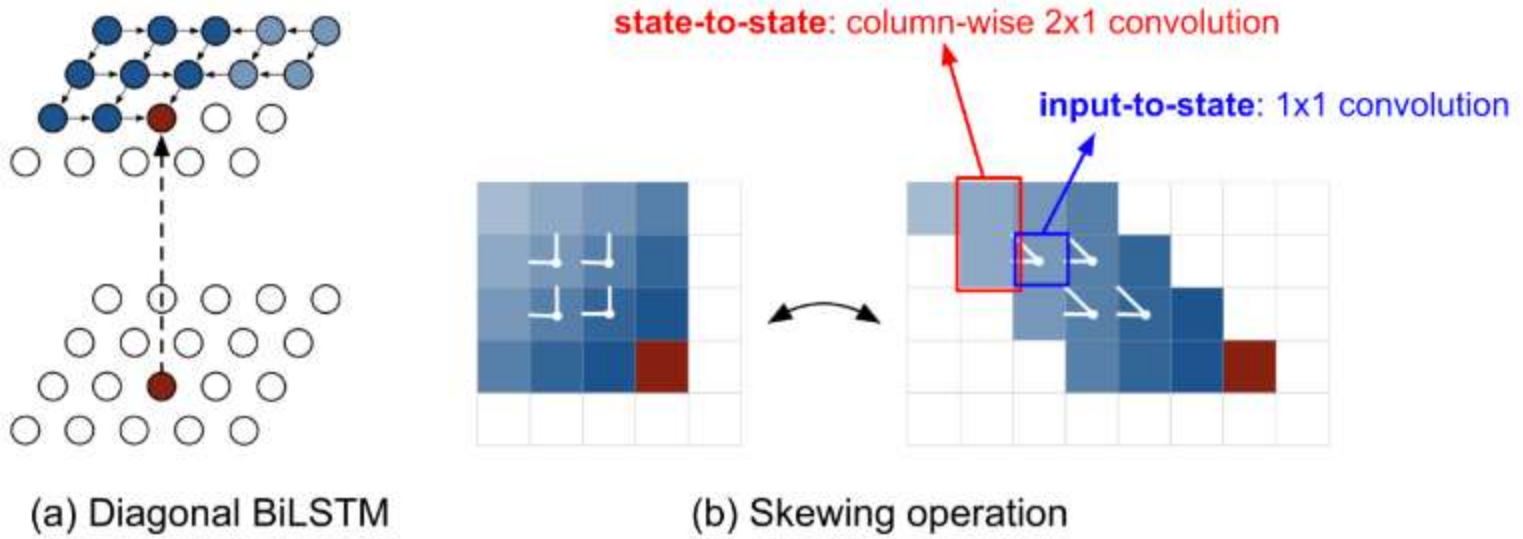
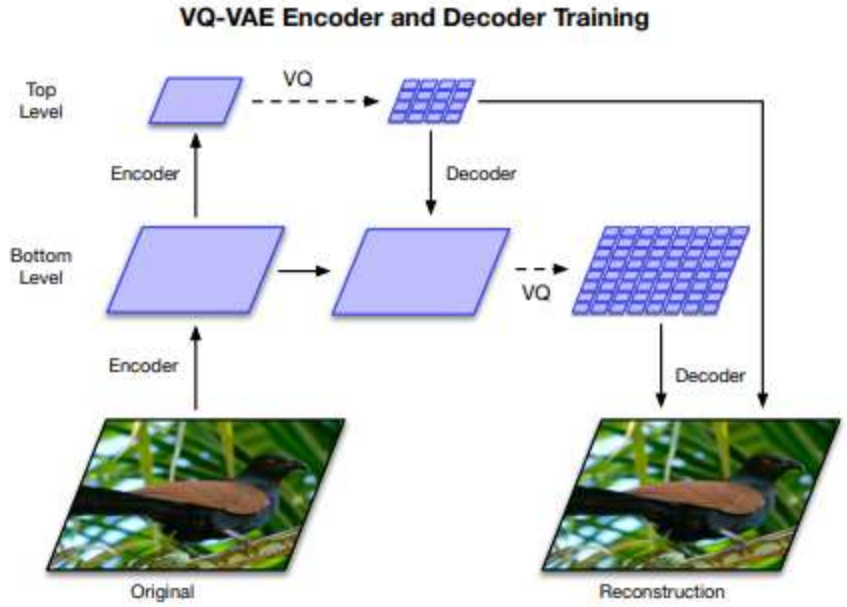
Recap.

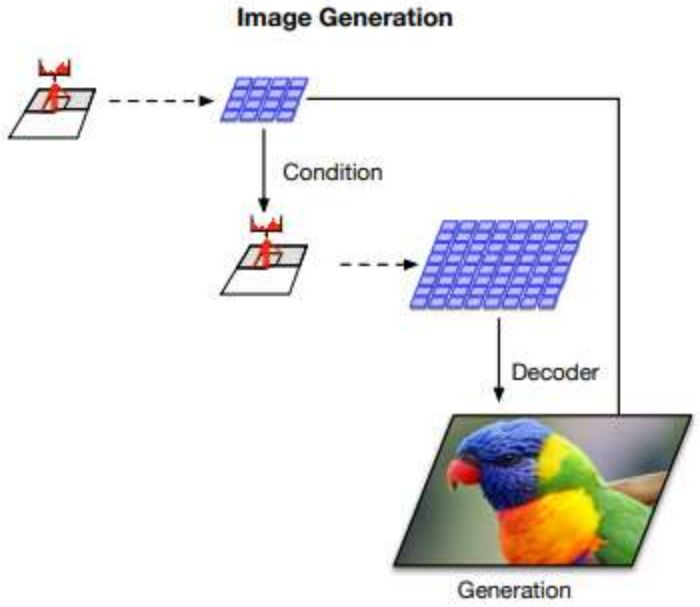
PixelRNN *[van der Oord et al. 2016]*



VQ-VAE and VQ-VAE2

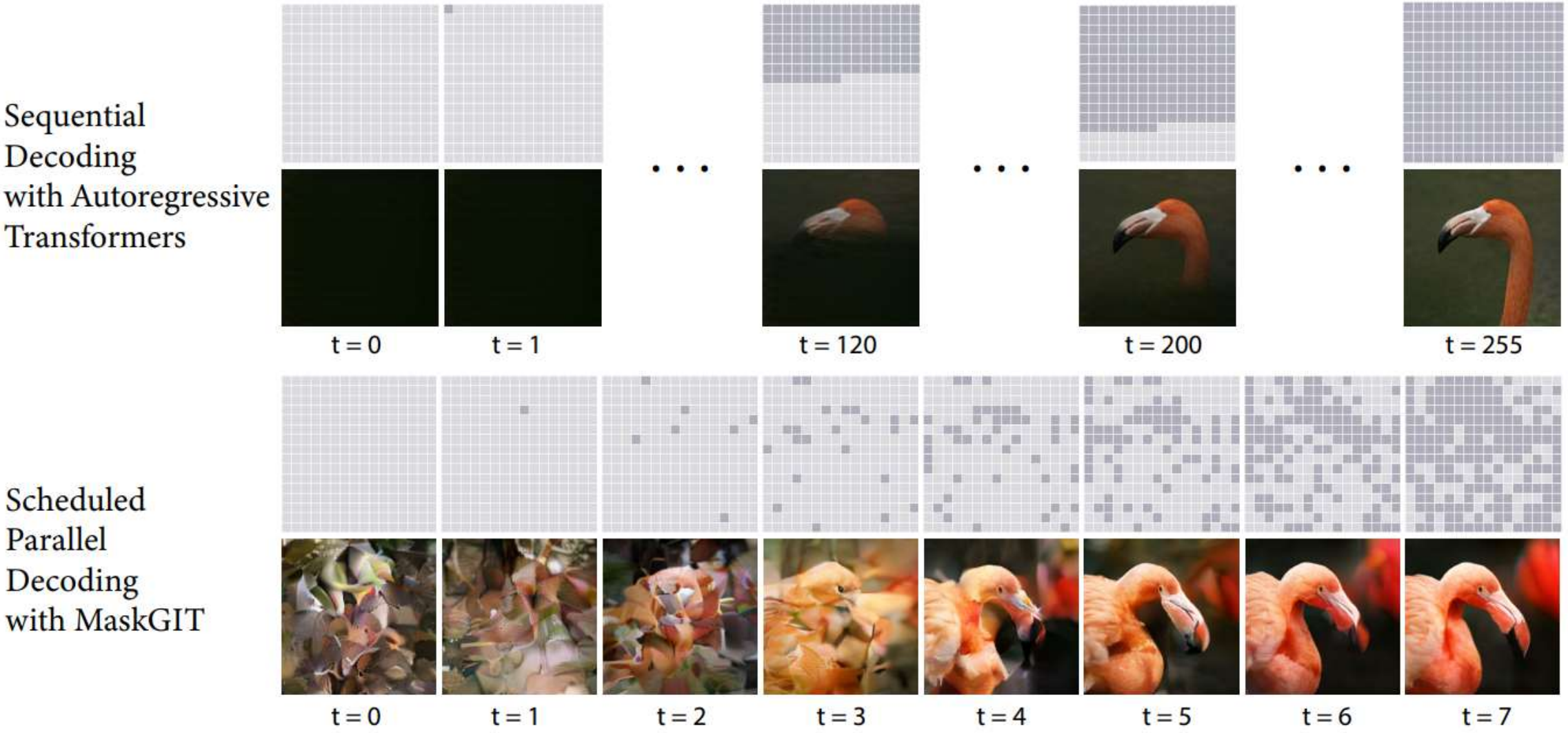


(a) Overview of the architecture of our hierarchical VQ-VAE. The encoders and decoders consist of deep neural networks. The input to the model is a 256×256 image that is compressed to quantized latent maps of size 64×64 and 32×32 for the *bottom* and *top* levels, respectively. The decoder reconstructs the image from the two latent maps.

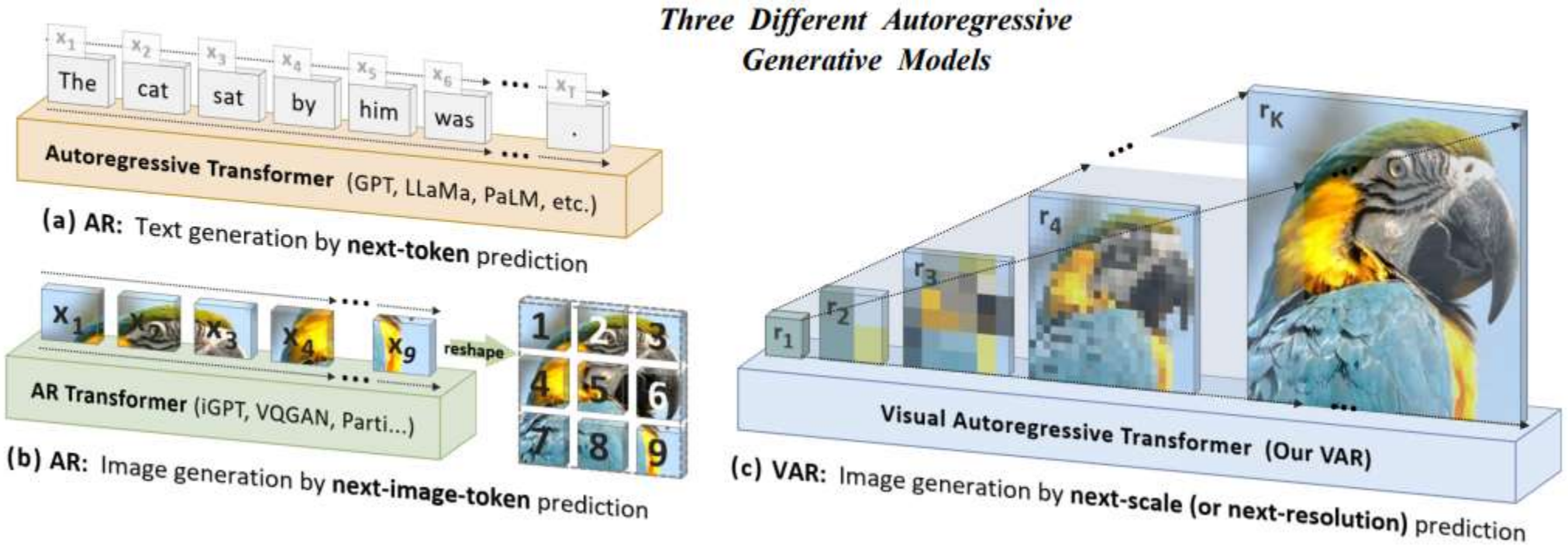


(b) Multi-stage image generation. The top-level PixelCNN prior is conditioned on the class label, the bottom level PixelCNN is conditioned on the class label as well as the first level code. Thanks to the feed-forward decoder, the mapping between latents to pixels is fast. (The example image with a parrot is generated with this model).

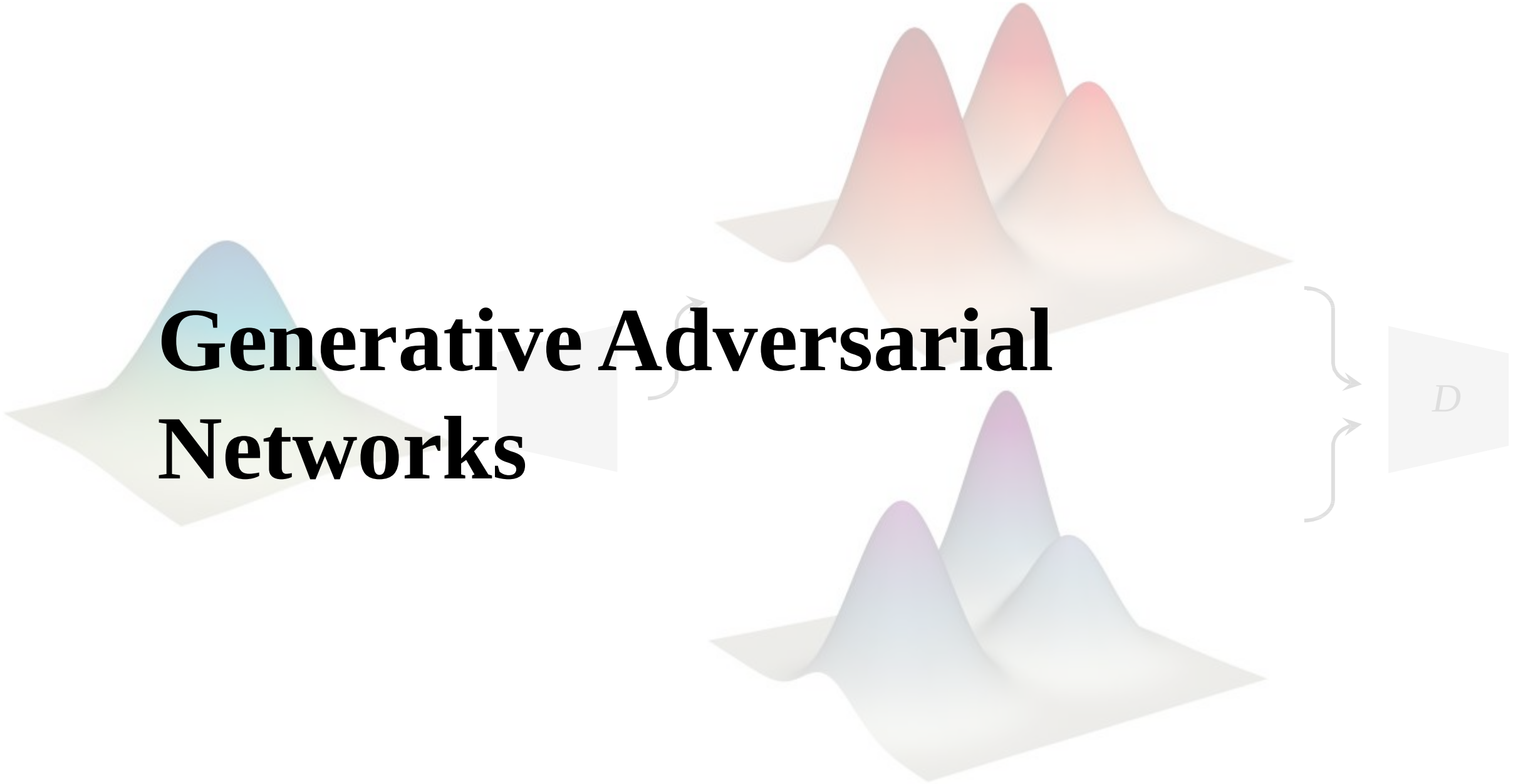
MaskGIT: Masked Generative Image Transformer



VAR: Visual Autoregressive Modeling



Generative Adversarial Networks



Overview

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

Generative Adversarial Networks (GAN)

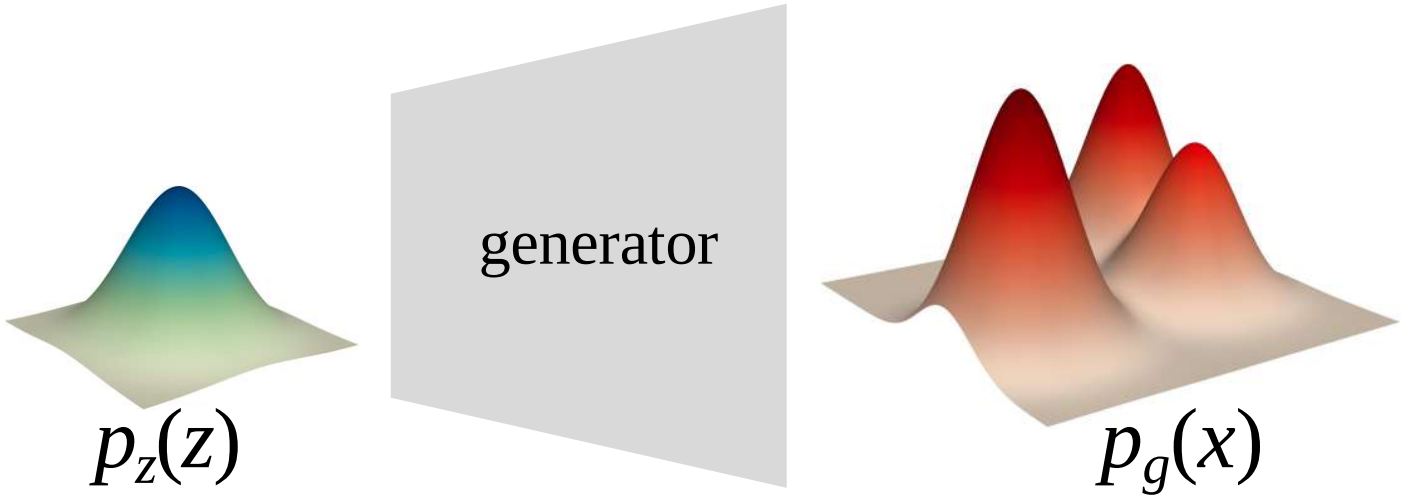
Introduction

- “**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

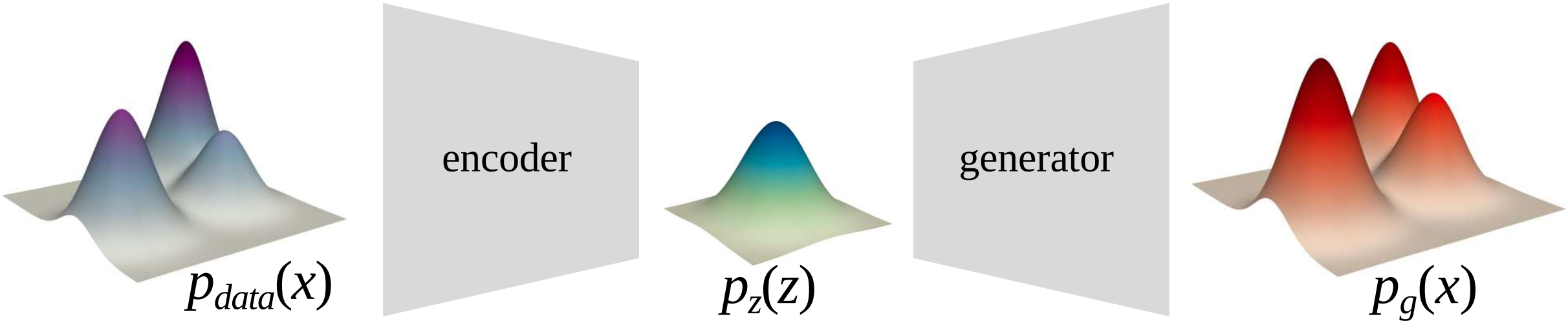


See Lecture 2, Variational Autoencoder: https://mit-6s978.github.io/assets/pdfs/lec2_vae.pdf

Recap: Variational Autoencoder (VAE)

Autoencoding distributions:

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



"Reconstruction" loss

What's the implication of a "*reconstruction*" loss?

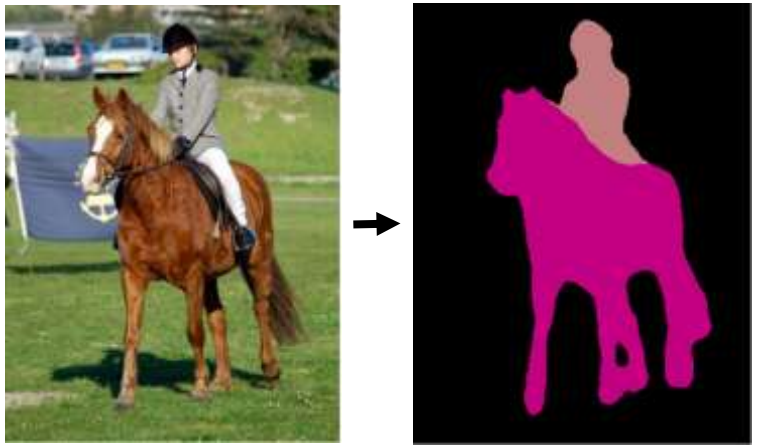
- Elements (e.g., pixels) are **independently** distributed
- Each element follows a **simple** distribution (Gaussian/Bernoulli/...)

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

Can we measure the distribution difference in another way?

Data translation problems ("structured prediction")

Semantic segmentation



[Long et al. 2015, …]

Edge detection

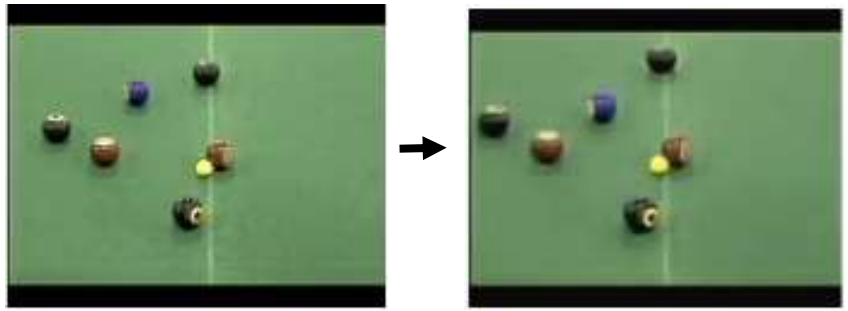
[Xie et al. 2015, …]

Text-to-photo

"this small bird has a pink breast and crown…"

[Reed et al. 2014, …]

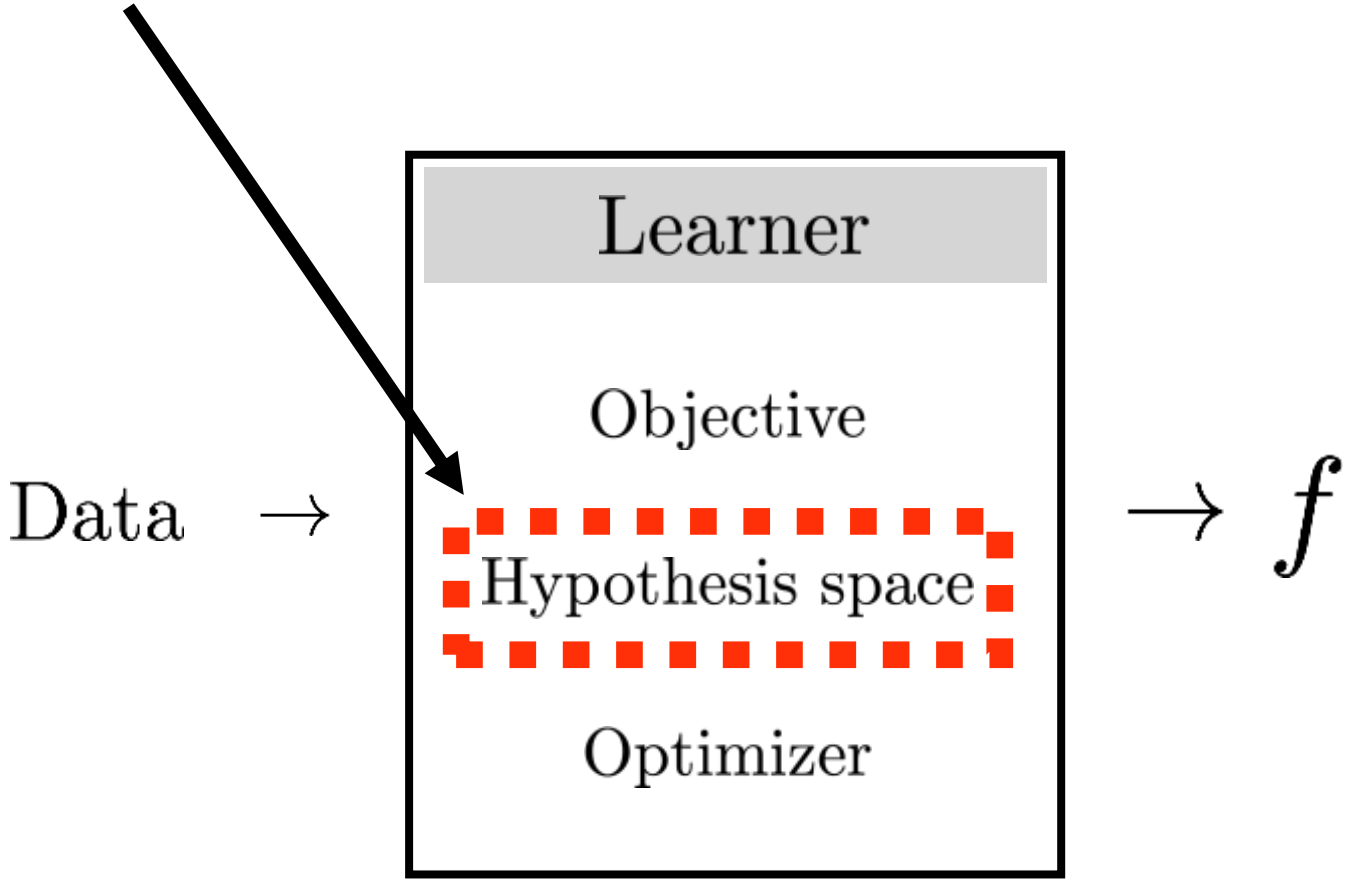
Future frame prediction



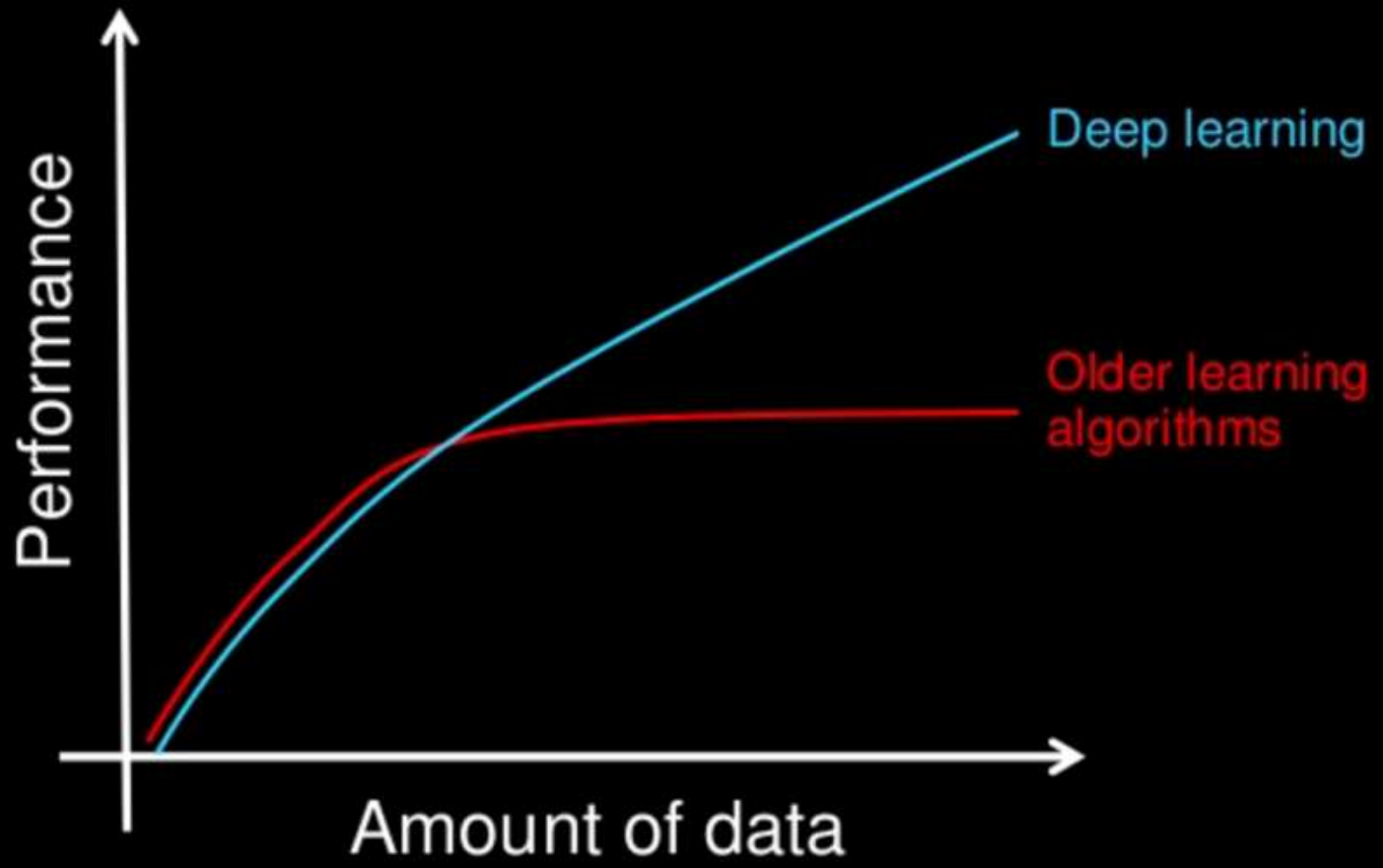
[Mathieu et al. 2016, …]

Deep learning in 2012

Use a **hypothesis space** that can model complex structure
(e.g., a CNN, nearest-neighbor)



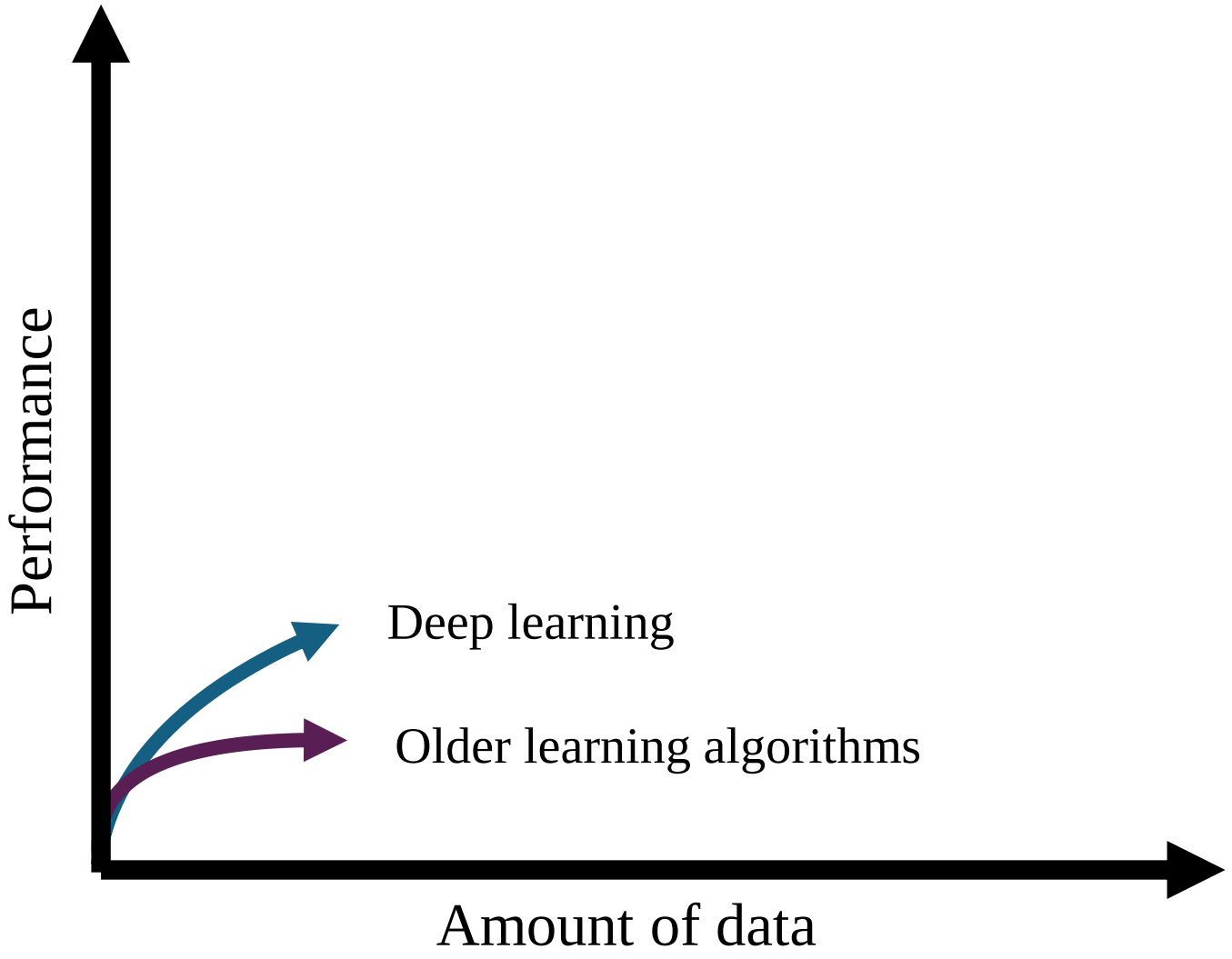
Why deep learning



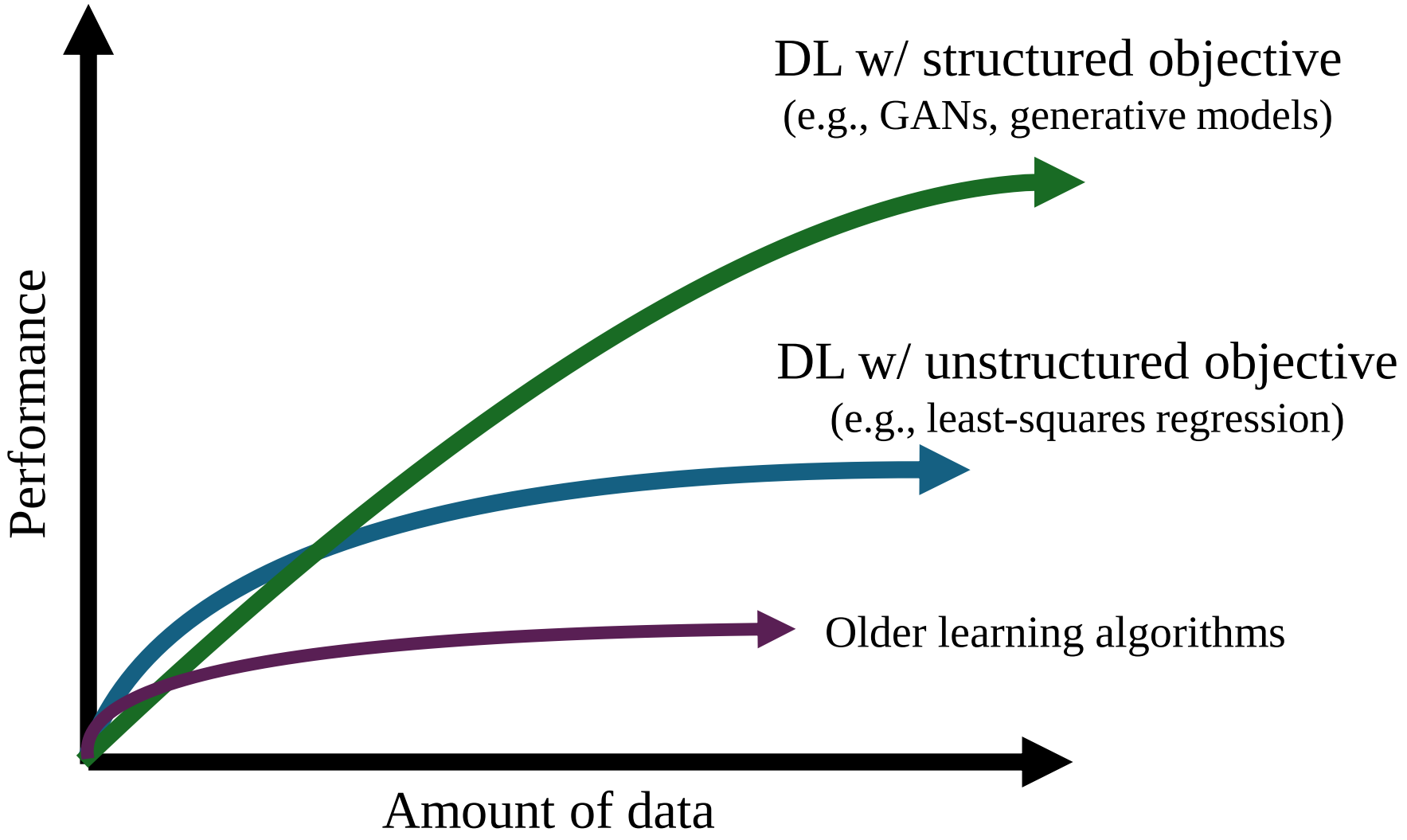
How do data science techniques scale with amount of data?

