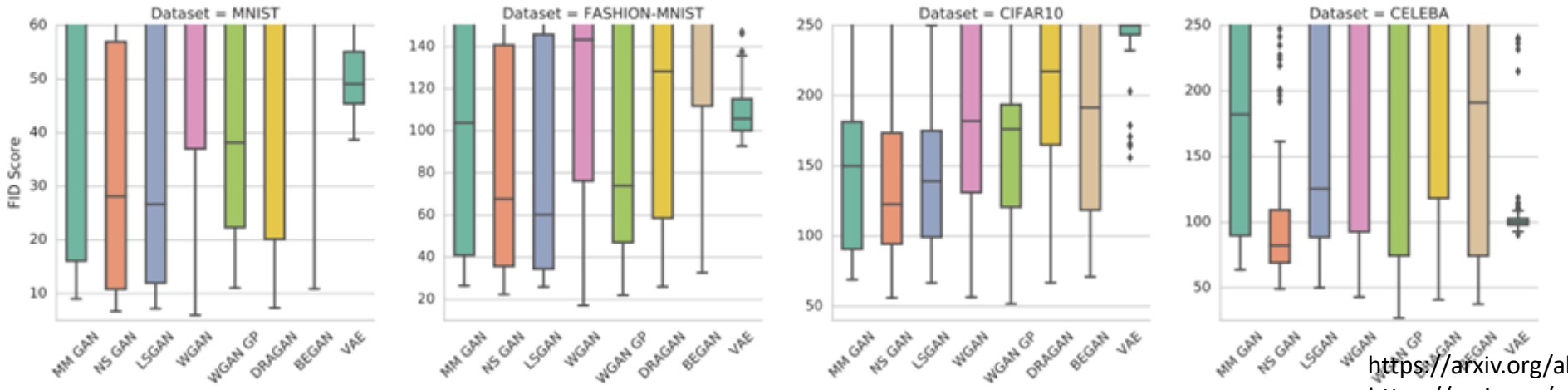
- Modified GAN-CLS - Generate the corresponding image from Text Description using Modified Algorithm
- ModularGAN - Modular Generative Adversarial Networks
- MolGAN - MolGAN: An implicit generative model for small molecular graphs
- MPM-GAN - Message Passing Multi-Agent GANs
- MS-GAN - Temporal Coherency based Criteria for Predicting Video Frames using Deep Multi-s Adversarial Networks
- MTGAN - MTGAN: Speaker Verification through Multitasking Triplet Generative Adversarial Net
- MuseGAN - MuseGAN: Symbolic-domain Music Generation and Accompaniment with Multi-tr Generative Adversarial Networks
- MV-BiGAN - Multi-view Generative Adversarial Networks
- N2RPP - N2RPP: An Adversarial Network to Rebuild Plantar Pressure for ACLD Patients
- NAN - Understanding Humans in Crowded Scenes: Deep Nested Adversarial Learning and A N Multi-Human Parsing
- NCE-GAN - Dihedral angle prediction using generative adversarial networks
- ND-GAN - Novelty Detection with GAN
- NetGAN - NetGAN: Generating Graphs via Random Walks
- OCAN - One-Class Adversarial Nets for Fraud Detection
- OptionGAN - OptionGAN: Learning Joint Reward-Policy Options using Generative Adversarial I Reinforcement Learning
- ORGAN - Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Gener
- ORGAN - 3D Reconstruction of Incomplete Archaeological Objects Using a Generative Adversa
- OT-GAN - Improving GANs Using Optimal Transport
- PacGAN - PacGAN: The power of two samples in generative adversarial networks
- PAN - Perceptual Adversarial Networks for Image-to-Image Transformation
- PassGAN - PassGAN: A Deep Learning Approach for Password Guessing
- PD-WGAN - Primal-Dual Wasserstein GAN
- Perceptual GAN - Perceptual Generative Adversarial Networks for Small Object Detection
- PGAN - Probabilistic Generative Adversarial Networks
- PGD-GAN - Solving Linear Inverse Problems Using GAN Priors: An Algorithm with Provable Gu



https://github.com/hindupuravinash/the-gan-zoo

GANs & Performance

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] + \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = -\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}}$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}}$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} - = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}}$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x} - 1)^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = -\mathcal{L}_{\mathrm{D}}^{\text{NS GAN}}$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x - \mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$



