Computer Vision:
Generative models

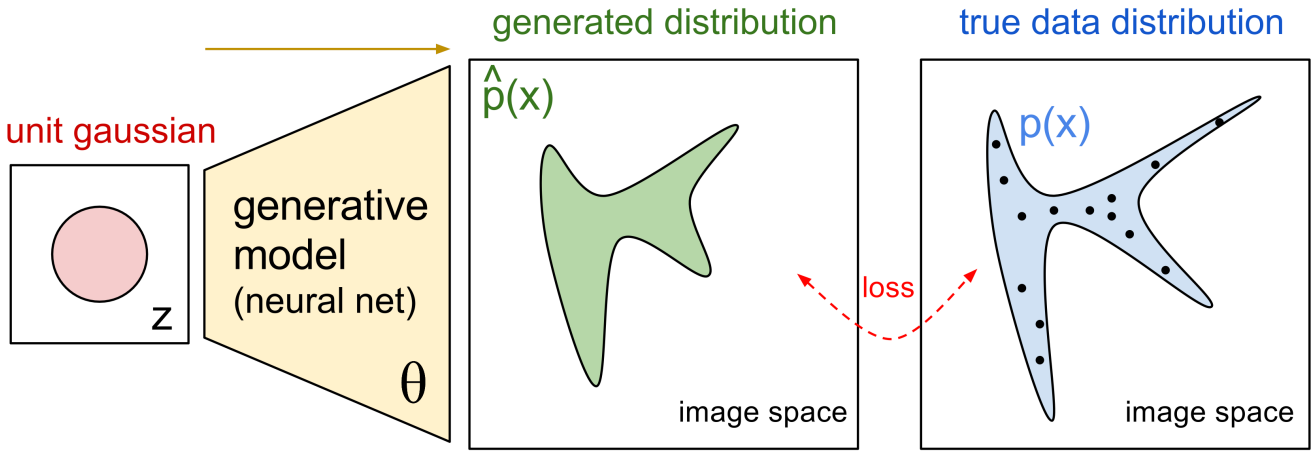
Siheng Chen 陈思衡

Outline

- What are generative models?
- Overview of generative models
- Implicit density models
- GAN framework
- Why GAN is hard to train?
- Extensions to GAN
- Normalization Flow
- Diffusion Models

What are generative models?

Target: train a network that **models a distribution**, such as a distribution over images



Criterion: advocate/penalize samples within the blue/white region.

Then?

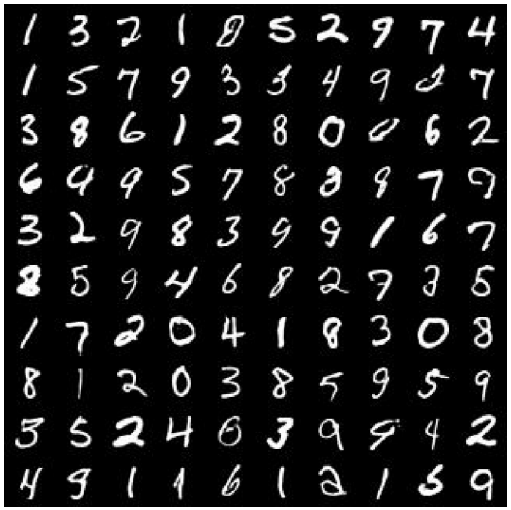
What are generative models?

Sample z and generate (fake) data samples!

Examples:

GAN

DALL·E 2



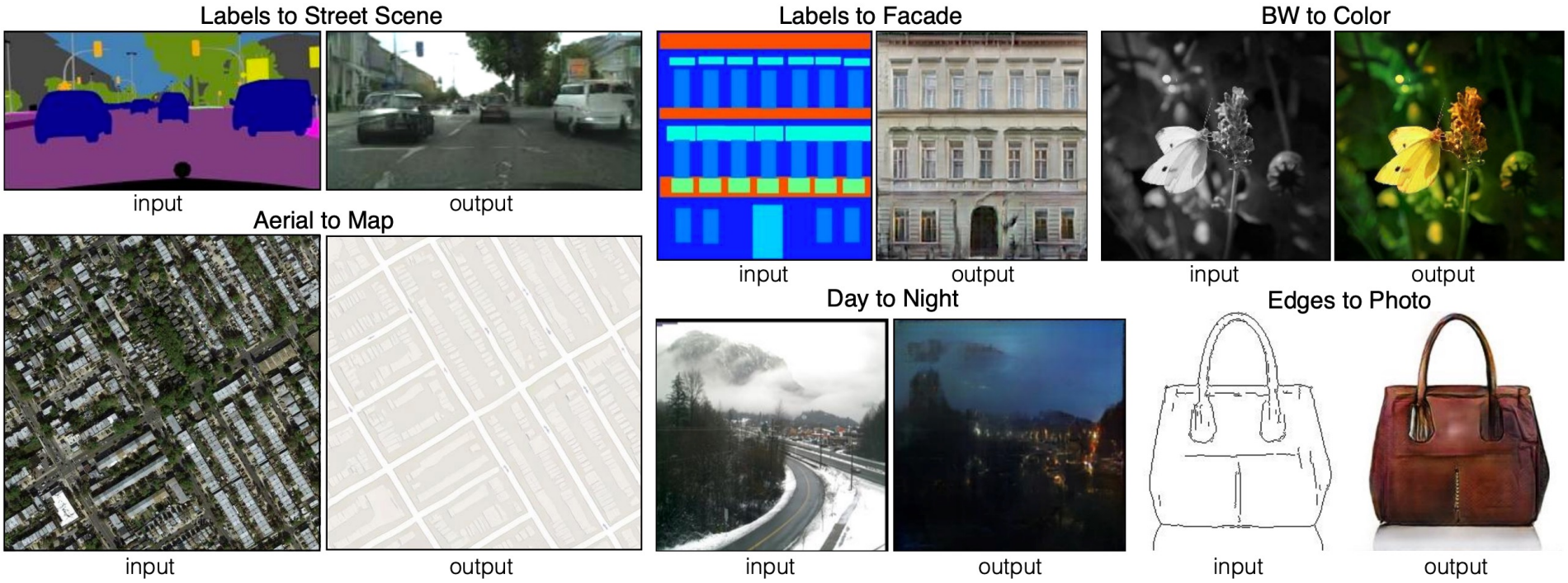
2009

2014

2022

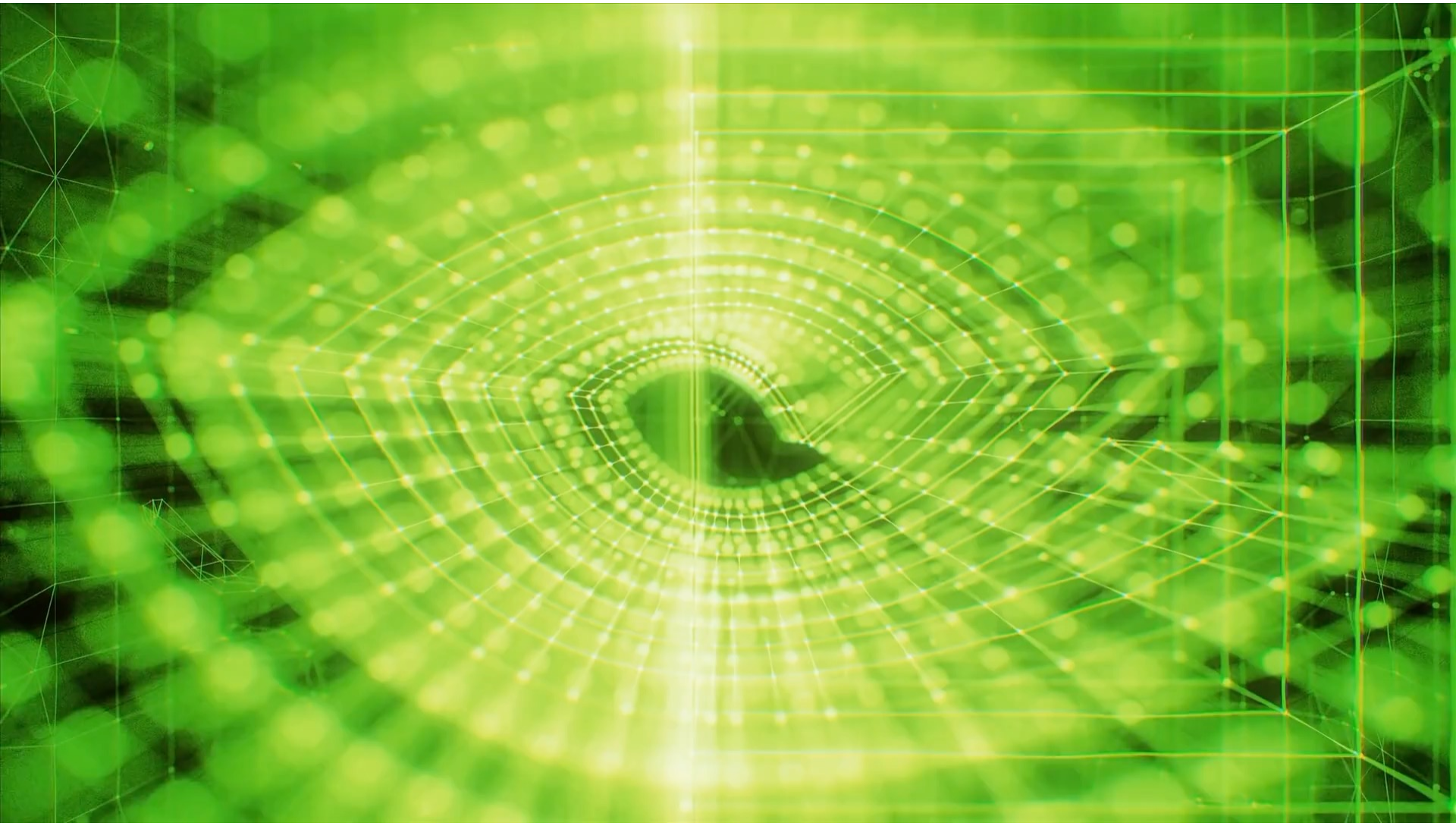
Ian Goodfellow et al. “Generative Adversarial Nets”, NIPS 2014.
Ramesh et al., “Hierarchical Text-Conditional Image Generation with CLIP Latents”, arXiv 2022.

What are generative models?



