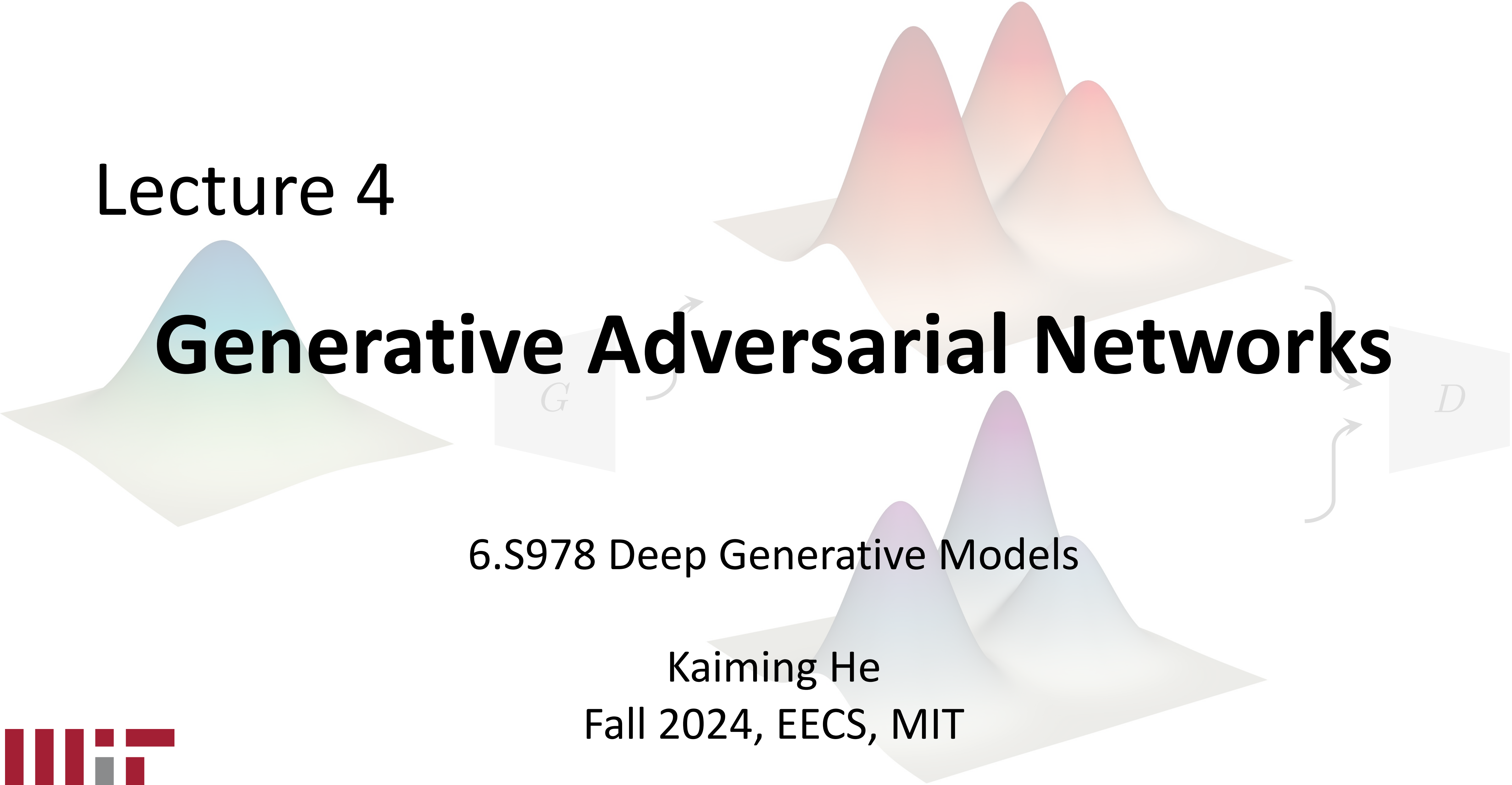


Lecture 4

Generative Adversarial Networks

6.S978 Deep Generative Models

Kaiming He
Fall 2024, EECS, MIT

Overview

- Generative Adversarial Networks (GAN)

- Wasserstein GAN (W-GAN)

- Adversary as a Loss Function

Generative Adversarial Networks (GAN)

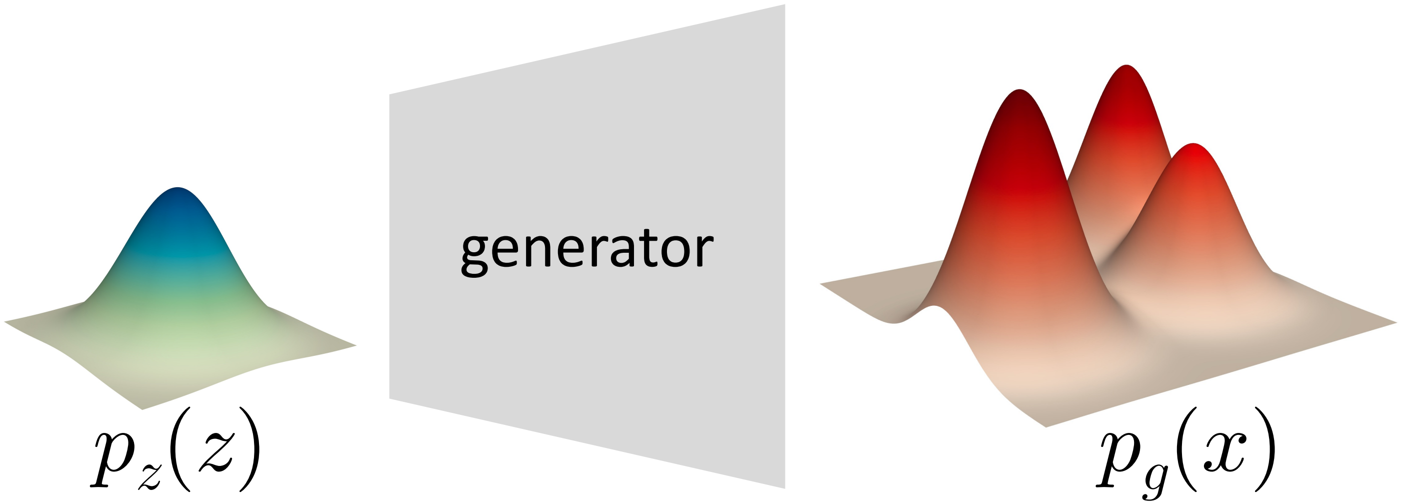
Introduction

- “**Generative**”
  - “Discriminative” was dominant back then
- “**Adversarial**”
  - Generative models w/ discriminative models
  - Min-max process
- “**Networks**”
  - SGD + backprop for problem solving

Recap: Latent Variable Models

Represent a distribution by a neural network:

- z - latent variables
- x - observed variables

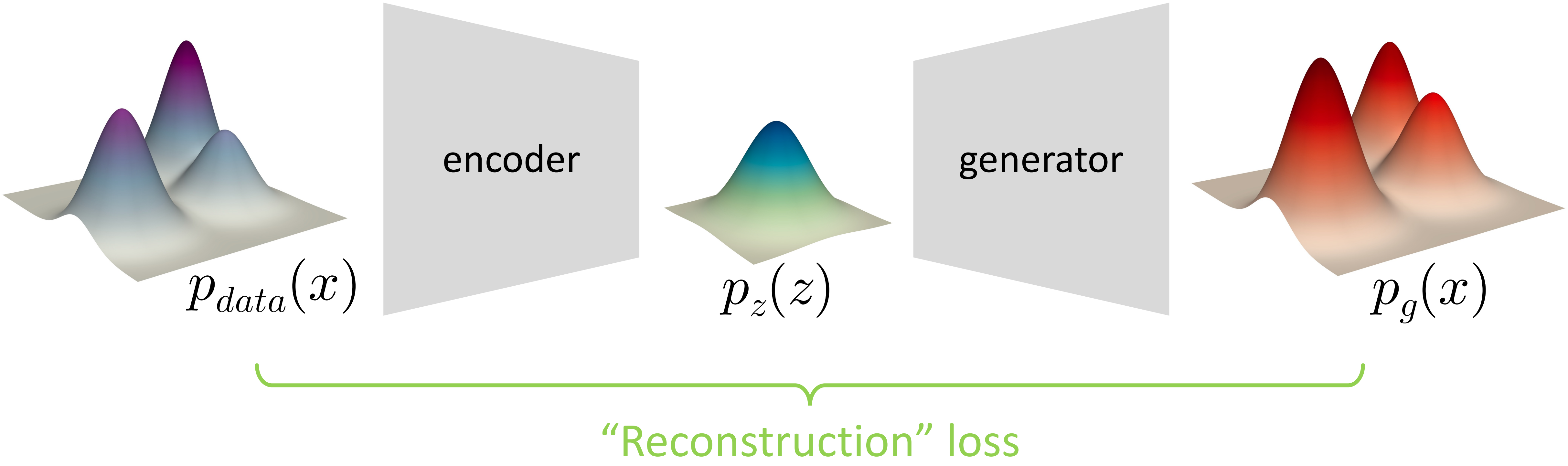


See Lecture 2, Variational Autoencoder: https://mit-6s978.github.io/assets/pdfs/lec2_vae.pdf

Recap: Variational Autoencoder (VAE)

Autoencoding distributions:

"Encoding" data distribution p_{data} into latent distribution p_z



See Lecture 2, Variational Autoencoder: https://mit-6s978.github.io/assets/pdfs/lec2_vae.pdf

What’s the implication of a “*reconstruction*” loss?

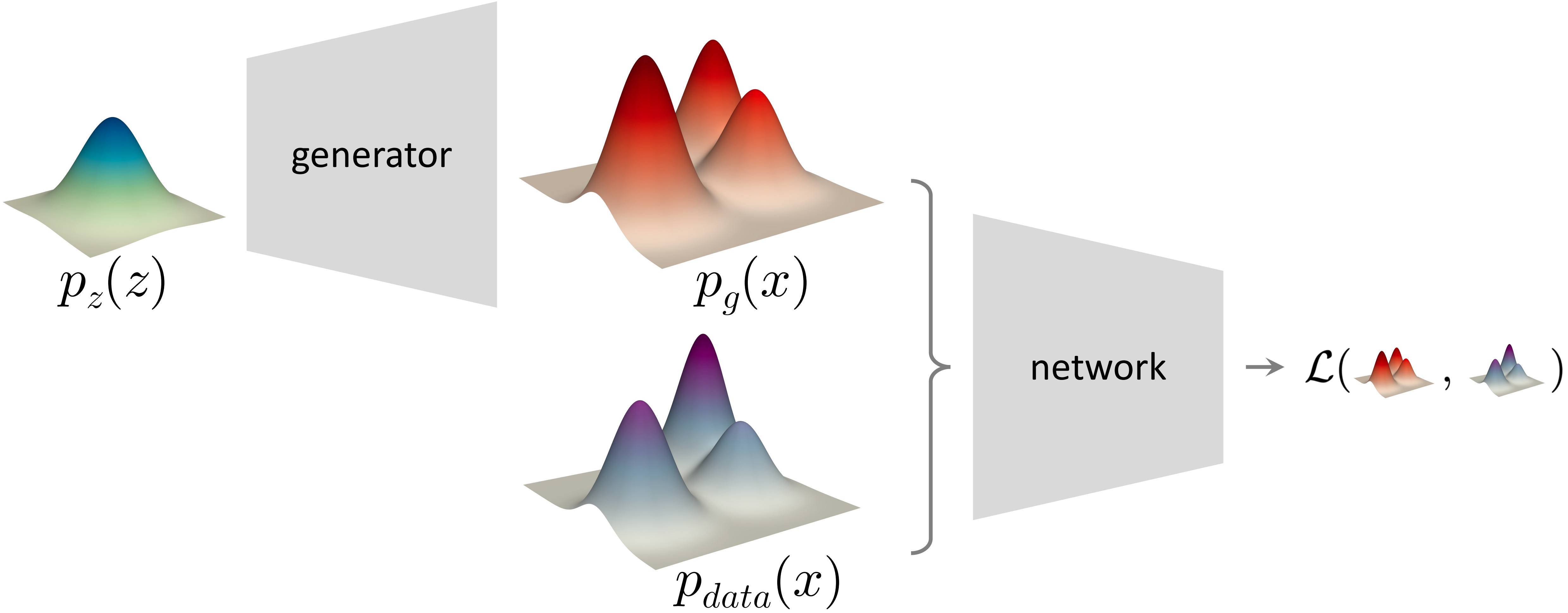
- Elements (e.g., pixels) are **independently** distributed
- Each element follows a **simple** distribution (Gaussian/Bernoulli/...)

Assumptions are too strict for **high-dim** variables

Can we measure the distribution difference in another way?

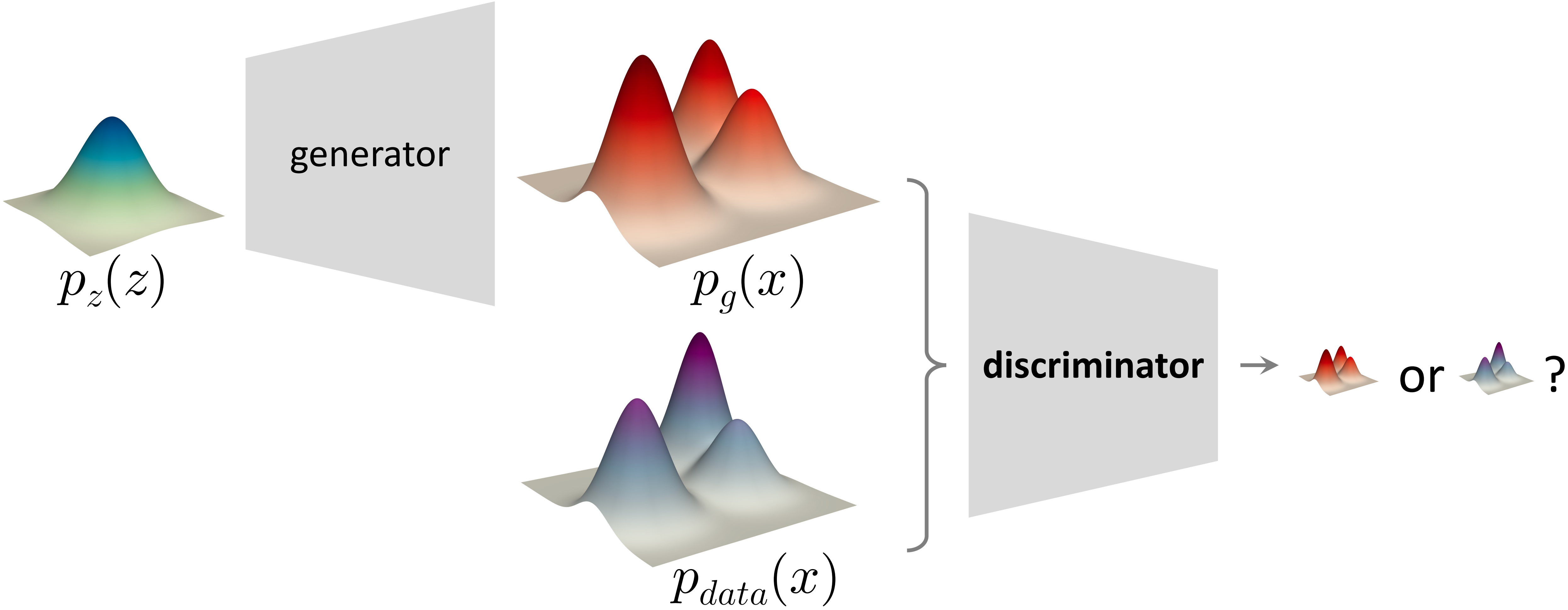
Generative Adversarial Networks

Representing **distribution difference** by a neural network



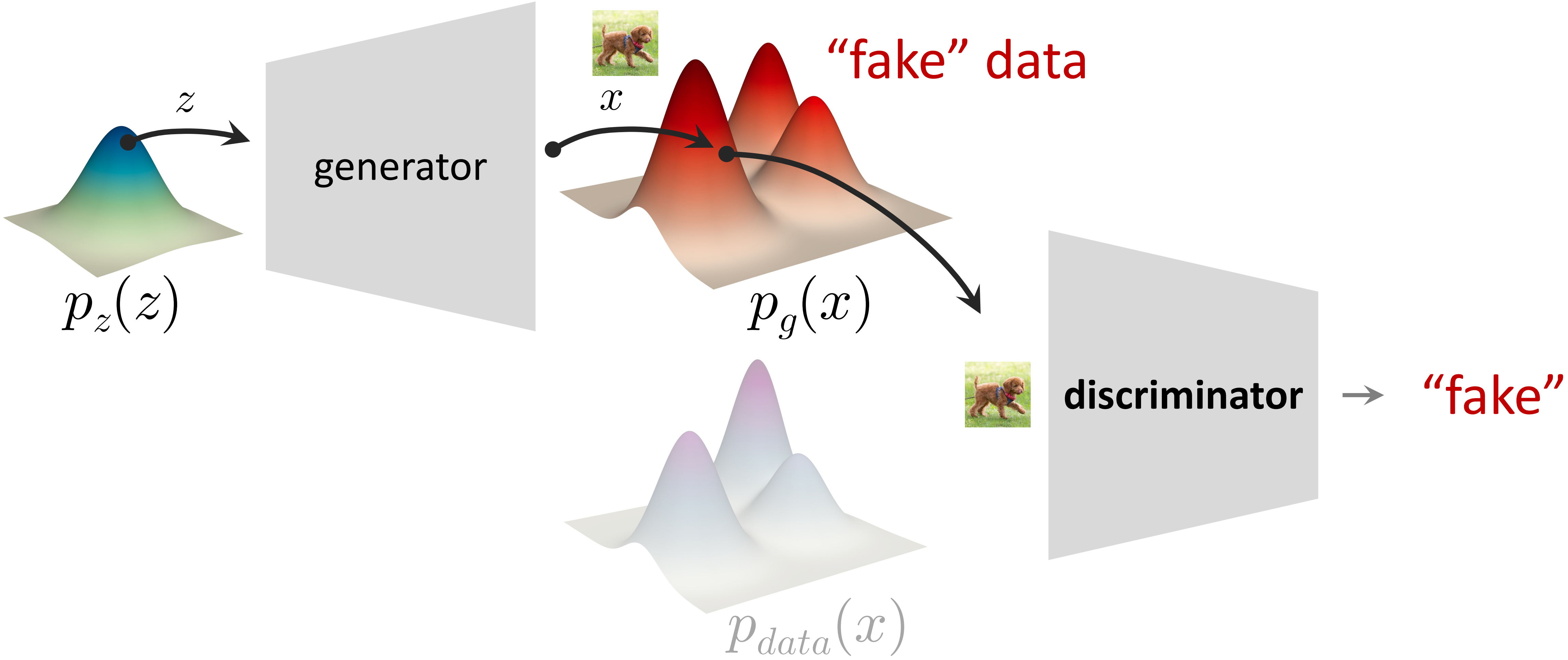
Generative Adversarial Networks

Representing **distribution difference** by a neural network



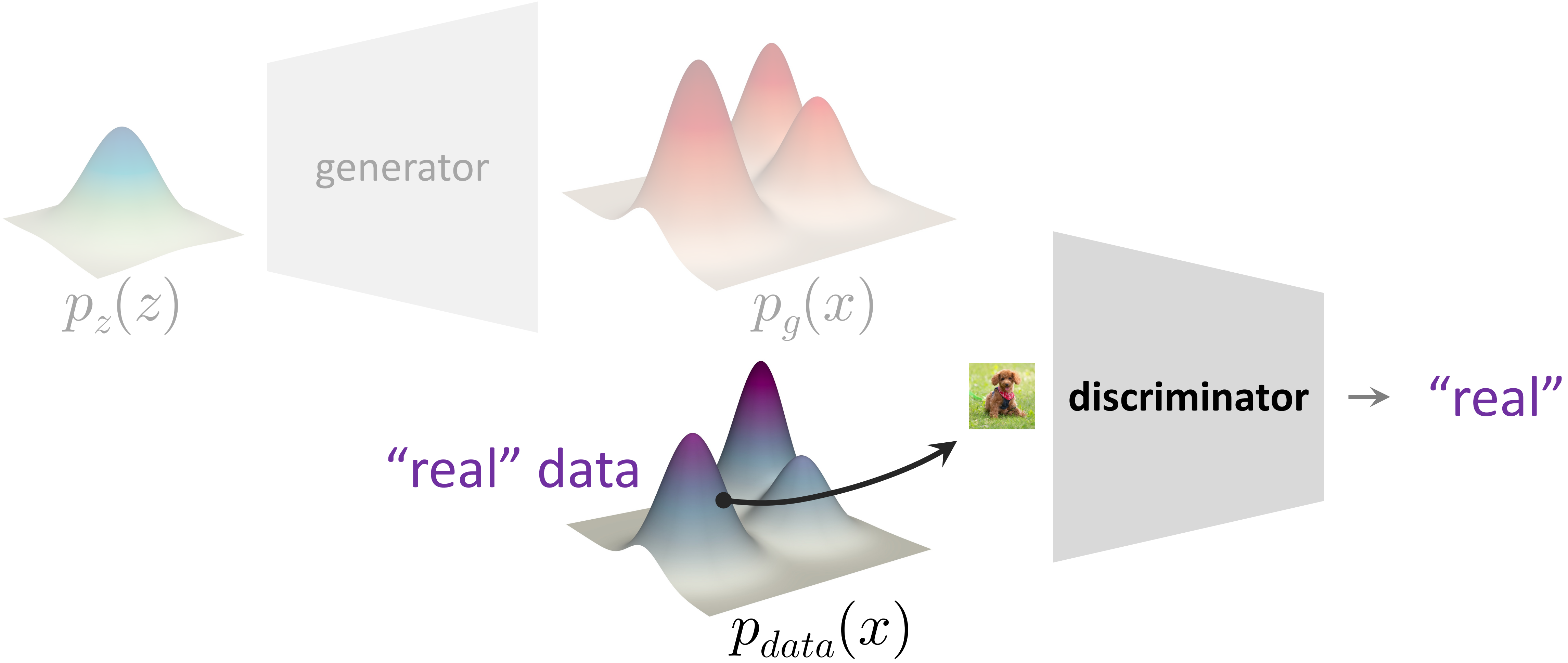
Generative Adversarial Networks

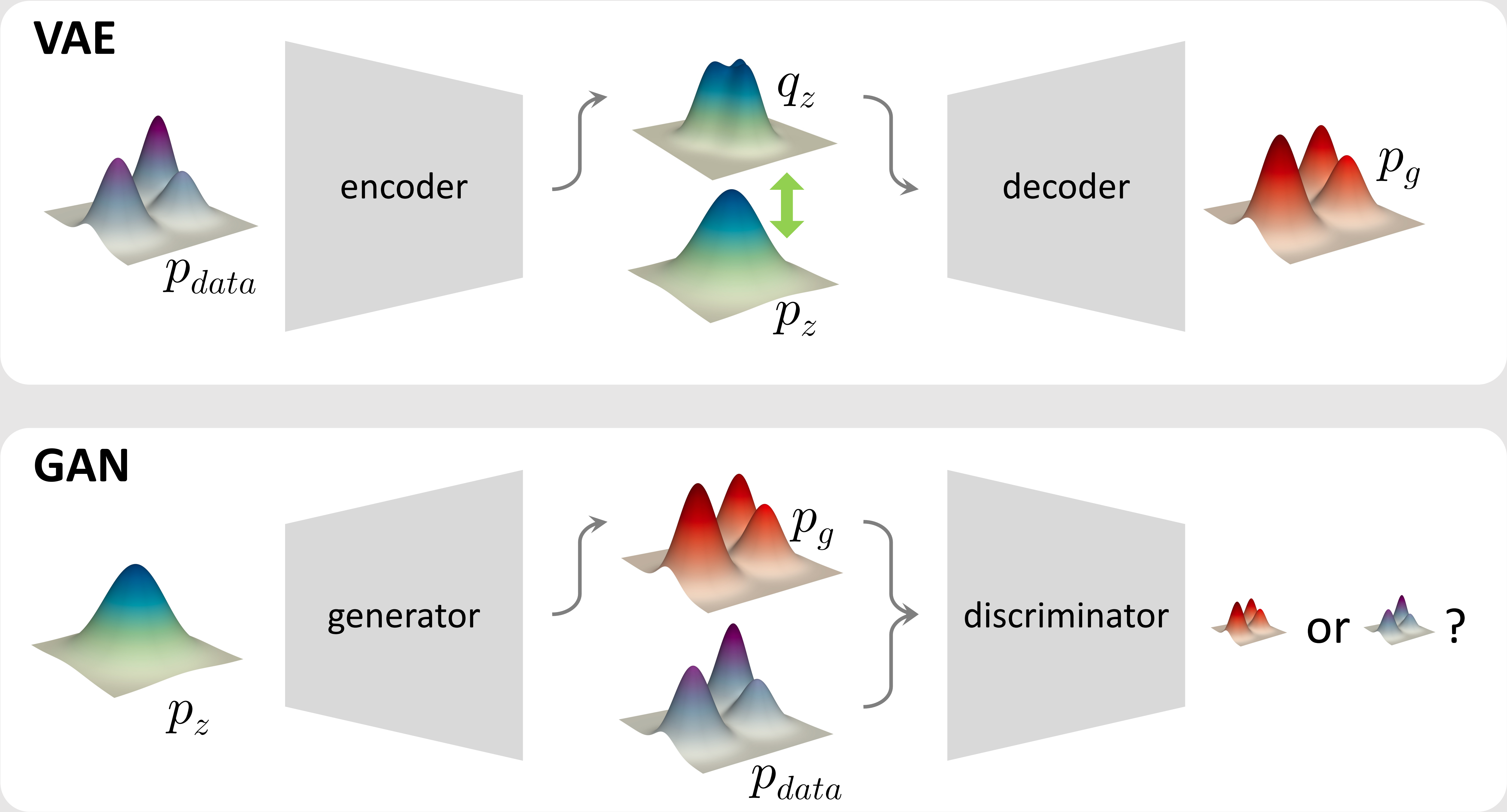
Representing **distribution difference** by a neural network

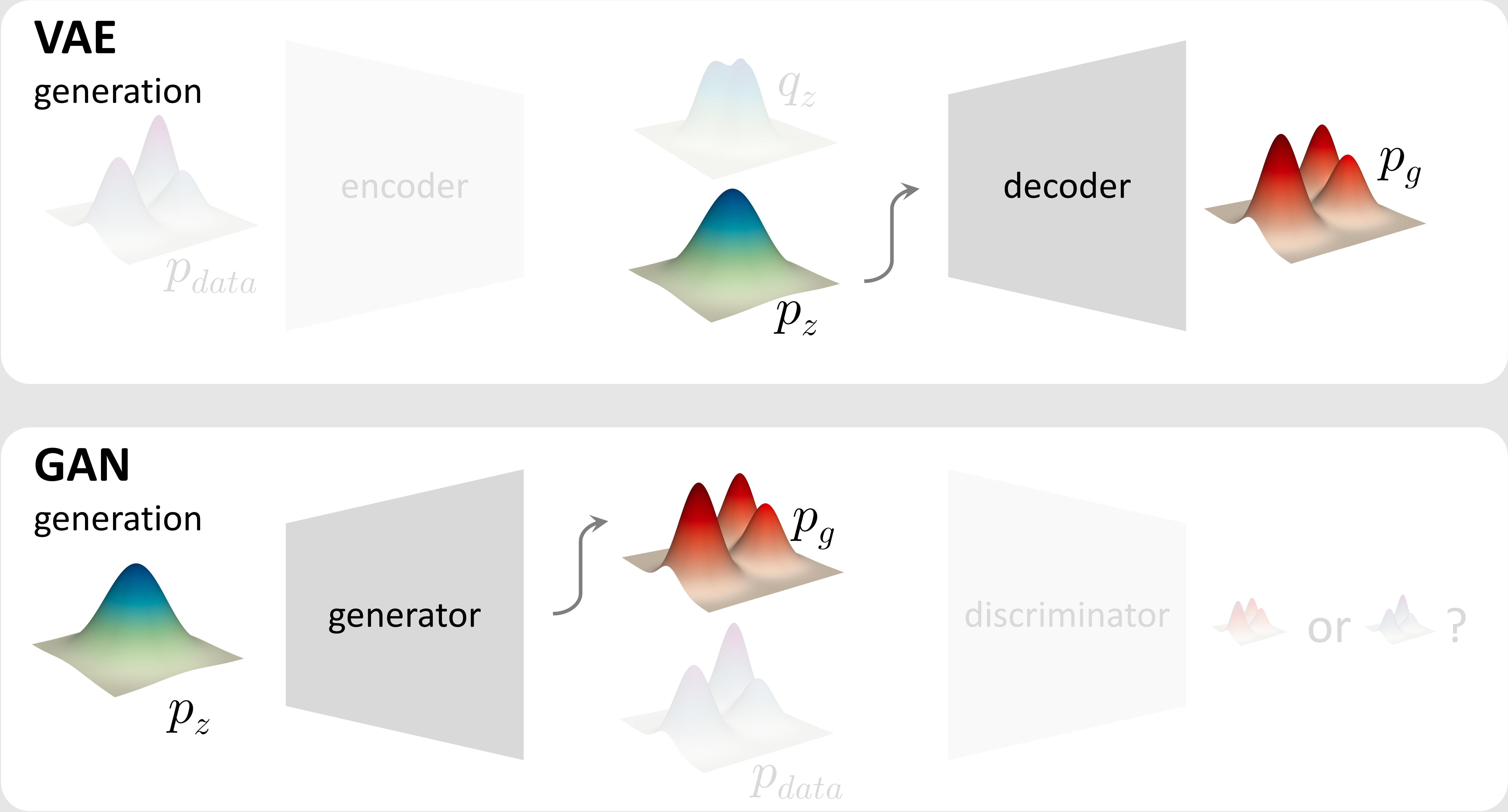


Generative Adversarial Networks

Representing **distribution difference** by a neural network




VAE
p_{data}
encoder
q_z
p_z
decoder
p_g
GAN
p_z
generator
p_g
p_{data}
discriminator
or
?