https://arxiv.org/abs/1711.10337
https://arxiv.org/pdf/1706.08500

The smaller, the better

Evaluation

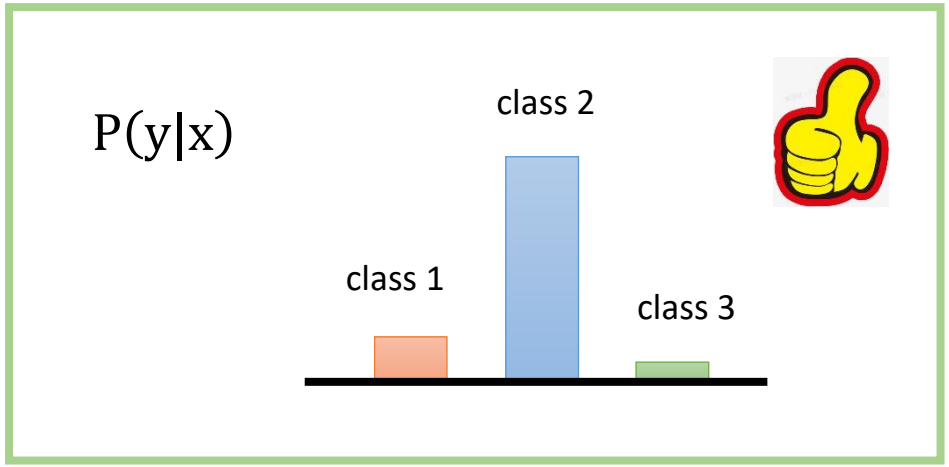
—— How good is a GAN?

Inception Score

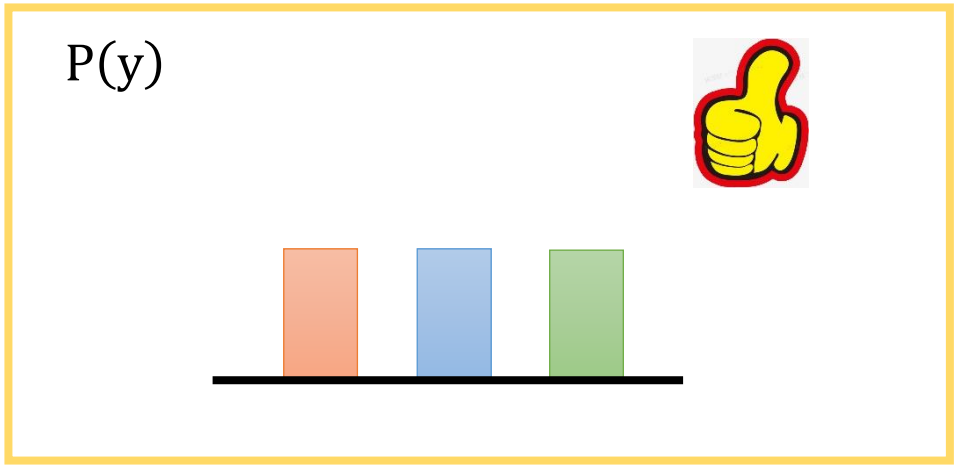
x: samples

y: classes

Quality



Diversity



$$\text{Inception Score} = \sum_{x,y} P(y|x)\log P(y|x) - \sum_{y} P(y)\log P(y)$$

Negative entropy of P(y|x)

Entropy of P(y)

https://arxiv.org/pdf/1801.01973.pdf

Tricks

—— What are the tricks to make a GAN work?

Tips and tricks to make GANs work

- Normalize the samples between -1 and 1 and use Tanh as the last layer of the generator
- Sample from a Gaussian distribution rather than a Uniform distribution as the input of generator
- Replace Relu with LeakyReLU both in discriminator and generator
- Use soft and noisy labels by replacing label 0 with a random number between 0.0 and 0.3 and replacing label 1 with a random number between 0.8 and 1.2
- Increase discriminator and generator network capacity with more filters and more layers
- Add artificial noise to the real and generated samples before feeding them into the discriminator, and decay over time
- Feed real samples and generated samples into the discriminator separately in different batches
- Feed the discriminator with all recent generated samples to avoid overfitting
- Try different loss functions, though no single loss function always performs best among all datasets
- Stabilize training with Batch Normalization
- Adjust the learning rate to ease model collapse
- ……

arXiv:1609.05158 arXiv:1609.04468
https://github.com/soumith/ganhacks

Practice

—— How to code the first line?