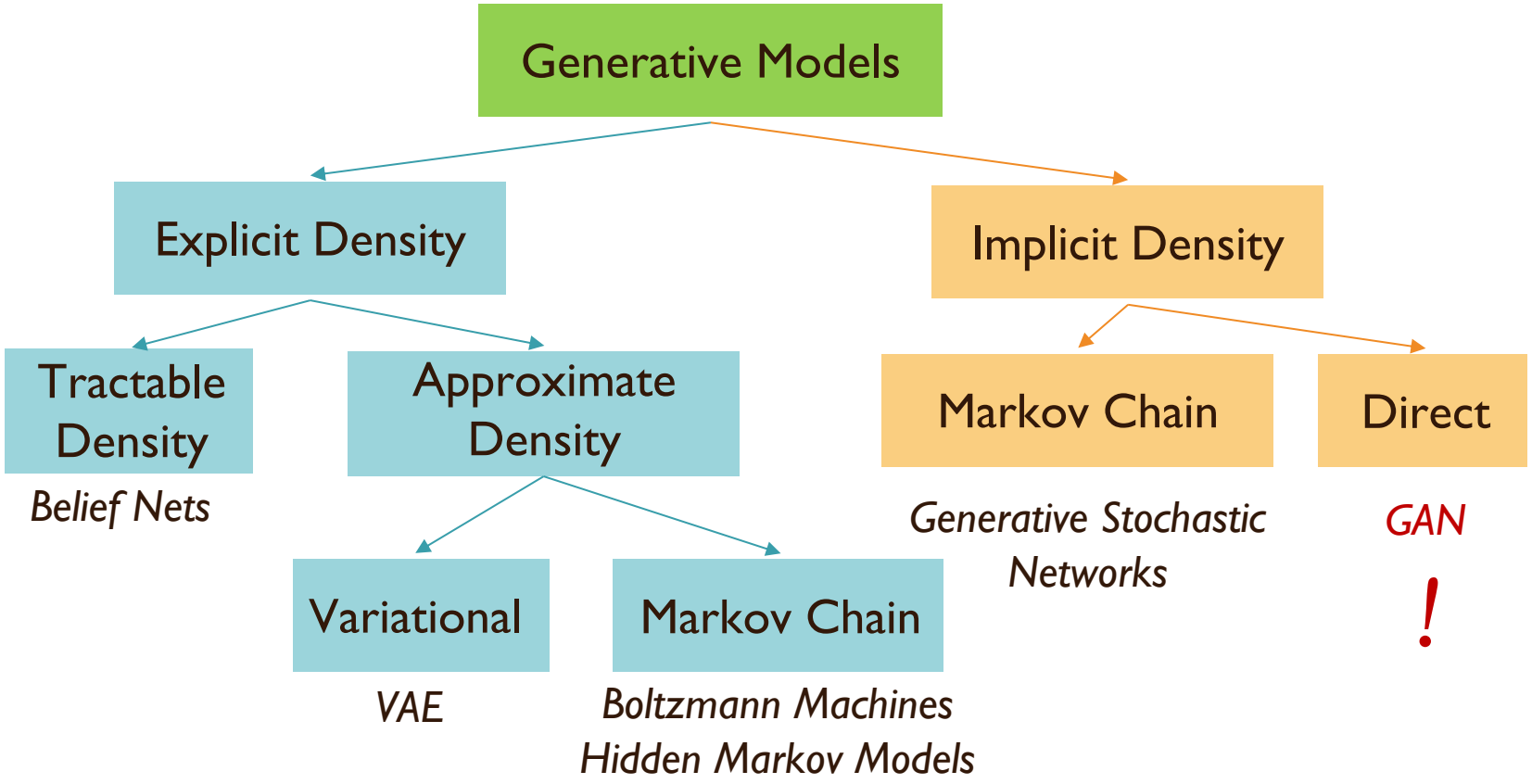
Phillip Isola Jun-Yan Zhu Tinghui Zhou Alexei A. Efros, Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

What are generative models?



What are generative models?

Overview of generative models



- **Explicit density models**: Model data directly from a probability distribution. We explicitly define the probability and solve for it.
- **Implicit density models**: Learn to sample from a probability distribution without defining what that distribution is.

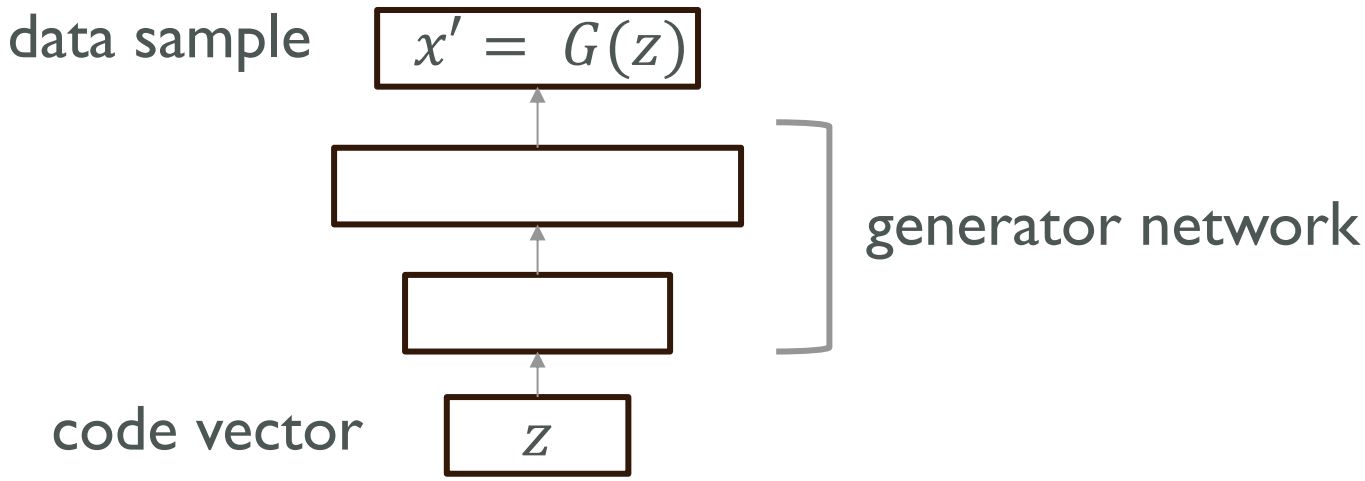
Outline

- What are generative models?
- Overview of generative models
- Implicit density models
- GAN framework
- Why GAN is hard to train?
- Extensions to GAN
- Normalization Flow
- Diffusion Models

Implicit density models

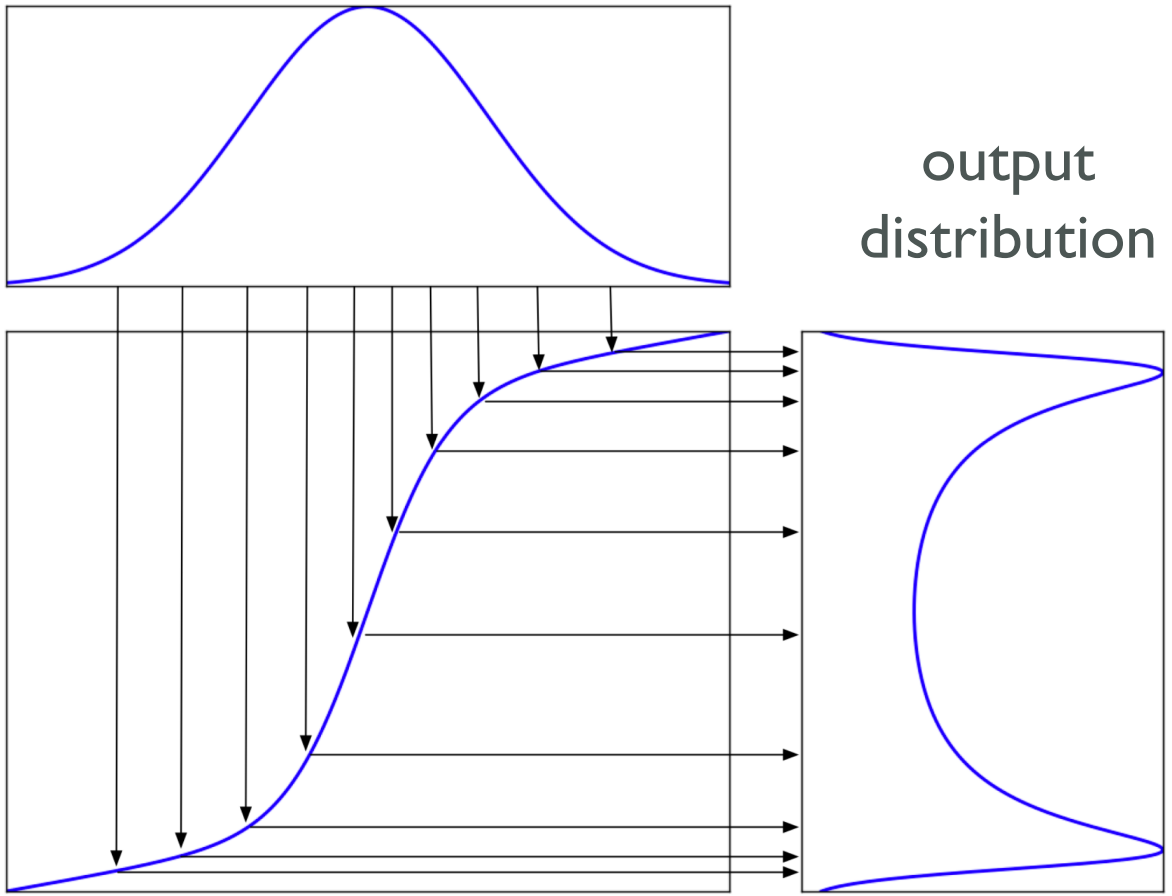
also known as *implicit generative models*.

- Implicit density models implicitly define a probability distribution.
- Start by sampling the code vector z from a fixed, simple distribution (e.g. spherical Gaussian).
- The generator network computes a differentiable function G mapping z to an x' in data space.



Implicit density models

A one-dimensional example:



example from *https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf*

Implicit density models

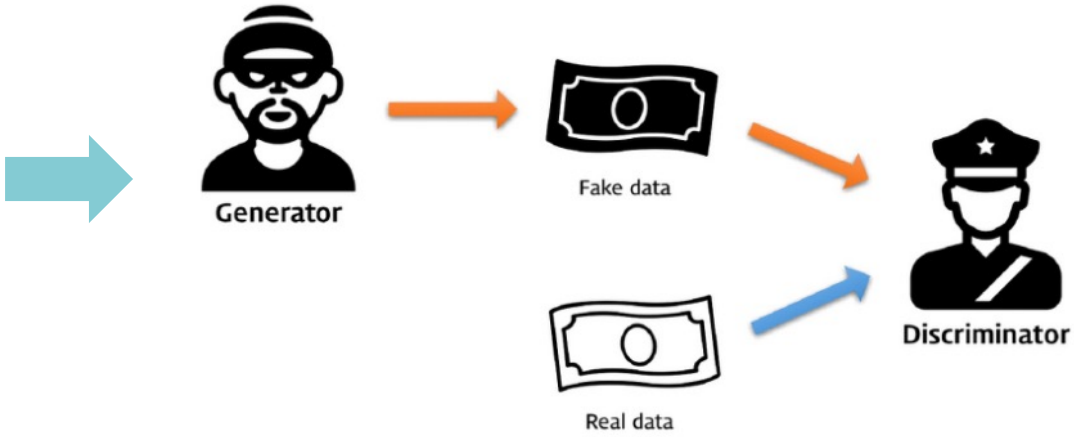
Advantage of implicit density models

If you have some criterion for evaluating the quality of samples, then you can compute its gradient with respect to the network parameters, and update the network's parameters to make the sample better.

