

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)

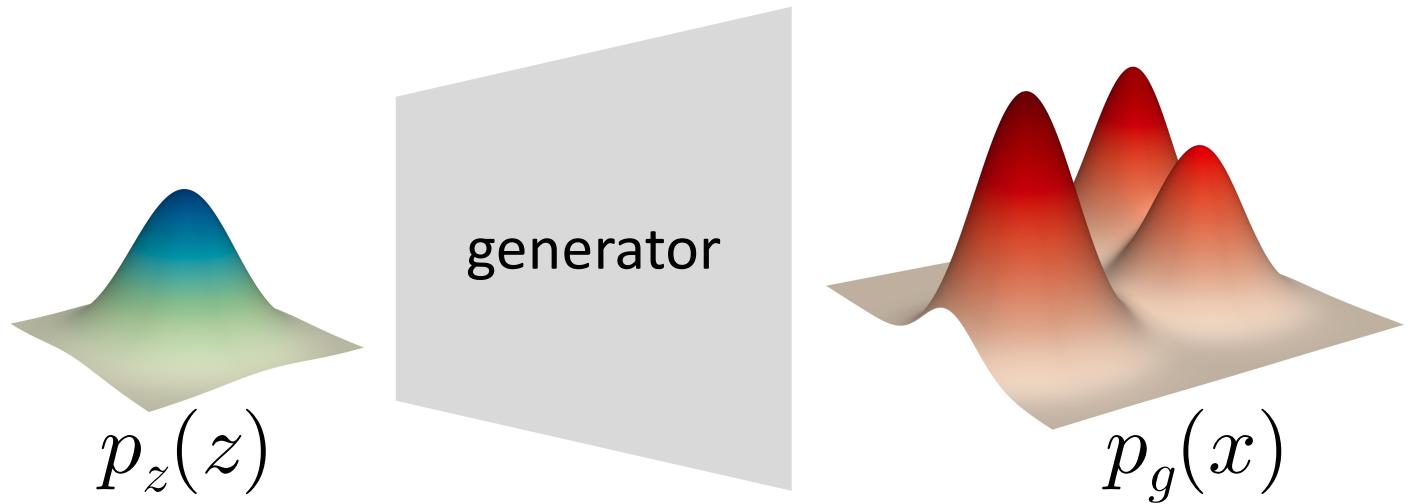
Introduction

- “**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:

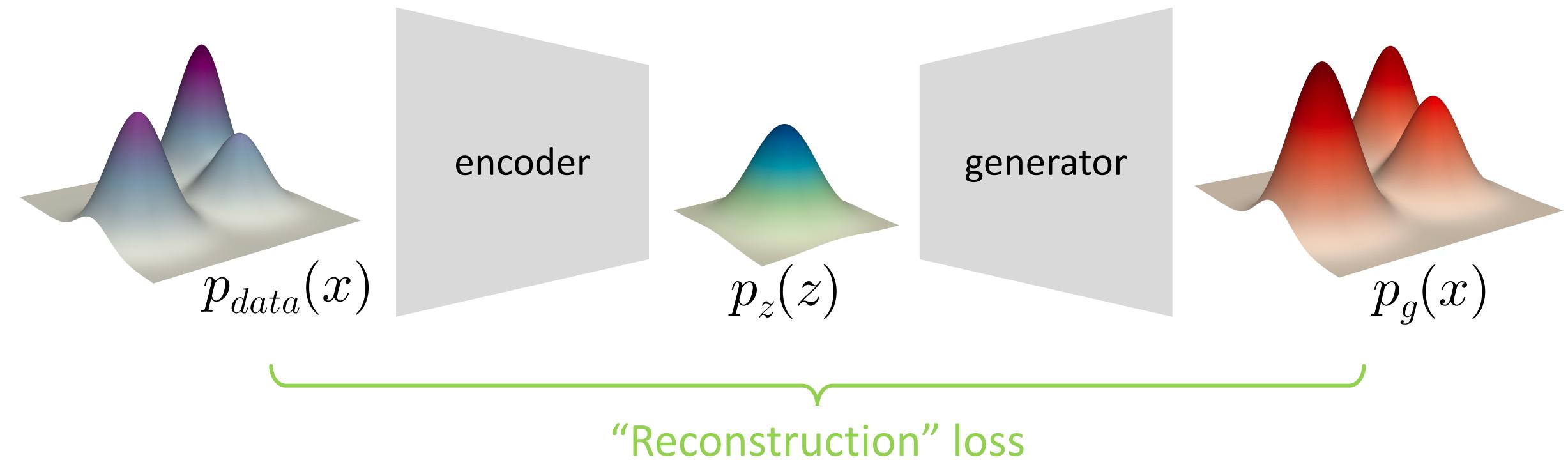
- z - latent variables
- x - observed variables



Recap: Variational Autoencoder (VAE)

Autoencoding distributions:

“Encoding” data distribution p_{data} into latent distribution p_z



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

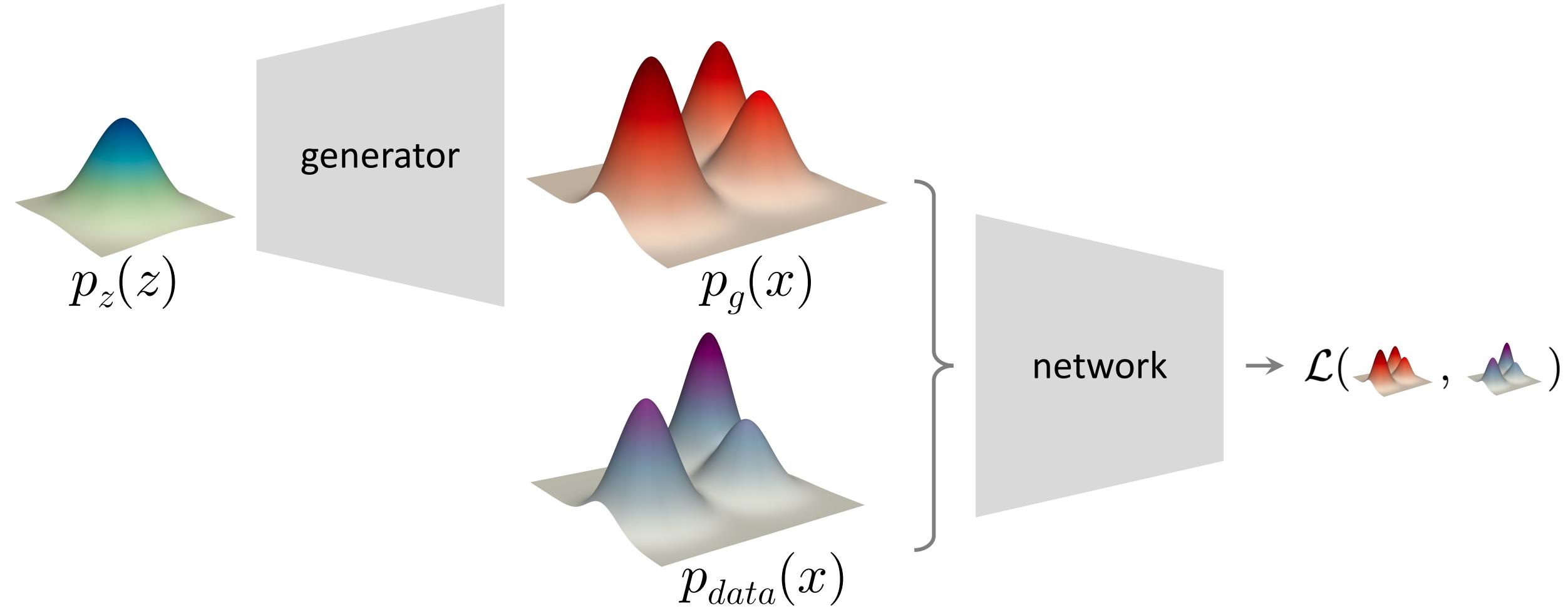
- Elements (e.g., pixels) are **independently** distributed
- Each element follows a **simple** distribution (Gaussian/Bernoulli/...)

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

Can we measure the distribution difference in another way?

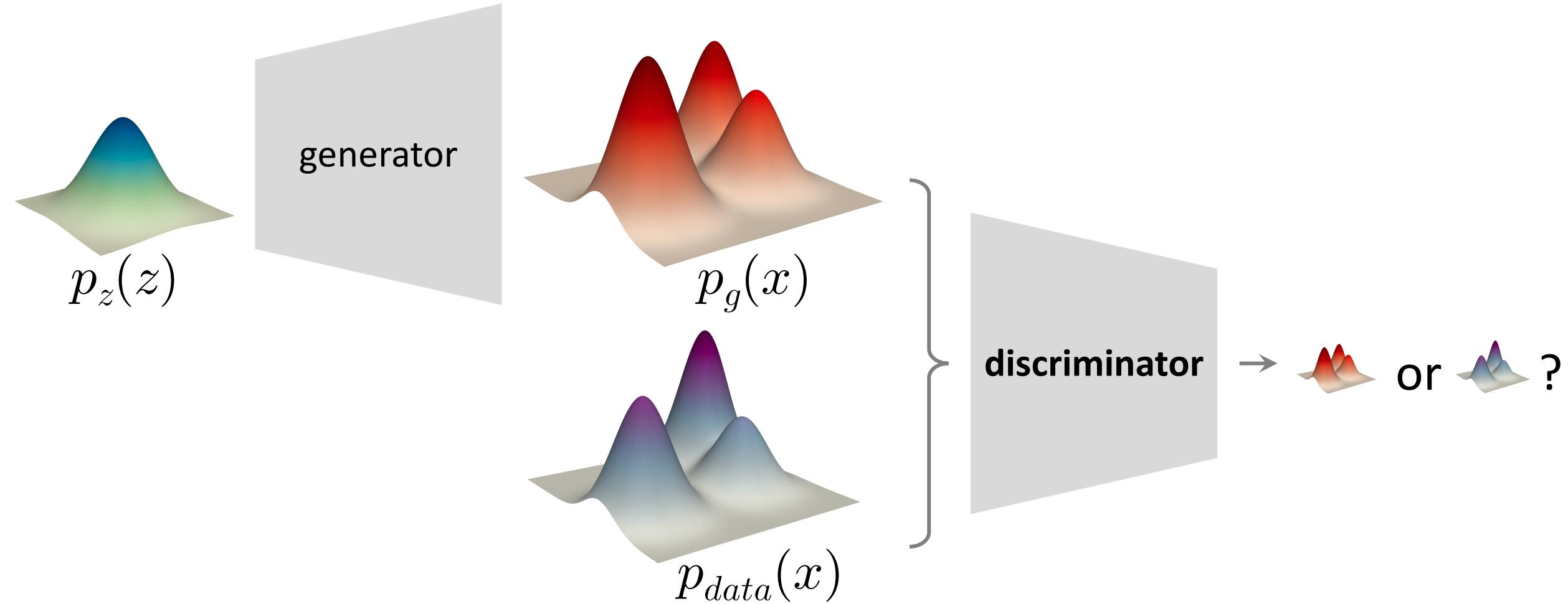
Generative Adversarial Networks

Representing **distribution difference** by a neural network



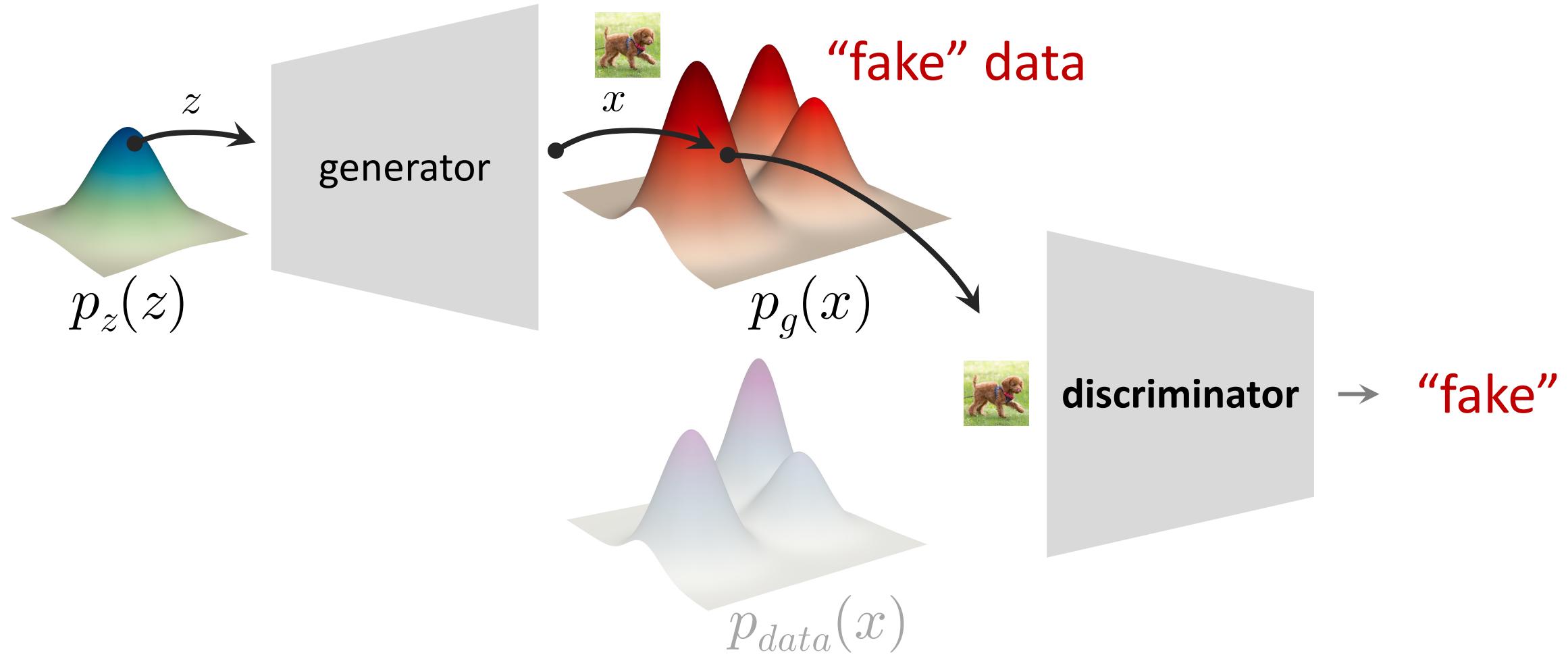
Generative Adversarial Networks

Representing **distribution difference** by a neural network



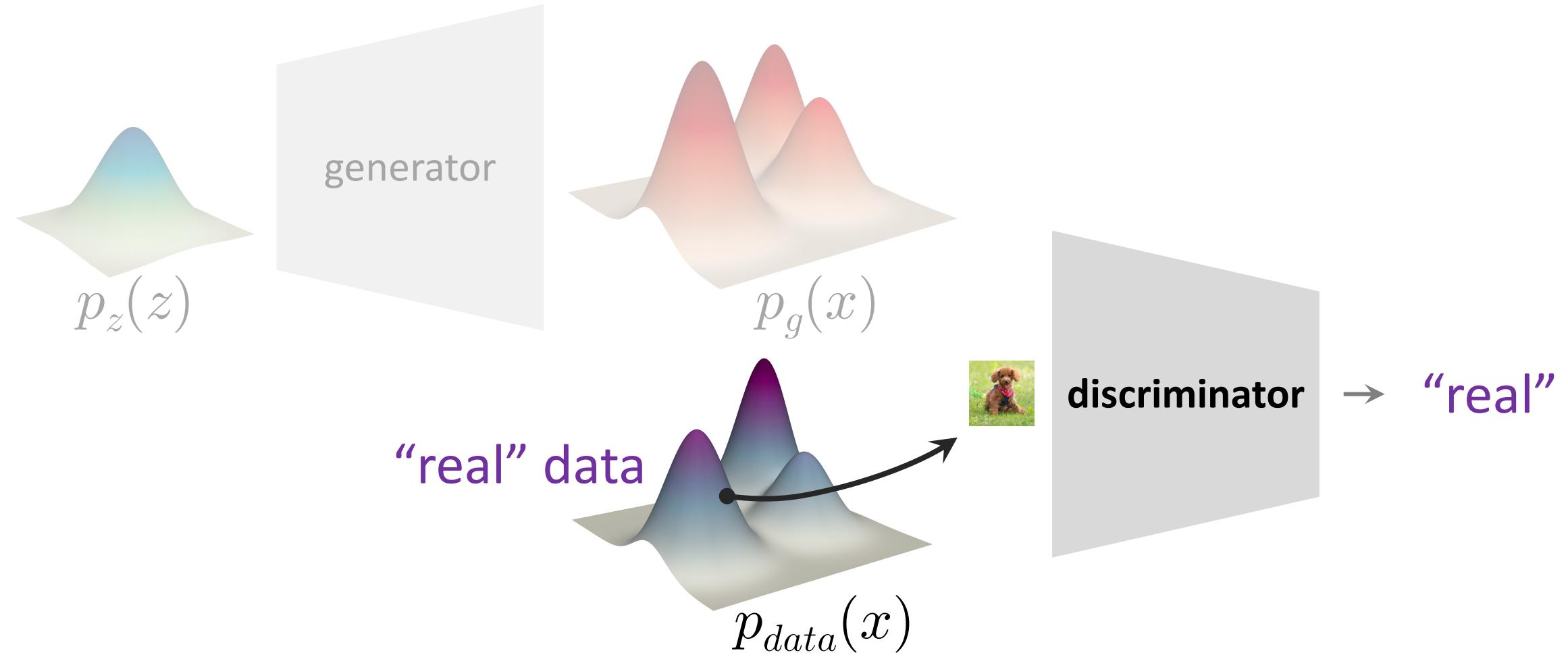
Generative Adversarial Networks

Representing **distribution difference** by a neural network

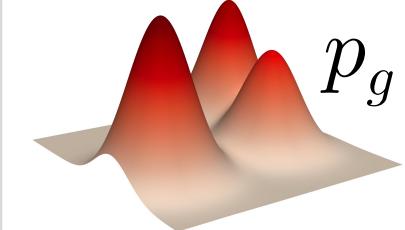
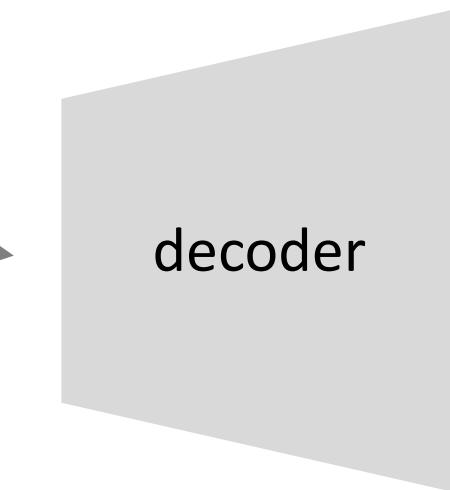
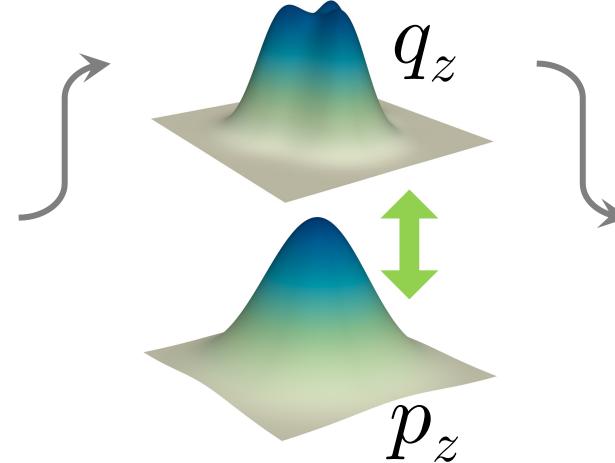
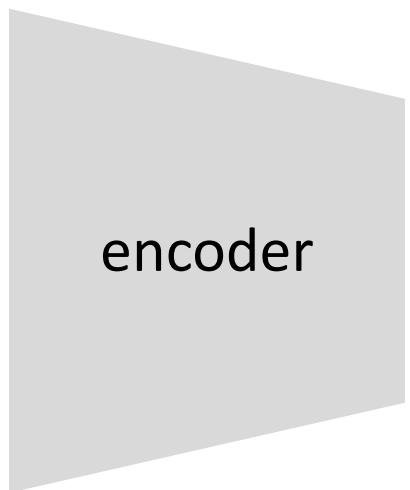
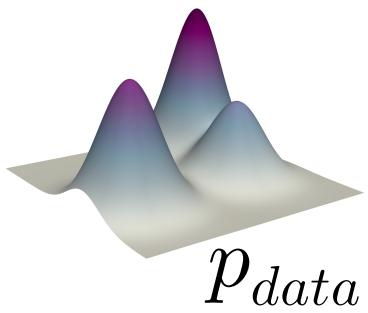


Generative Adversarial Networks

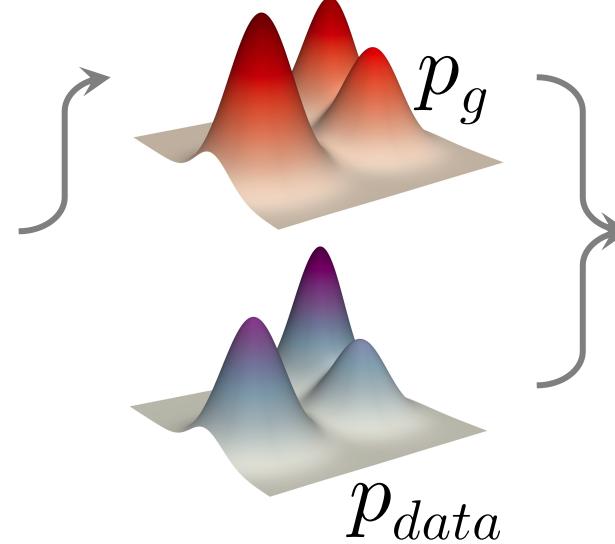
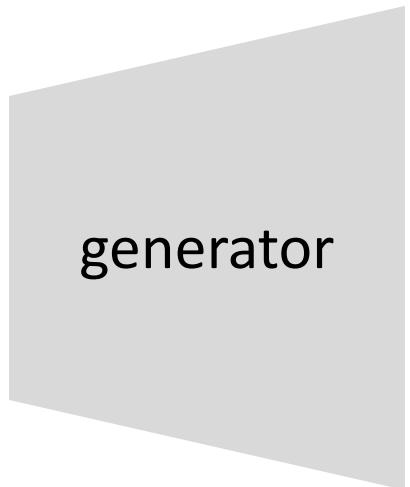
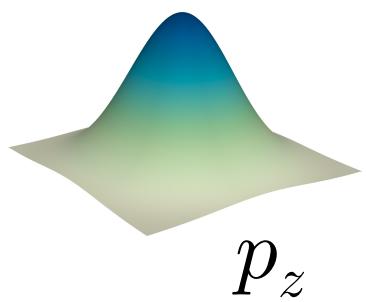
Representing **distribution difference** by a neural network



VAE



GAN



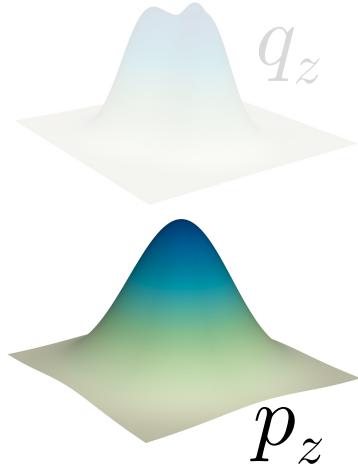
or ?

VAE

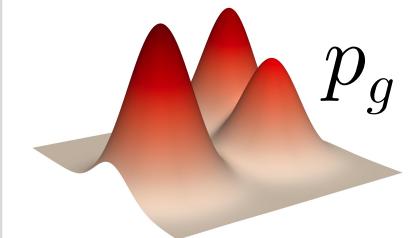
generation



encoder

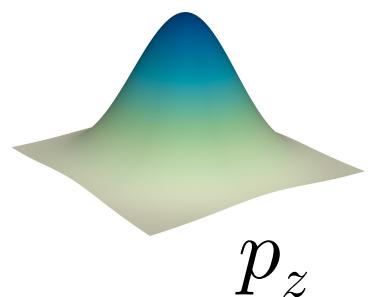


decoder

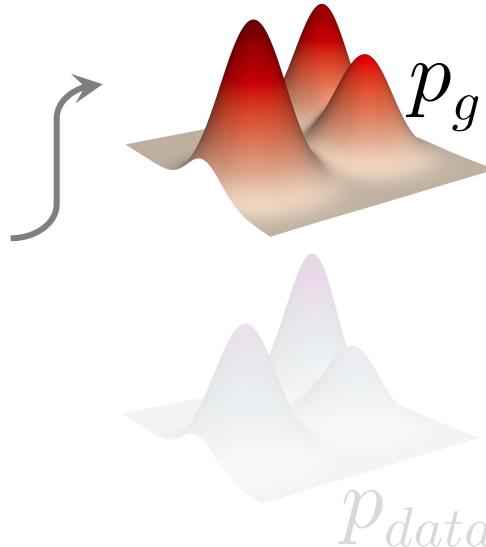


GAN

generation



generator



discriminator

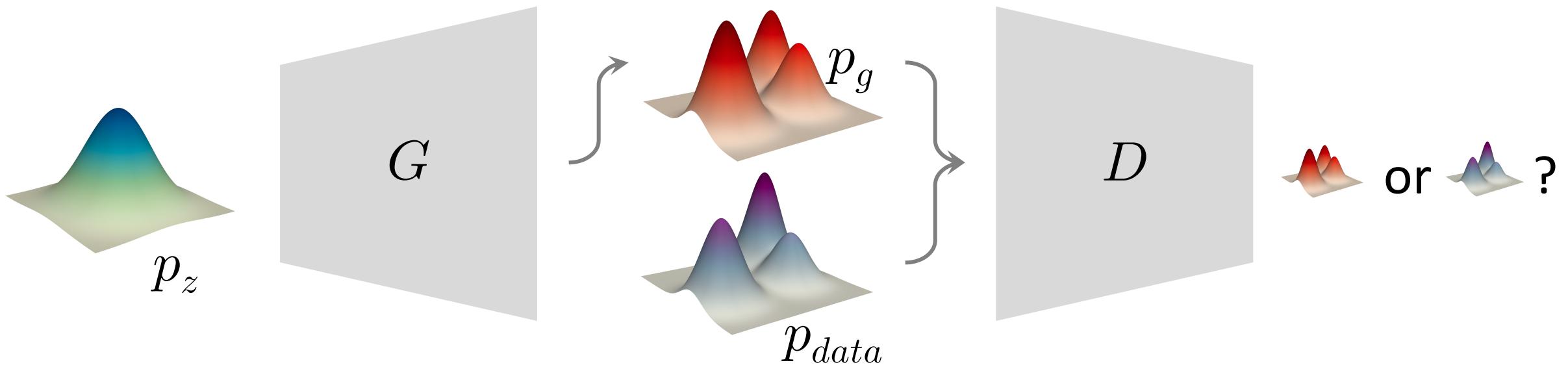
or ?