[Slide credit: Andrew Ng]

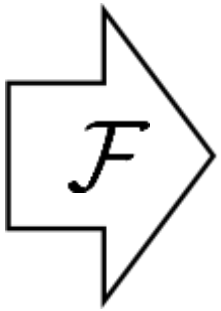
Why structured objectives

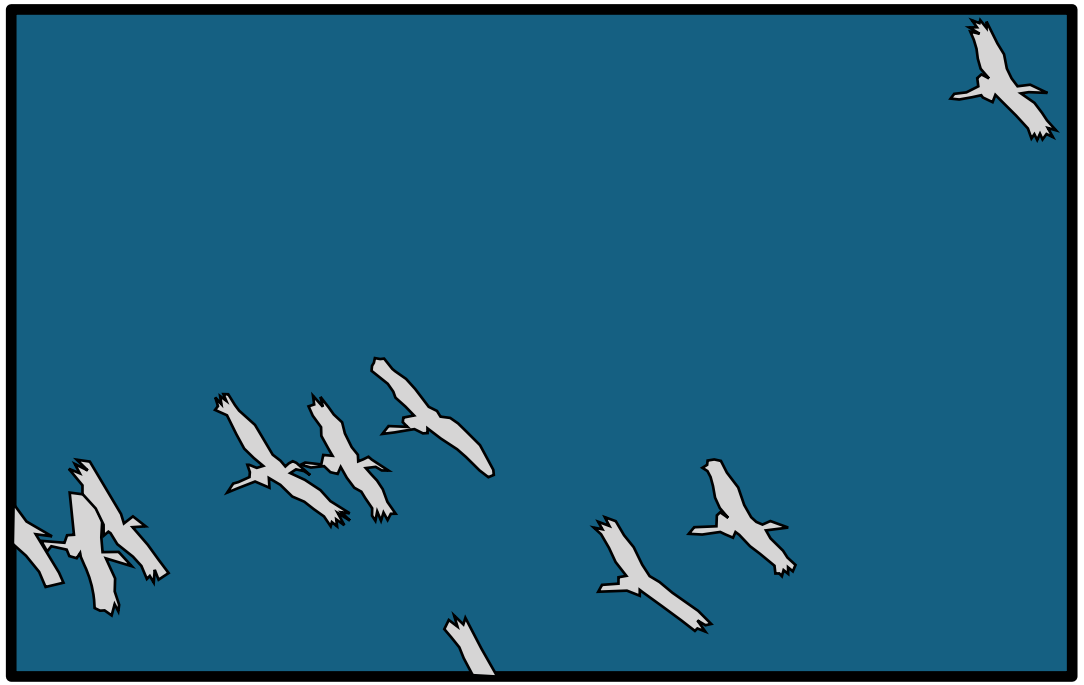


Why structured objectives



[Photo credit: Fredo Durand]





(Colors represent one-hot codes)

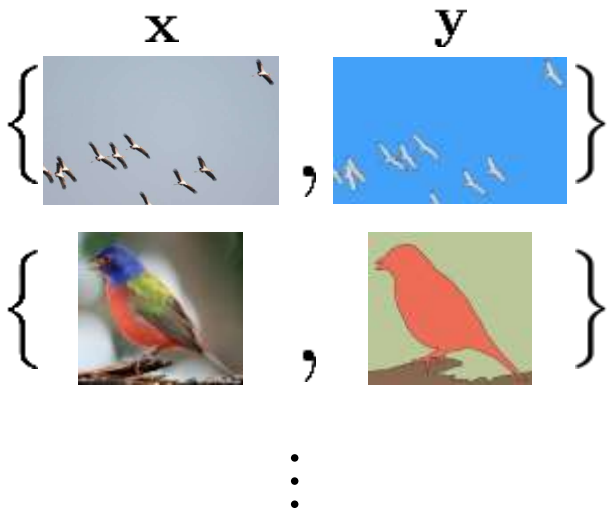
$$\arg\min_{\mathcal{F}} \mathbb{E}_{\mathbf{x},\mathbf{y}}[L(\mathcal{F}(\mathbf{x}), \mathbf{y})]$$

Hypothesis space

Objective function (loss)

Semantic Segmentation

Data



$$\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$$
$$\mathbf{y} \in \mathbb{R}^{H \times W \times K}$$

$\rightarrow$

Learner

Objective

$$f^* = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} H(\mathbf{y}_i, \hat{\mathbf{y}}_i)$$

Hypothesis space

Convolutional neural net

Optimizer

Stochastic gradient descent

$\rightarrow$ f

Sat2Map

Data

$\mathbf{x}$ $\mathbf{y}$

$$\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$$
$$\mathbf{y} \in \mathbb{R}^{H \times W \times 3}$$

$\rightarrow$

Learner

Objective

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} (f_\theta(\mathbf{x})_i - y_i)^2$$

Hypothesis space

Convolutional neural net

Optimizer

Stochastic gradient descent

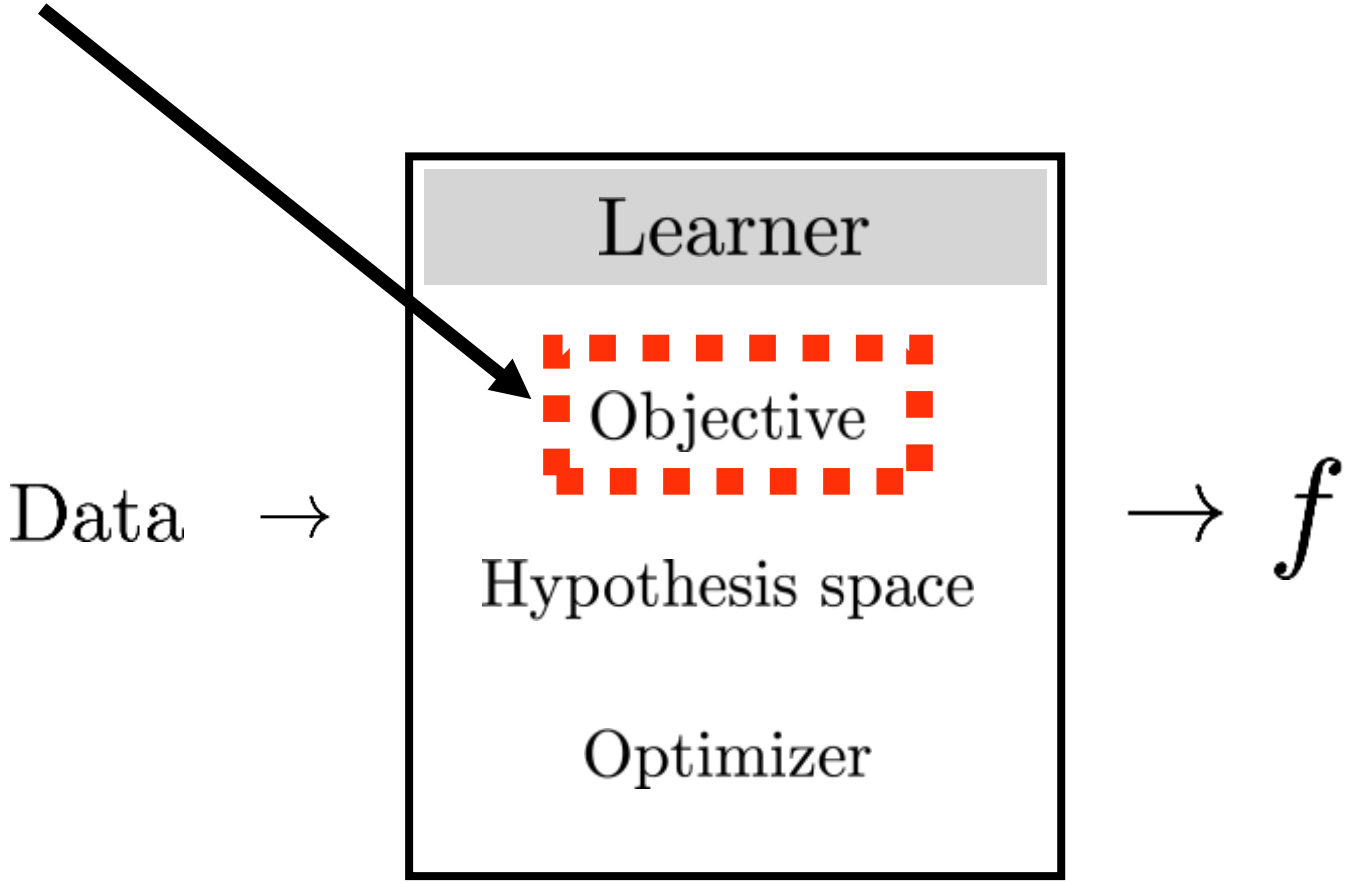
$\rightarrow$ f

Input

Deep net output

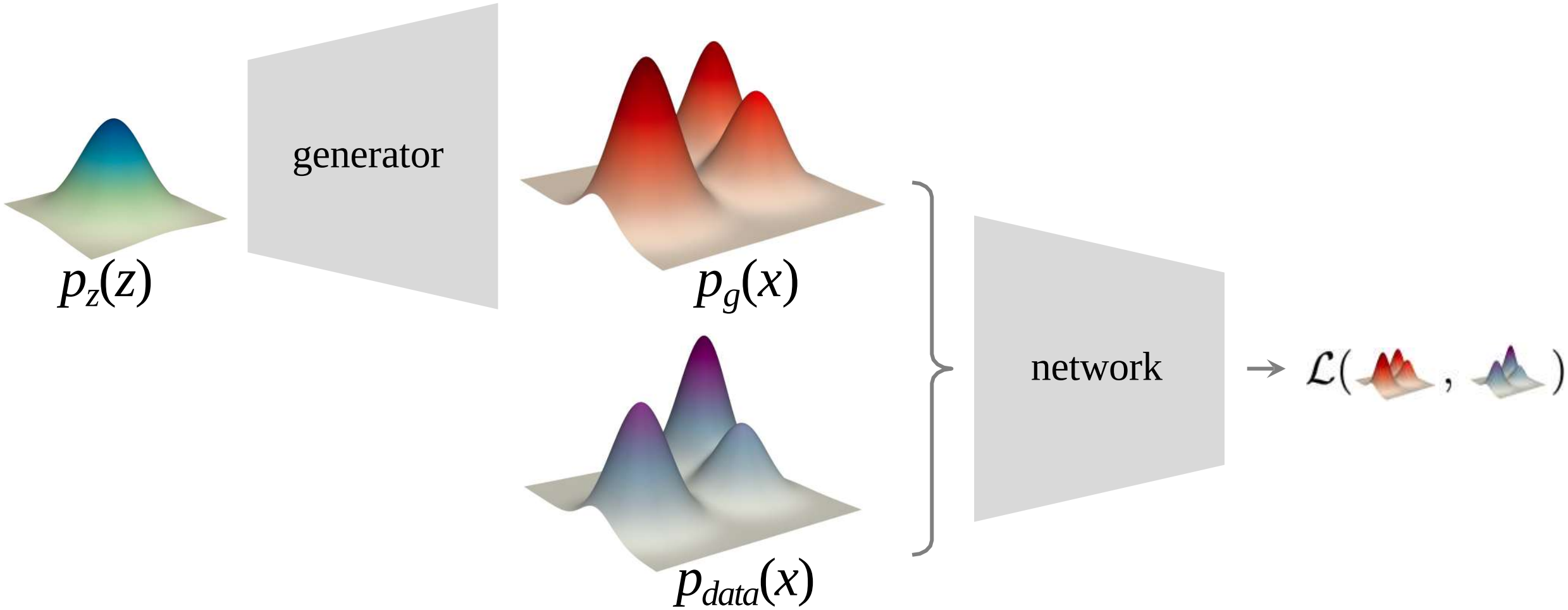
Structured prediction

Use an **objective** that can model structure! (e.g., a graphical model, a GAN, etc)



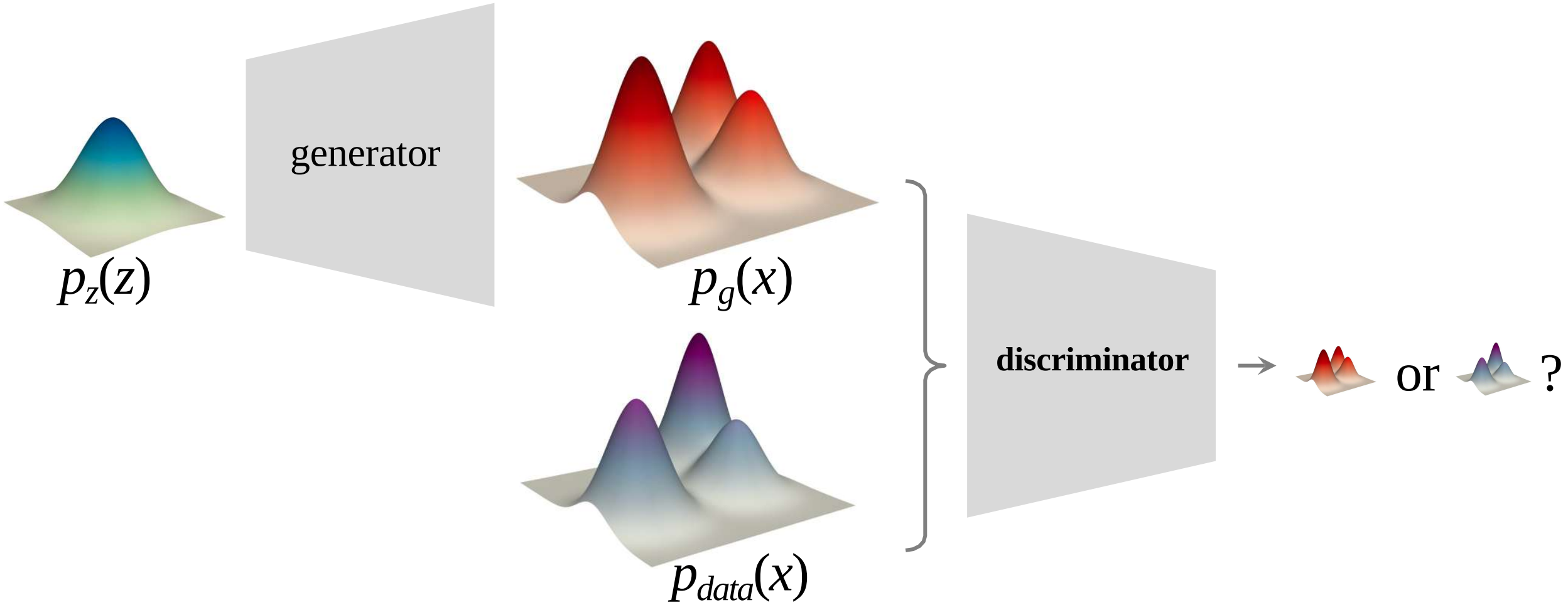
Generative Adversarial Networks

Representing **distribution difference** by a neural network



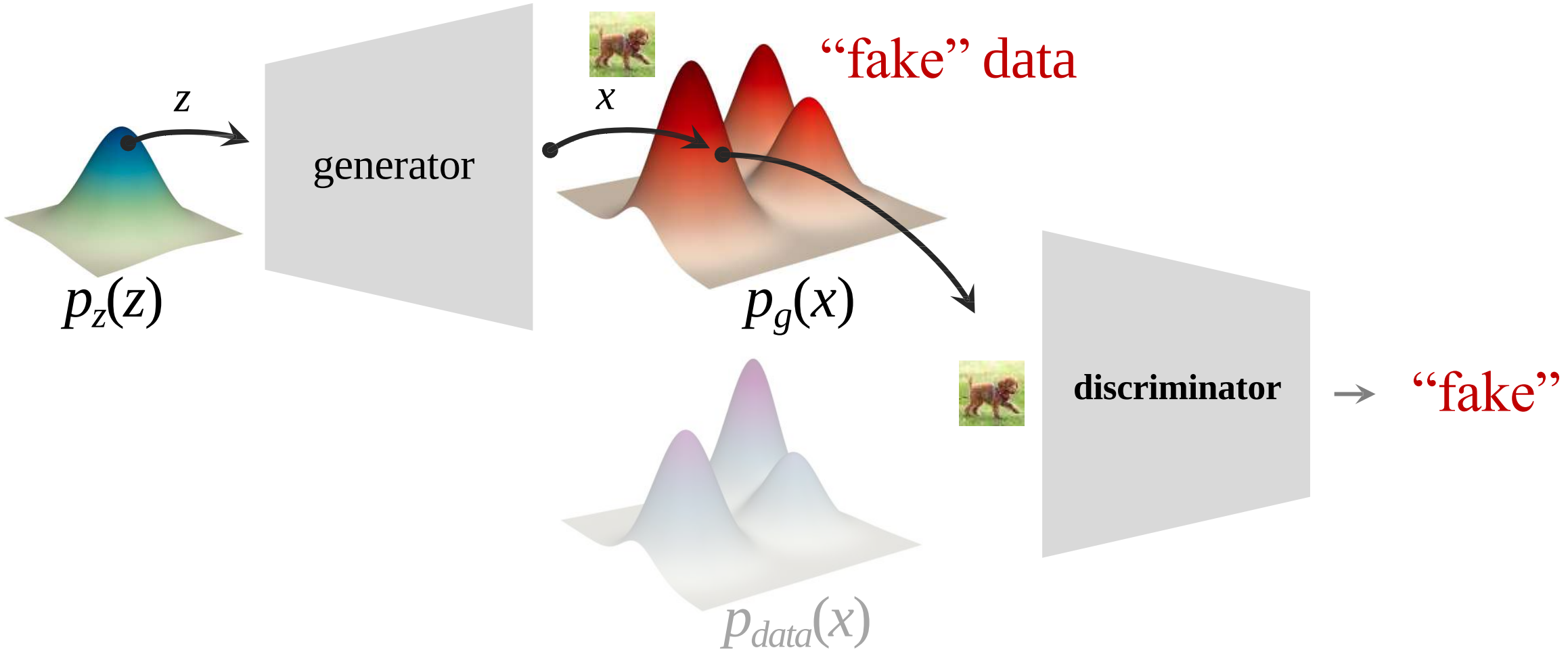
Generative Adversarial Networks

Representing **distribution difference** by a neural network



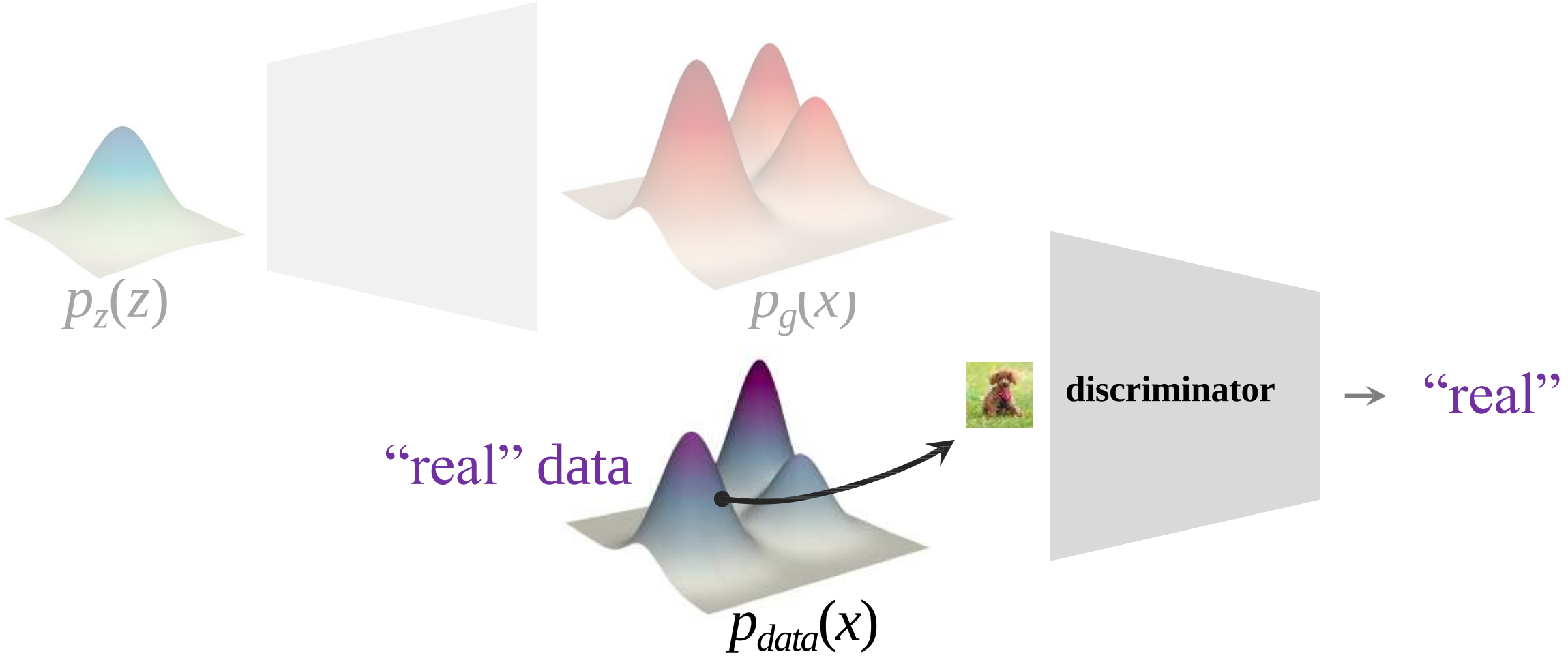
Generative Adversarial Networks

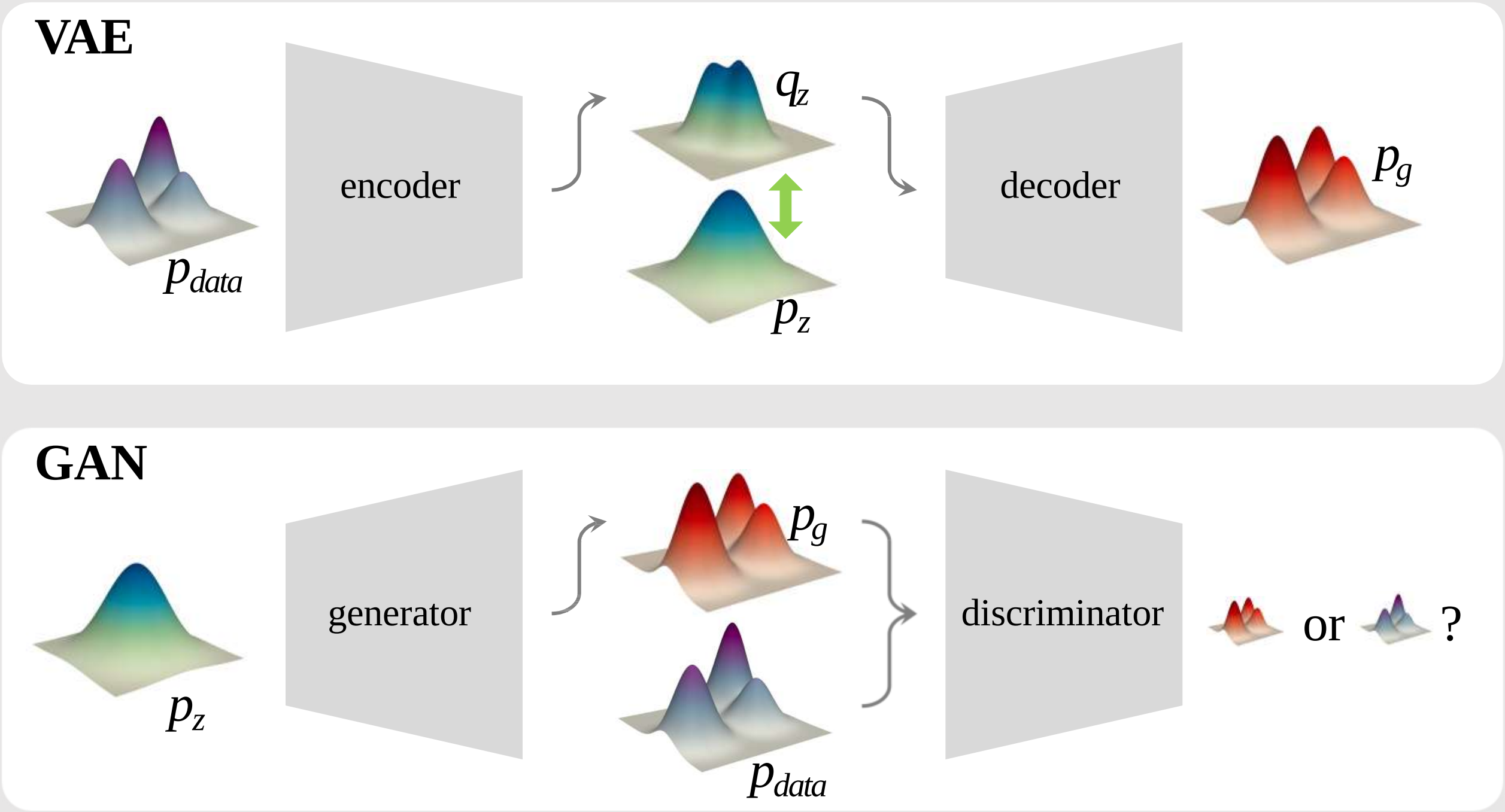
Representing **distribution difference** by a neural network



Generative Adversarial Networks

Representing **distribution difference** by a neural network



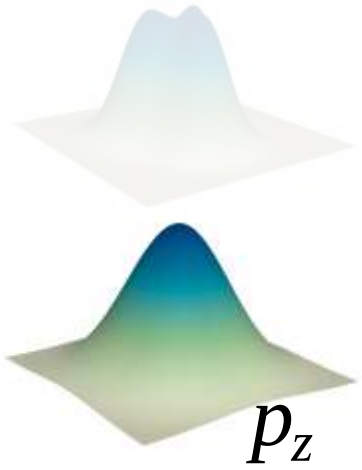

VAE
p_{data}
encoder
q_z
p_z
decoder
p_g
GAN
p_z
generator
p_g
p_{data}
discriminator
or
?

VAE

generation

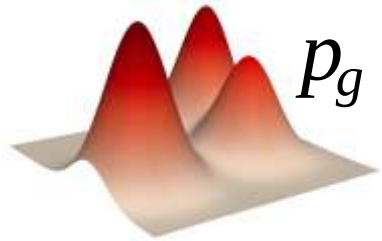
p_{data}

encoder



p_z

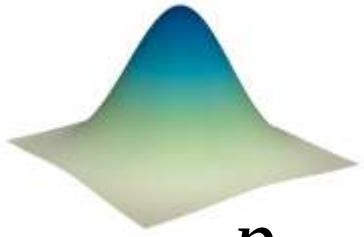
decoder



p_g

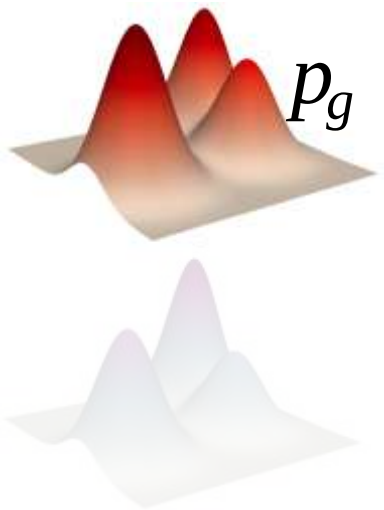
GAN

generation



p_z

generator



p_g

discriminator

or

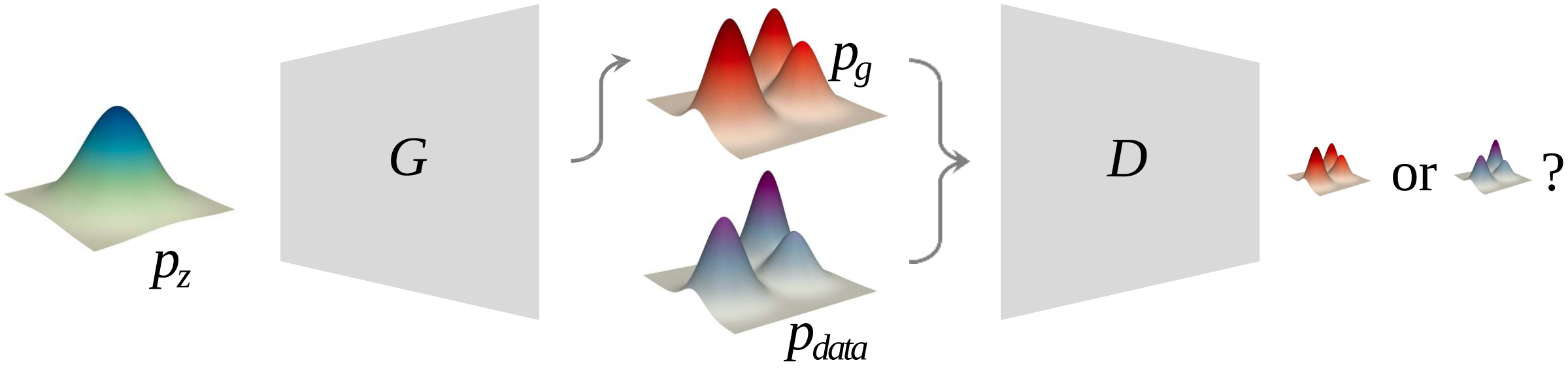
?

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

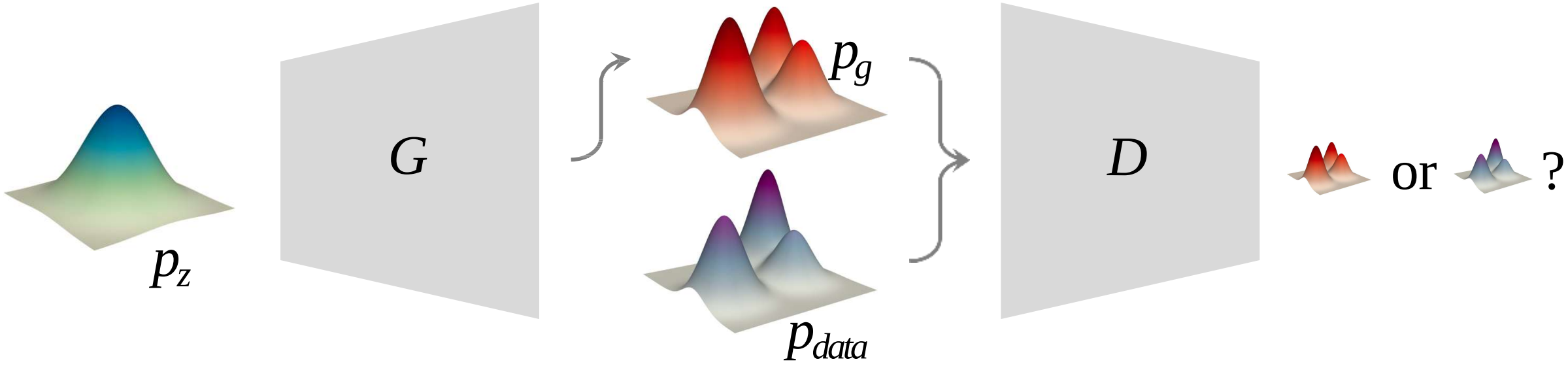
(vs. EM's max-max process)



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*



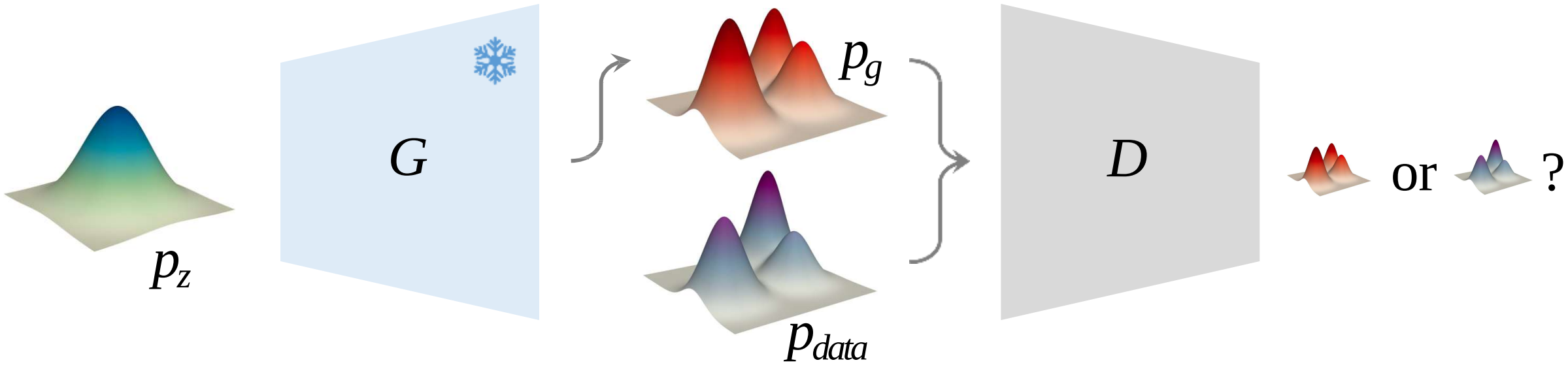
Adversarial Objective: D-step

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

push to 1 push to 0

D-step: fix *G*, optimize *D*

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



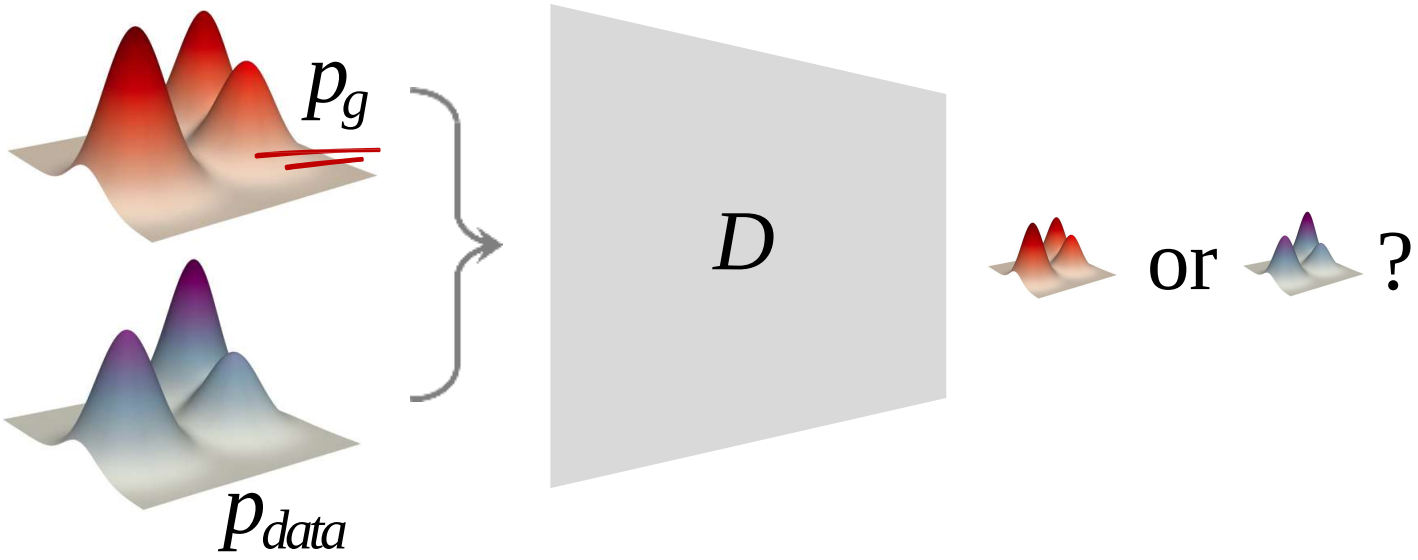
Adversarial Objective: D-step

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

push to 1 push to 0

D-step: fix *G*, optimize *D*

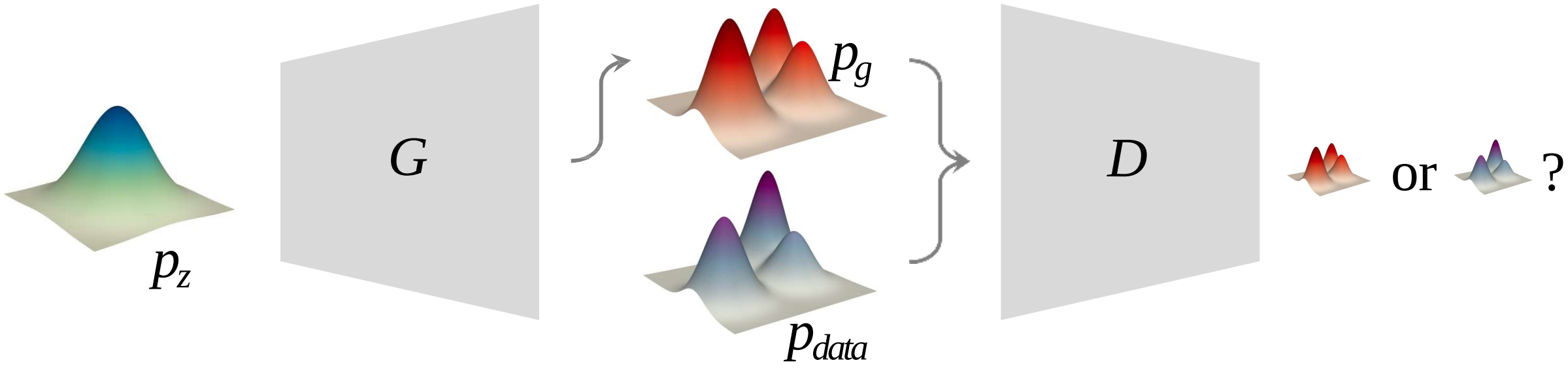
- *D* to classify real or fake
- binary logistic regression (sigmoid + cross-entropy)



Adversarial Objective: G-step

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

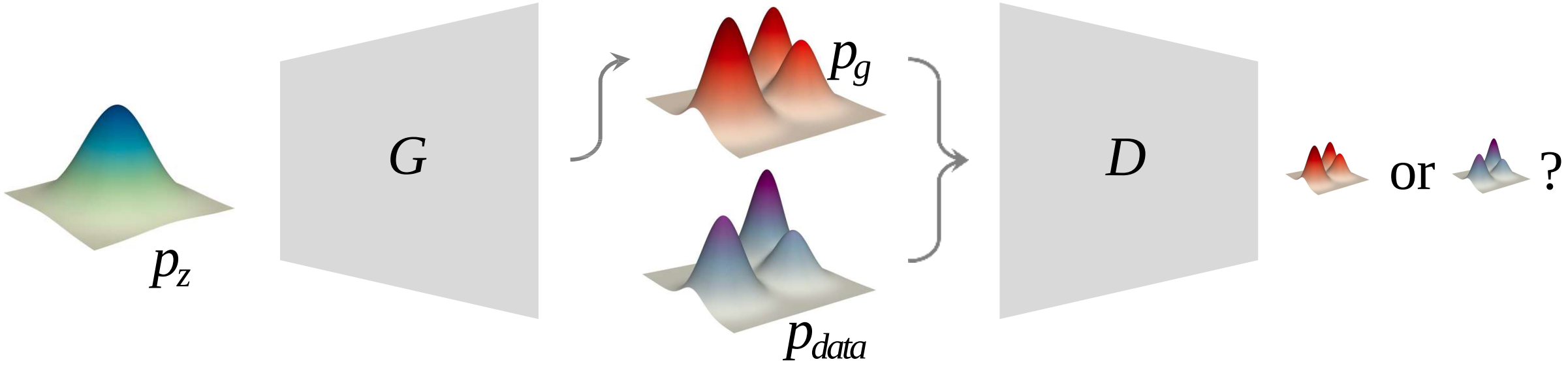
G-step: fix *D*, optimize *G*



Adversarial Objective: G-step

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

G-step: fix *D*, optimize *G*

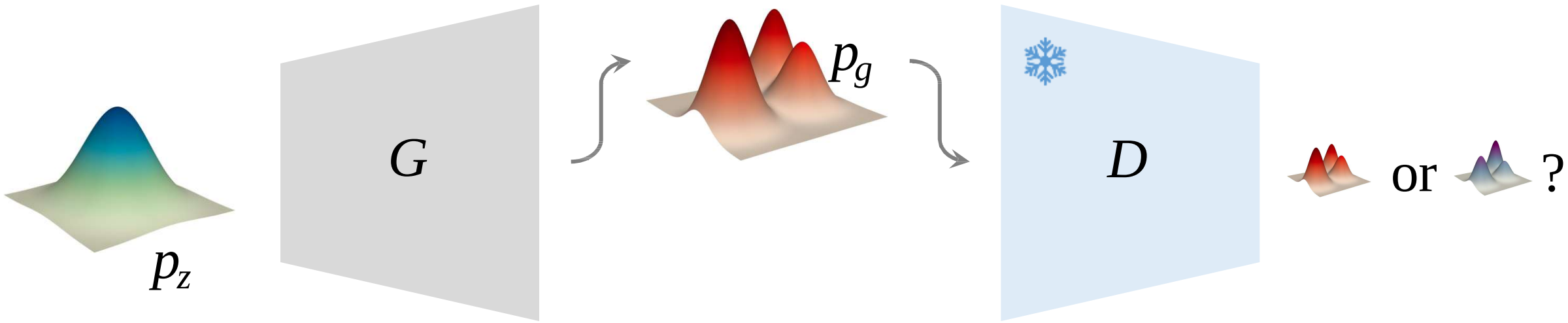


Adversarial Objective: G-step

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

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

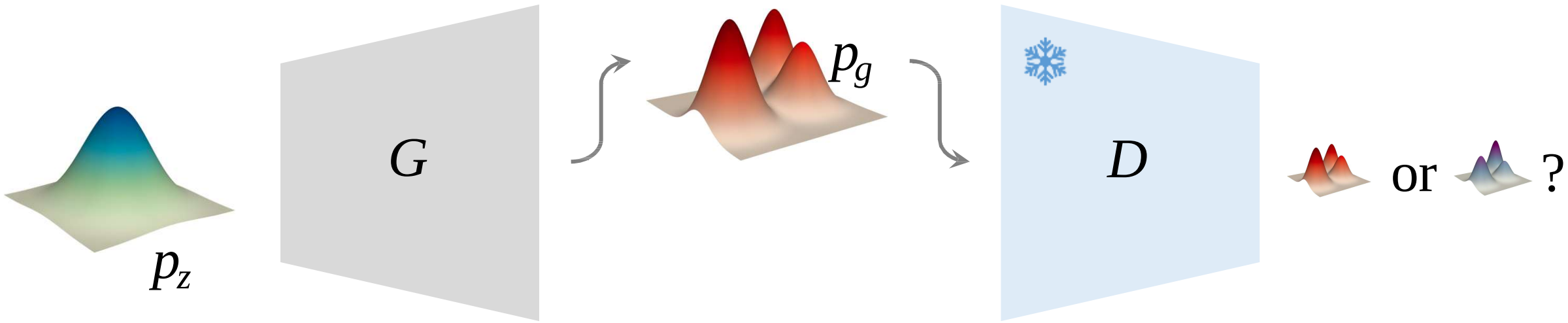
a "flip" trick:

max

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

push to 1

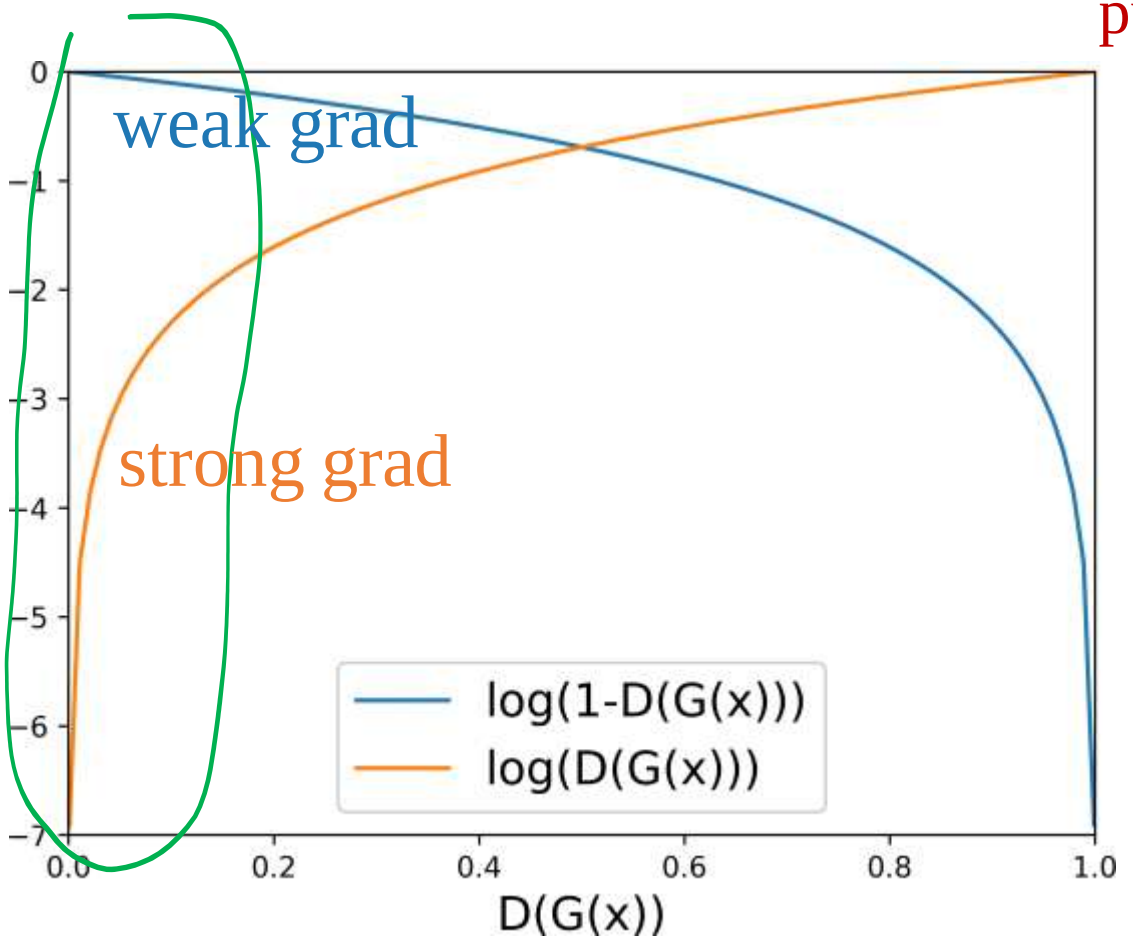
- *G*-step: fix *D*, optimize *G*
- generate fake data such that *D* classifies it as "real"
- *G* to "confuse" *D*



Adversarial Objective: G-step

a "flip" trick: max $\min_G \mathcal{L}(G) = \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$

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:

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

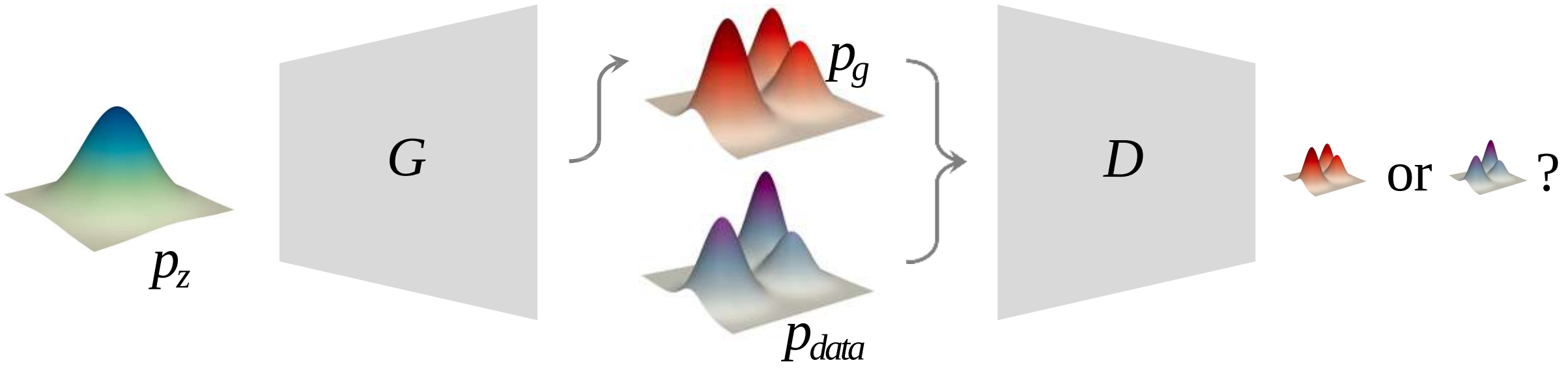
  end for

  - 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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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:

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

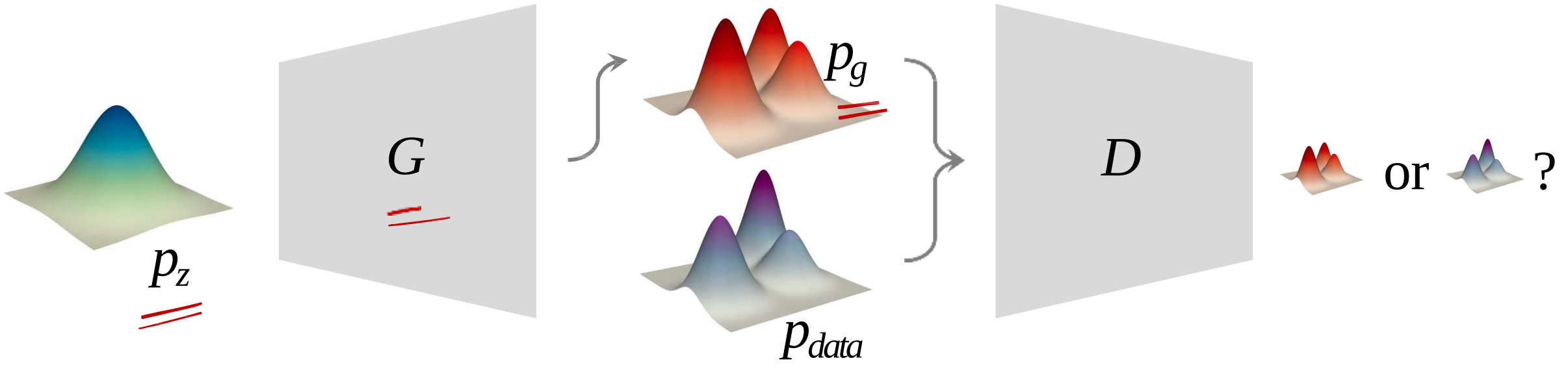
  end for

  - 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:

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

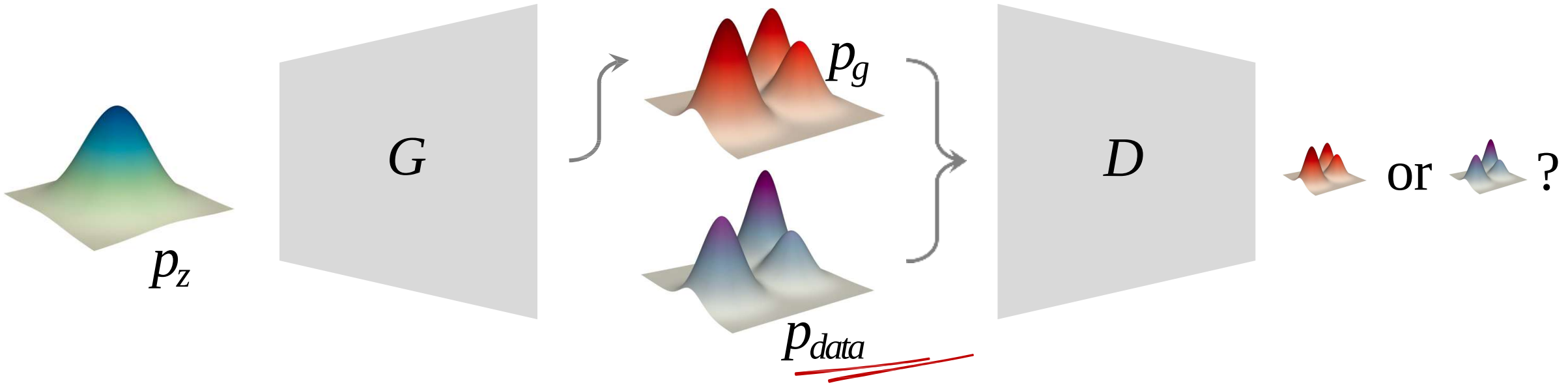
  end for

  - 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:

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

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:

D-step

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

end for

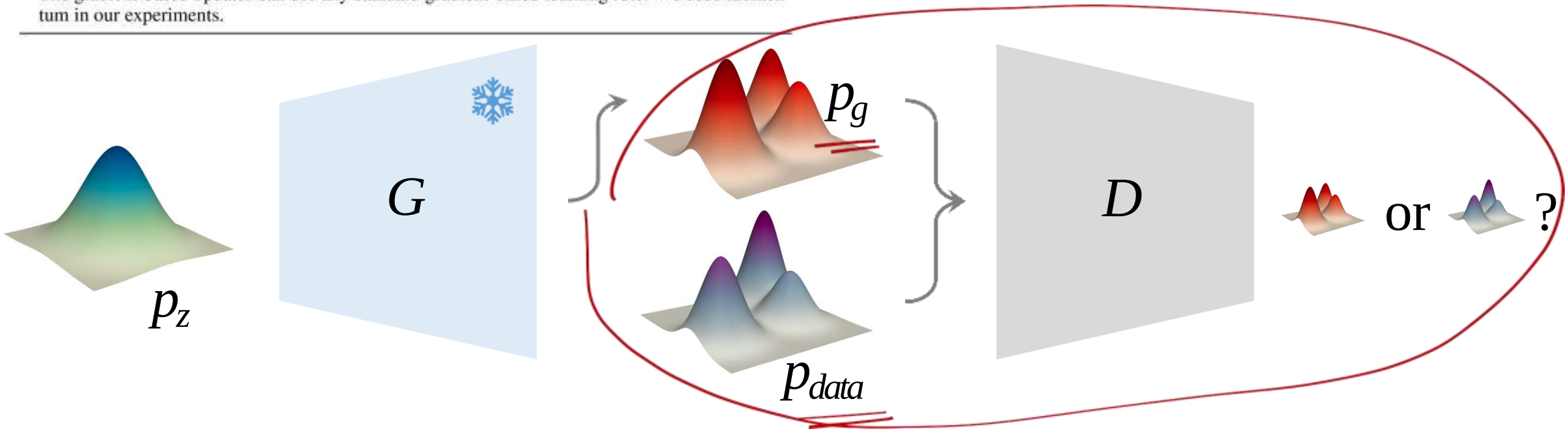
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

end for

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

gradient **ascend**
(maximize)



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:

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

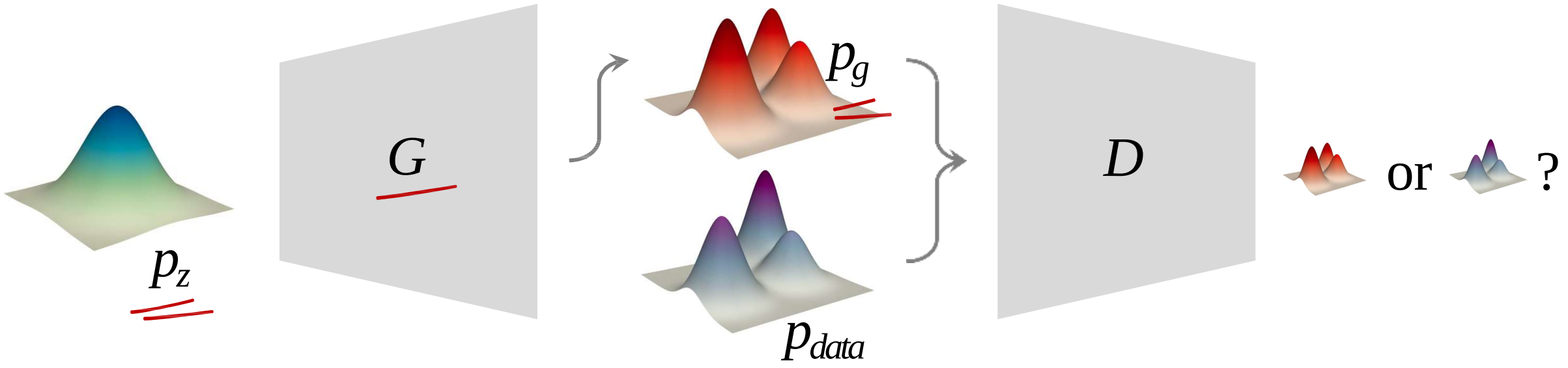
  end for

  - 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

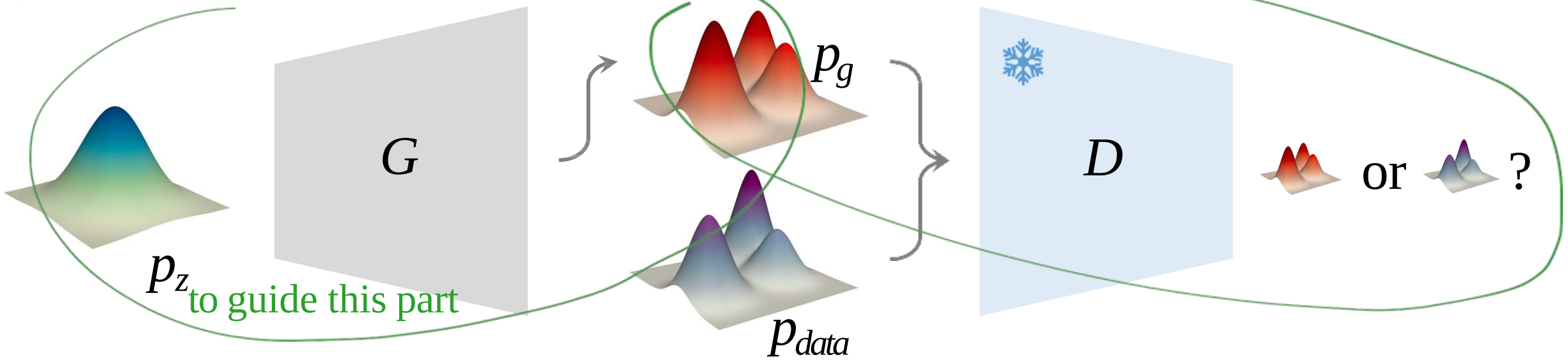
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

gradient **descend** (minimize)

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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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

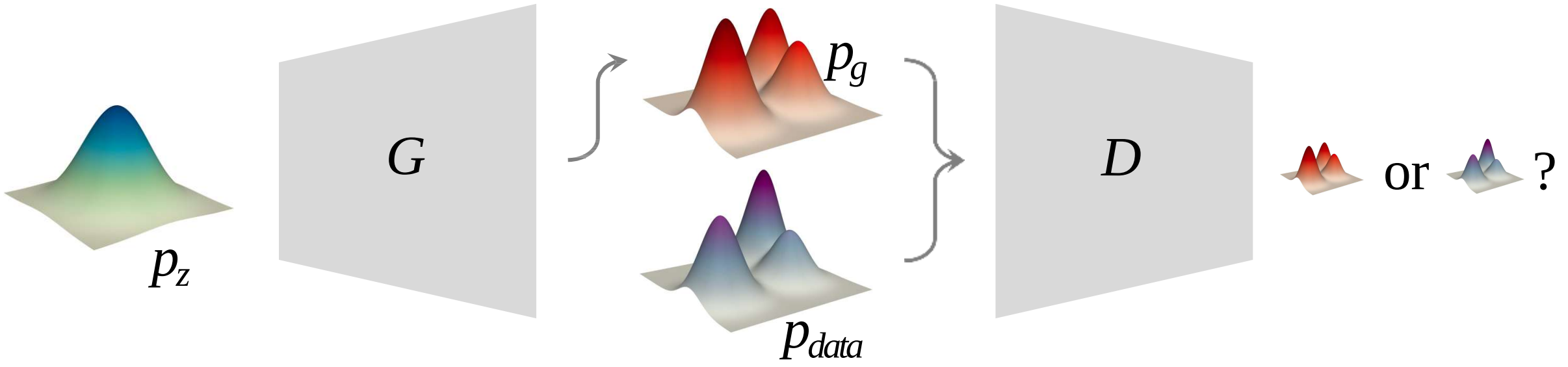
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

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

end for

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

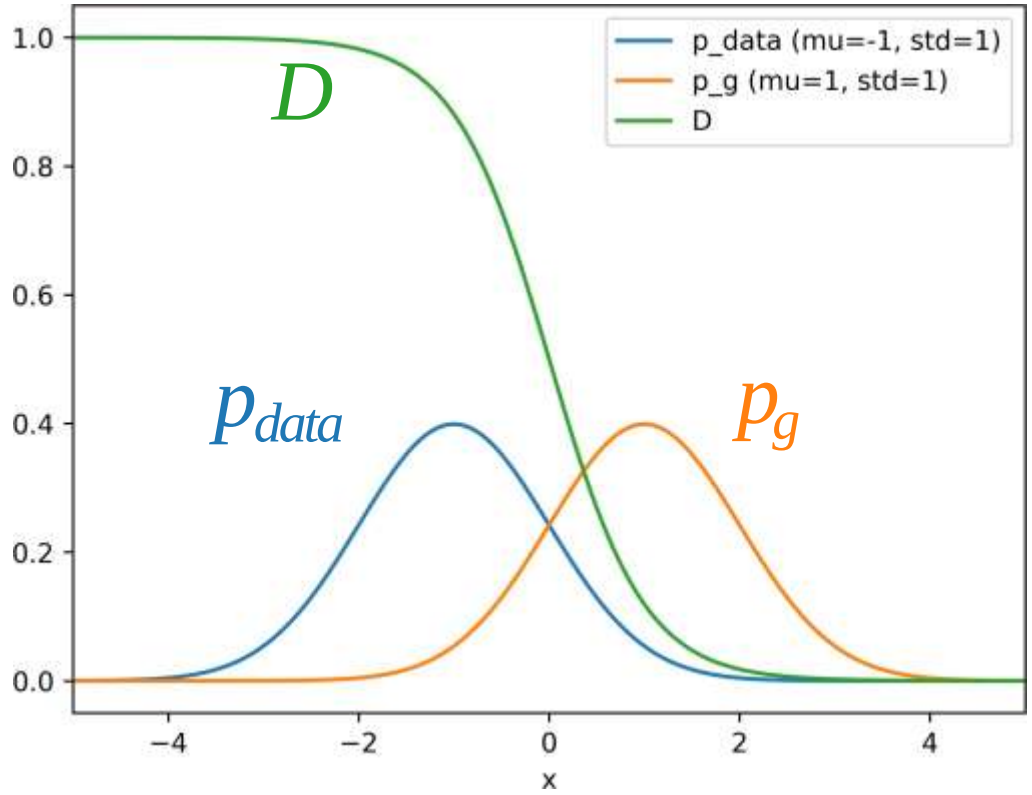
iterating
min-max



Theoretical Results

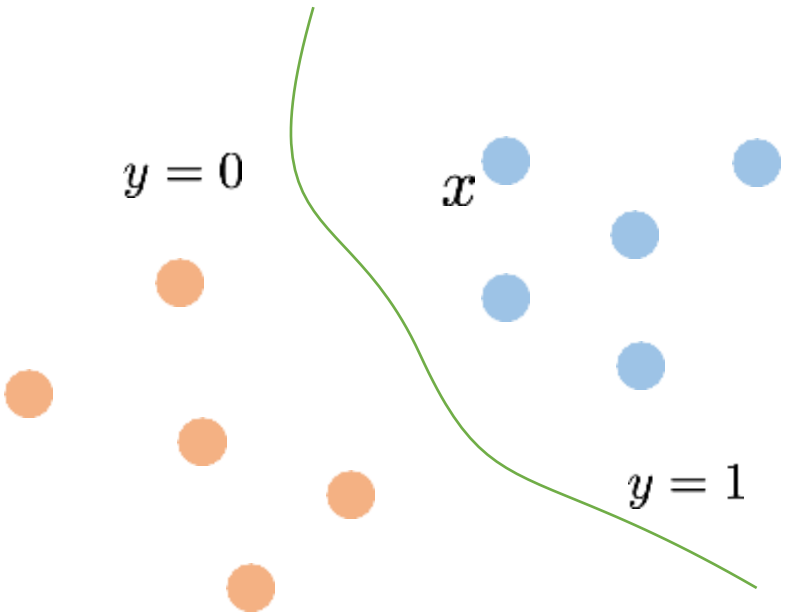
1. For any given *G*, the optimal *D* is:

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

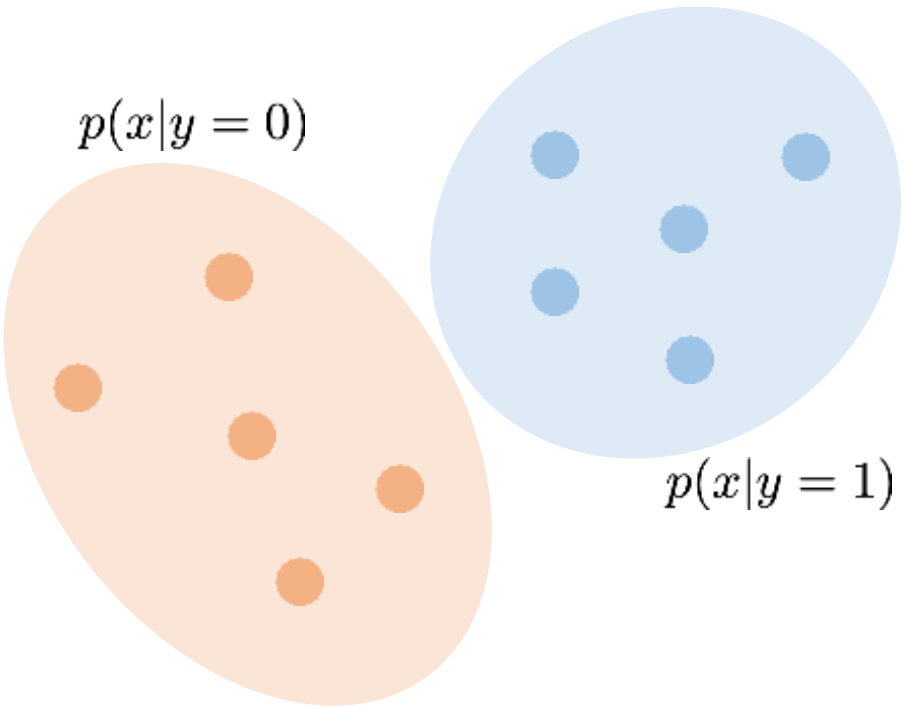


Recap : Discriminative vs. Generative

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

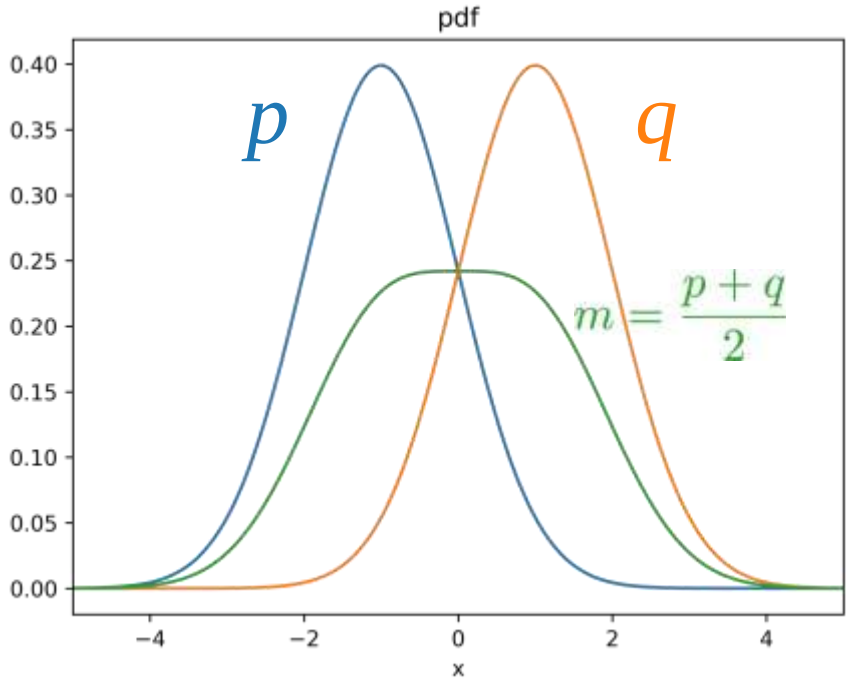
See proof in “Generative Adversarial Nets”, Goodfellow, et al., 2014

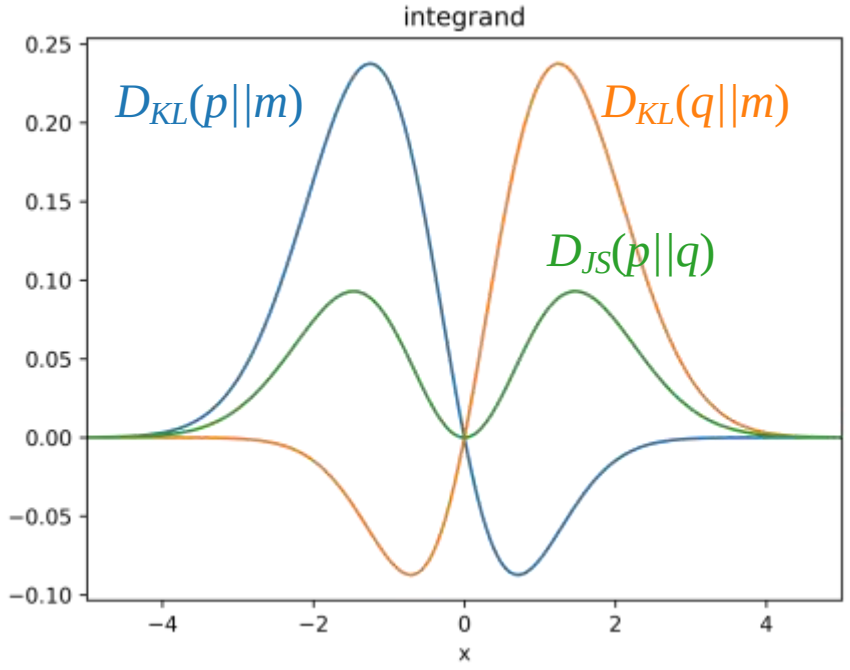
Background: Jensen–Shannon divergence

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

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

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





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:

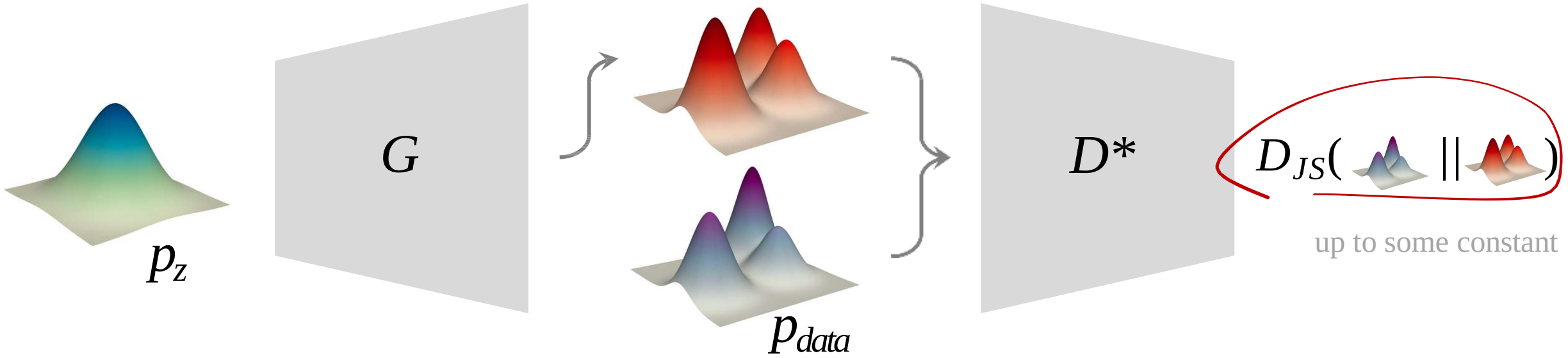
- D_{JS} is symmetric; D_{KL} is not
- D_{JS} is bounded: [0, log2]; D_{KL} is unbounded: [0, inf)
- D_{JS} is more stable

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.

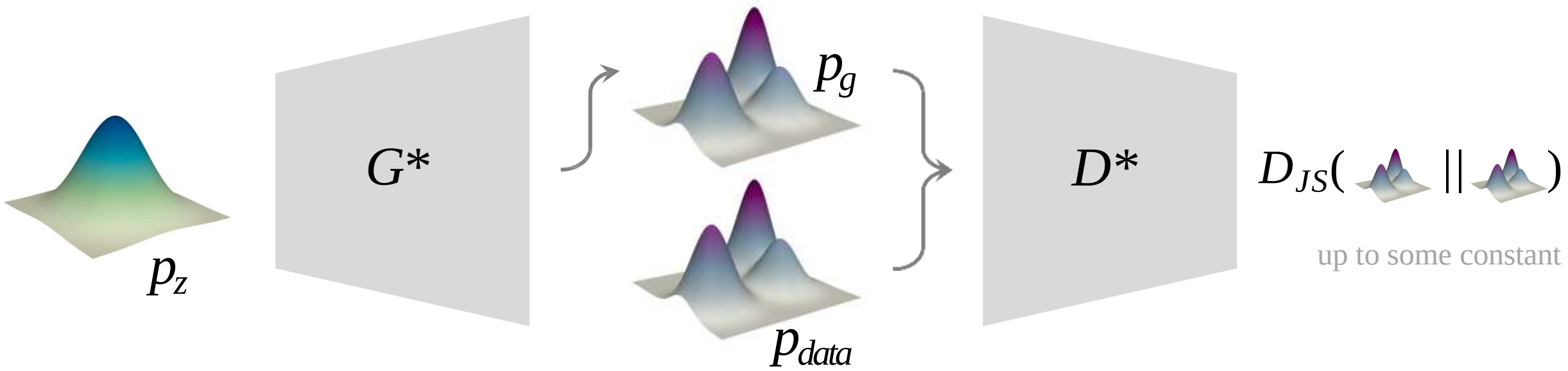


See proof in "Generative Adversarial Nets", Goodfellow, et al., 2014

Theoretical Results

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

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



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 $p_g = p_{data}$

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

Theoretical Results: Summary

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

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

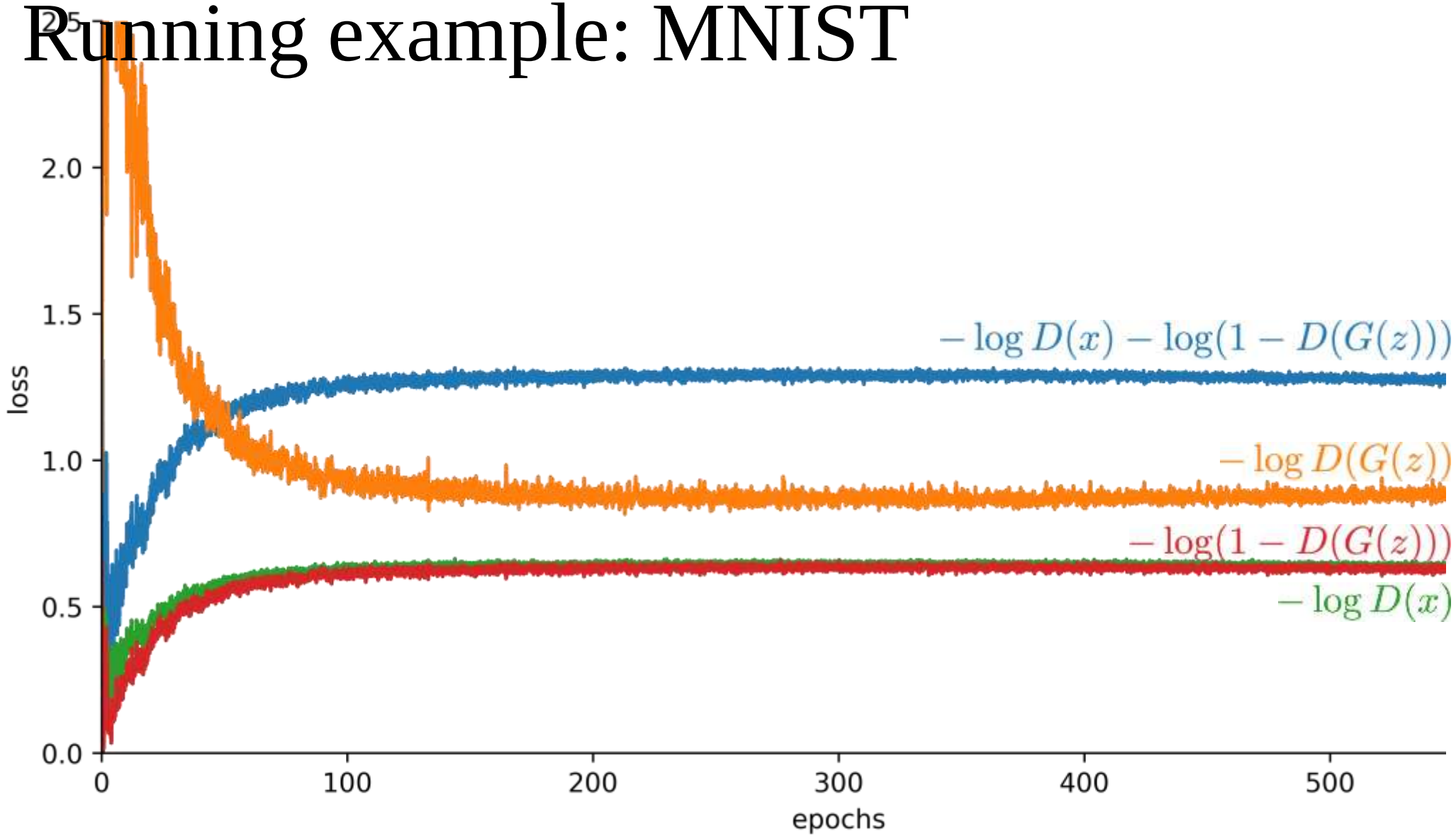
2. With optimal D_G, GAN optimizes for Jensen–Shannon divergence:

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

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

$$\begin{aligned} L(G, D^*) &= \int_x \Big(p_r(x) \log(D^*(x)) + p_g(x) \log(1 - D^*(x)) \Big) dx \\ &= \log \frac{1}{2} \int_x p_r(x) dx + \log \frac{1}{2} \int_x p_g(x) dx \qquad D^*(x) = 1/2 \\ &= -2 \log 2 \end{aligned}$$