GAN toy

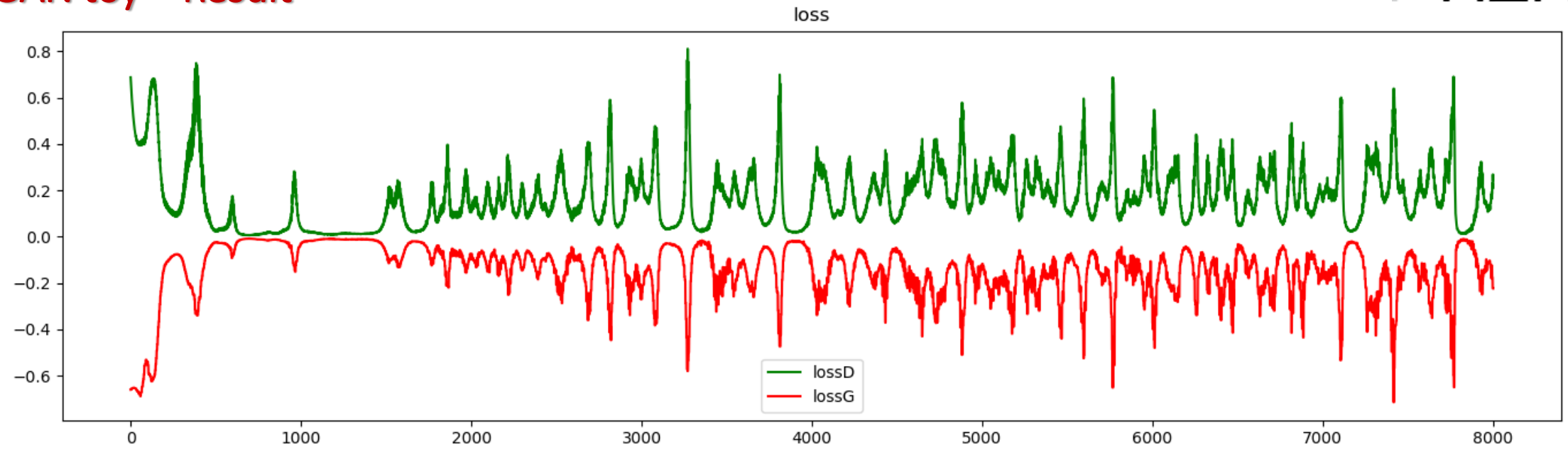
https://github.com/pp-jiang/GANs4Joint_GlueX_EIC_PANDA_ML_Workshop.git

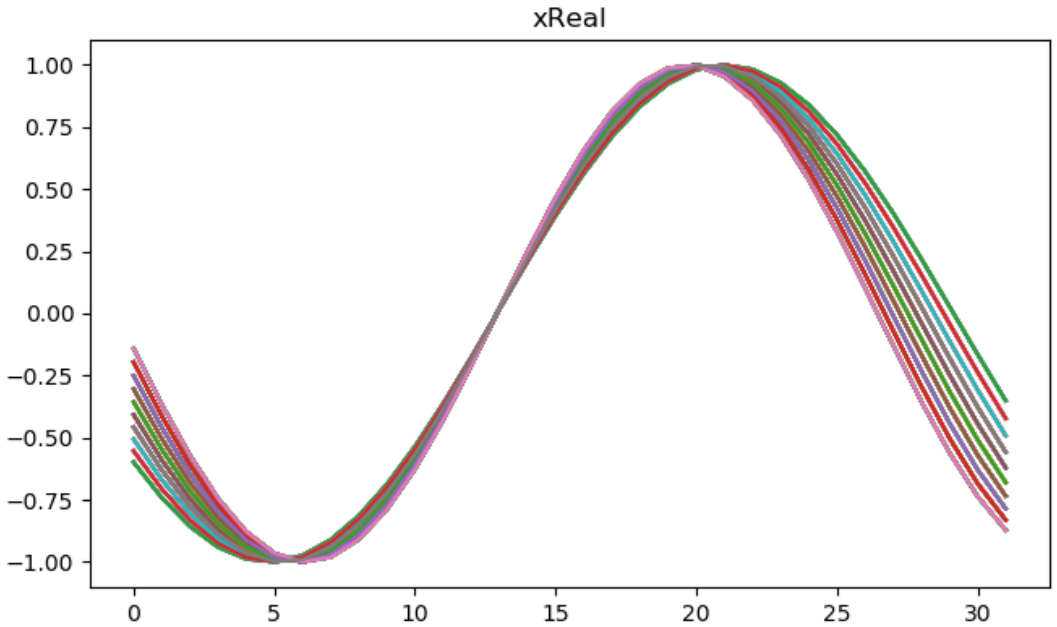
```
import numpy as np; import matplotlib.pyplot as plt

import torch;from torch.autograd import Variable; import torch.nn as nn; import torch.nn.functional as F

Tensor = torch.FloatTensor

numSample=10; numEpoch=10000; num_DG=3; dimSample = 32; dimLatent=10; batchSize=128; bceLoss = torch.nn.BCELoss()


G = nn.Sequential(nn.Linear(dimLatent, 128),nn.LeakyReLU(0.2, inplace=True),nn.Linear(128, 256),nn.LeakyReLU(0.2, inplace=True),

  nn.Linear(256, 512),nn.LeakyReLU(0.2, inplace=True),nn.Linear(512, 1024),nn.LeakyReLU(0.2, inplace=True),nn.Linear(1024, dimSample),nn.Tanh())

D = nn.Sequential(nn.Linear(dimSample, 512),nn.LeakyReLU(0.2, inplace=True),nn.Linear(512, 256),nn.LeakyReLU(0.2, inplace=True),nn.Linear(256, 1),nn.Sigmoid())

dataSet=Variable(torch.sin(torch.mm(torch.linspace(1,1.2,numSample).unsqueeze(1),torch.linspace(-2.5,3.5,dimSample).unsqueeze(0))),requires_grad=False)


optimG = torch.optim.Adam(G.parameters(),lr=0.00001); optimD = torch.optim.SGD(D.parameters(),lr=0.001)  #optimD = torch.optim.Adam(D.parameters(),lr=0.00001)


lossDList=[]; lossGList=[]
```