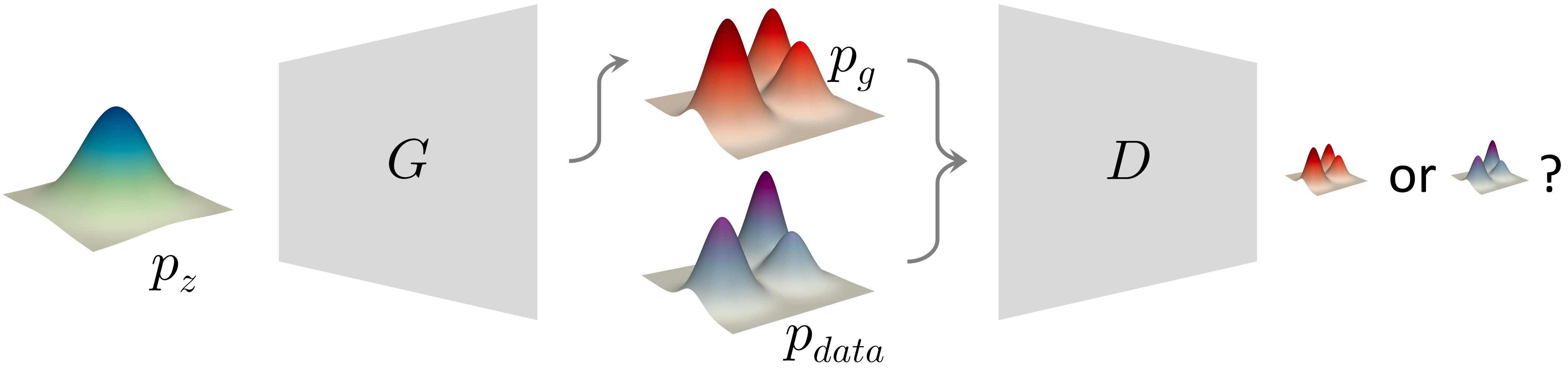

VAE
generation
p_{data}
encoder
q_z
p_z
decoder
p_g
GAN
generation
p_z
generator
p_g
p_{data}
discriminator
or
?

Adversarial Objective

$$\min_{G} \max_{D} \mathcal{L}(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

min-max process

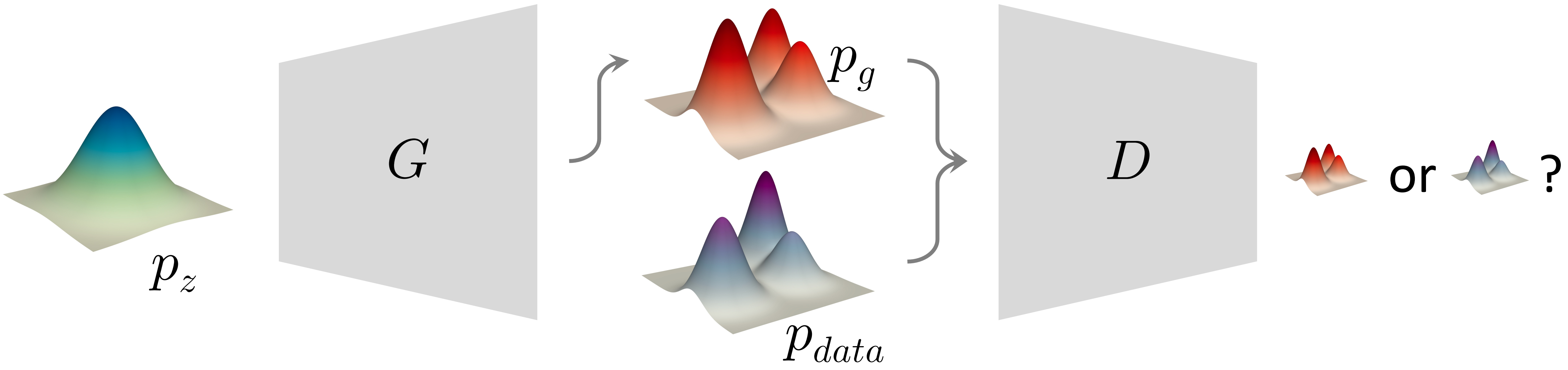
(vs. EM's max-max process)



Adversarial Objective: D-step

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

D-step: fix G, optimize D



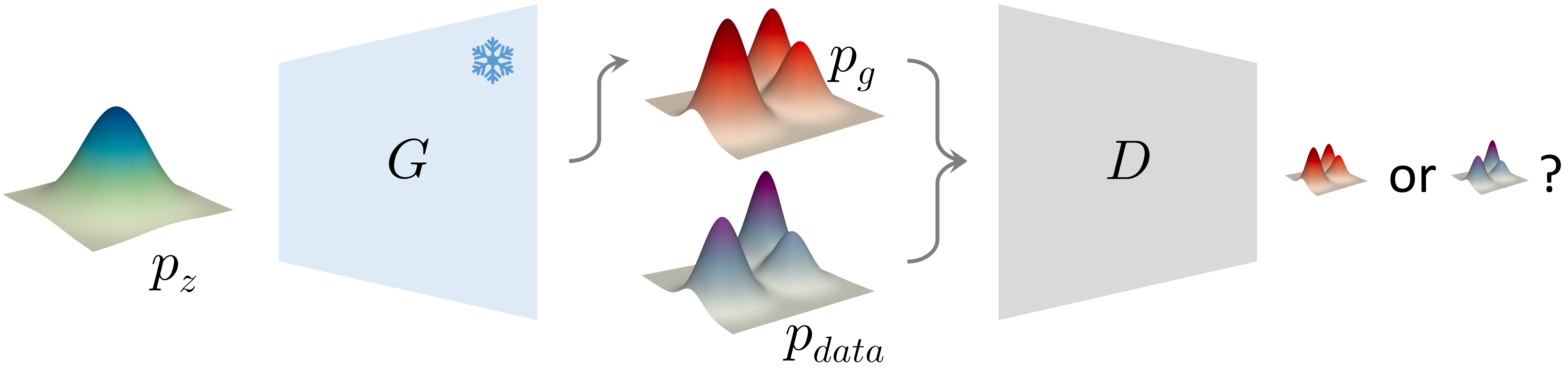
Adversarial Objective: D-step

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

push to 1 push to 0

D-step: fix G, optimize D

- D to classify real or fake
- binary logistic regression (sigmoid + cross-entropy)



Adversarial Objective: D-step

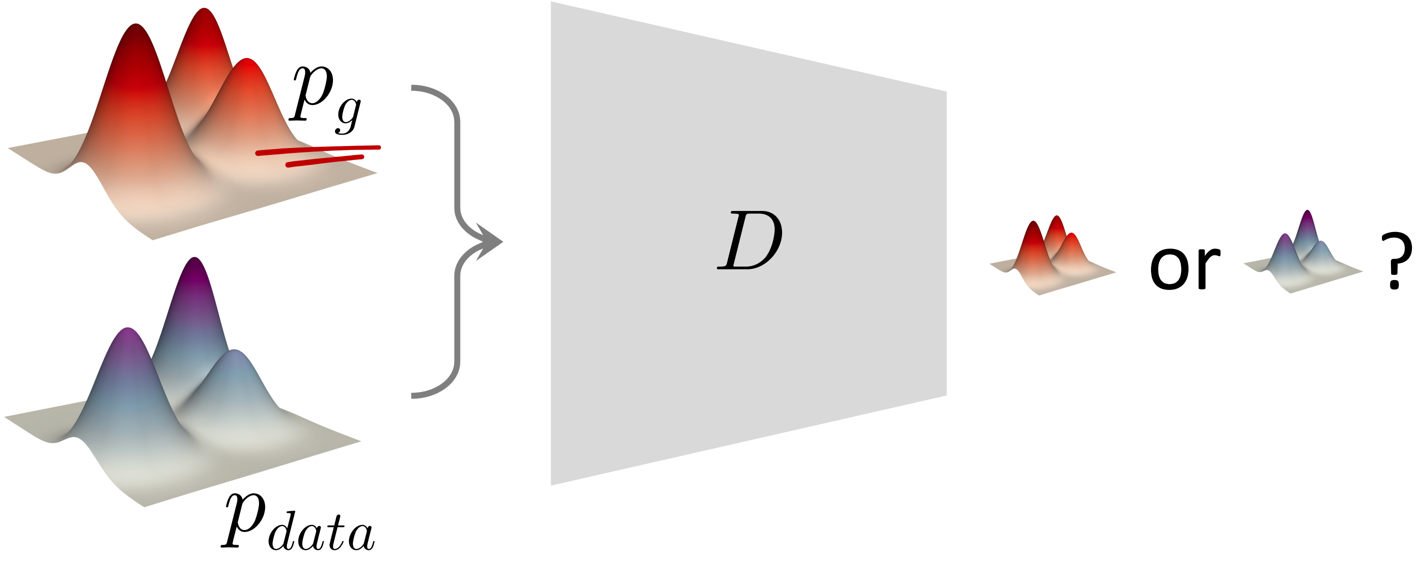
$$\max_D \mathcal{L}(D) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{x\sim p_g}[\log(1 - D(x))]$$

push to 1

push to 0

D-step: fix G, optimize D

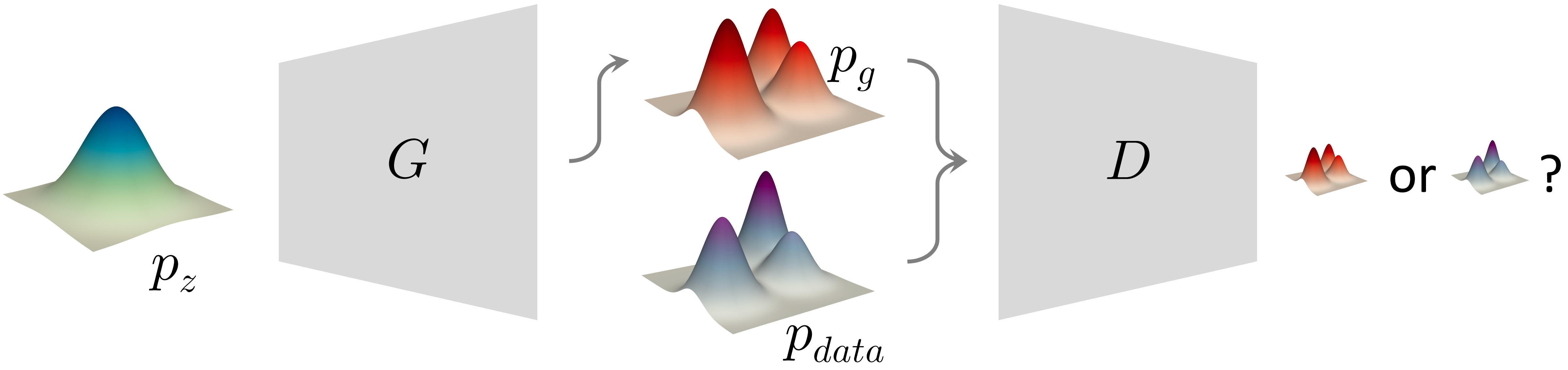
- D to classify real or fake
- binary logistic regression (sigmoid + cross-entropy)



Adversarial Objective: G-step

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

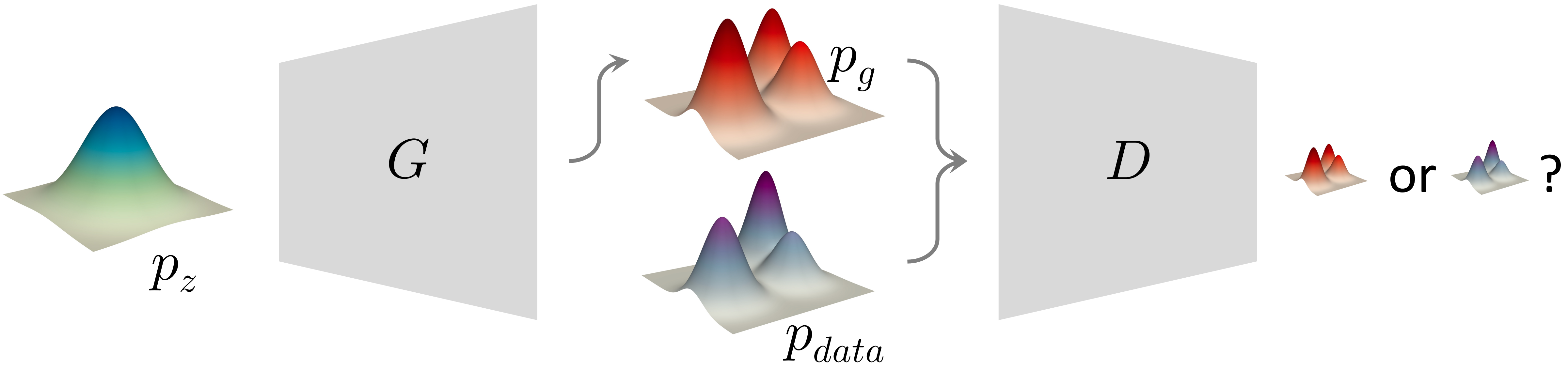
G-step: fix D, optimize G



Adversarial Objective: G-step

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

G-step: fix D, optimize G



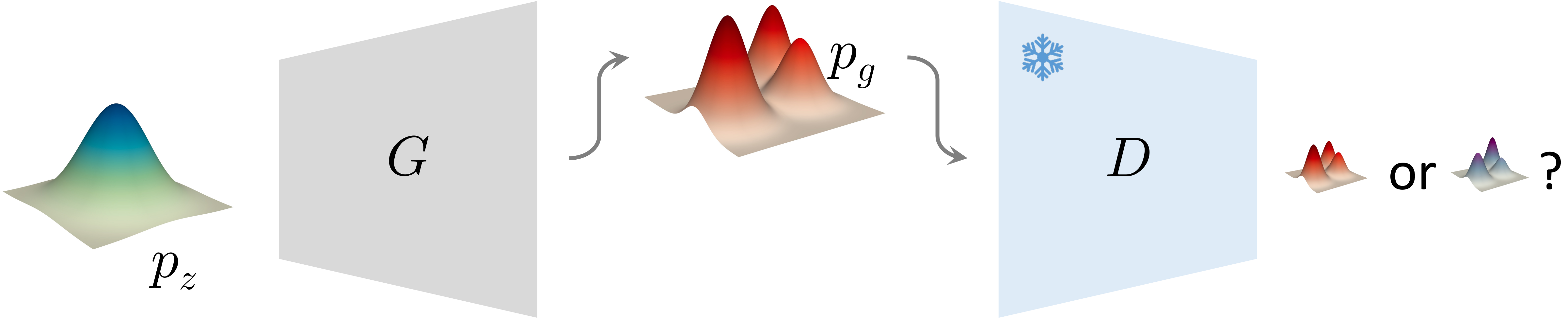
Adversarial Objective: G-step

$$\min_G \mathcal{L}(G) = \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

push to 1

G-step: fix D, optimize G

- generate fake data such that D classifies it as “real”
- G to “confuse” D



Adversarial Objective: G-step

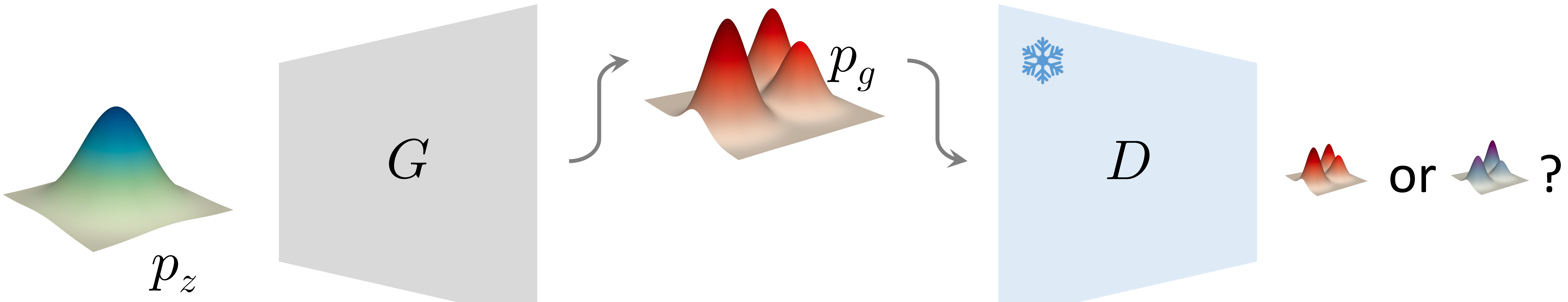
a "flip" trick:

$$\overset{\max}{\cancel{\min}}_{G} \mathcal{L}(G) = \mathbb{E}_{z\sim p_z}[\log(\cancel{1-}D(G(z)))]$$

push to 1

G-step: fix D, optimize G

- generate fake data such that D classifies it as "real"
- G to "confuse" D



Adversarial Objective: G-step

a "flip" trick:

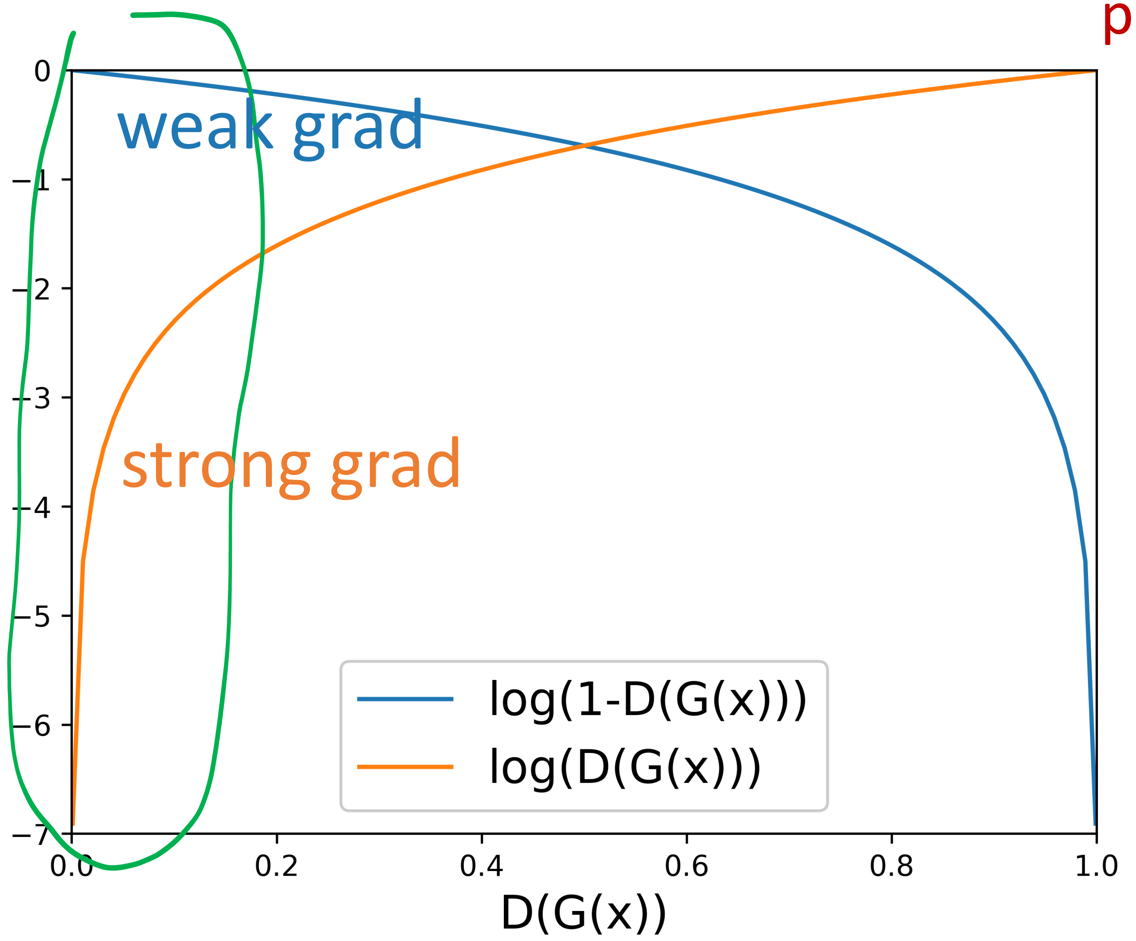
max

$$\min_{G} \mathcal{L}(G) = \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$

push to 1

Early in training:

- G is poor
- $D(G)$ is near 0



GAN algorithm annotated

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

minibatch SGD

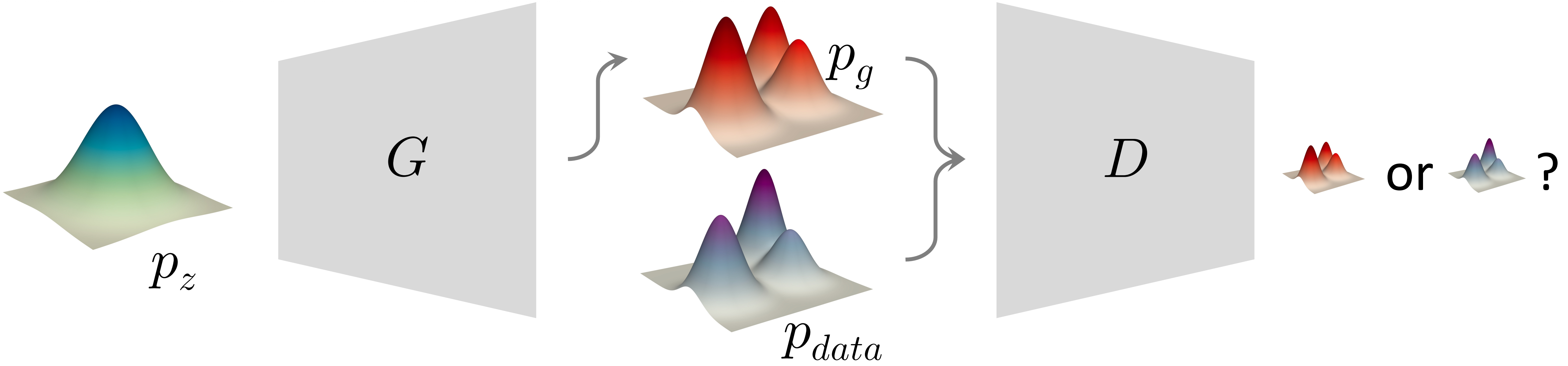
for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



GAN algorithm annotated

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

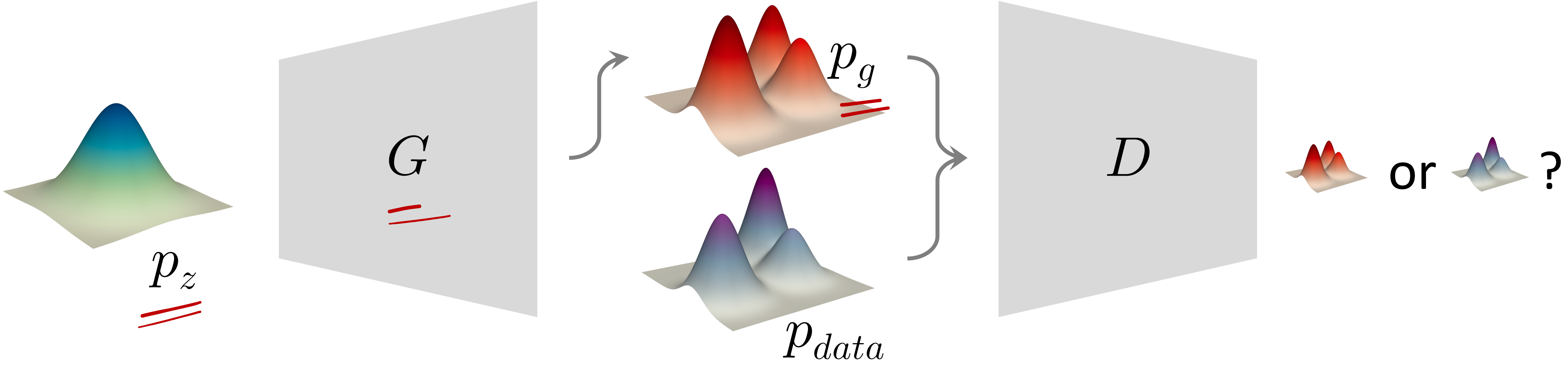
  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