Running example: MNIST

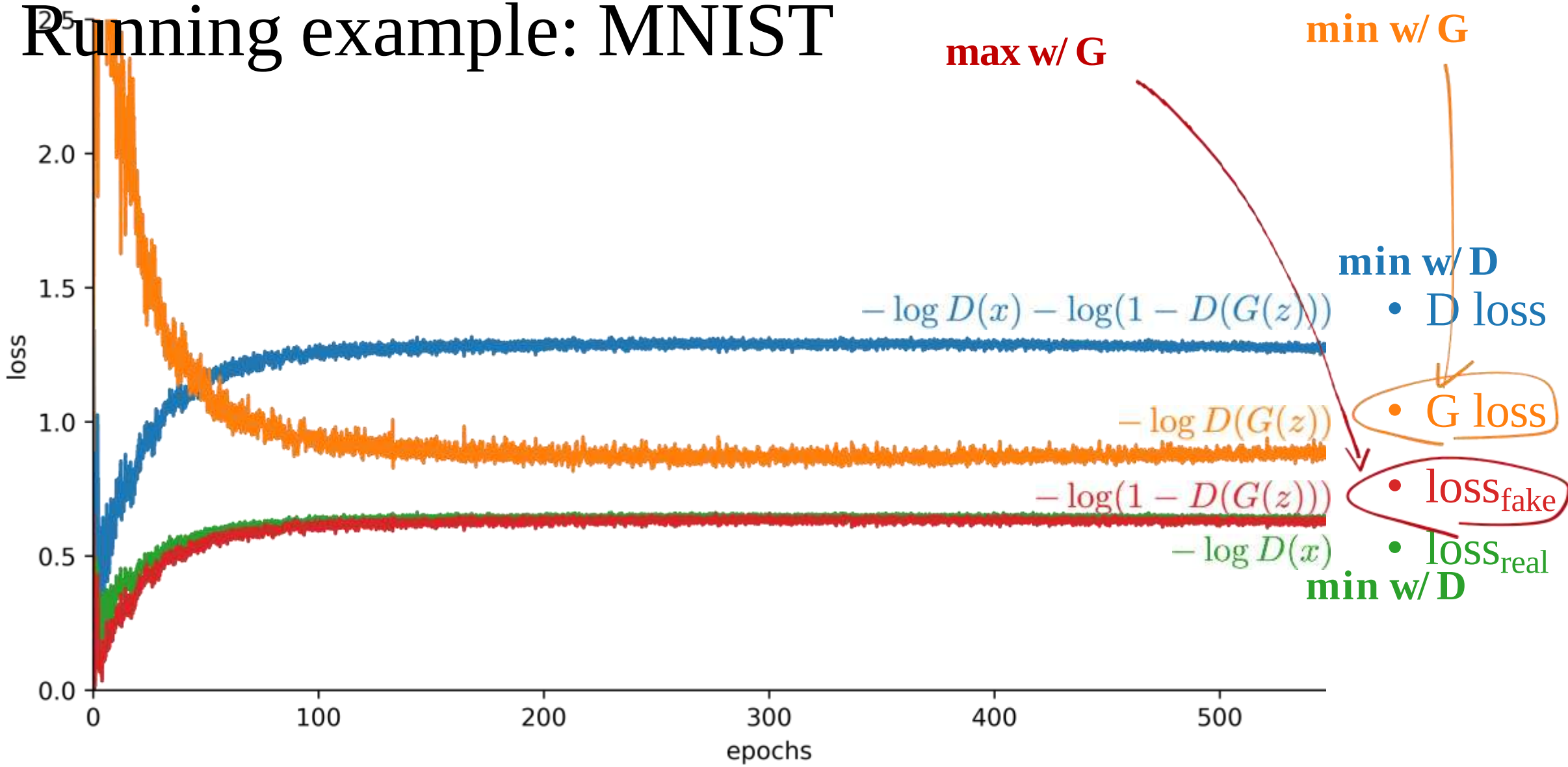


- D loss
- G loss
- $\text{loss}_{\text{fake}}$
- $\text{loss}_{\text{real}}$

*All objectives are negative of their original form

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

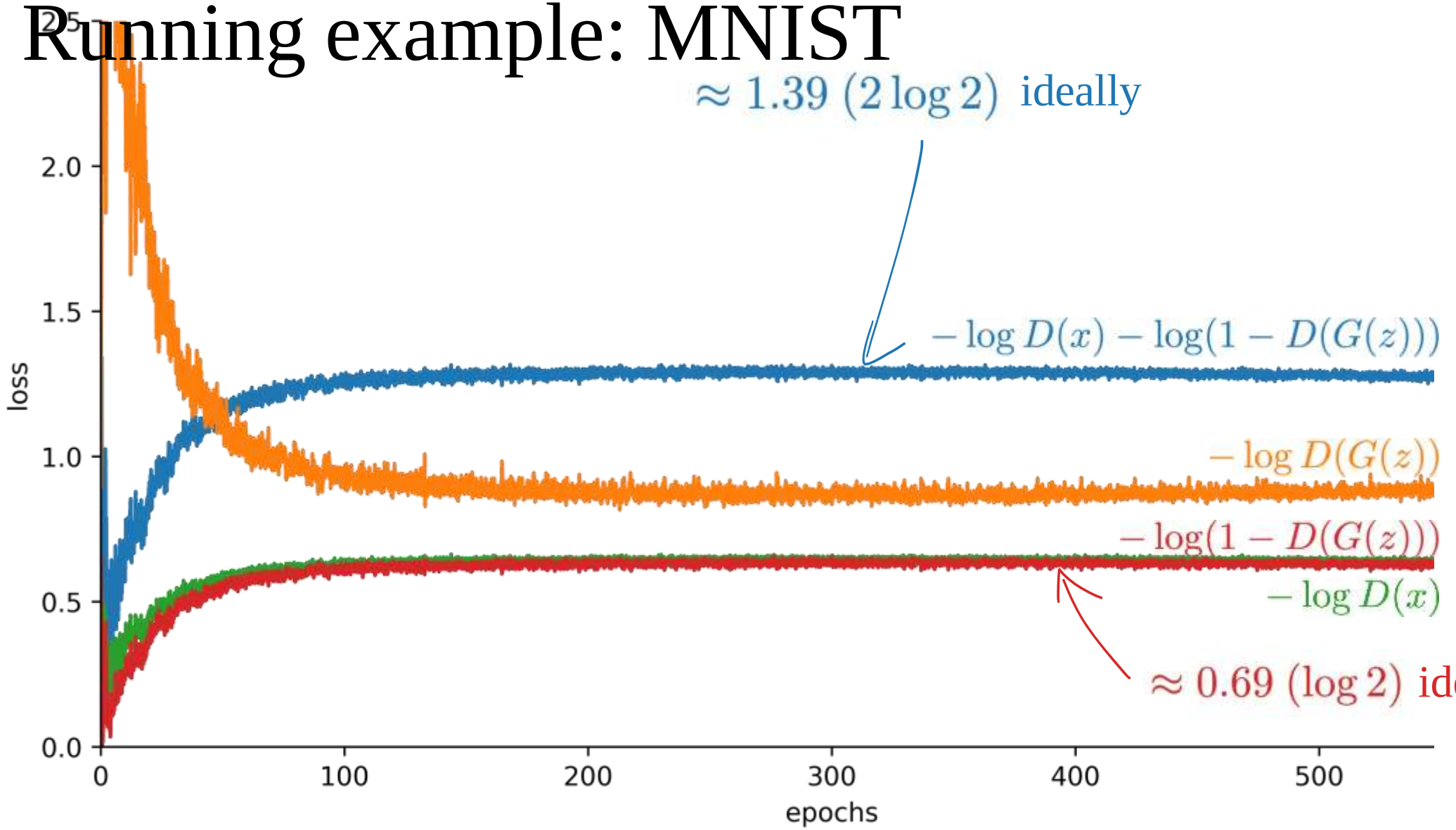
Running example: MNIST



*All objectives are negative of their original form

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

Running example: MNIST

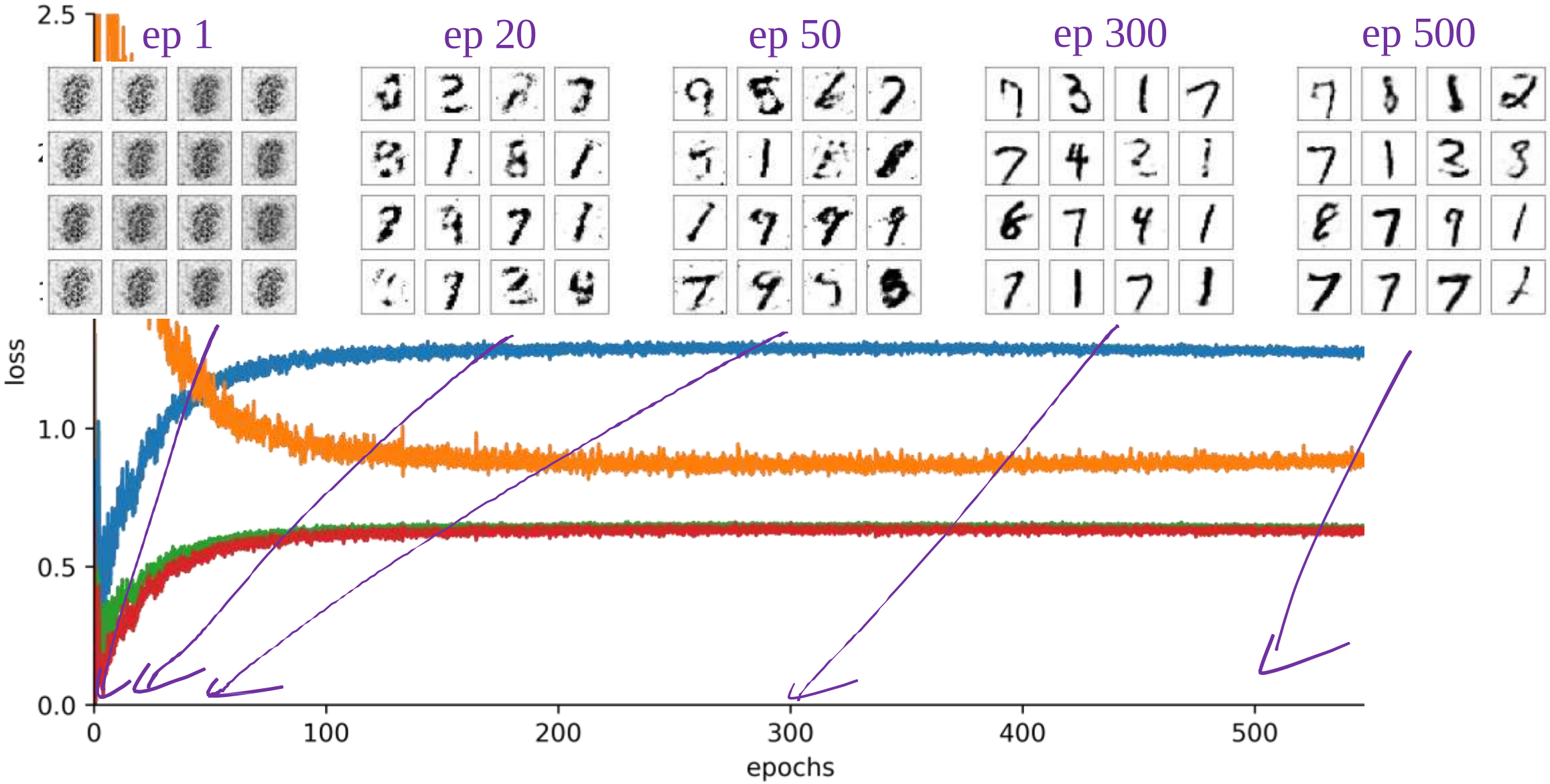


- D loss
- G loss
- $\text{loss}_{\text{fake}}$
- $\text{loss}_{\text{real}}$

*All objectives are negative of their original form

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

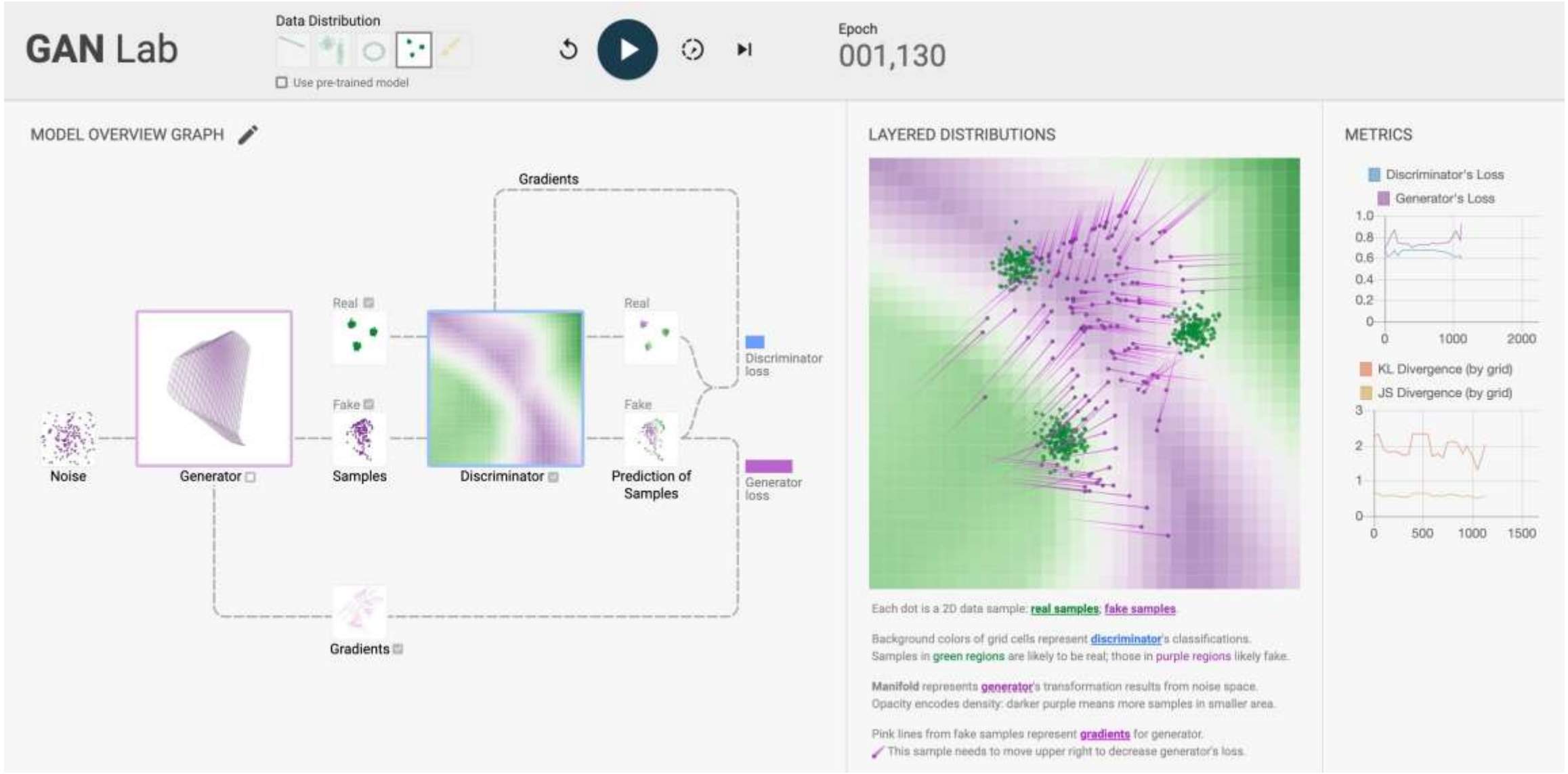
Running example: MNIST



*All objectives are negative of their original form

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

Running example: GAN Lab https://poloclub.github.io/ganlab/



Adversary as a Loss Function

Adversary as a Loss Function

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

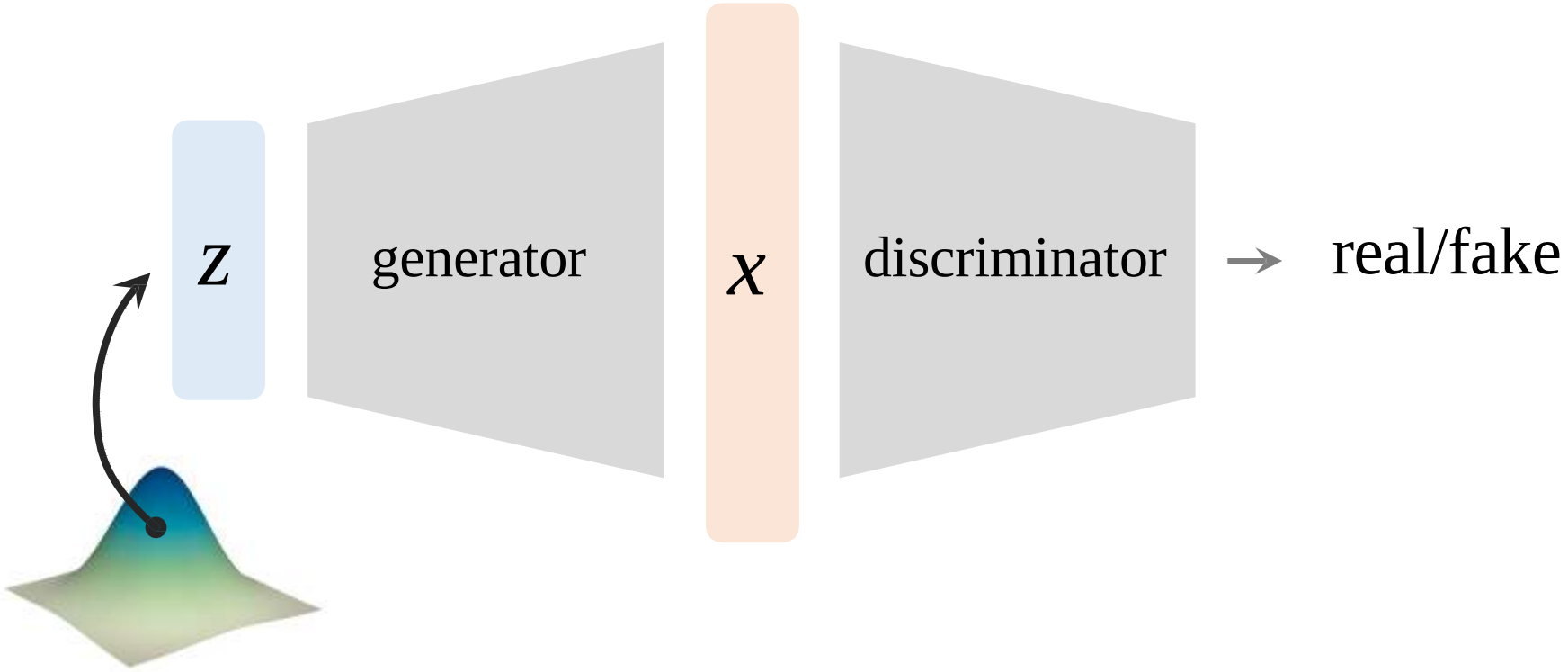
- Input to networks is **not** necessarily random/noise

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

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

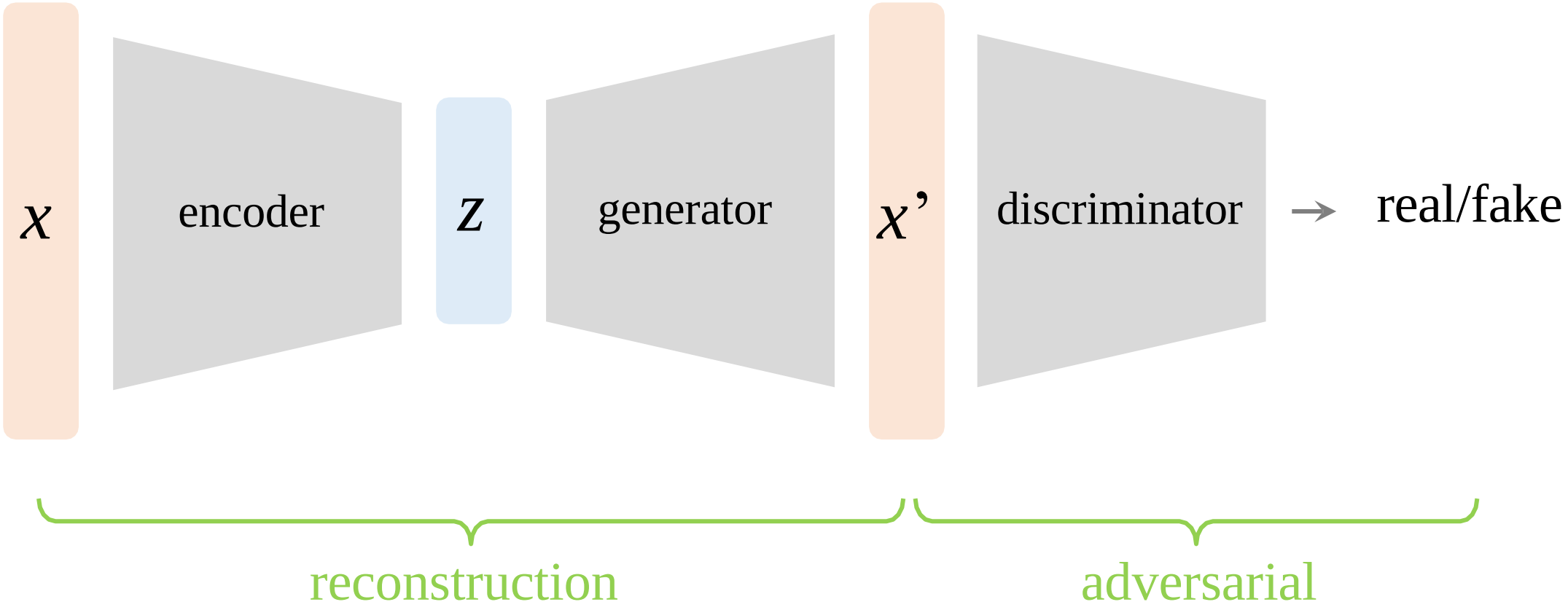
Adversary as a Loss Function

- GAN: input is random



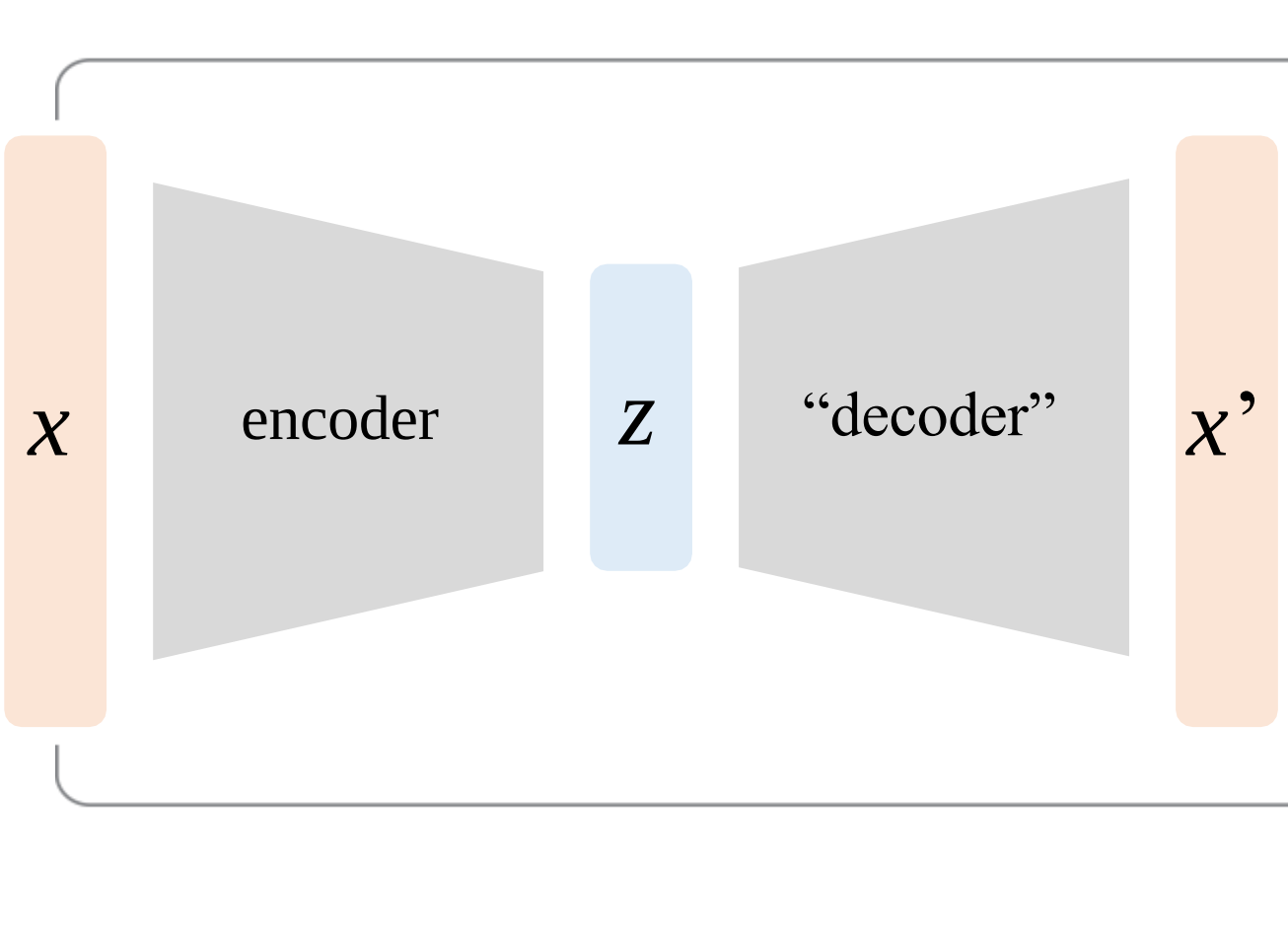
Adversary as a Loss Function

- Input can be from another source



Adversary as a Loss Function

- Input can be from another source



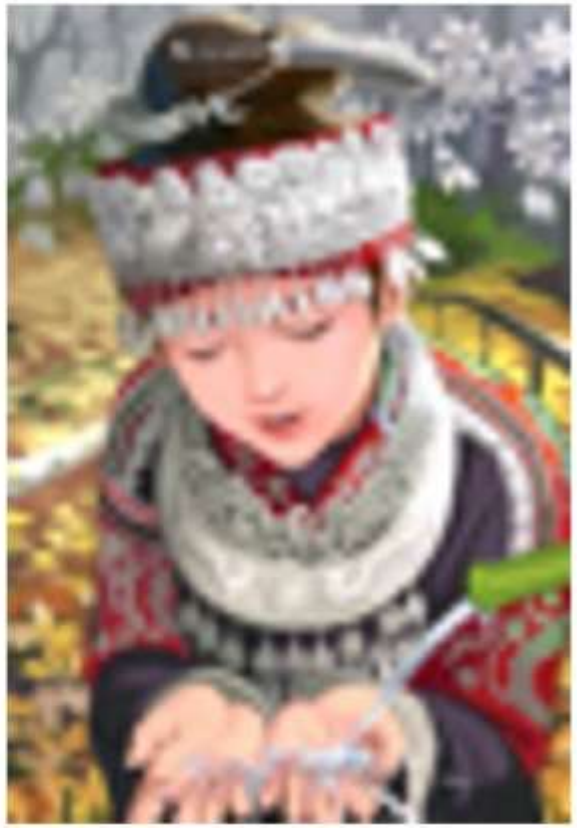
- parameterized loss function
- trained alternately

Example: Super-Resolution GAN

- better PSNR
- worse PSNR, but better visual quality

original

bicubic
(21.59dB/0.6423)

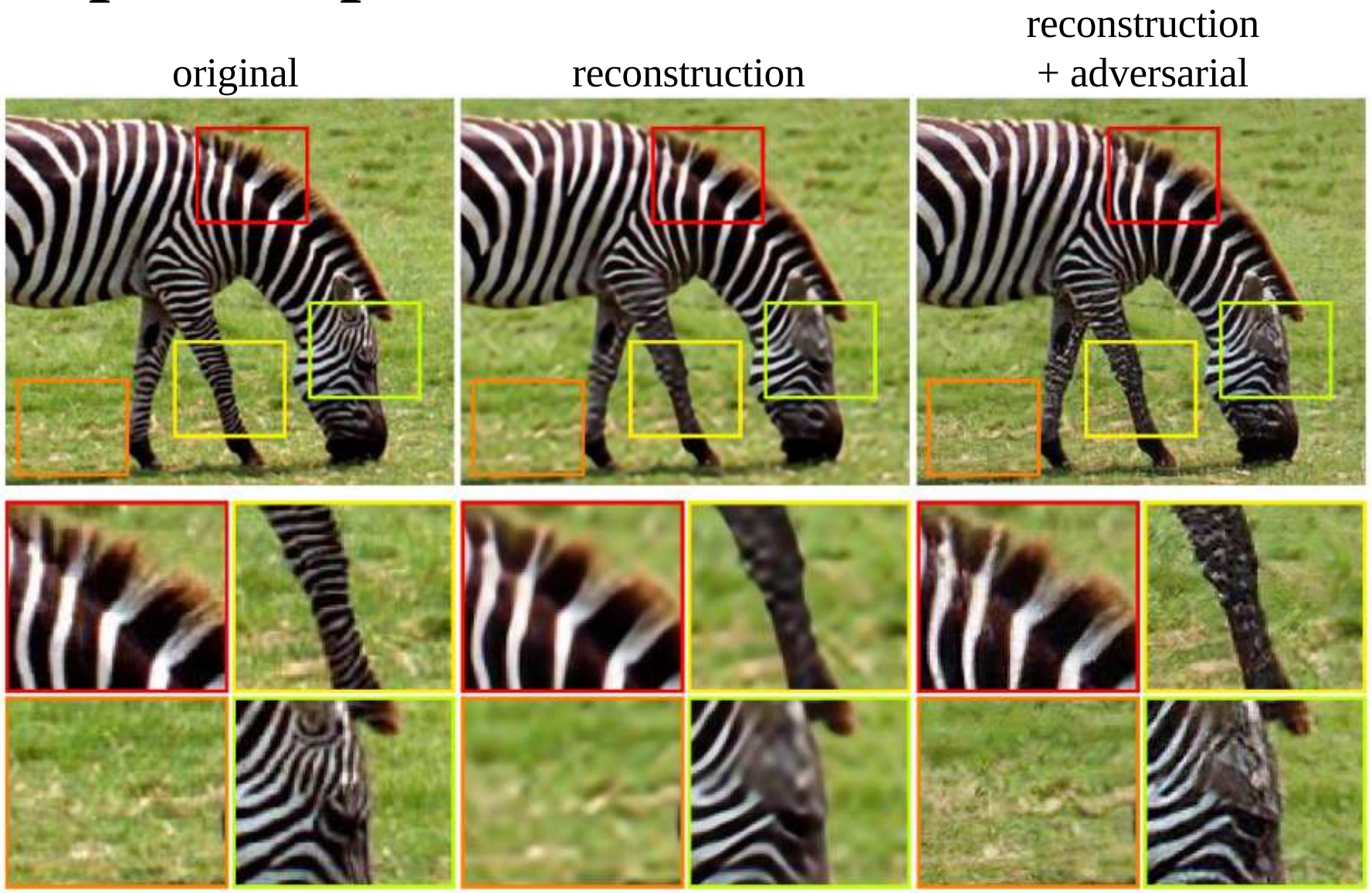


SRResNet
(23.44dB/0.7777)

SRGAN
(20.34dB/0.6562)

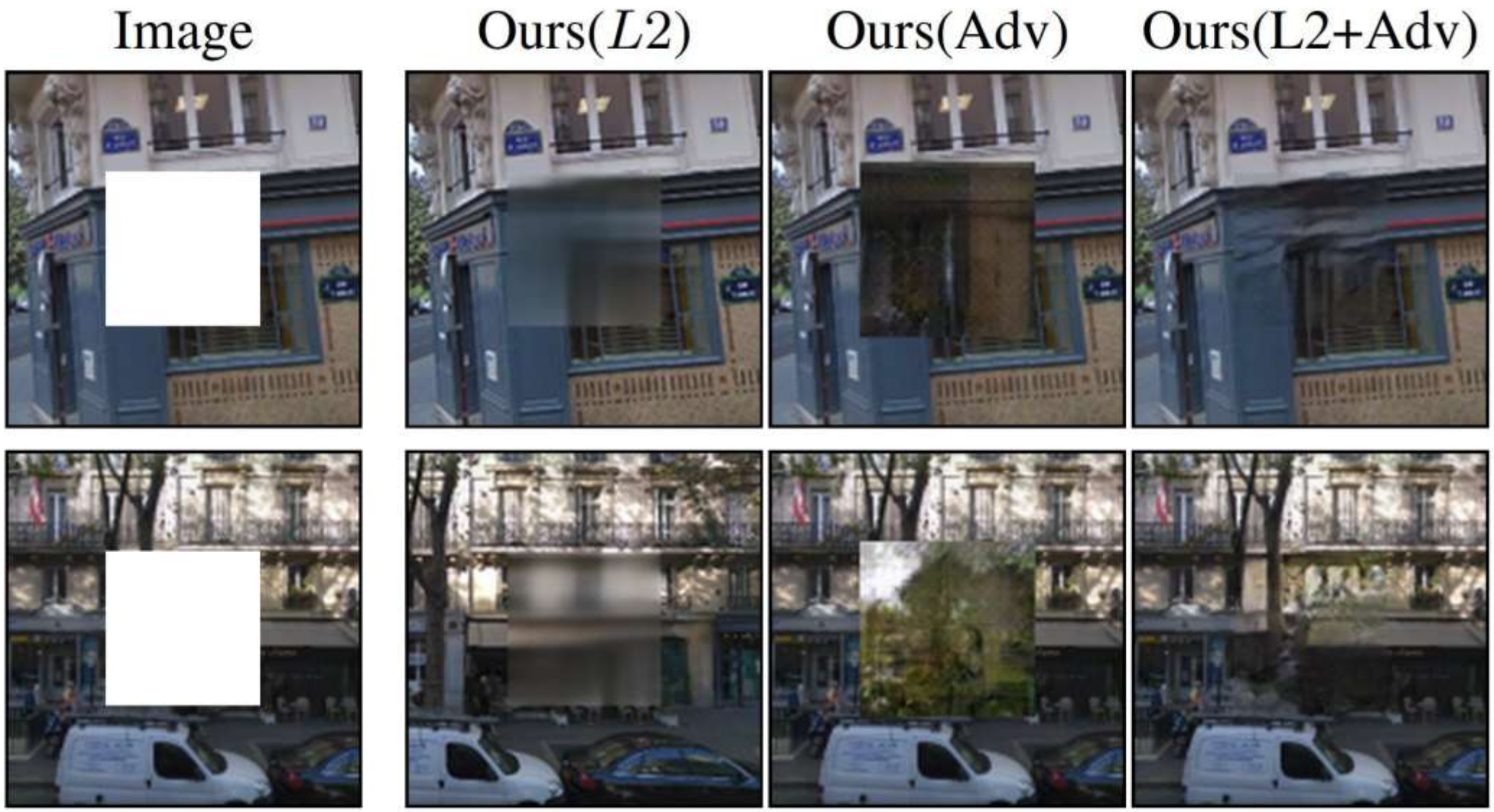
Ledig, et al., "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", CVPR

Example: Super-Resolution GAN



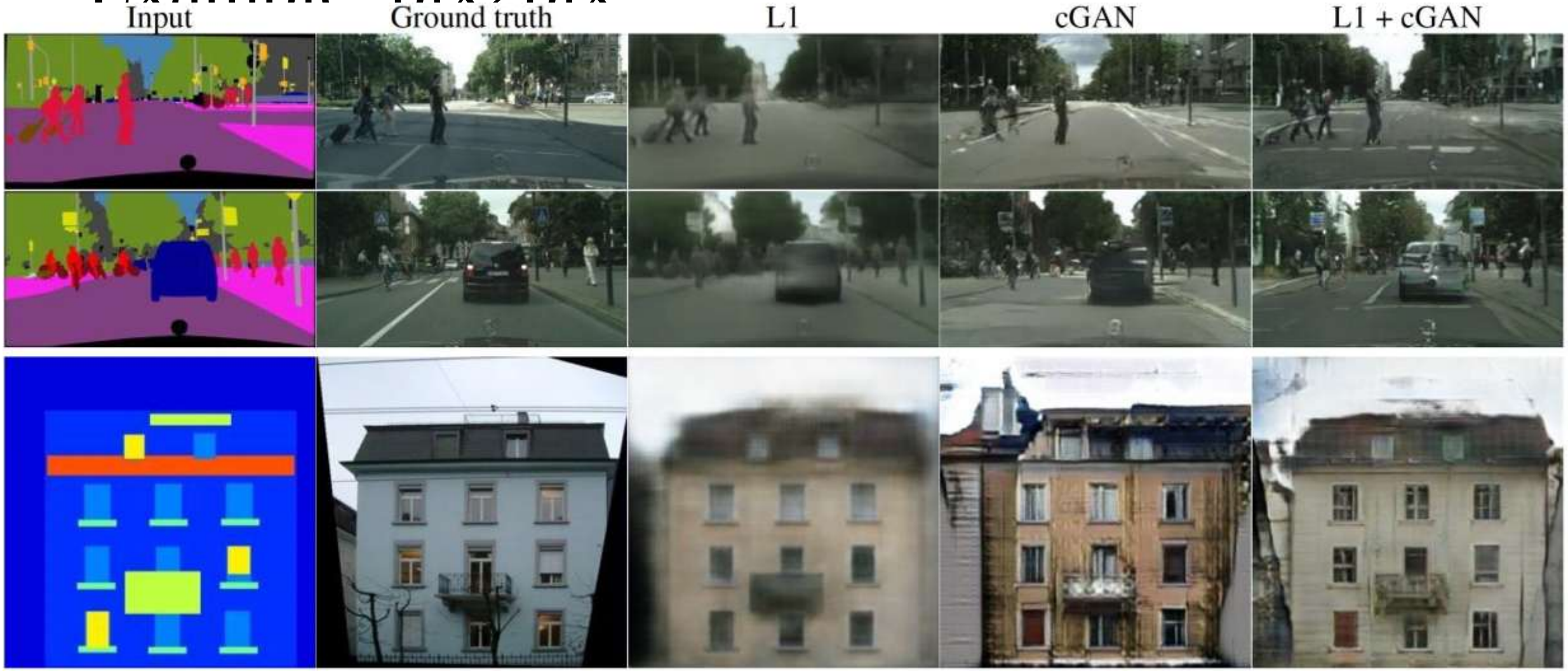
Ledig, et al., “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, CVPR

E



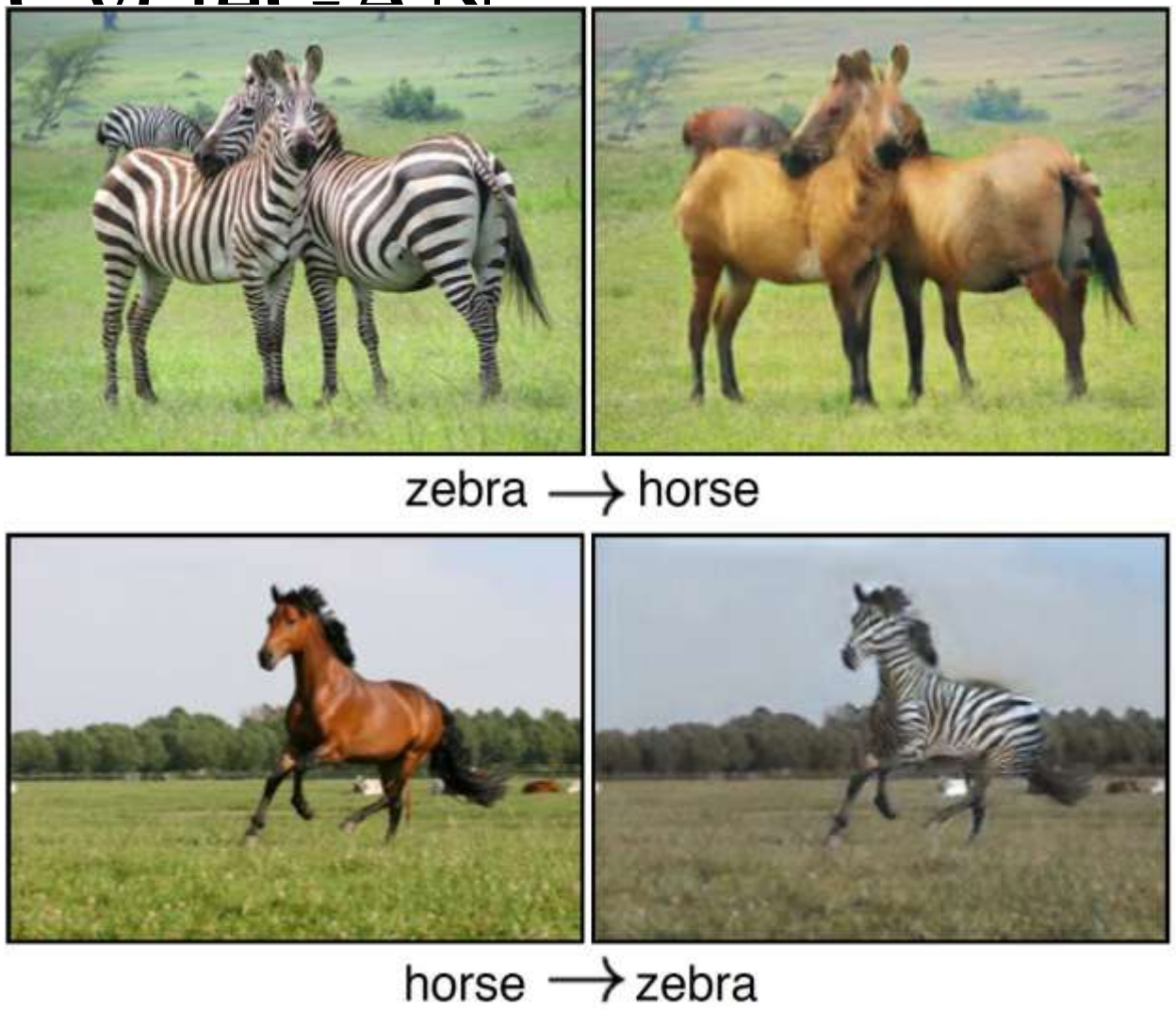
Pathak, et al., “Context Encoders: Feature Learning by Inpainting”, CVPR

Example: pix2pix

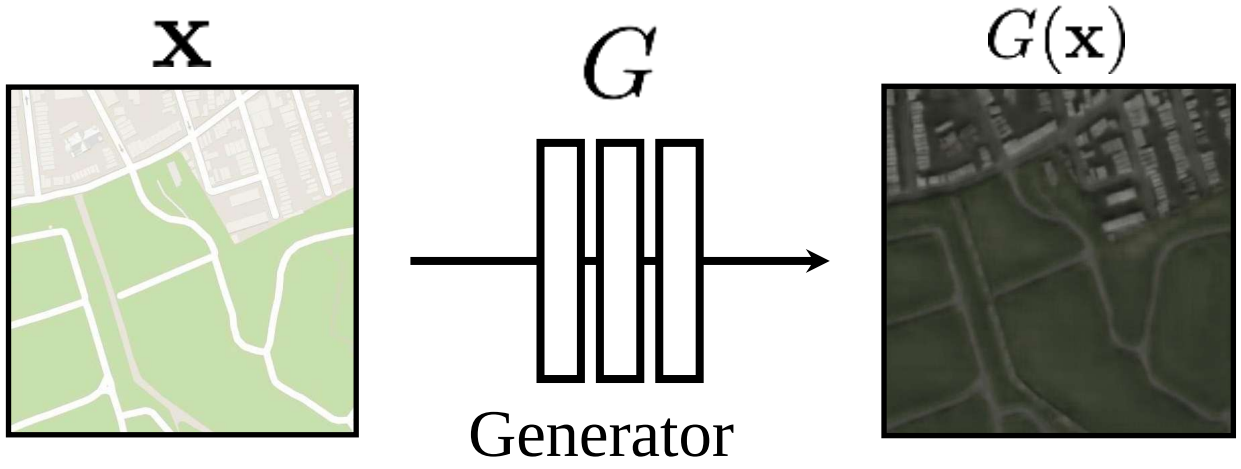


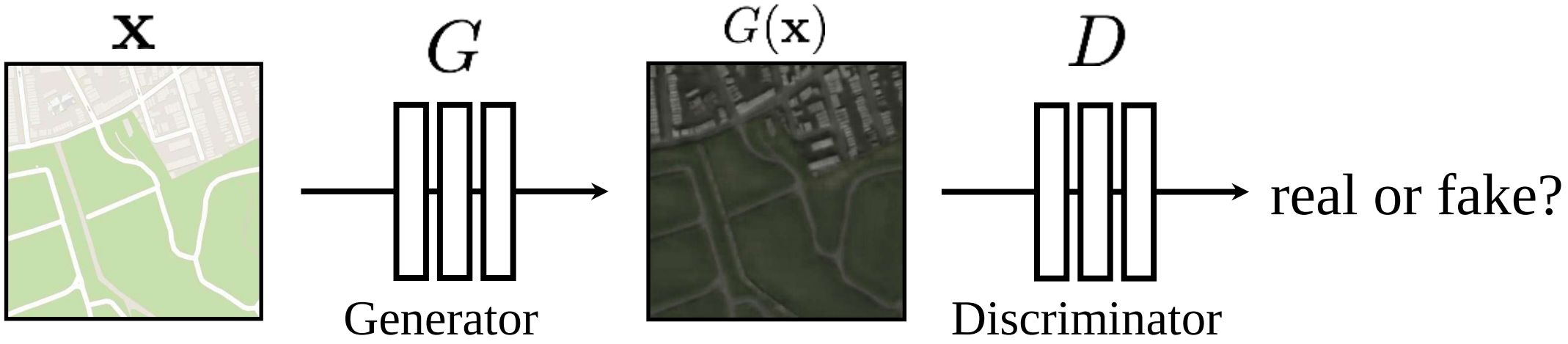
Isola, et al., “Image-to-Image Translation with Conditional Adversarial Networks”, CVPR

Example: CycleGAN



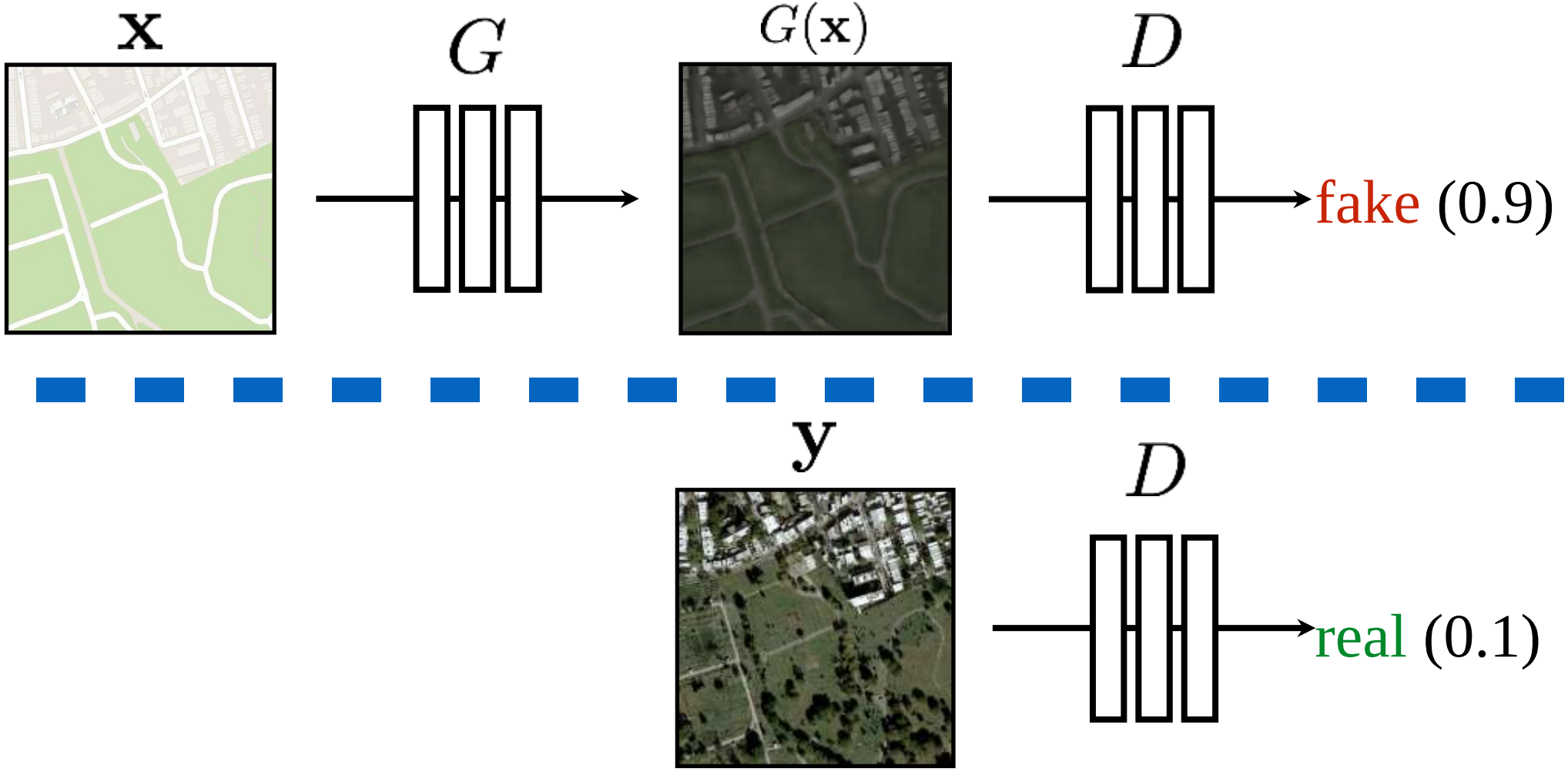
Zhu, et al., “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”, ICCV


$\mathbf{x}$
G
$G(\mathbf{x})$
Generator



G tries to synthesize fake images that fool D

D tries to identify the fakes



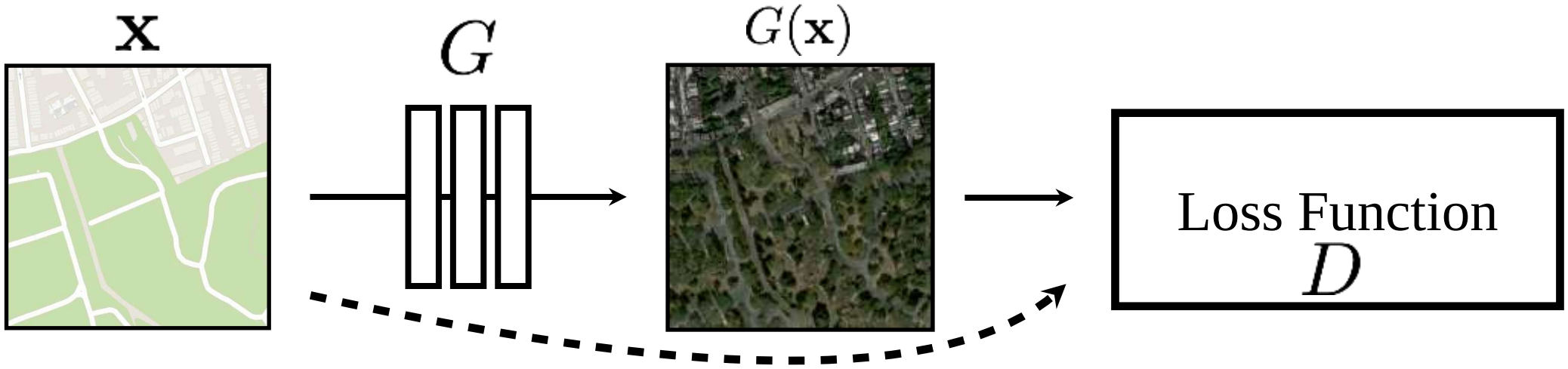
$$\arg\max_{D} \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\, \log D(G(\mathbf{x})) \; + \; \log(1 - D(\mathbf{y}))\,]$$

G tries to synthesize fake images that *fool* D:

$$\arg \min_G \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y}))\ ]$$

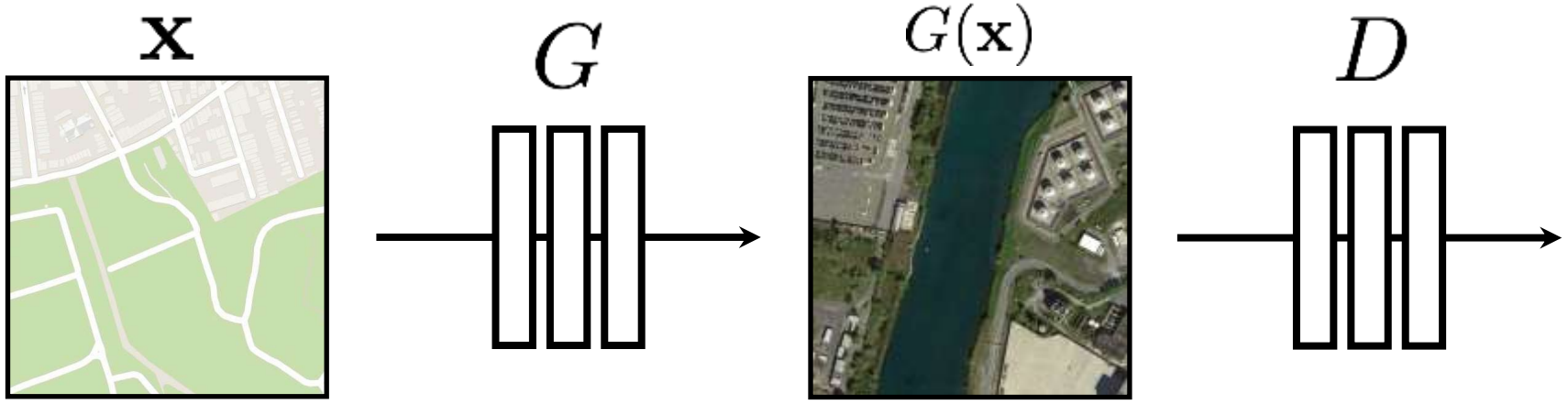
G tries to synthesize fake images that *fool* the *best* D:

$$\arg \min_G \max_D \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y}))\ ]$$

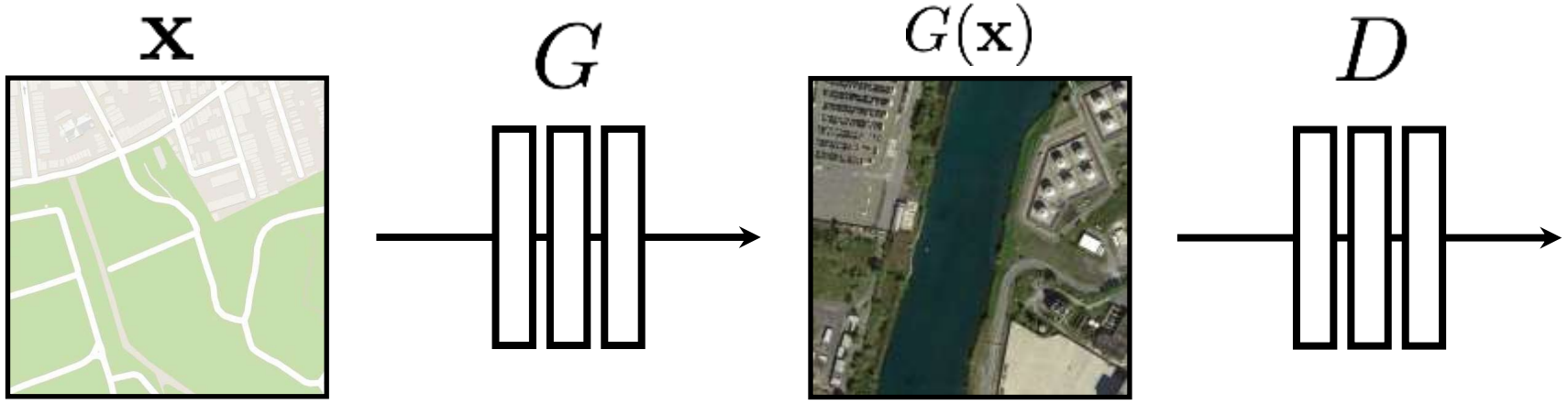


G's perspective: **D** is a loss function.