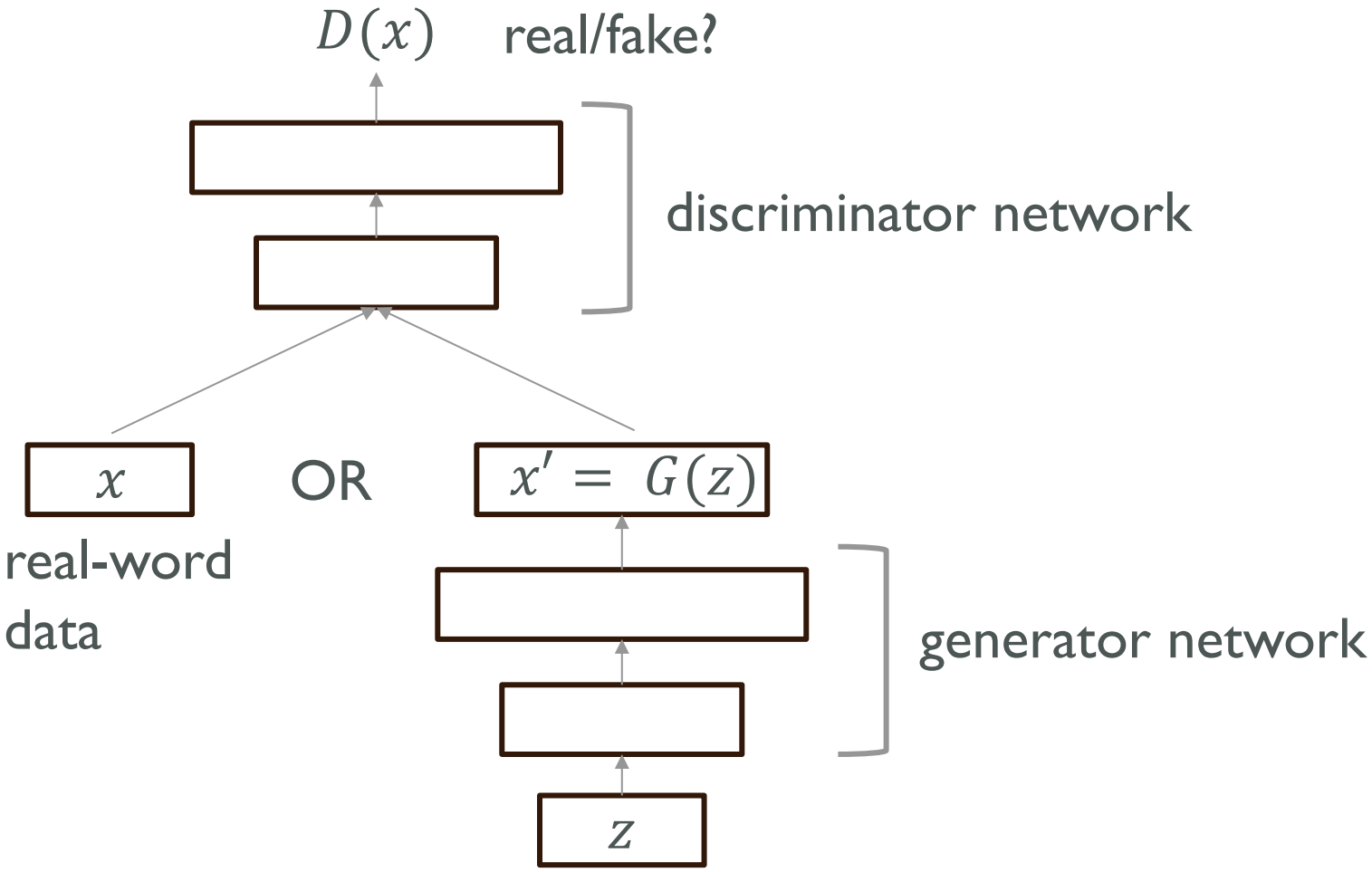
The idea behind *Generative Adversarial Networks (GAN)!*

generator network + discriminator network

GAN framework



GAN framework

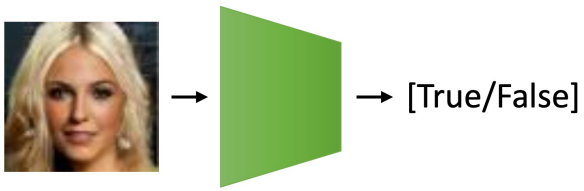


GAN framework

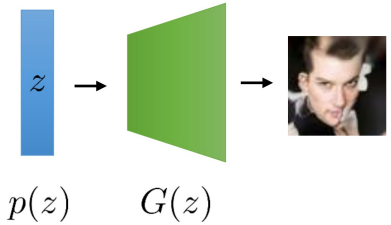
A minimax game

1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ random initialization!
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. update $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$ (1 SGD step)
5. use $-\log D(x)$ as "loss" to update $G(z)$ (1 SGD step)

(in reality there are a variety of different losses, but similar idea…)



discriminator



generator

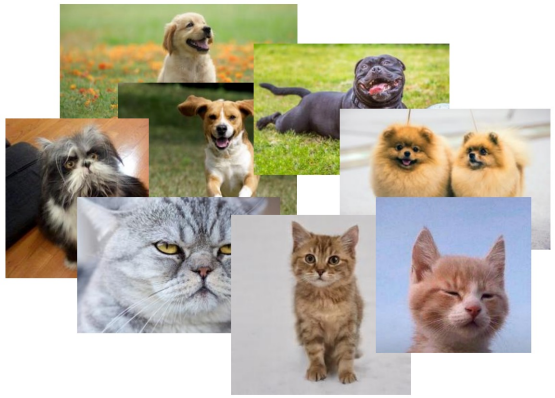
what does $G(z)$ want to do?

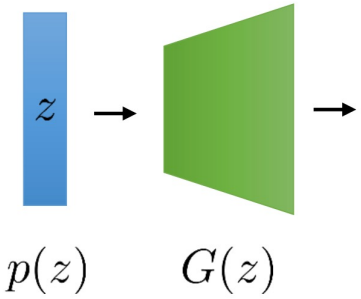
make $D(x) = 0.5$ for all generated x

- Generate images that look realistic
- Generate all possible realistic images

GAN framework

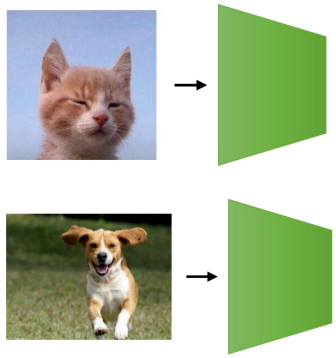
Why do GANs learn distributions?





very realistic, but only dogs

The generator will do better if it not only generates realistic pictures, but if it generates all realistic pictures



True = 1.0!

True = 0.25!

GAN framework

A minimax game

1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ random initialization!
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. update $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$ (1 SGD step)
5. use $-\log D(x)$ as "loss" to update $G(z)$ (1 SGD step)

(in reality there are a variety of different losses, but similar idea…)

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))]$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} \log D(x_i) \quad x_i \in \mathcal{D}_T$$

$$\approx \frac{1}{N}\sum_{j=1}^{N} \log(1 - D(x_j))$$

$x_j = G(z_j)$ random numbers

GAN framework

A minimax game

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1-D(G(z)))]$$

$$\min_\theta \max_\phi V(\theta,\phi) = E_{x\sim p_{\text{data}}(x)}[\log D_\phi(x)] + E_{z\sim p(z)}[\log(1-D_\phi(G_\theta(z)))]$$

$$\phi \leftarrow \phi + \alpha\nabla_\phi V(\theta,\phi) \approx \nabla_\phi \left(\frac{1}{N}\sum_{i=1}^{N} \log D_\phi(x_i) + \frac{1}{N}\sum_{j=1}^{N} \log(1-D_\phi(x_j)) \right)$$

$x_i \in \mathcal{D}_T$ $\quad$ $x_j = G(z_j)$ cross-entropy loss!

$$\theta \leftarrow \theta - \alpha\nabla_\theta V(\theta,\phi) \approx \nabla_\theta \left(\frac{1}{N}\sum_{j=1}^{N} \log(1-D_\phi(G_\theta(z_j))) \right)$$

random numbers

GAN framework

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:

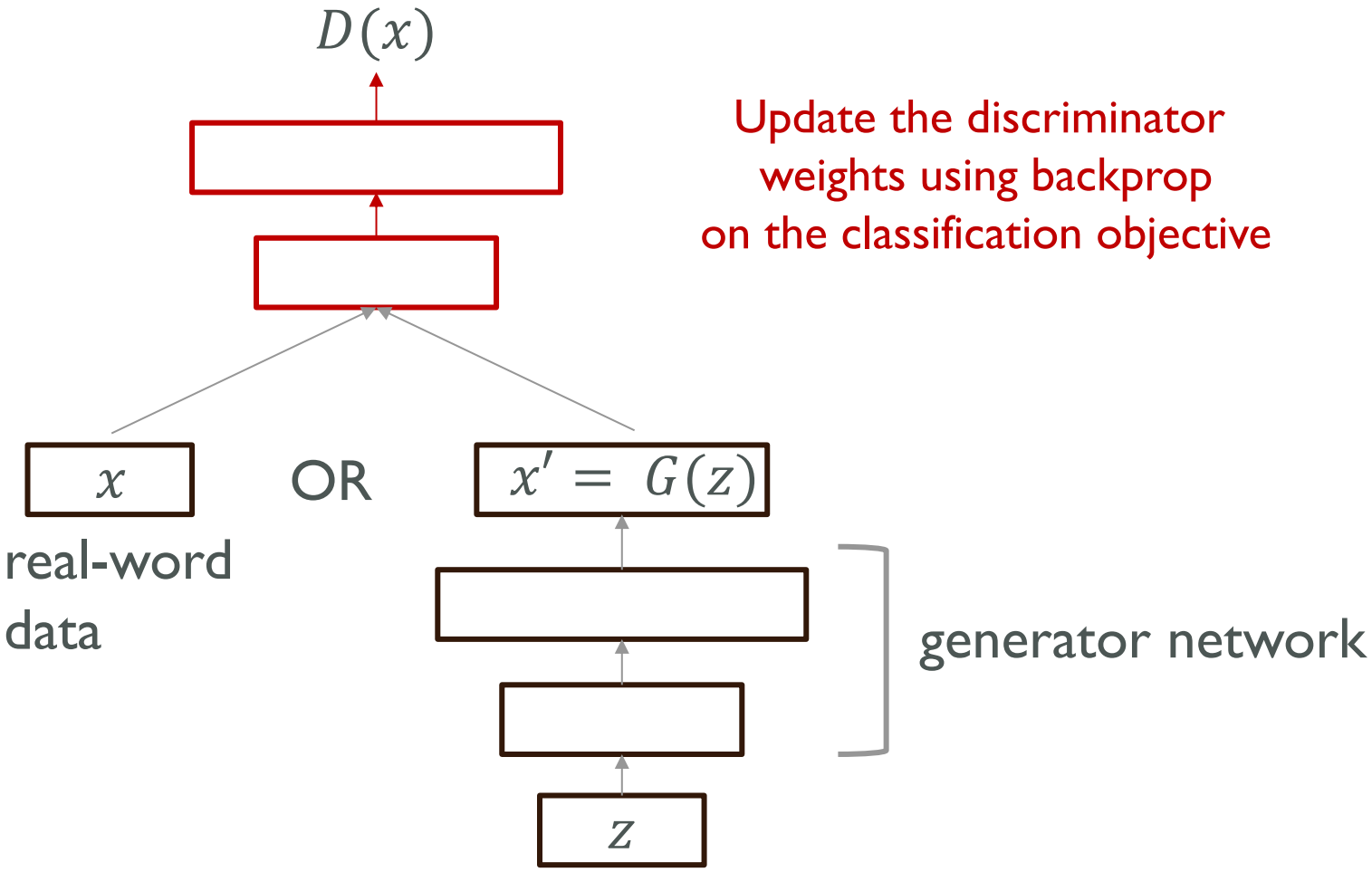
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