GAN algorithm annotated

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

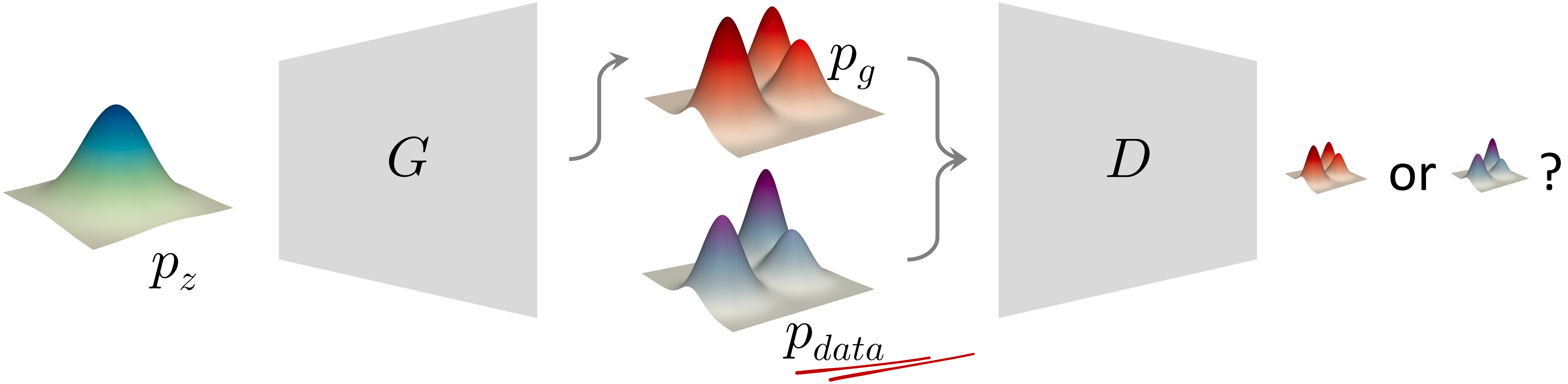
for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



GAN algorithm annotated

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

D-step

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

gradient **ascend**
(maximize)

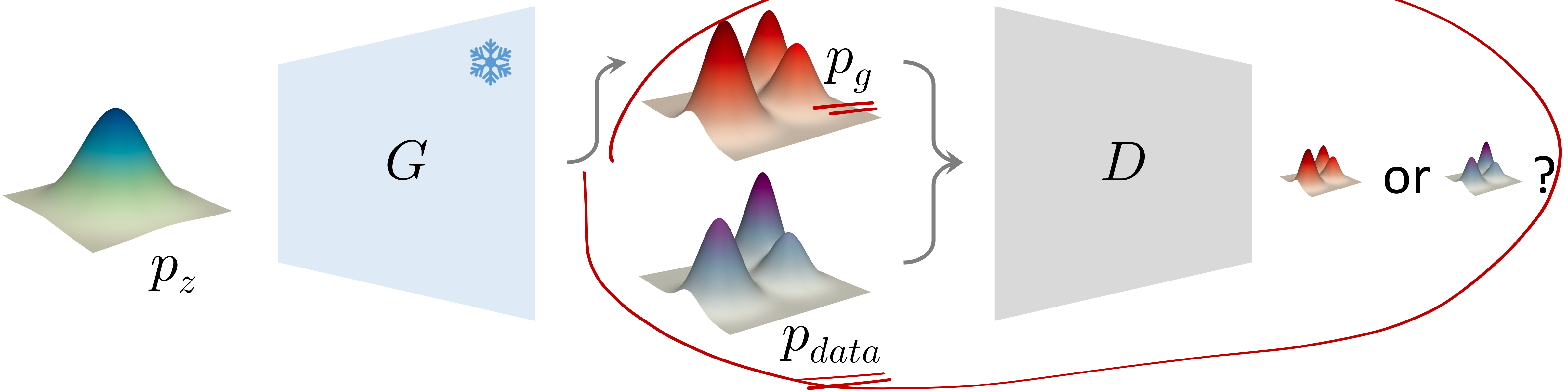
  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



GAN algorithm annotated

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

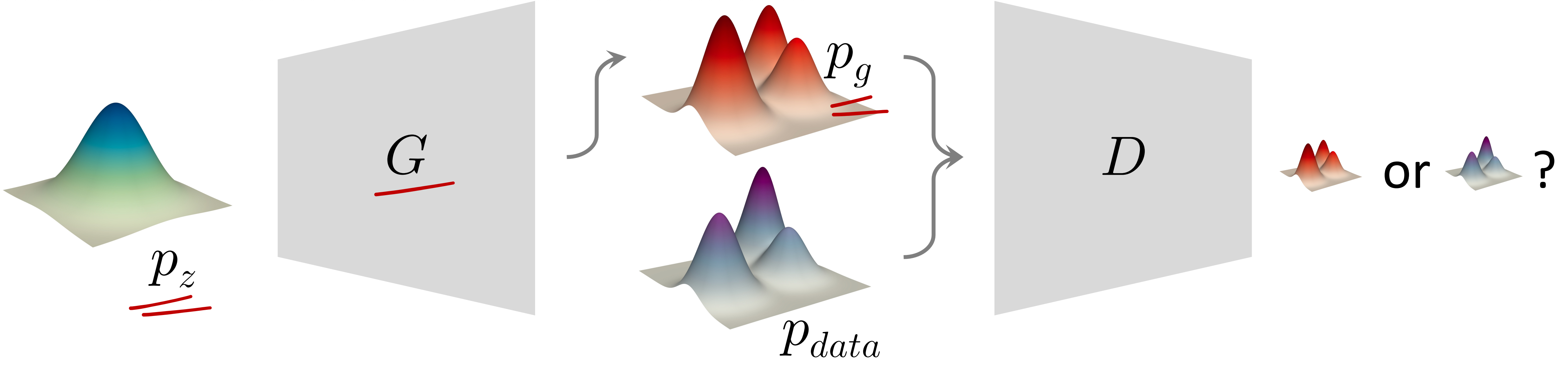
  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



GAN algorithm annotated

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

G-step

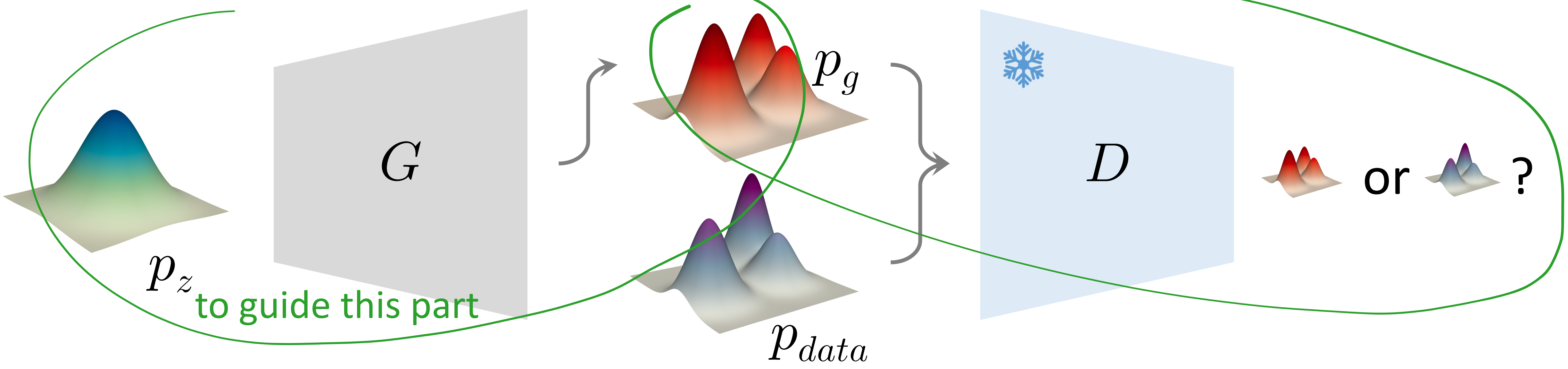
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

gradient **descend** (minimize)

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

a parameterized loss function



GAN algorithm annotated

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

  • Update the discriminator by ascending its stochastic gradient:

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

  end for

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

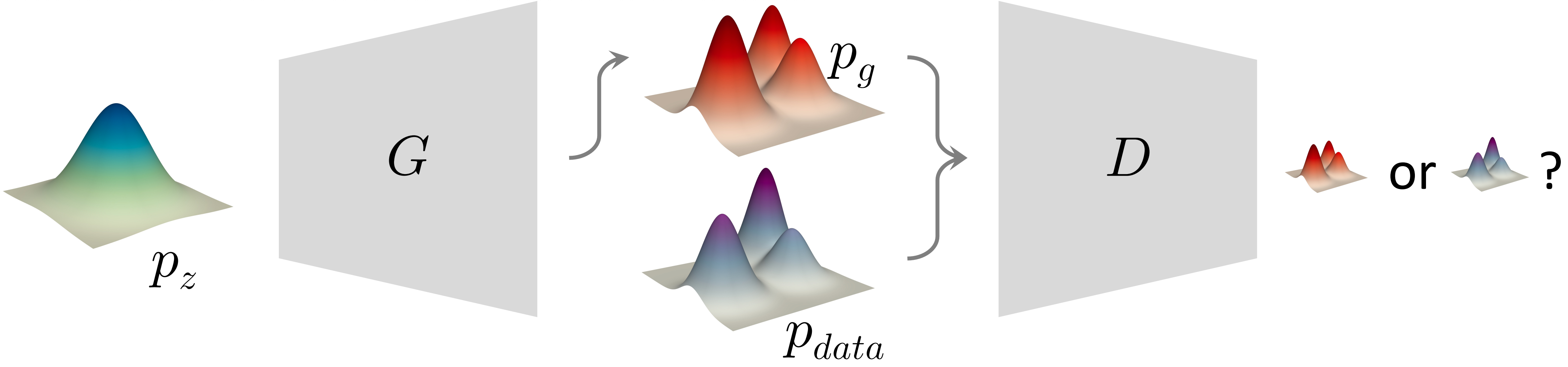
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

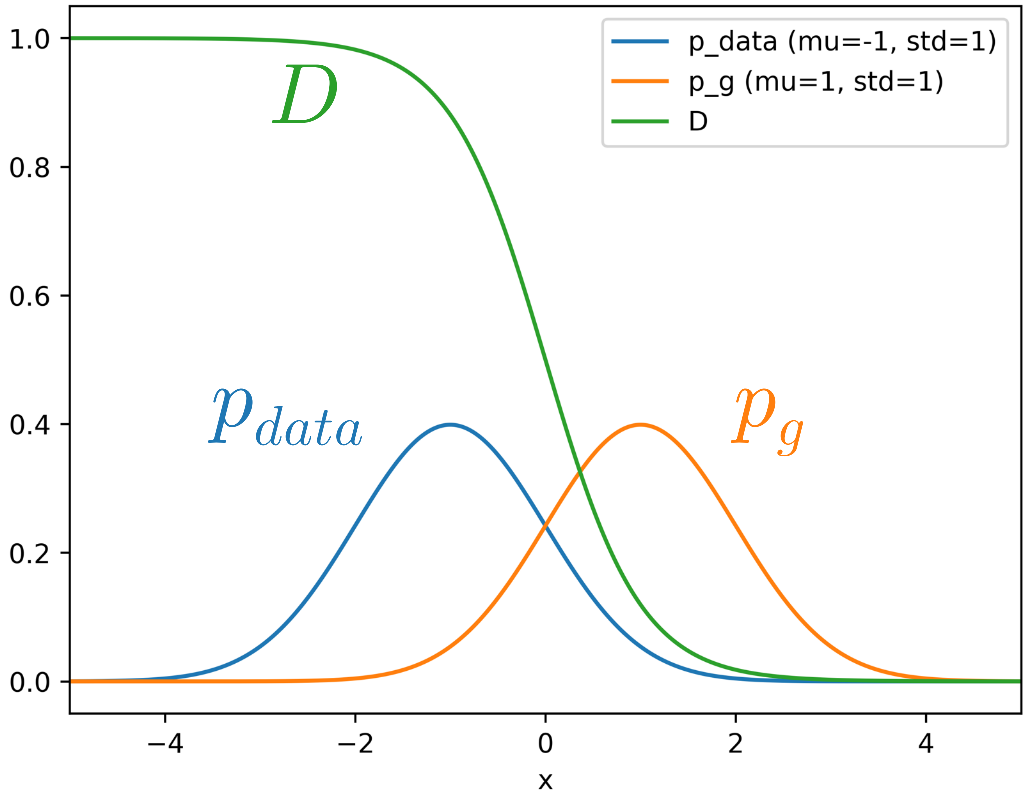
iterating min-max



Theoretical Results

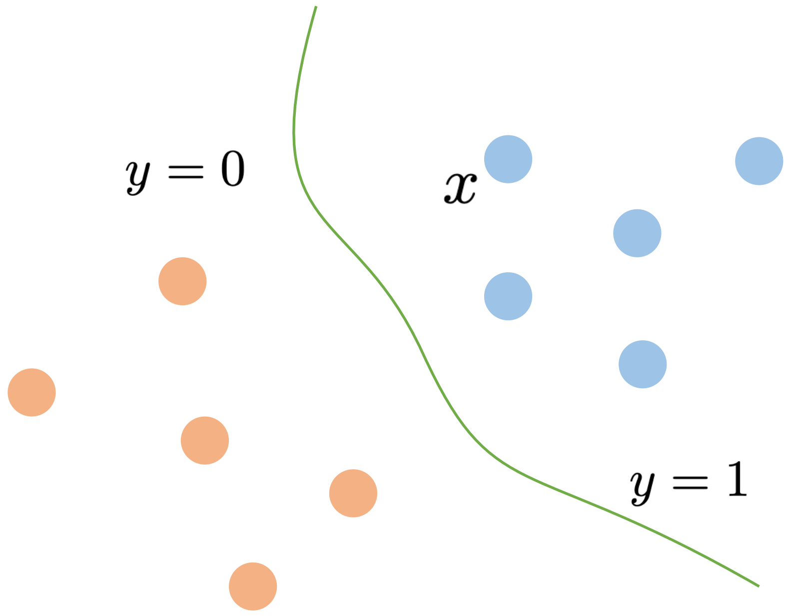
1. For any given G, the optimal D is:

$$D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$$

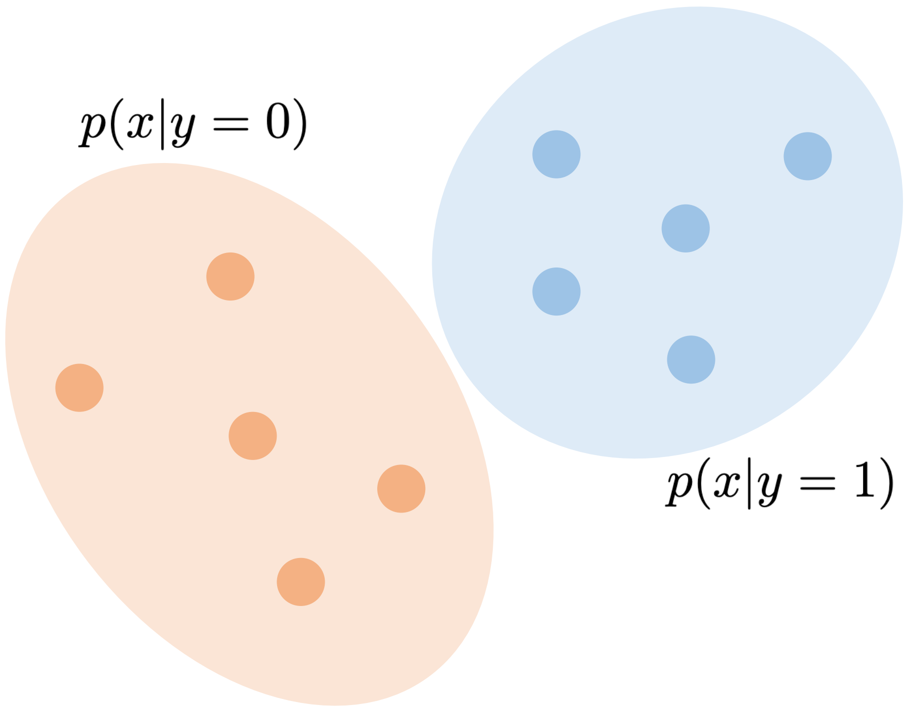


Recap (Lec. 1): Discriminative vs. Generative

discriminative



generative



Theoretical Results

2. With the optimal D_G, the objective function is:

$$\mathcal{L}(D^*, G) = 2D_{JS}(p_{\text{data}} \| p_g) - 2\log 2$$

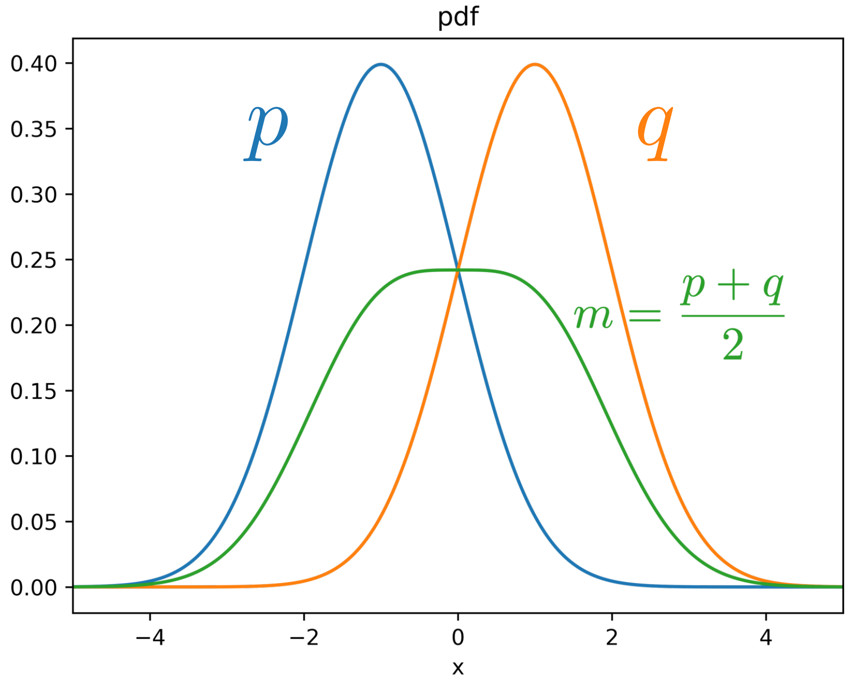
where D_{JS} is Jensen–Shannon divergence

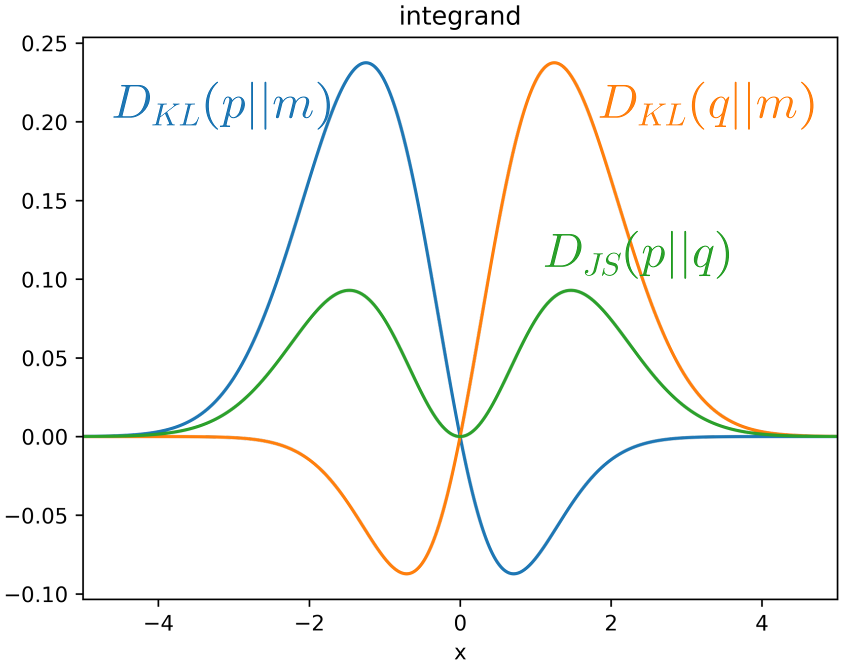
See proof in “Generative Adversarial Nets”, Goodfellow, et al., 2014

Background: Jensen–Shannon divergence

D_{JS}: "total divergence to the average"

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





Background: Jensen–Shannon divergence

D_{JS}: “total divergence to the average”

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

Properties:

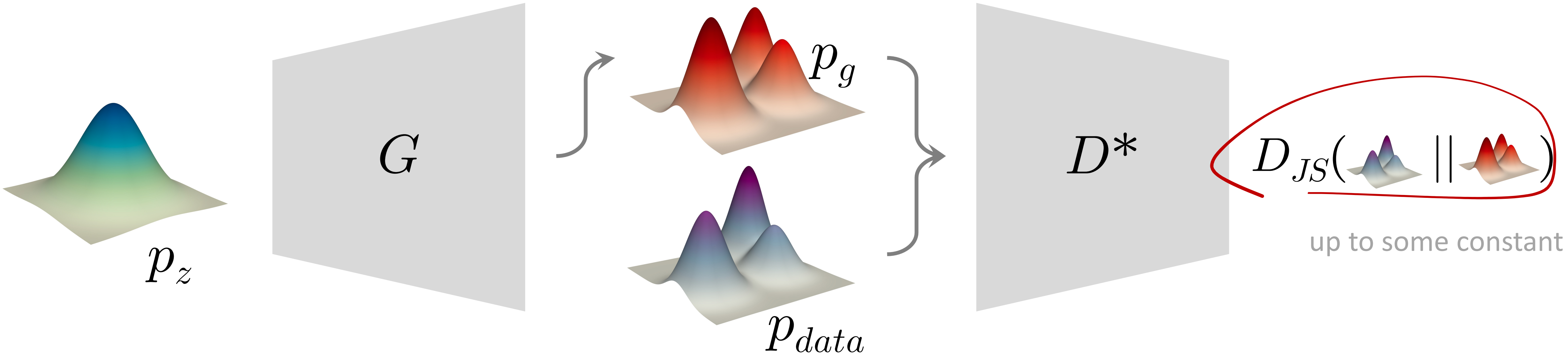
- D_{JS} is symmetric; D_{KL} is not
- D_{JS} is bounded: $[0,\ \log 2]$; D_{KL} is unbounded: $[0,\ \inf)$
- D_{JS} is more stable

Theoretical Results

2. With the optimal D_G, the objective function is:

$$\mathcal{L}(D^*, G) = 2D_{JS}(p_{\text{data}} \| p_g) - 2\log 2$$

GAN optimizes for Jensen–Shannon divergence.



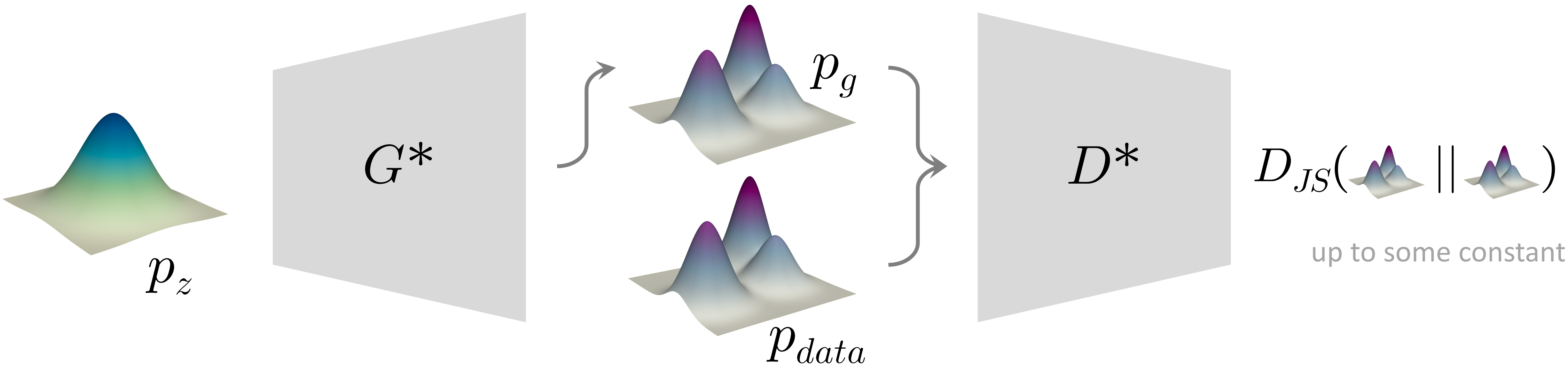
See proof in "Generative Adversarial Nets", Goodfellow, et al., 2014