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

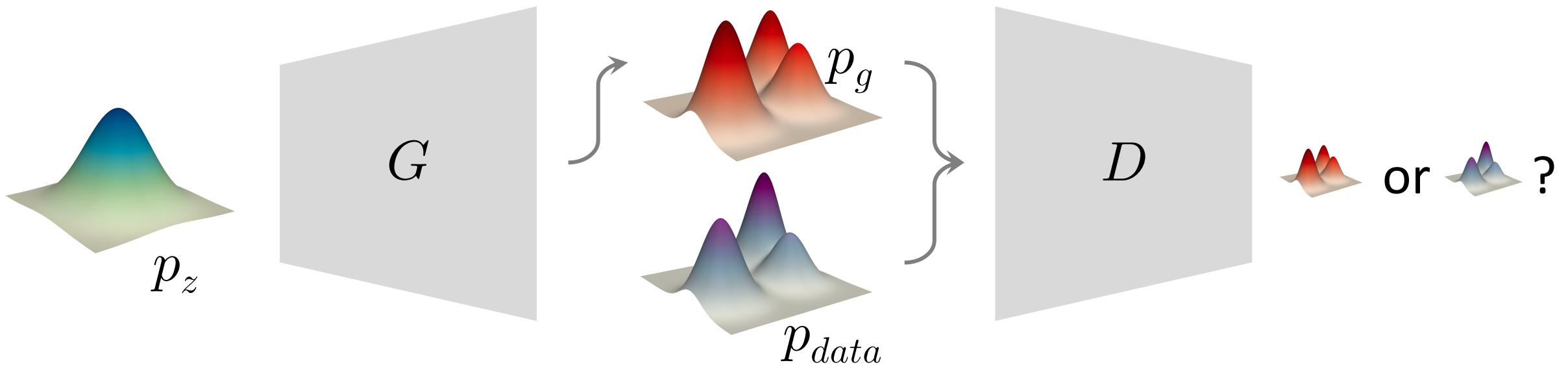
(vs. EM's max-max process)



Adversarial Objective: D-step

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

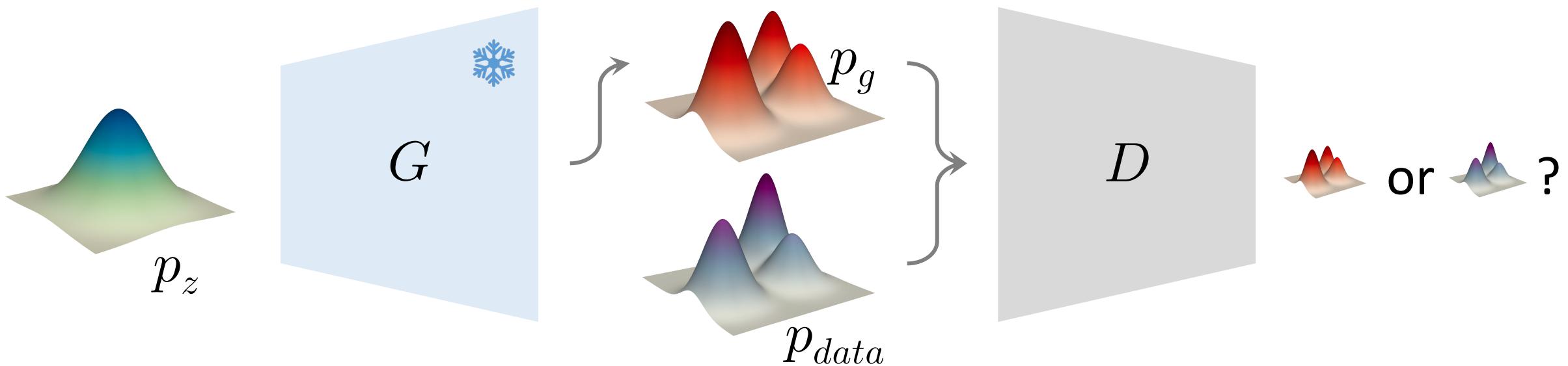
D-step: fix G , optimize D



Adversarial Objective: D-step

D-step: fix G , optimize D

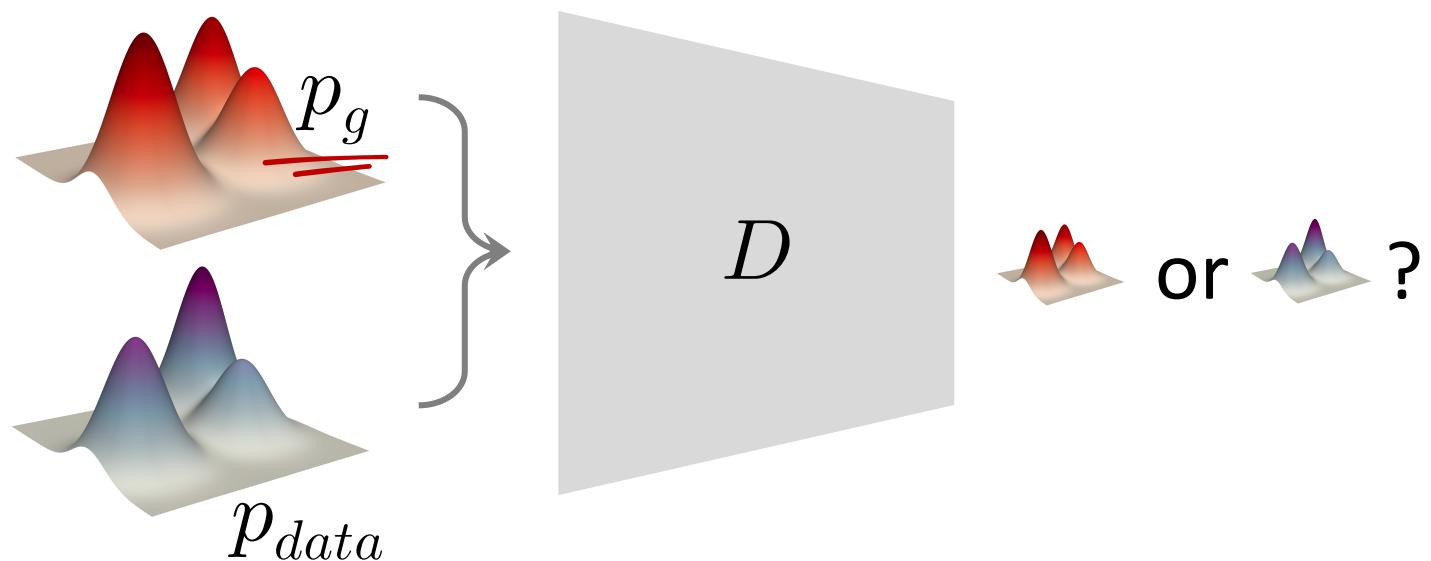
- D to classify real or fake
 - binary logistic regression (sigmoid + cross-entropy)



Adversarial Objective: D-step

D-step: fix G , optimize D

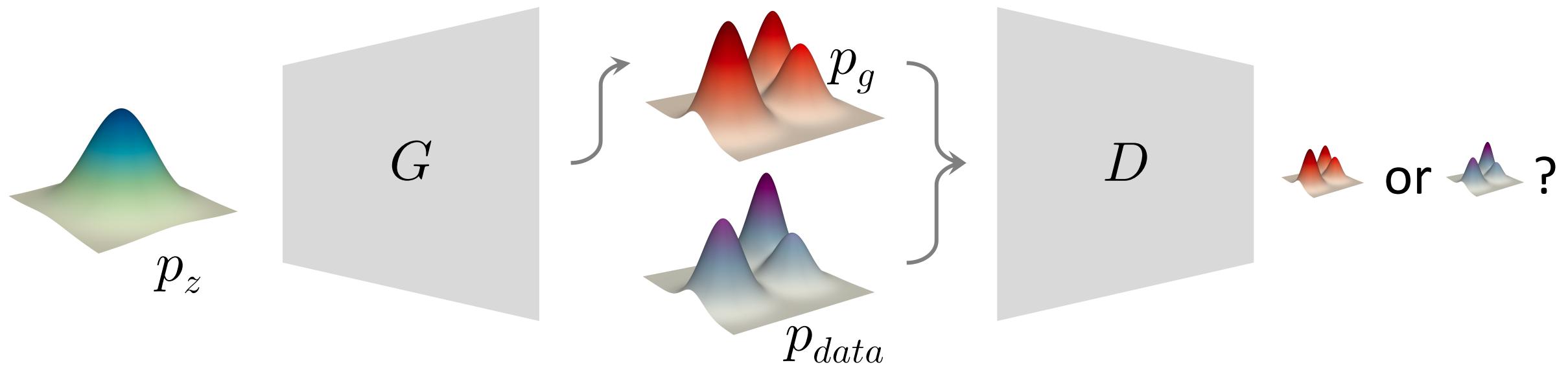
- D to classify real or fake
 - binary logistic regression (sigmoid + cross-entropy)



Adversarial Objective: G-step

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

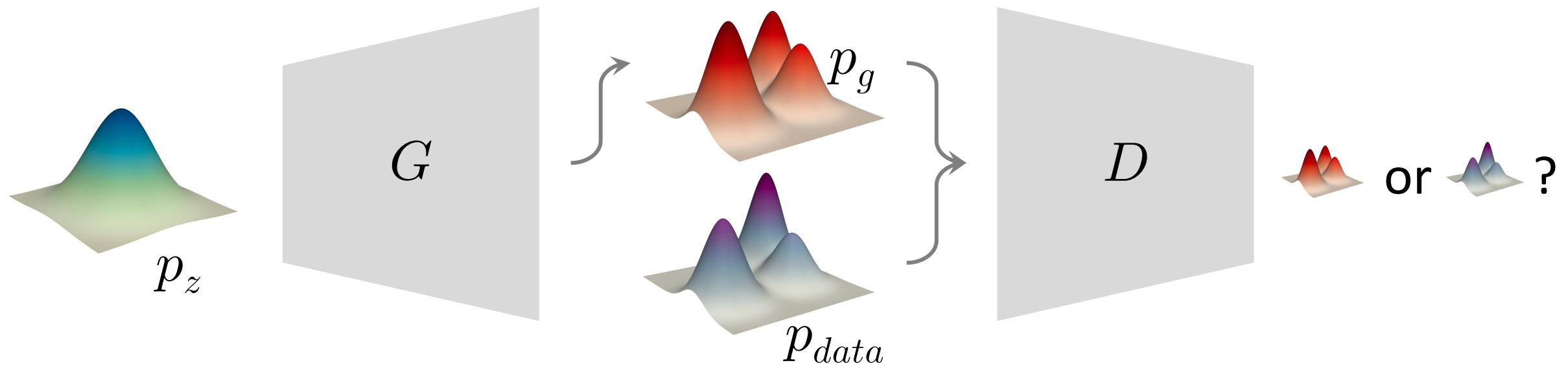
G-step: fix D , optimize G



Adversarial Objective: G-step

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

G-step: fix D , optimize G



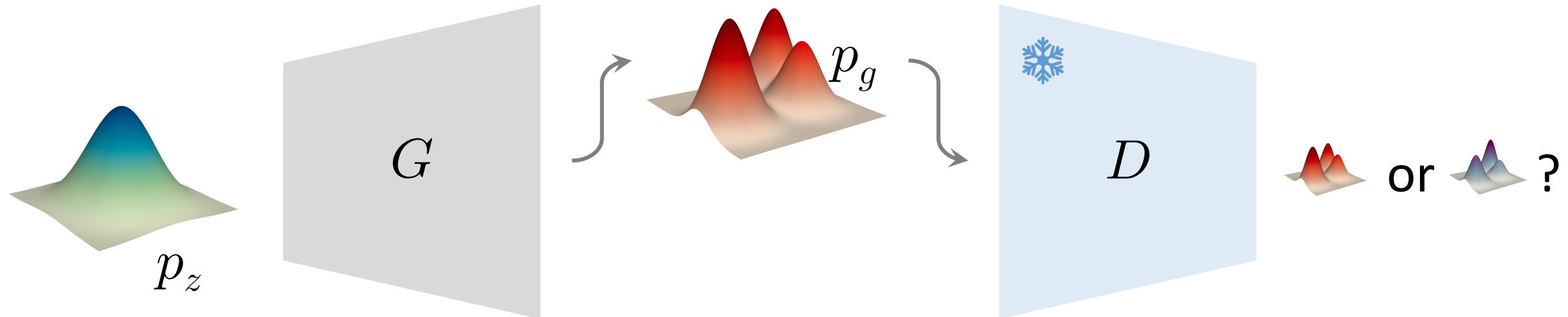
Adversarial Objective: G-step

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

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

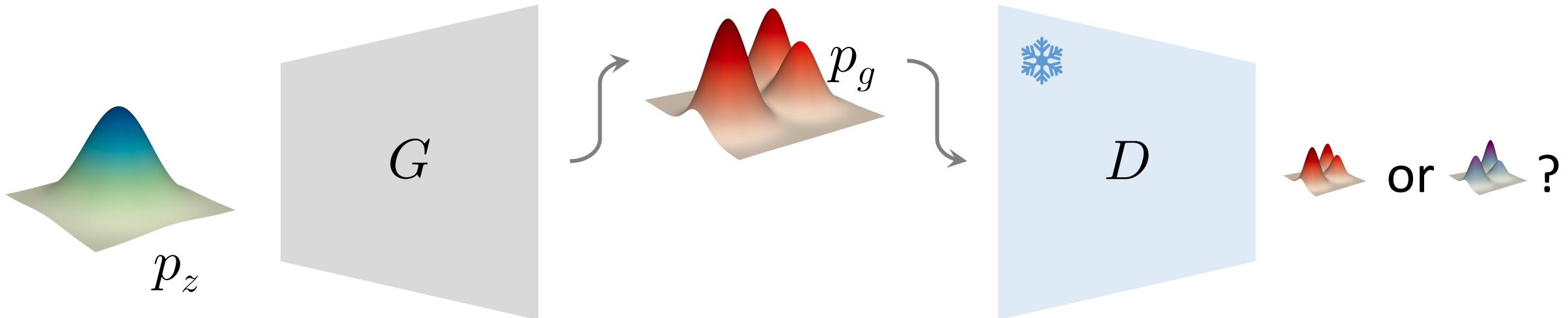
a “flip” trick:

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

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

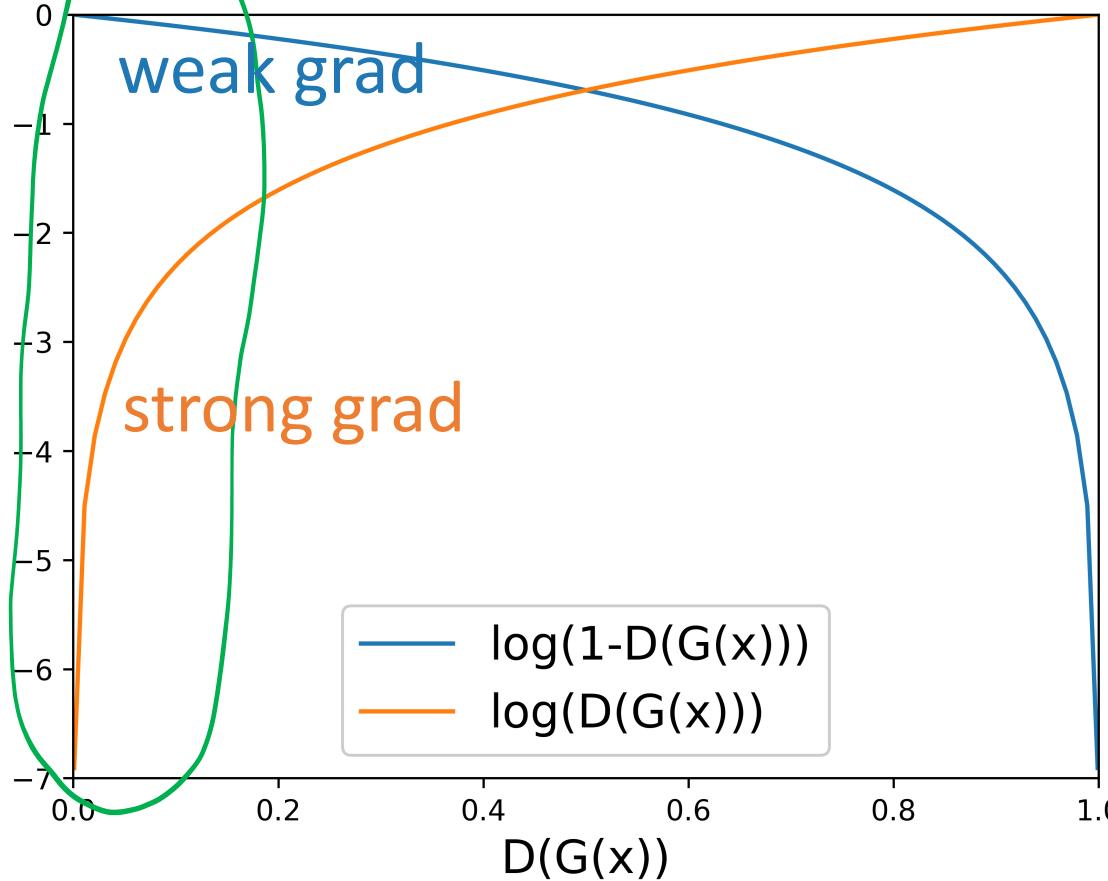
a “flip” trick:

$$\max_G \mathbb{E}_{z \sim p_z} [\log(\frac{1}{1 - D(G(z))})]$$

push to 1

Early in training:

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



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

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

end for

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

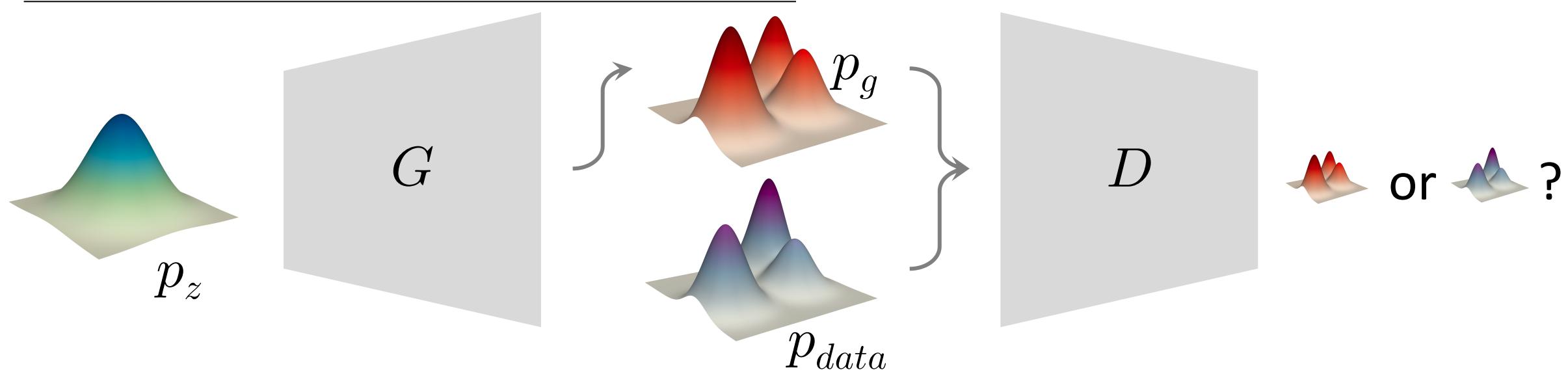
- Update the generator by descending its stochastic gradient:

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

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

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

end for

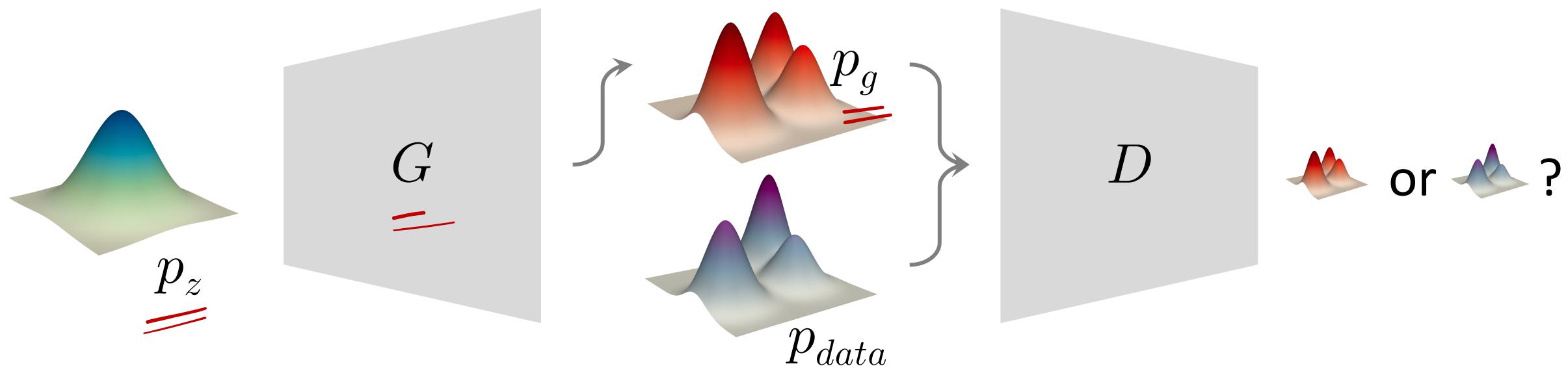
- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

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

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

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

end for

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

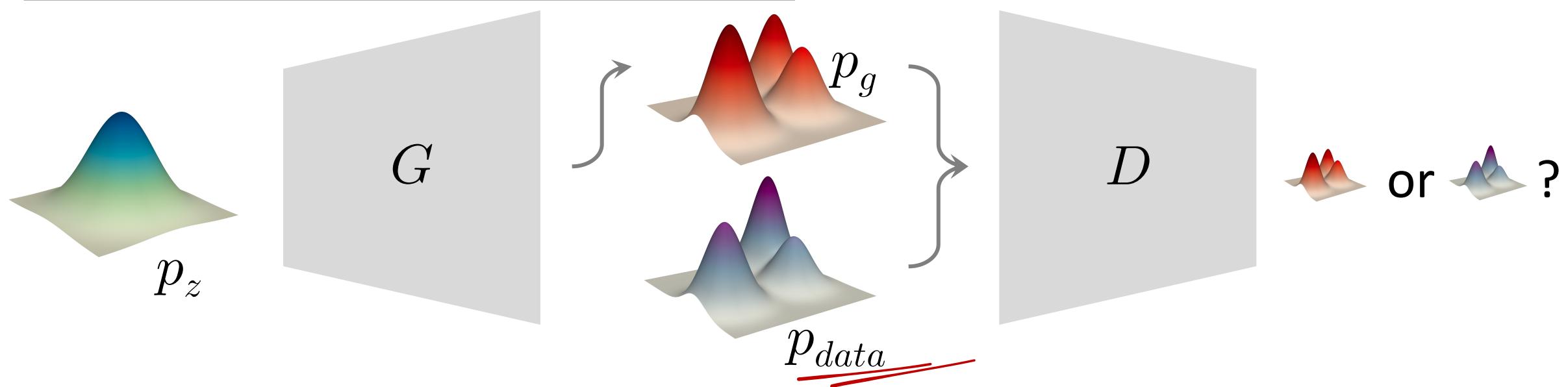
- Update the generator by descending its stochastic gradient:

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

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 $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.