GAN framework

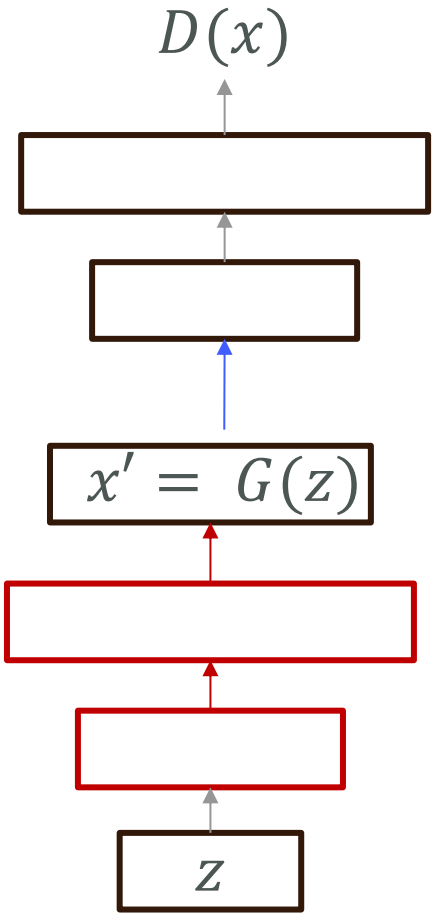
Training the discriminator



Update the discriminator weights using backprop on the classification objective

GAN framework

Training the generator



backprop the derivatives,
but don't modify the
discriminator weights

flip the sign
of the derivatives

update the generator
weights using backprop

GAN framework

Theoretical aspect

$$\min_{\theta} \max_{\phi} V(\theta, \phi) = E_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] + E_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

$\mathcal{D}_T = \{x_i \sim p(x)\}$ $\mathcal{D}_F = \{x_j \sim q(x)\}$

$$D^\star = \arg\max_D E_p[\log D(x)] + E_q[\log(1 - D(x))]$$

$$\nabla_D = E_p\left[\frac{1}{D(x)}\right] - E_q\left[\frac{1}{1 - D(x)}\right] = 0$$

plug in $D^\star(x) = \frac{p(x)}{p(x) + q(x)}$ optimal discriminator

$$\sum_x \cancel{p(x)} \frac{p(x) + q(x)}{\cancel{p(x)}} - \sum_x \cancel{q(x)} \frac{p(x) + q(x)}{\cancel{q(x)}} = 0$$

Optimal discriminator

$$D_G^\star(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

$$x = G(z) \quad z \sim p(z)$$

$$V(D_G^\star, G) =$$

$$E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log(p_{\text{data}}(x) + p_G(x))] + E_{p_G(x)}[\log p_G(x) - \log(p_{\text{data}}(x) + p_G(x))]$$

GAN framework

Theoretical aspect

$$V(D_G^\star, G) =$$
$$E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log(p_{\text{data}}(x) + p_G(x))] + E_{p_G(x)}[\log p_G(x) - \log(p_{\text{data}}(x) + p_G(x))]$$

$$\text{let } q(x) = \frac{p_{\text{data}}(x) + p_G(x)}{2}$$

$$V(D_G^\star, G) = \underbrace{E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log q(x)]}_{D_{\text{KL}}(p_{\text{data}} \| q(x))} + \underbrace{E_{p_G(x)}[\log p_G(x) - \log q(x)]}_{D_{\text{KL}}(p_G \| q(x))} - \log 4$$

$$= D_{\text{JS}}(p_{\text{data}} \| p_G)$$

Jensen-Shannon divergence

- goes to zero if the distributions match
- symmetric (unlike KL-divergence)

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \qquad \text{JSD}(P \parallel Q) = \frac{1}{2} D(P \parallel M) + \frac{1}{2} D(Q \parallel M) \qquad M = \frac{1}{2}(P + Q)$$

GAN really is trying to match the data distribution!

Why GAN is hard to train?

Problem 1 Hard to achieve Nash equilibrium.

- Two models are trained simultaneously to find a Nash equilibrium to a two-player non-cooperative game.
- However, each model updates its cost independently with no respect to another player in the game.
- Updating the gradient of both models concurrently **cannot guarantee a convergence**.

Why GAN is hard to train?

Problem 1 Hard to achieve Nash equilibrium.

Example: Player one control x to minimize $f_1(x) = xy$

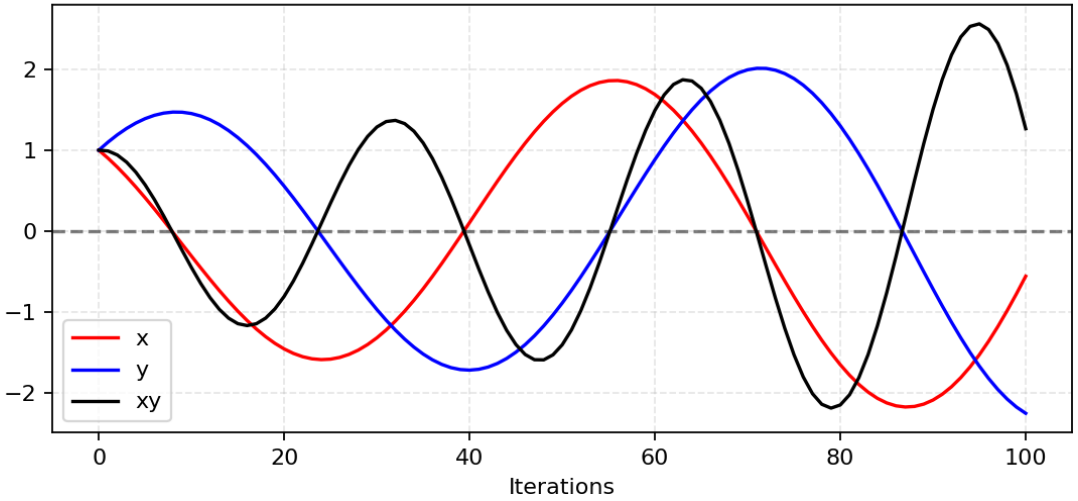
Player two control y to minimize $f_2(y) = -xy$

$$\Rightarrow \frac{\partial f_1}{\partial x} = y, \frac{\partial f_2}{\partial y} = -x$$

$\Rightarrow$ update

$$x = x - \eta \cdot y$$
$$y = y + \eta \cdot x$$

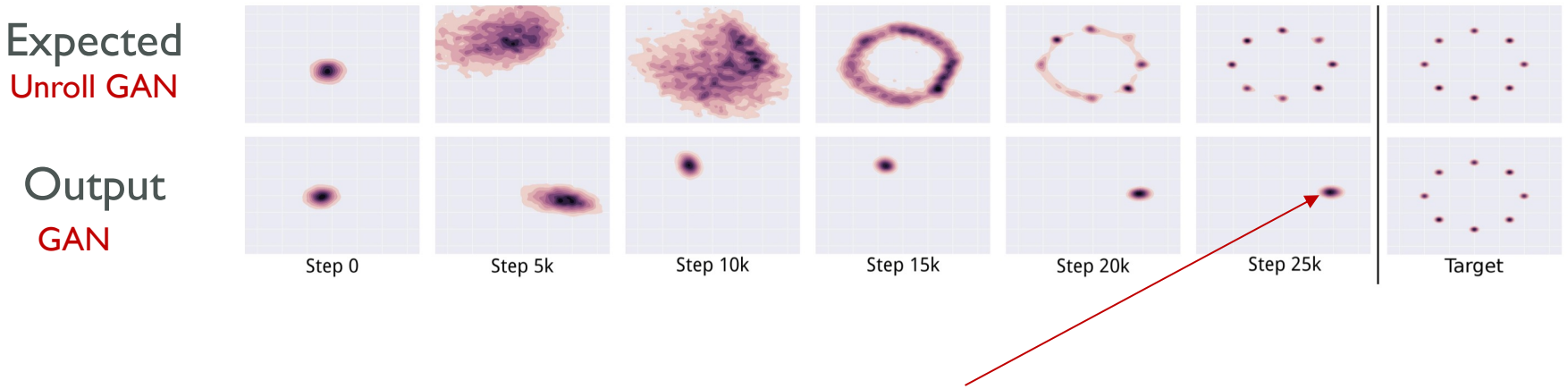
η: learning rate



A simulation of the example for updating x to minimize xy and updating y to minimize $-xy$. The learning rate $\eta = 0.1$. With more iterations, the oscillation grows more and more unstable.

Why GAN is hard to train?

Problem 2 Mode collapse



Generator excels in a subspace but does-not cover entire real distribution

Luke Metz et al. “Unrolled Generative Adversarial networks”. ICLR 2017.

Why GAN is hard to train?

Problem 3 Vanishing gradient

When the discriminator is perfect…

$$D(x) = \begin{cases} 1, & \forall\ x \in p_r \\ 0, & \forall\ x \in p_g \end{cases}$$

Recall $L = \mathbb{E}_{x \sim p_r(x)} \log D(x) + \mathbb{E}_{x \sim p_g(x)} \log\big(1 - D(x)\big)$

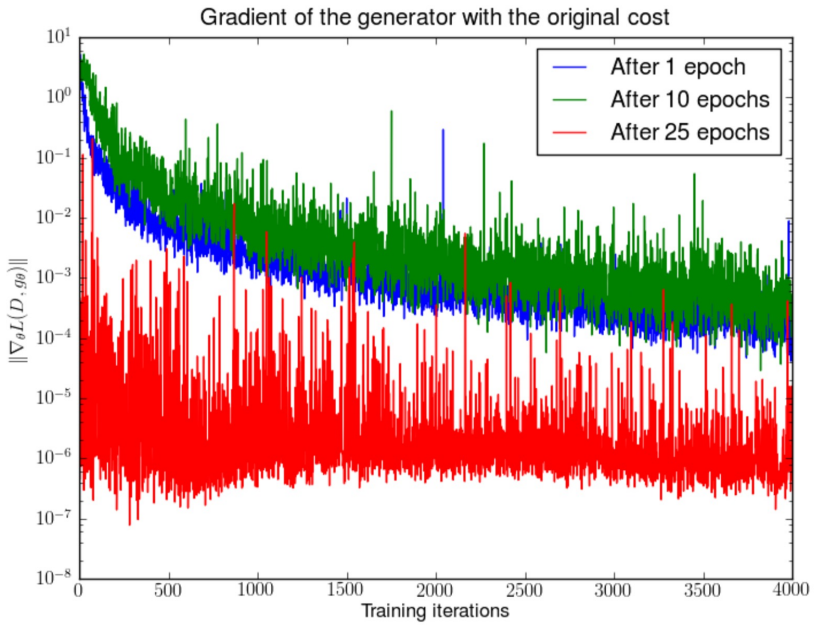
The loss function L falls to zero and we end up with no gradient to update the loss during learning iterations.

Why GAN is hard to train?