Theoretical Results

3. Global optimality is achieved at $p_g = p_{data}$

$$\mathcal{L}(D^*, G^*) = 2D_{JS}(p_{\text{data}} \| p_g) - 2\log 2$$

$= 0$



See proof in "Generative Adversarial Nets", Goodfellow, et al., 2014

Theoretical Results: Summary

1. For any given G, the optimal D is:

$$D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$$

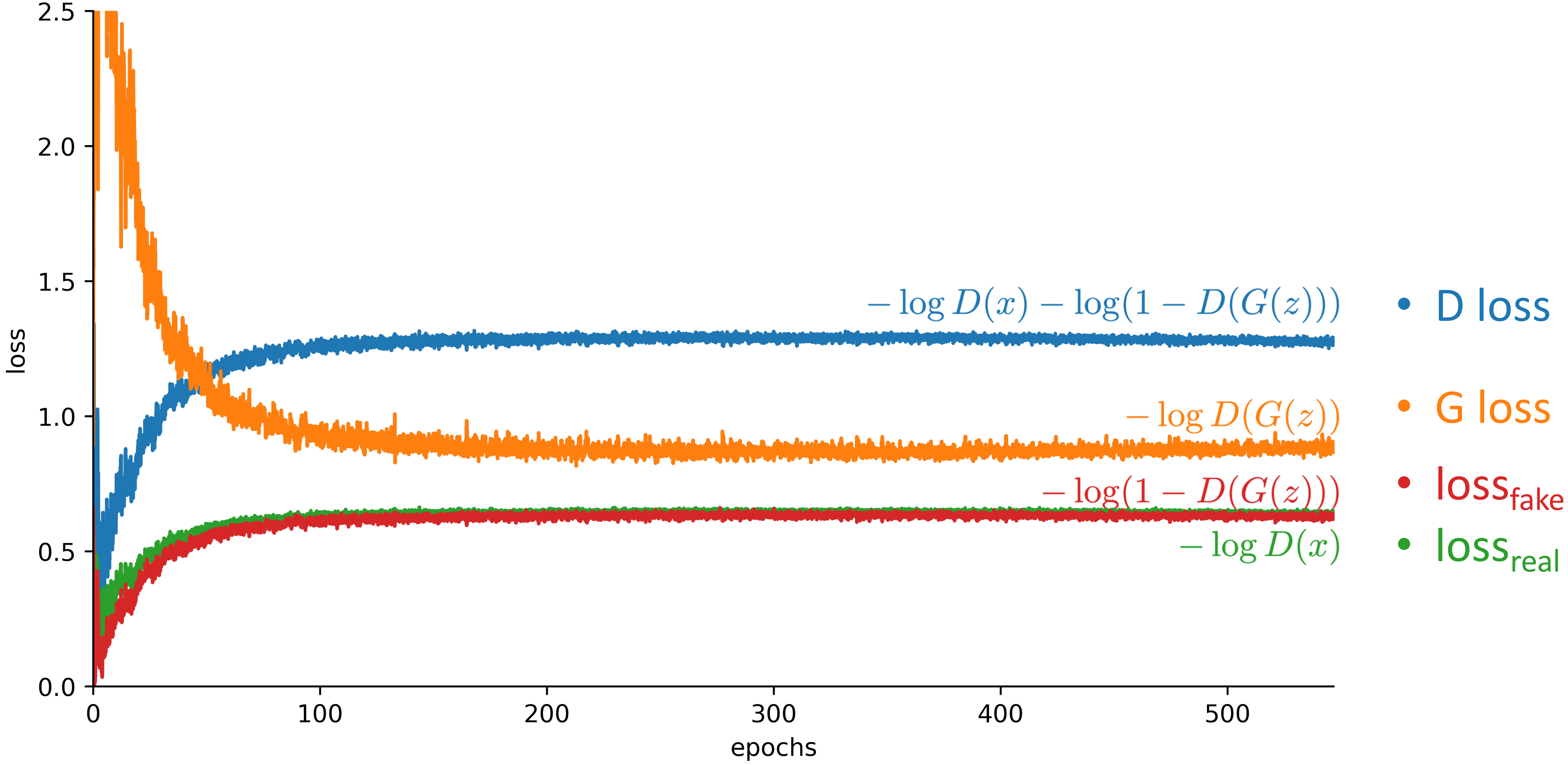
2. With optimal D_G, GAN optimizes for Jensen–Shannon divergence:

$$\mathcal{L}(D^*, G) = 2D_{JS}(p_{\text{data}} \| p_g) - 2\log 2$$

3. Global optimality is achieved at $p_g = p_{data}$

$$\mathcal{L}(D^*, G^*) = -2\log 2$$

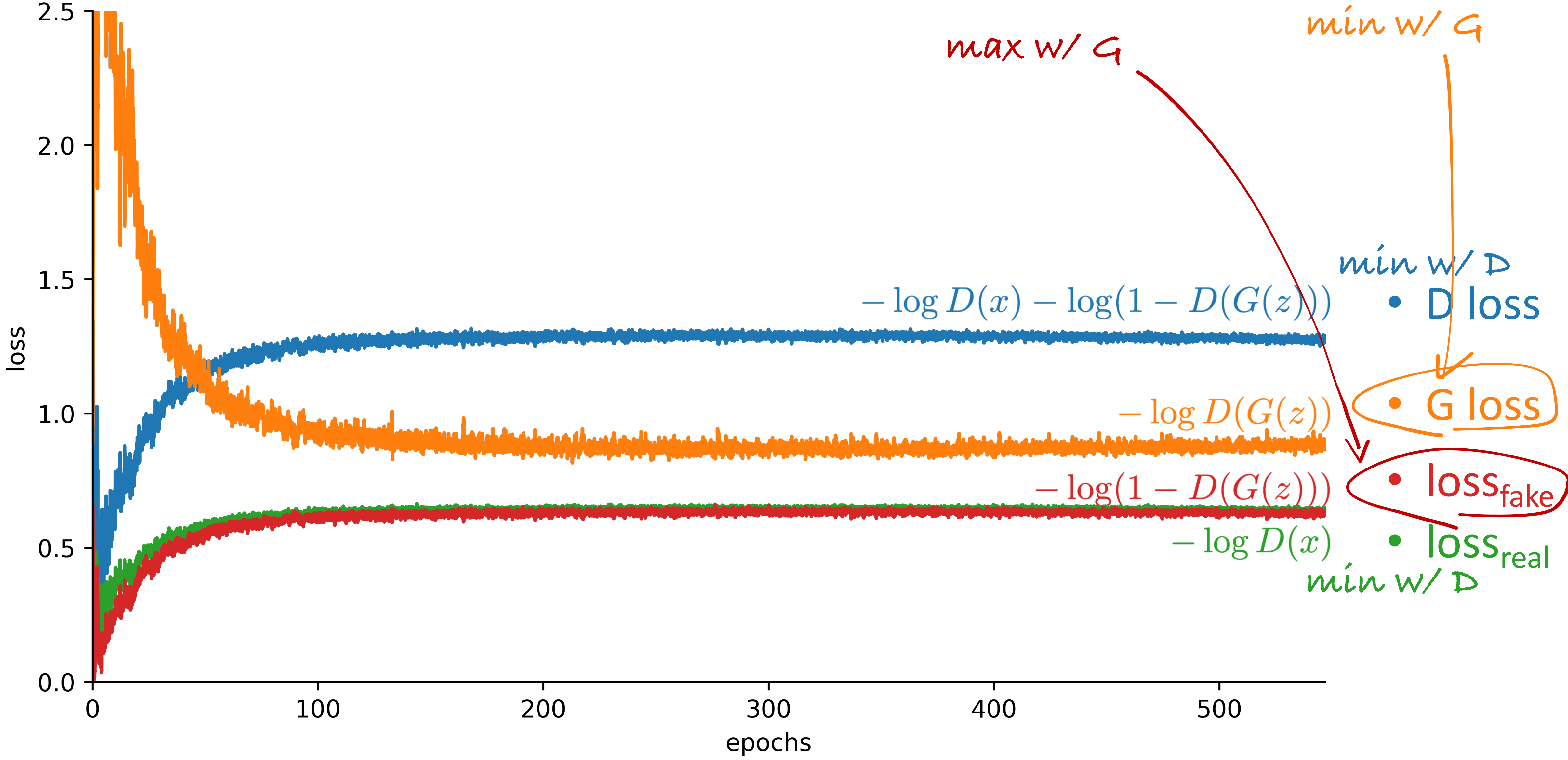
Running example: MNIST



*All objectives are negative of their original form

Code adapted from: https://github.com/prcastro/pytorch-gan/tree/master

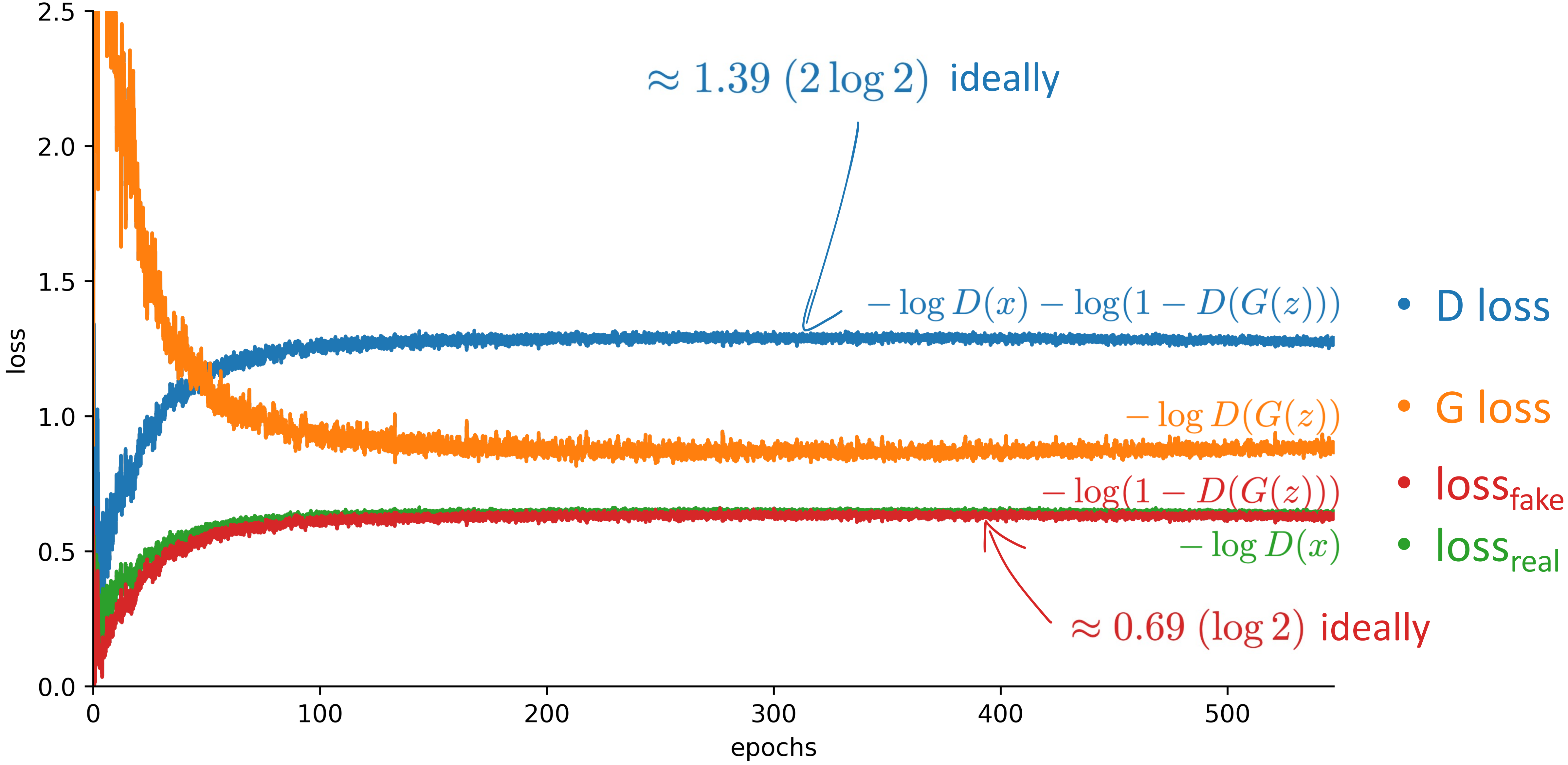
Running example: MNIST



*All objectives are negative of their original form

Code adapted from: https://github.com/prcastro/pytorch-gan/tree/master

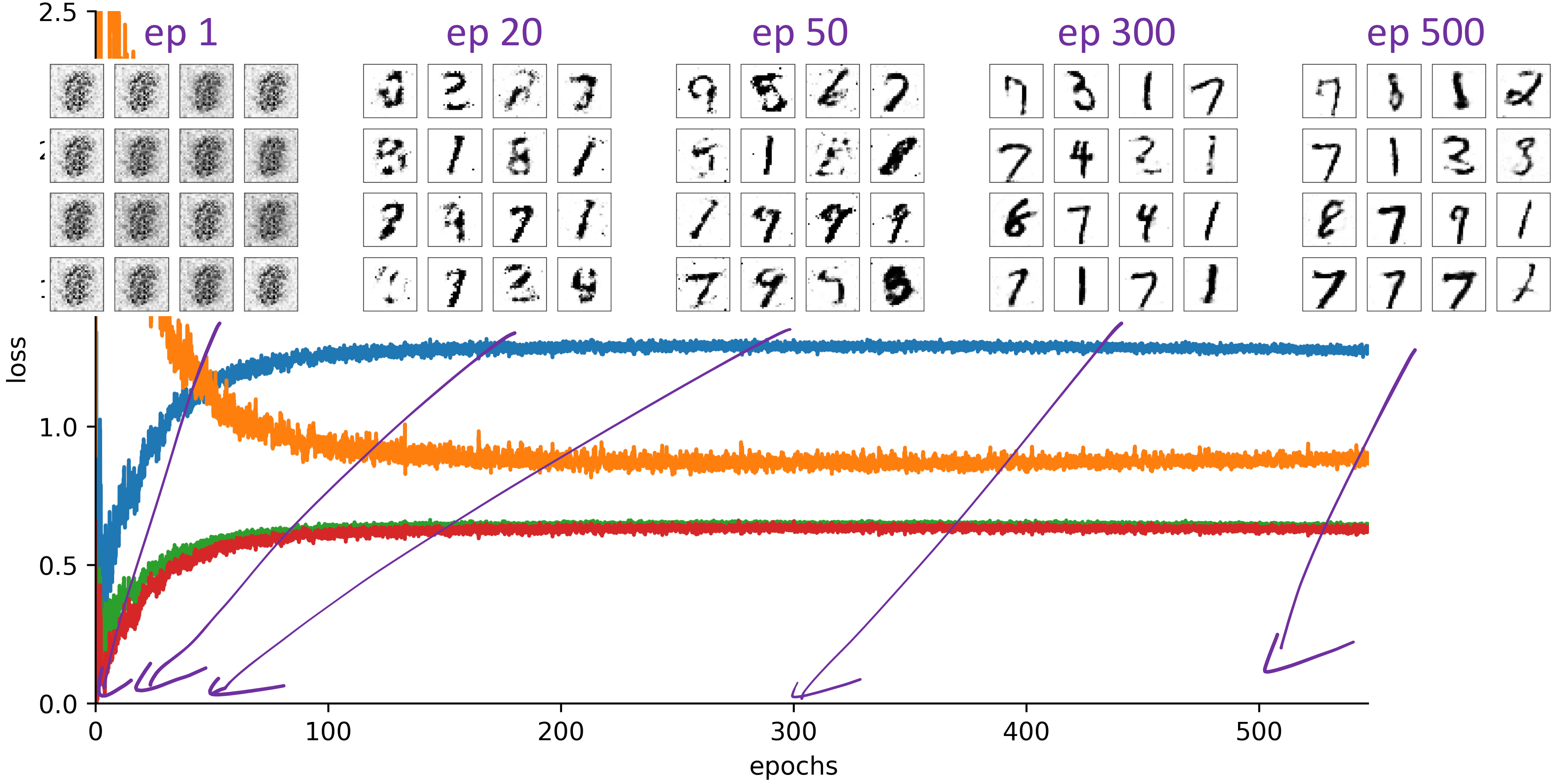
Running example: MNIST



*All objectives are negative of their original form

Code adapted from: https://github.com/prcastro/pytorch-gan/tree/master

Running example: MNIST



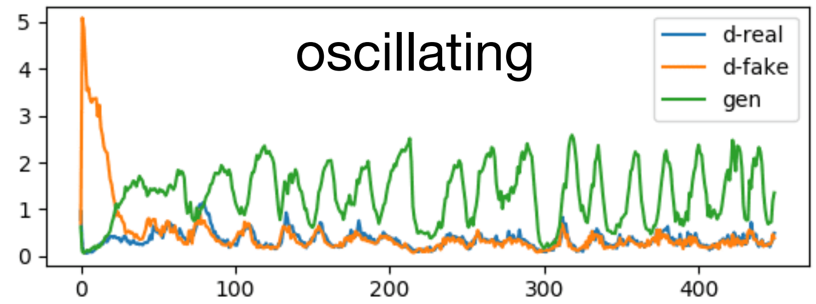
*All objectives are negative of their original form

Code adapted from: https://github.com/prcastro/pytorch-gan/tree/master

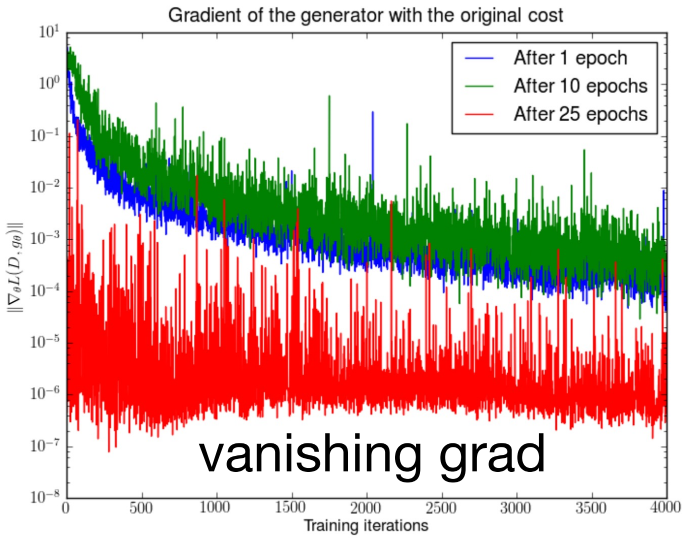
Problems of GAN

Difficult to train/converge

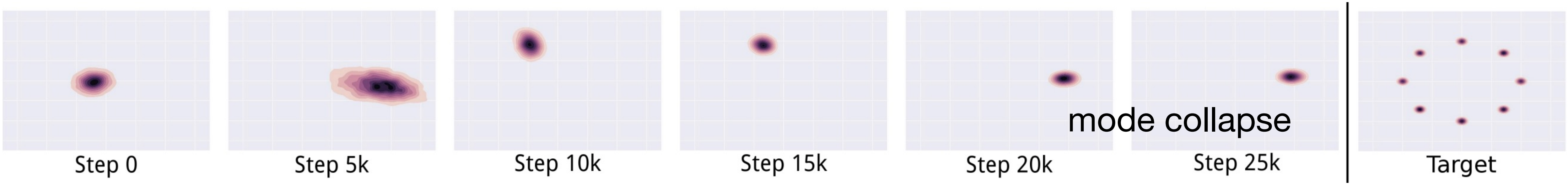
- Hard to achieve equilibrium
- Vanishing gradients
- Mode collapse



J. Brownlee, “How to Identify and Diagnose GAN Failure Modes”



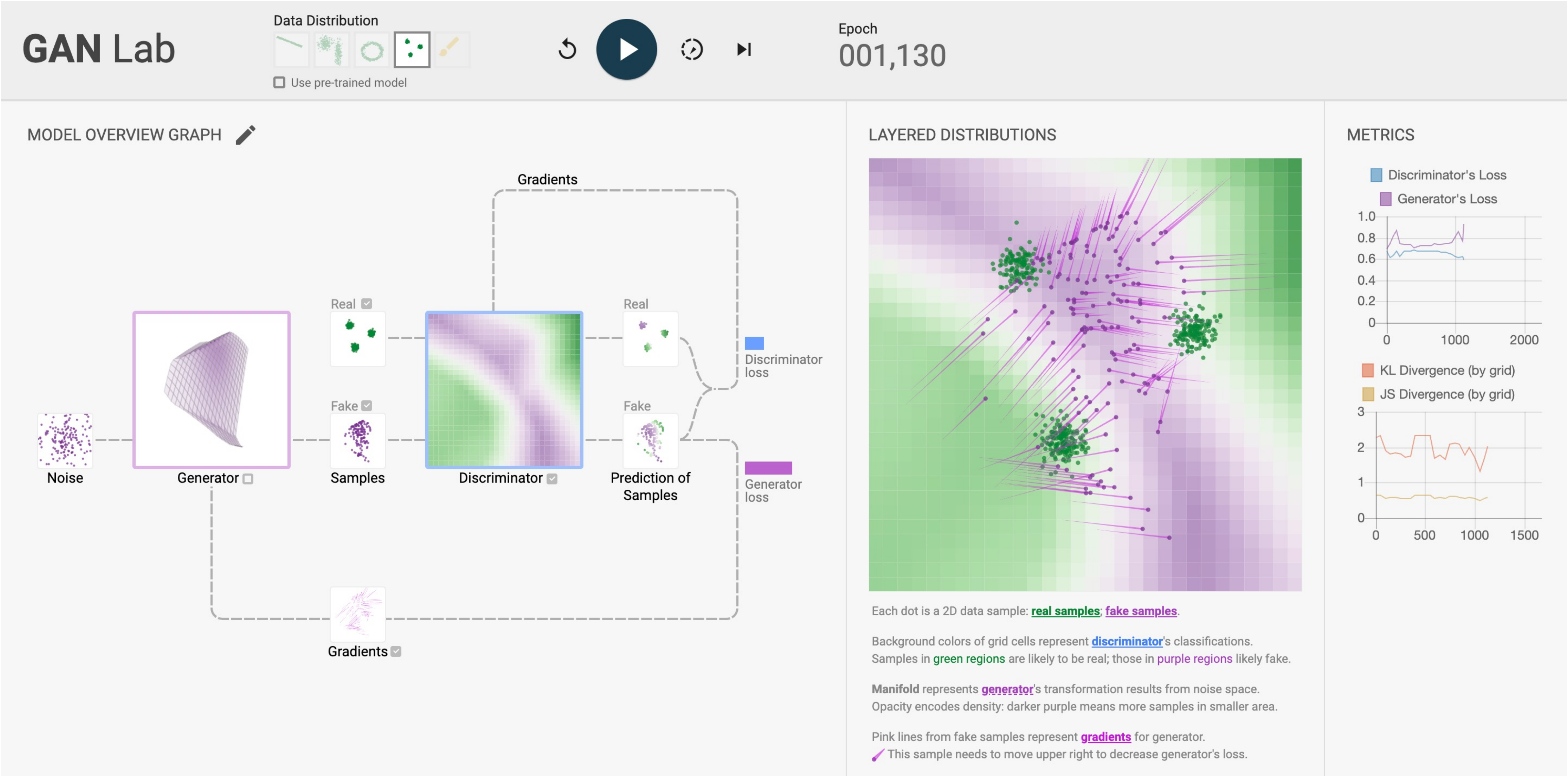
Arjovsky & Bottou, “Towards Principled Methods for Training GANs”



L. Metz, “Unrolled Generative Adversarial Networks”

Running example: GAN Lab

https://poloclub.github.io/ganlab/



Wasserstein GAN

W-GAN in Short

For mathematicians:

- Wasserstein distance, instead of JS divergence

For engineers:

- remove logarithms
- clip weights

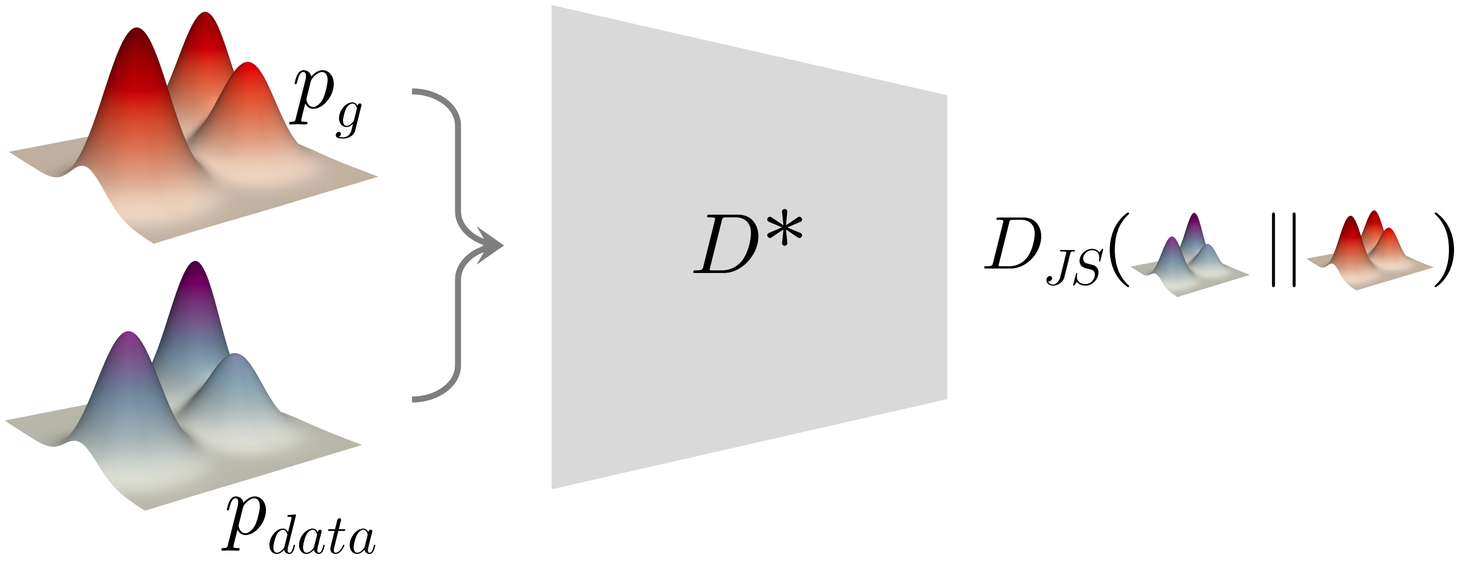
For laymen:

- art critic, instead of forgery expert

Summarized by Reddit user danielvarga: https://www.reddit.com/r/MachineLearning/comments/5qxoaz/comment/dd7aomb/