Update the discriminator by ascending its stochastic gradient:

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

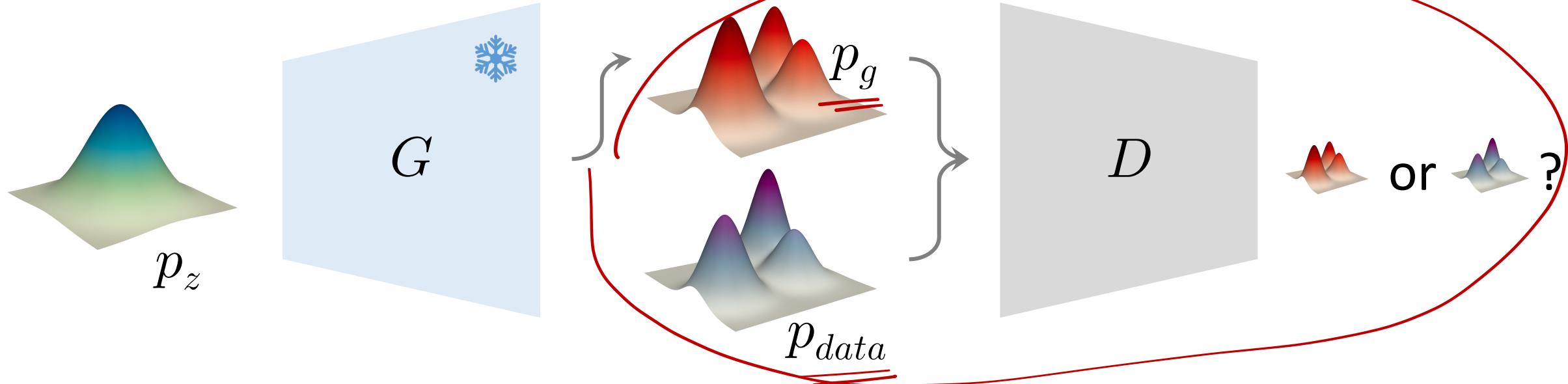
end for

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

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

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

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

end for

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

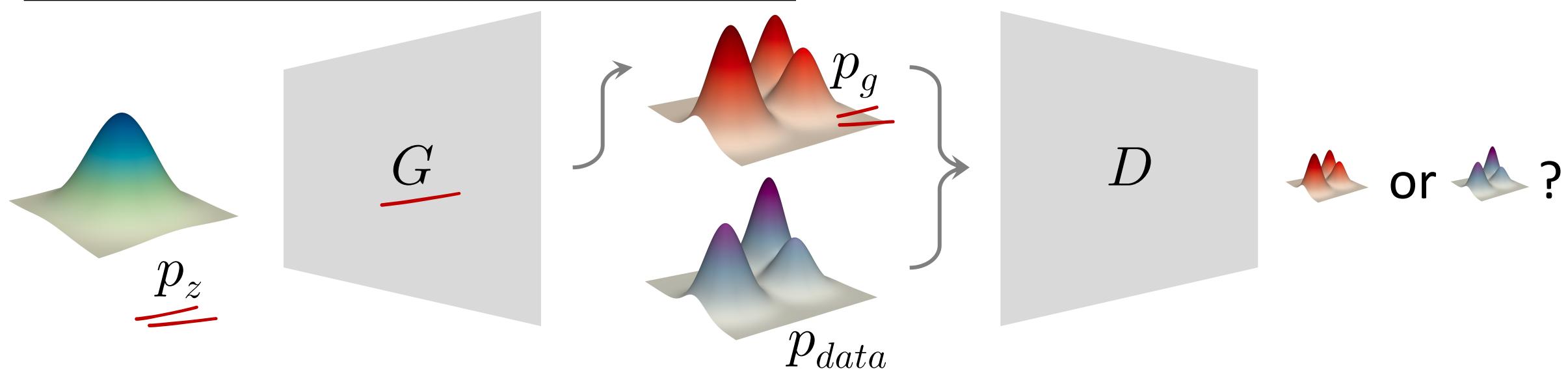
- Update the generator by descending its stochastic gradient:

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

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

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

end for

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

G-step

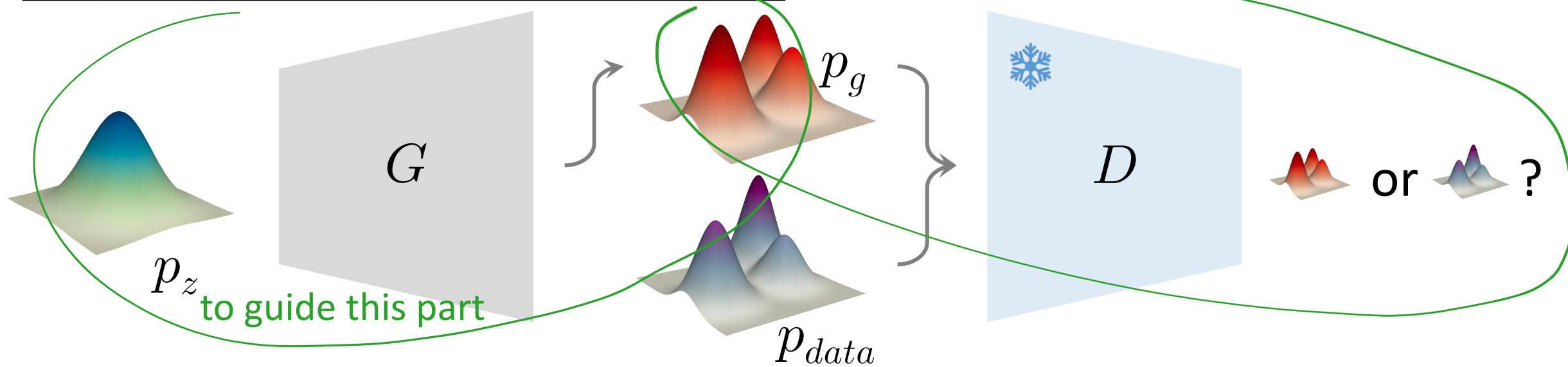
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^{(i)}))).$$

gradient descend
(minimize)

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

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

end for

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

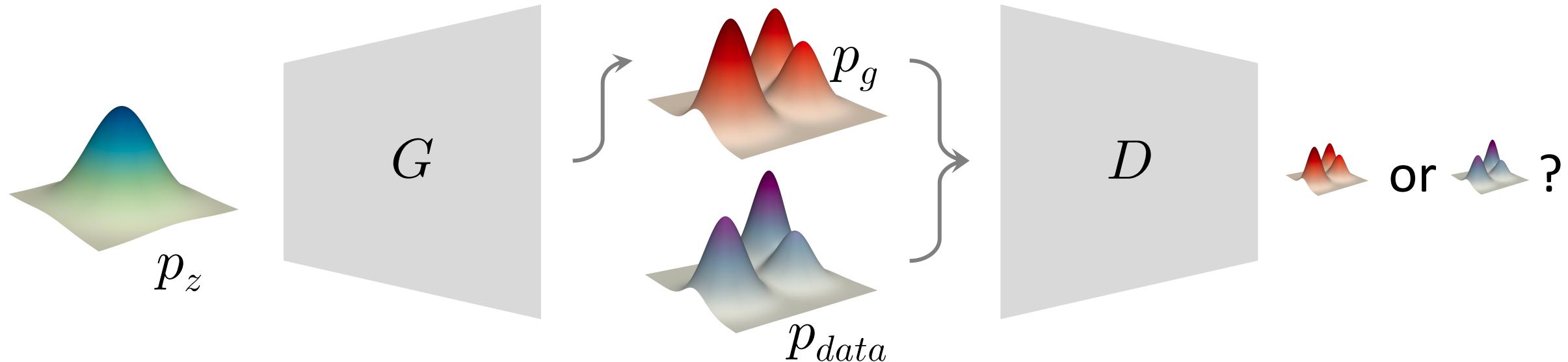
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^{(i)}))).$$

end for

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

GAN algorithm annotated

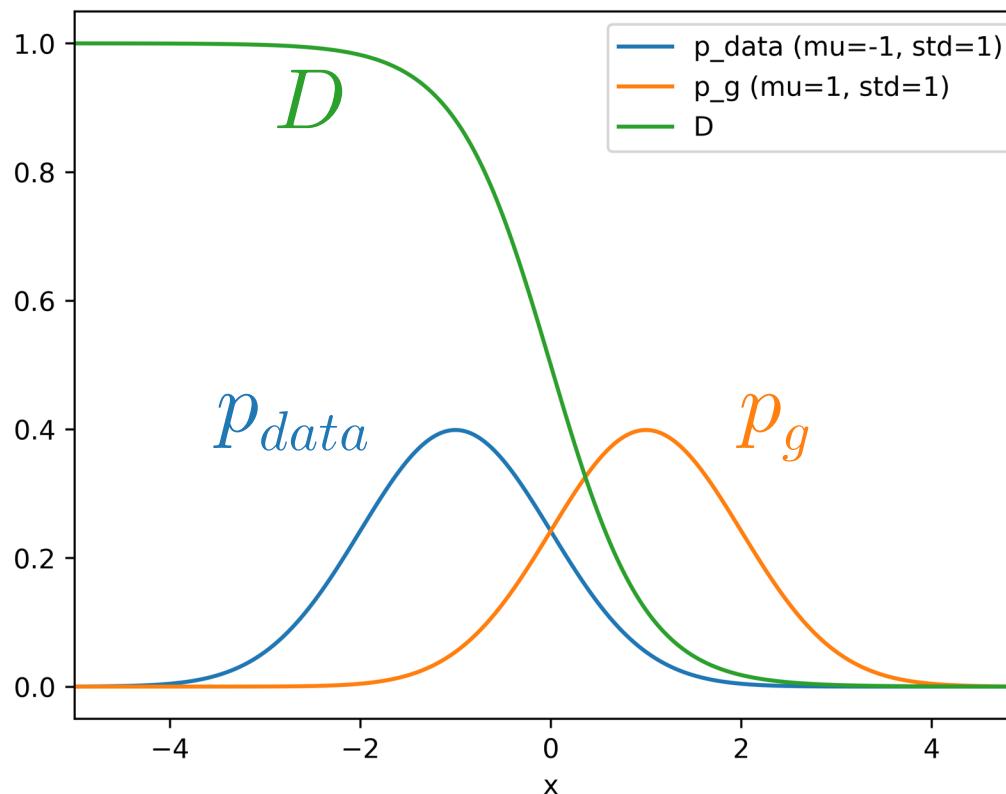
iterating
min-max



Theoretical Results

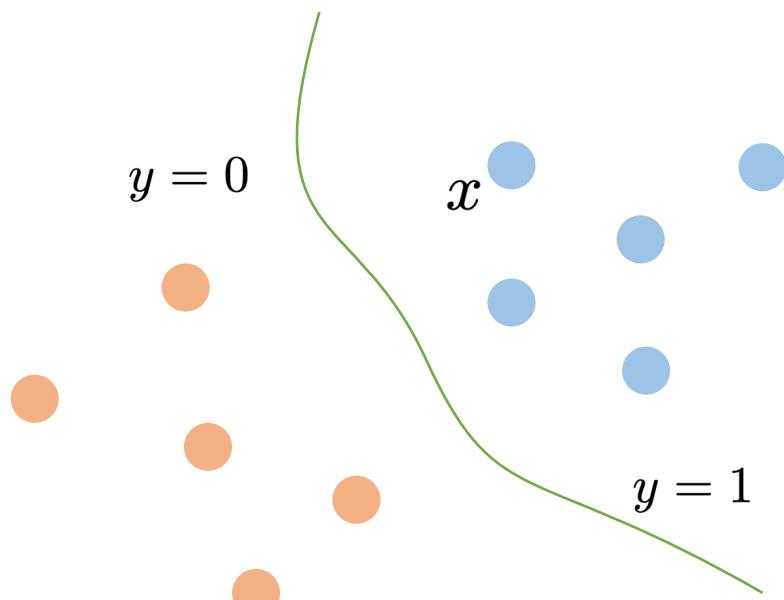
1. For any given G , the optimal D is:

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

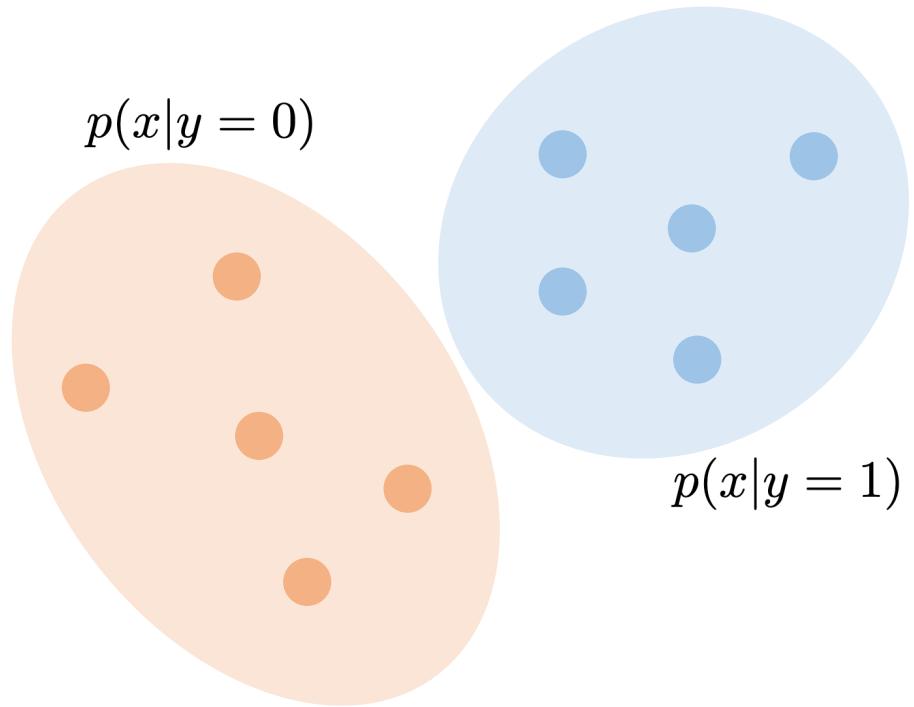


Recap (Lec. 1): Discriminative vs. Generative

discriminative



generative



Theoretical Results

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

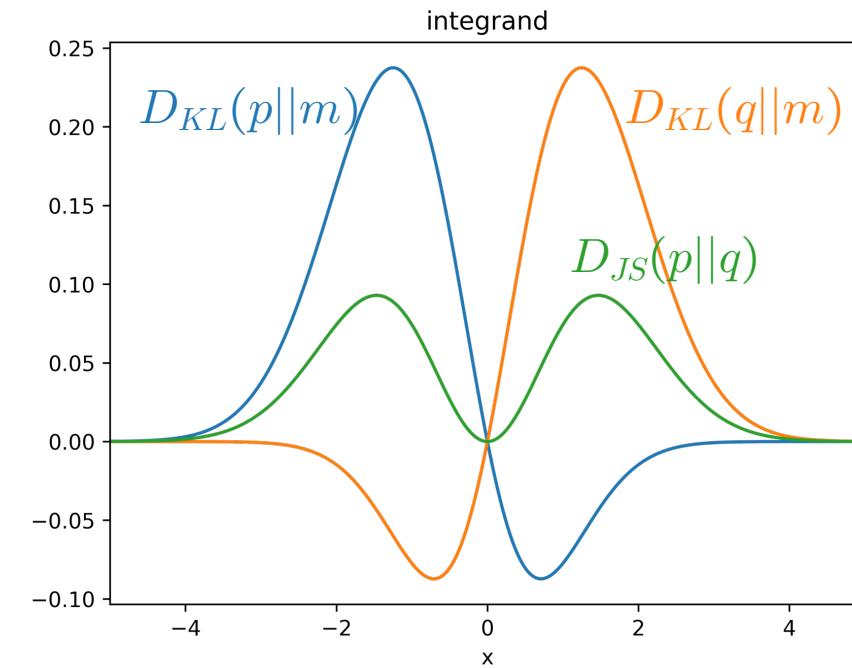
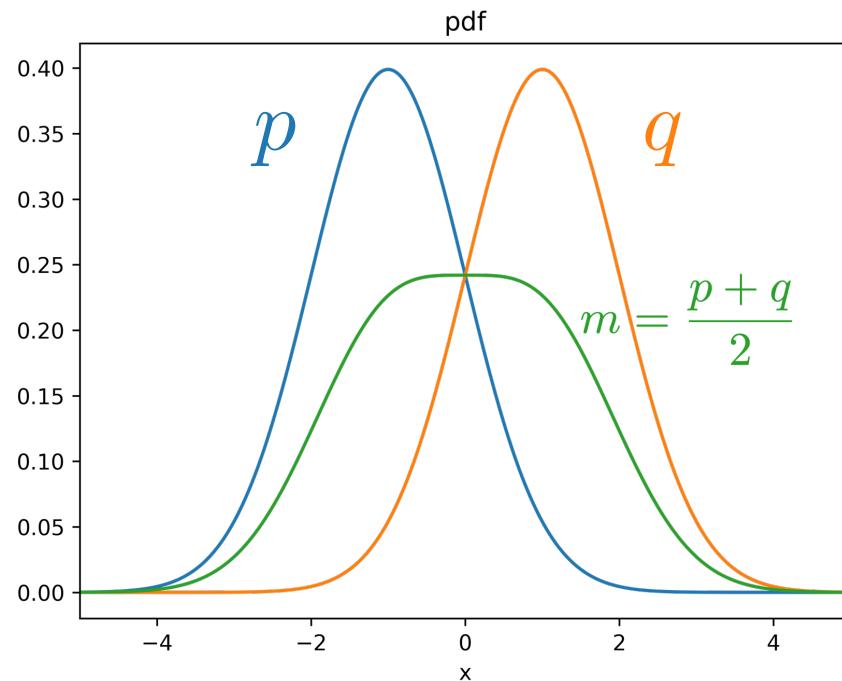
$$\mathcal{L}(D^*, G) = 2D_{JS}(p_{\text{data}} \| p_g) - 2 \log 2$$

where D_{JS} is Jensen–Shannon divergence

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”

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

Properties:

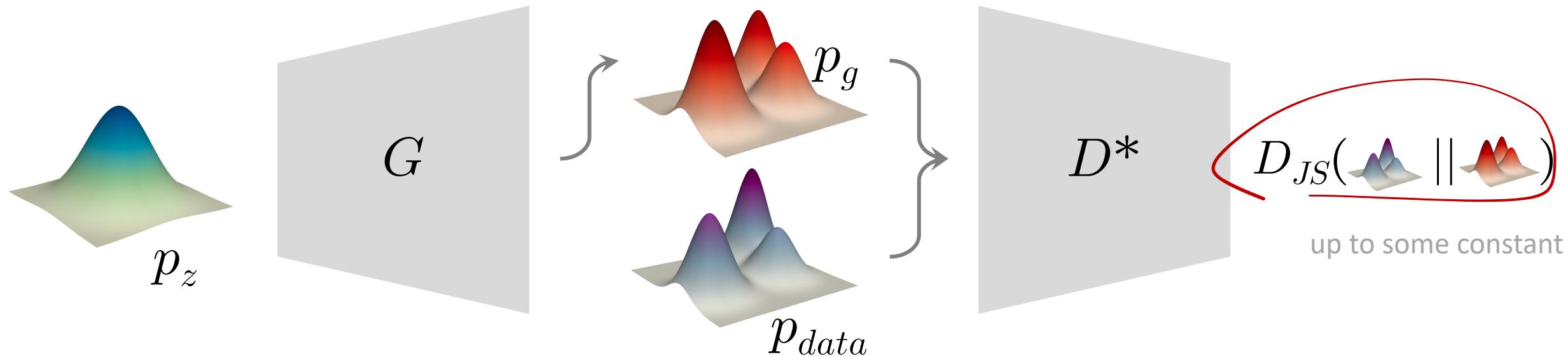
- D_{JS} is symmetric; D_{KL} is not
- D_{JS} is bounded: [0, 1]; D_{KL} is unbounded: [0, inf)
- D_{JS} is more stable

Theoretical Results

- With the optimal D_G , the objective function is:

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

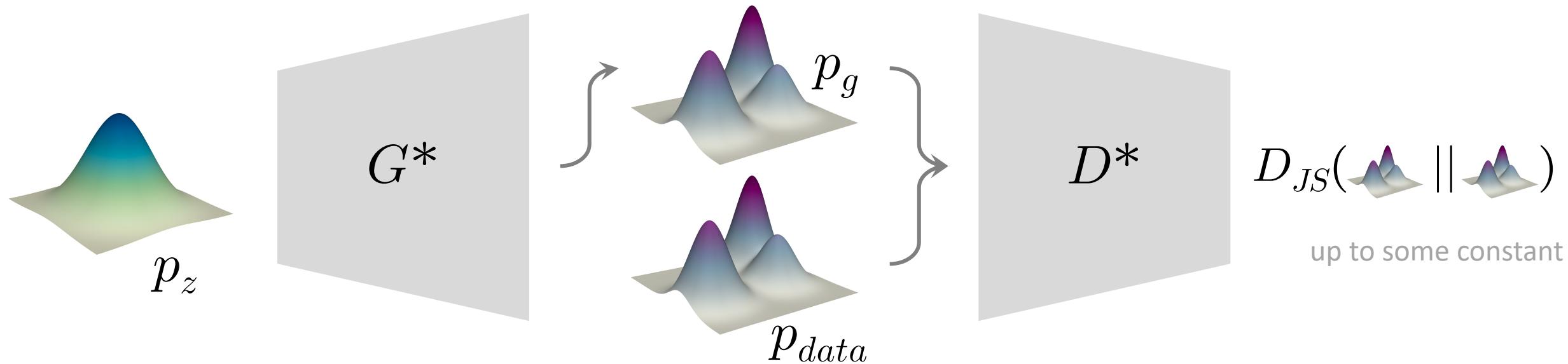
GAN optimizes for Jensen–Shannon divergence.



Theoretical Results

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

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



Theoretical Results: Summary

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

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

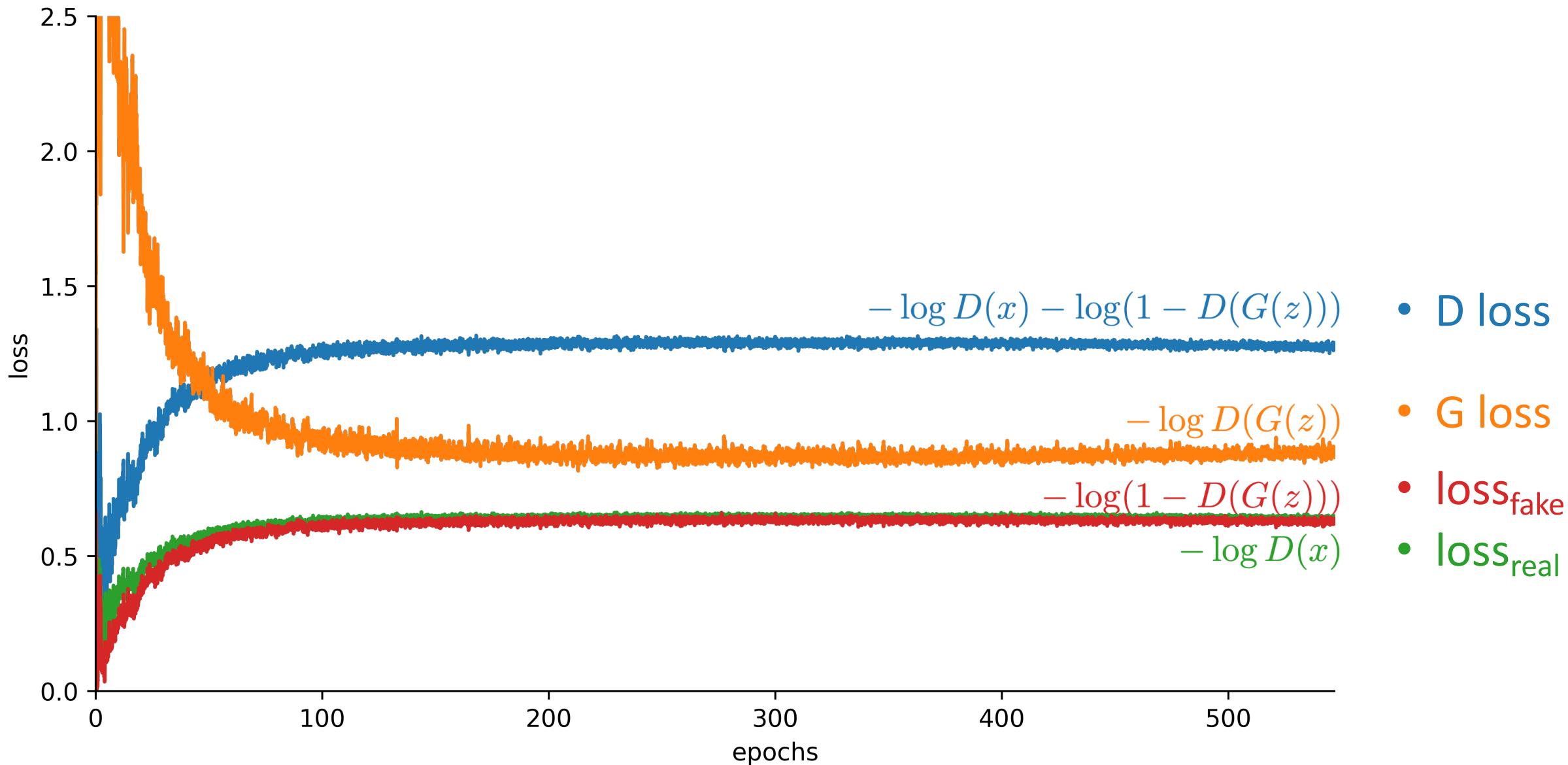
2. With optimal D_G , GAN optimizes for Jensen–Shannon divergence:

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

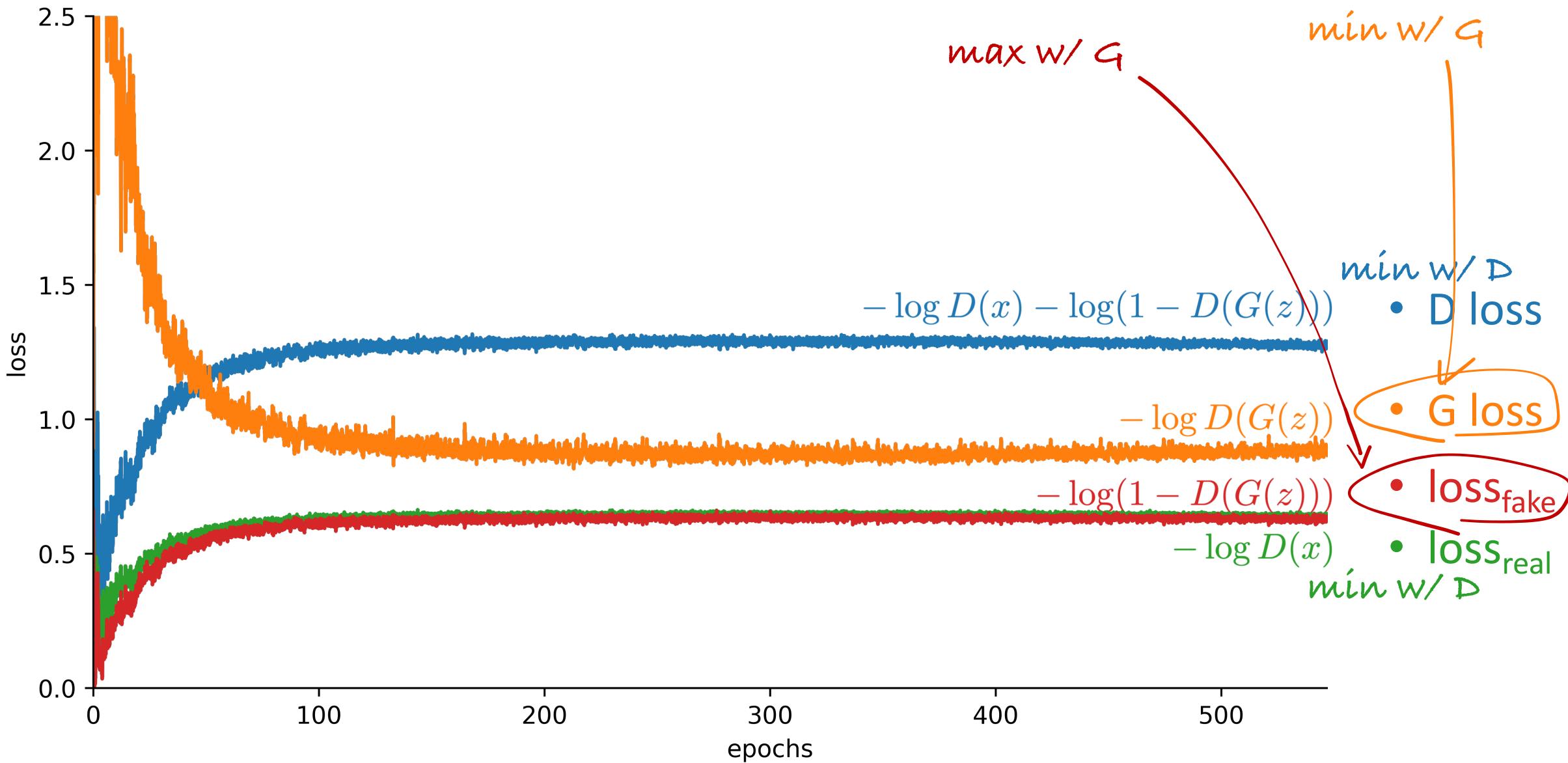
3. Global optimality is achieved at $p_g = p_{\text{data}}$