Problem 3 Vanishing gradient

When the discriminator is perfect…

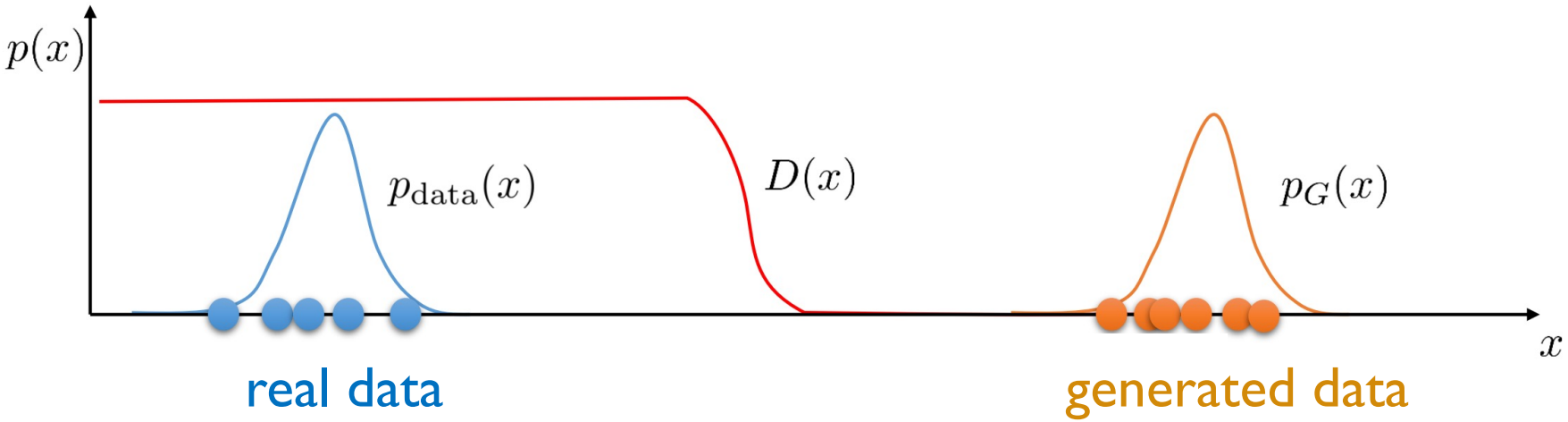
$$D(x) = \begin{cases} 1, & \forall\ x \in p_r \\ 0, & \forall\ x \in p_g \end{cases}$$



1. A DCGAN is trained for 1, 10 and 25 epochs.
2. With the **generator fixed**, a discriminator is trained from scratch and measure the gradients with the original cost function.

We see the gradient norms **decay quickly** (in log scale), in the best case 5 orders of magnitude after 4000 discriminator iterations. (Image source: Arjovsky and Bottou, 2017)

Why GAN is hard to train?



No gradient for the generator!

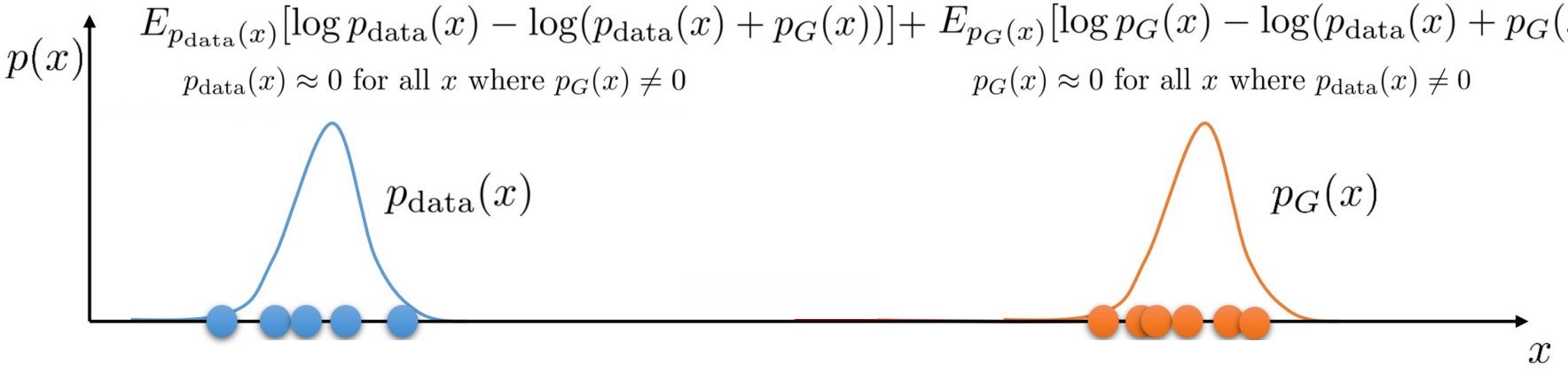
Extensions to GAN

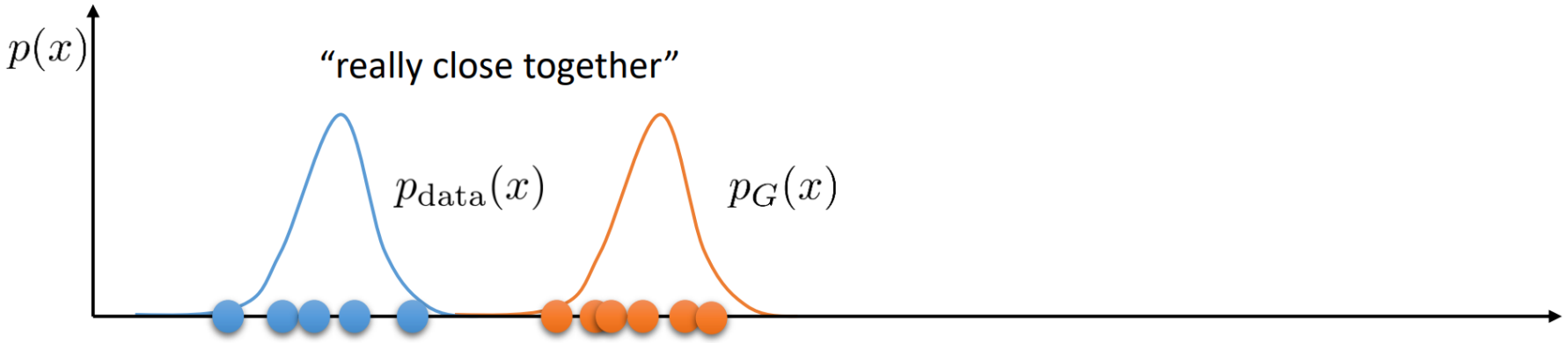
the JS divergence used by the classic GAN doesn't account for "distance"

$$V(D_G^\star, G) = E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log(p_{\text{data}}(x) + p_G(x))] + E_{p_G(x)}[\log p_G(x) - \log(p_{\text{data}}(x) + p_G(x))]$$

$p_{\text{data}}(x) \approx 0$ for all x where $p_G(x) \neq 0$

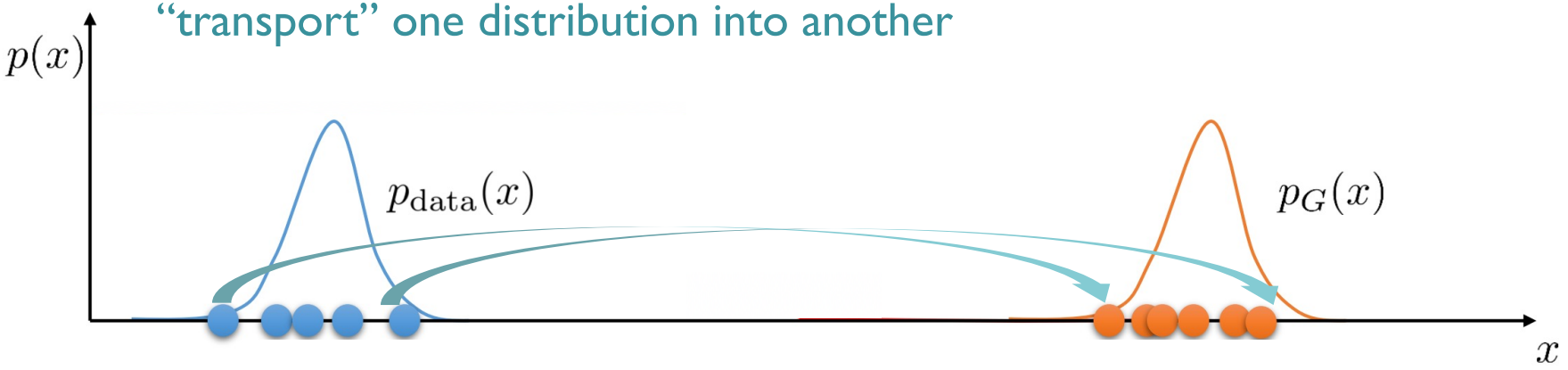
$p_G(x) \approx 0$ for all x where $p_{\text{data}}(x) \neq 0$

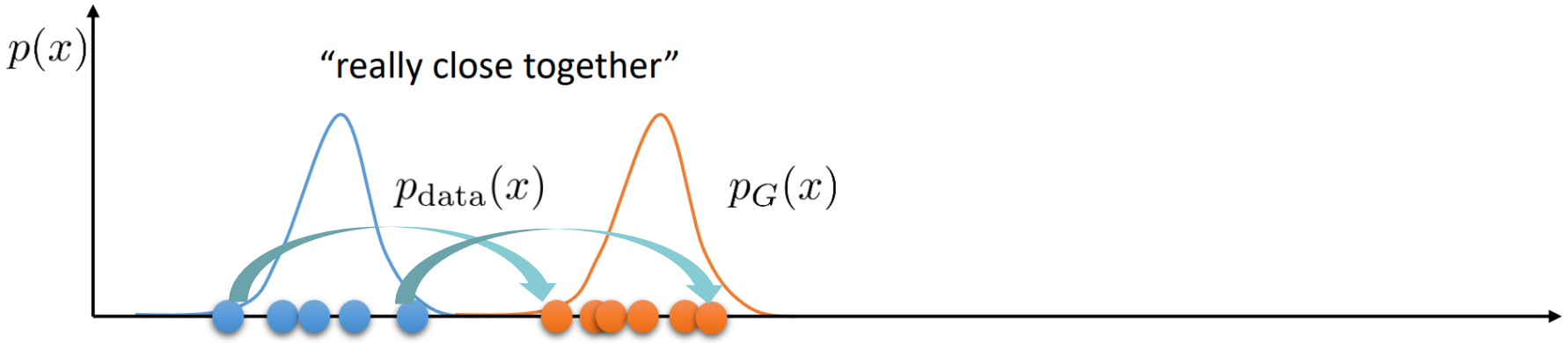




Extensions to GAN

optimal transport ("Earth mover's distance") – how far do you have to go to "transport" one distribution into another





Extensions to GAN

Wasserstein GAN (WGAN) a better distance function

Jensen-Shannon (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m)$$

Wasserstein Distance (Earth-Mover):

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||]$$

where $\Pi(p_r, p_g)$ is the set of all joint distributions $\gamma(x, y)$ whose marginal distributions are respectively p_r and p_g.

- well defined even if p_r and p_g have different support.
- leads to a very simple algorithm using a similar variational trick.