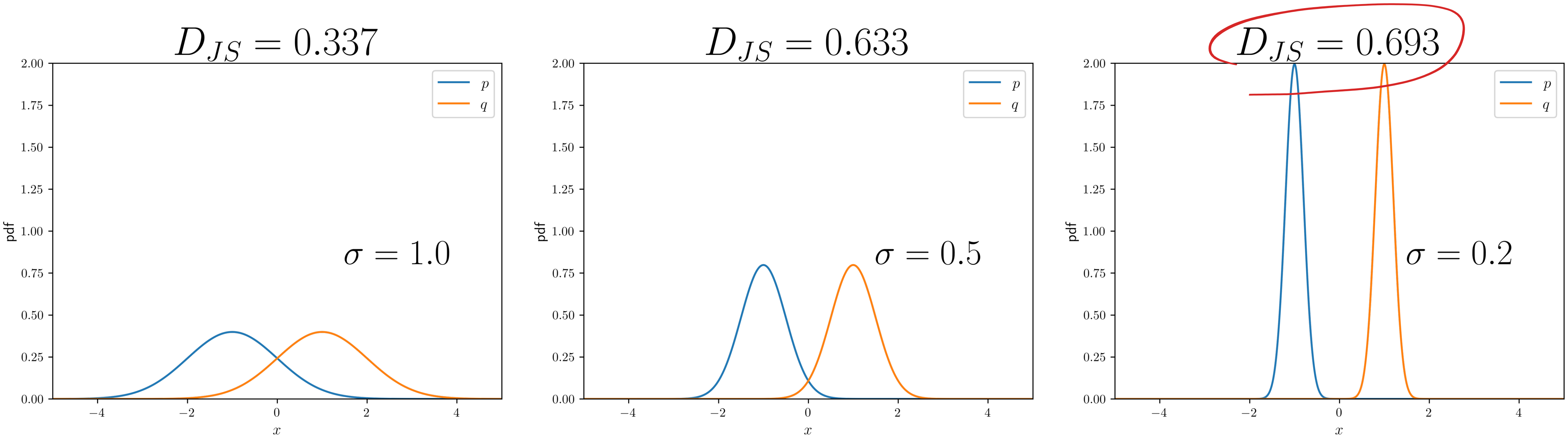
Recap: GAN optimizes for D_{JS}

$$\mathcal{L}(D^*, G) = 2D_{JS}(p_{\text{data}} \| p_g) - 2\log 2$$



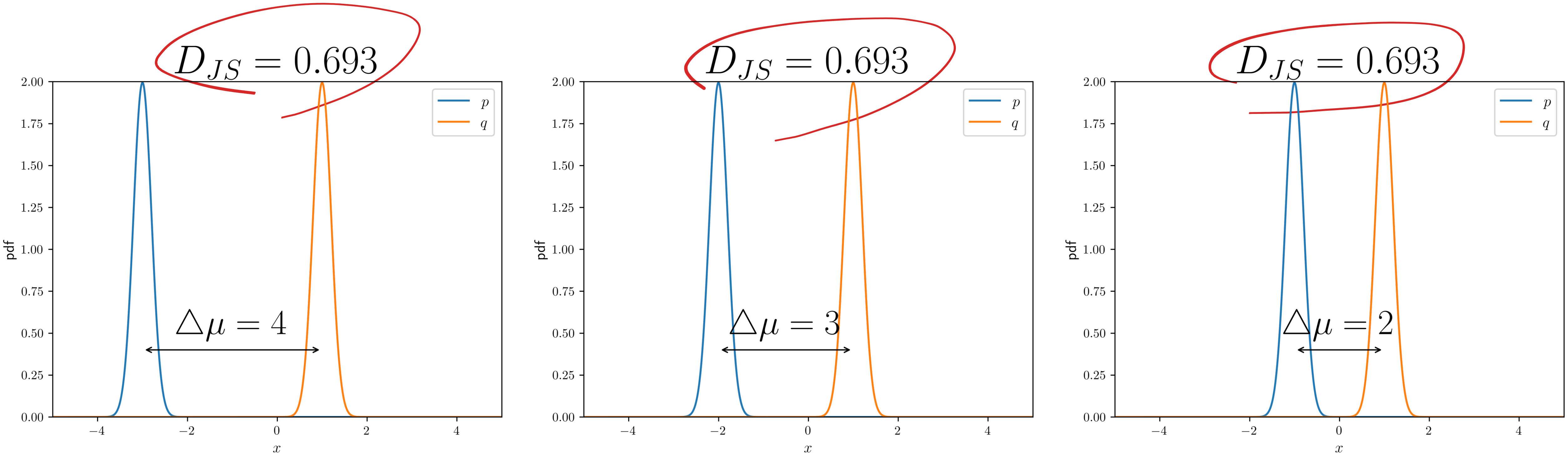
Problems of D_{JS}

If p and q don't overlap, D_{JS} is a constant (log2), i.e., no gradient



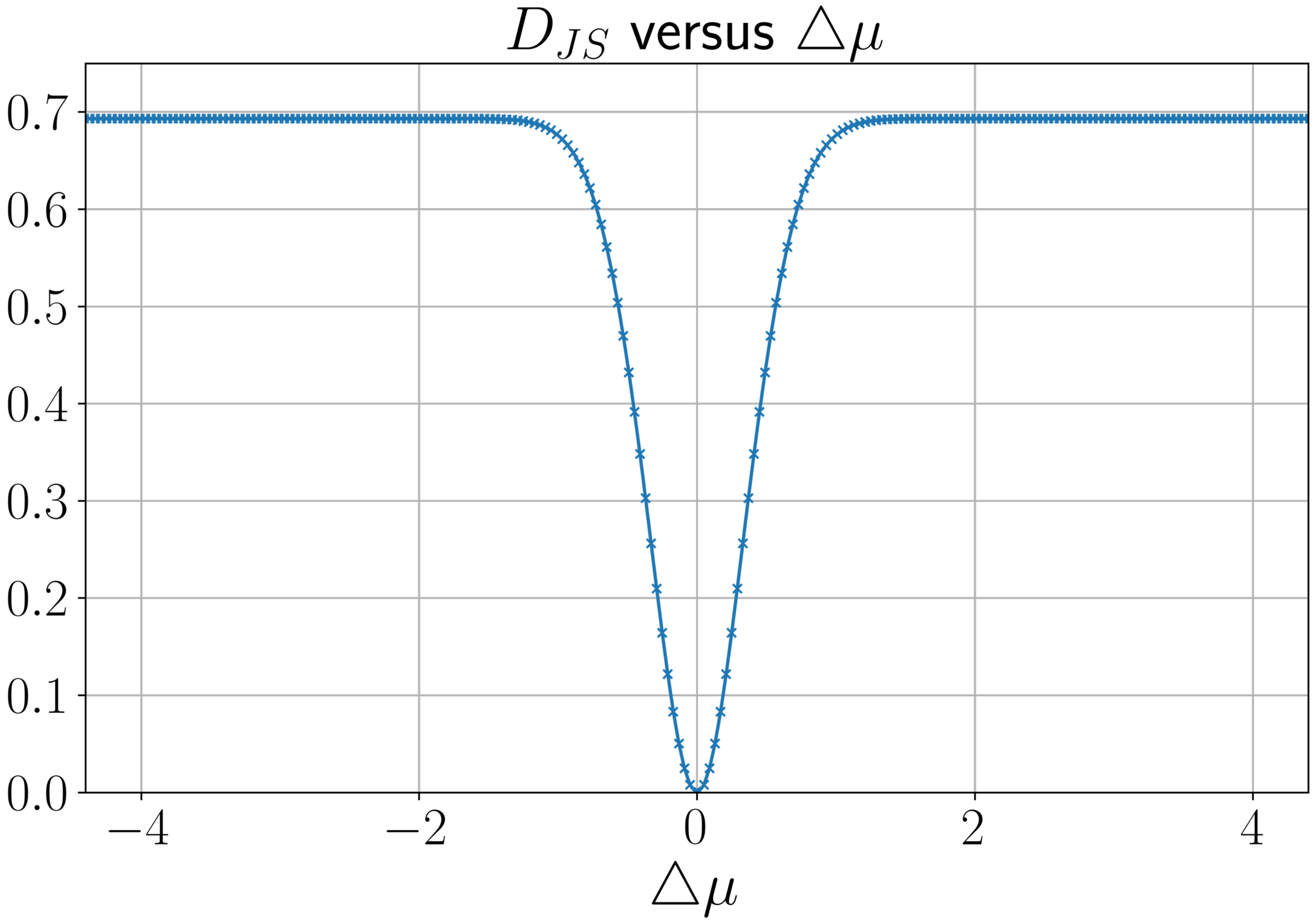
Problems of D_{JS}

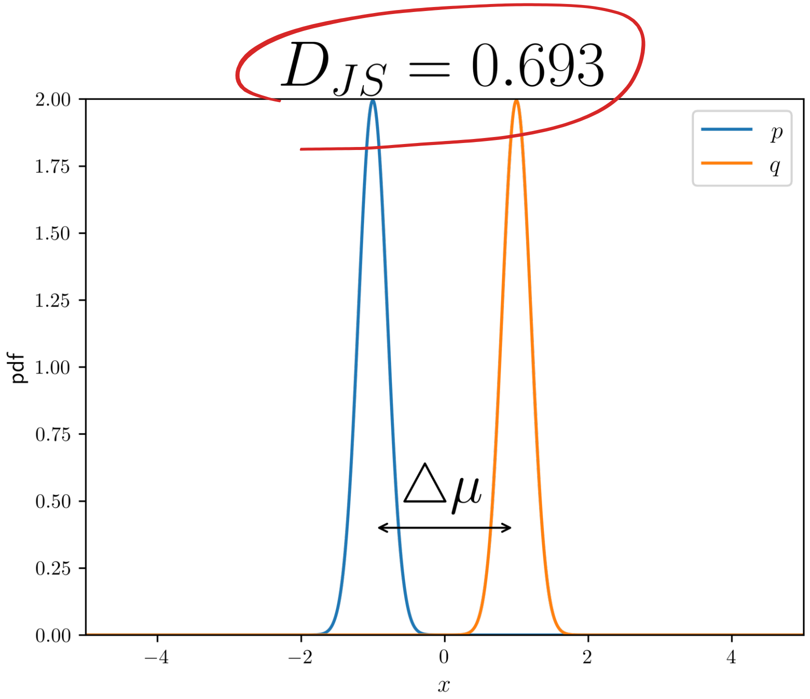
If p and q don't overlap, D_{JS} is a constant (log2), i.e., no gradient



Problems of D_{JS}

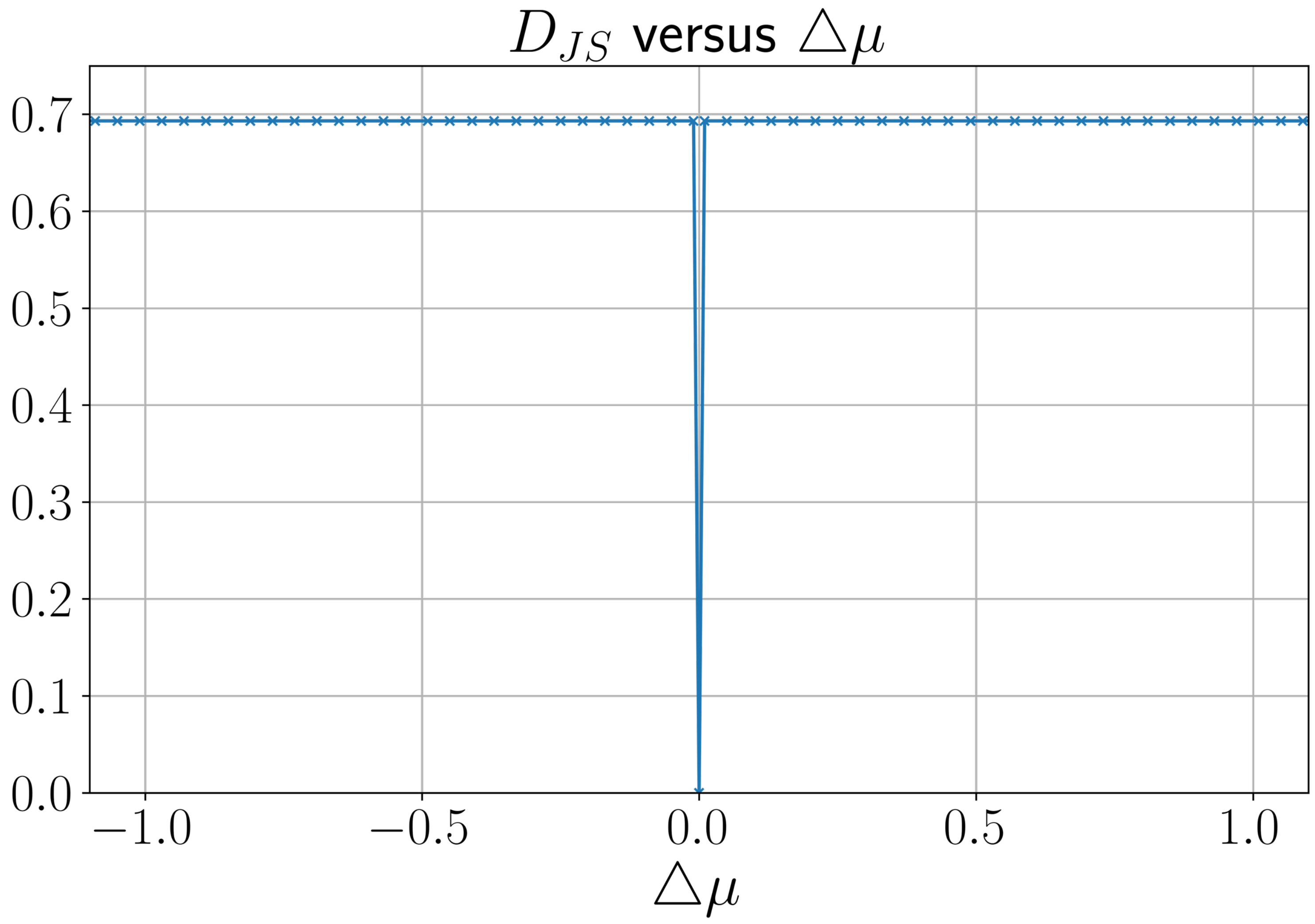
- D_{JS} is useful only if p and q are close

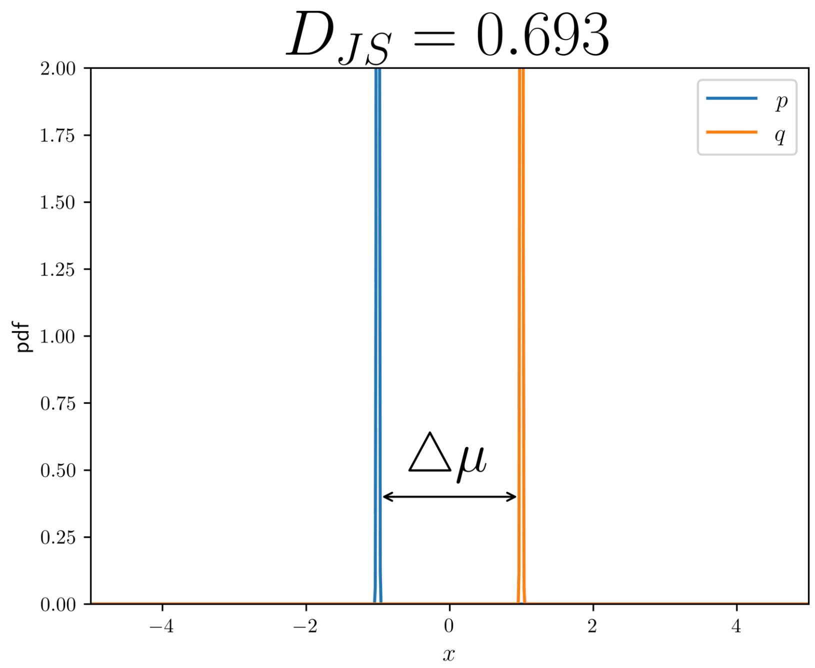




Problems of D_{JS}

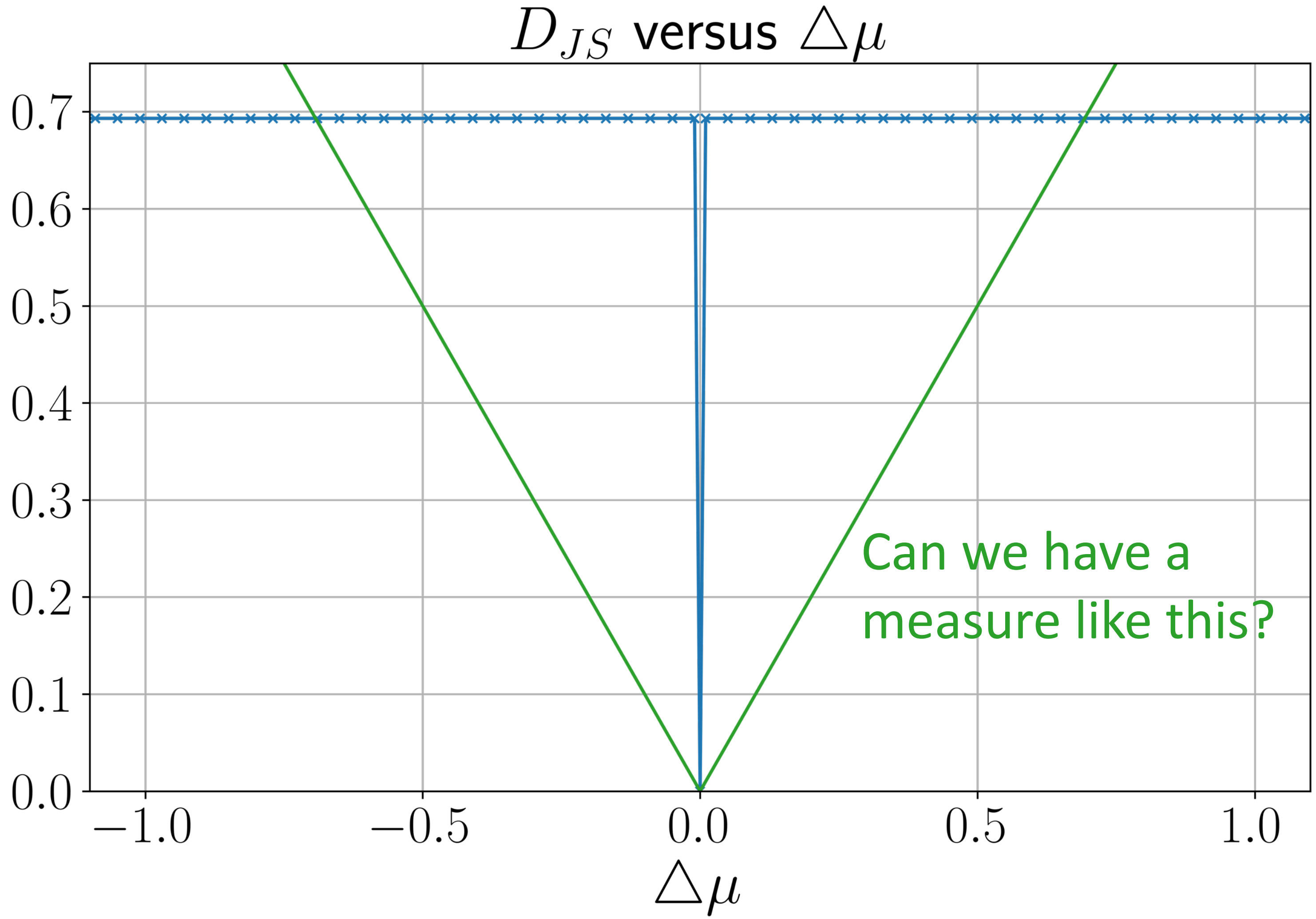
- D_{JS} is a delta function when p and q are delta functions

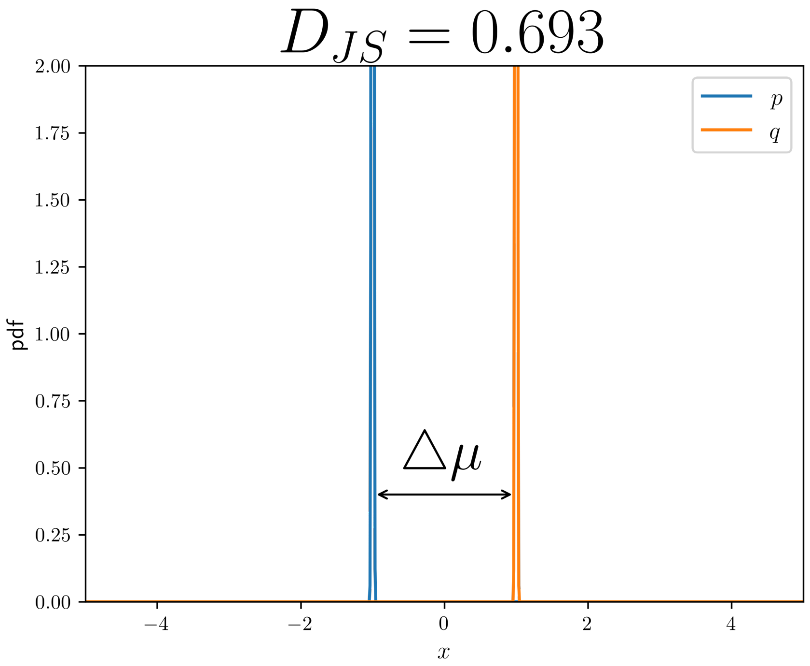




Problems of D_{JS}

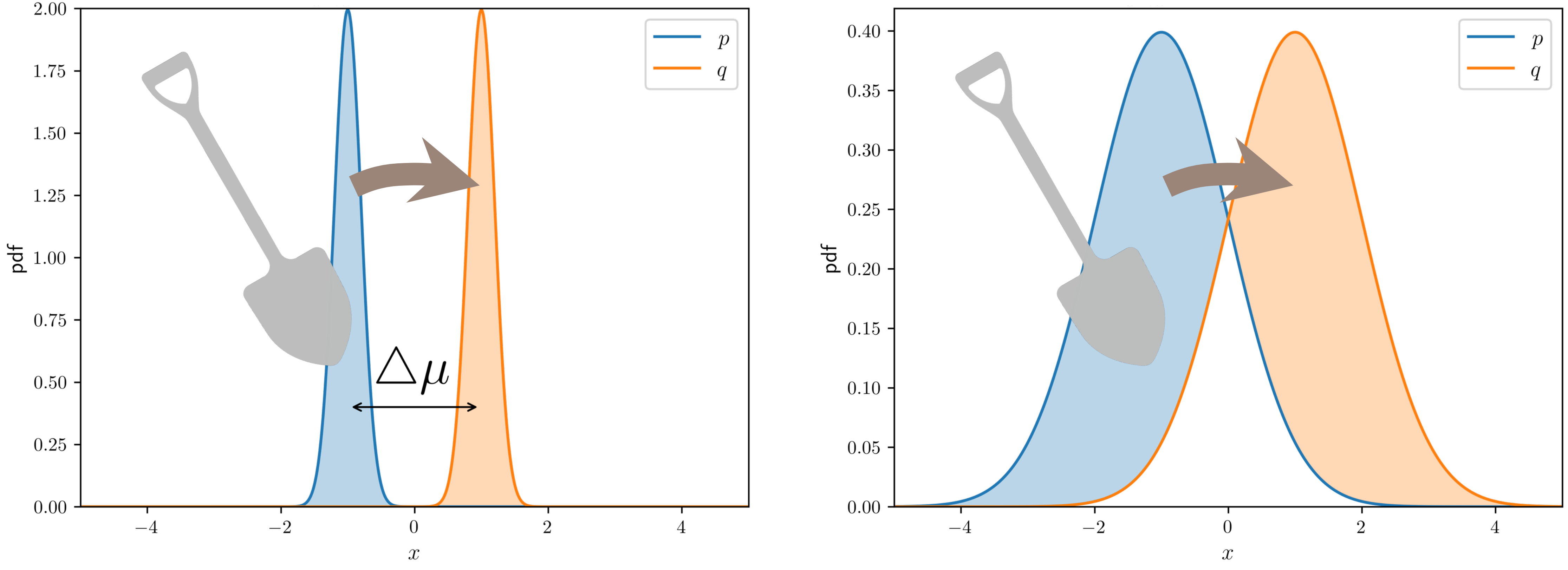
- D_{JS} is a delta function when p and q are delta functions



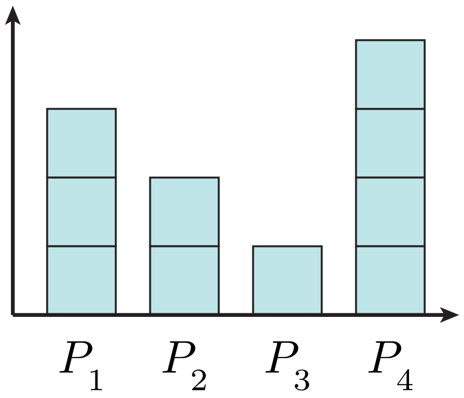


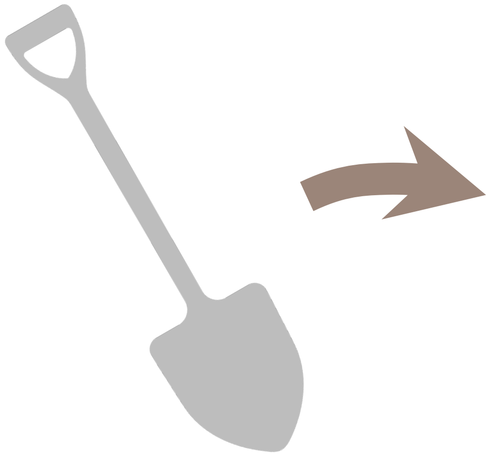
Wasserstein Distance

"Earth Mover's Distance"



Running example: Wasserstein Distance





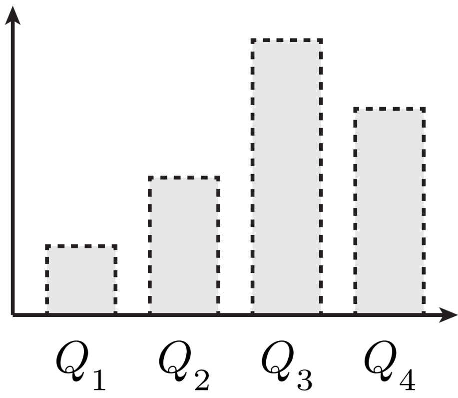
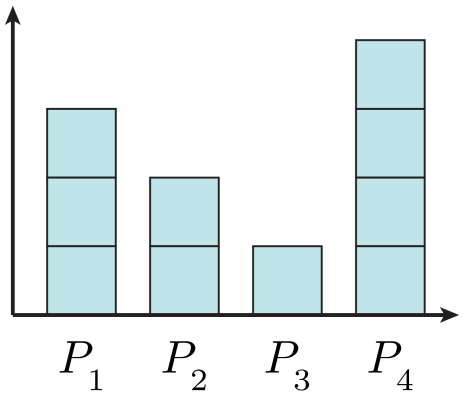
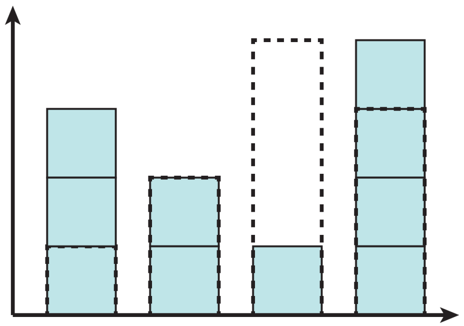


Figure inspired by: Lilian Weng, "From GAN to WGAN", arXiv:1904.08994

Running example: Wasserstein Distance





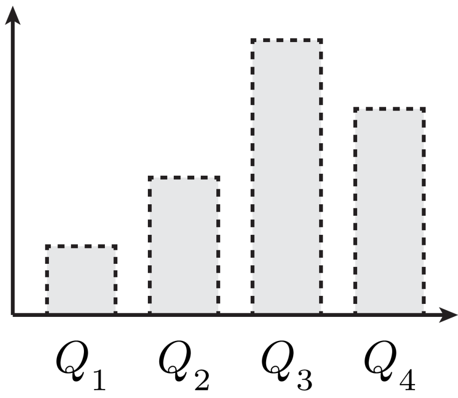
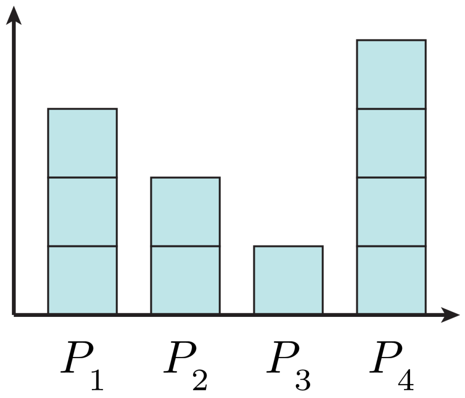
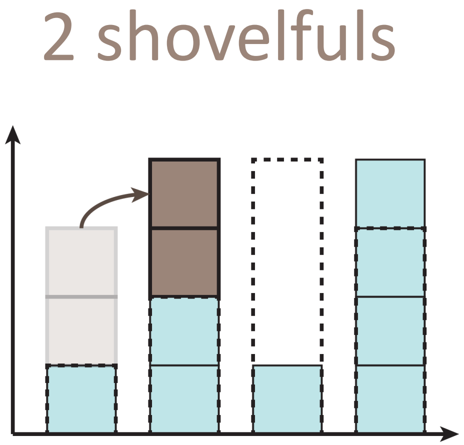