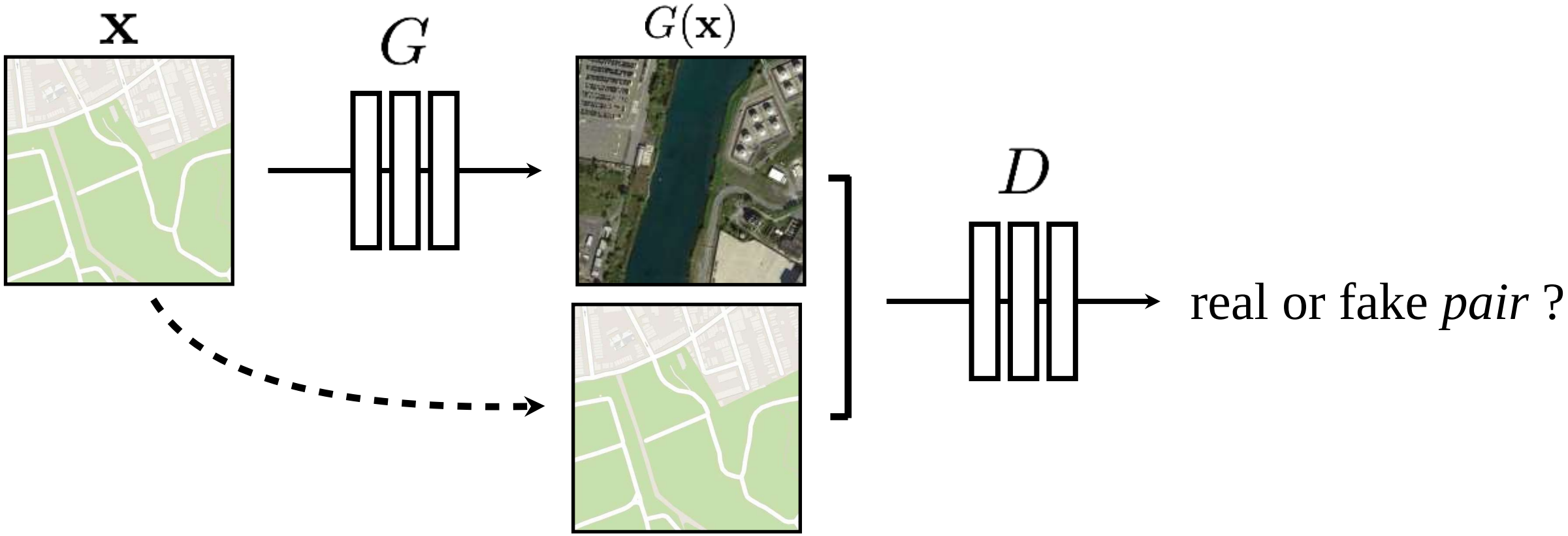
Rather than being hand-designed, it is *learned* and *highly structured*.



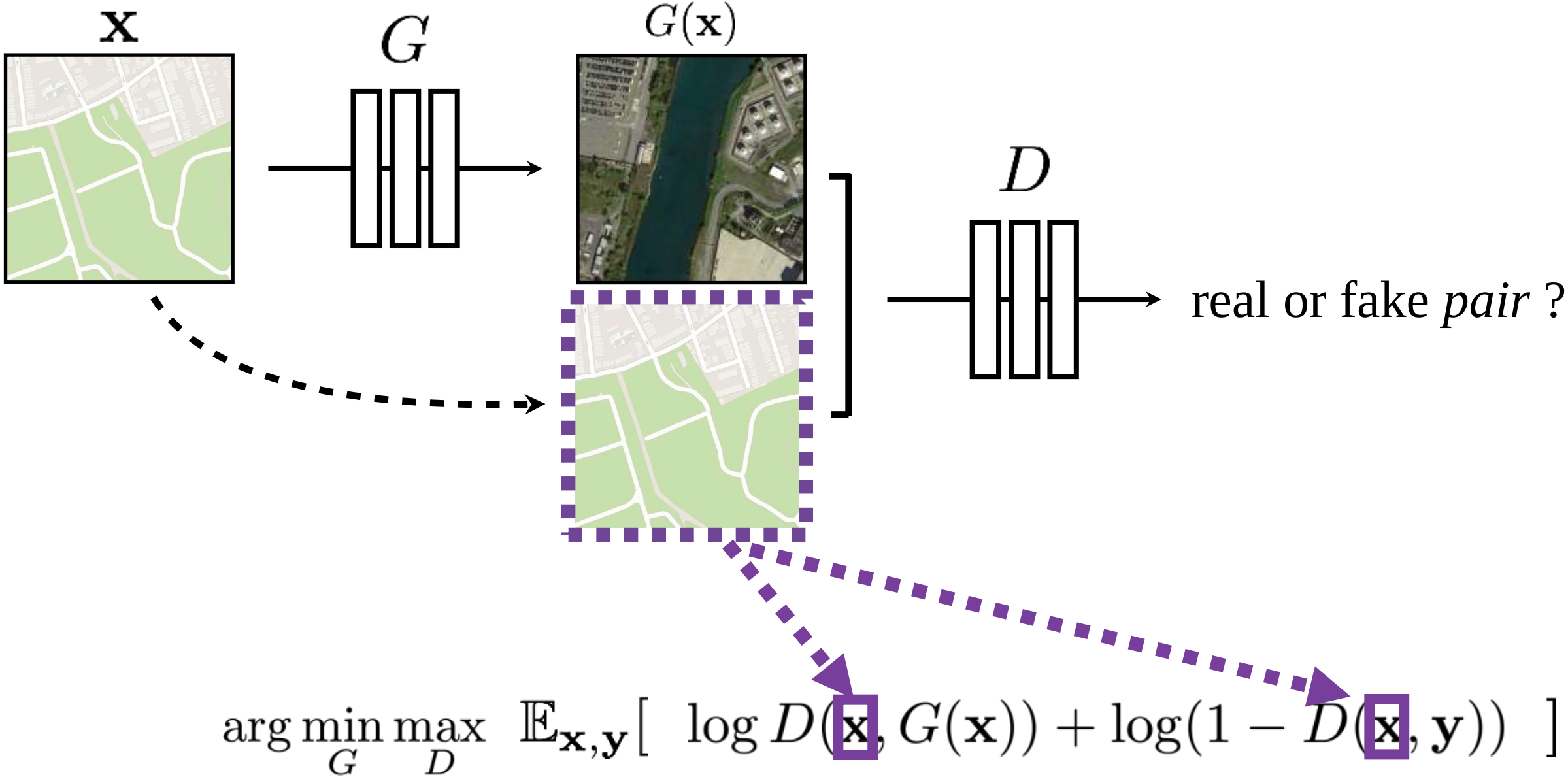
$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y})) \;\;]$$



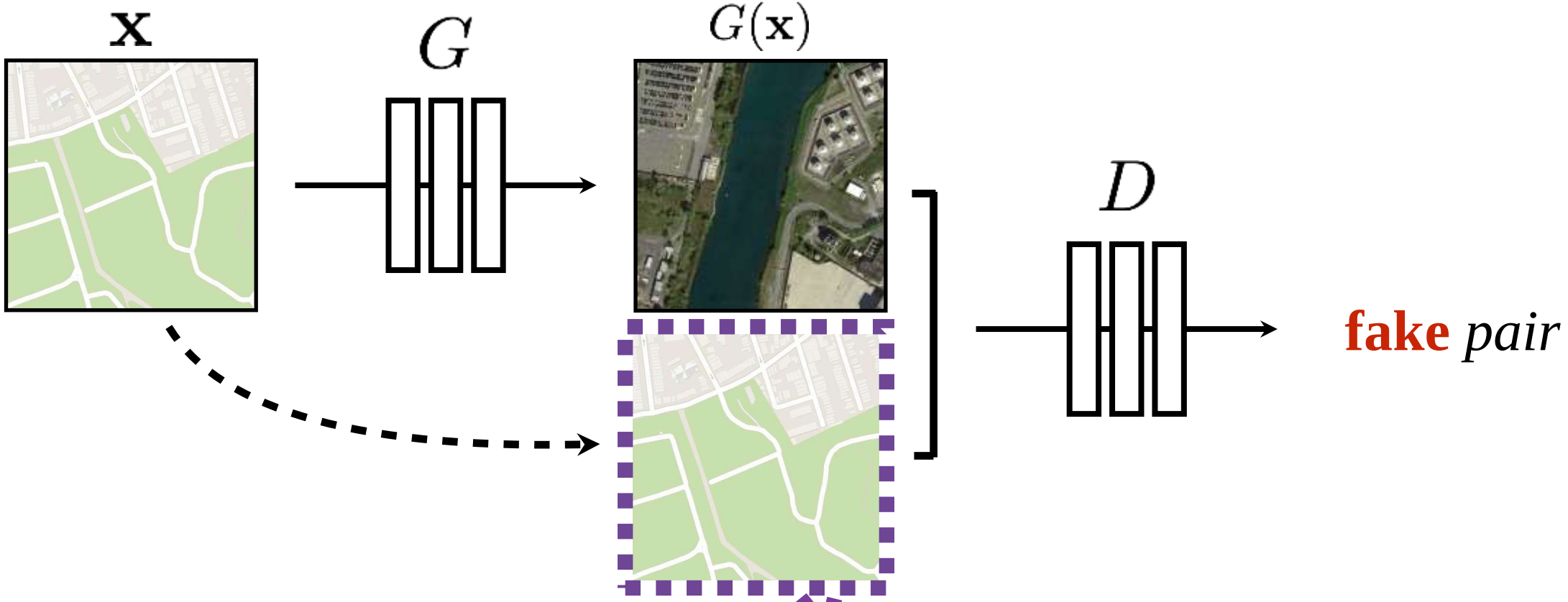
$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y})) \quad]$$



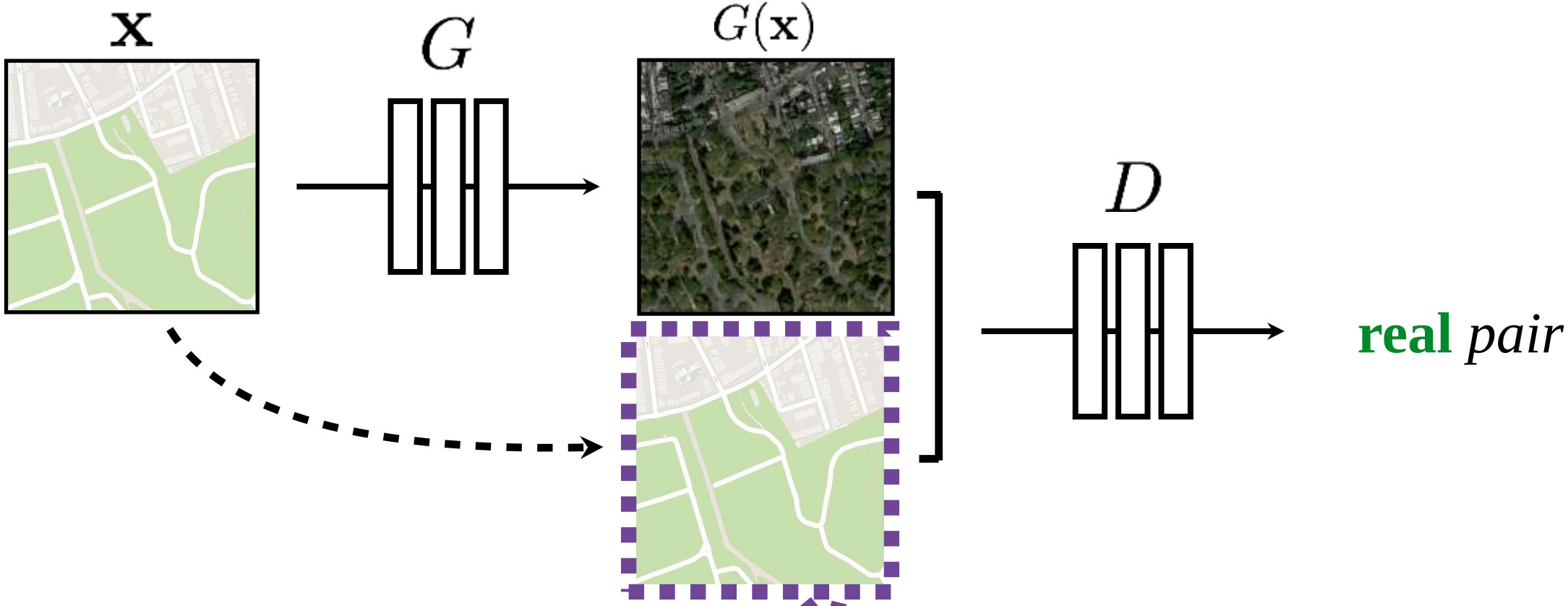
$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y}))\;]$$



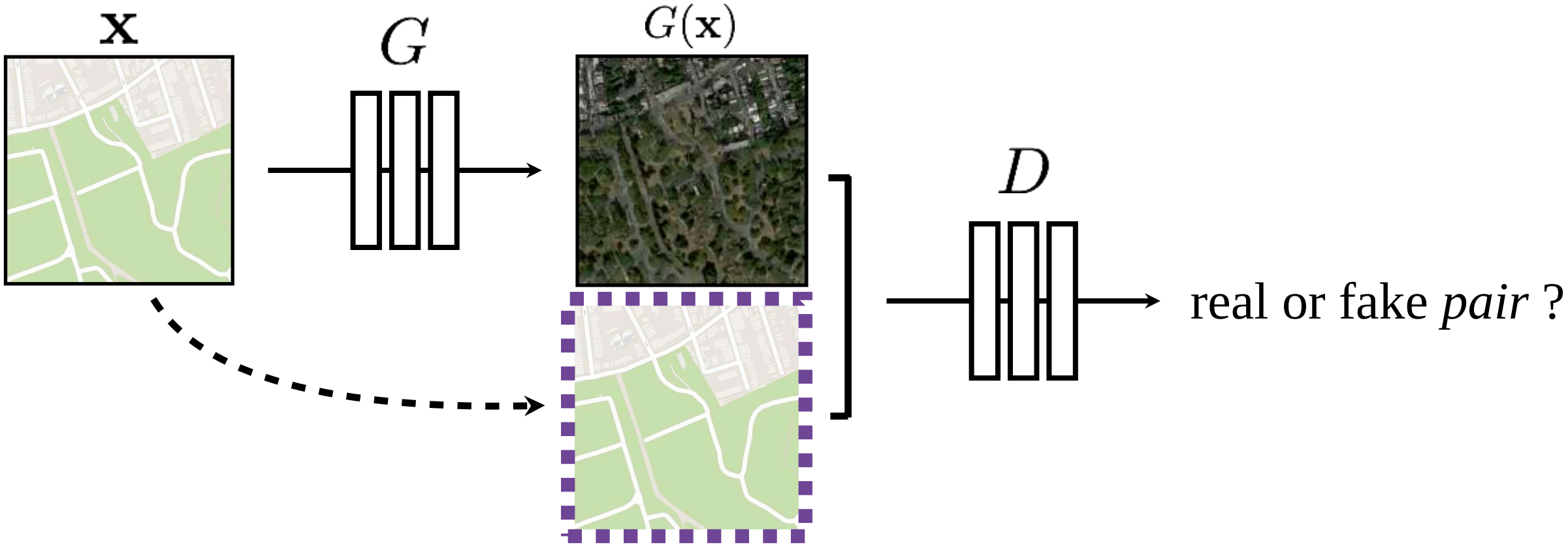
$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(\mathbf{x}, G(\mathbf{x})) + \log(1 - D(\mathbf{x}, \mathbf{y}))\;]$$



$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(\mathbf{x}, G(\mathbf{x})) + \log(1 - D(\mathbf{x}, \mathbf{y}))\;]$$

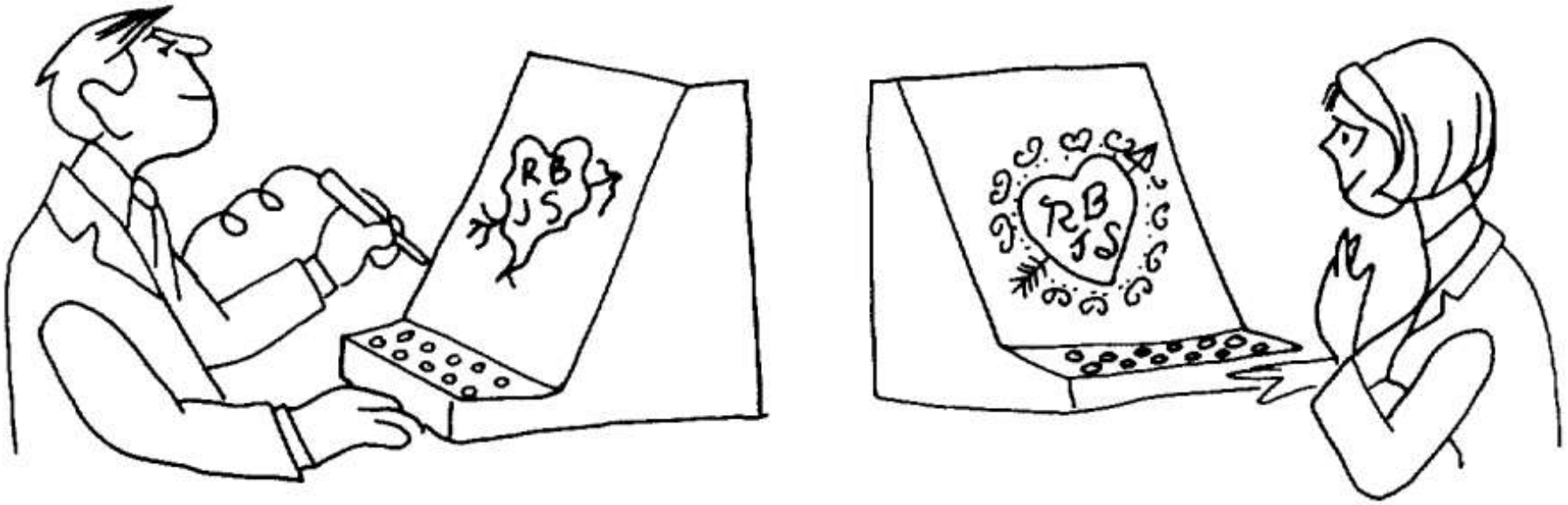


$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(\mathbf{x}, G(\mathbf{x})) + \log(1 - D(\mathbf{x}, \mathbf{y}))\;]$$



$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(\mathbf{x}, G(\mathbf{x})) + \log(1 - D(\mathbf{x}, \mathbf{y})) \;]$$

1. Image synthesis
2. Structured prediction
3. Domain mapping

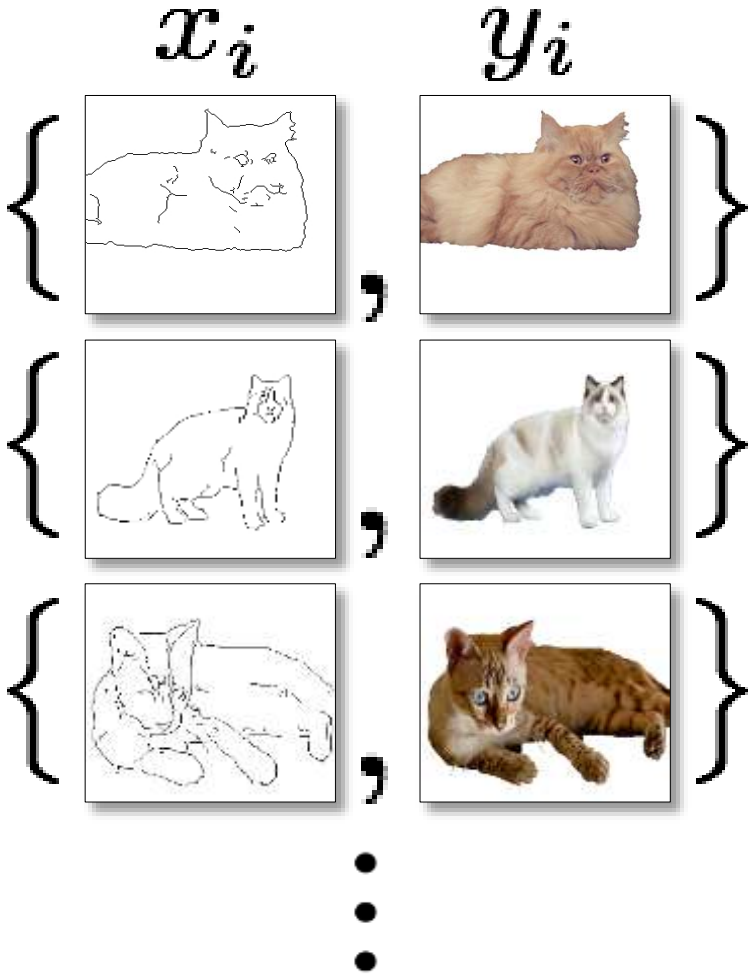


Domain mapping

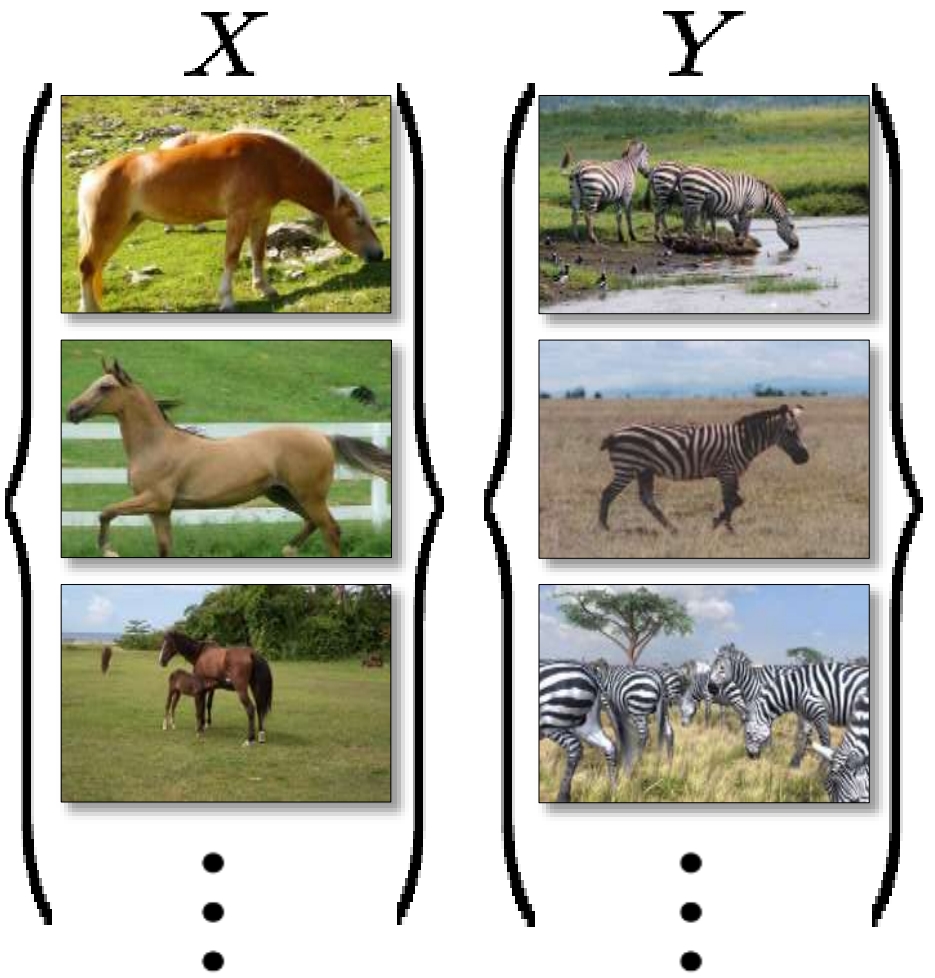
[Includes slides from Jun-Yan Zhu, Taesung Park]

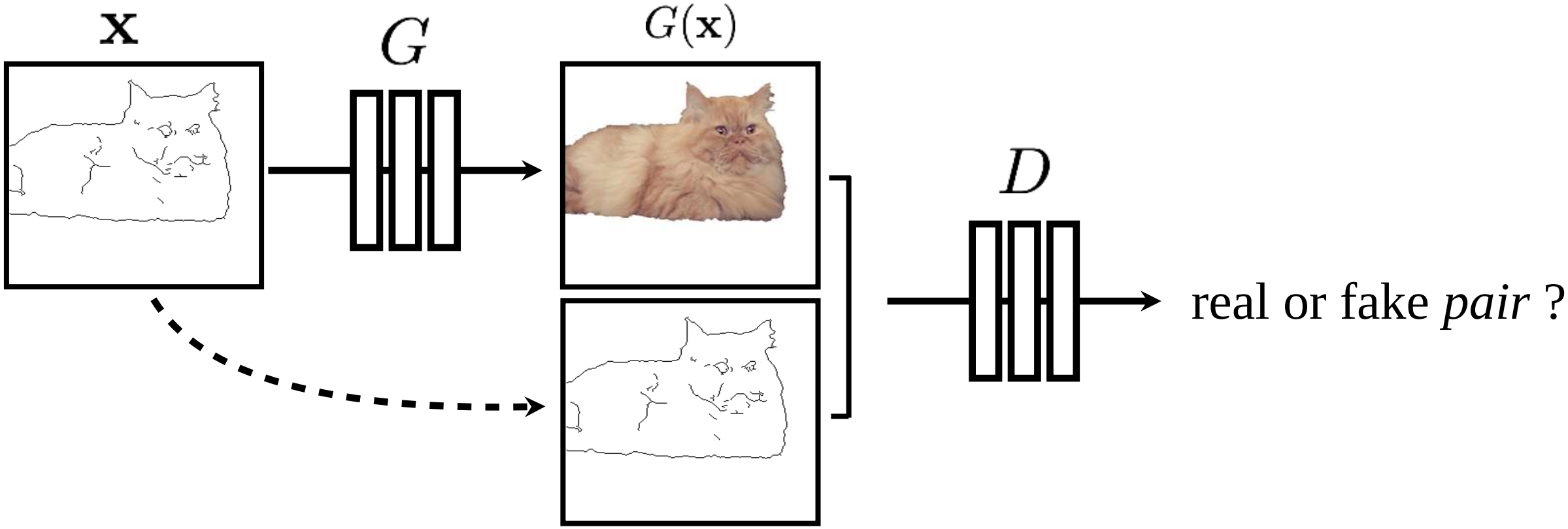
[Cartoon: The Computer as a Communication Device, Licklider & Taylor 1968]

Paired data

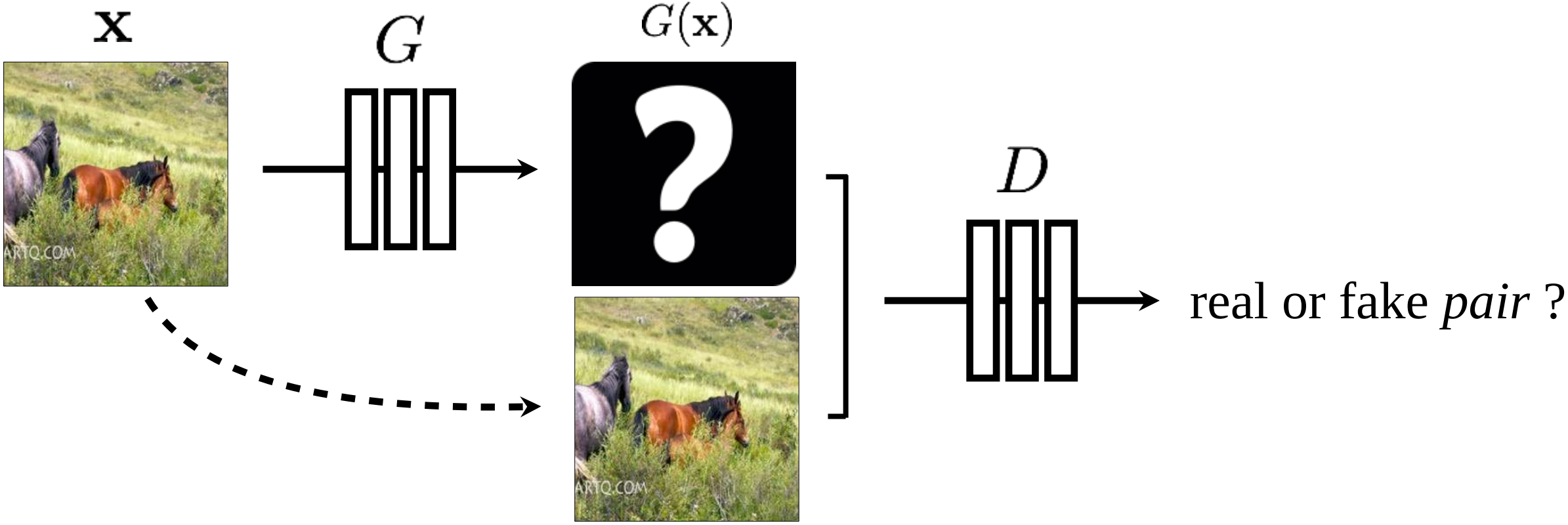


Unpaired data



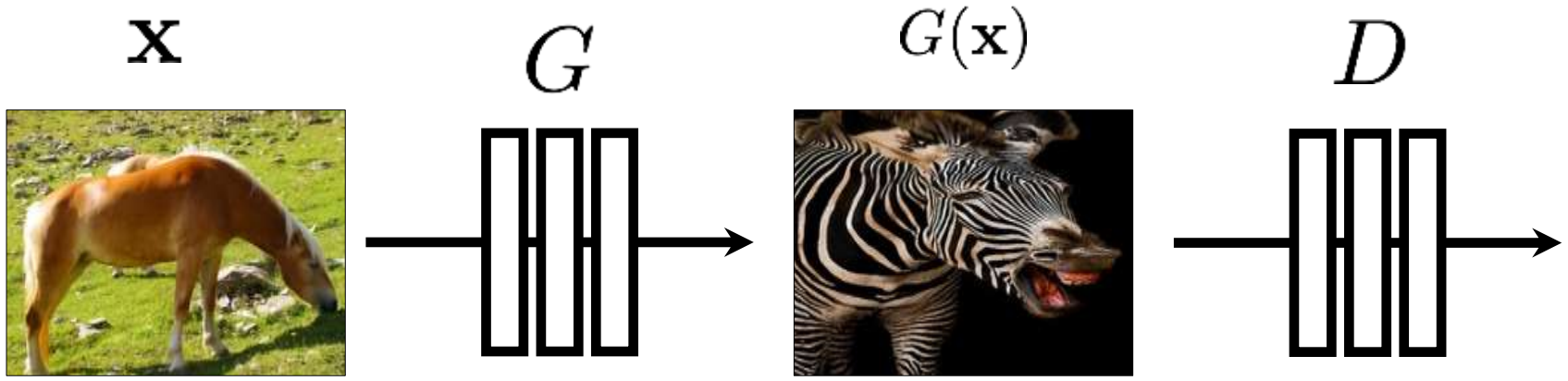


$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\;\; \log D(\mathbf{x}, G(\mathbf{x})) + \log(1 - D(\mathbf{x}, \mathbf{y}))\;\;]$$



$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(\mathbf{x}, G(\mathbf{x})) + \log(1 - D(\mathbf{x}, \mathbf{y})) \;]$$

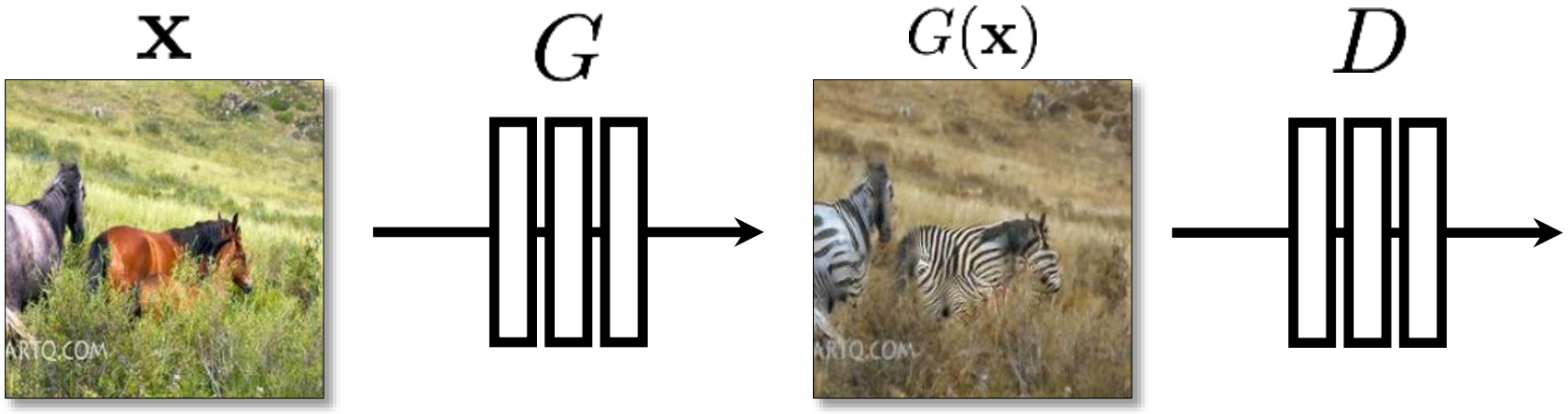
No input-output pairs!

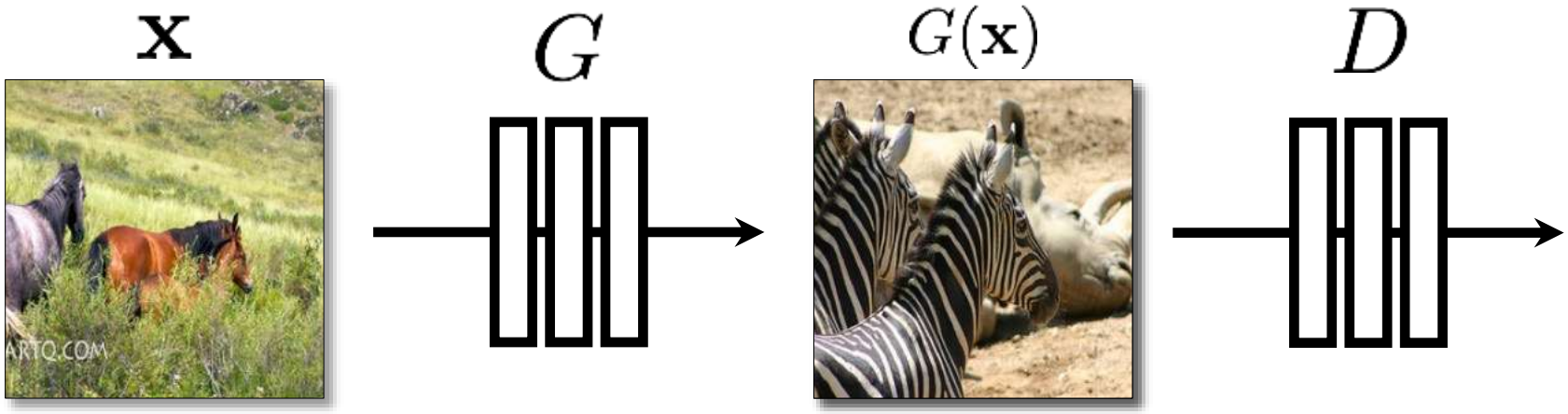


$$\arg\min_G \max_D \; \mathbb{E}_{\mathbf{x},\mathbf{y}}[\; \log D(G(\mathbf{x})) \quad + \quad \log(1 - D(\mathbf{y}))\;]$$

Usually loss functions check if output matches a target *instance*

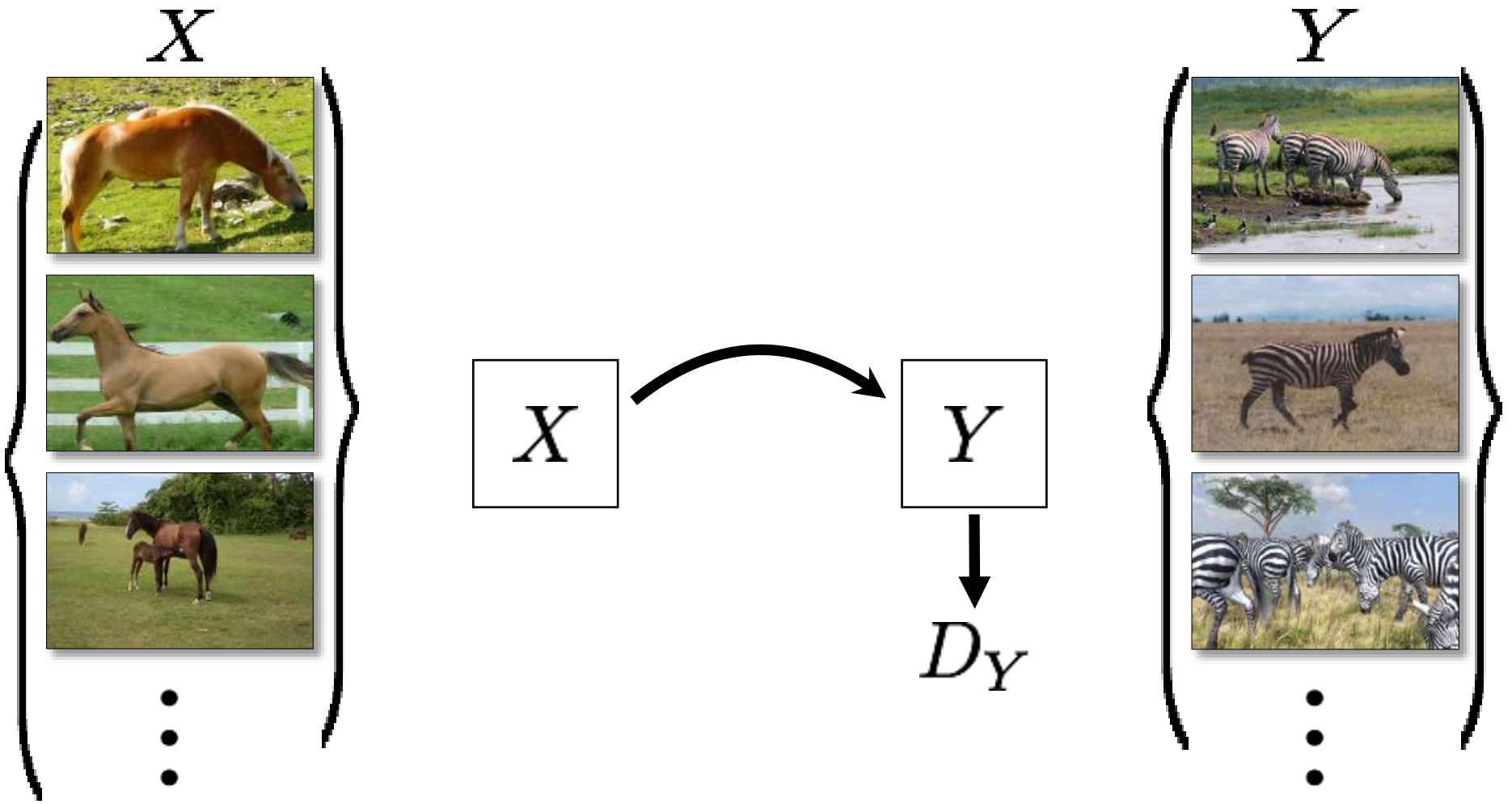
GAN loss checks if output is part of an admissible *set*


$\mathbf{x}$
G
$G(\mathbf{x})$
D
Real!