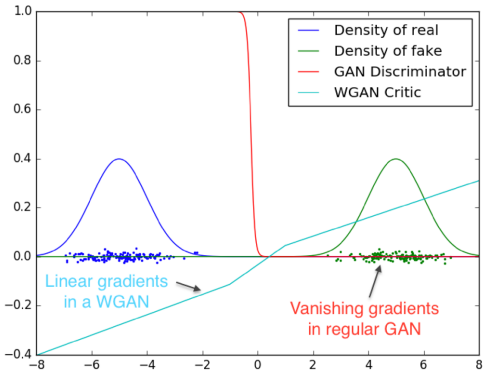
Martin Arjovsky, Soumith Chintala, and L´eon Bottou, Wasserstein GAN, ICML 2017

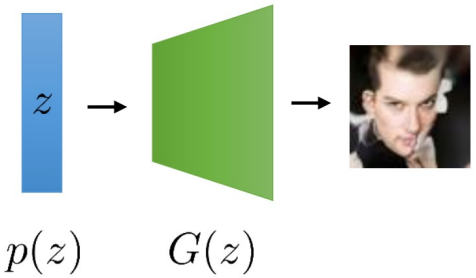
Extensions to GAN

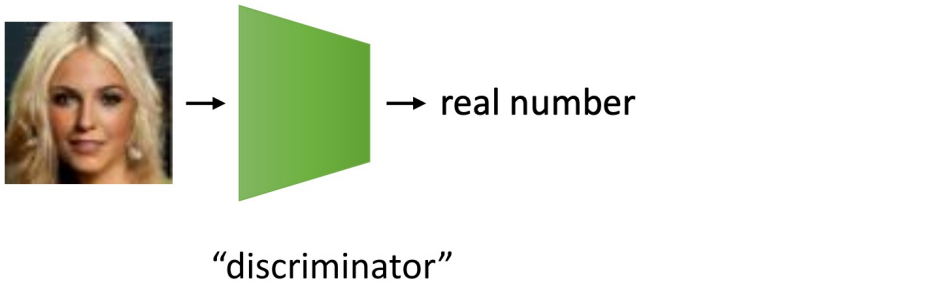
Wasserstein GAN (WGAN) a better distance function

$$W(p_{\text{data}}, p_G) = \sup_{||f||_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$

1. update f_θ using gradient of $E_{x \sim p_{\text{data}}}[f_\theta(x)] - E_{z \sim p(z)}[f_\theta(G(x))]$
2. clip all weight matrices in θ to $[-c, c]$
3. update generator to *maximize* $E_{z \sim p(z)}[f_\theta(G(z))]$







discriminator outputs real numbers (not true/false probability)

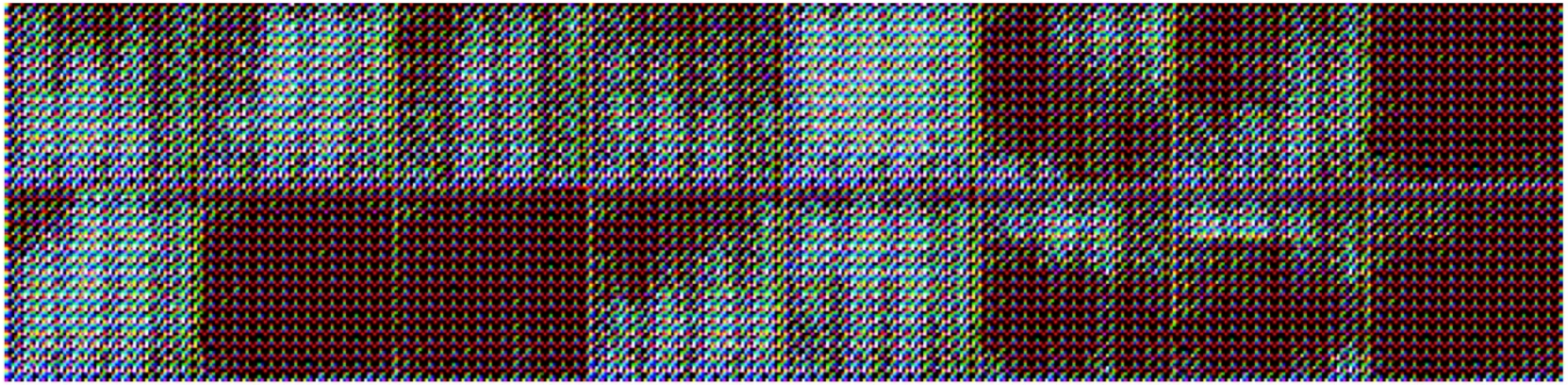
e.g., $f_\theta(x) = W_2 \text{ReLU}(W_1 x + b_1) + b_2$

Martin Arjovsky, Soumith Chintala, and L´eon Bottou, Wasserstein GAN, ICML 2017

Extensions to GAN

Wasserstein GAN (WGAN) a better distance function

WGAN



GAN

Martin Arjovsky, Soumith Chintala, and L´eon Bottou, Wasserstein GAN, ICML 2017

Extensions to GAN

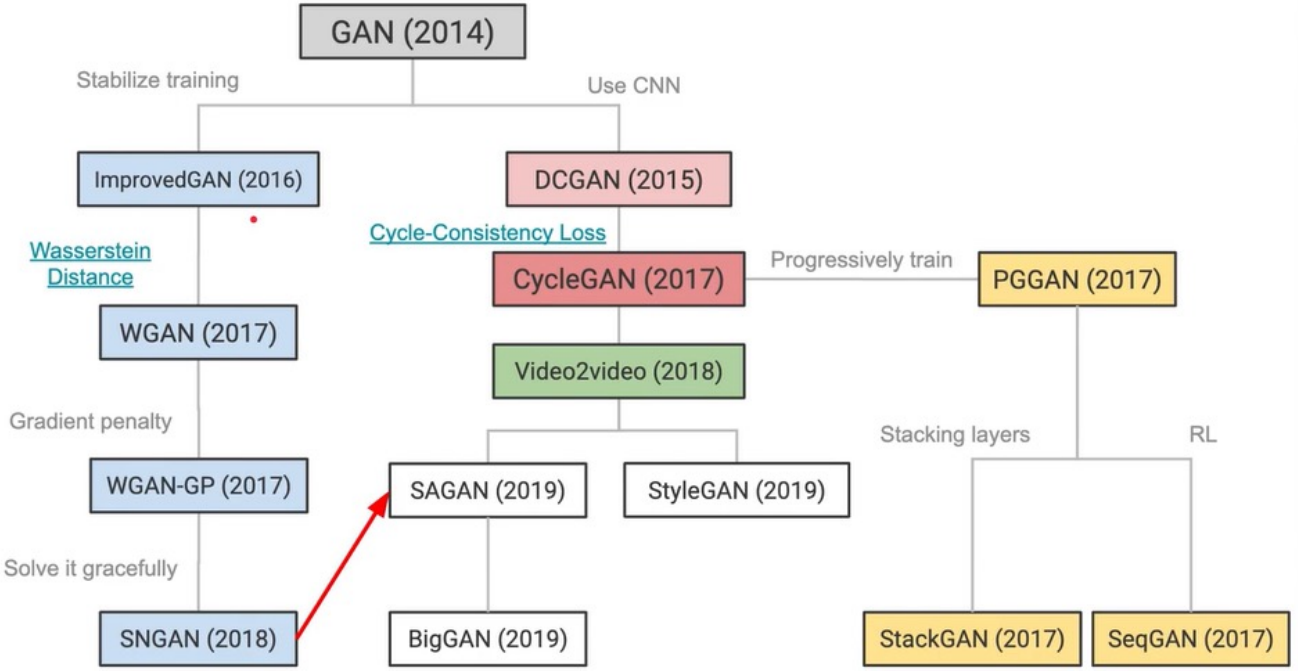
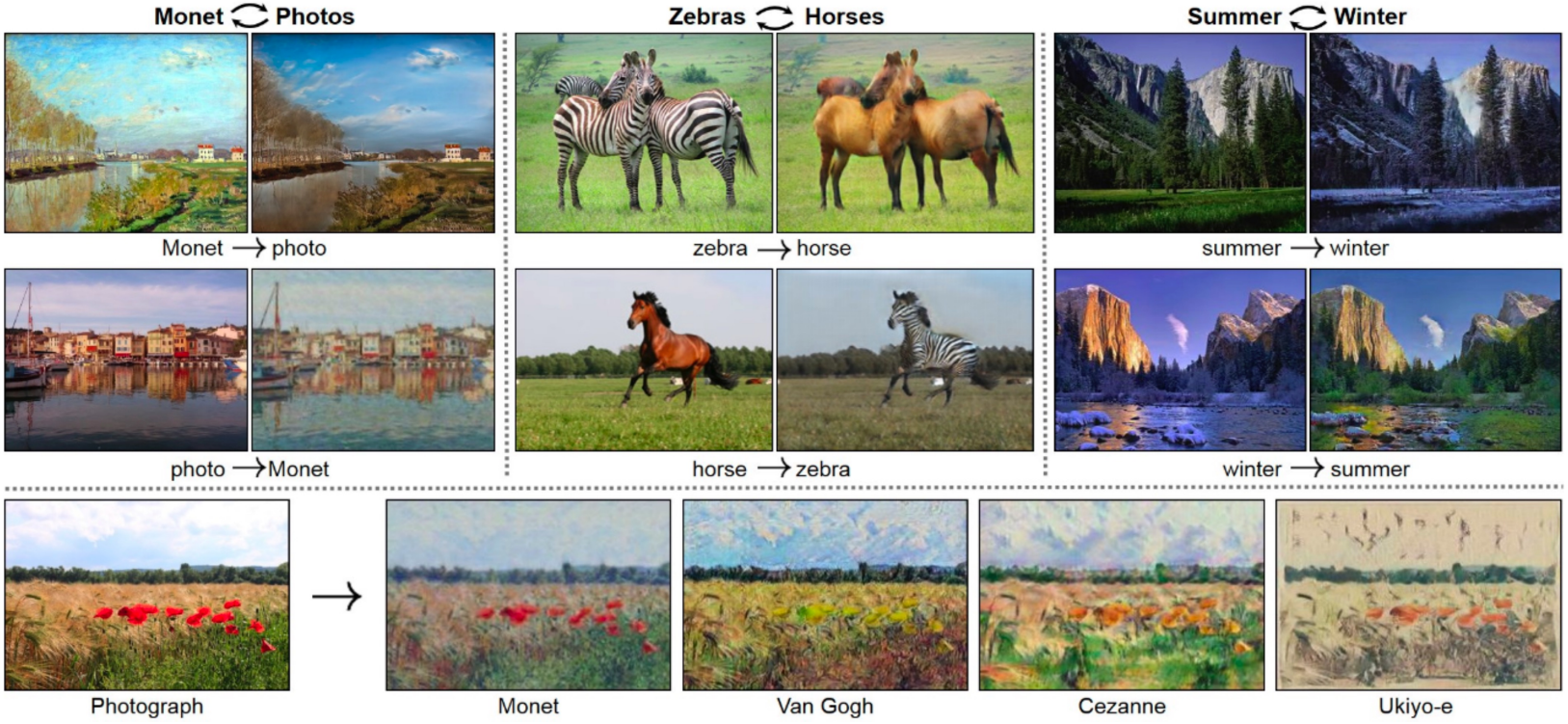


Image from https://blog.csdn.net/weixin_42437114/article/details/118935205

Extensions to GAN

CycleGAN style transfer

Problem: change the style of an image while preserving the content.