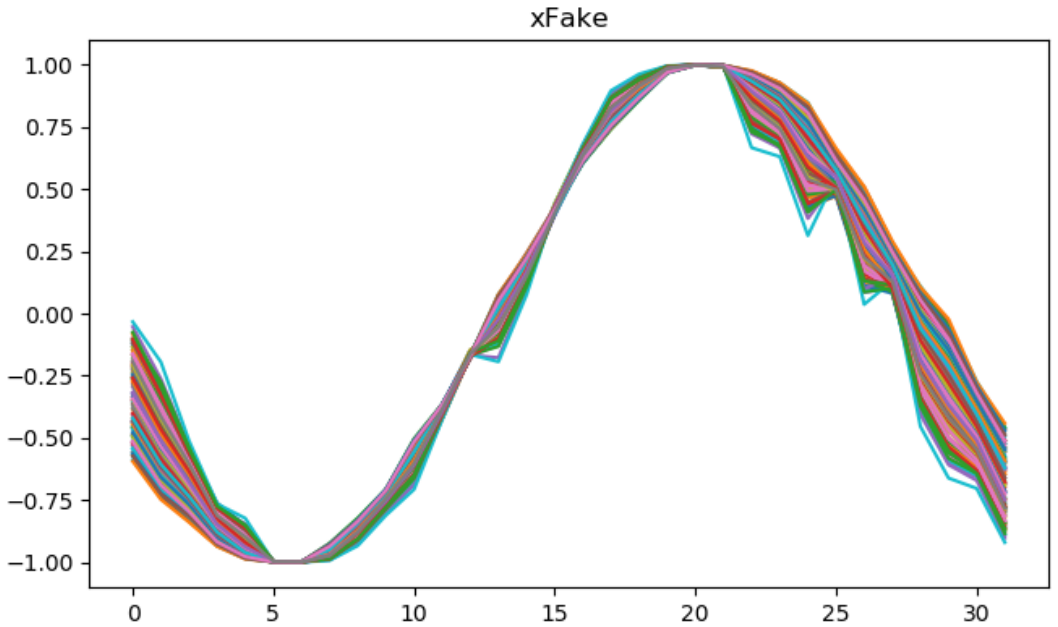
GAN toy - Training

```
for epoch in range(numEpoch):
    xReal=dataSet[torch.randint(0,numSample,(batchSize,)),:]   # xReal
    z = torch.normal(0, 1, size=(batchSize, dimLatent), requires_grad=False, out=None)   # latent noise
    yReal= Variable(Tensor(batchSize, 1).fill_(1.0), requires_grad=False)   # real labels
    yFake= Variable(Tensor(batchSize, 1).fill_(0.0), requires_grad=False)   # fake labels
    ## Train Discriminator
    optimD.zero_grad()
    lossD=(bceLoss(D(G(z)),yFake)+bceLoss(D(xReal),yReal))/2.    # lossG = BCE Loss
    lossD.backward(); optimD.step()
    ## Train Generator
    if epoch%num_DG==0:   # Repeat num_DG times training D and then run once training G
        optimG.zero_grad()
        lossG=-bceLoss(D(G(z)),yFake)    # lossG = -BCE Loss
        lossG.backward(); optimG.step()

    lossDListot.append(lossD.item()); lossGList.append(lossG.item())
```

GAN toy – Result







Thank you for your attention!