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance





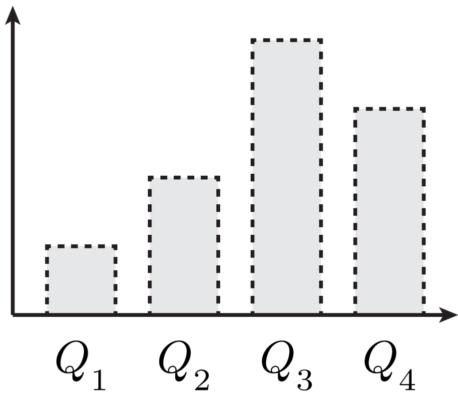


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

2 shovelfuls

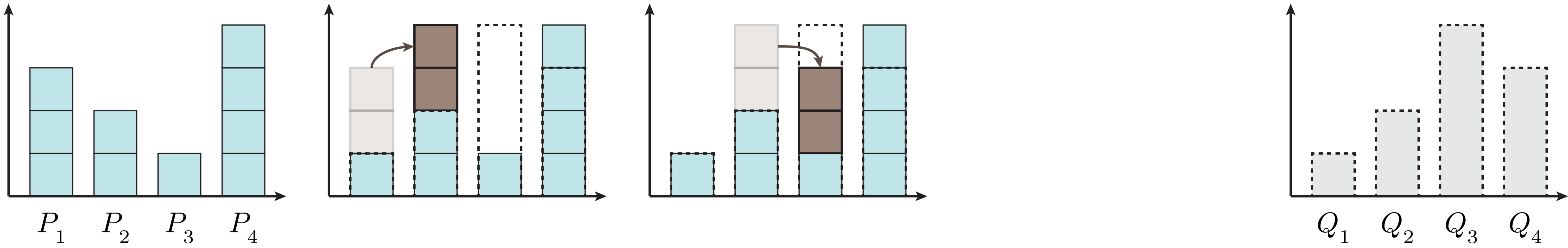


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

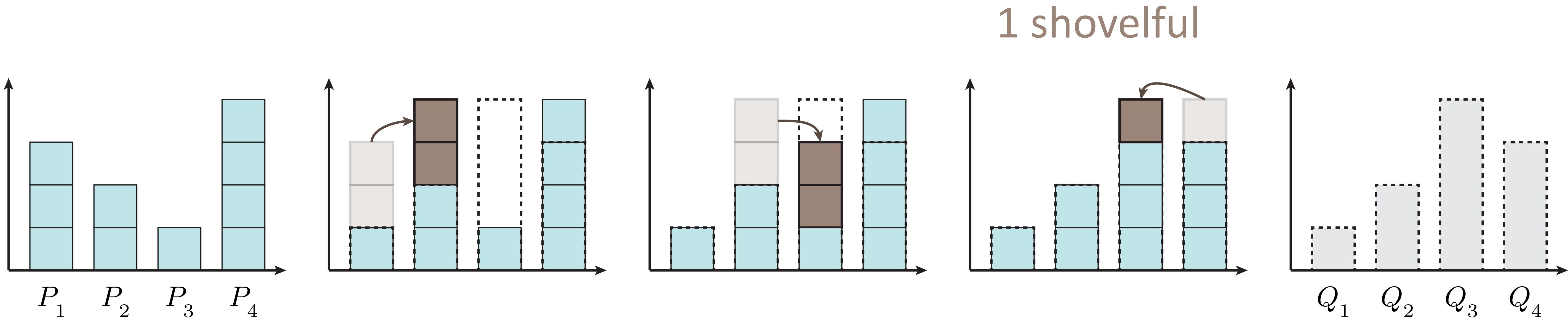


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

$$W(P,Q) = 5\times$$

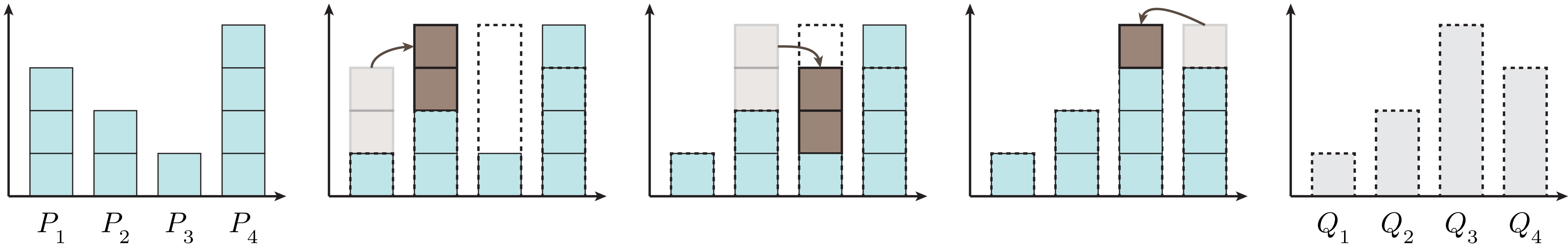
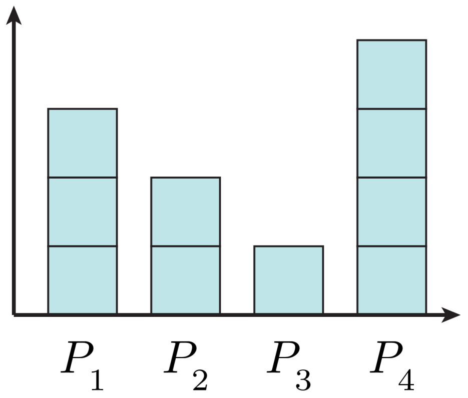


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance



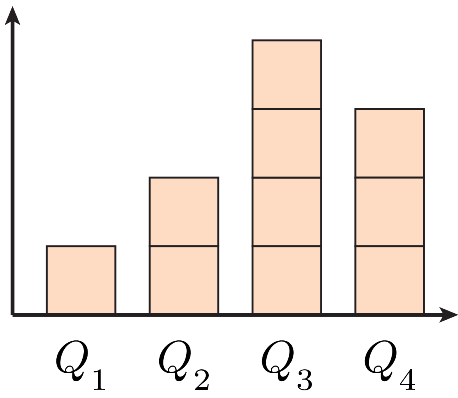
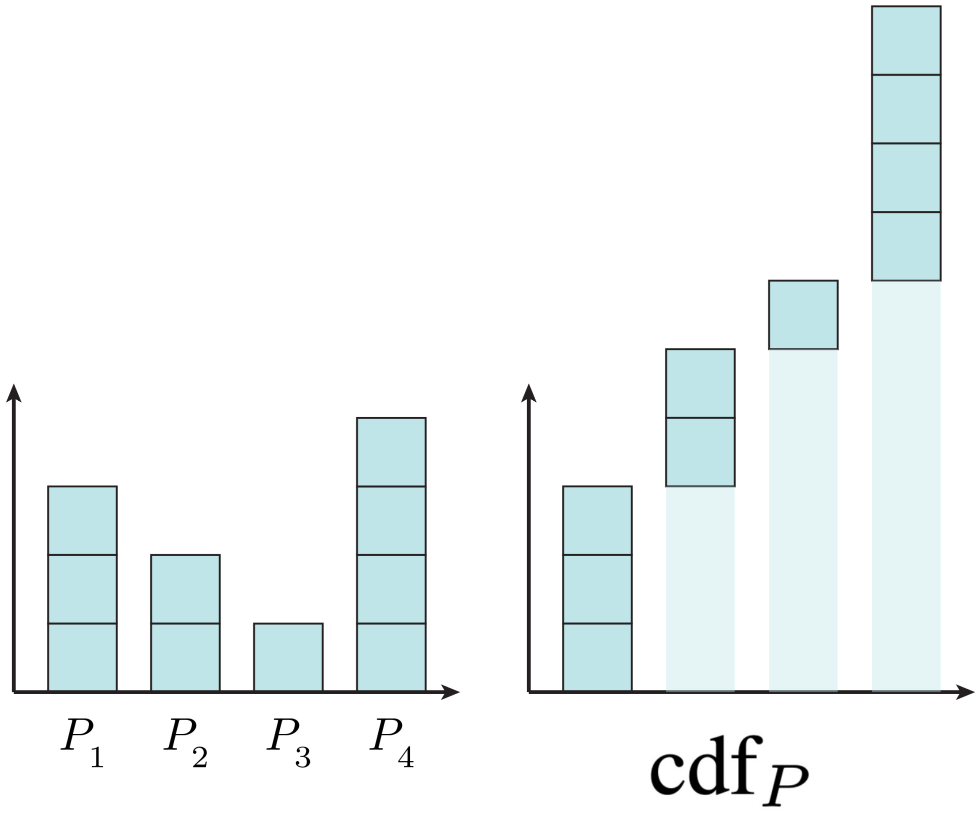
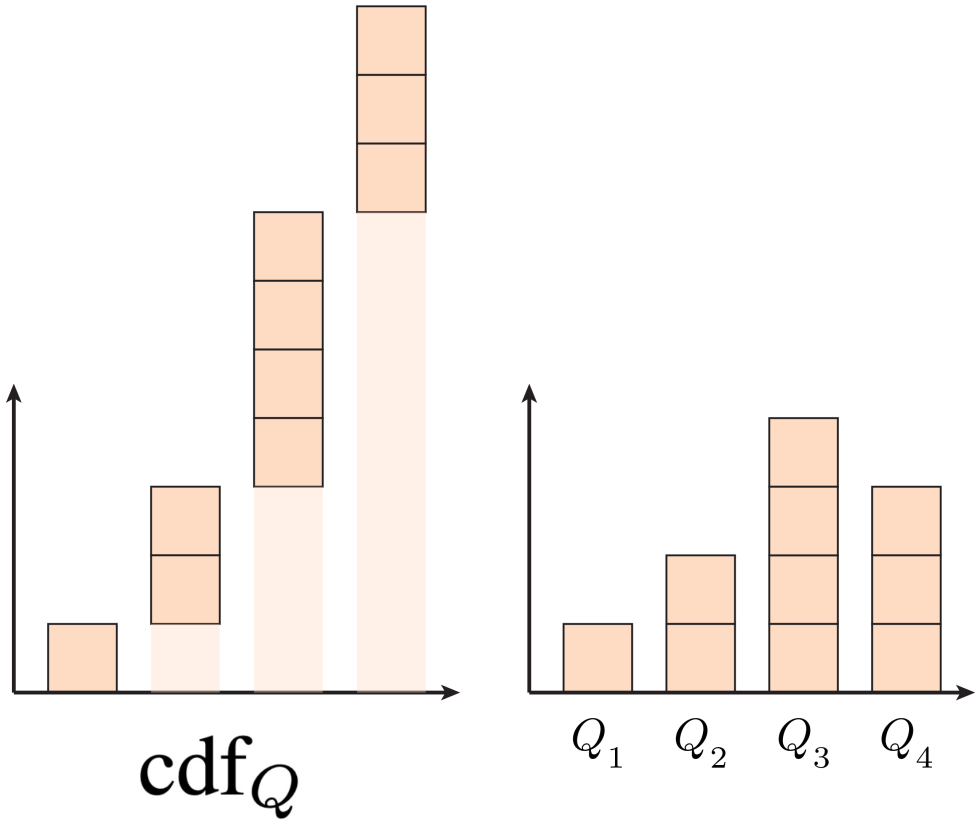


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

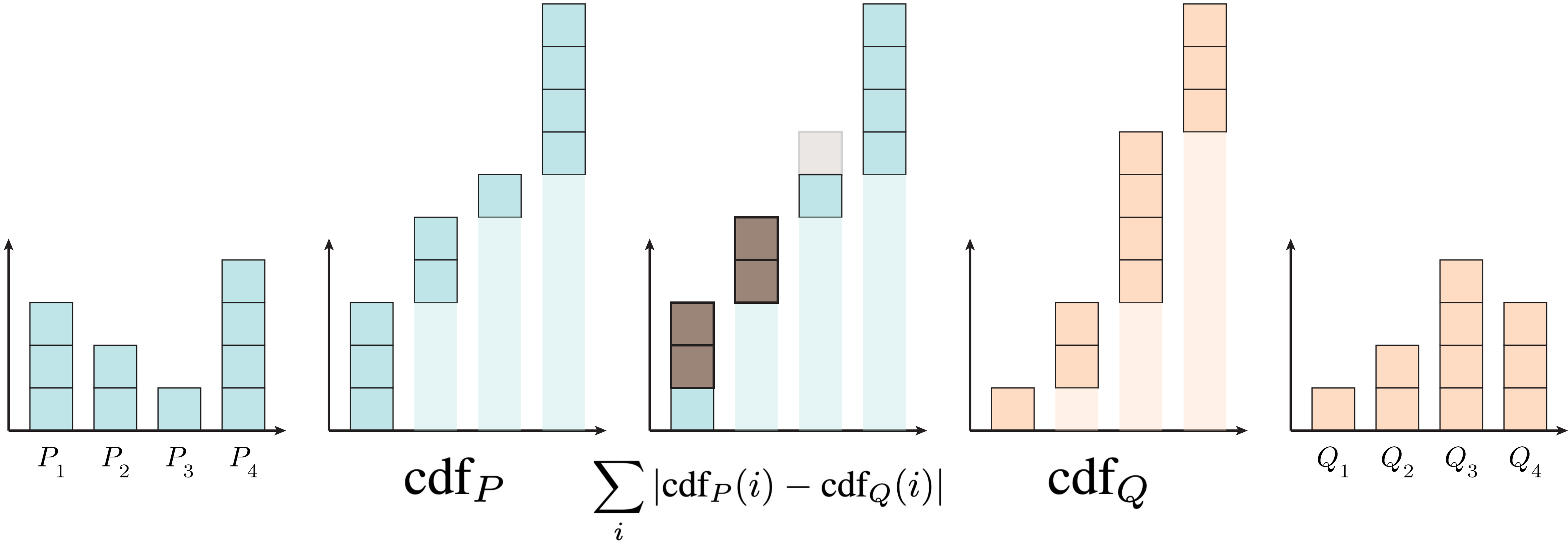




- cdf: cumulative distribution function

Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance



$$W(P, Q) = 5 \times \blacksquare$$

Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Wasserstein Distance

- 1-Wasserstein Distance (1-d, discrete)

l_1-norm

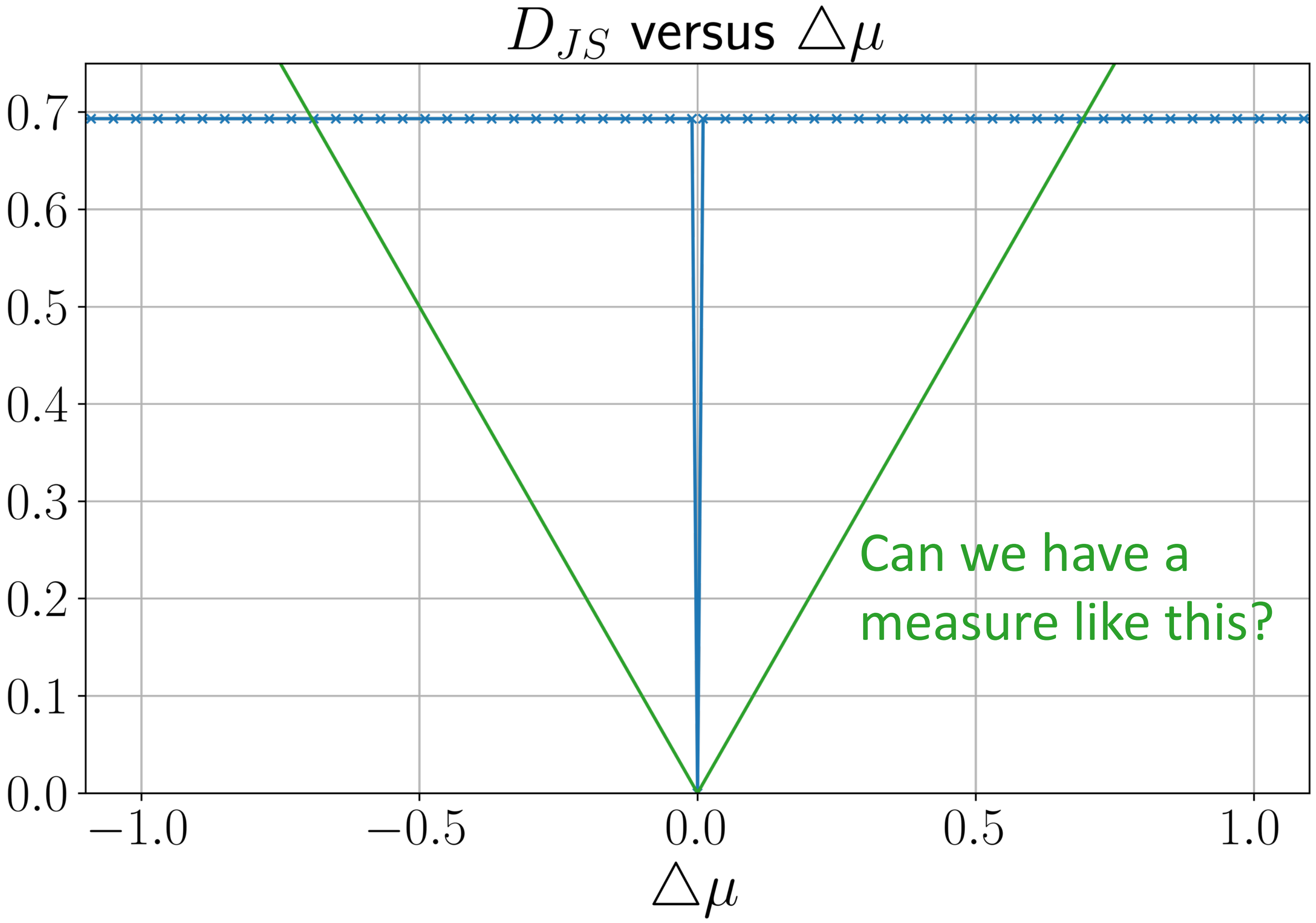
$$W_1(P,Q) = \sum_i |\mathrm{cdf}_P(i) - \mathrm{cdf}_Q(i)|$$

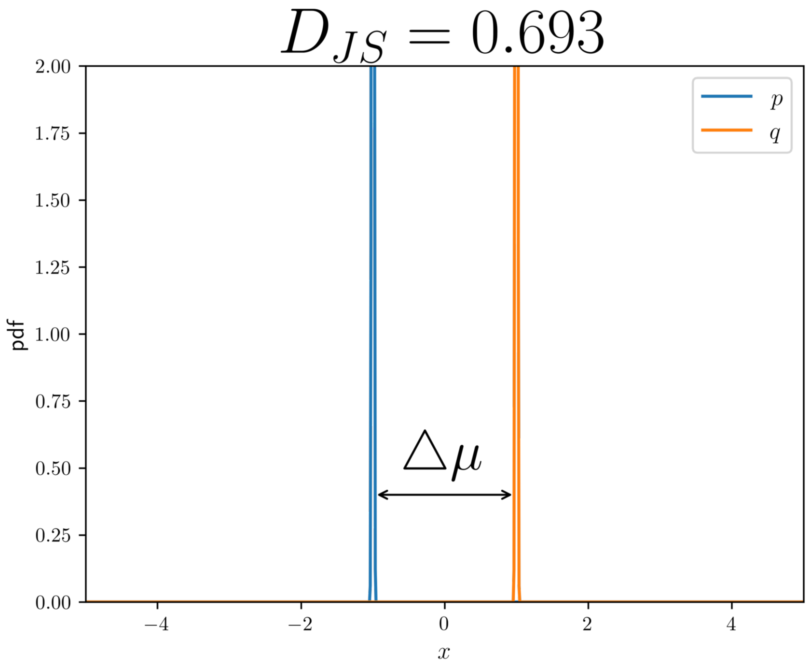
- 1-Wasserstein Distance (1-d, continuous)

$$W_1(p,q) = \int_x |\mathrm{cdf}_p(x) - \mathrm{cdf}_q(x)|dx$$

Recap: D_{JS}

- D_{JS} is a delta function when p and q are delta functions

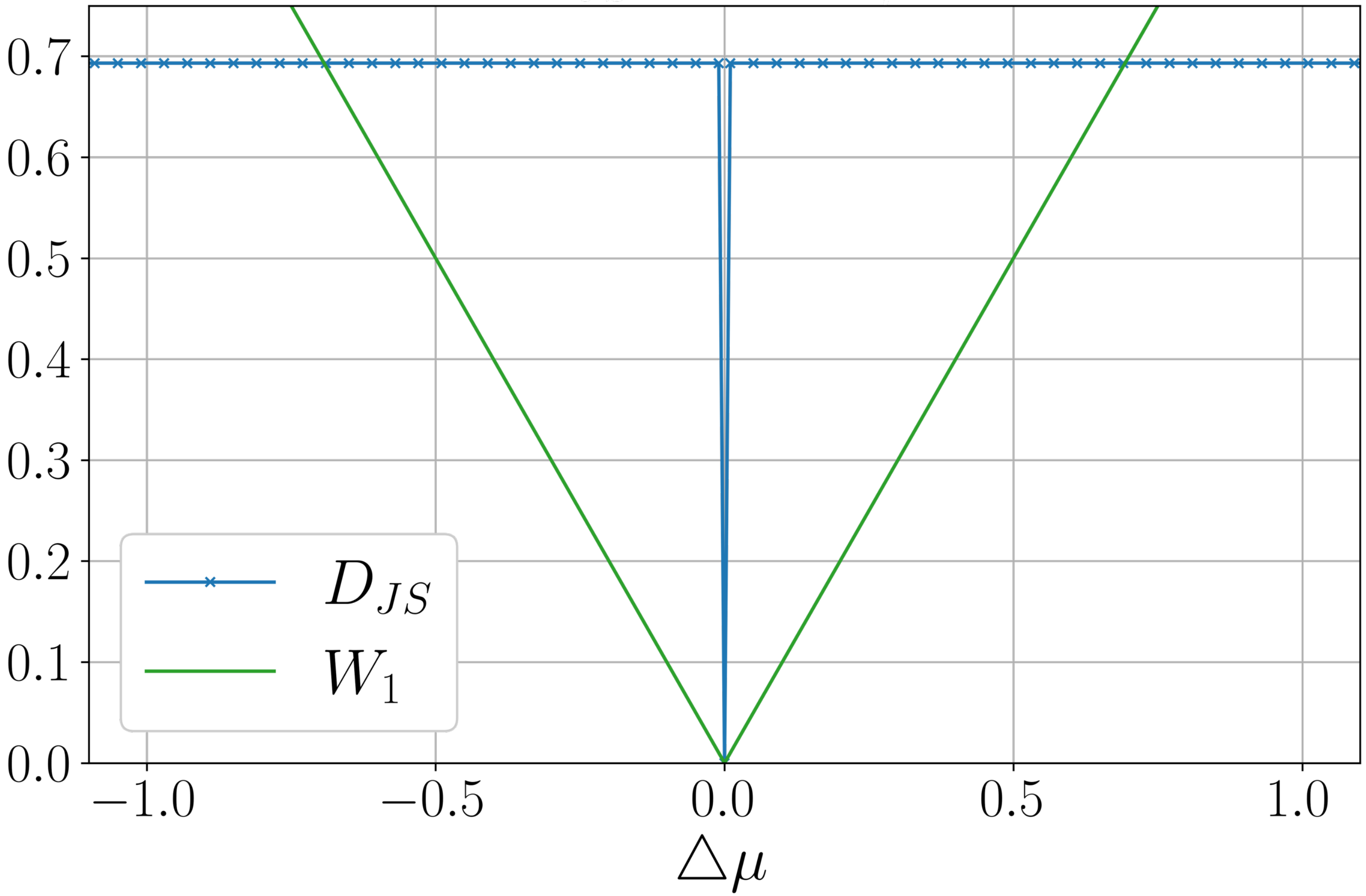


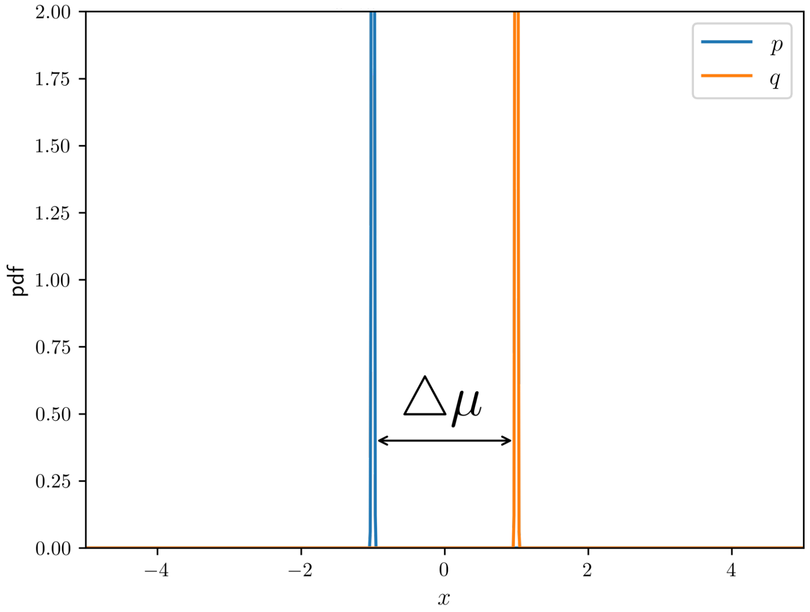


Wasserstein Distance

- when p and q are delta functions:

$$W_1(p, q) = |\mu_p - \mu_q|$$





Wasserstein Distance

- 1-Wasserstein Distance (high-dim, continuous)

$$W_1(p, q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||]$$

- all joint distributions $\gamma(x, y)$ whose marginals are p and q

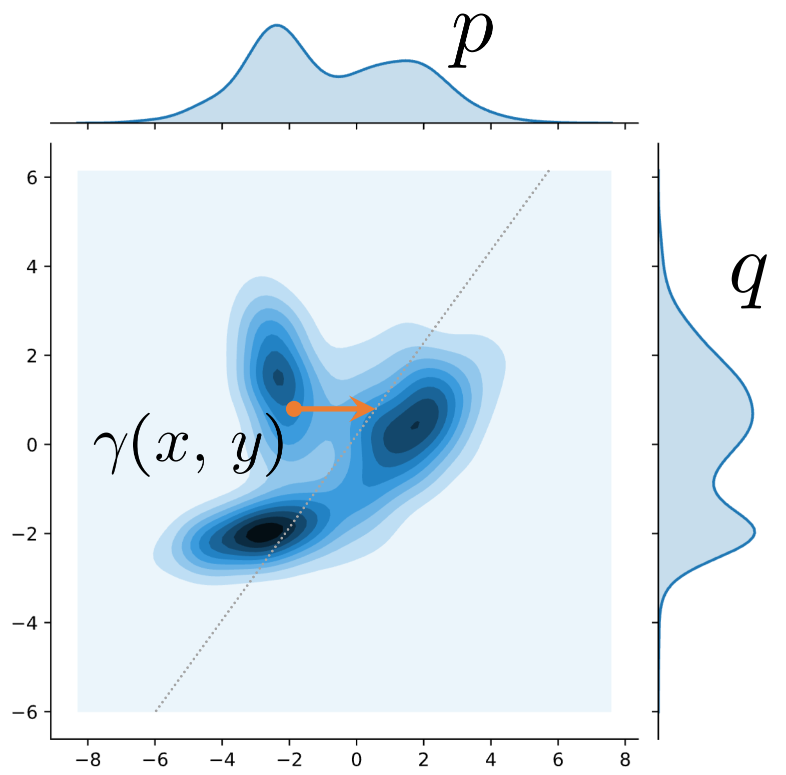
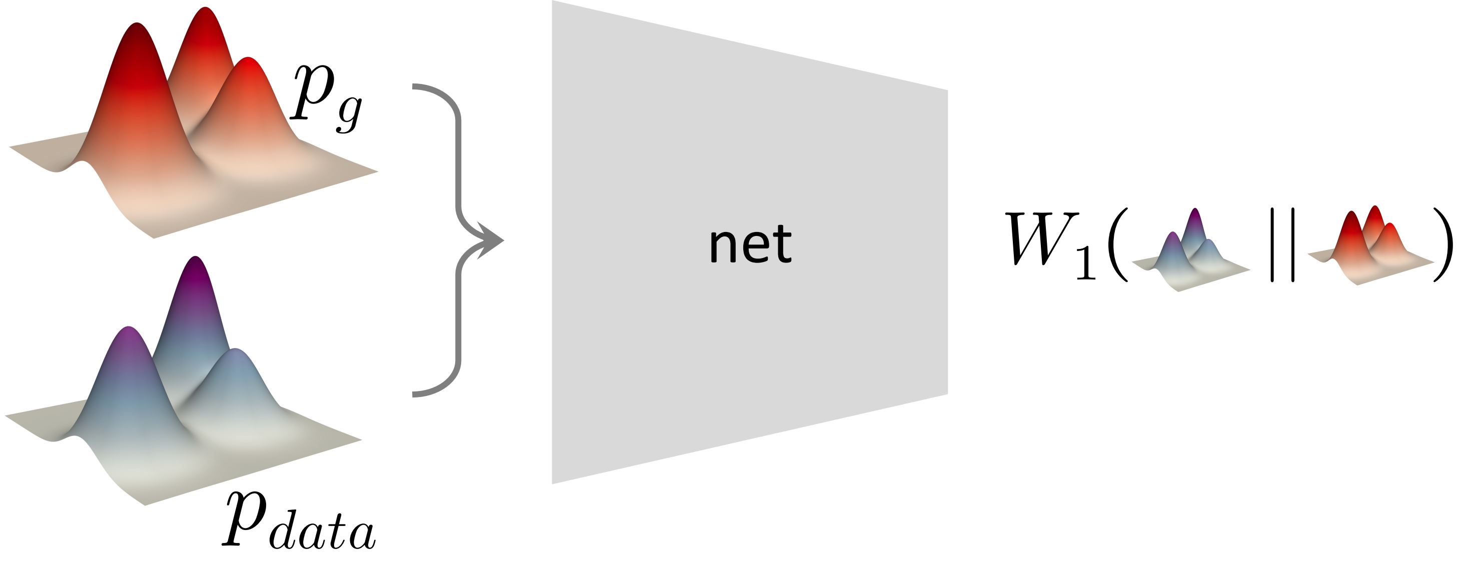