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

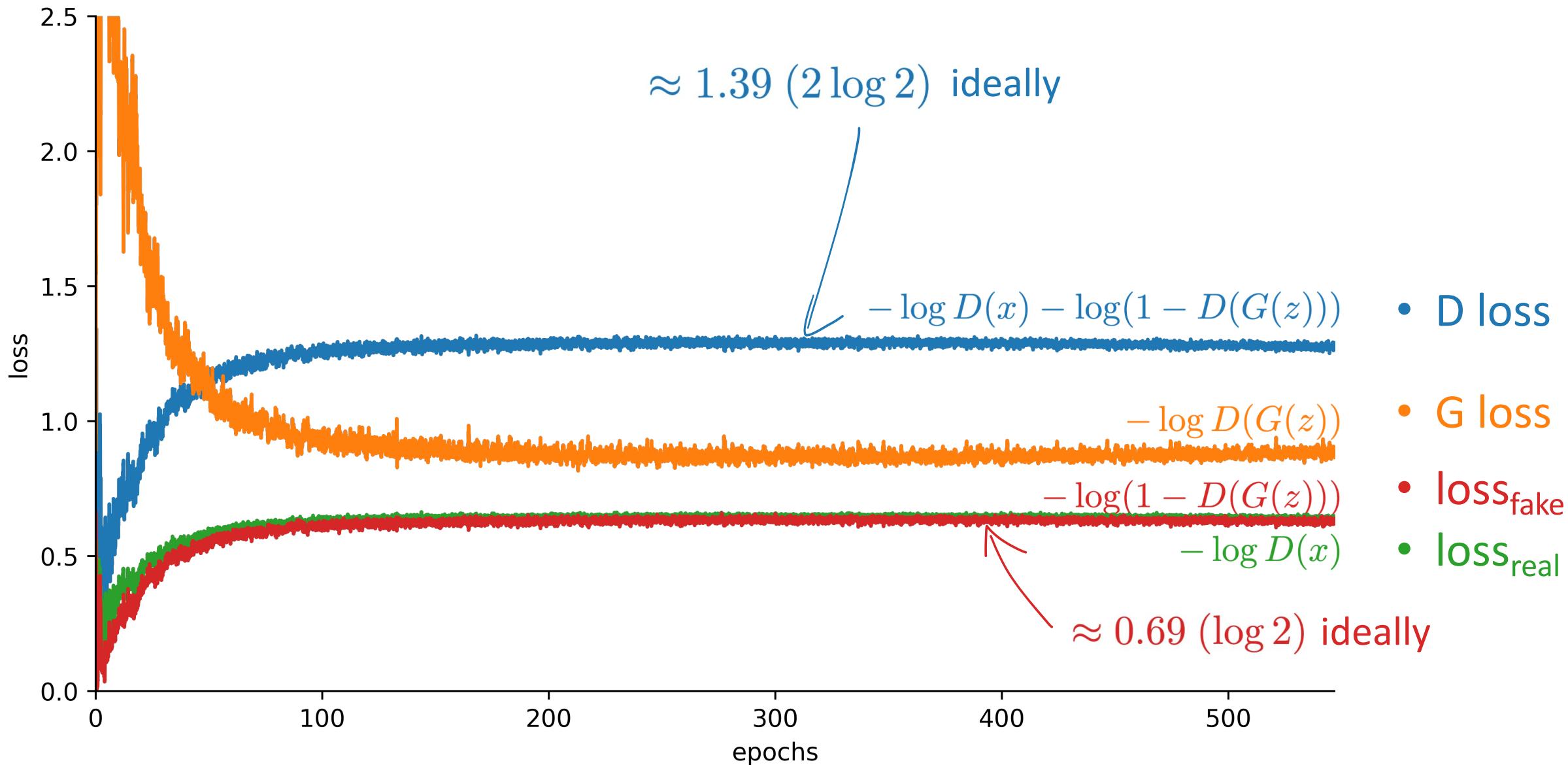
Running example: MNIST



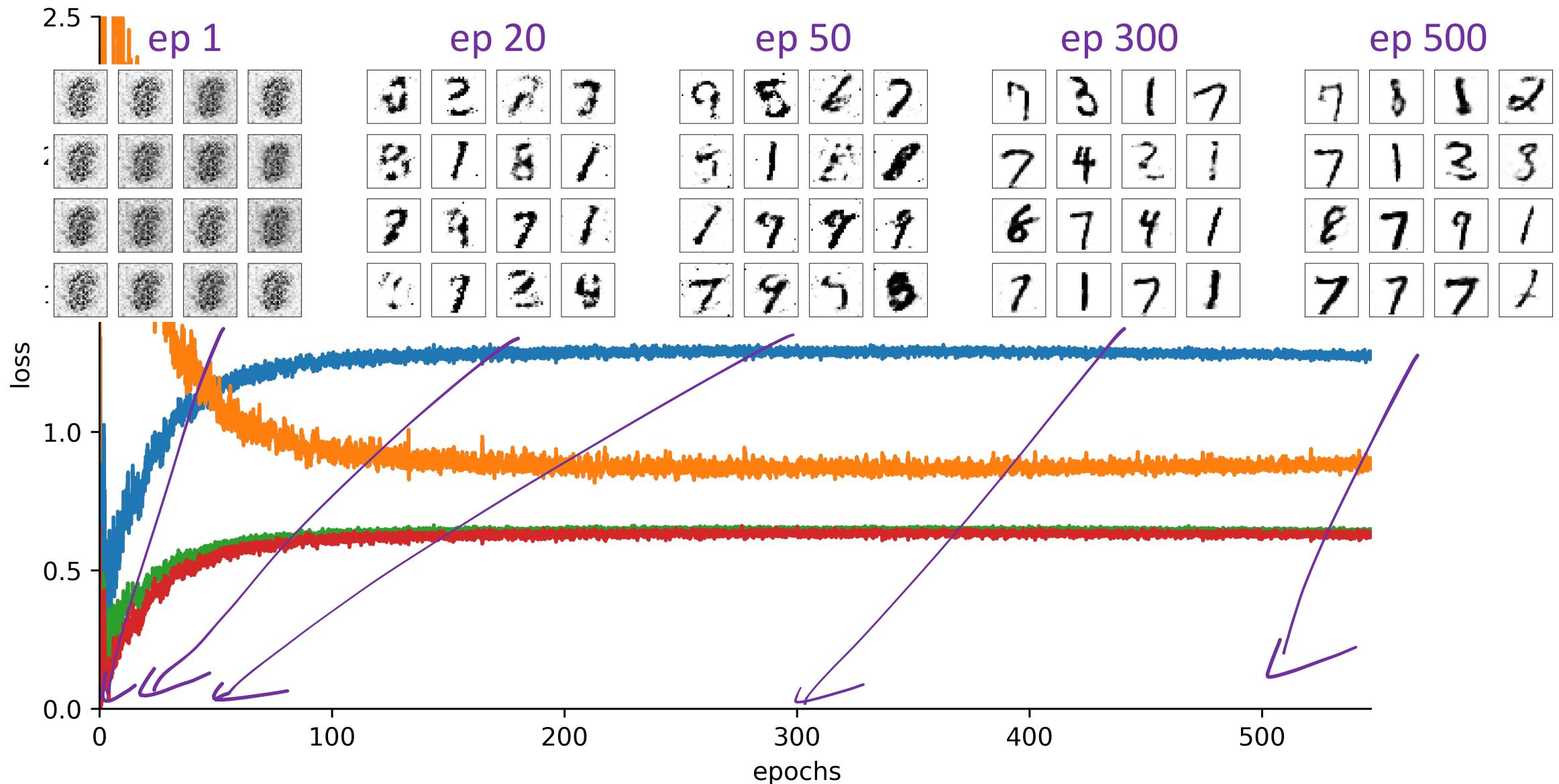
Running example: MNIST



Running example: MNIST



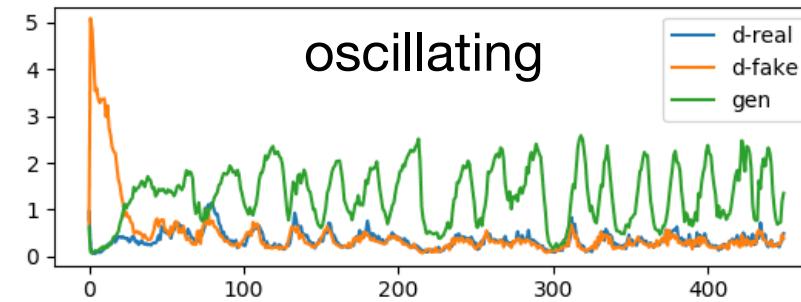
Running example: MNIST



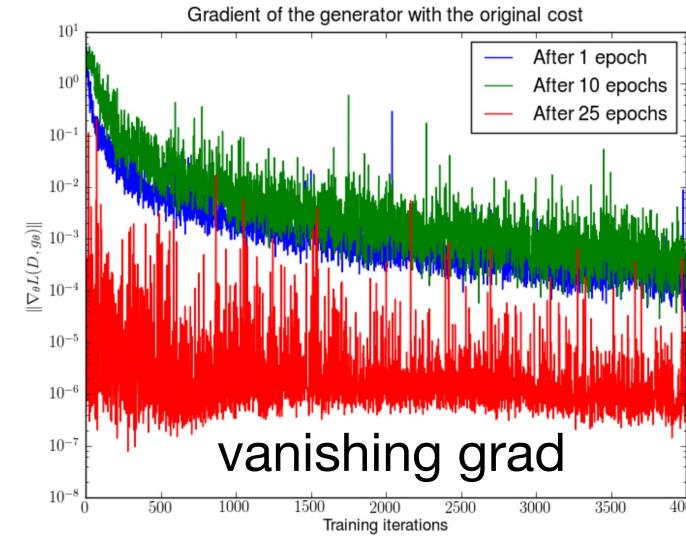
Problems of GAN

Difficult to train/converge

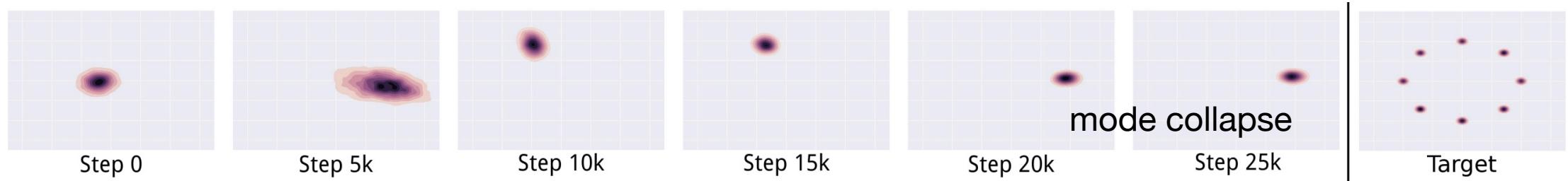
- Hard to achieve equilibrium
- Vanishing gradients
- Mode collapse



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



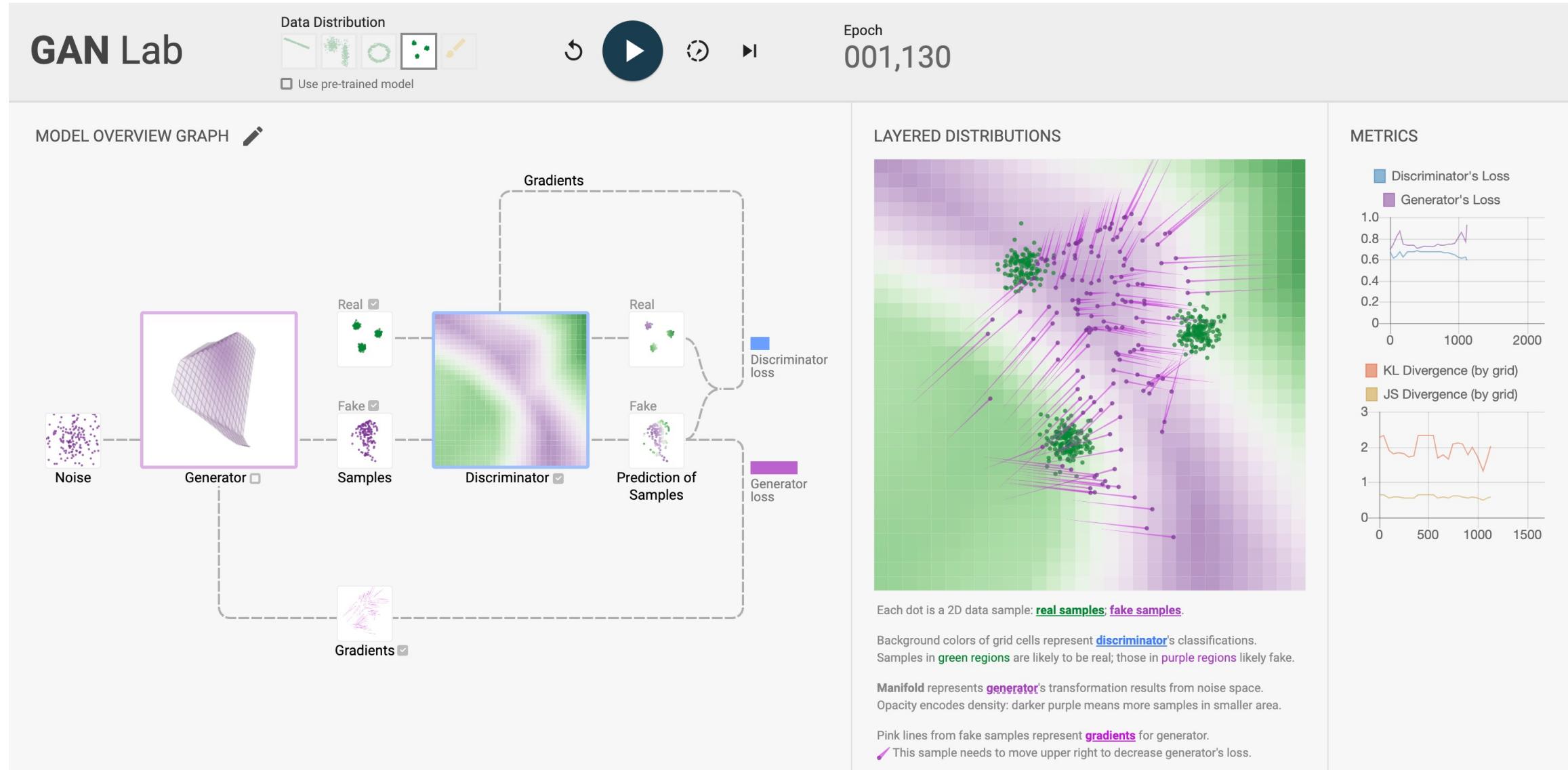
Arjovsky & Bottou, "Towards Principled Methods for Training GANs"



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:

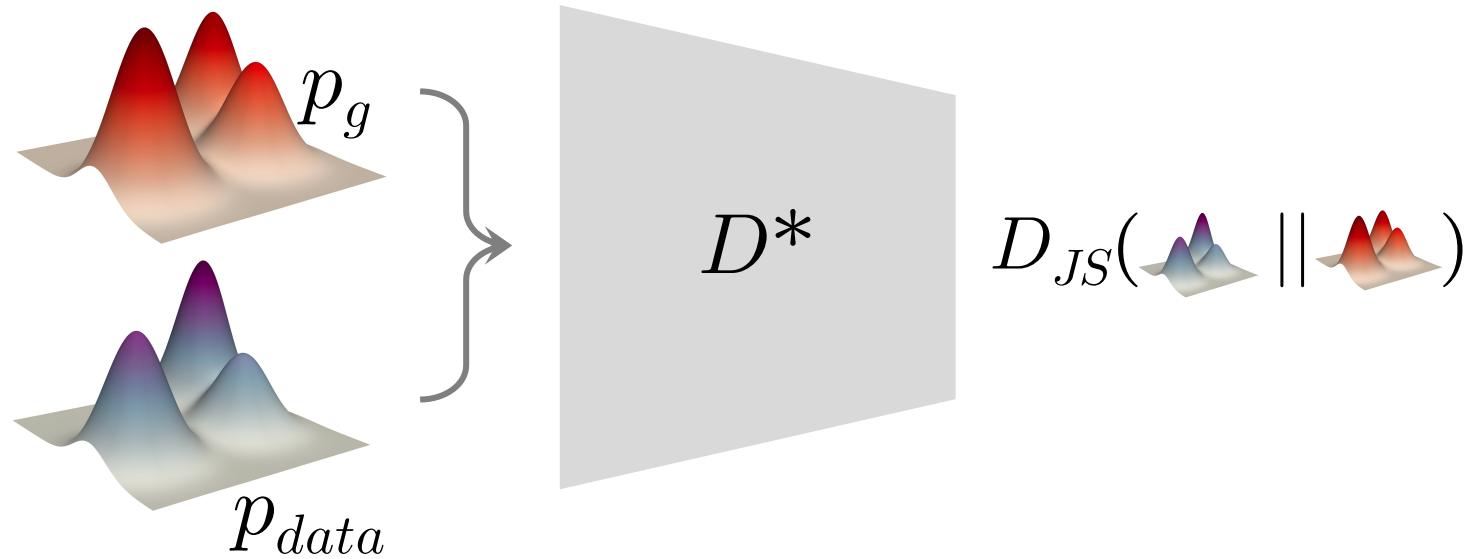
- remove logarithms
- clip weights

For laymen:

- art critic, instead of forgery expert

Recap: GAN optimizes for D_{JS}

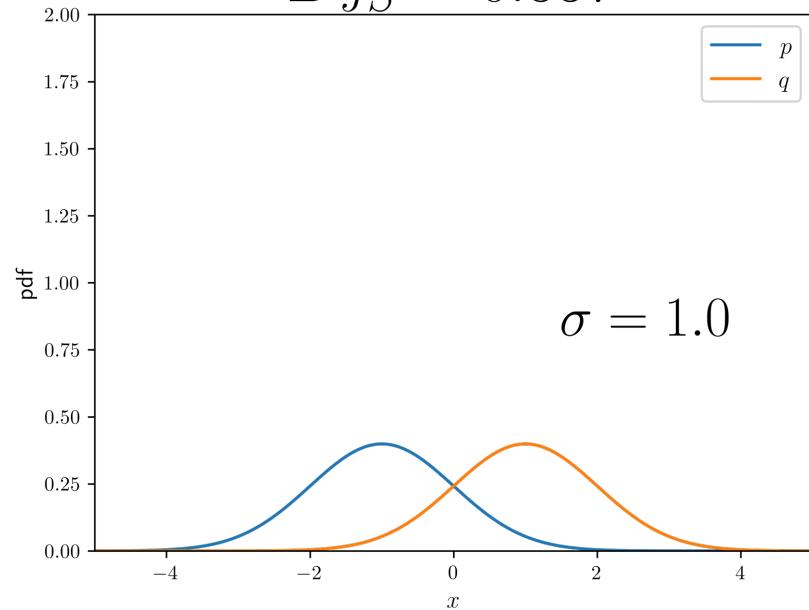
$$\mathcal{L}(D^*, G) = 2D_{JS}(p_{\text{data}} \| p_g) - 2 \log 2$$



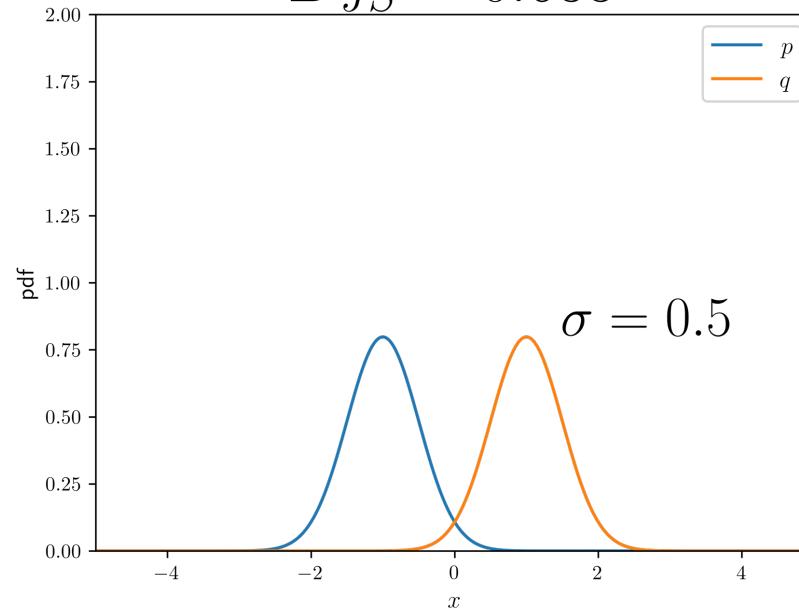
Problems of D_{JS}

If p and q don't overlap, D_{JS} is a constant ($\log 2$), i.e., no gradient

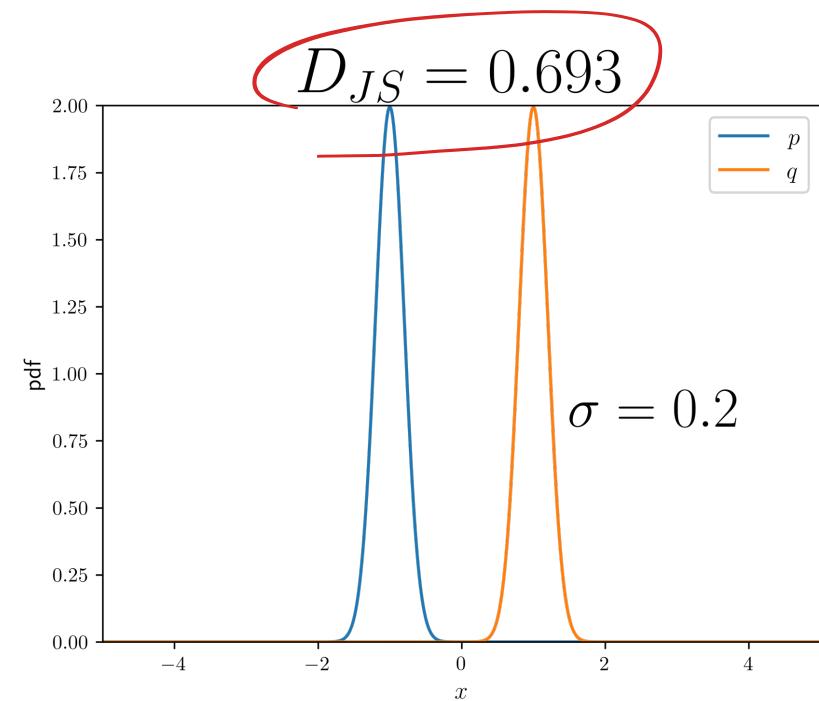
$$D_{JS} = 0.337$$



$$D_{JS} = 0.633$$

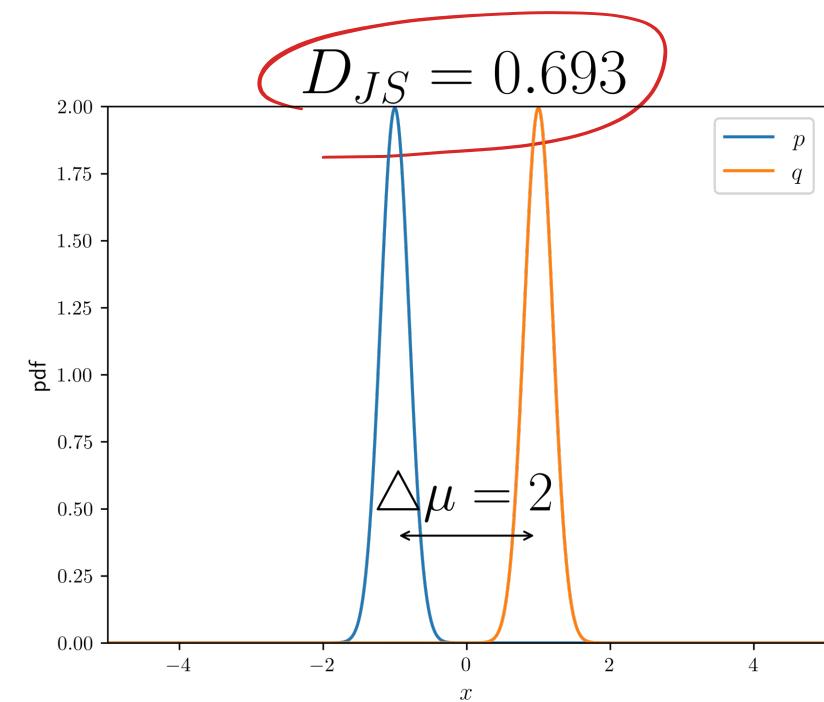
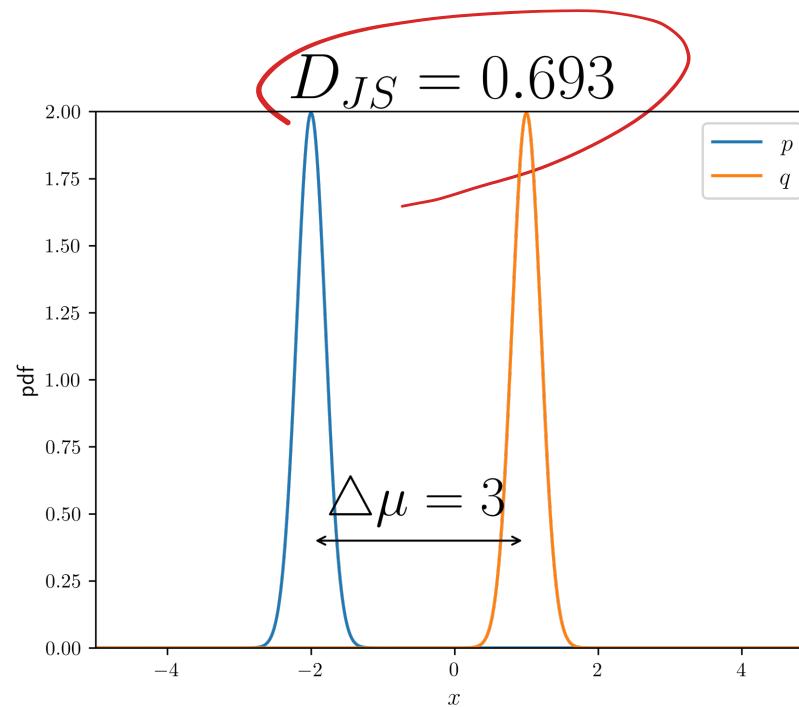
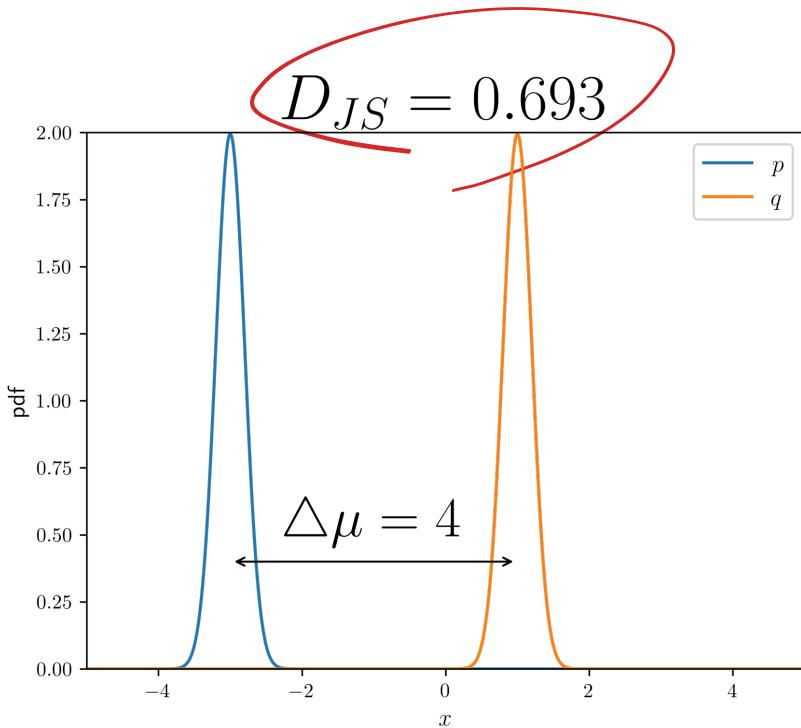