Data: Two unrelated collections of images, one for each style

Extensions to GAN

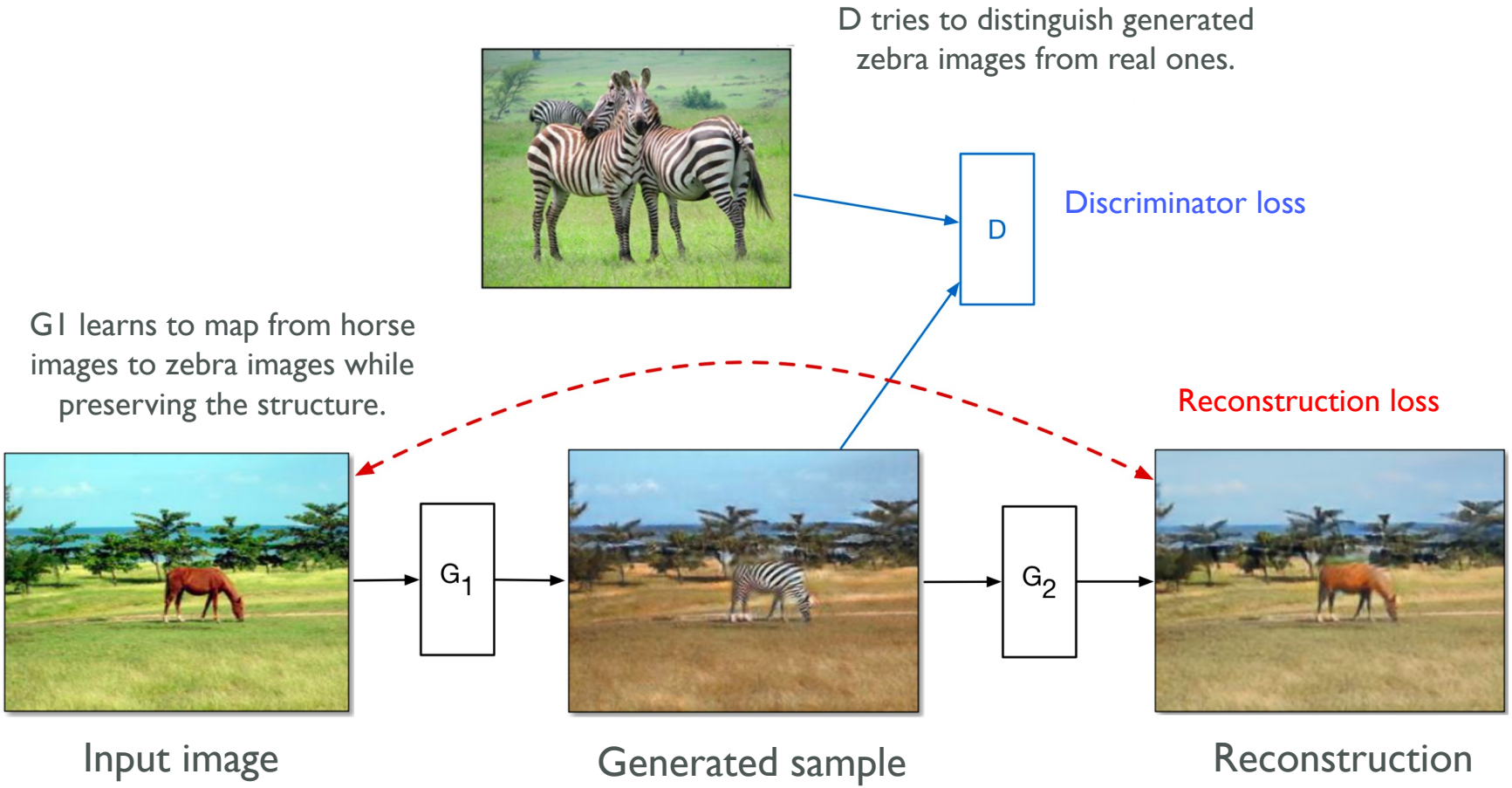
CycleGAN style transfer

The CycleGAN architecture learns to do it from unpaired data.

- Train two different generator nets to go from style 1 to style 2, and vice versa.
- Make sure the generated samples of style 2 are indistinguishable from real images by a discriminator net.
- Make sure the generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.

Extensions to GAN

CycleGAN style transfer



Total loss = Discriminator loss + Reconstruction loss

Outline

- What are generative models?
- Overview of generative models
- Implicit density models
- GAN framework
- Why GAN is hard to train?
- Extensions to GAN
- Normalization Flow
- Diffusion Models

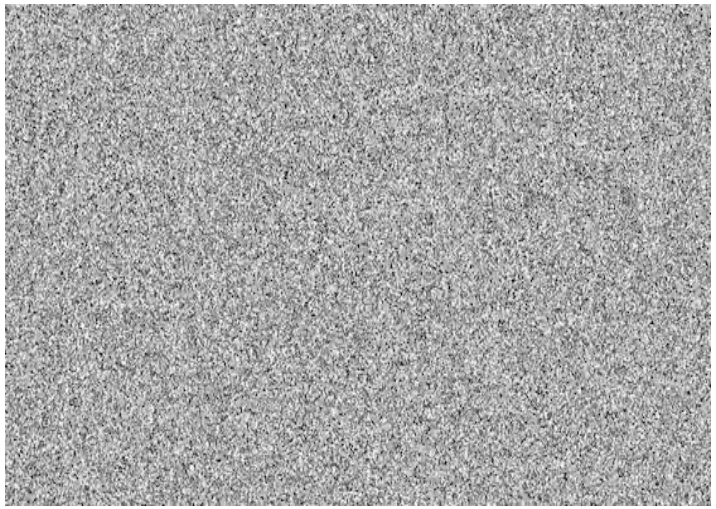
Normalization flows

Mapping an image to noises

x

$f(x)$

$f^{-1}(z)$



z

Danilo Jimenez Rezende, Shakir Mohamed, Variational Inference with Normalizing Flows, ICML 2015

Normalization flows

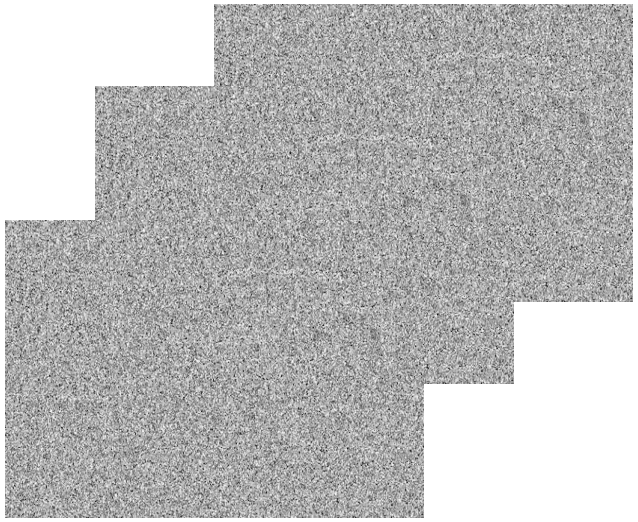
Learn a flow $f(x)$ to transform $p_X(x)$ into $p_Z(z)$

$$p_X(x) = p_Z(f(x))|\det Df(x)|$$

$p_X(x)$

$f(x)$

$f^{-1}(z)$



$p_Z(z)$ $\mathcal{N}(0, I)$

Danilo Jimenez Rezende, Shakir Mohamed, Variational Inference with Normalizing Flows, ICML 2015

Normalization flows

Training via maximum (log-)likelihood

$$\max_{\theta} \sum_{i=1}^{N} \log p_Z(f(x_i|\theta)) + \log \left|\det Df(x_i|\theta)\right|$$

Ivan Kobyzev, Simon J.D. Prince, Marcus A. Brubaker, Normalizing Flows: An Introduction and Review of Current Methods, IEEE T-PAMI, 2020

Normalization flows

Learn a flow $f(x)$ to transform $p_X(x)$ into $p_Z(z)$

$$p_X(x) = p_Z(f(x))|\det Df(x)|$$

A flow is a parametric function which:

- is invertible
- is differentiable
- has an efficiently computable inverse and Jacobian determinant

Ivan Kobyzev, Simon J.D. Prince, Marcus A. Brubaker, Normalizing Flows: An Introduction and Review of Current Methods, IEEE T-PAMI, 2020

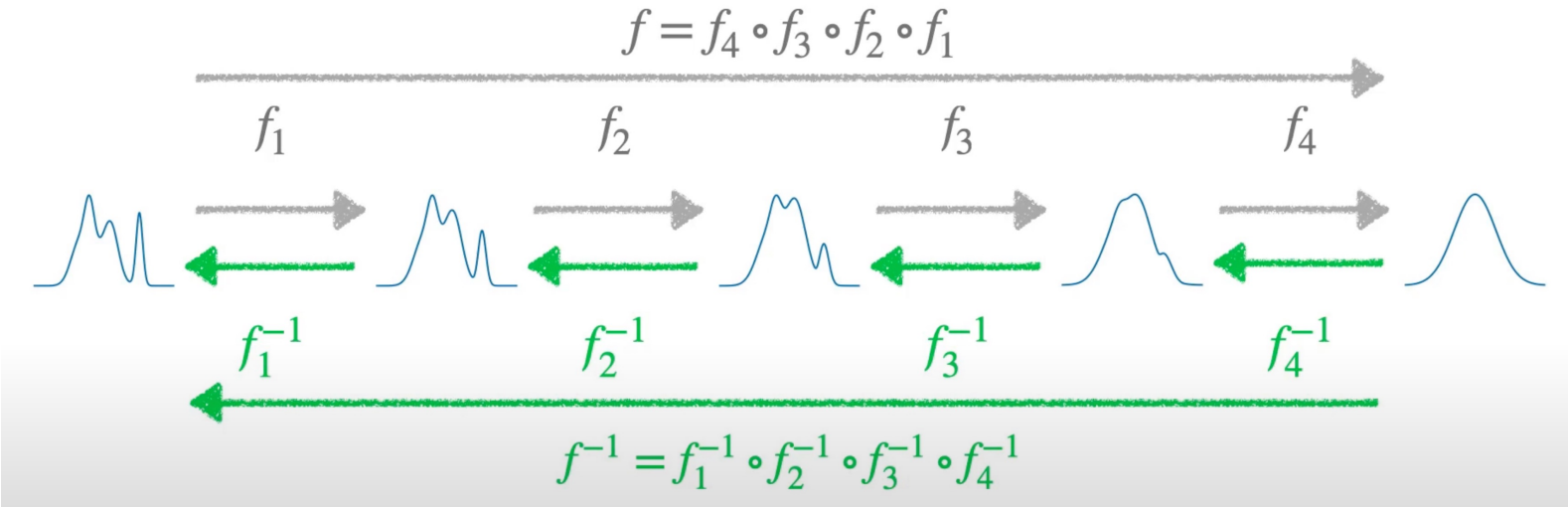
Normalization flows

Composition of flows

- Invertible, differentiable functions are closed under composition

$$f = f_K \circ f_{K-1} \circ \cdots \circ f_2 \circ f_1$$

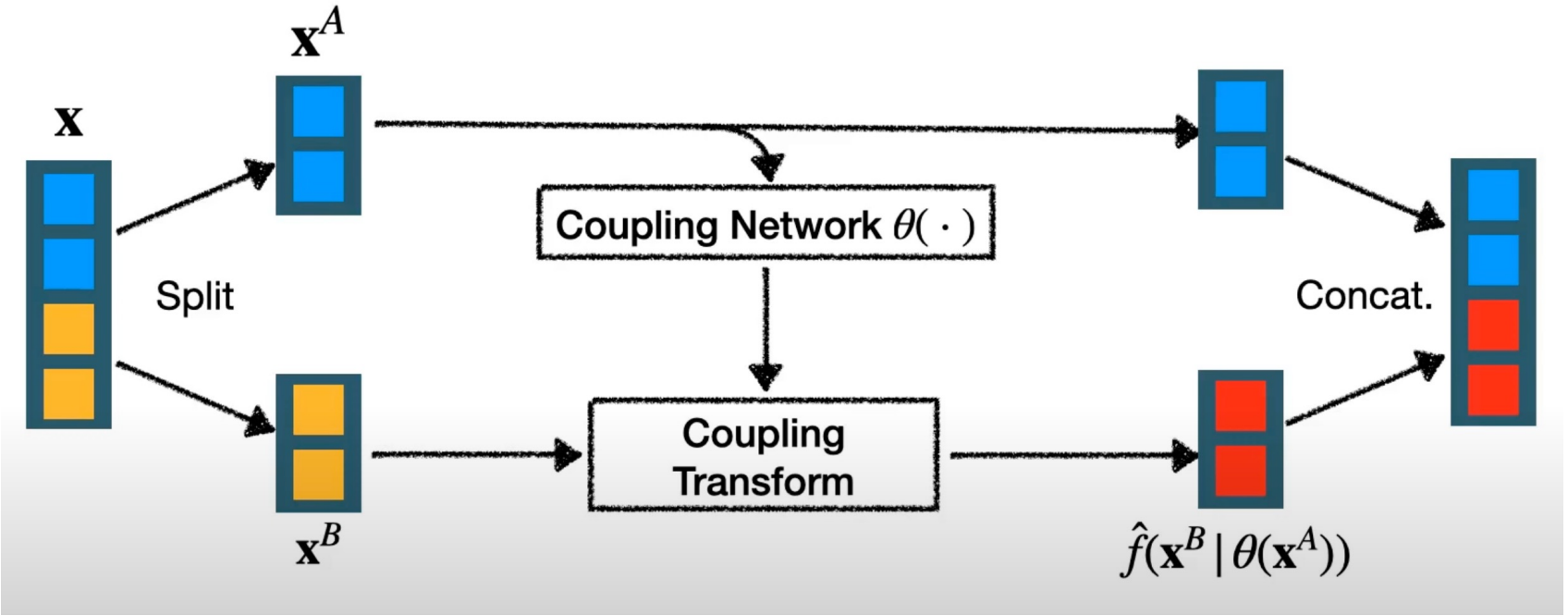
- Build up a complex flow from composition of simpler flows