Figure: https://en.wikipedia.org/wiki/Wasserstein_metric

W-GAN optimizes for Wasserstein Distance

$$W_1(p, q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$



W-GAN optimizes for Wasserstein Distance

- Kantorovich-Rubinstein duality:

$$W_1(p, q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p}[f(x)] - \mathbb{E}_{x \sim q}[f(x)]$$

- all 1-Lipschitz functions

W-GAN optimizes for Wasserstein Distance

- Kantorovich-Rubinstein duality:

$$W_1(p,q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)]$$

- all 1-Lipschitz functions

K-Lipschitz continuity:

$$|f(x) - f(y)| \leq K|x - y|, \quad \forall x, y$$

gradient is bounded:

$$\frac{|f(x) - f(y)|}{|x - y|} \leq K$$

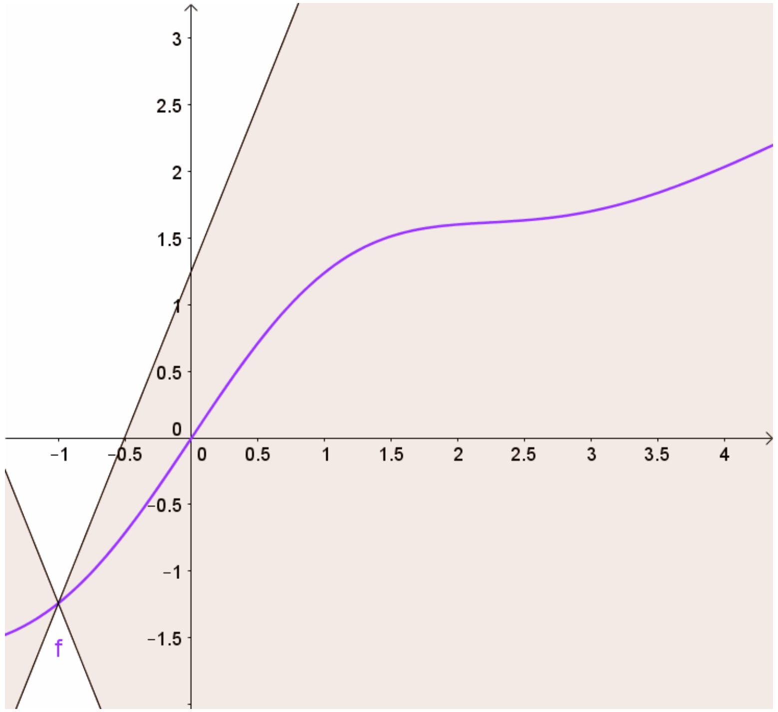


Figure from: https://en.wikipedia.org/wiki/Lipschitz_continuity

W-GAN optimizes for Wasserstein Distance

- Kantorovich-Rubinstein duality:

$$W_1(p,q) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)]$$

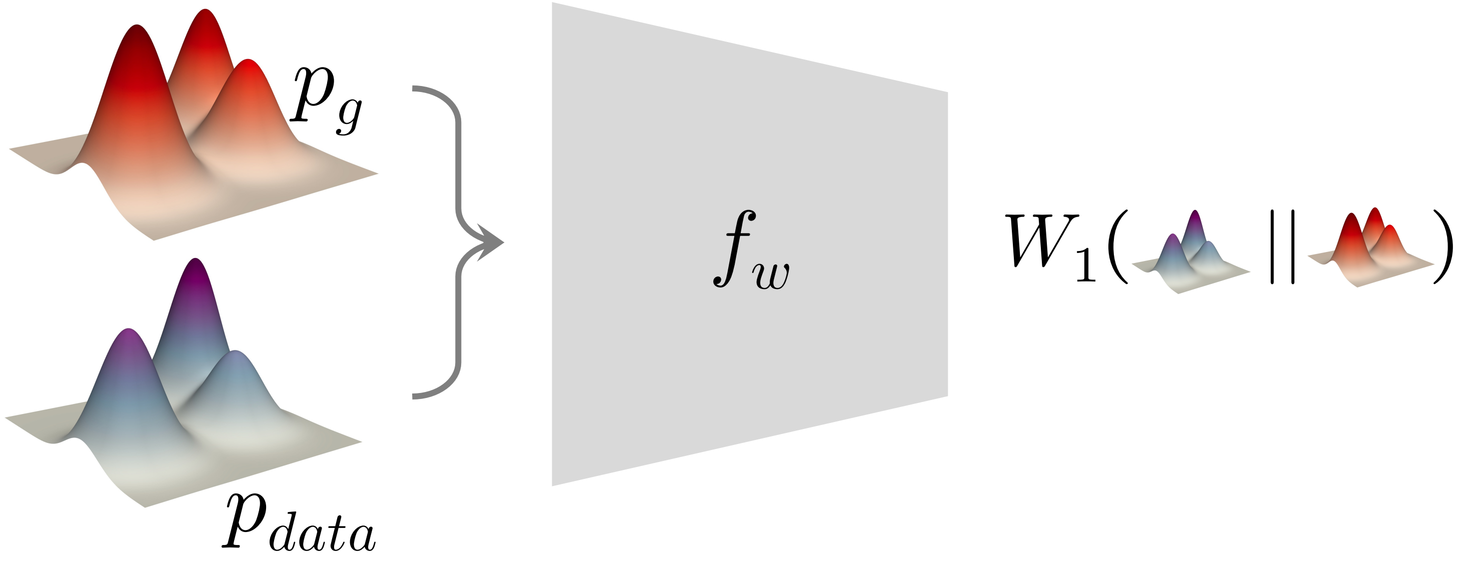
K-Lipschitz continuity:

$$|f(x) - f(y)| \leq K|x - y|, \quad \forall x, y$$

W-GAN optimizes for Wasserstein Distance

- W-GAN's objective function:

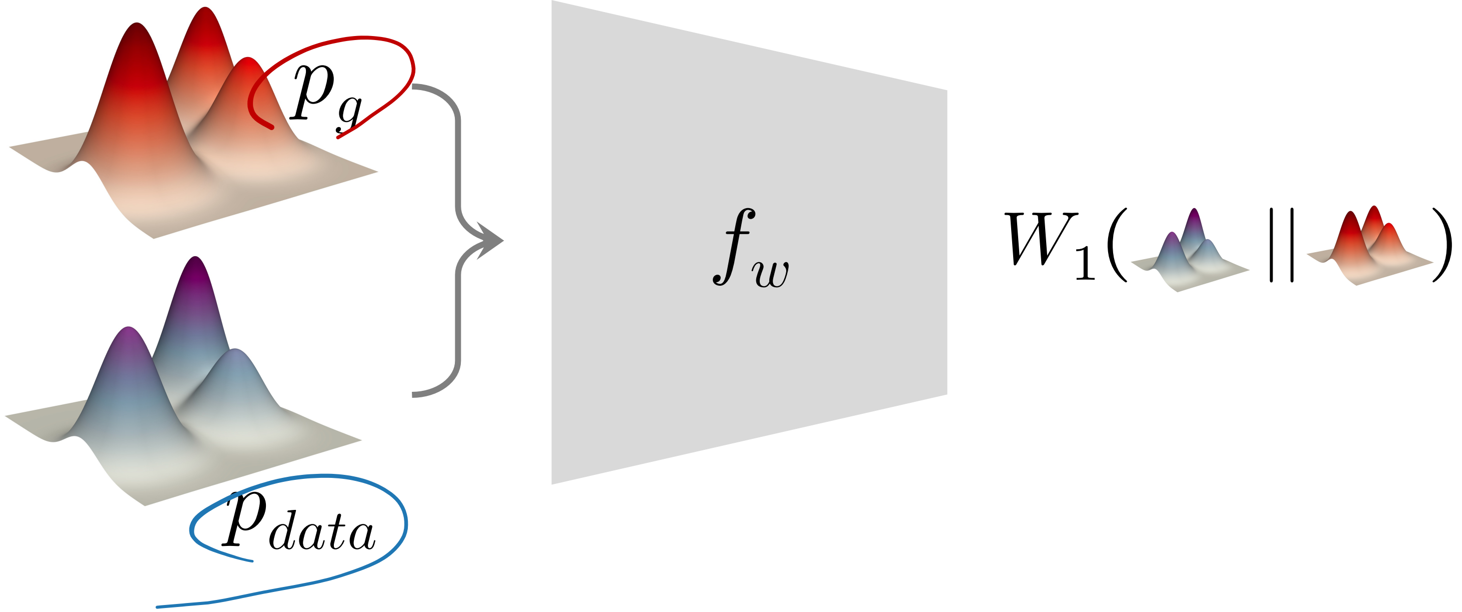
$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_{\text{data}}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]$$



W-GAN optimizes for Wasserstein Distance

- W-GAN's objective function:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_{\text{data}}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]$$

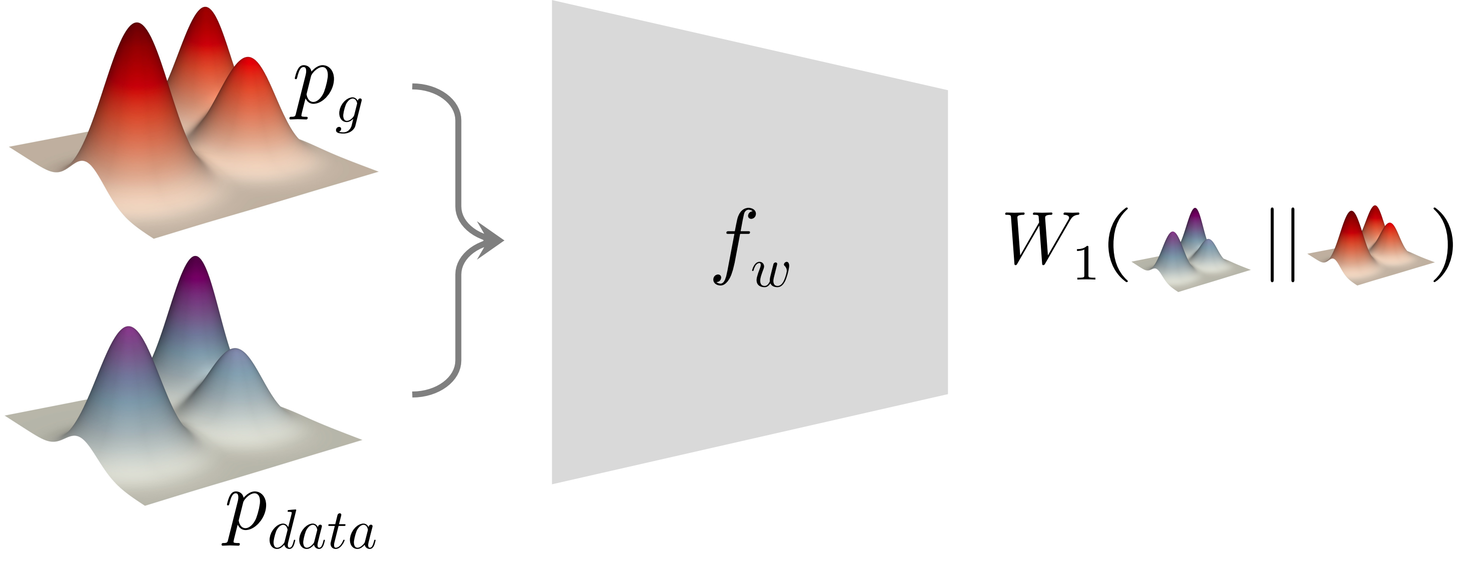


W-GAN optimizes for Wasserstein Distance

- W-GAN's objective function:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_{\text{data}}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]$$

  - weights are bounded: in practice, clipped $[-0.01,\ 0.01]$



W-GAN vs. original GAN

- W-GAN's objective function:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_{\text{data}}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]$$

- clip weights
- remove logarithms

- original GAN's objective function (D-step):

$$\max_{D} \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{x \sim p_g}[\log(1 - D(x))]$$

W-GAN vs. original GAN

"art critic"
- value/merit/quality/...
- direction to improve (gradients)

- W-GAN's objective function:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_{\text{data}}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]$$

- original GAN's objective function (D-step):

$$\max_{D} \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{x \sim p_g}[\log(1 - D(x))]$$

"forgery expert"
- real/fake

W-GAN algorithm annotated

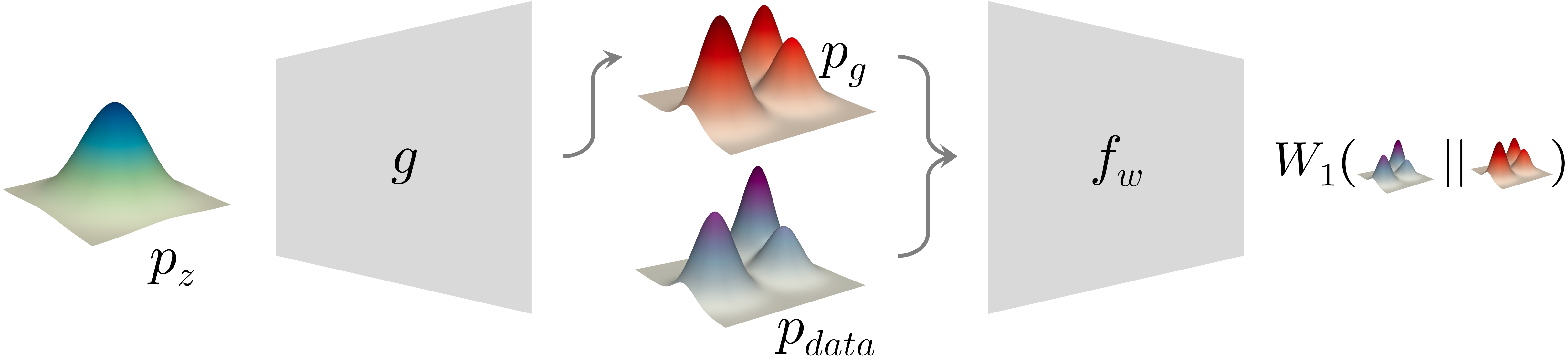
Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

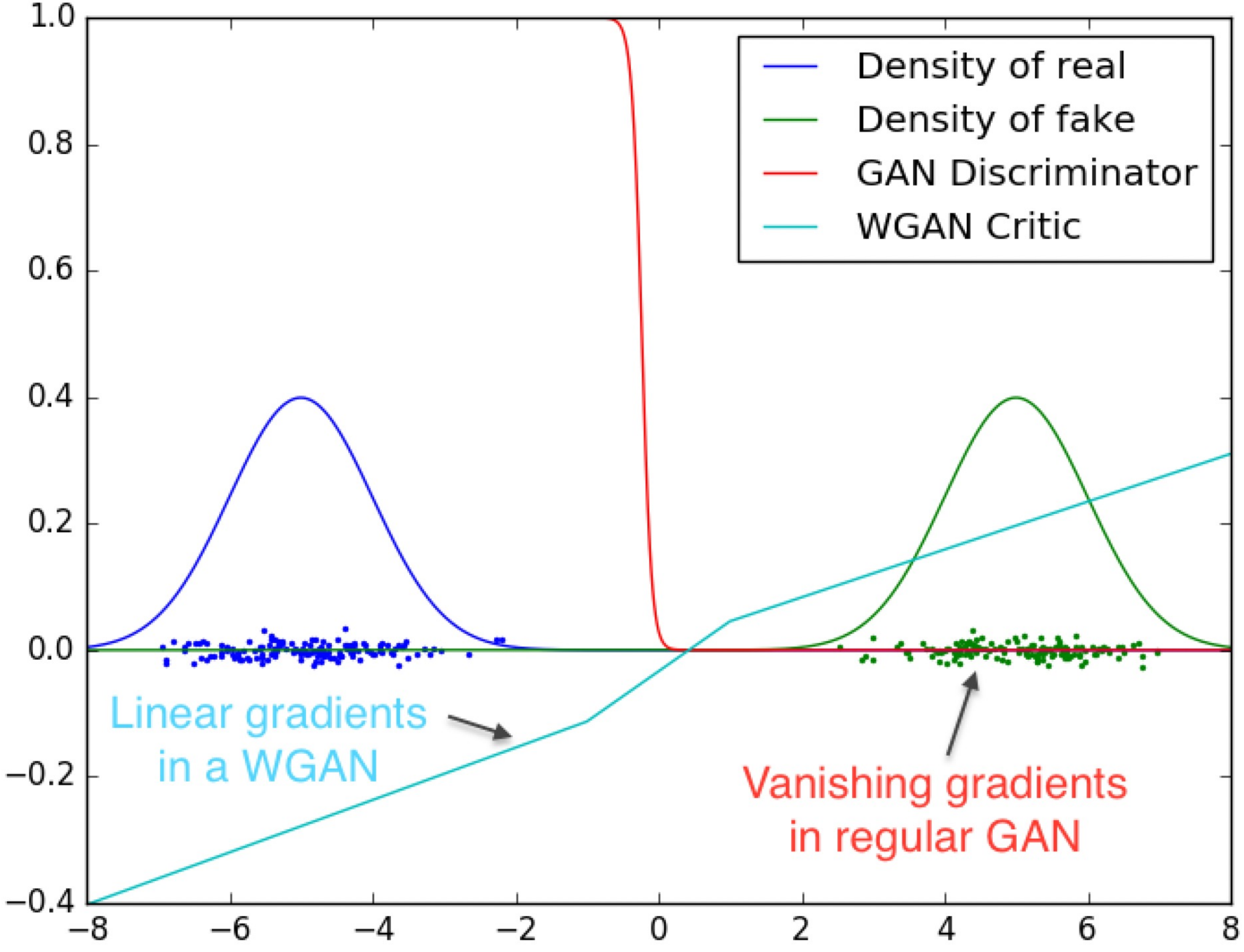
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

remove logarithms

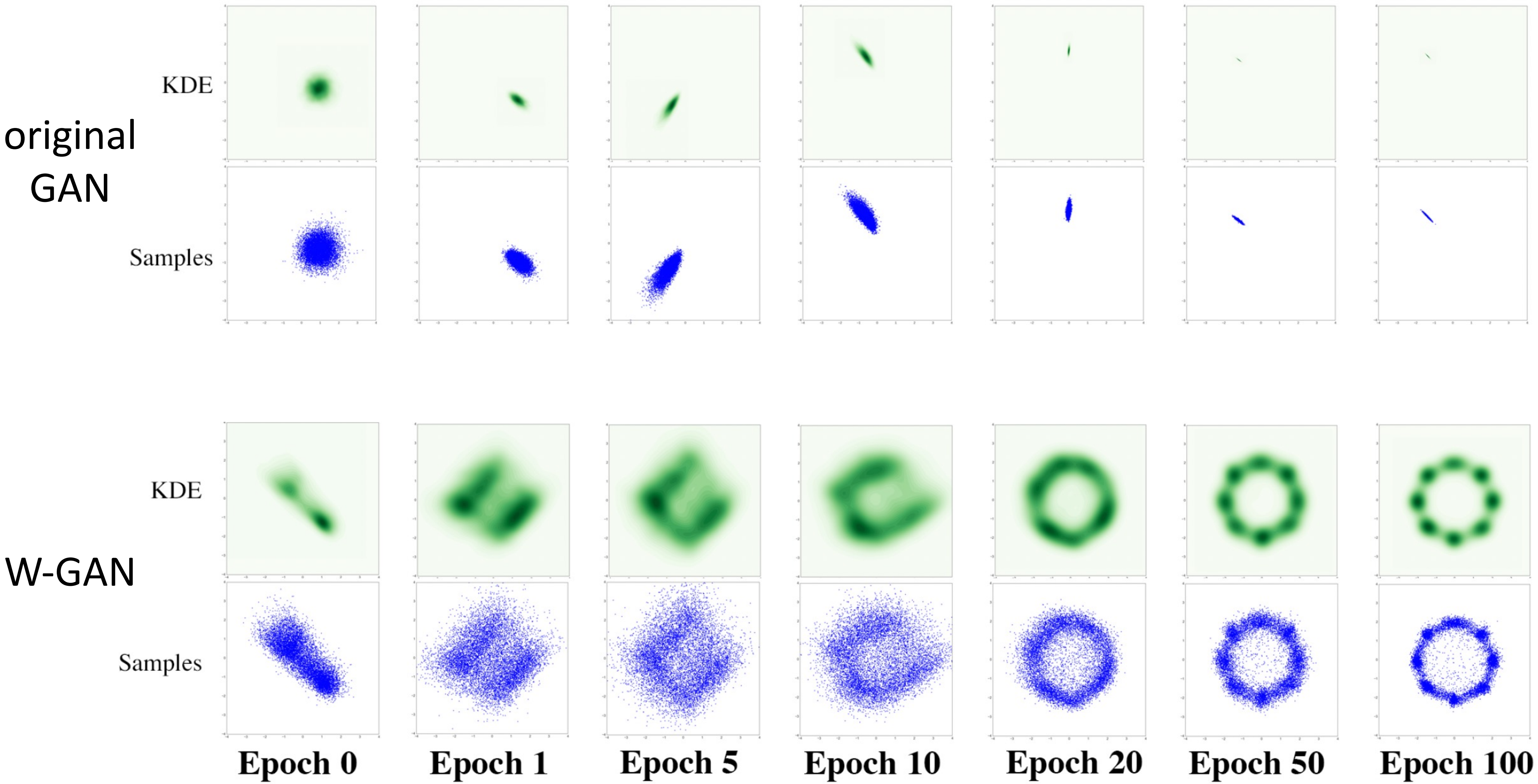
clip weights



W-GAN vs. original GAN



W-GAN vs. original GAN



W-GAN in Short

For mathematicians:

- Wasserstein distance, instead of JS divergence

For engineers:

- remove logarithms Wasserstein distance
- clip weights

  Lipschitz continuity

For laymen:

- art critic instead of a forgery expert

  gradients

Summarized by Reddit user danielvarga: https://www.reddit.com/r/MachineLearning/comments/5qxoaz/comment/dd7aomb/

Brief: LSGAN, EBGAN

- Least Square (LS) GAN:

$$\mathbb{E}_{x\sim p_{\text{data}}}(D(x)-b)^2 \quad + \quad \mathbb{E}_{x\sim p_g}(D(x)-a)^2$$

- Energy-based (EB) GAN:

$$\mathbb{E}_{x\sim p_{\text{data}}}D(x) \quad + \quad \mathbb{E}_{x\sim p_g}[m-D(x)]^+$$

Mao, et al., "Least Squares Generative Adversarial Networks", ICCV 2017
Zhao, et al., "Energy-based Generative Adversarial Networks", ICLR 2017

Adversary as a Loss Function

Adversary as a Loss Function

- GAN essentially defines an **adversarial loss** function

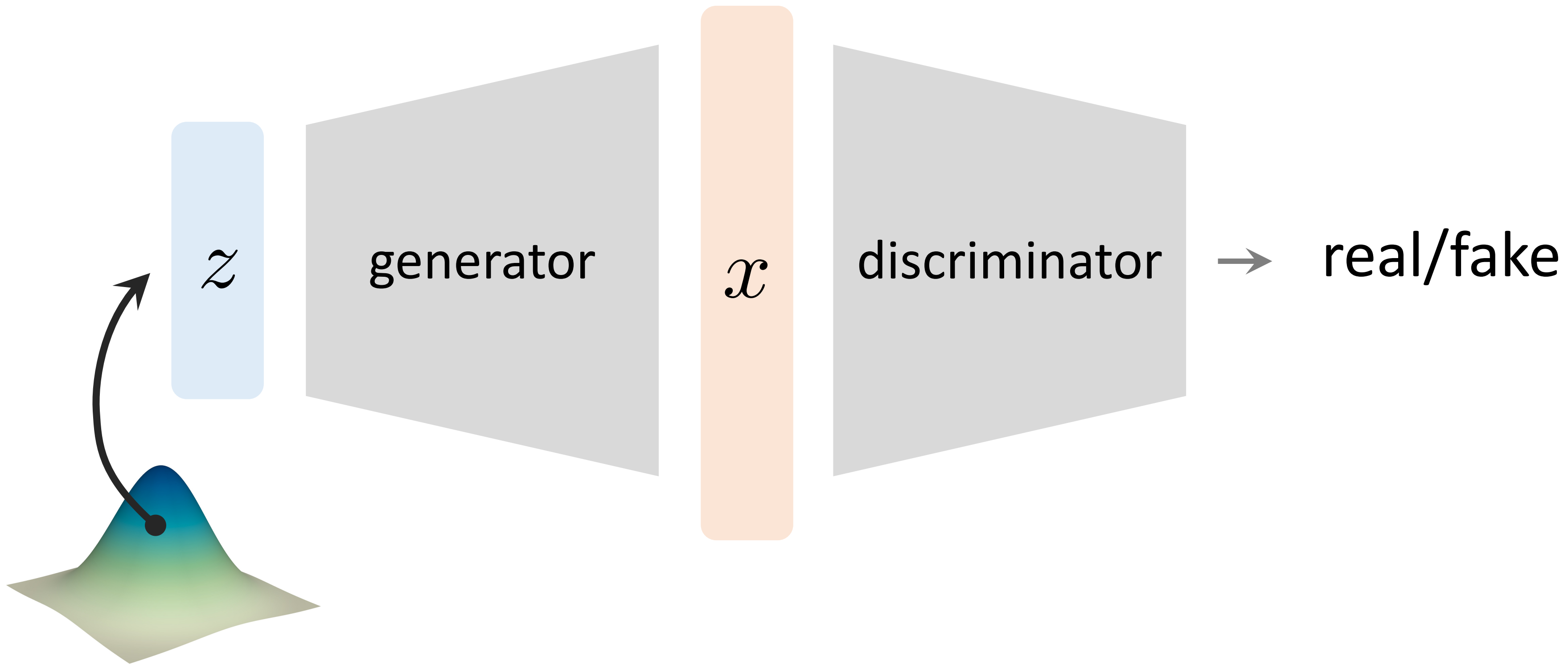
- Input to networks is **not** necessarily random/noise