$$D_{JS} = 0.693$$



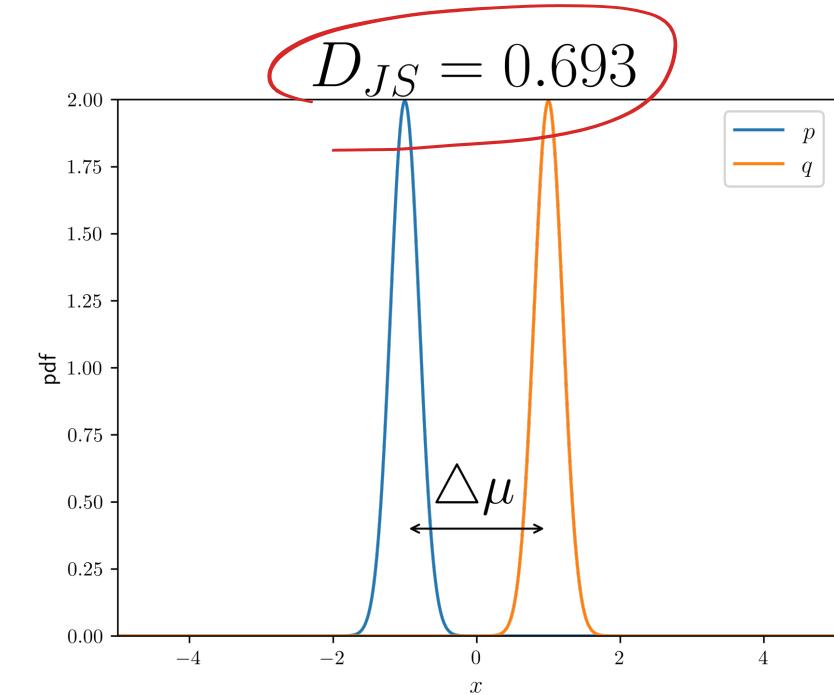
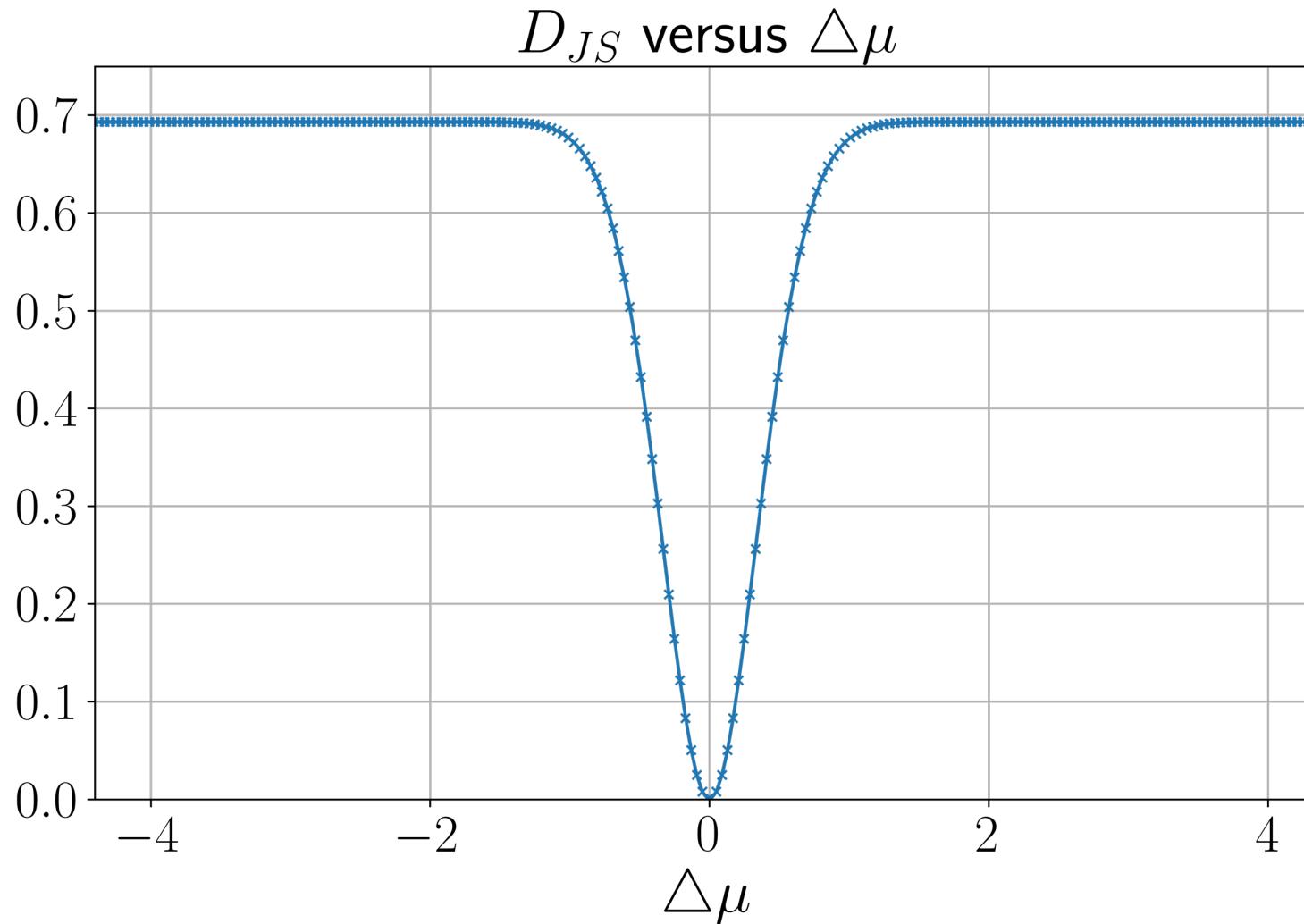
Problems of D_{JS}

If p and q don't overlap, D_{JS} is a constant ($\log 2$), i.e., no gradient



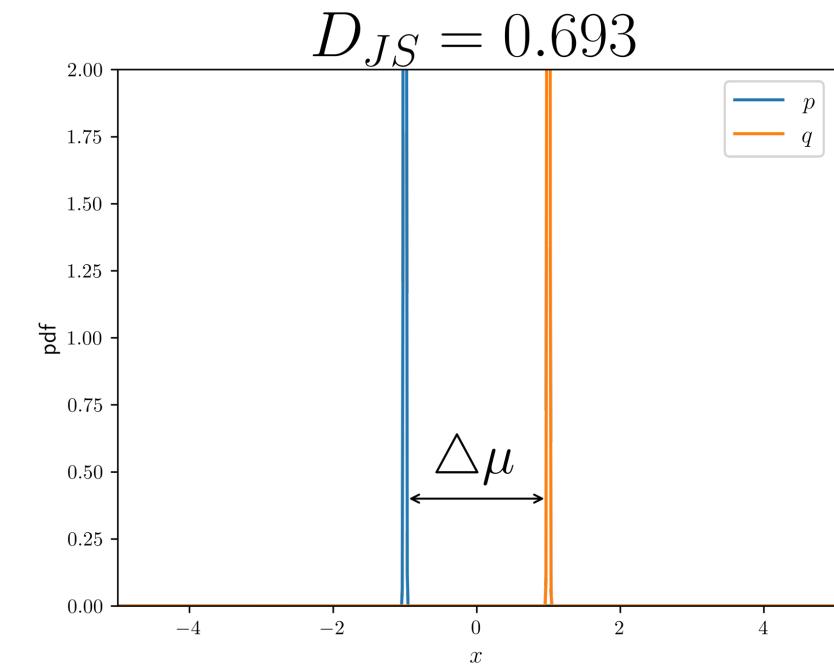
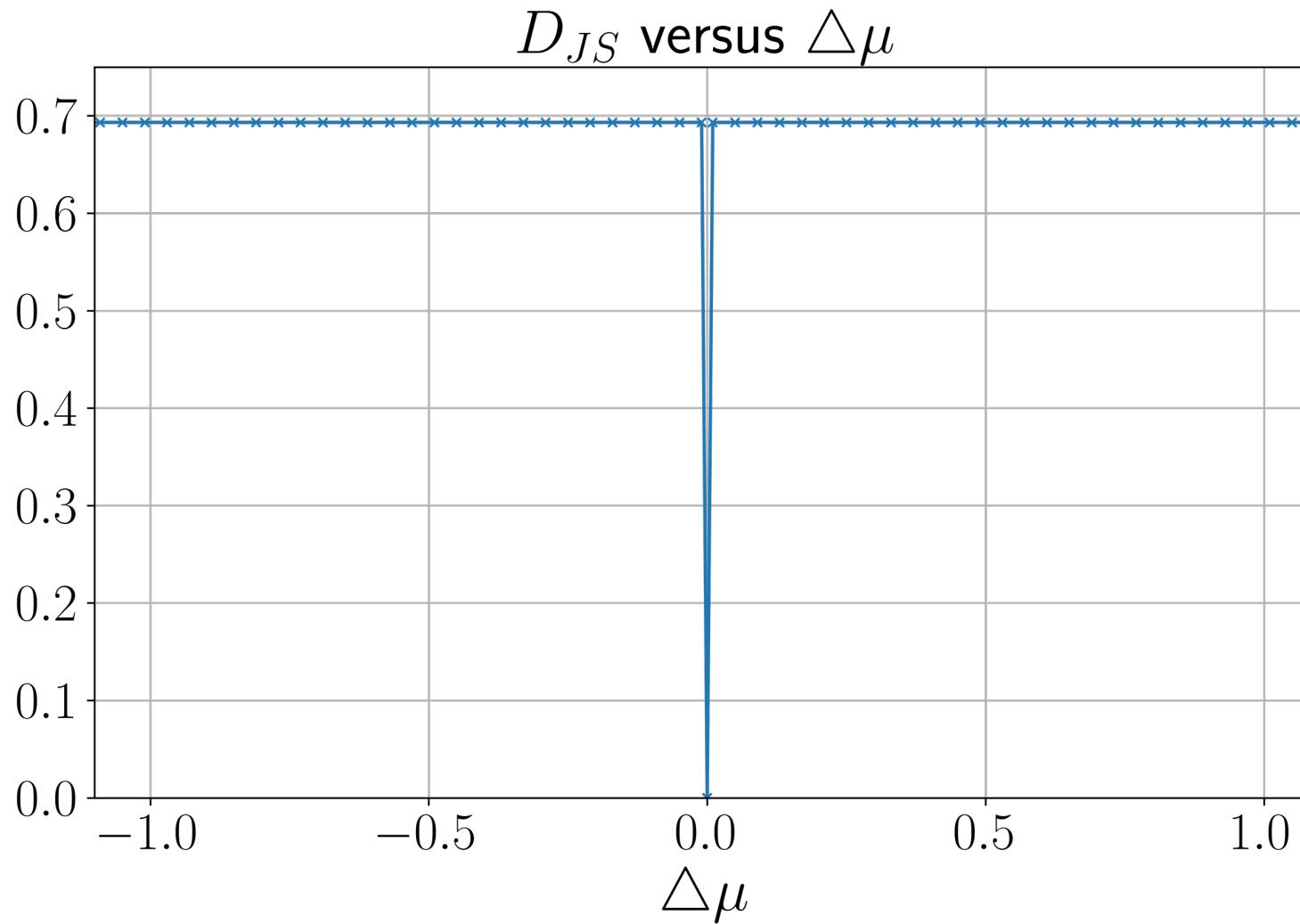
Problems of D_{JS}

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



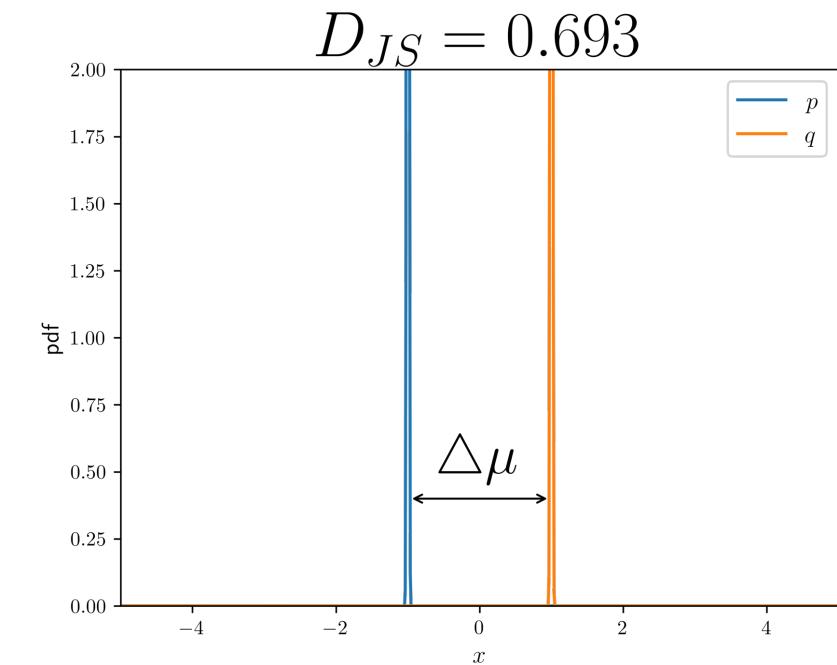
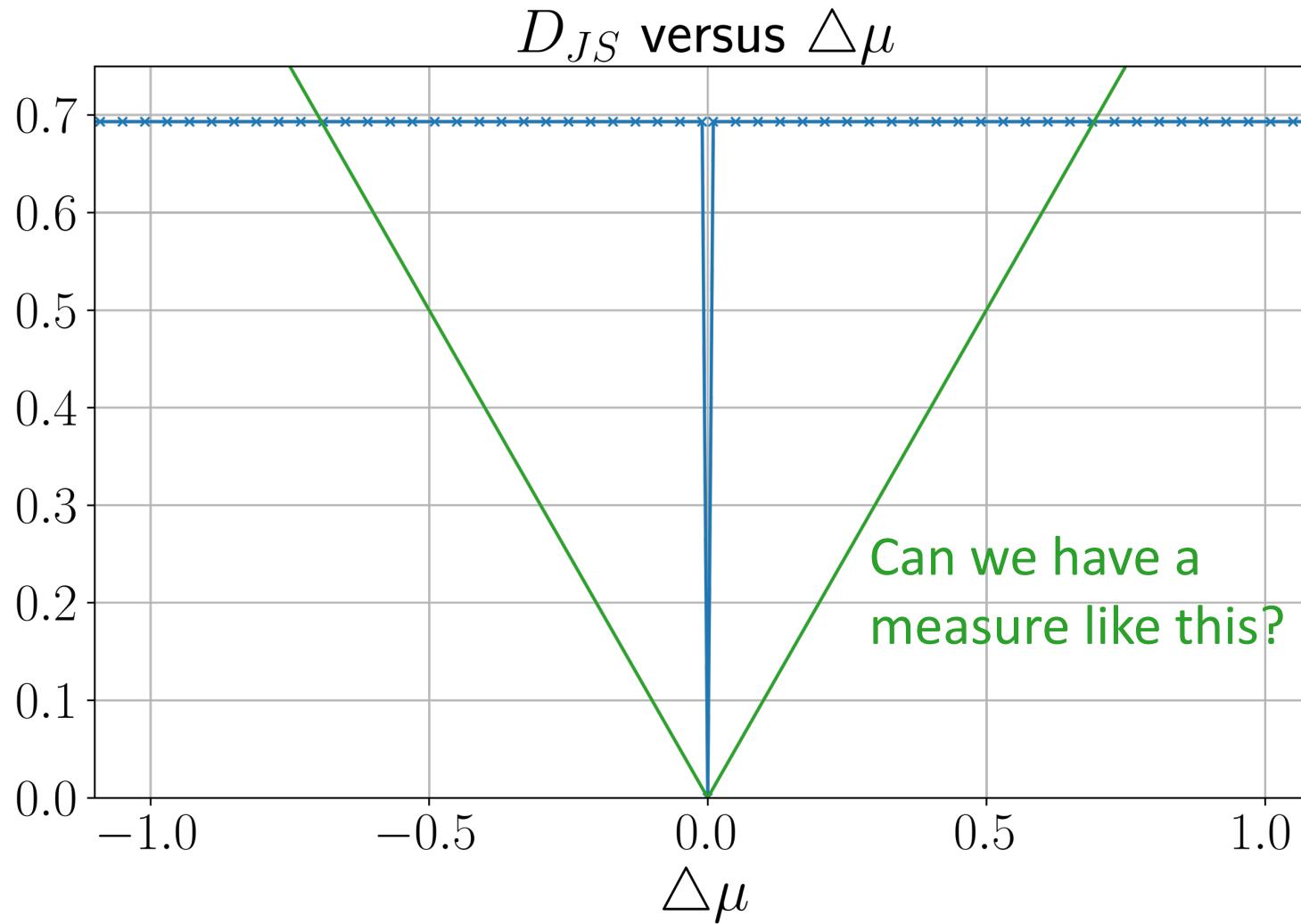
Problems of D_{JS}

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



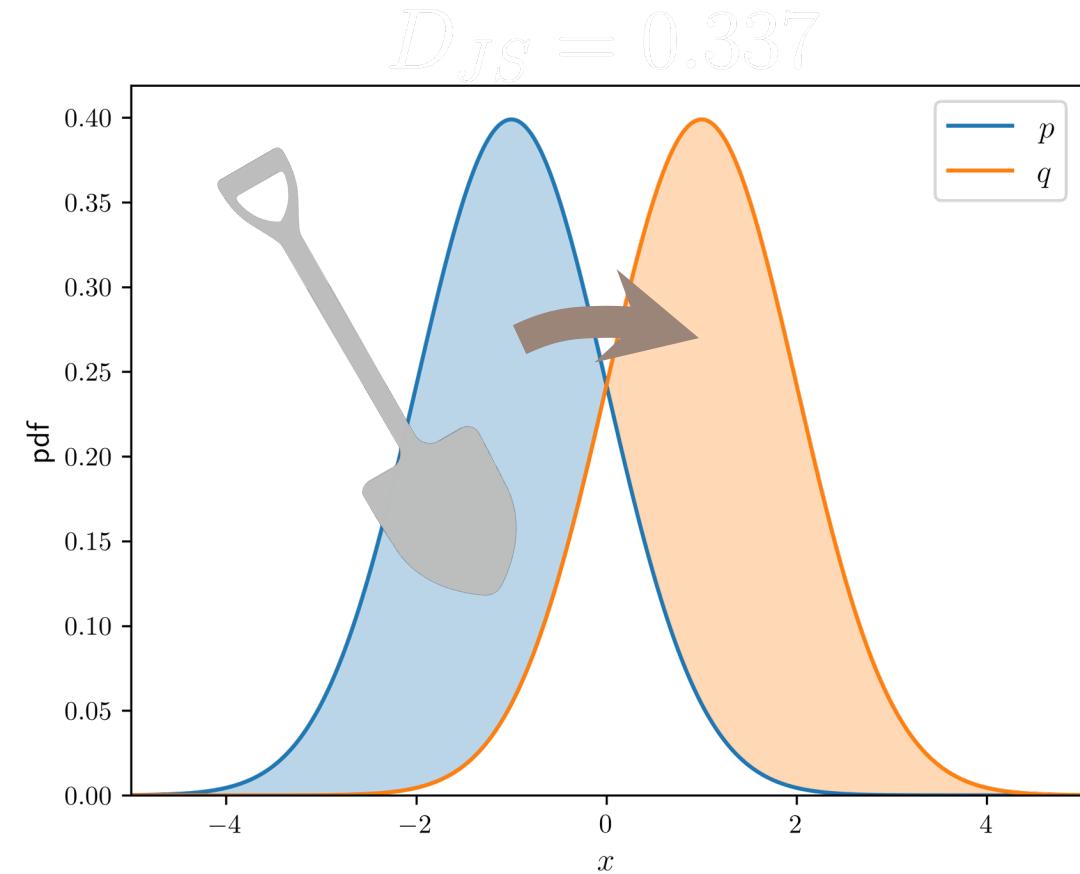
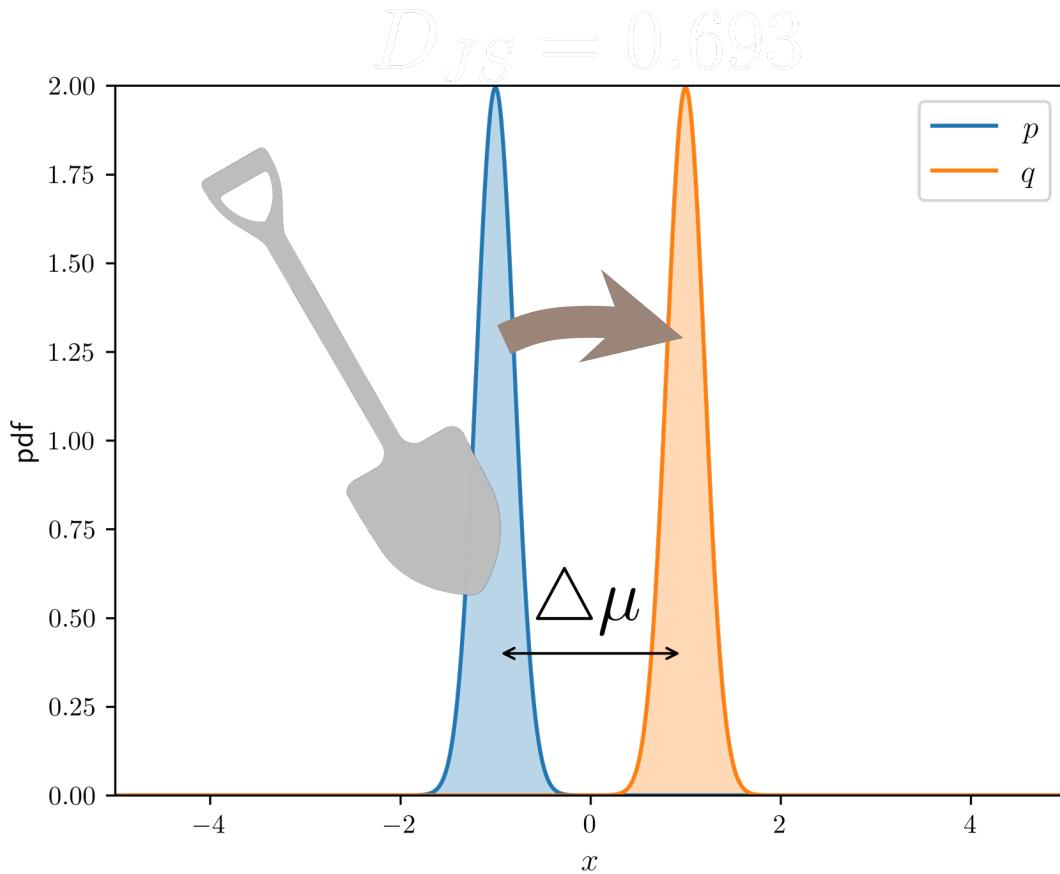
Problems of D_{JS}

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



Wasserstein Distance

“Earth Mover’s Distance”



Running example: Wasserstein Distance

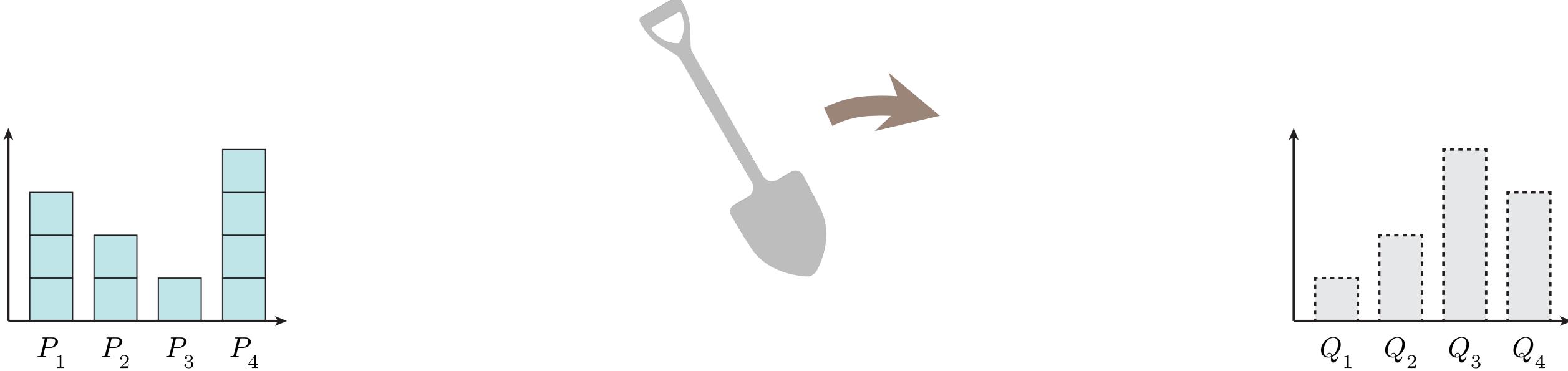
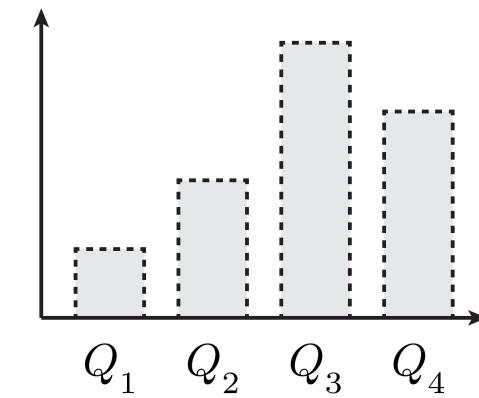
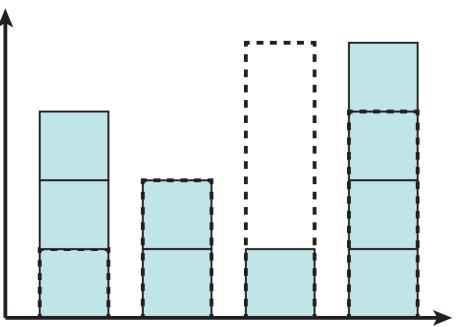
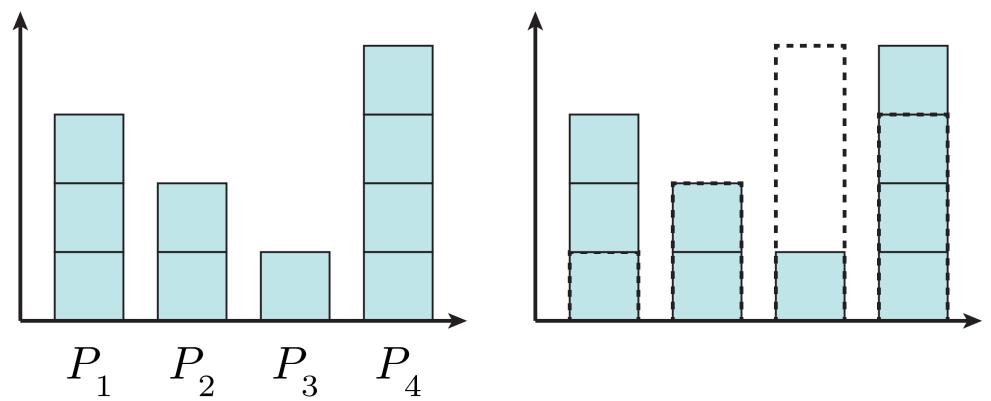