Ivan Kobyzev, Simon J.D. Prince, Marcus A. Brubaker, Normalizing Flows: An Introduction and Review of Current Methods, IEEE T-PAMI, 2020

Normalization flows

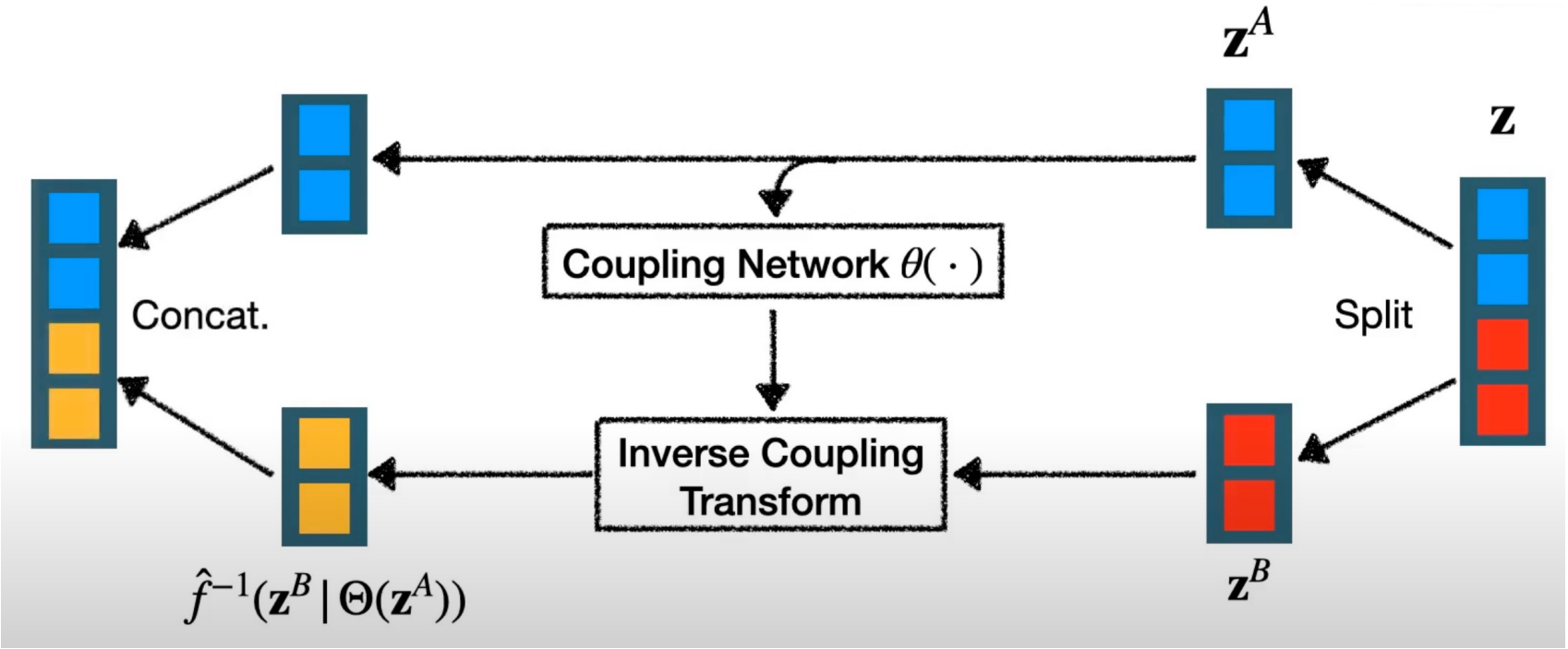
Coupling flows: Forward



Ivan Kobyzev, Simon J.D. Prince, Marcus A. Brubaker, Normalizing Flows: An Introduction and Review of Current Methods, IEEE T-PAMI, 2020

Normalization flows

Coupling flows: Inverse



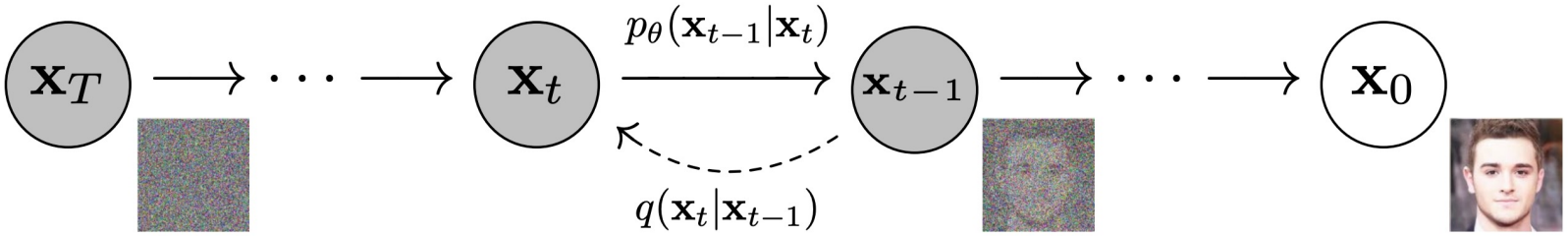
Ivan Kobyzev, Simon J.D. Prince, Marcus A. Brubaker, Normalizing Flows: An Introduction and Review of Current Methods, IEEE T-PAMI, 2020

Outline

- What are generative models?
- Overview of generative models
- Implicit density models
- GAN framework
- Why GAN is hard to train?
- Extensions to GAN
- Normalization Flow
- Diffusion Models

Diffusion models

Progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding



forward process $q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

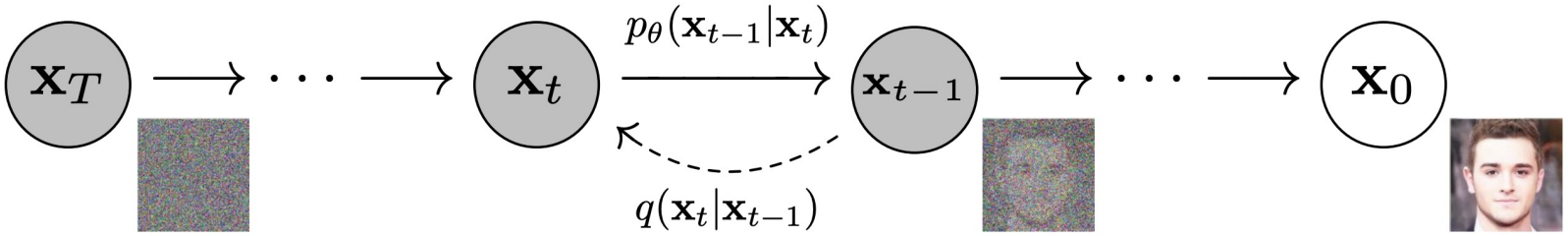
$$\alpha_t := 1-\beta_t \quad \bar{\alpha}_t := \textstyle\prod_{s=1}^{t} \alpha_s$$

reverse process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$

Jonathan Ho, Ajay Jain, Pieter Abbeel, Denoising Diffusion Probabilistic Models, NeurIPS 2020

Diffusion models

Progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding



Algorithm 1 Training

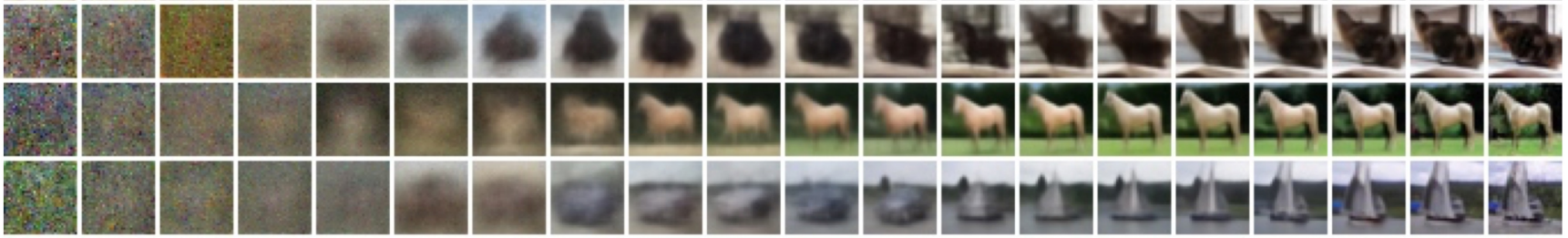
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

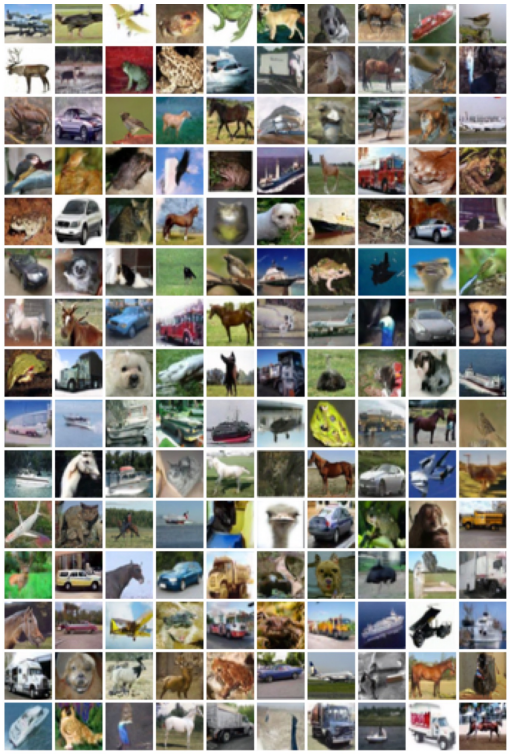
Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Jonathan Ho, Ajay Jain, Pieter Abbeel, Denoising Diffusion Probabilistic Models, NeurIPS 2020

Diffusion models





Jonathan Ho, Ajay Jain, Pieter Abbeel, Denoising Diffusion Probabilistic Models, NeurIPS 2020

Thank you very much!

sihengc@sjtu.edu.cn