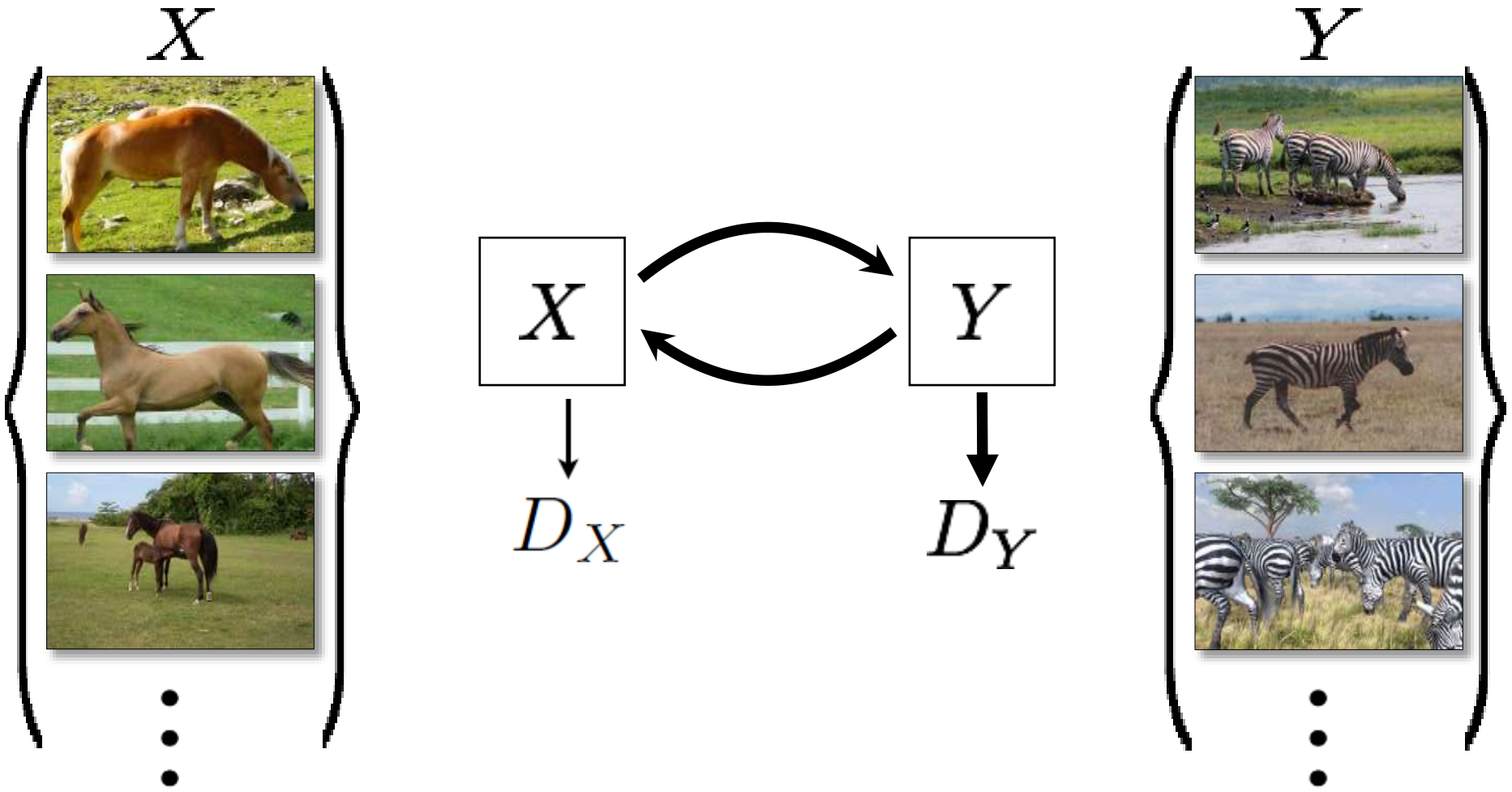
Nothing to force output to correspond to input

CycleGAN, or there and back aGAN

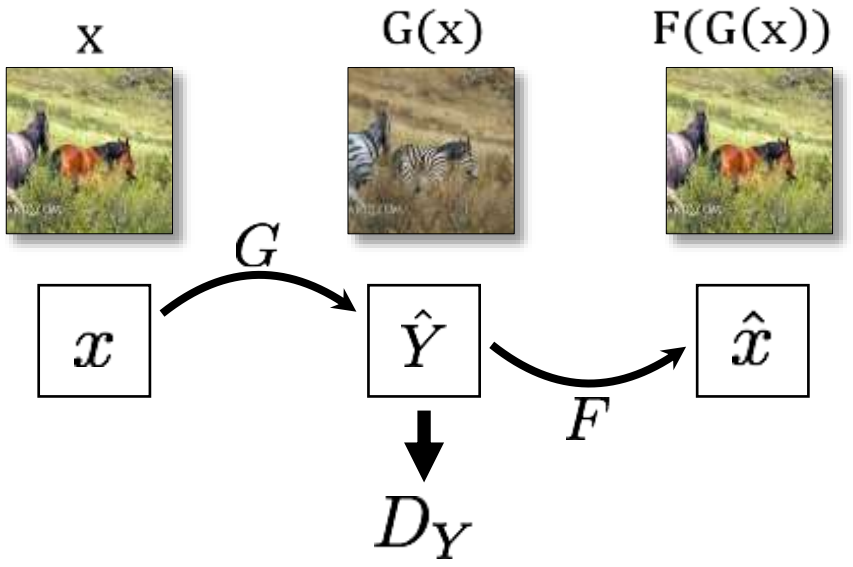


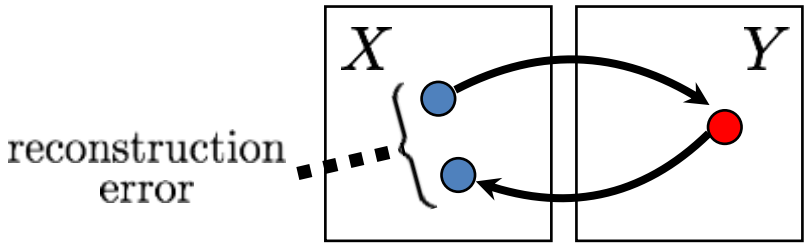
[Zhu*, Park* et al. 2017], [Yi et al. 2017], [Kim et al. 2017]

CycleGAN, or there and back aGAN



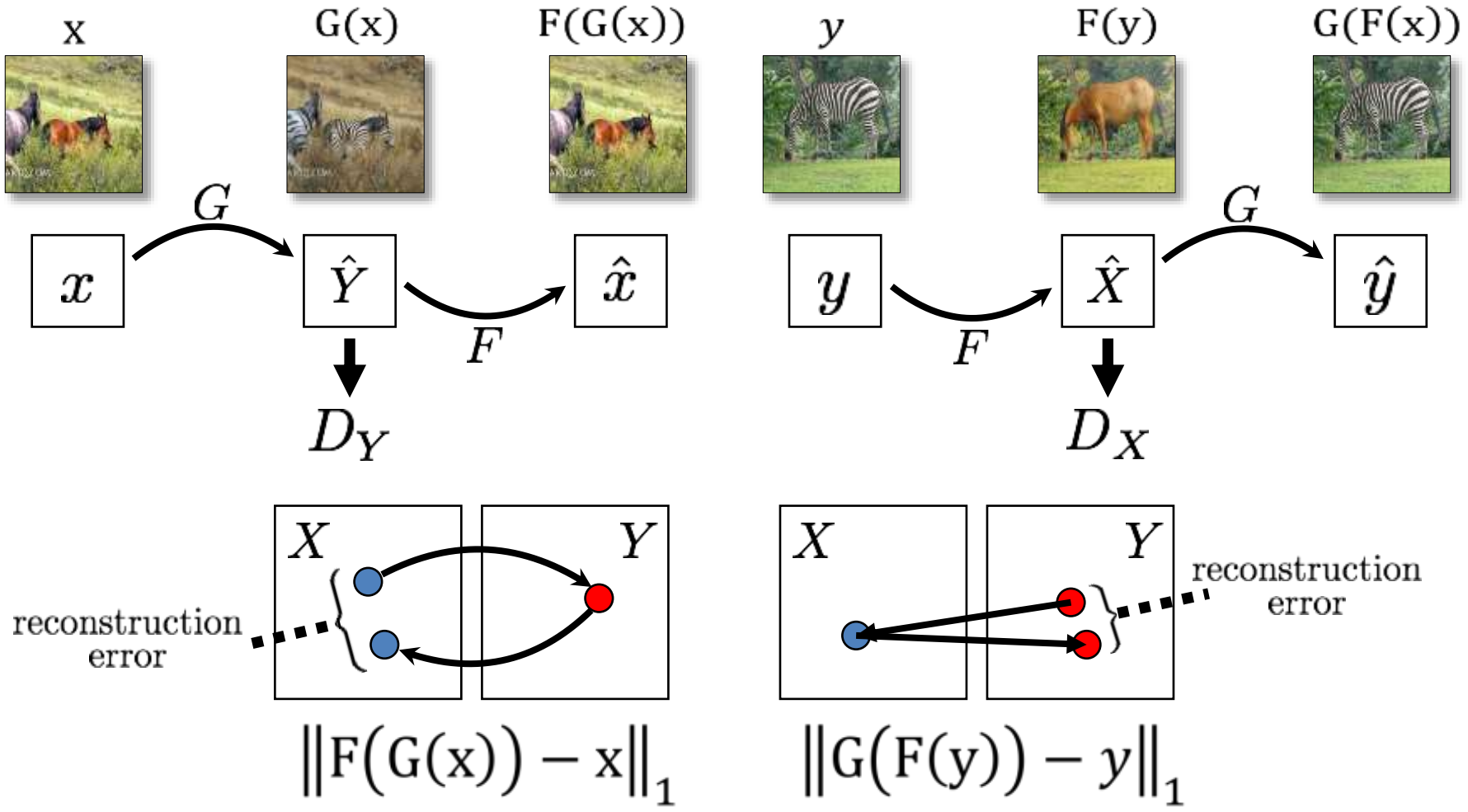
Cycle Consistency Loss





$$\|F(G(x)) - x\|_1$$

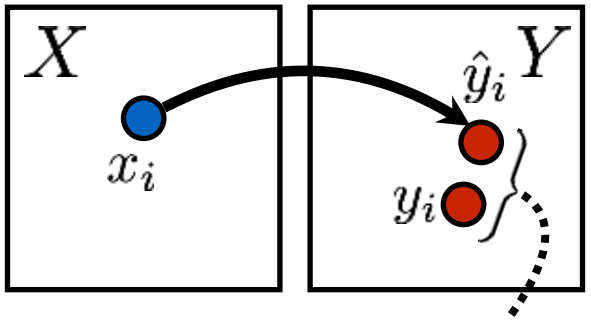
Cycle Consistency Loss



Paired translation

Training data

Objective



regression error

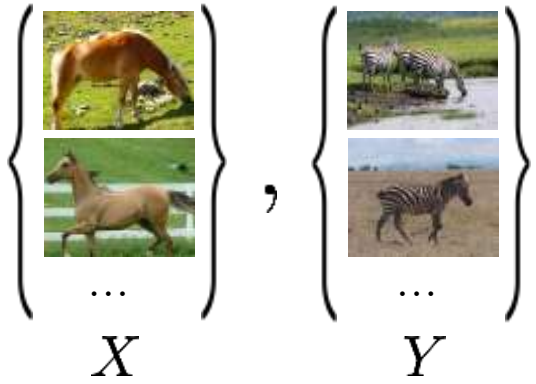
Input

Result

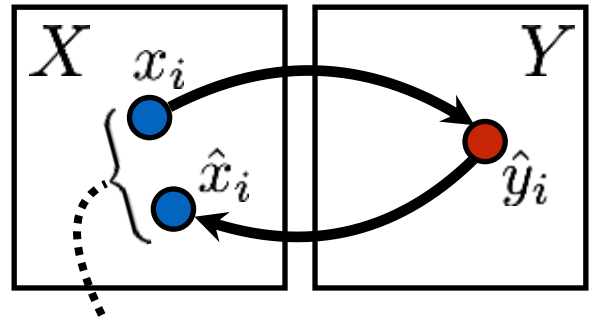
["pix2pix", Isola, Zhu, Zhou, Efros, 2017]

Unpaired translation

Training data



Objective

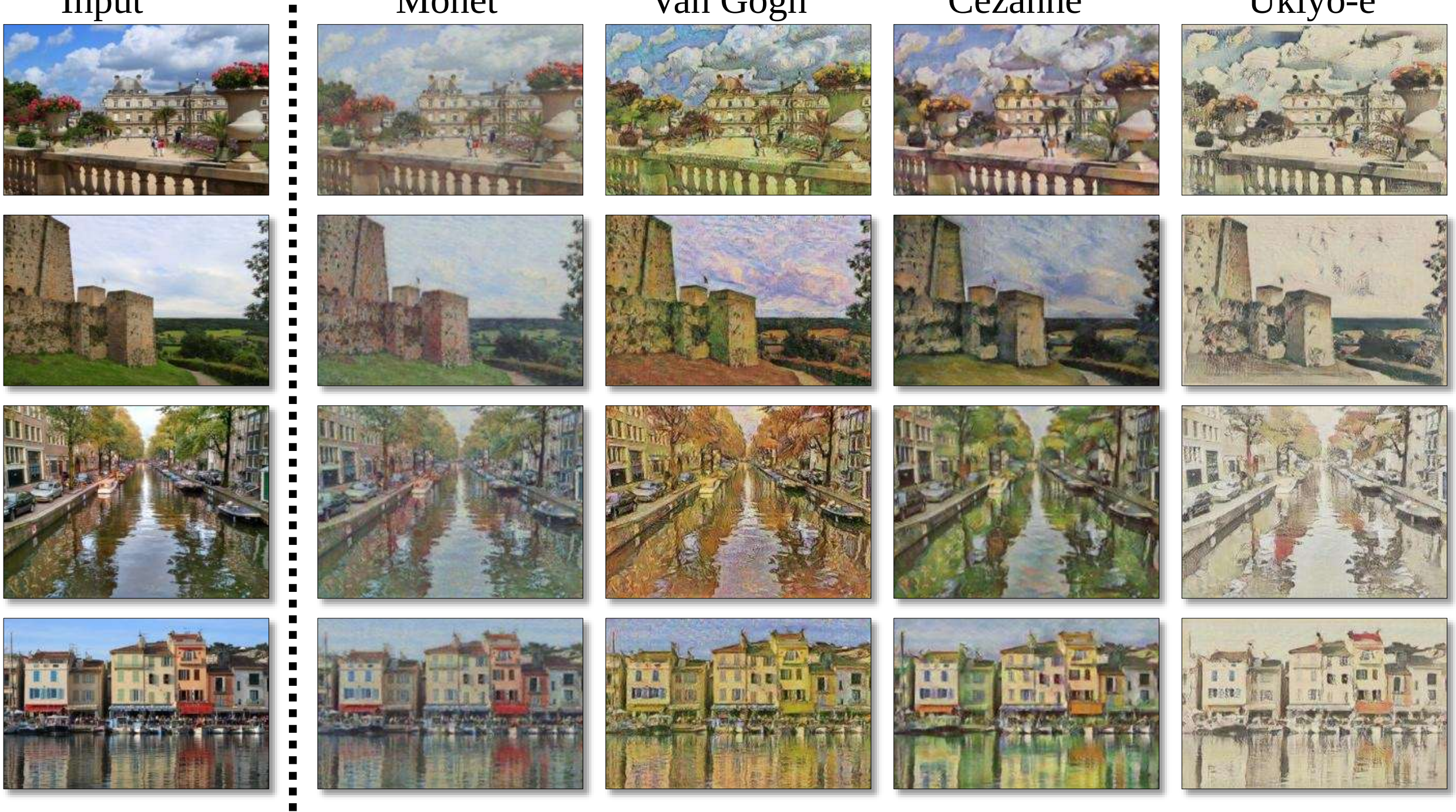


cycle-consistency error

Input

Result

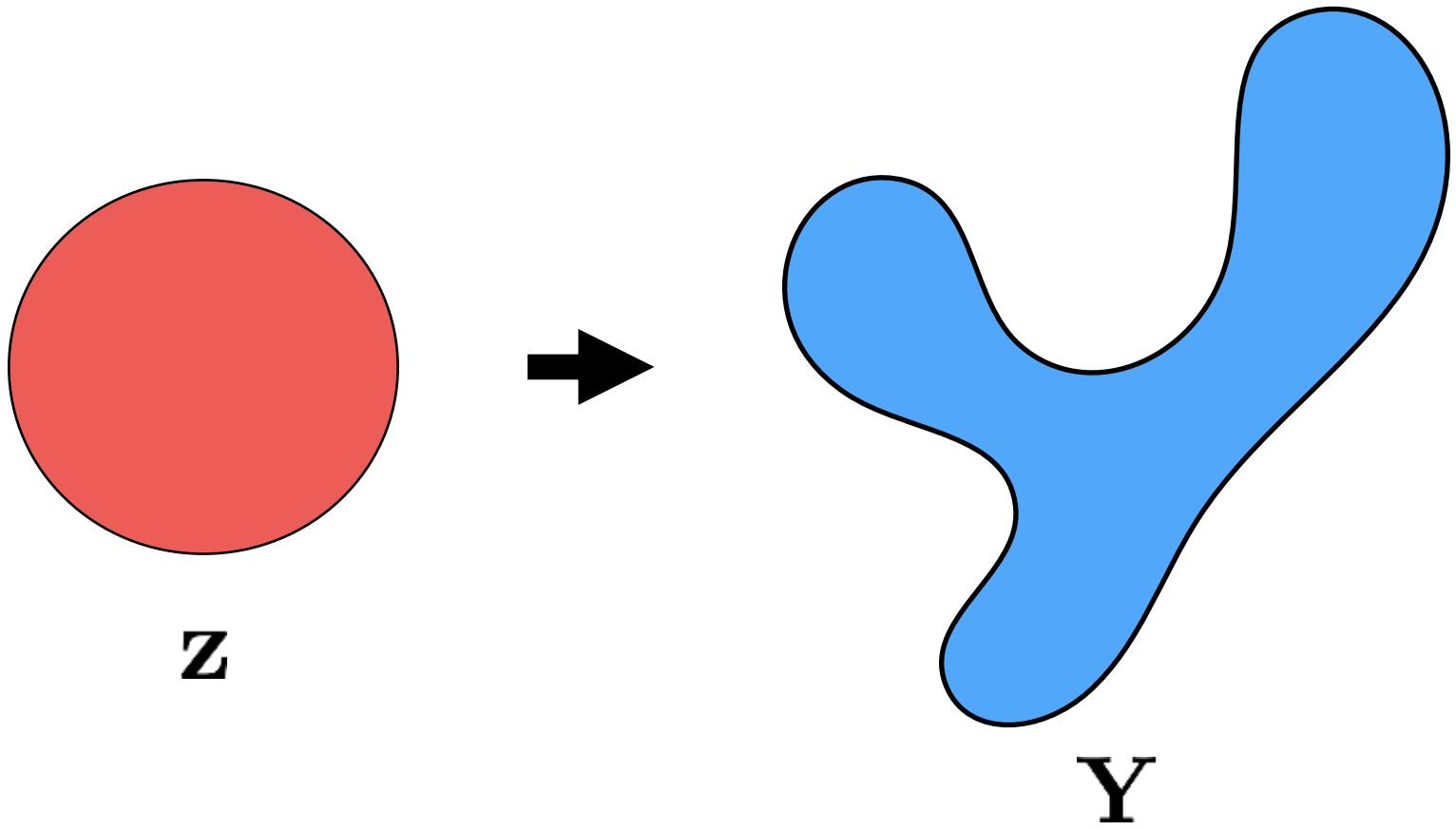
["CycleGAN", Zhu*, Park*, Isola, Ef


Input
Monet
Van Gogh
Cezanne
Ukiyo-e

GANs

Gaussian

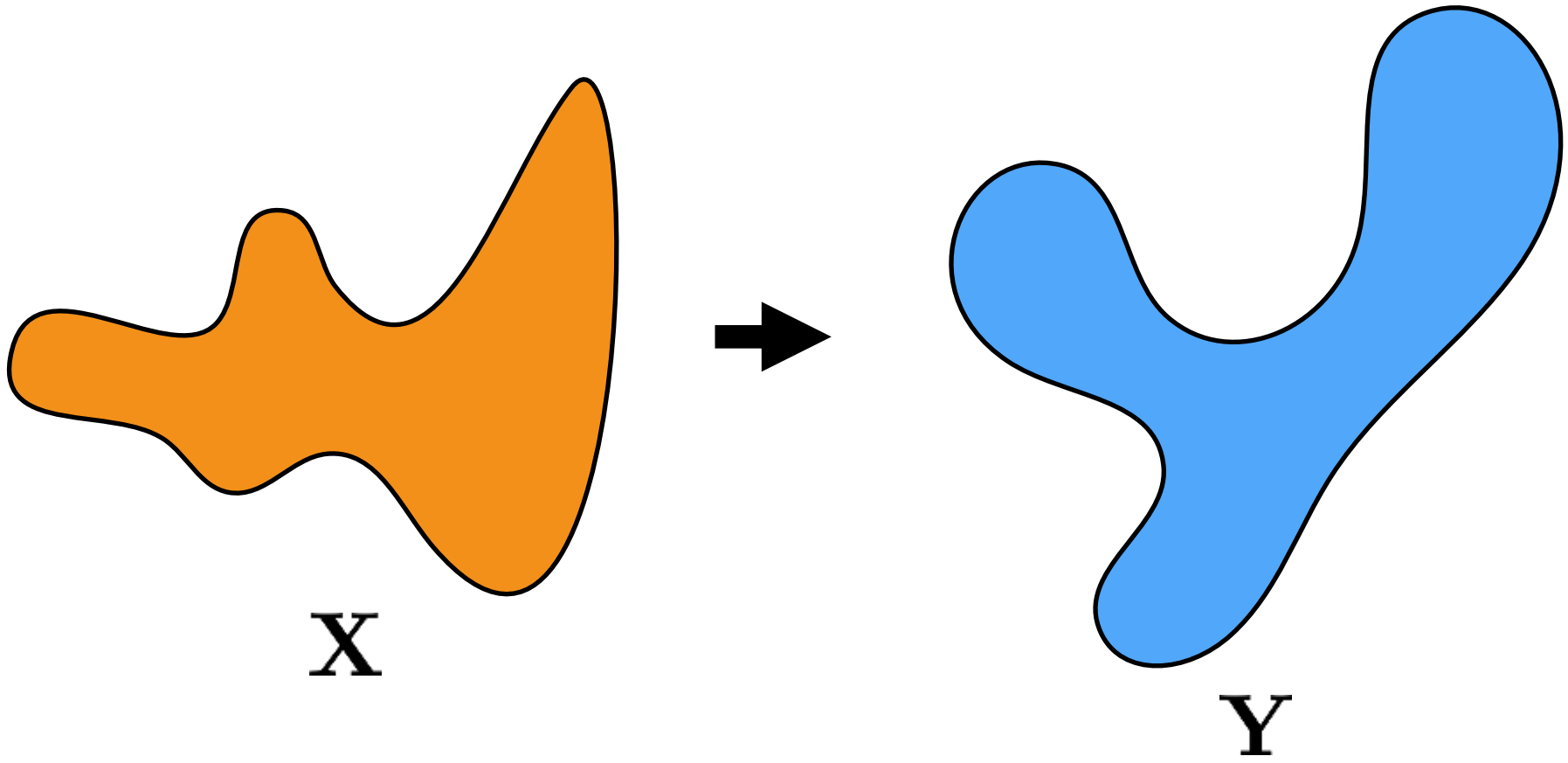
Target distribution



CycleGAN

Horses

Zebras



Wasserstein GAN

W-GAN in Short

For mathematicians:

- Wasserstein distance, instead of JS divergence

For engineers:

- remove logarithms
- clip weights

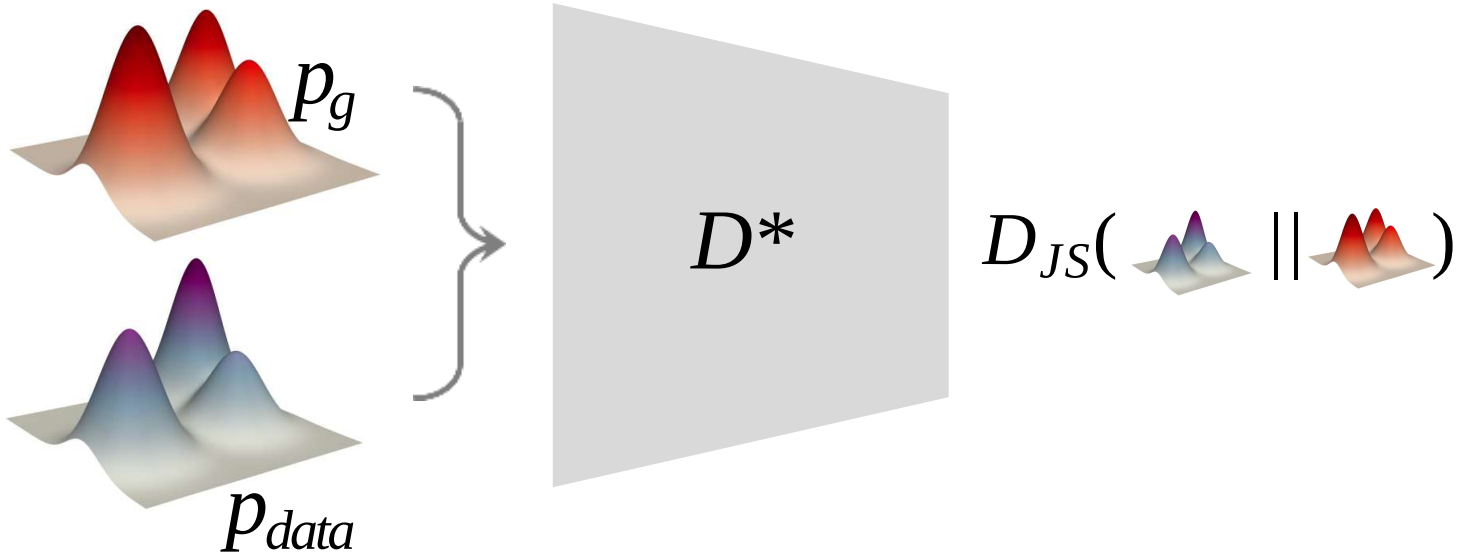
For laymen:

- art critic, instead of forgery expert

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

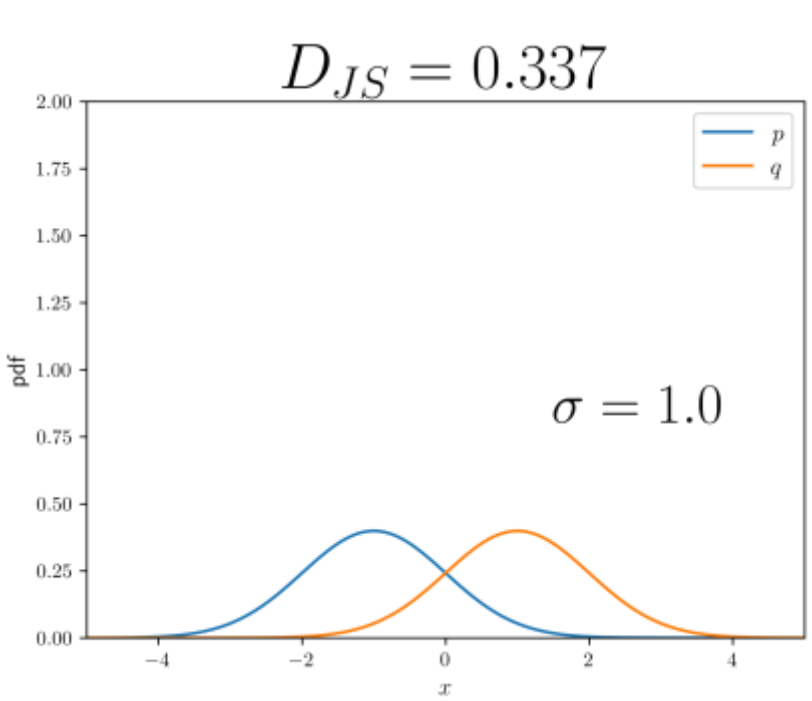
Recap: GAN optimizes for D_{JS}

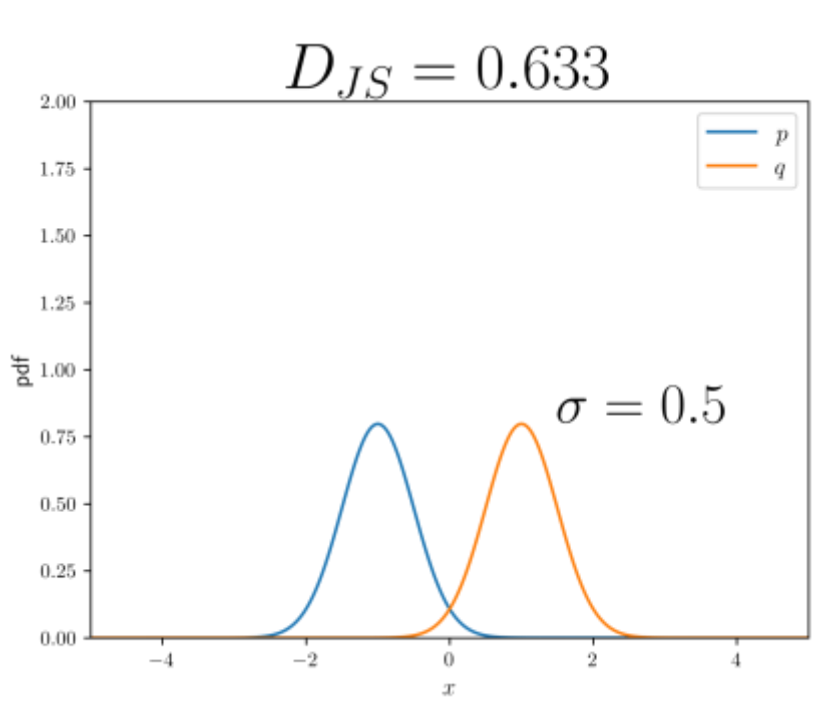
$$\mathcal{L}(D^*, G) = 2D_{JS}(p_{\text{data}} \| p_g) - 2\log 2$$



Problems of D_{JS}

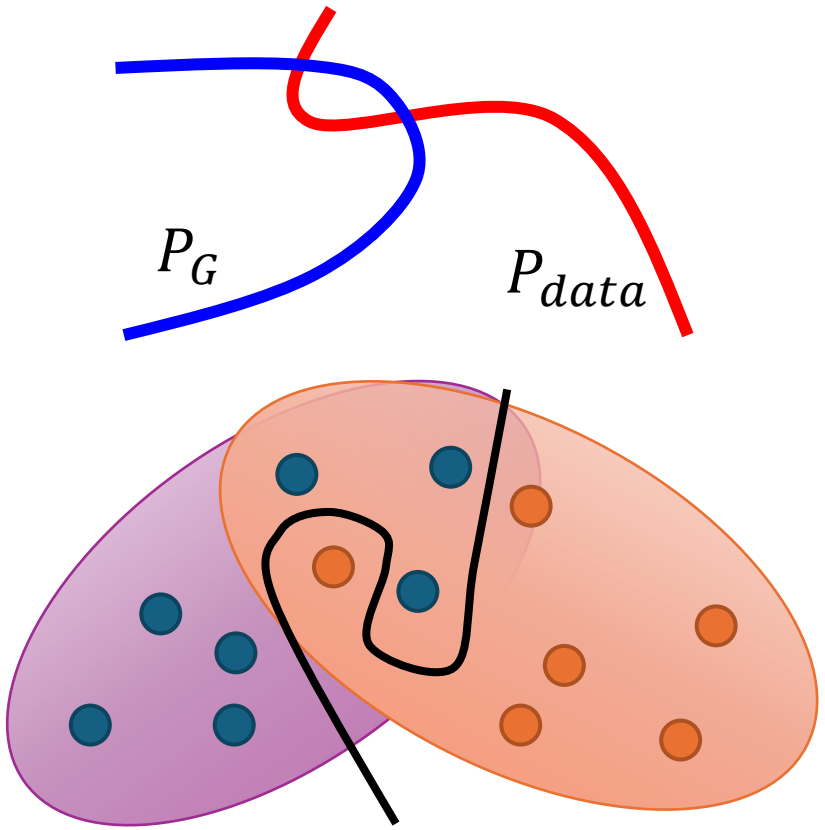
If p and q don't overlap, D_{JS} is a constant (log2), i.e., no gradient





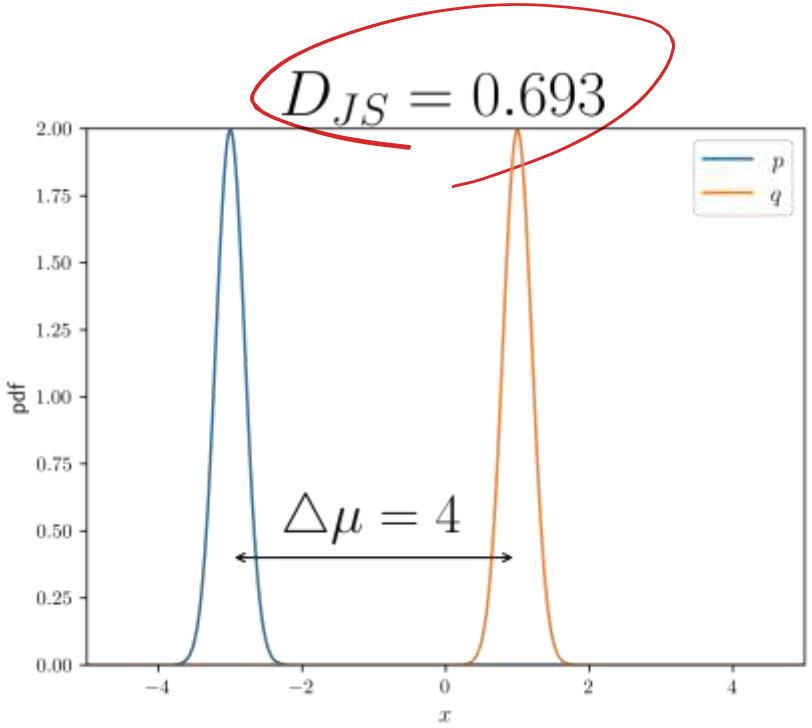
Problems of D_{JS}

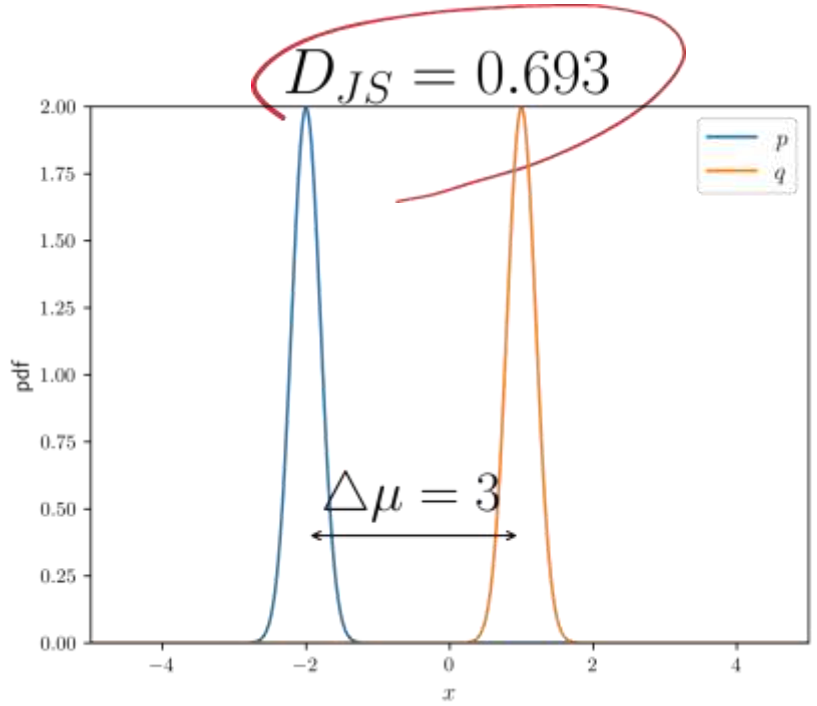
Why it is common p and q don't overlap?

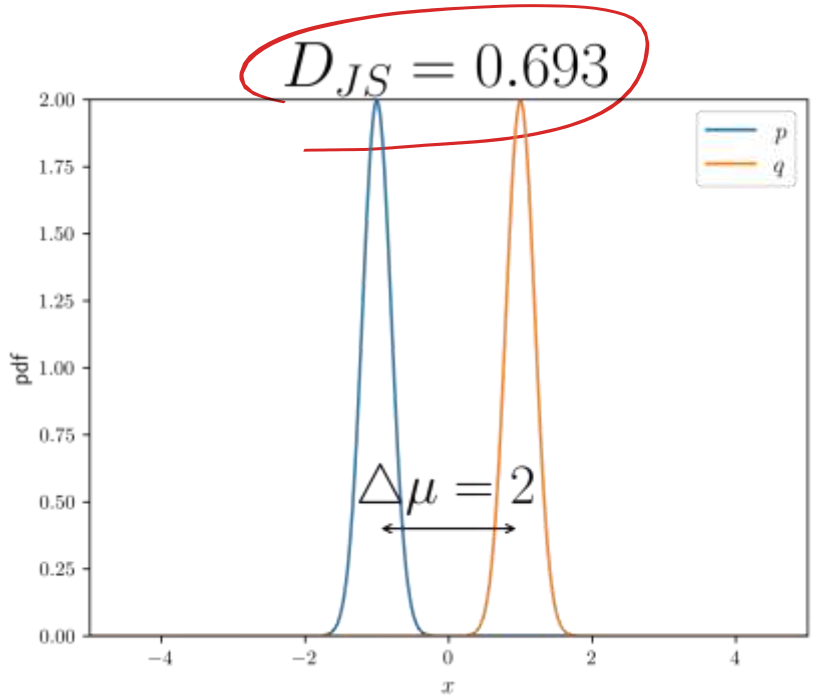


Problems of D_{JS}

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

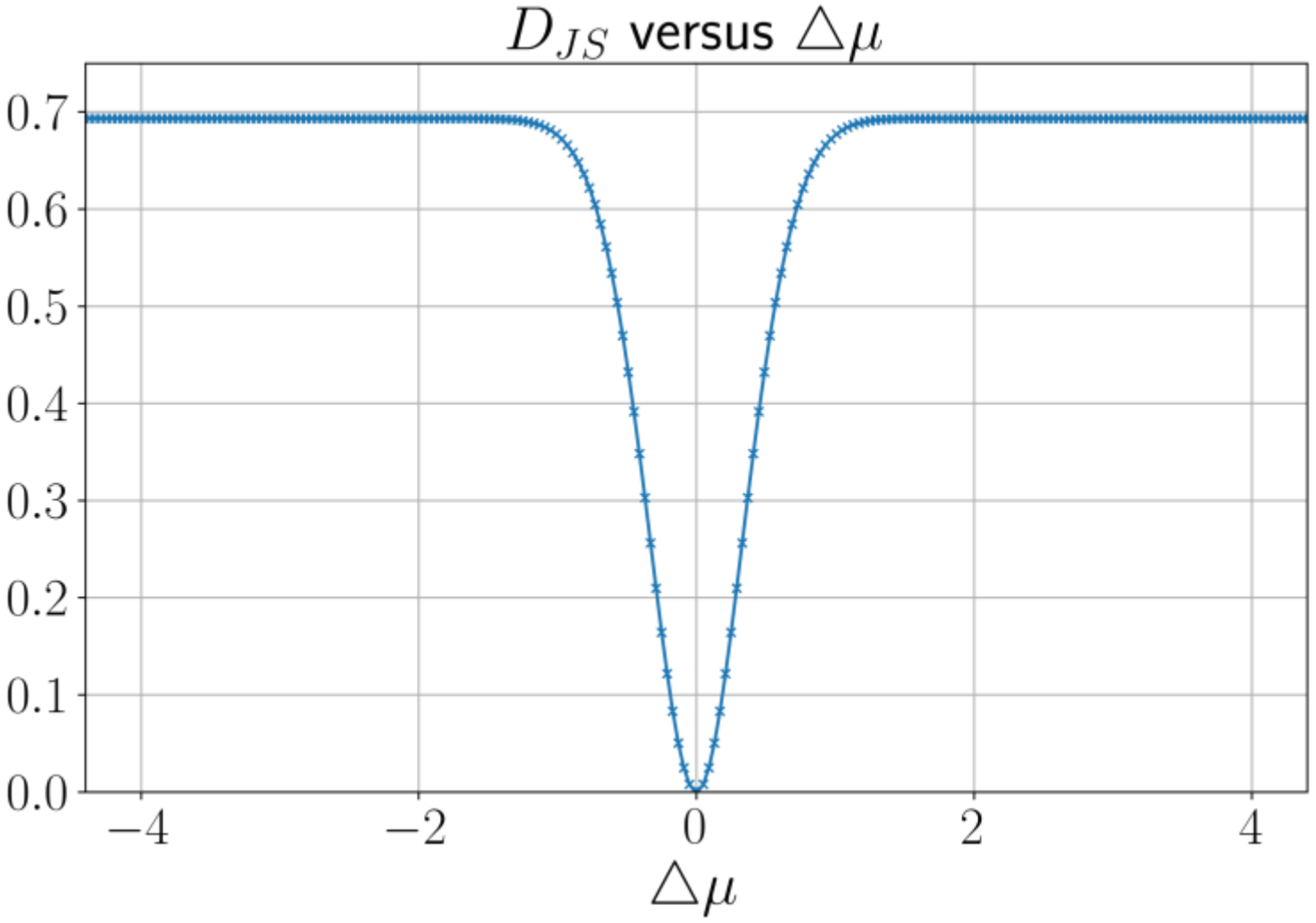


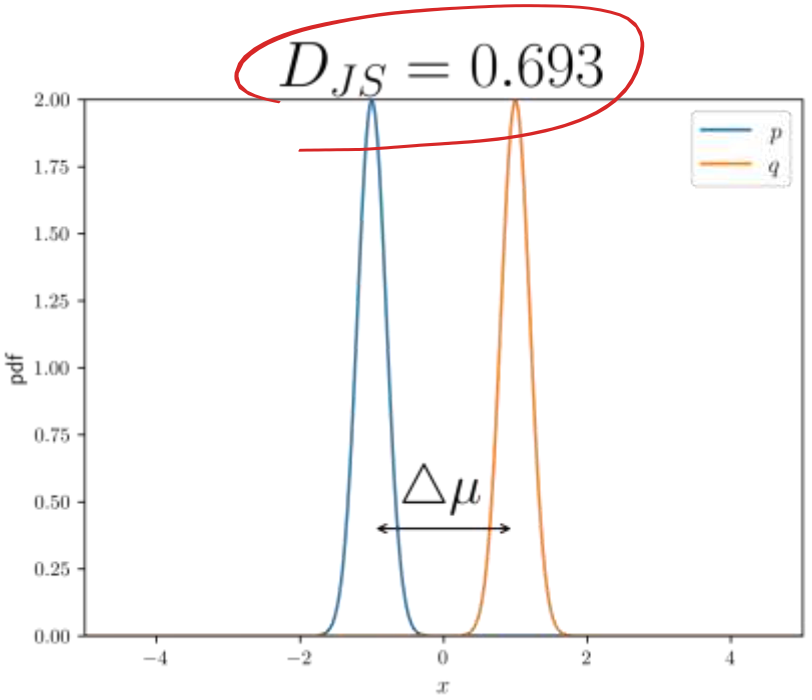




Problems of D_{JS}

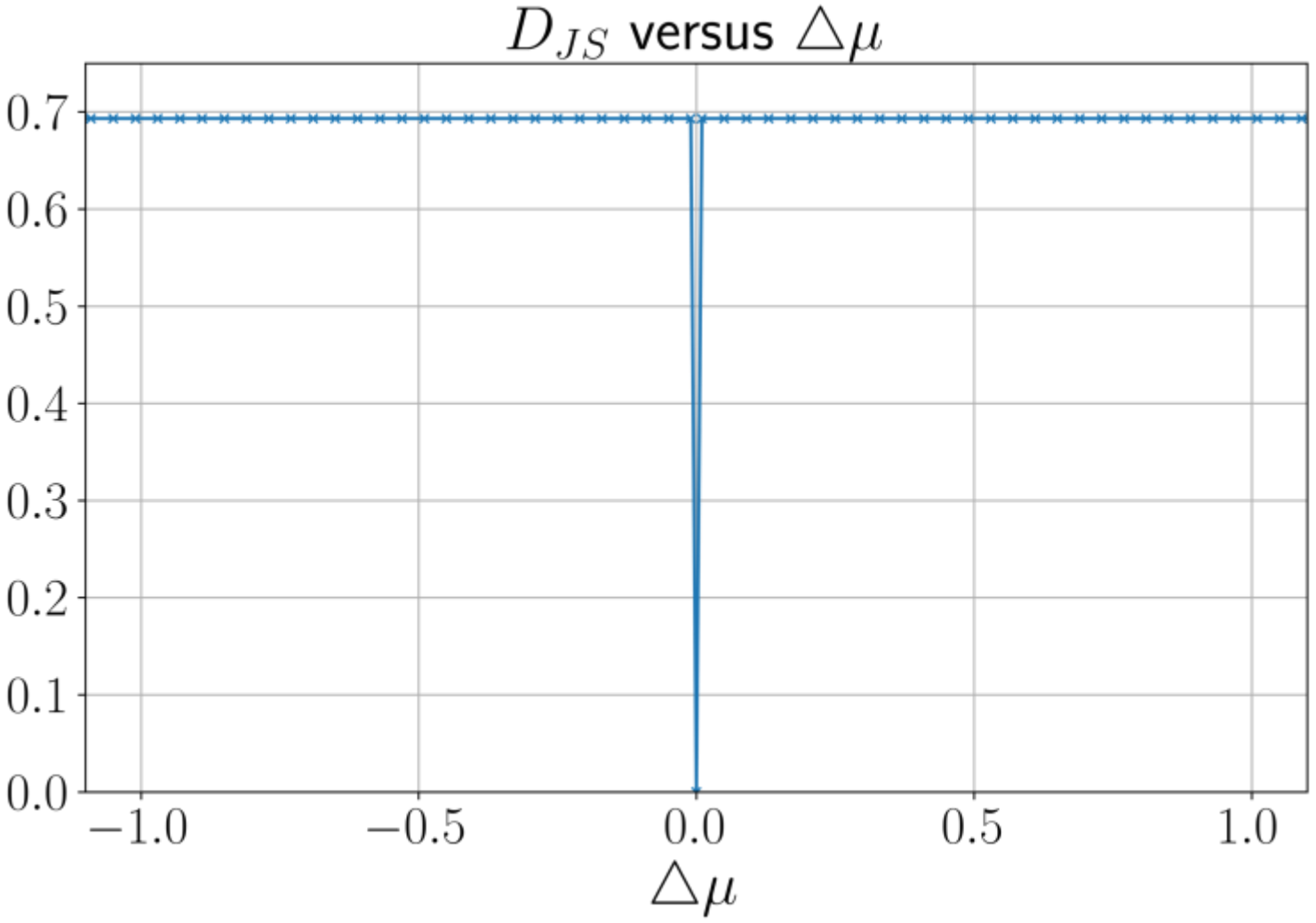
- D_{JS} is useful only if p and q are close