- **Beyond L2/L1**: adversarial loss encourages output to look “realistic”

- **Combined with L2/L1**: reconstruction loss largely stabilizes training

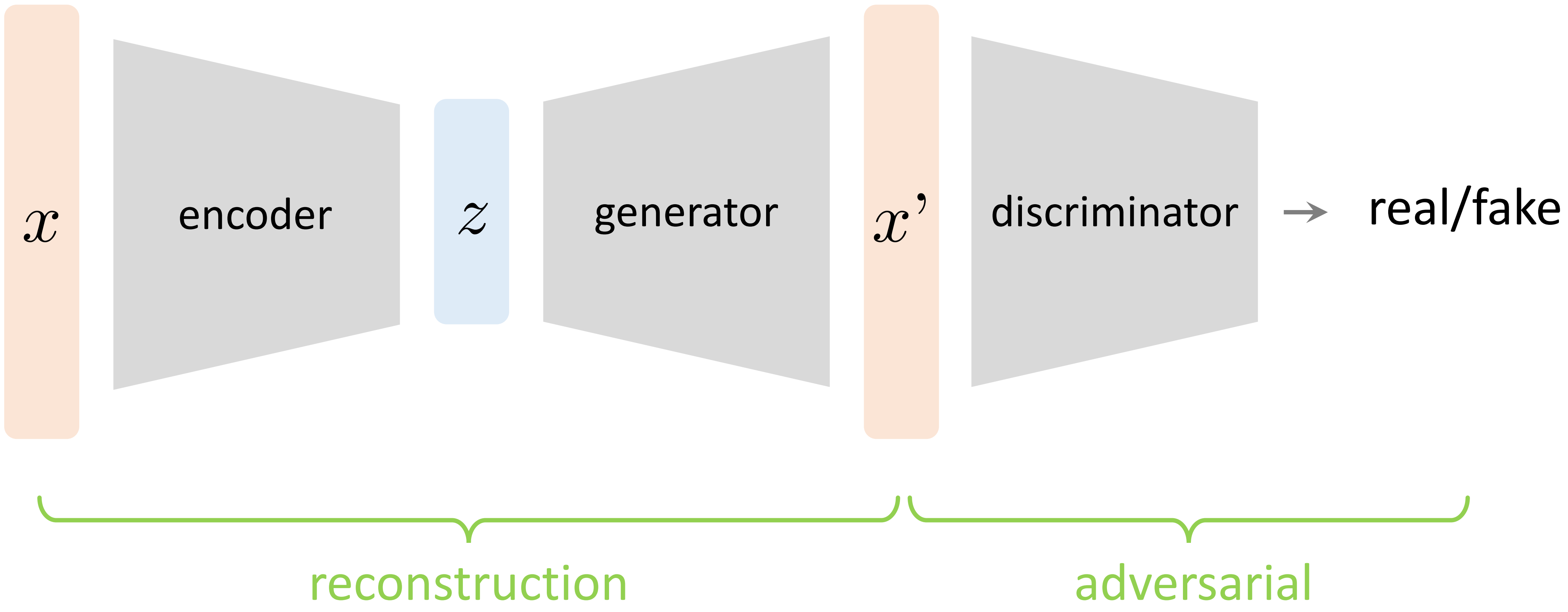
Adversary as a Loss Function

- GAN: input is random



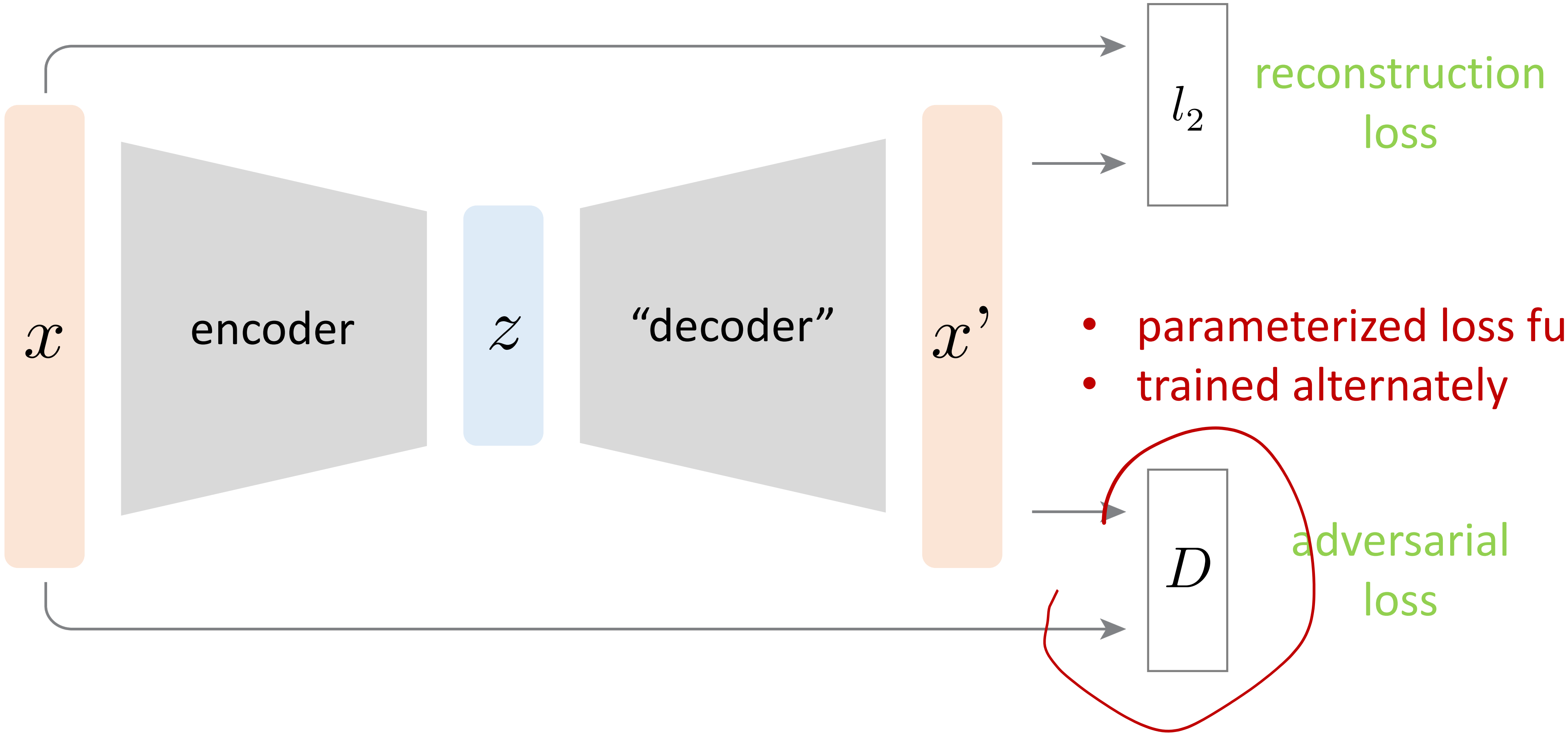
Adversary as a Loss Function

- Input can be from another source



Adversary as a Loss Function

- Input can be from another source



- parameterized loss function
- trained alternately

Example: Super-Resolution GAN

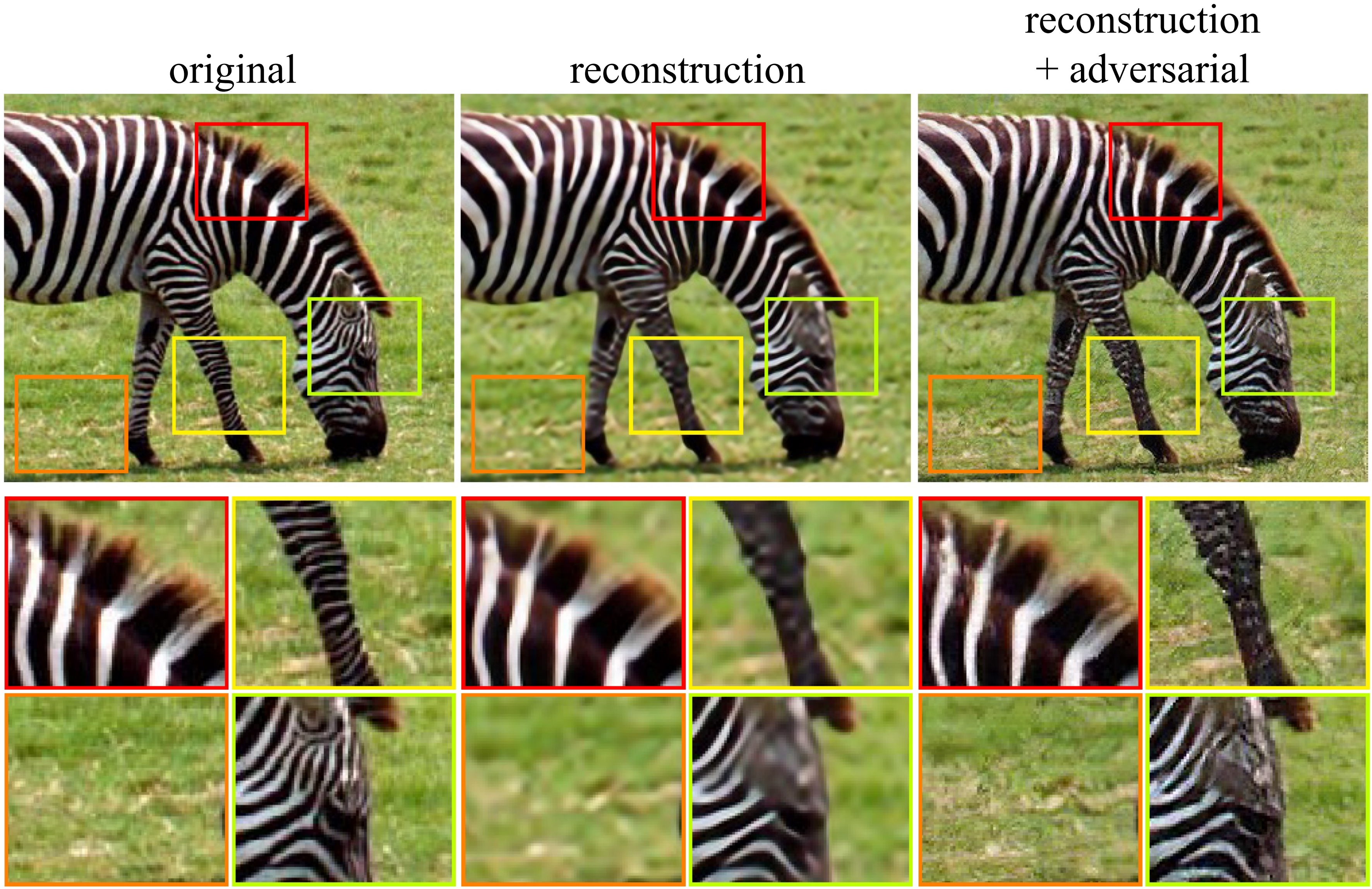
- worse PSNR, but better visual quality

- better PSNR

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

Ledig, et al., "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", CVPR 2016

Example: Super-Resolution GAN

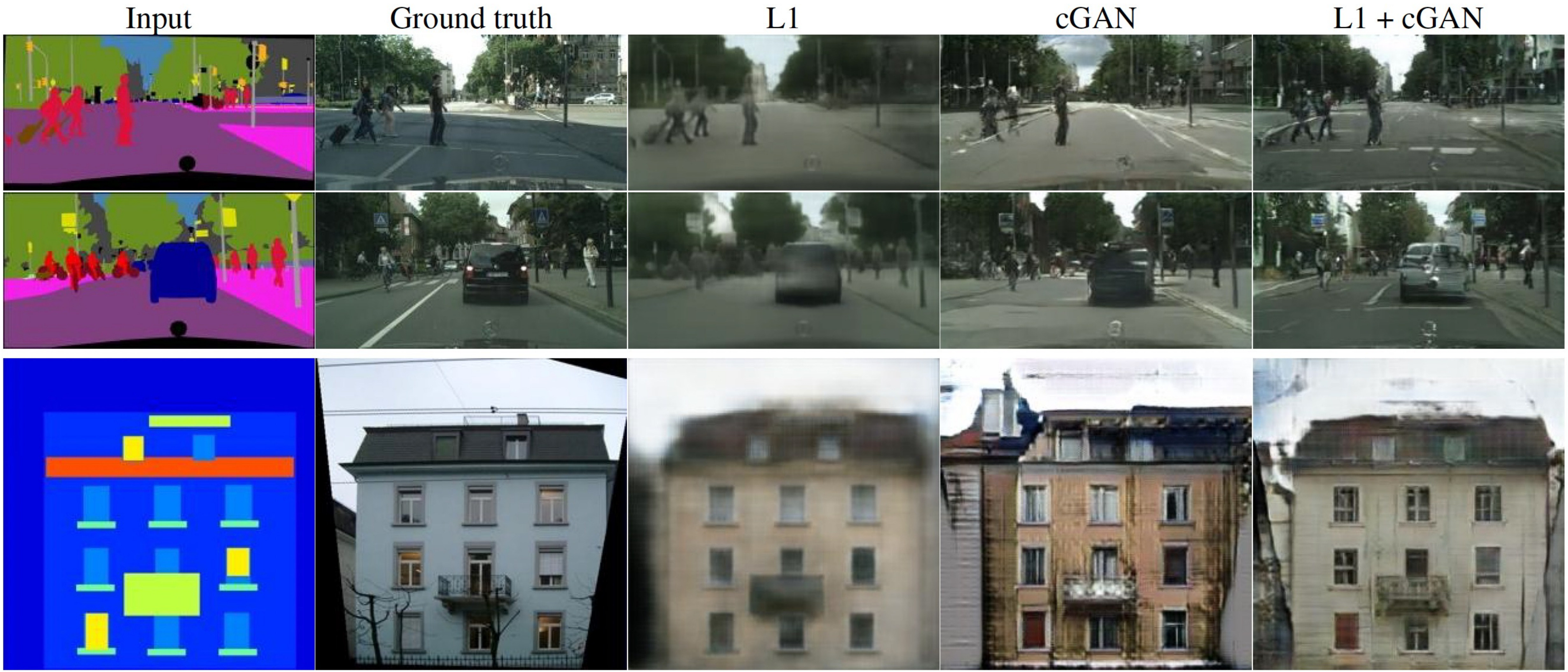


Ledig, et al., “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, CVPR 2016

Example: Context Encoder

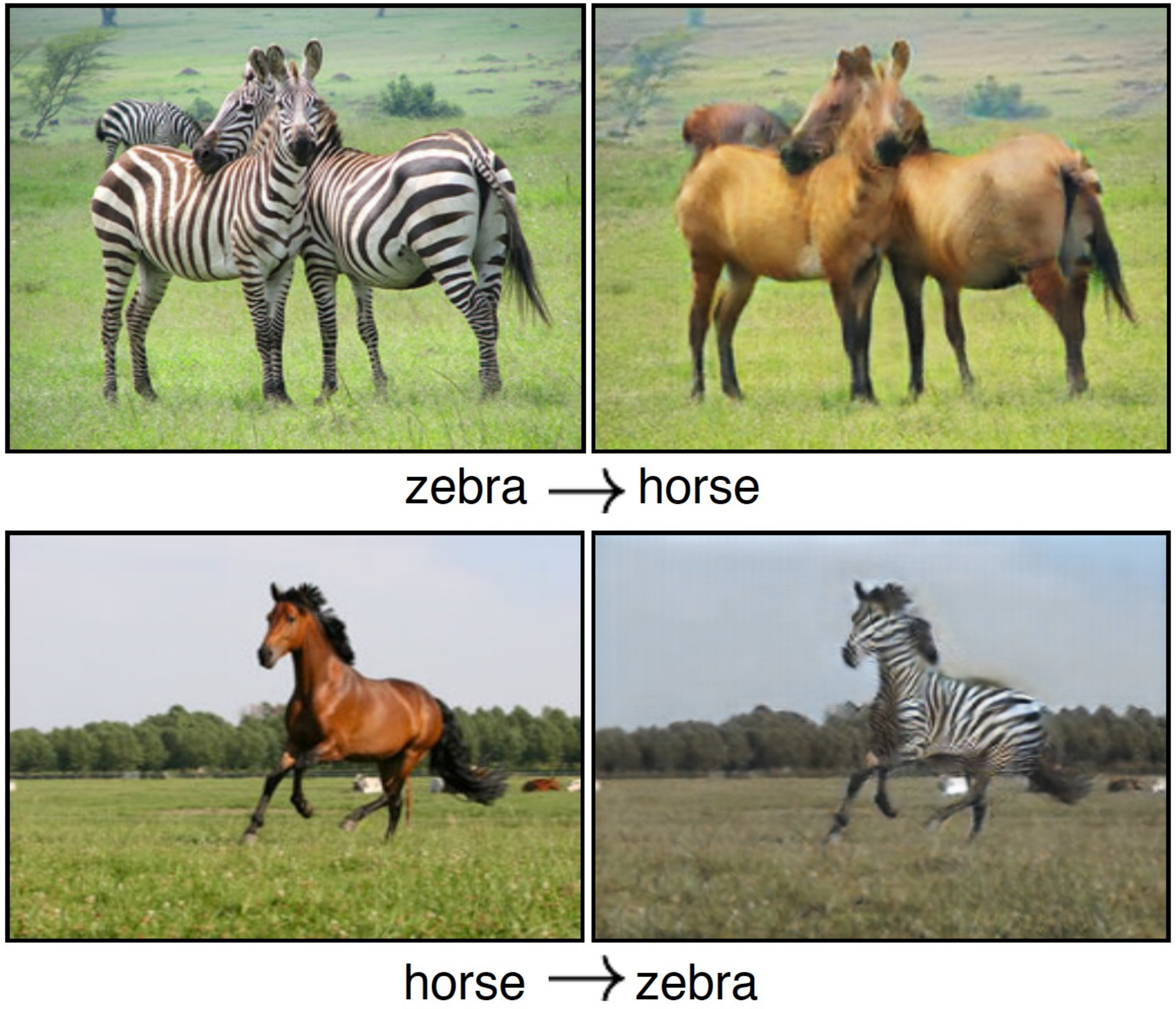
Pathak, et al., "Context Encoders: Feature Learning by Inpainting", CVPR 2016

Example: pix2pix



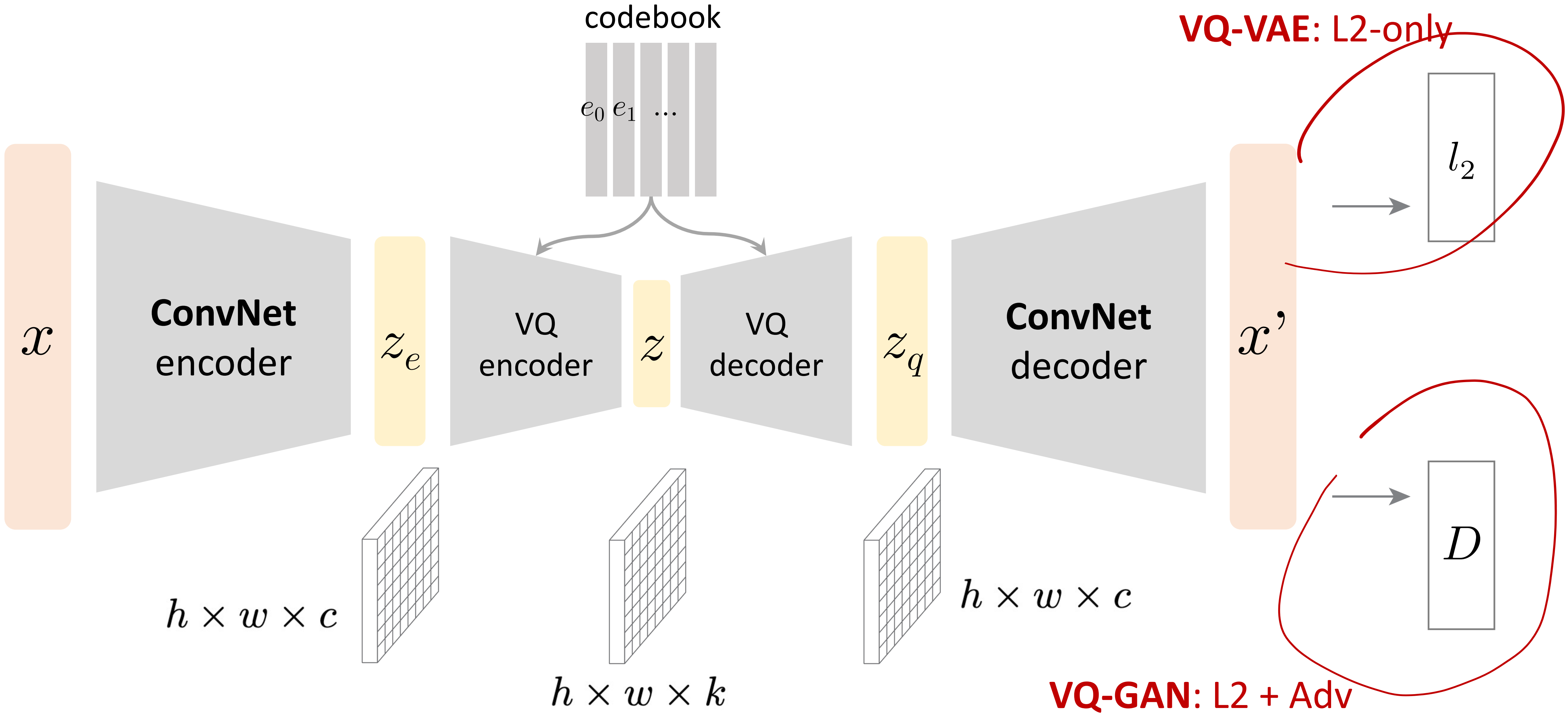
Isola, et al., "Image-to-Image Translation with Conditional Adversarial Networks", CVPR 2017

Example: CycleGAN



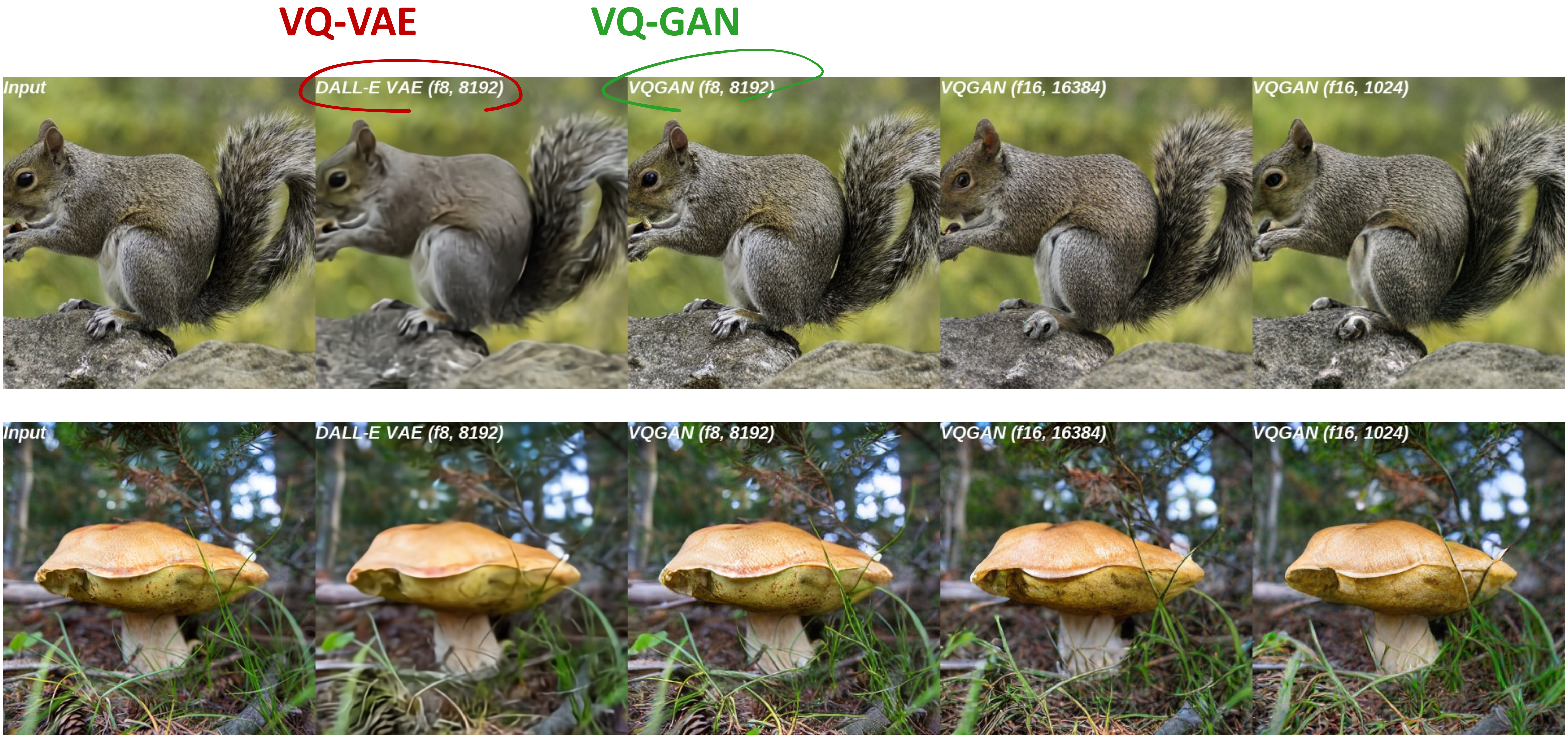
Zhu, et al., “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”, ICCV 2017

From VQ-VAE to VQ-GAN



Esser, et al., "Taming Transformers for High-Resolution Image Synthesis", CVPR 2021

From VQ-VAE to VQ-GAN



Esser, et al., “Taming Transformers for High-Resolution Image Synthesis”, CVPR 2021

Discussion

- To be precise: **VQ-GAN** = **VQ-VAE** + Adv Loss + Perceptual Loss
- w/o VQ, it's **VAE** + Adv Loss + Perceptual Loss
- Both are the *de facto* **tokenizers** in image generation
  - w/ VQ: e.g., Autoregressive Models
  - w/o VQ: e.g., Diffusion Models
- Commercial models (e.g., Stable Diffusion, Sora) use these tokenizers

It involves everything!

This Lecture

- Generative Adversarial Networks (GAN)
- Wasserstein GAN (W-GAN)
- Adversary as a Loss Function

Main References

- Goodfellow et al. “Generative Adversarial Nets”, NeurIPS 2014
- Arjovsky et al. “Wasserstein Generative Adversarial Networks”, ICML 2017