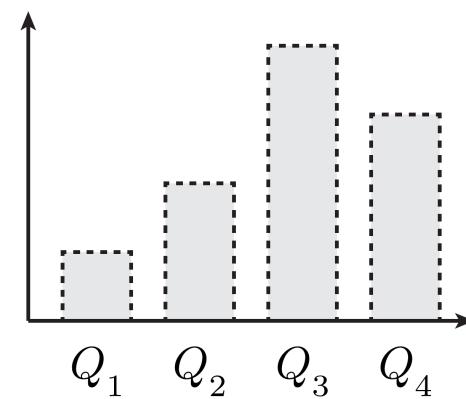
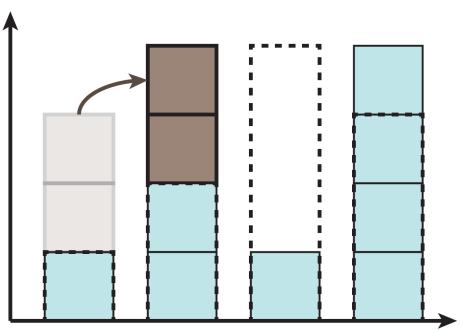
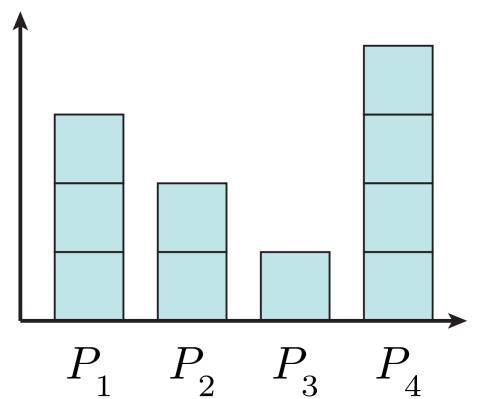
Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

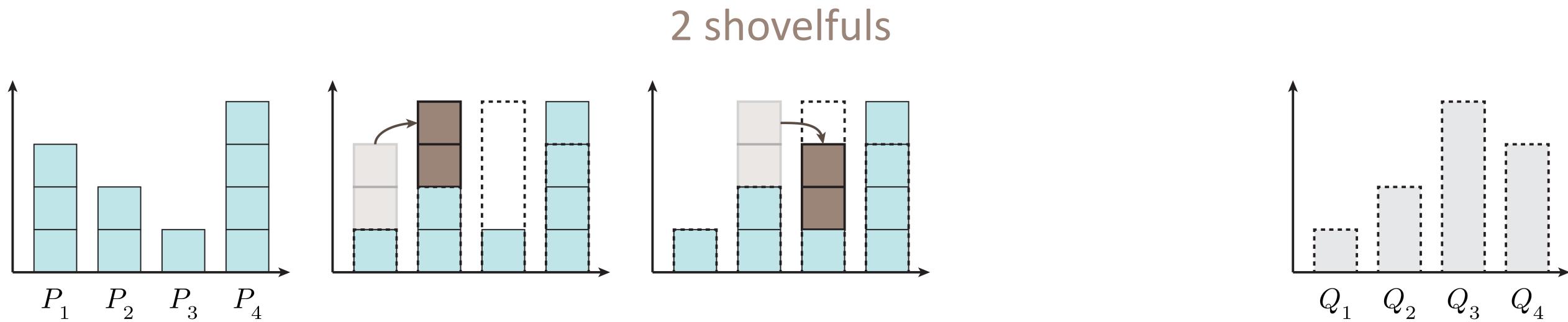


Running example: Wasserstein Distance

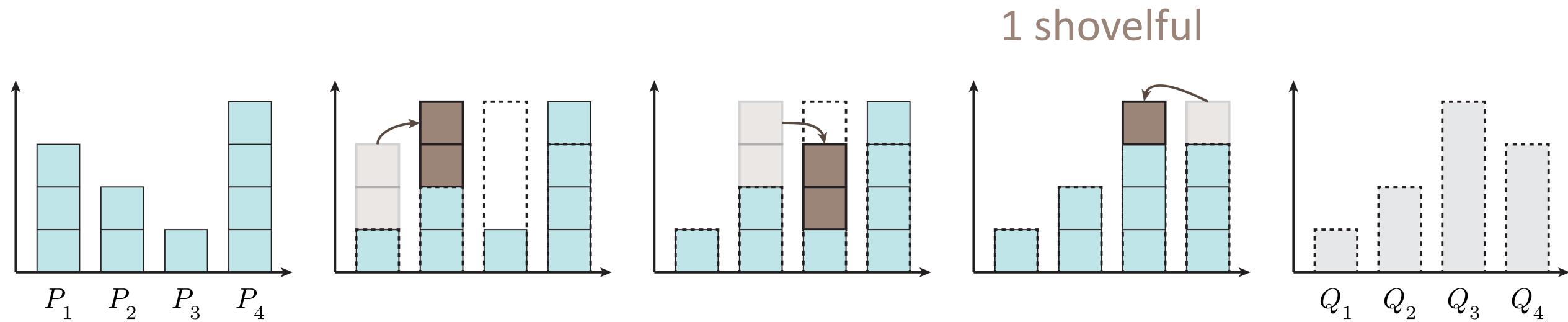
2 shovelfuls



Running example: Wasserstein Distance

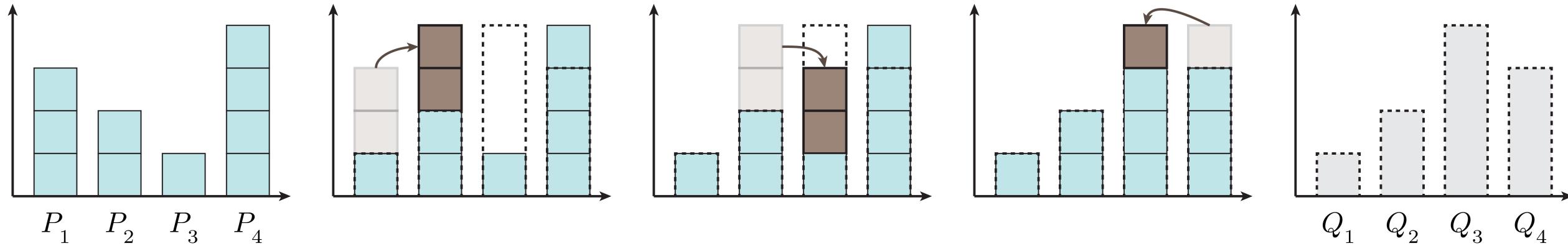


Running example: Wasserstein Distance

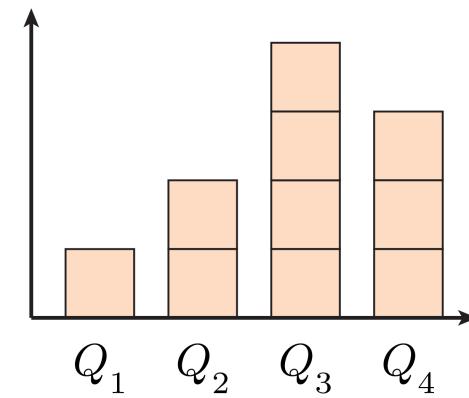
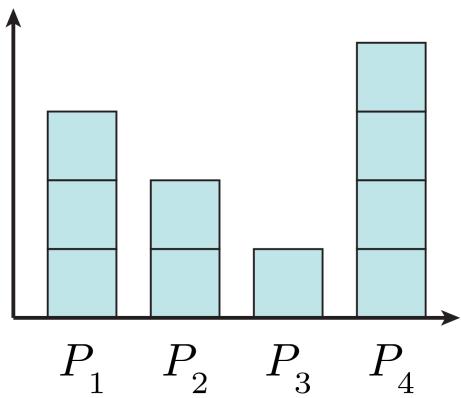


Running example: Wasserstein Distance

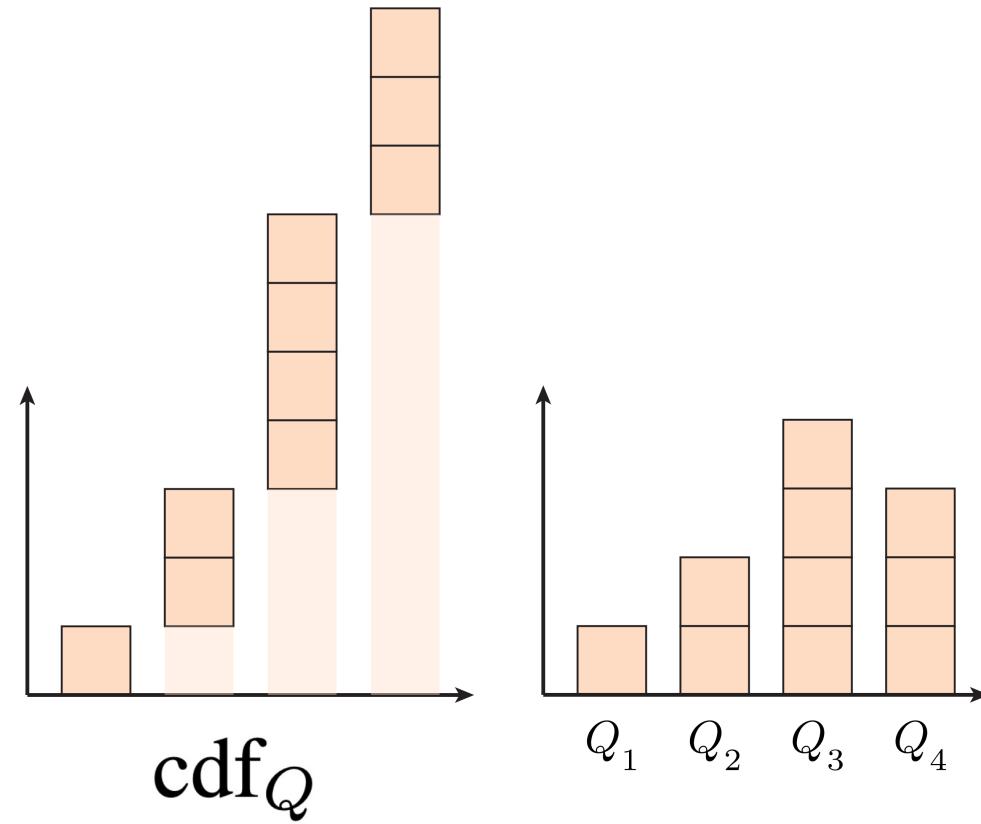
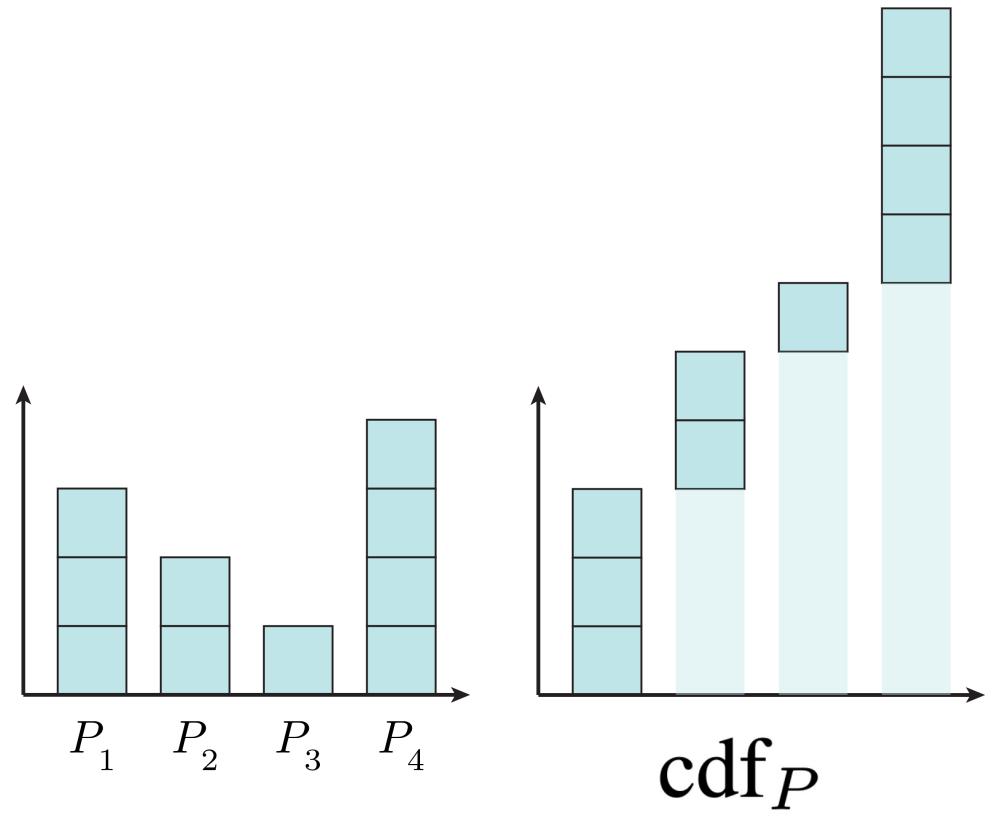
$$W(P, Q) = 5 \times \blacksquare$$



Running example: Wasserstein Distance

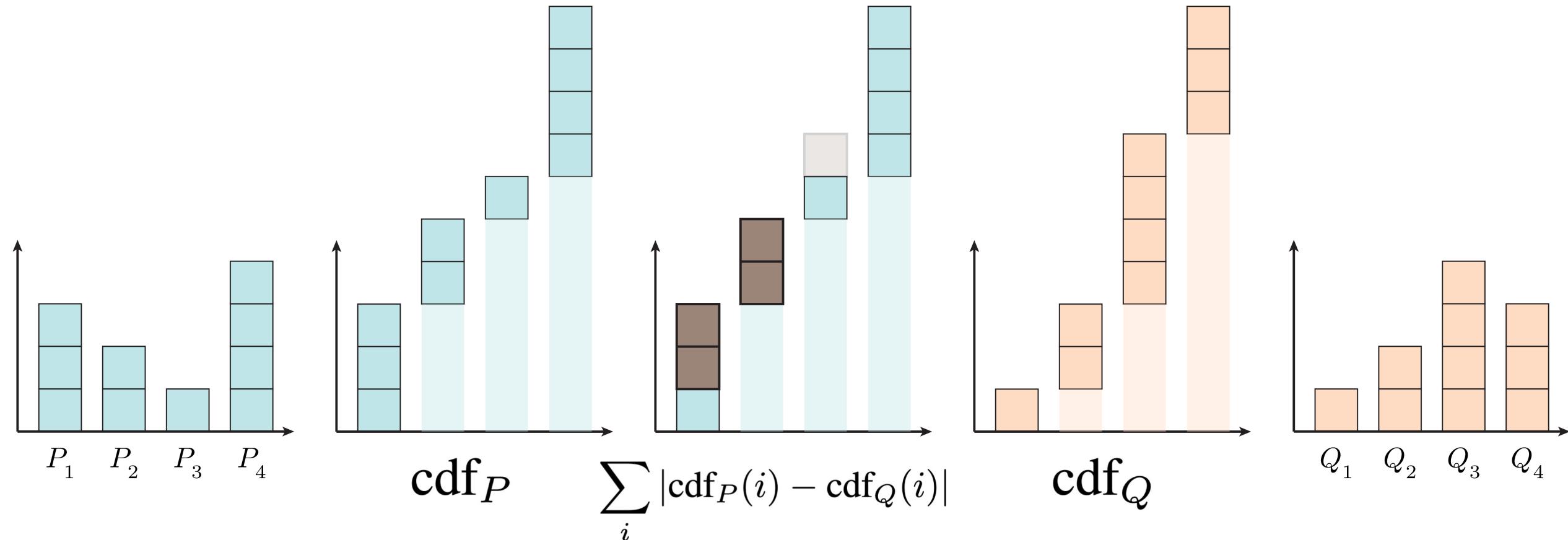


Running example: Wasserstein Distance



- cdf: cumulative distribution function

Running example: Wasserstein Distance



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

Wasserstein Distance

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

l_1 -norm

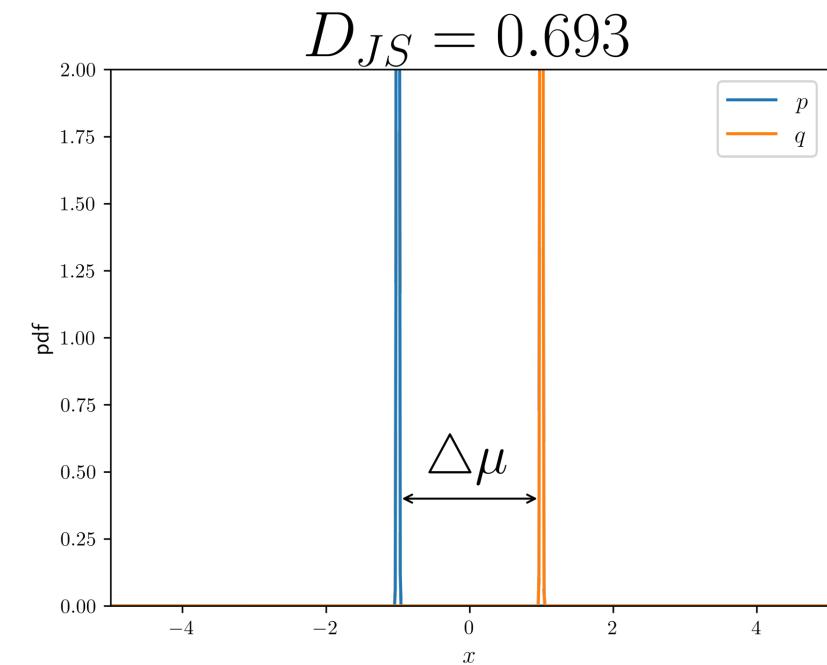
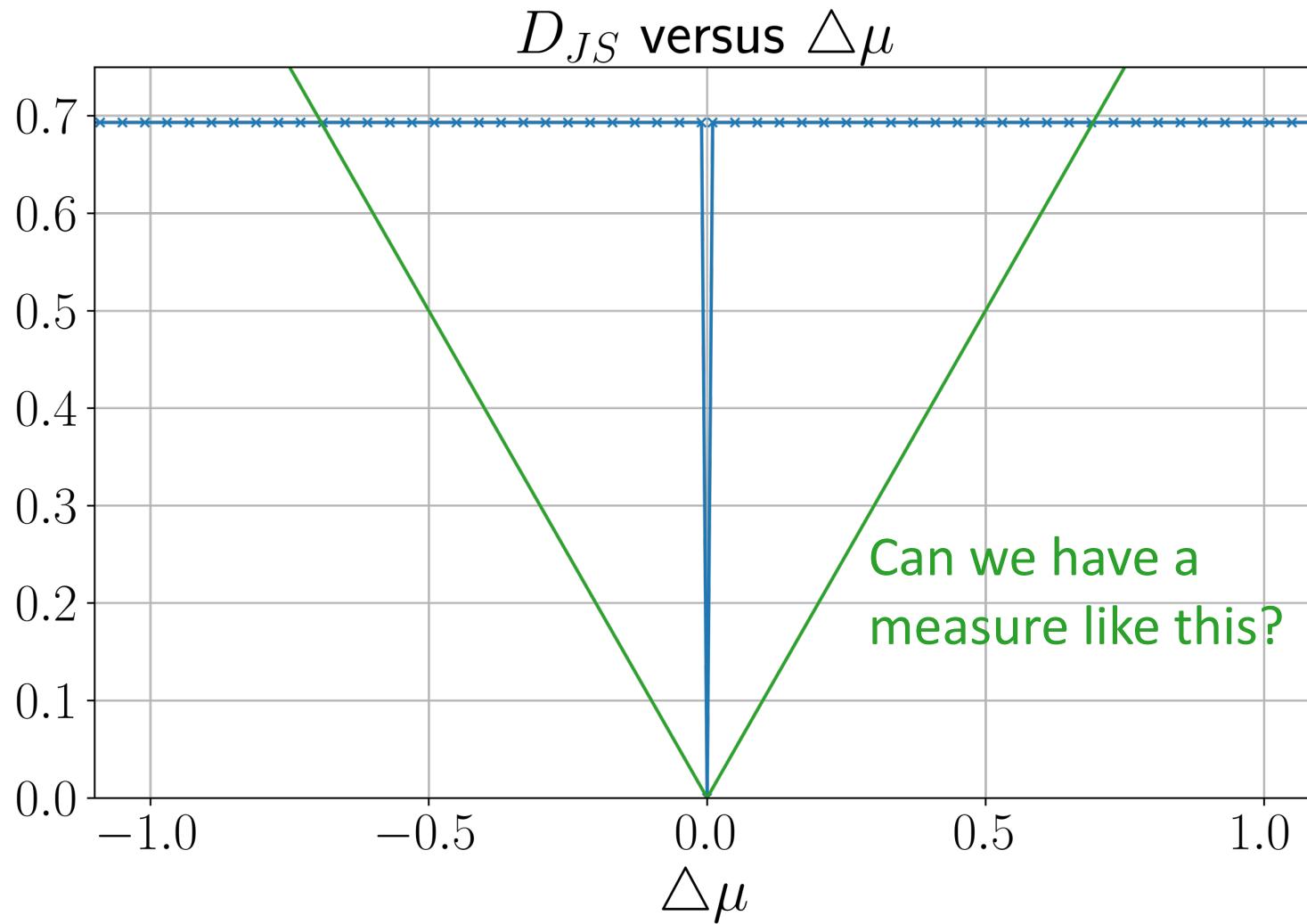
$$W_1(P, Q) = \sum_i |\text{cdf}_P(i) - \text{cdf}_Q(i)|$$

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

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

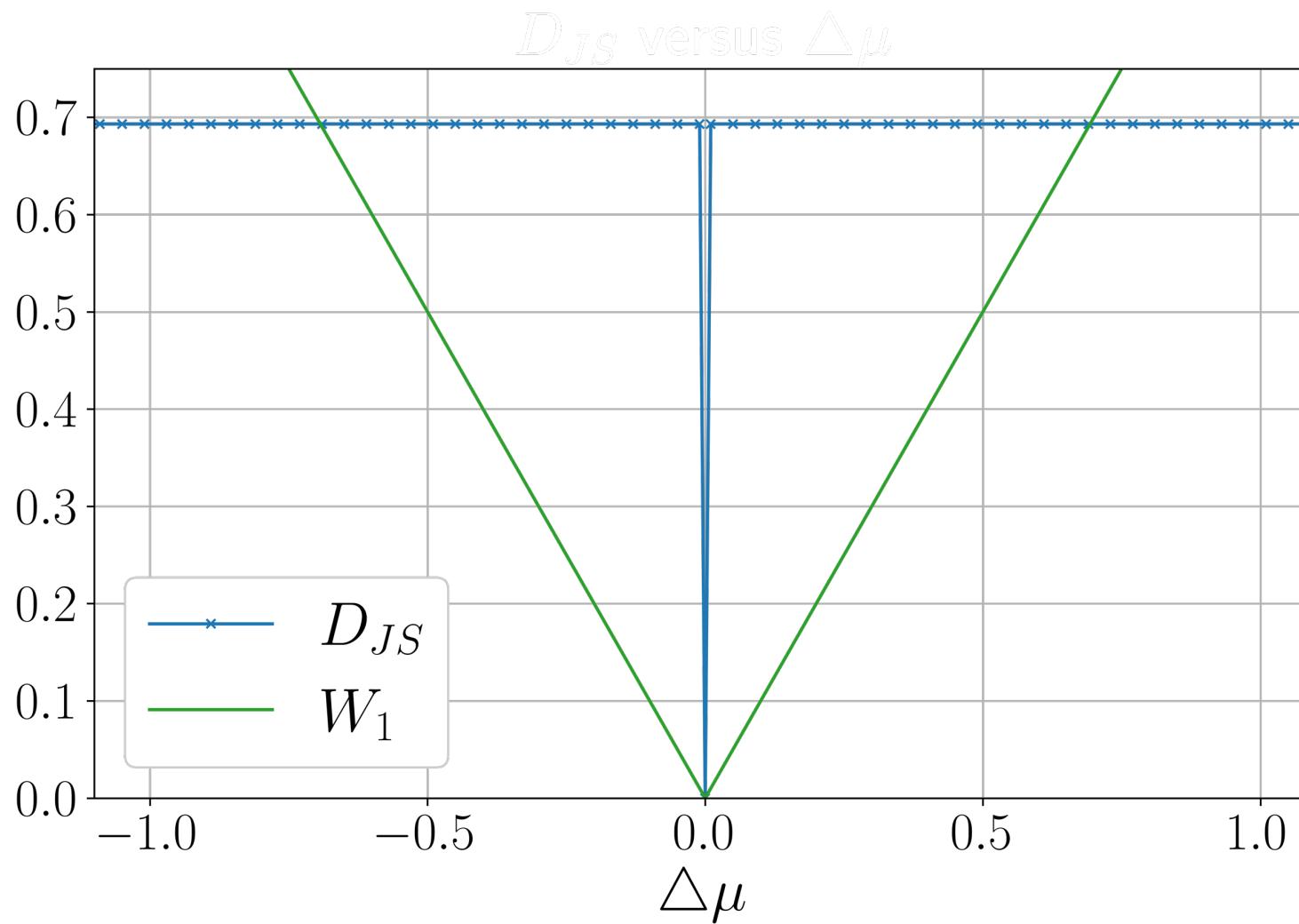
Recap: D_{JS}

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

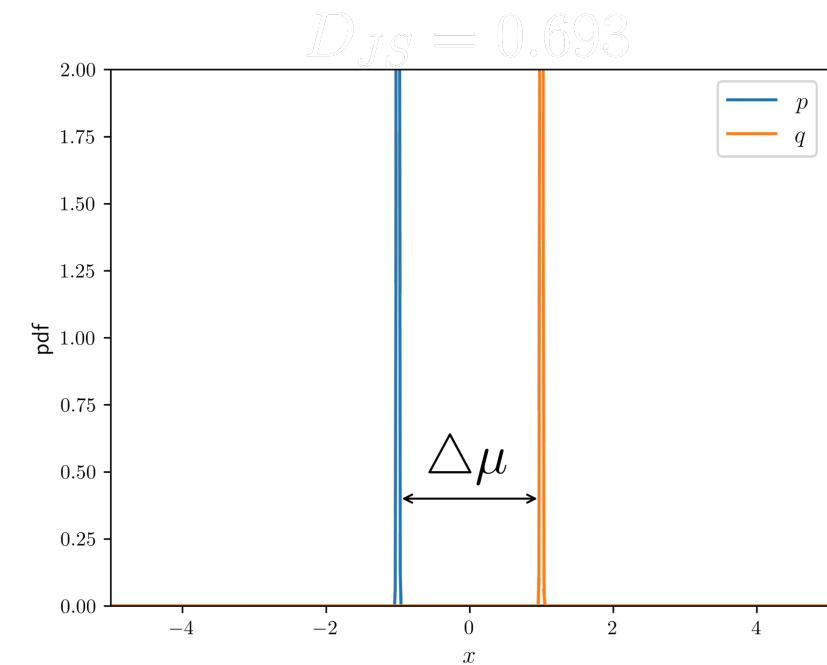


Wasserstein Distance

- when p and q are delta functions:



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

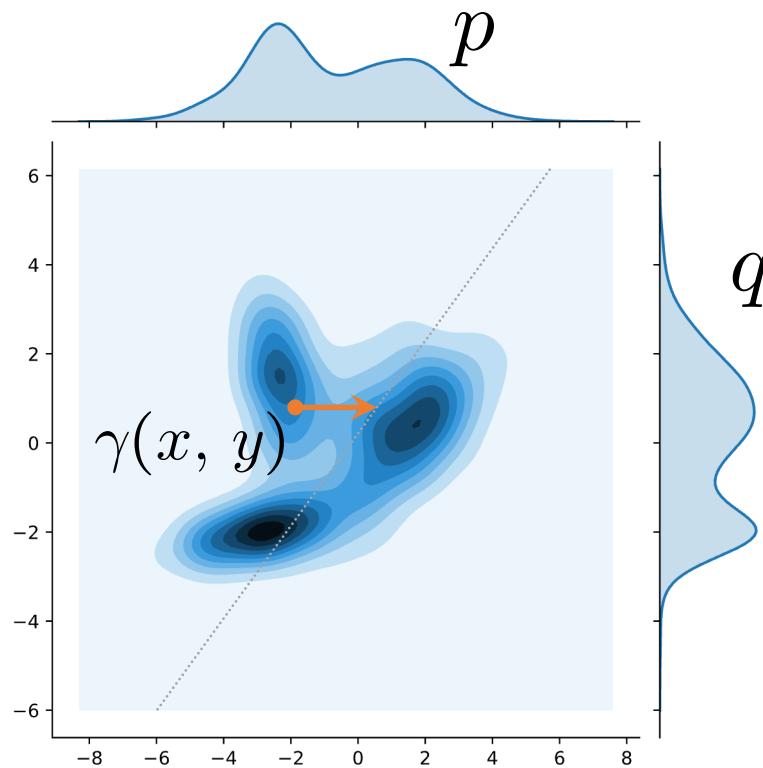


Wasserstein Distance

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

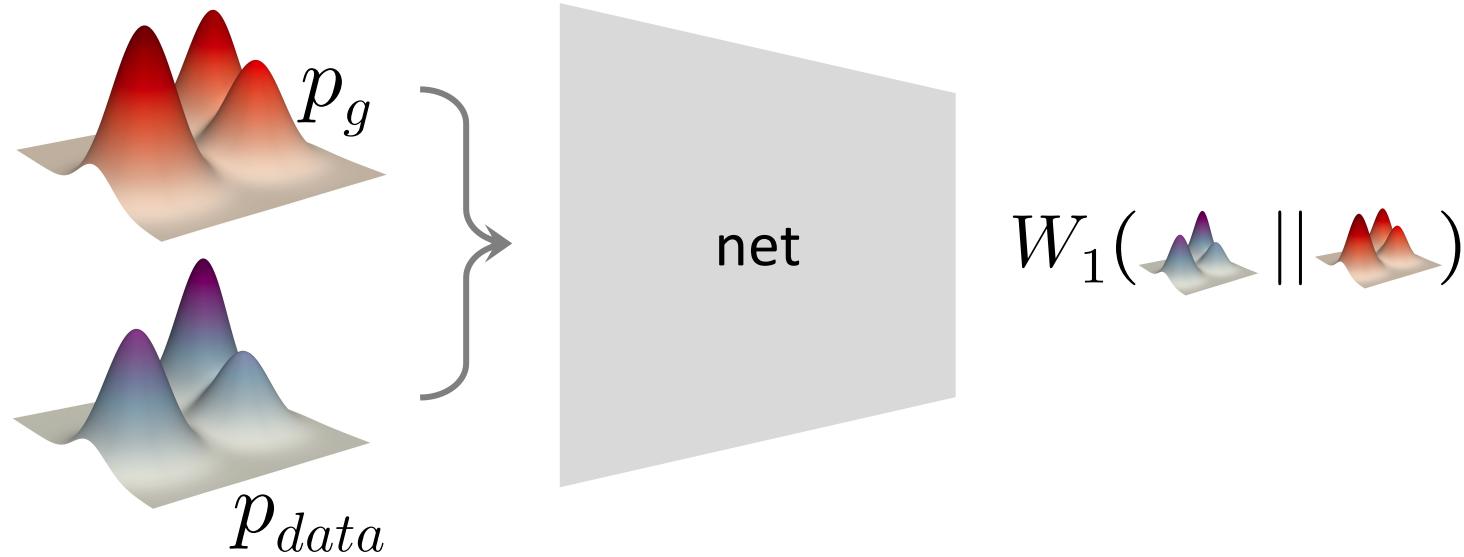
$$W_1(p, q) = \inf_{\gamma \in \Pi(p, q)} \mathbb{E}_{(x, y) \sim \gamma} [|x - y|]$$

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



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:

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

• 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:

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

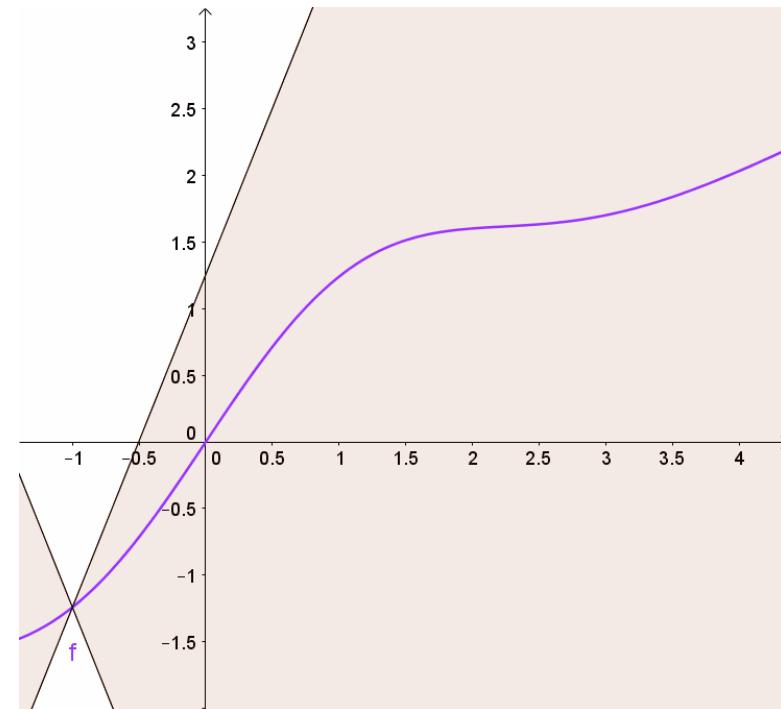


Figure from: https://en.wikipedia.org/wiki/Lipschitz_continuity

W-GAN optimizes for Wasserstein Distance

- Kantorovich-Rubinstein duality:

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

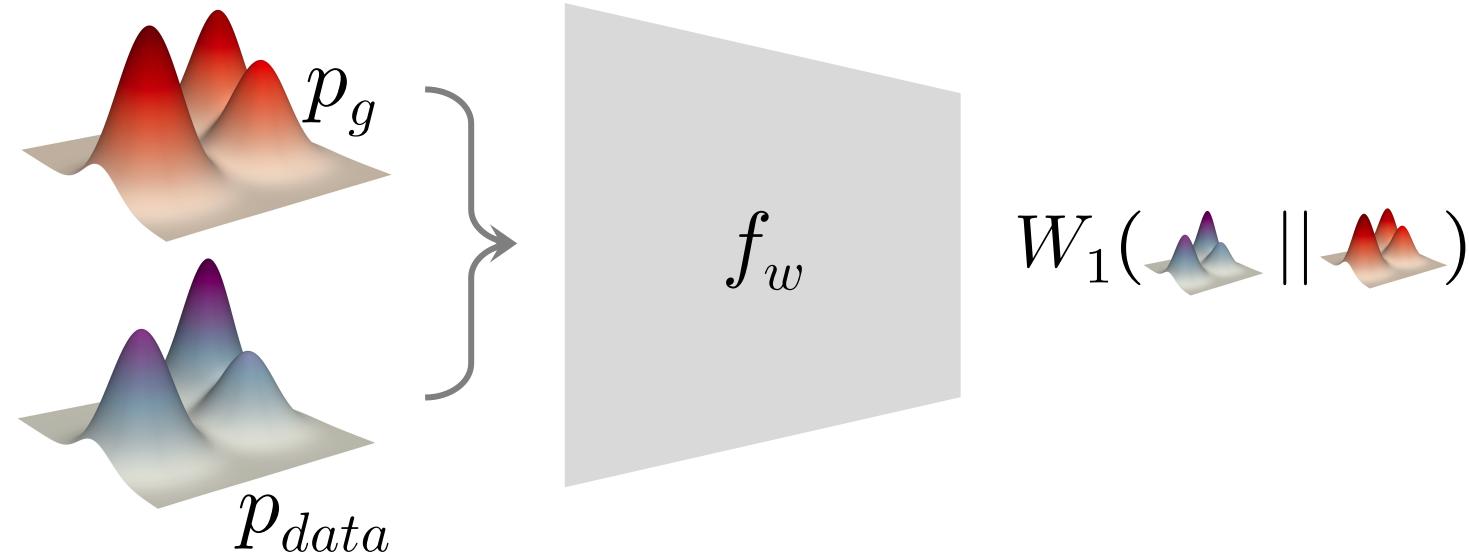
K -Lipschitz continuity:

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

W-GAN optimizes for Wasserstein Distance

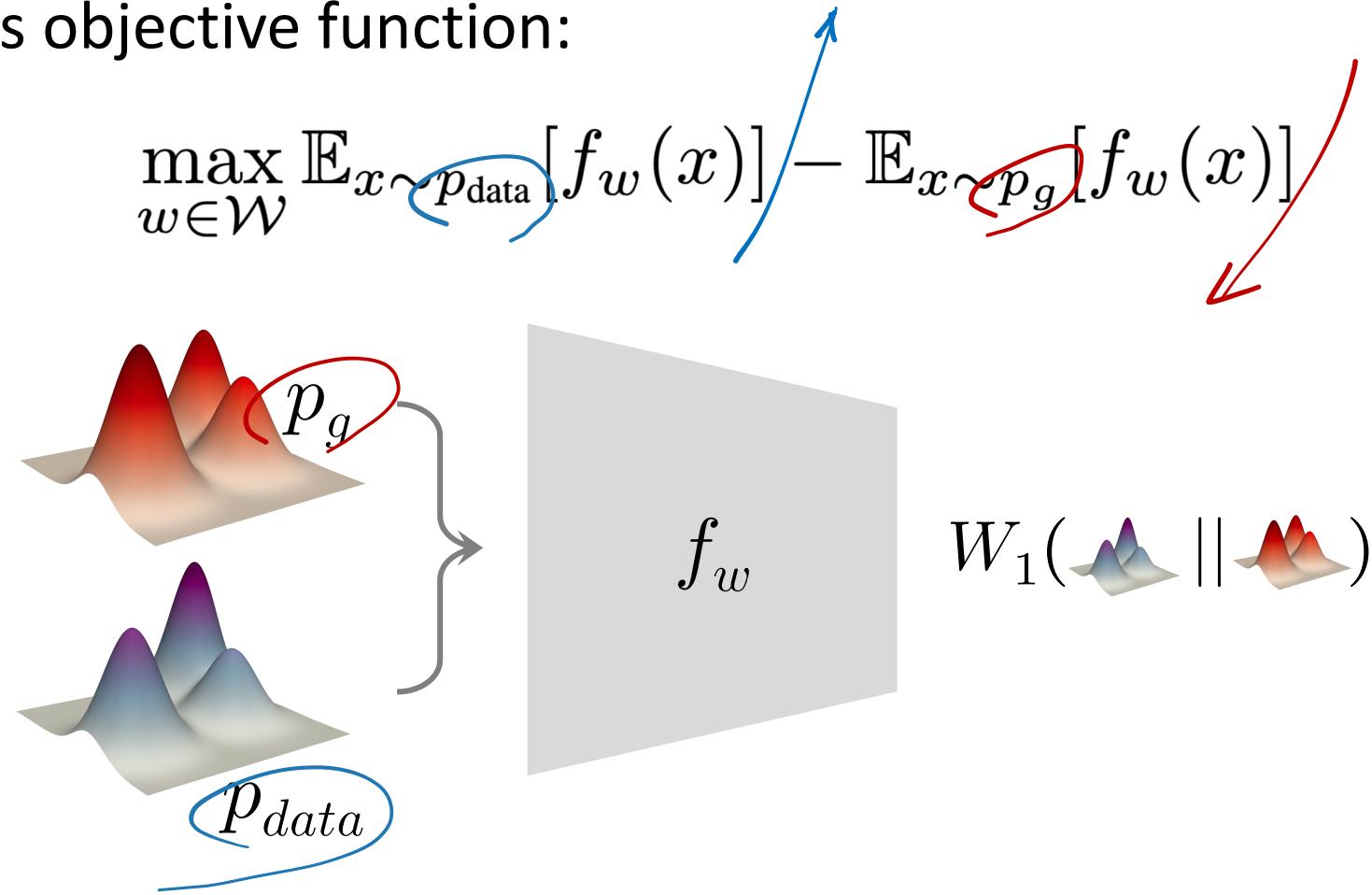
- W-GAN's objective function:

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



W-GAN optimizes for Wasserstein Distance

- W-GAN's objective function:

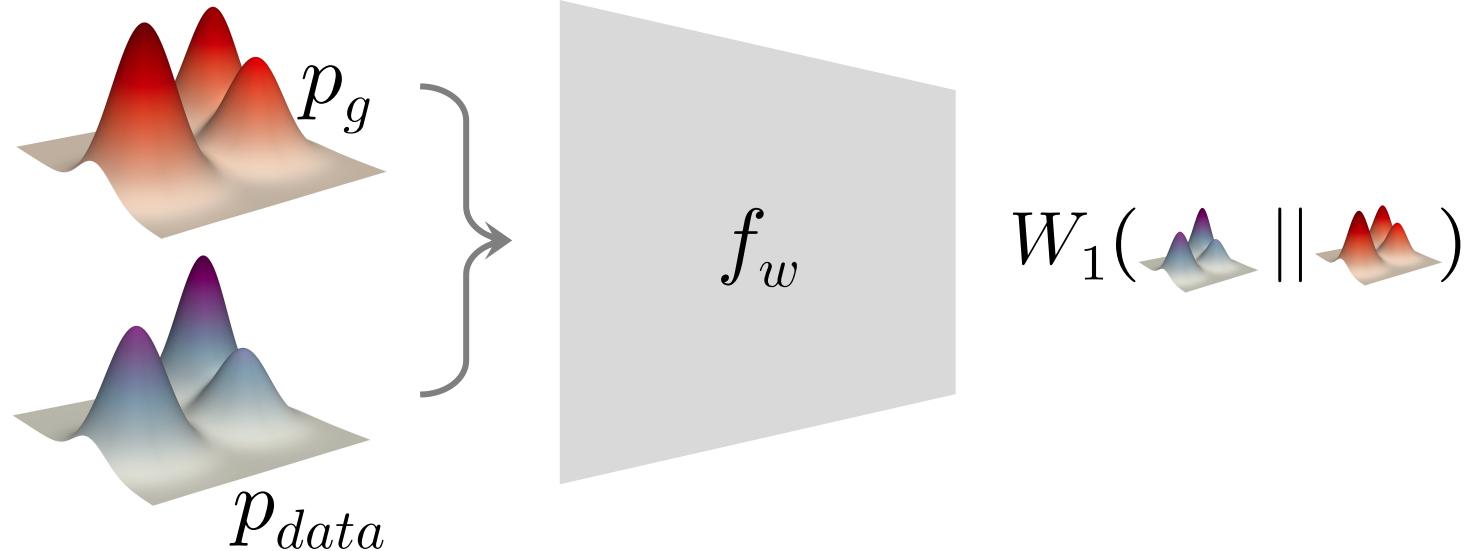


W-GAN optimizes for Wasserstein Distance

- 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

- W-GAN's objective function:

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

“art critic”

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

- 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 $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

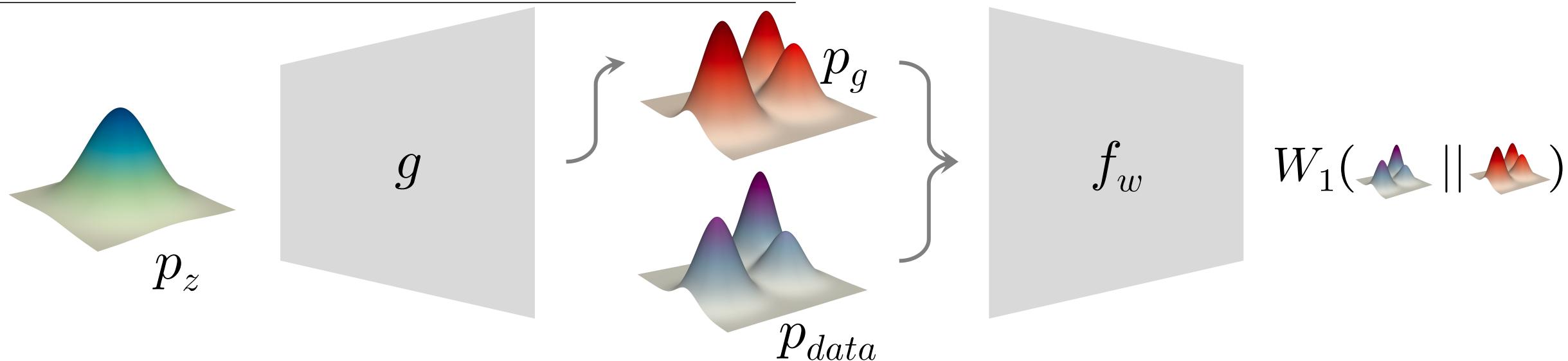
Require: : α , the learning rate. c , the clipping parameter. m , the batch size.
 n_{critic} , the number of iterations of the critic per generator iteration.

Require: : w_0 , initial critic parameters. θ_0 , initial generator's parameters.

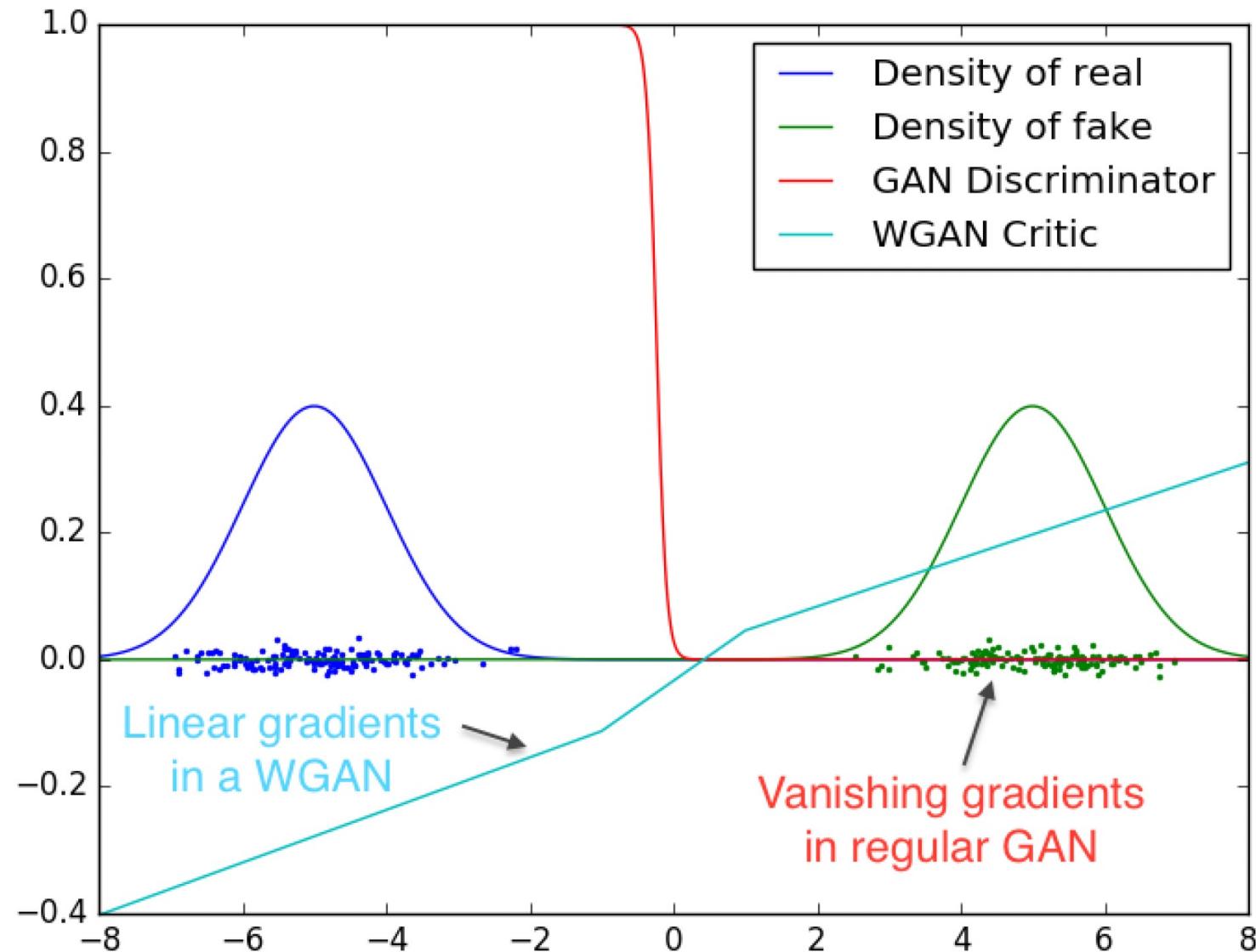
```

1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for clip weights
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$ 
12: end while
```

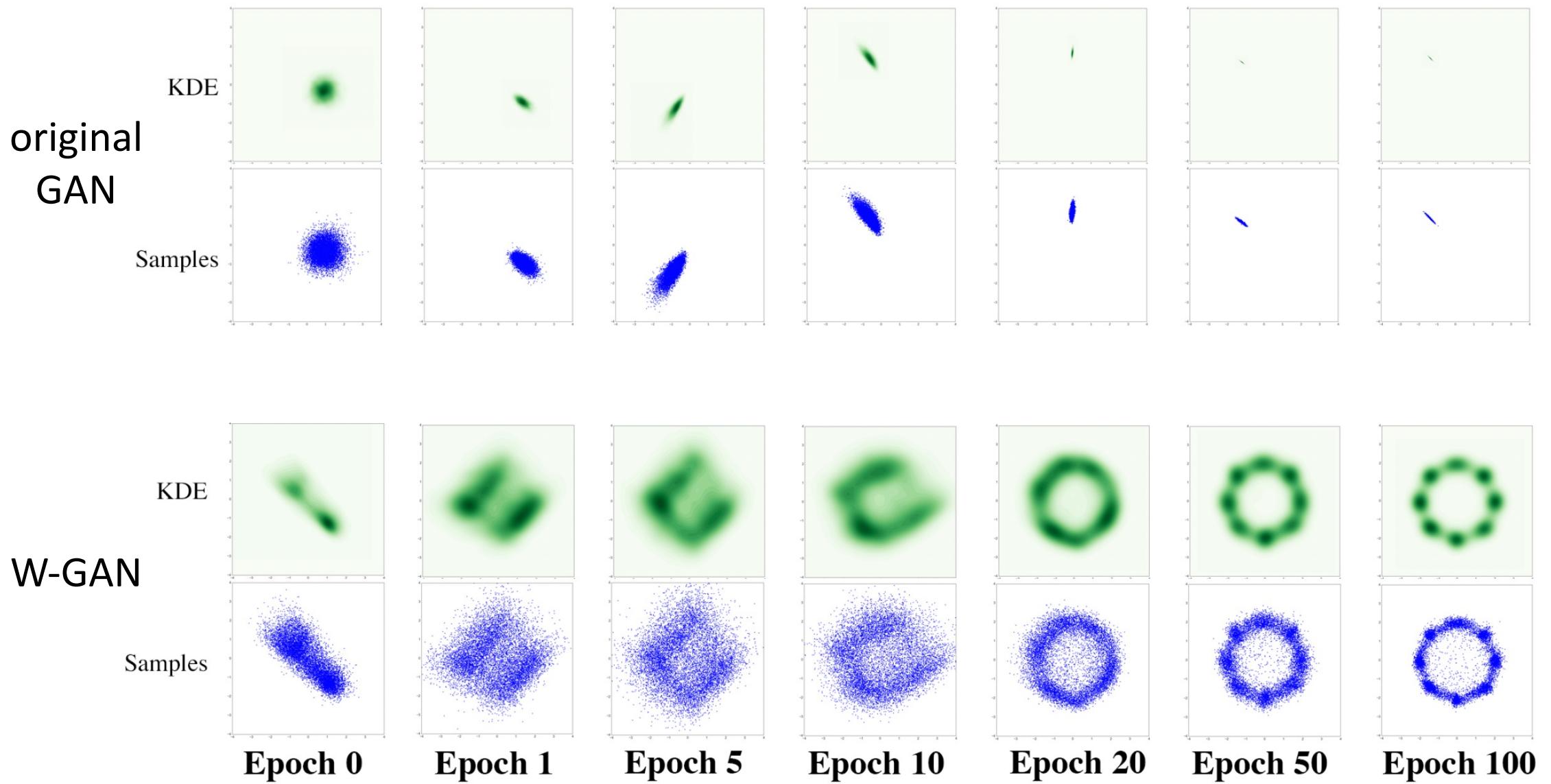
remove logarithms



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

Brief: LSGAN, EBGAN

- Least Square (LS) GAN:

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

- Energy-based (EB) GAN:

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

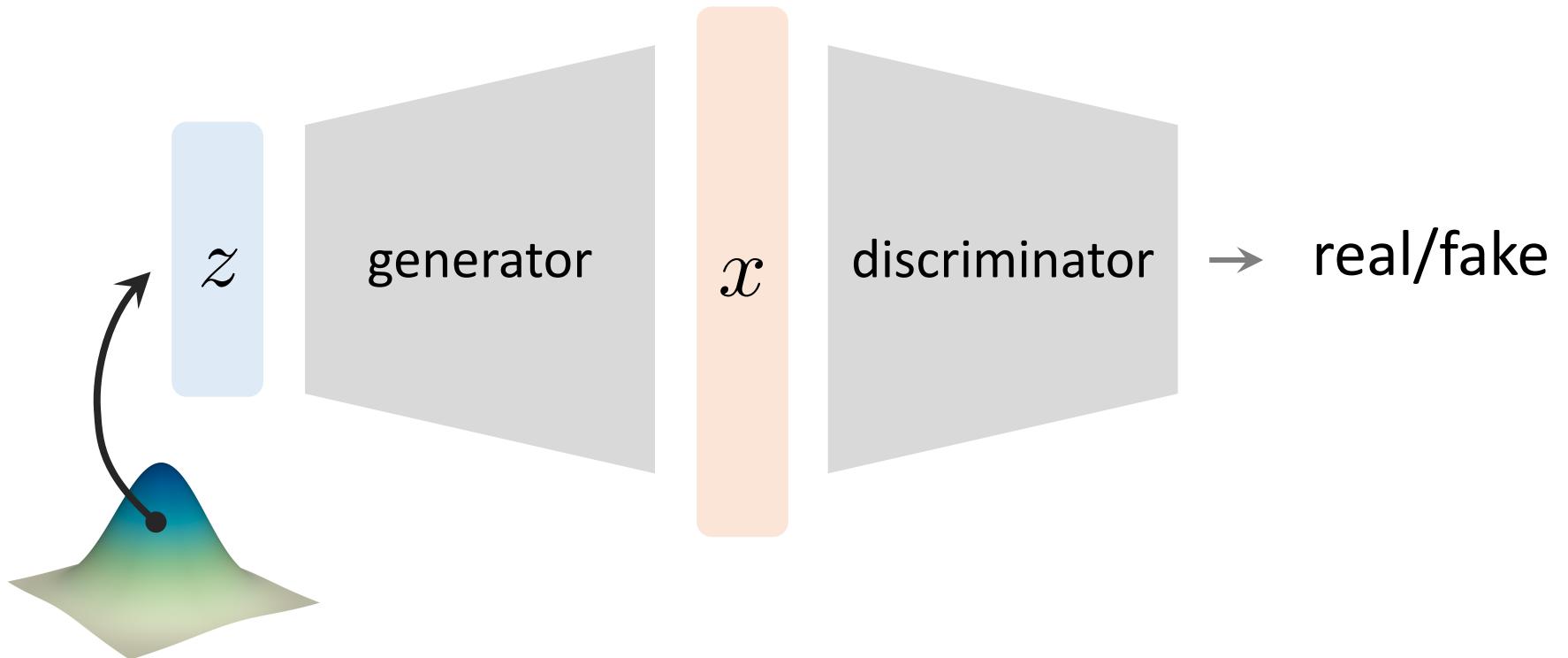
Adversary as a Loss Function

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

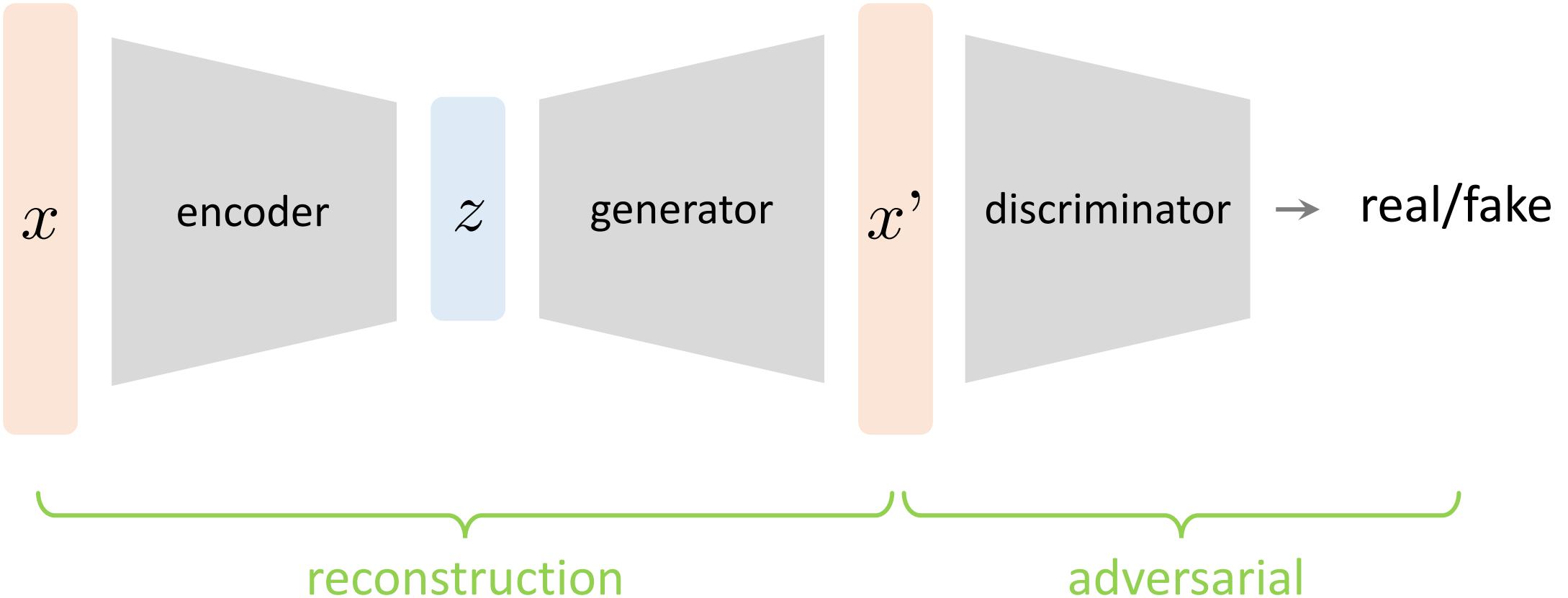
Adversary as a Loss Function

- GAN: input is random



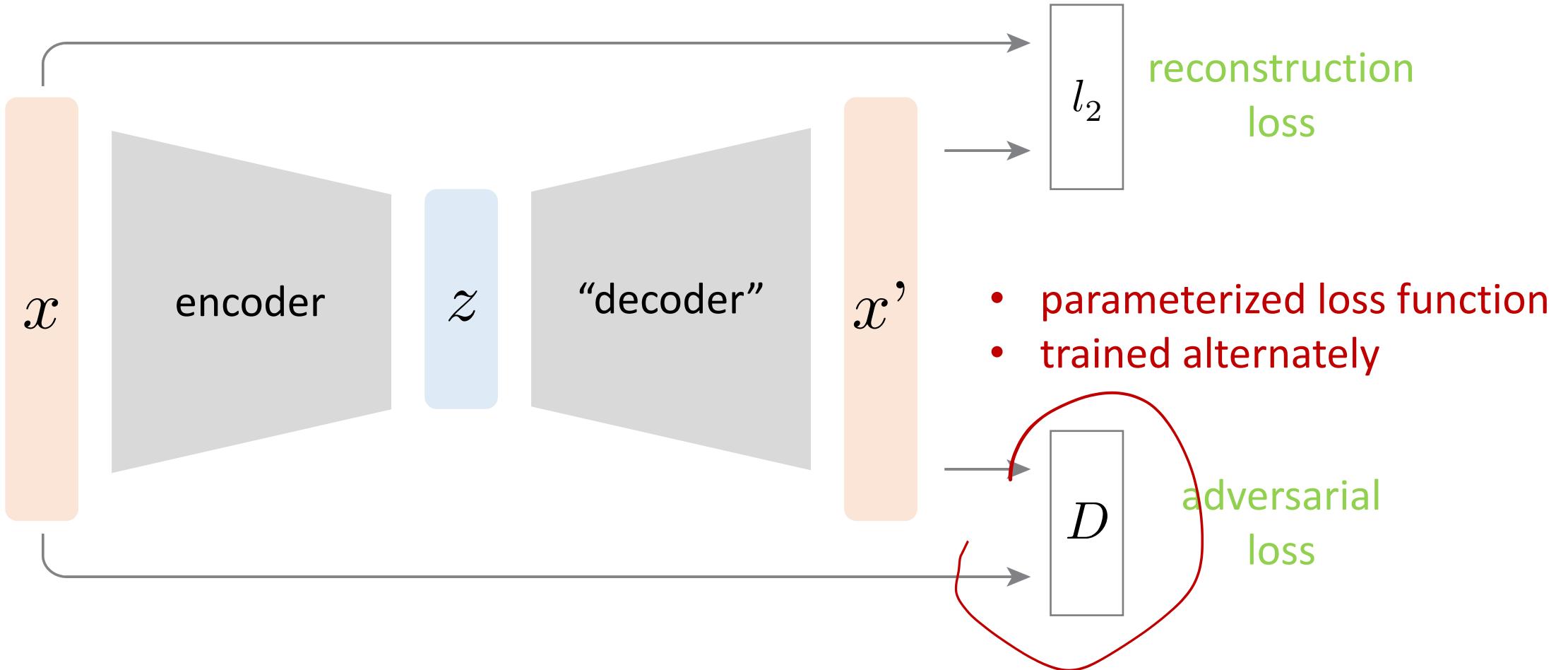
Adversary as a Loss Function

- Input can be from another source



Adversary as a Loss Function

- Input can be from another source



Example: Super-Resolution GAN

- better PSNR

original



bicubic
(21.59dB/0.6423)



SRResNet
(23.44dB/0.7777)

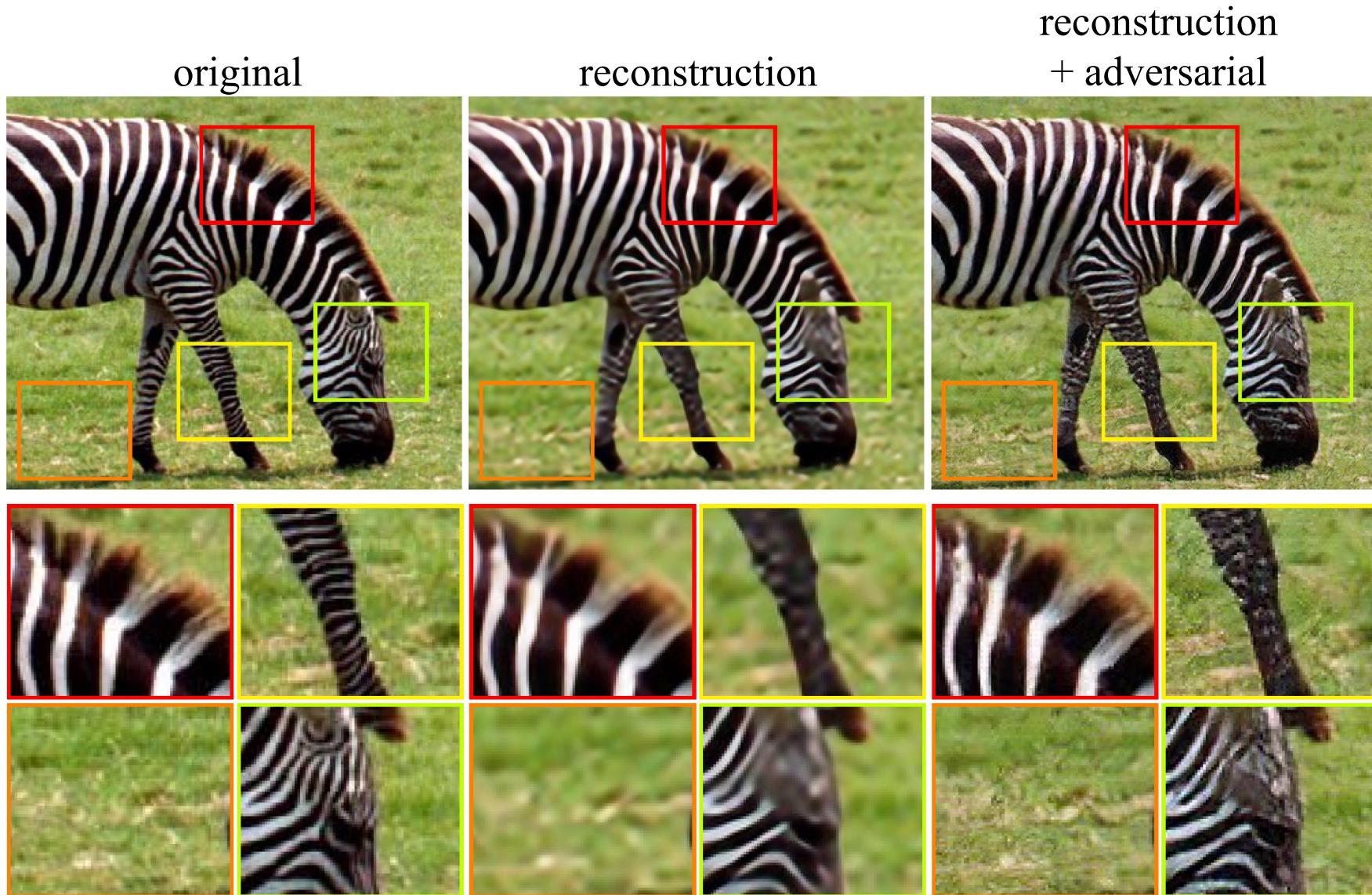


- worse PSNR, but better visual quality

SRGAN
(20.34dB/0.6562)



Example: Super-Resolution GAN



Example: Context Encoder

Image



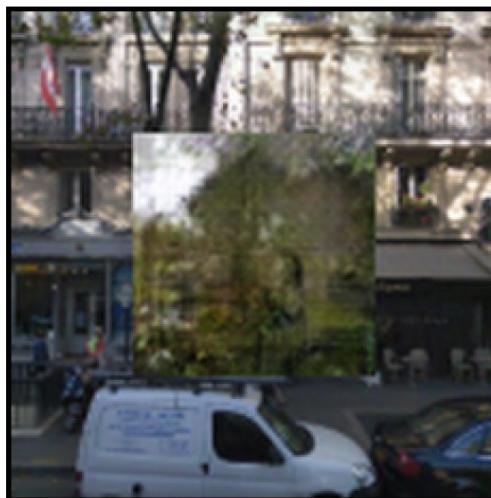
Ours(L_2)



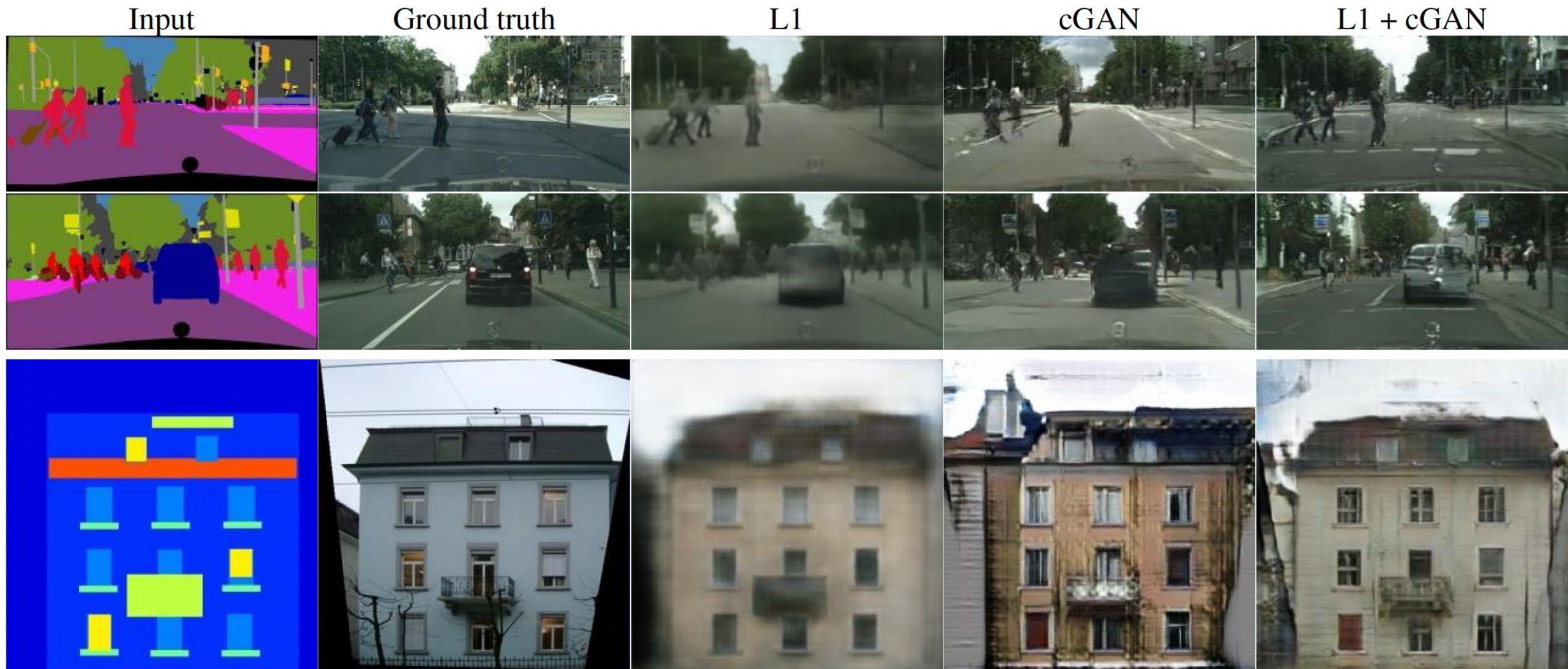
Ours(Adv)



Ours(L_2 +Adv)



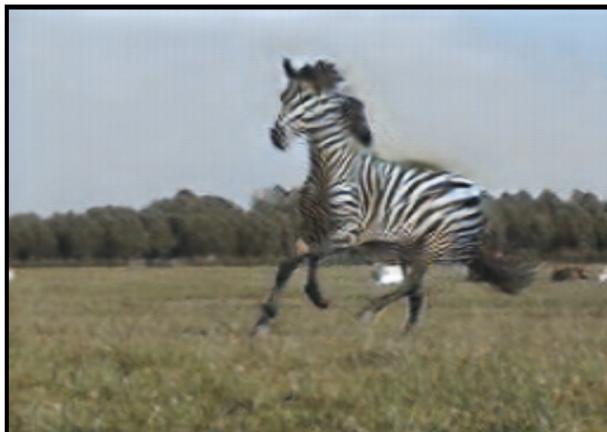
Example: pix2pix



Example: CycleGAN

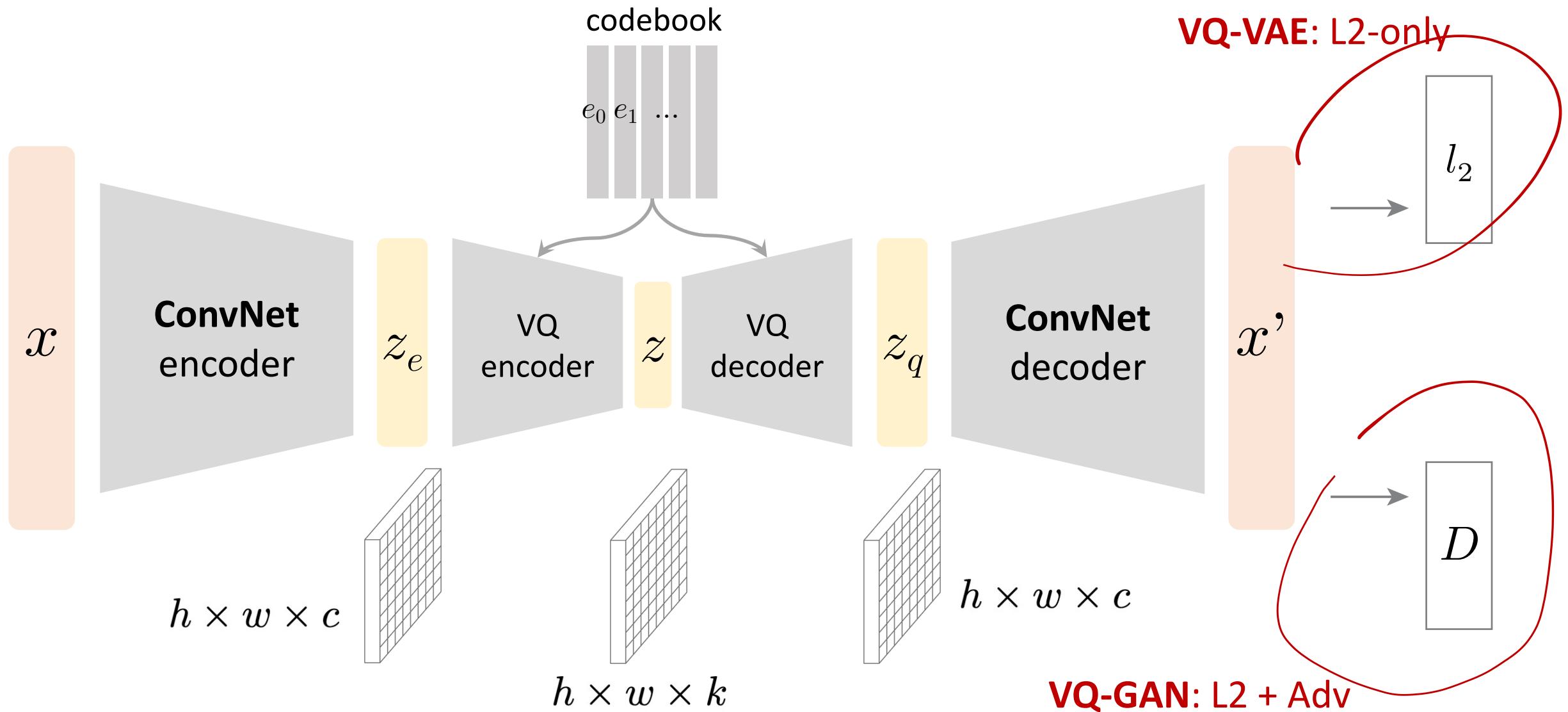


zebra → horse

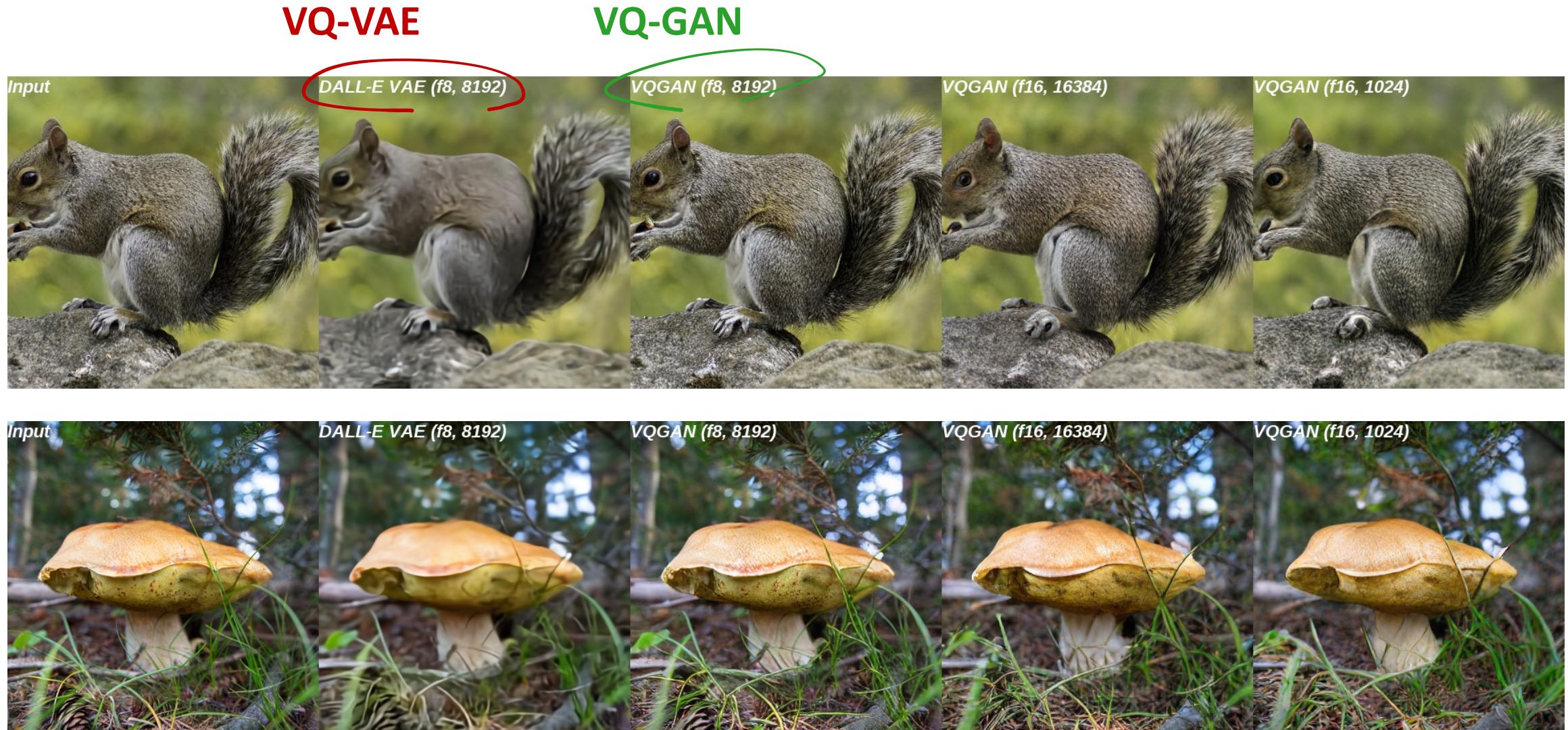


horse → zebra

From VQ-VAE to VQ-GAN



From VQ-VAE to VQ-GAN



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