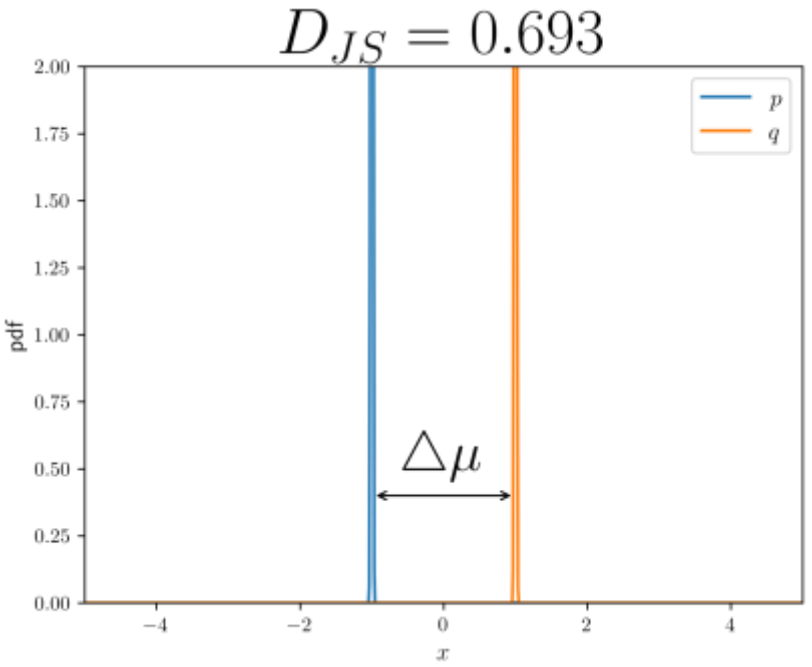




Problems of D_{JS}

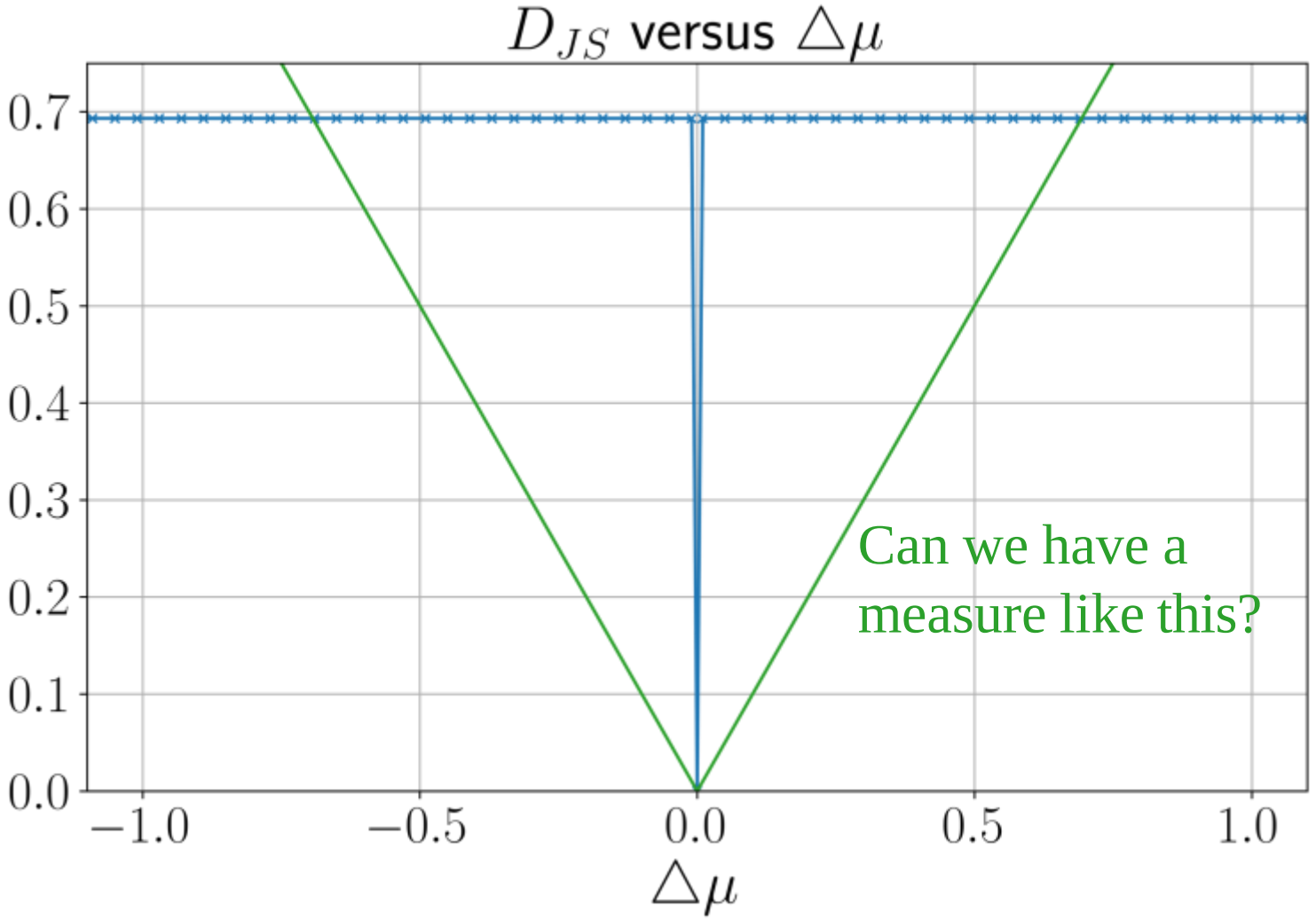
- D_{JS} is a delta function when p and q are delta functions

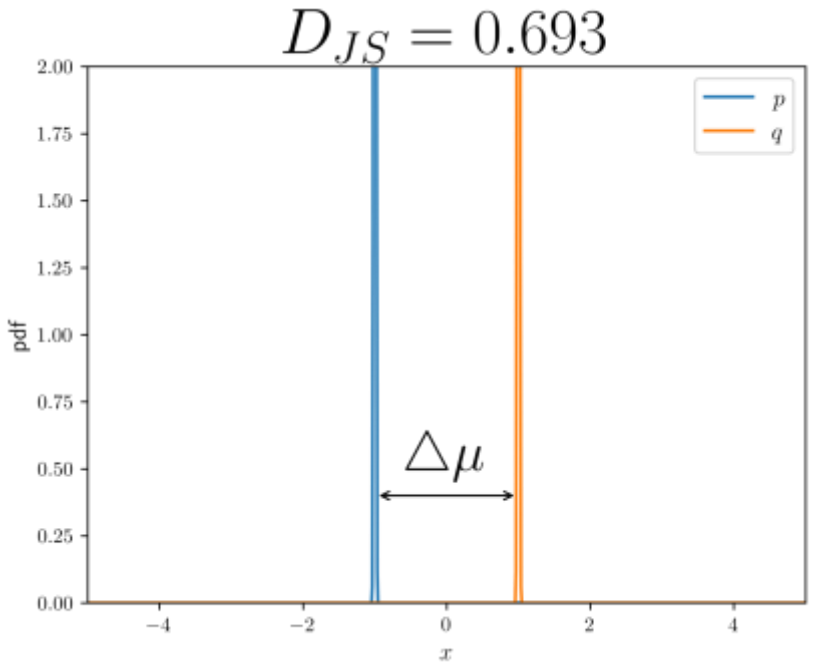




Problems of D_{JS}

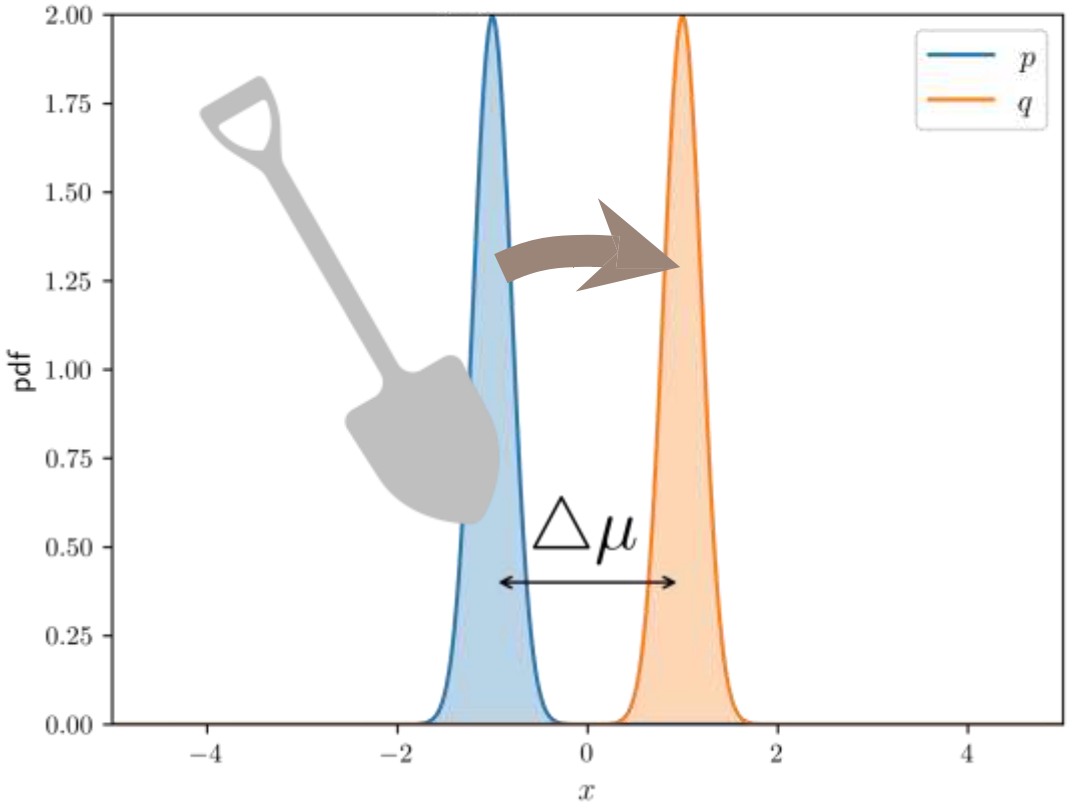
- D_{JS} is a delta function when p and q are delta functions

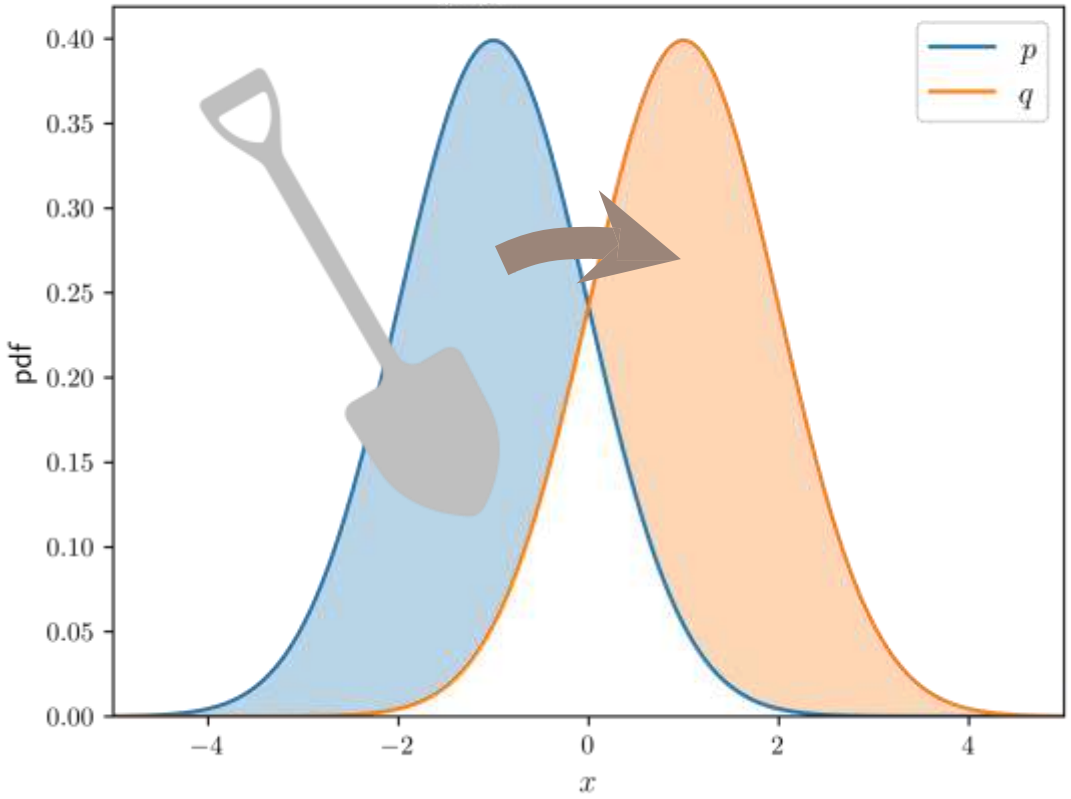




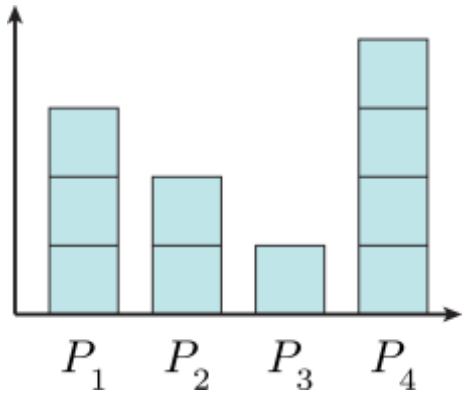
Wasserstein Distance

"Earth Mover's Distance"





Running example: Wasserstein Distance



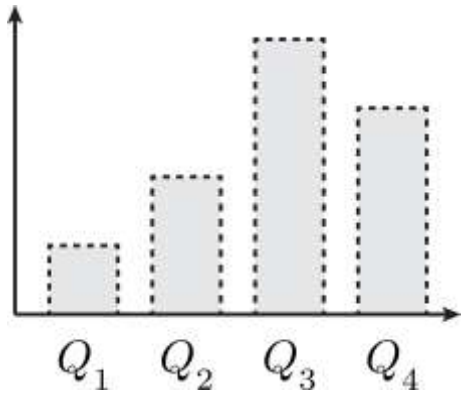
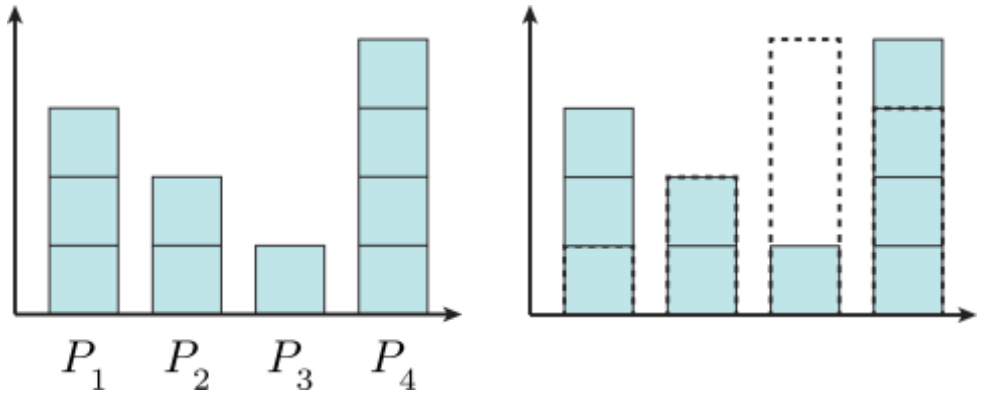


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance



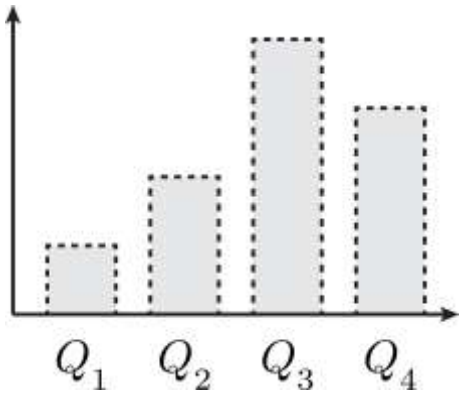
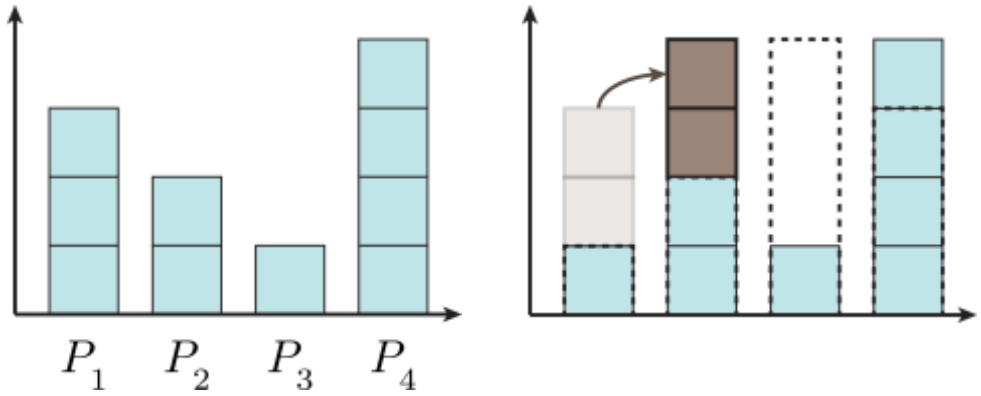


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

2 shovelfuls



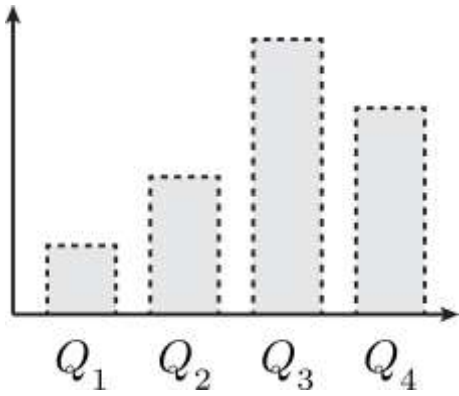


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

2 shovelfuls

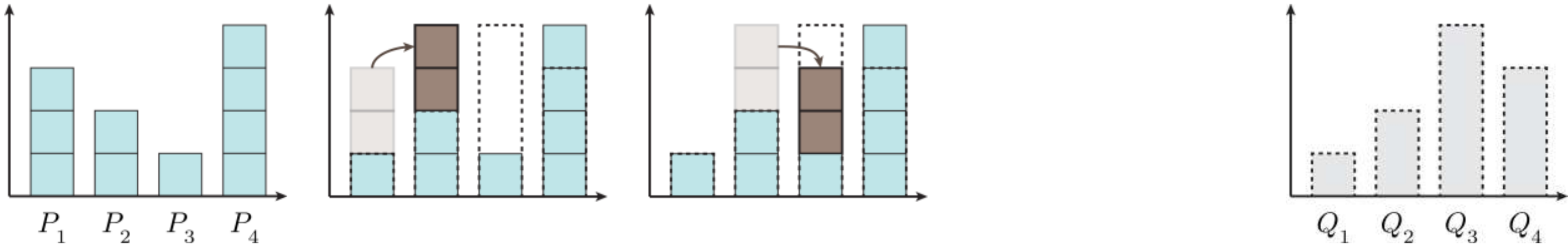


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

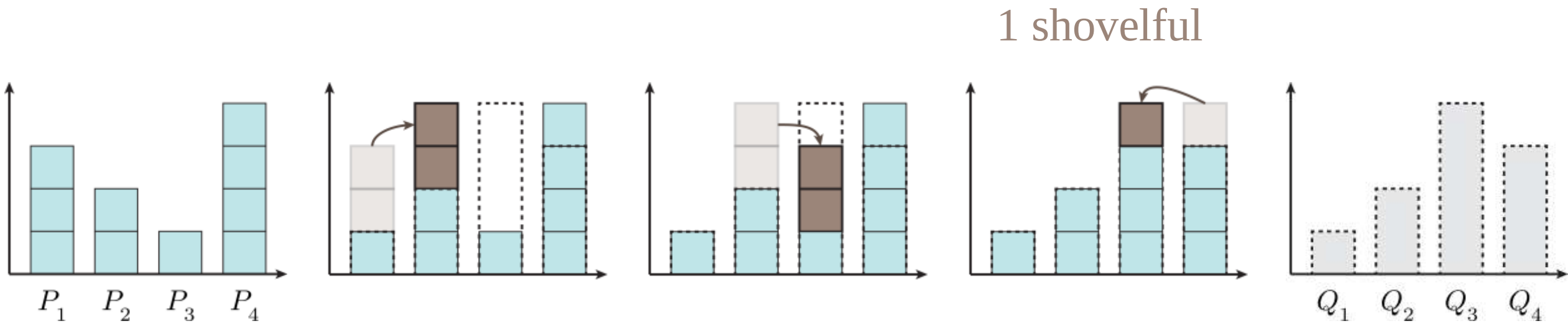


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

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

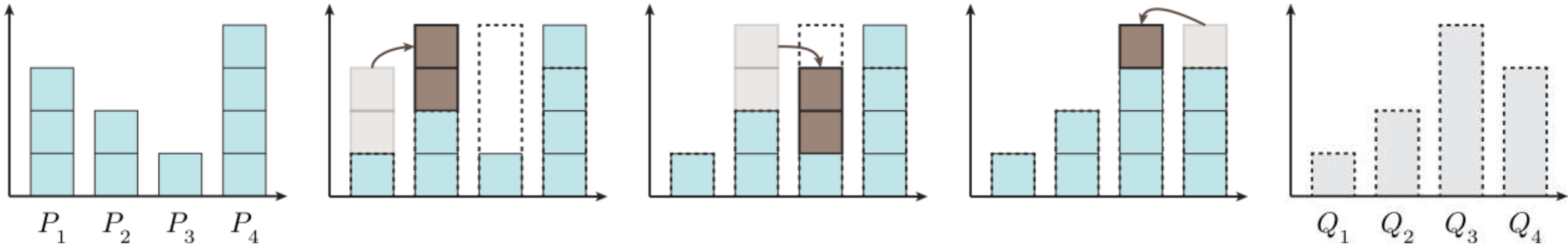
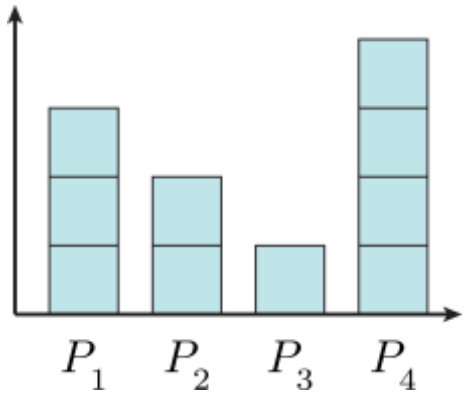


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance



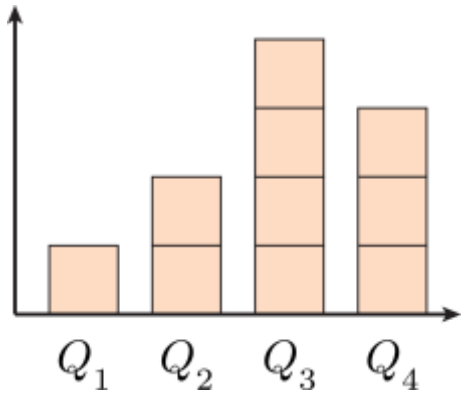
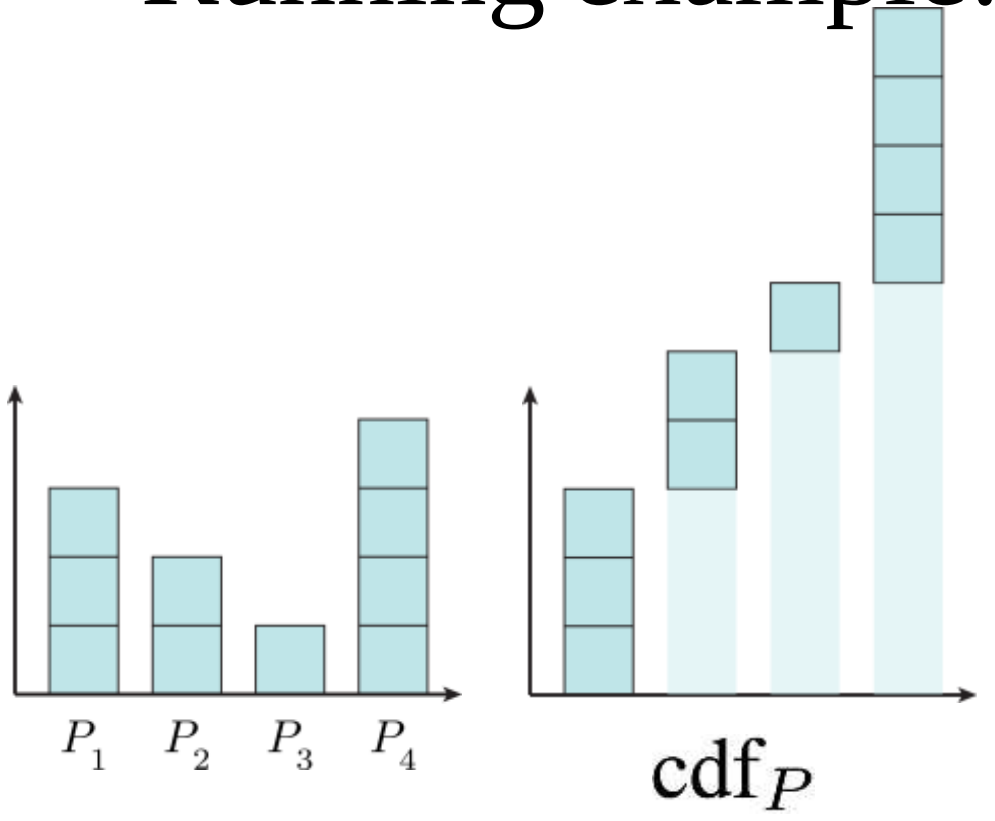
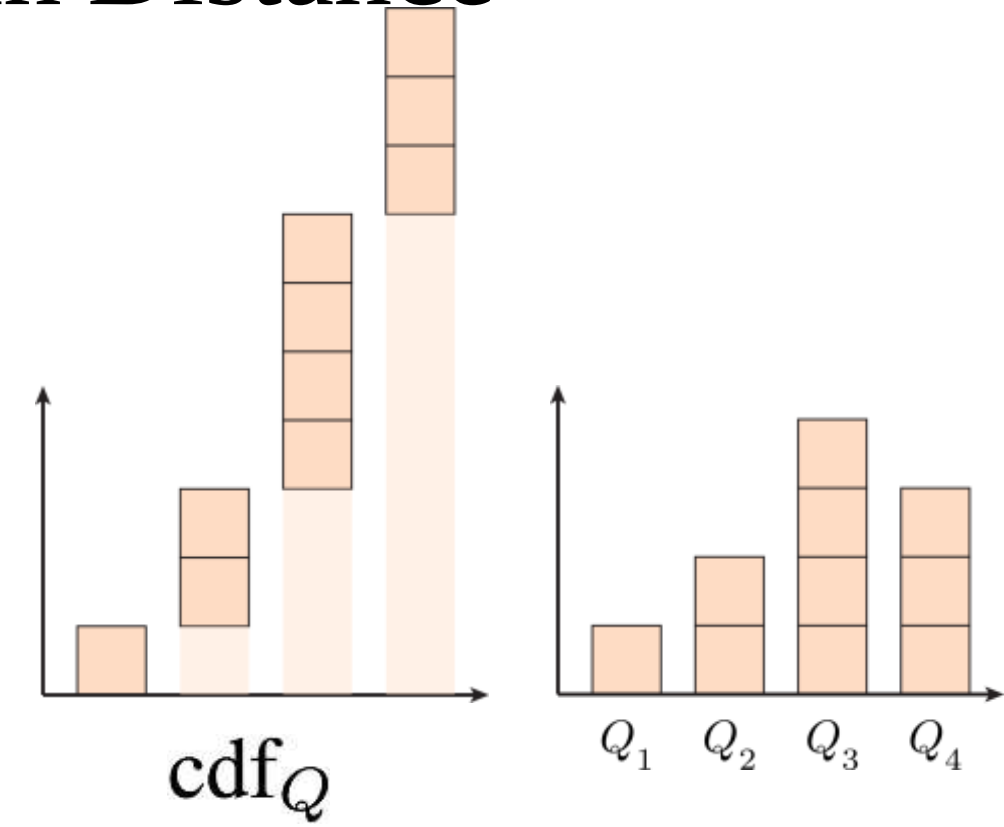


Figure inspired by: Lilian Weng, “From GAN to WGAN”, arXiv:1904.08994

Running example: Wasserstein Distance

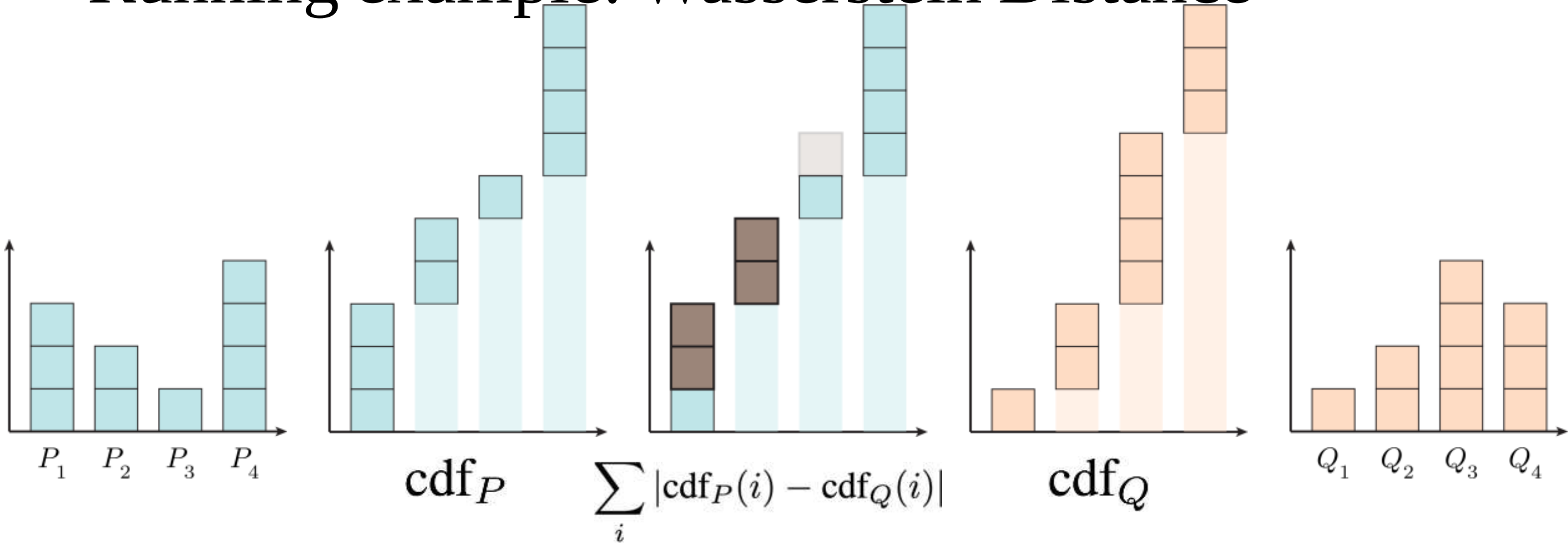




- cdf: cumulative distribution function

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

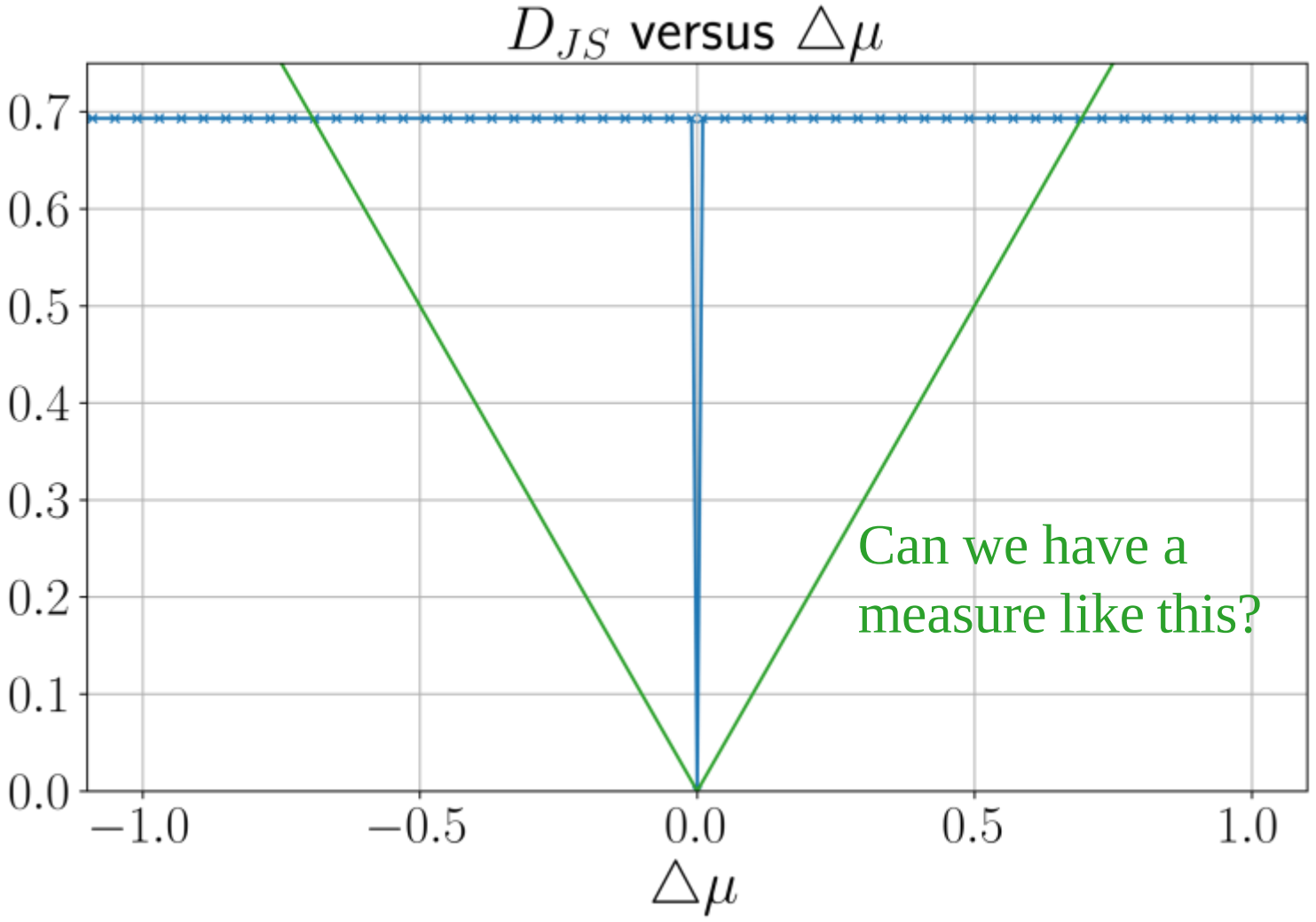
$$W_1(P, Q) = \sum_i |\text{cdf}_P(i) - \text{cdf}_Q(i)|$$

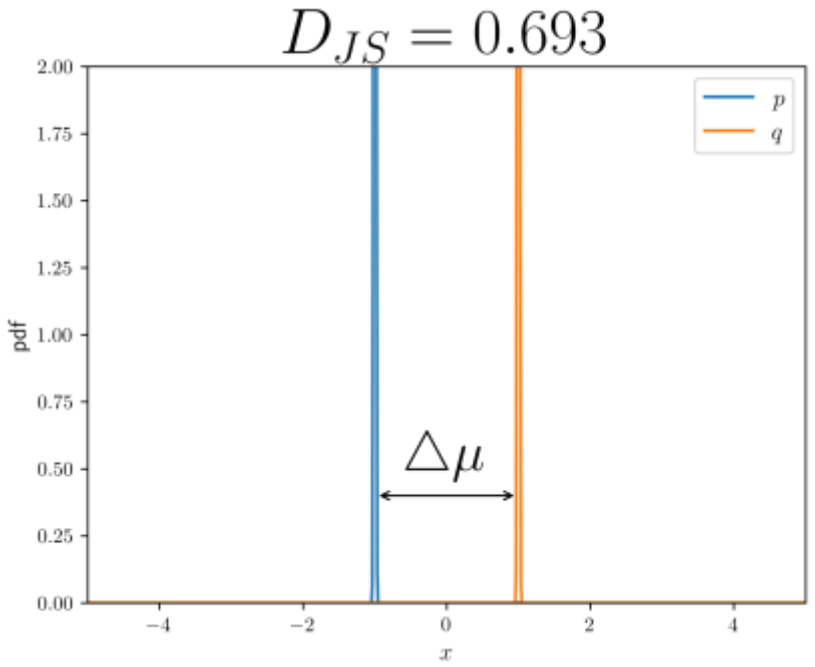
- 1-Wasserstein Distance (1-d, continuous)

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

Recap: D_{JS}

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

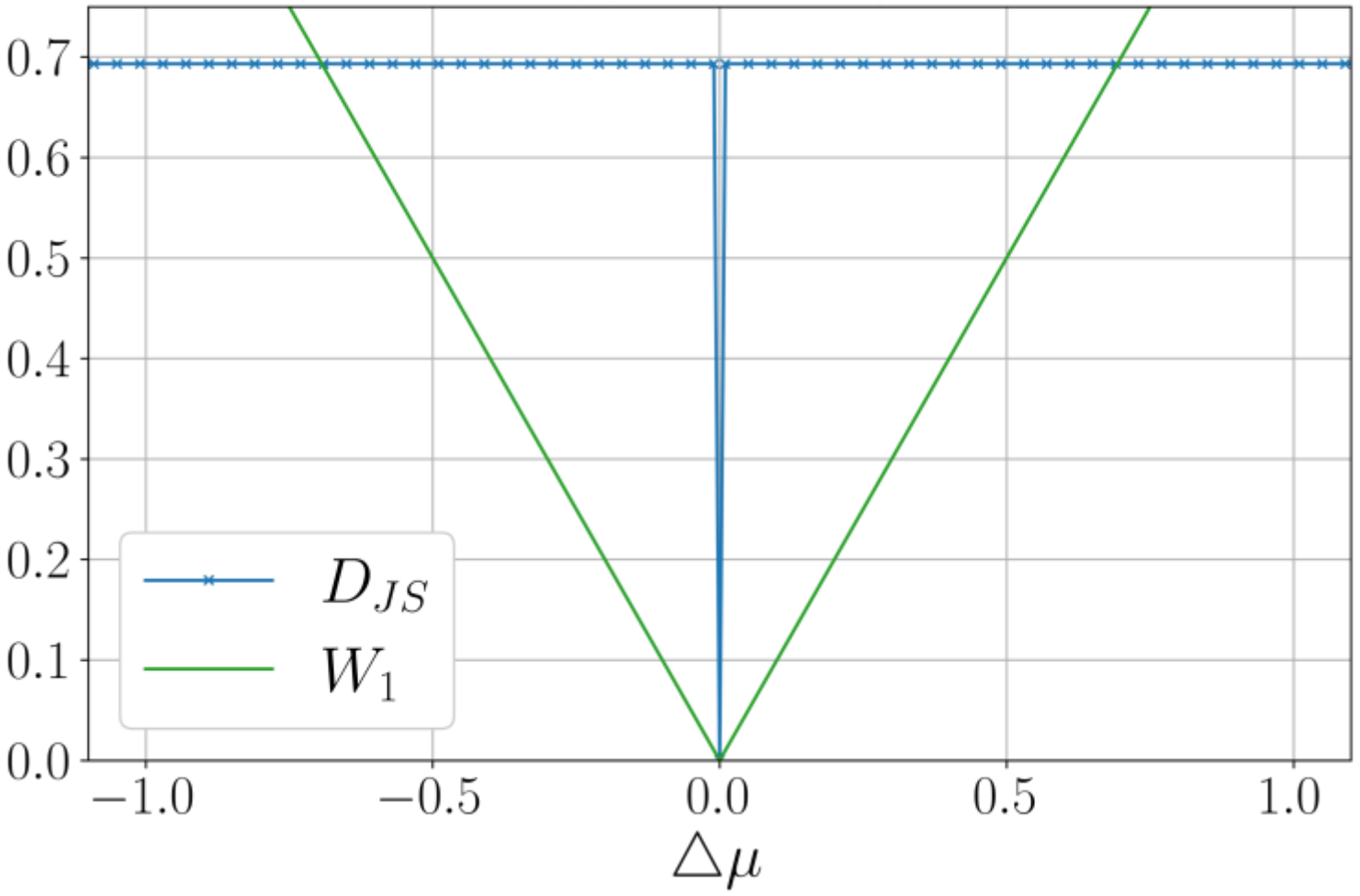


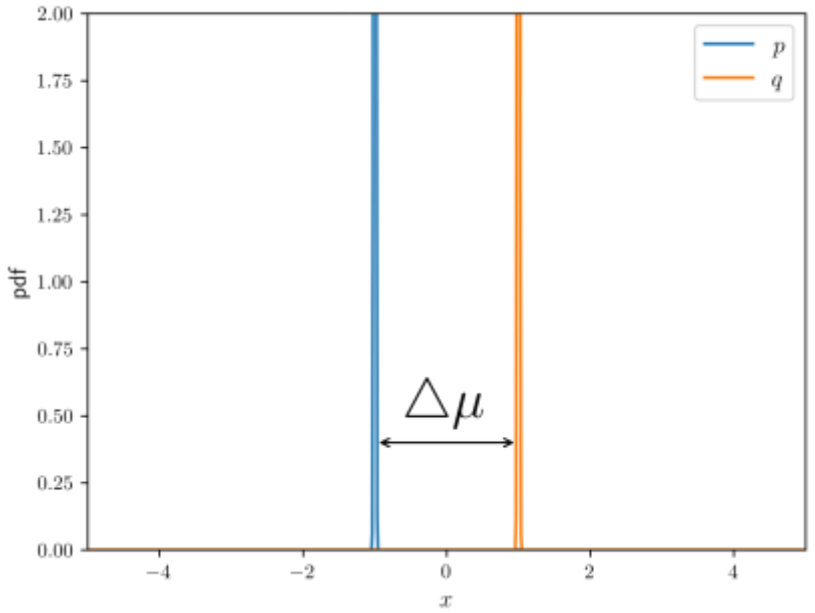


Wasserstein Distance

- when p and q are delta functions:

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





Q x_q

P

x_p

moving plan γ

All possible plan Π

A “moving plan” is a matrix

The value of the element is the amount of earth from one position to another.

Average distance of a plan γ:

$$B(\gamma) = \sum_{x_p, x_q} \gamma(x_p, x_q) \|x_p - x_q\|$$

Earth Mover’s Distance:

$$W(P, Q) = \min_{\gamma \in \Pi} B(\gamma)$$

The best plan

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

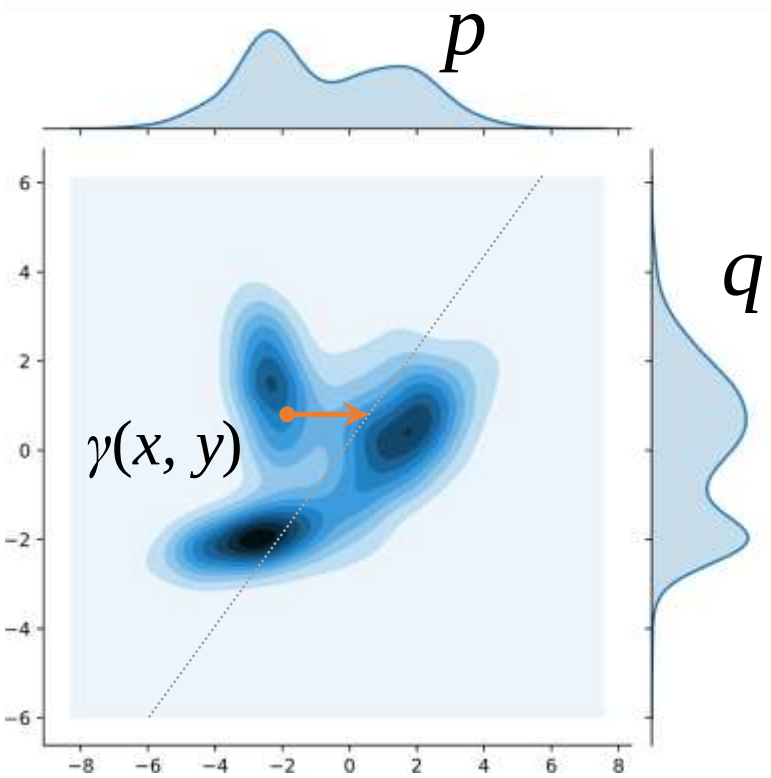
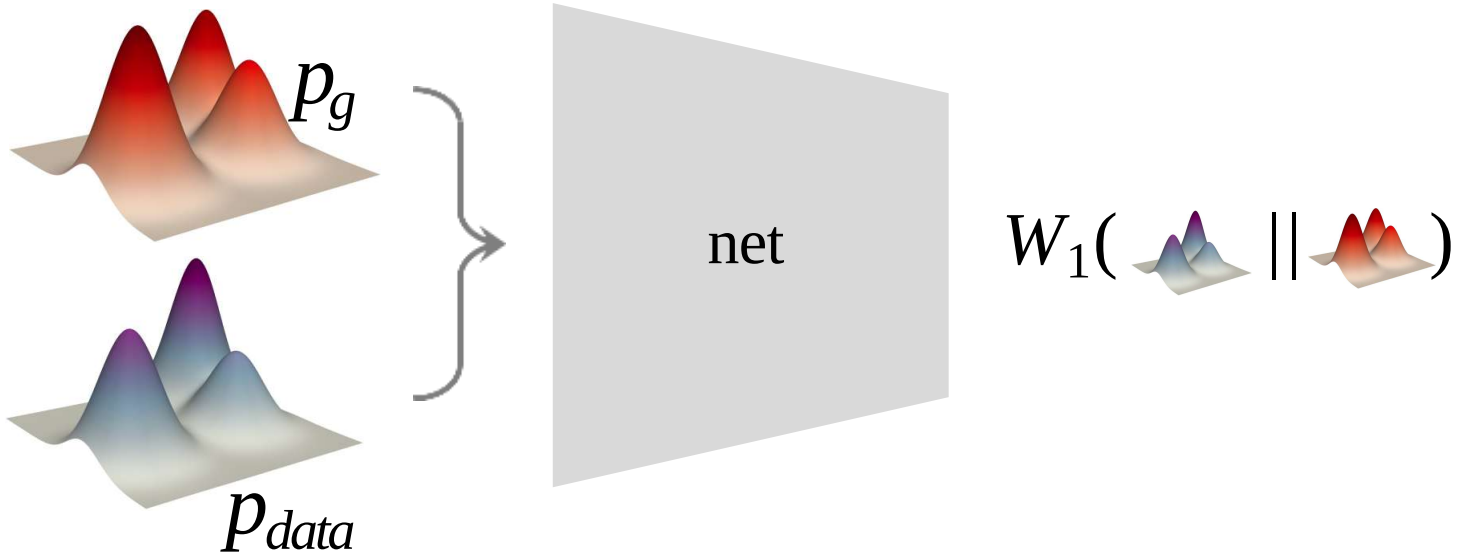


Figure: https://en.wikipedia.org/wiki/Wasserstein_metric

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:

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

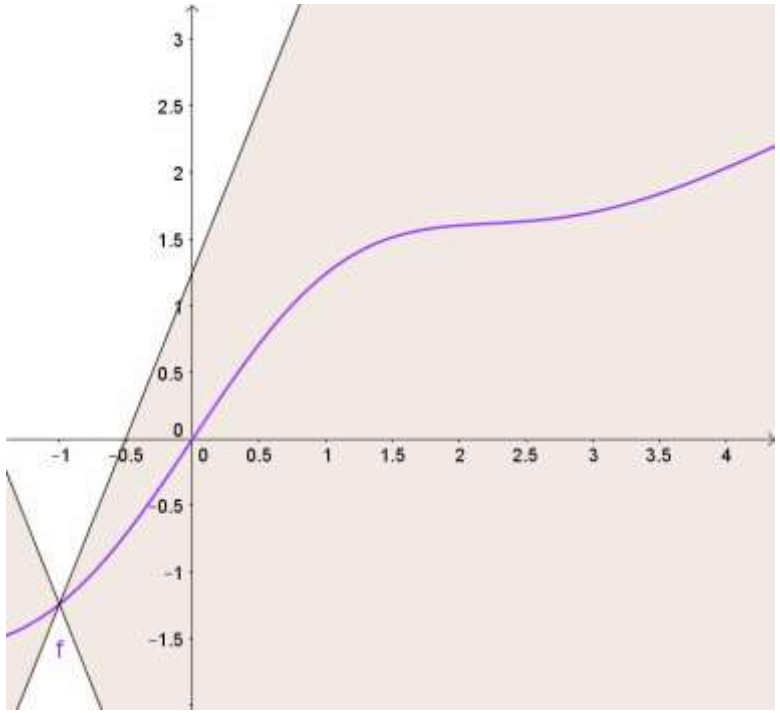


Figure from: https://en.wikipedia.org/wiki/Lipschitz_continuity

W-GAN optimizes for Wasserstein Distance

- Kantorovich-Rubinstein duality:

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

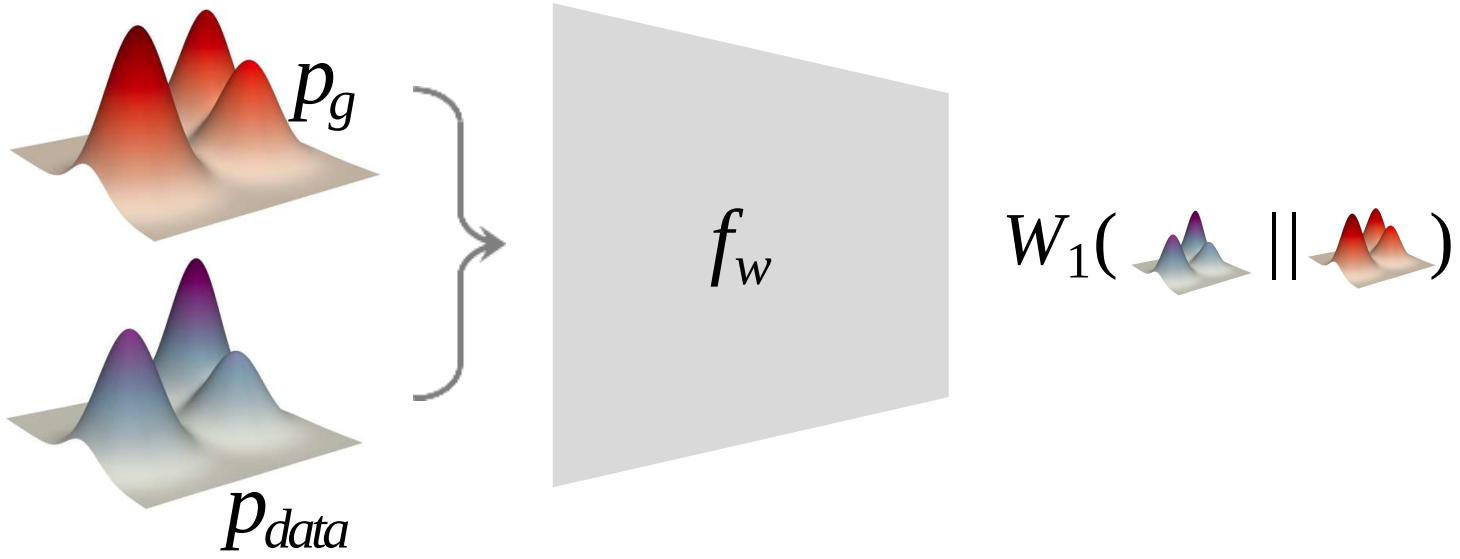
K-Lipschitz continuity:

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

W-GAN optimizes for Wasserstein Distance

- W-GAN's objective function:

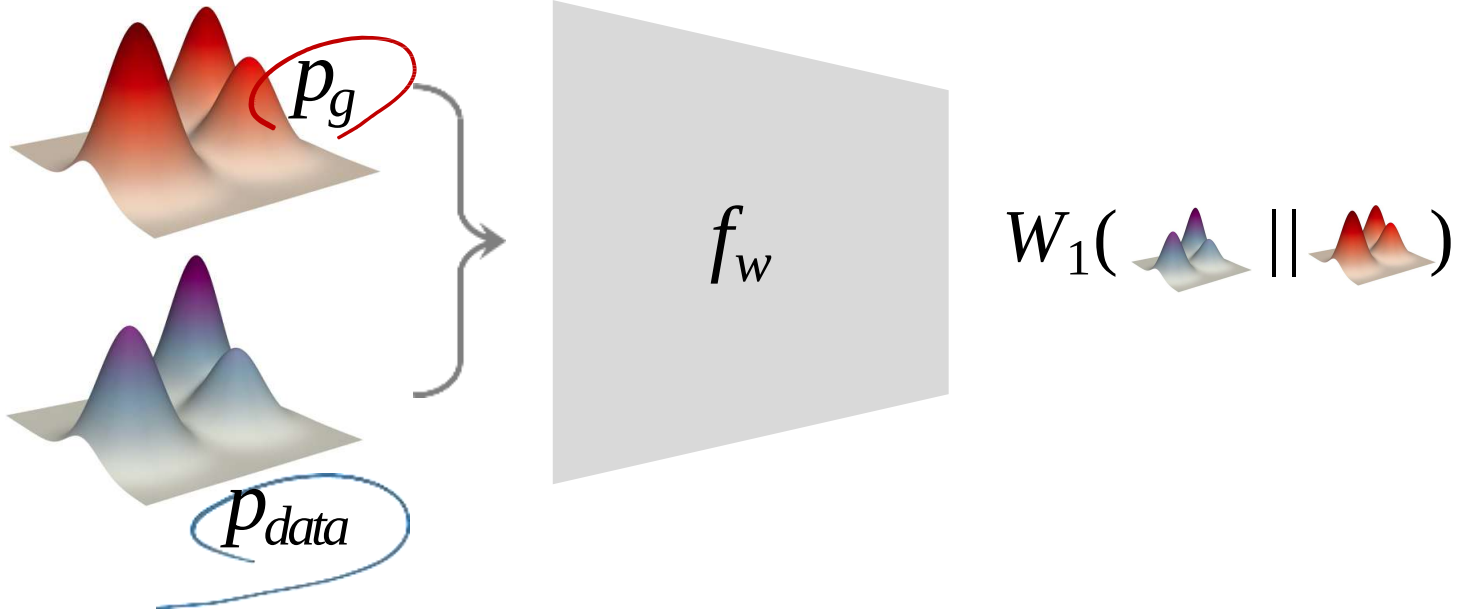
$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_{\text{data}}}[f_w(x)] - \mathbb{E}_{x \sim p_g}[f_w(x)]$$



W-GAN optimizes for Wasserstein Distance

- W-GAN's objective function:

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

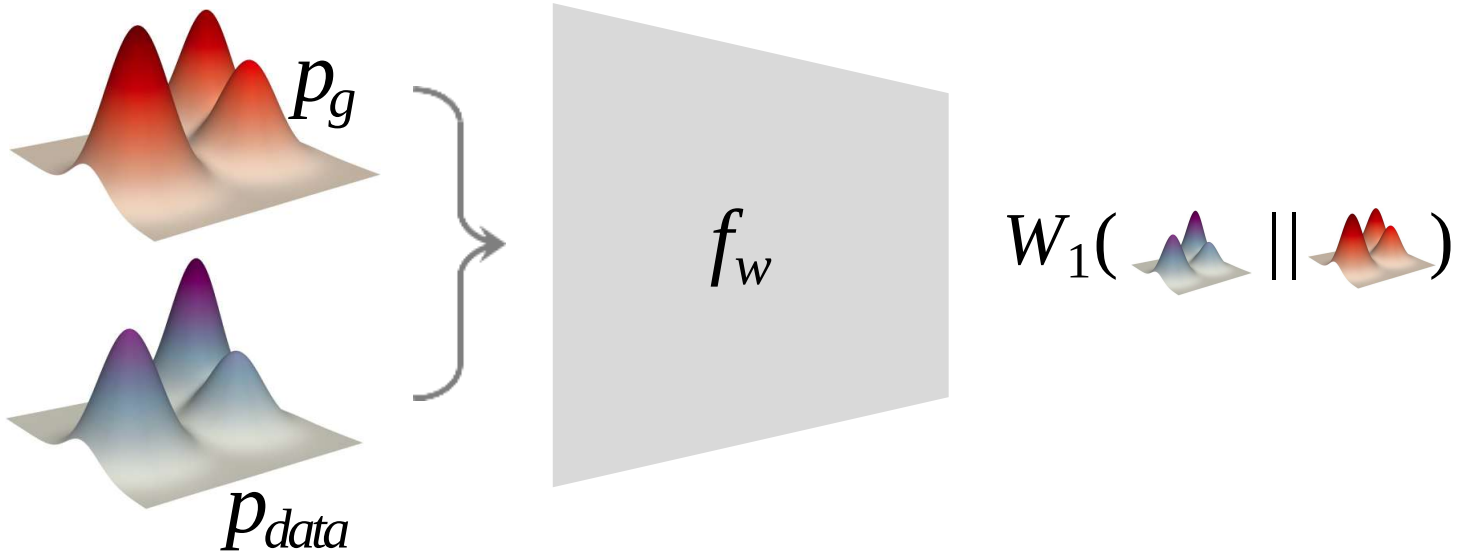


W-GAN optimizes for Wasserstein Distance

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

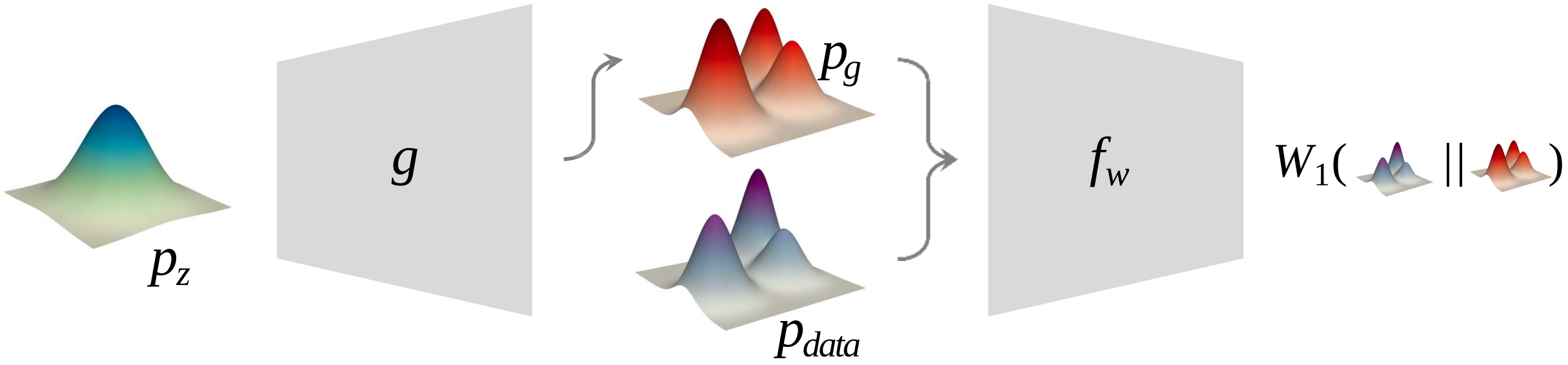
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

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

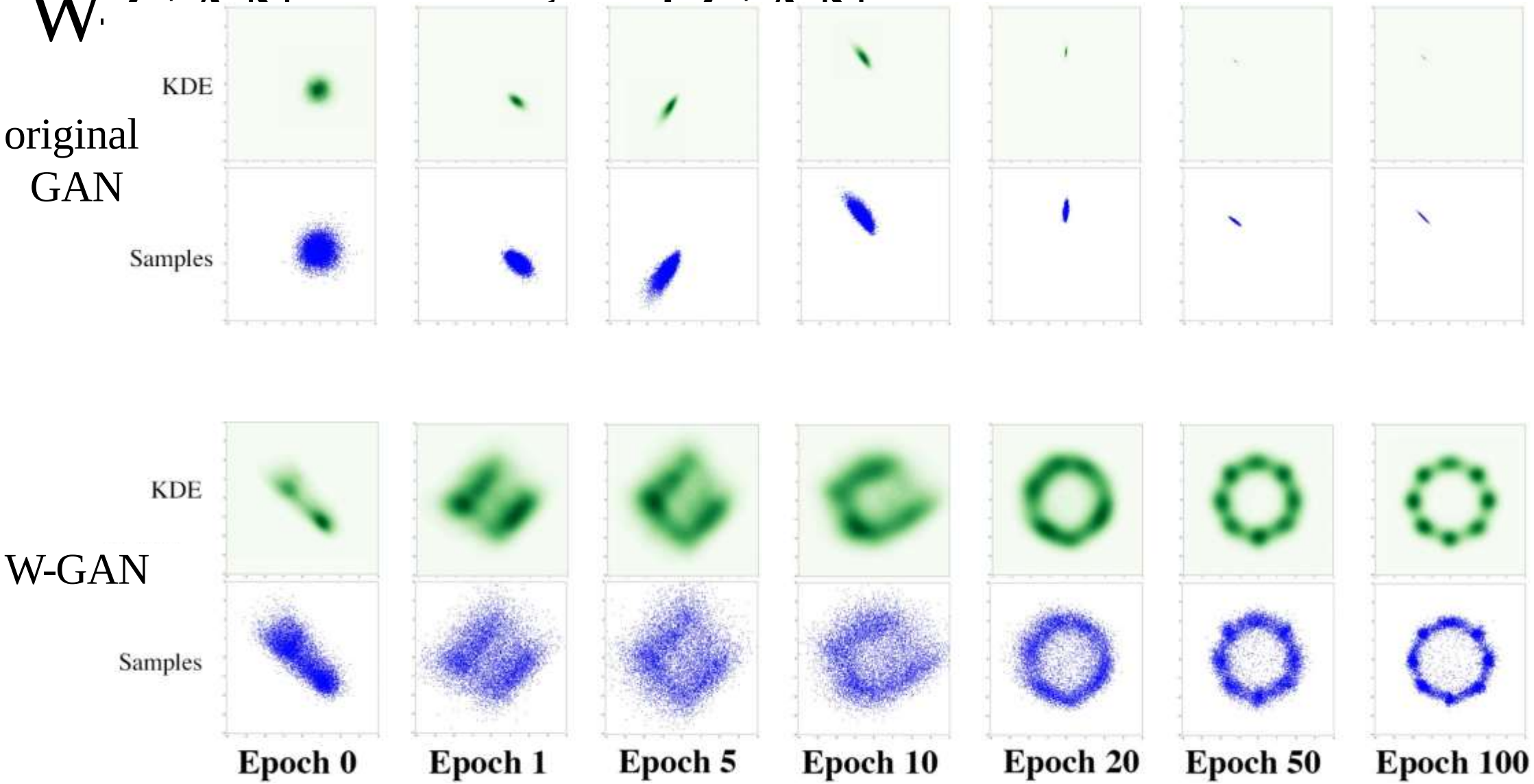
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)})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



[Martin Arjovsky, et al., arXiv, 2017]

WGAN

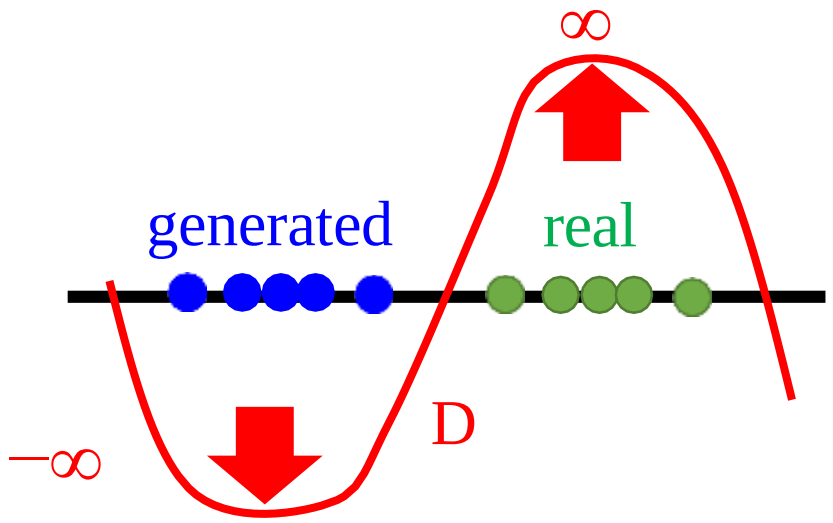
Evaluate wasserstein distance between P_{data} and P_G

$$V(G, D) = \max_{D \in 1\text{-}Lipschitz} \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)]\}$$

D has to be smooth enough.

Without the constraint, the training of D will not converge.

Keeping the D smooth forces D(x) become ∞ and $-\infty$



WGAN

Weight Clipping [Martin Arjovsky, et al., arXiv, 2017]

Force the parameters w between c and -c After parameter update, if w > c, w = c;
if w < -c, w = -c

Evaluate wasserstein distance between P_{data} and P_G

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

D has to be smooth enough.

How to fulfill this constraint?

Lipschitz Function

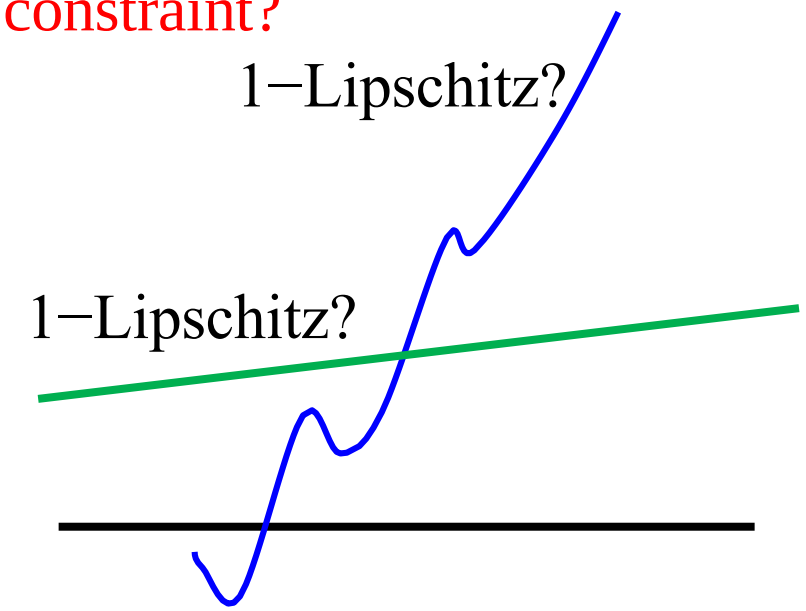
$$\|f(x_1) - f(x_2)\| \leq K \|x_1 - x_2\|$$

Output change

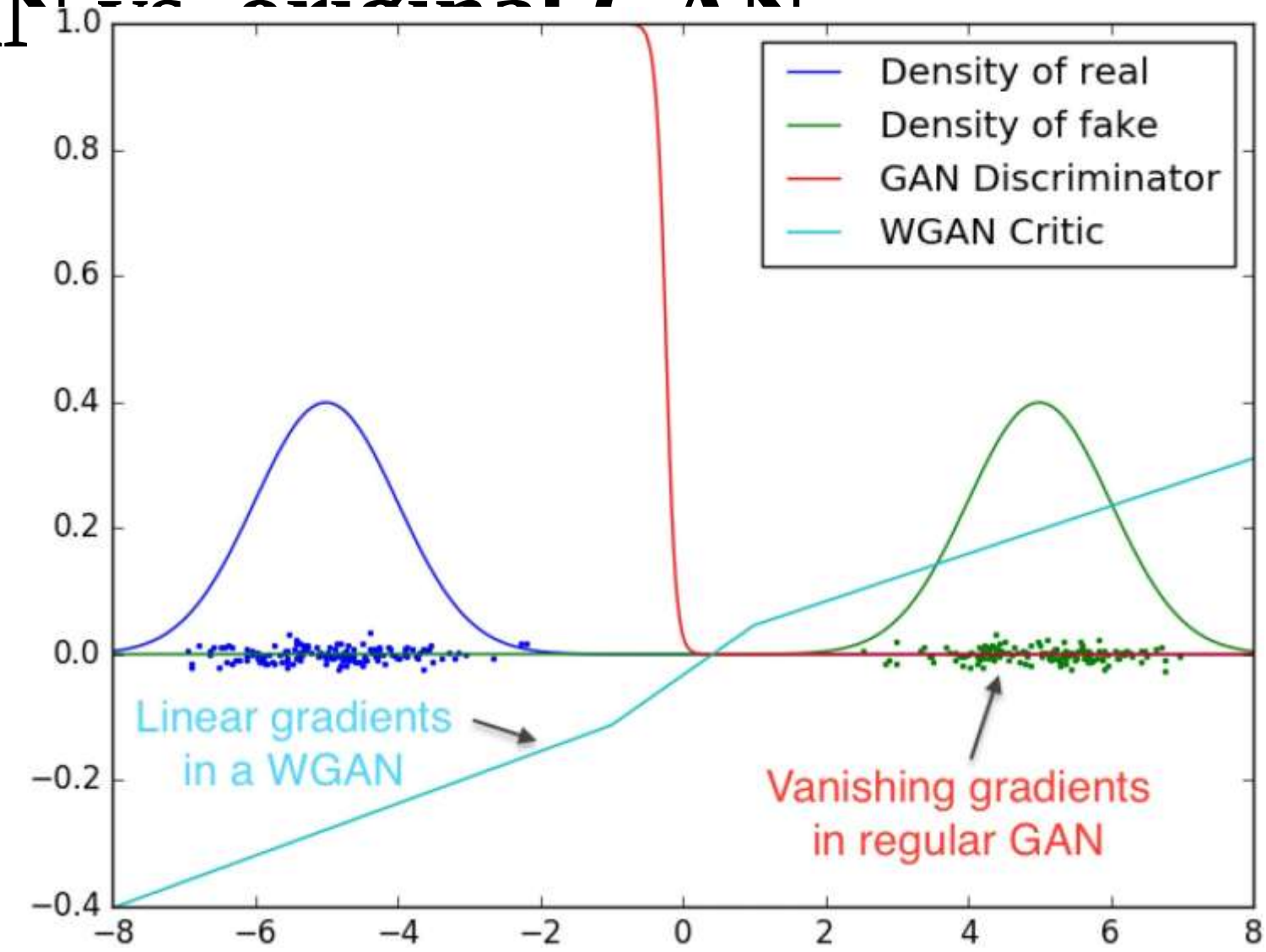
Input change

K=1 for "$1-Lipschitz$"

Do not change fast



W-GAN vs original GAN



Improved WGAN (WGAN-GP)

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

A differentiable function is 1-Lipschitz if and only if it has gradients with norm less than or equal to 1 everywhere.

$$D \in 1-Lipschitz \iff \|\nabla_x D(x)\| \leq 1 \text{ for all x}$$

$$V(G,D) \approx \max_D \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)]$$

~~$-\lambda \int_x max(0, \|\nabla_x D(x)\| - 1)dx\}$~~

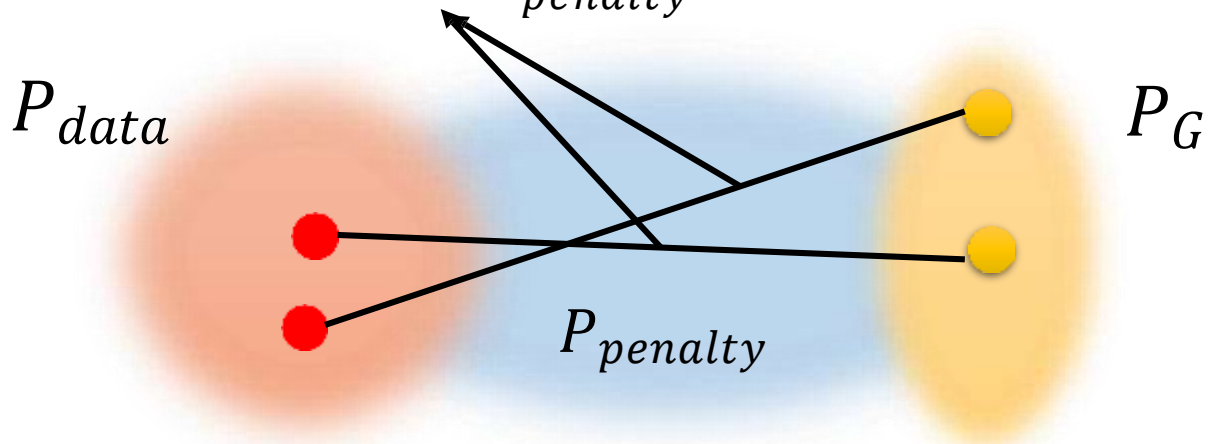
Prefer $\|\nabla_x D(x)\| \leq 1$ for all x

$$-\lambda E_{x \sim P_{penalty}}[max(0, \|\nabla_x D(x)\| - 1)]\}$$

Prefer $\|\nabla_x D(x)\| \leq 1$ for x sampling from $x \sim P_{penalty}$

Improved WGAN (WGAN-GP)

$$V(G,D) \approx \max_{D}\{E_{x\sim P_{data}}\ [D(x)] - E_{x\sim P_G}\ [D(x)]$$
$$-\lambda E_{x\sim P_{penalty}}\ [max(0, \|\nabla_x D(x)\| - 1)]\}$$



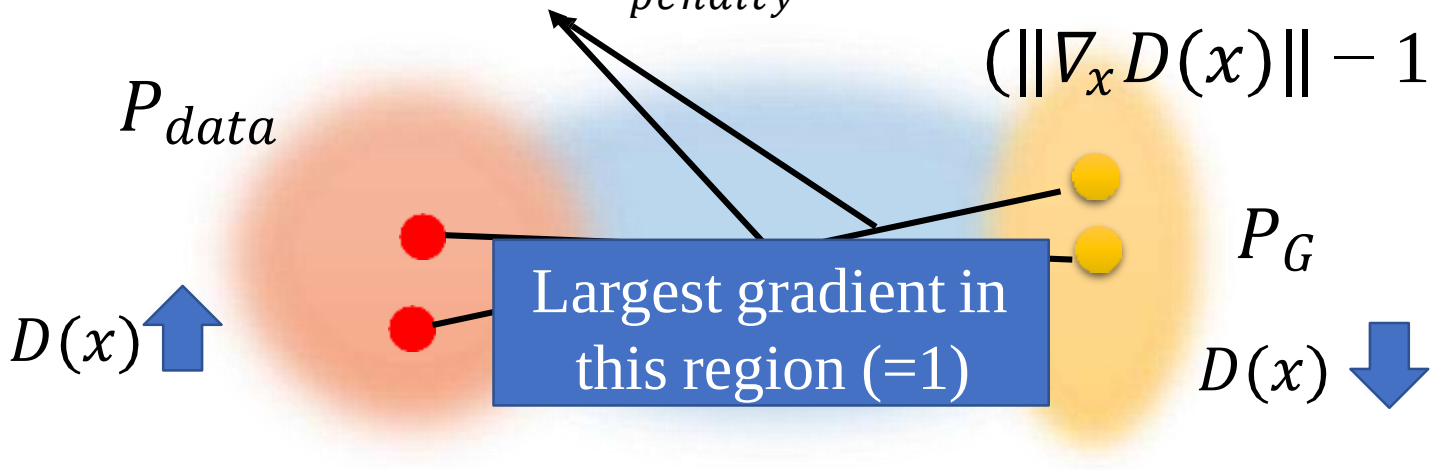
"Given that enforcing the Lipschitz constraint everywhere is intractable, enforcing it ***only along these straight lines*** seems sufficient and experimentally results in good performance."

Only give gradient constraint to the region between P_{data} and P_G because they influence how P_G moves to P_{data}

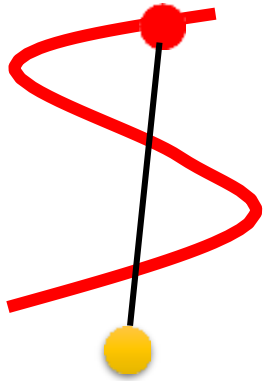
Improved WGAN (WGAN-GP)

$$V(G, D) \approx \max_{D}\{E_{x\sim P_{data}}[D(x)] - E_{x\sim P_G}[D(x)] - \lambda E_{x\sim P_{penalty}}[\cancel{max(0, \|\nabla_x D(x)\| - 1)}]\}$$

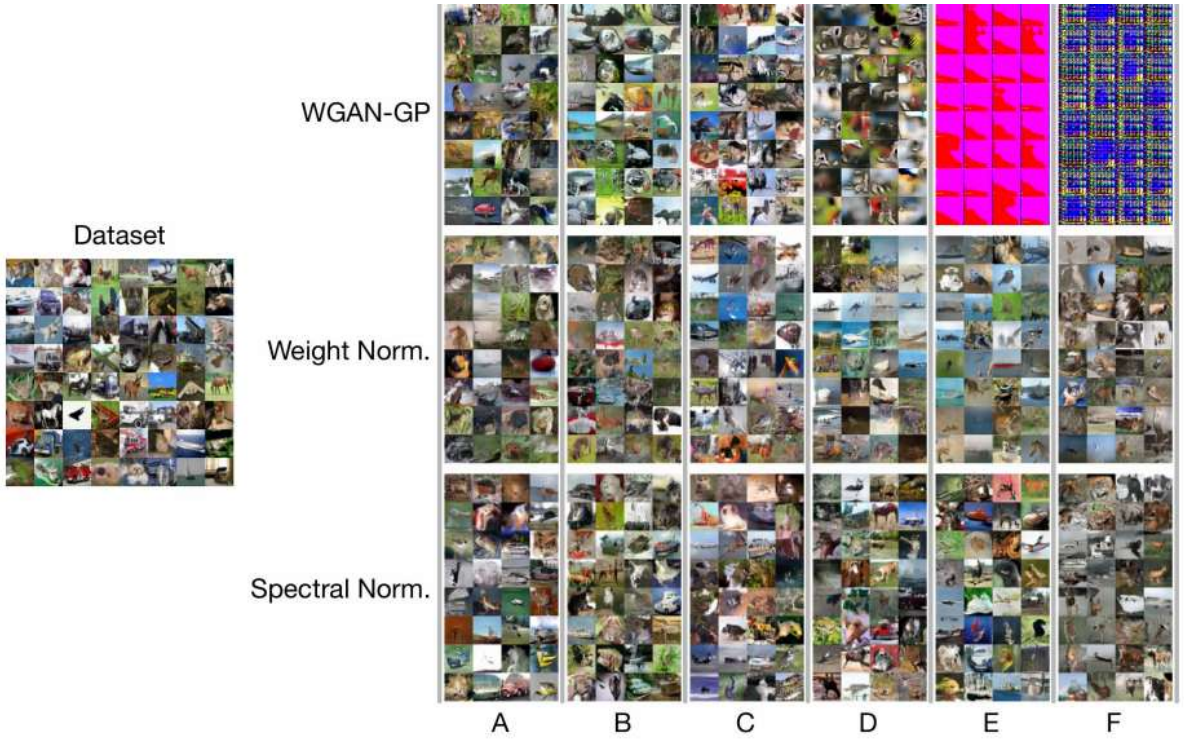
$$(\|\nabla_x D(x)\| - 1)^2$$



"Simply penalizing overly large gradients also works in theory, but experimentally we found that this approach converged faster and to better optima."



Spectrum Norm



Spectral Normalization → Keep gradient norm smaller than 1 everywhere [Miyato, et al., ICLR, 2018]

$$\|f\|_{\text{Lip}} \leq \|(\boldsymbol{h}_L \mapsto W^{L+1}\boldsymbol{h}_L)\|_{\text{Lip}} \cdot \|a_L\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_{L-1} \mapsto W^{L}\boldsymbol{h}_{L-1})\|_{\text{Lip}}$$

$$\cdots \|a_1\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_0 \mapsto W^{1}\boldsymbol{h}_0)\|_{\text{Lip}} = \prod_{l=1}^{L+1} \|(\boldsymbol{h}_{l-1} \mapsto W^{l}\boldsymbol{h}_{l-1})\|_{\text{Lip}} = \prod_{l=1}^{L+1} \sigma(W^l).$$

W-GAN in Short

For mathematicians:

- Wasserstein distance, instead of JS divergence

For engineers:

- remove logarithms
- clip weights

Wasserstein distance

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