香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Deep Generative Learning and its Application

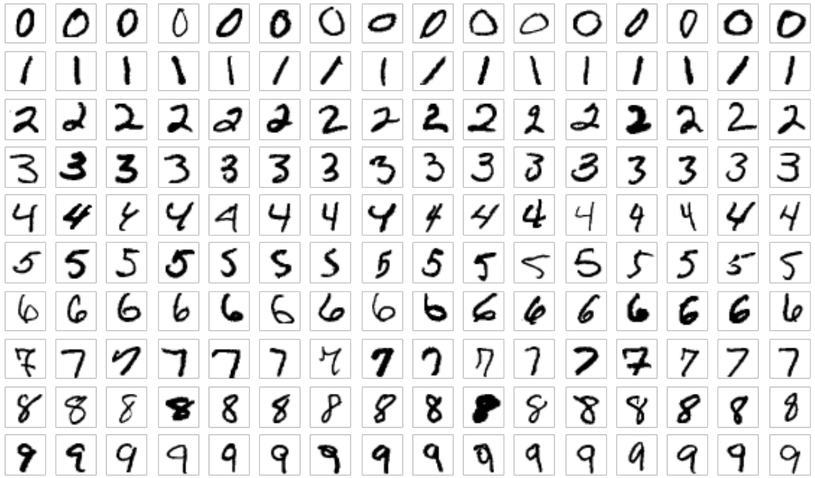
Prof. Yang Wang

Email: yangwang@ust.hk

The Hong Kong University of Science and Technology

Generative Adversarial Nets (GAN)

- Groundbreaking work by Ian Goodfellow et al (2014)
- It tried to address the following question: Given a set of data (say, a set of human faces or Van Gogh paintings, can we generate data that are “similar”?
- The authors have proposed GAN, which uses two neural networks “competing against” each other to obtain desired outcome
- Yann LeCun has called “this (GAN) and the variations that are now best interesting idea in the last 10 years in ML, in my opinion.”



An example: Image inpainting / colorization

Input

Output

"A lack of information cannot be remedied by any mathematical trickery."
Lanczos, Cornelius. 1964. Linear Differential Operators.

It is almost impossible if we restrict ourselves to one single image.

AI Art at Christie's Sells for $432,500

The New York Times

ARTS

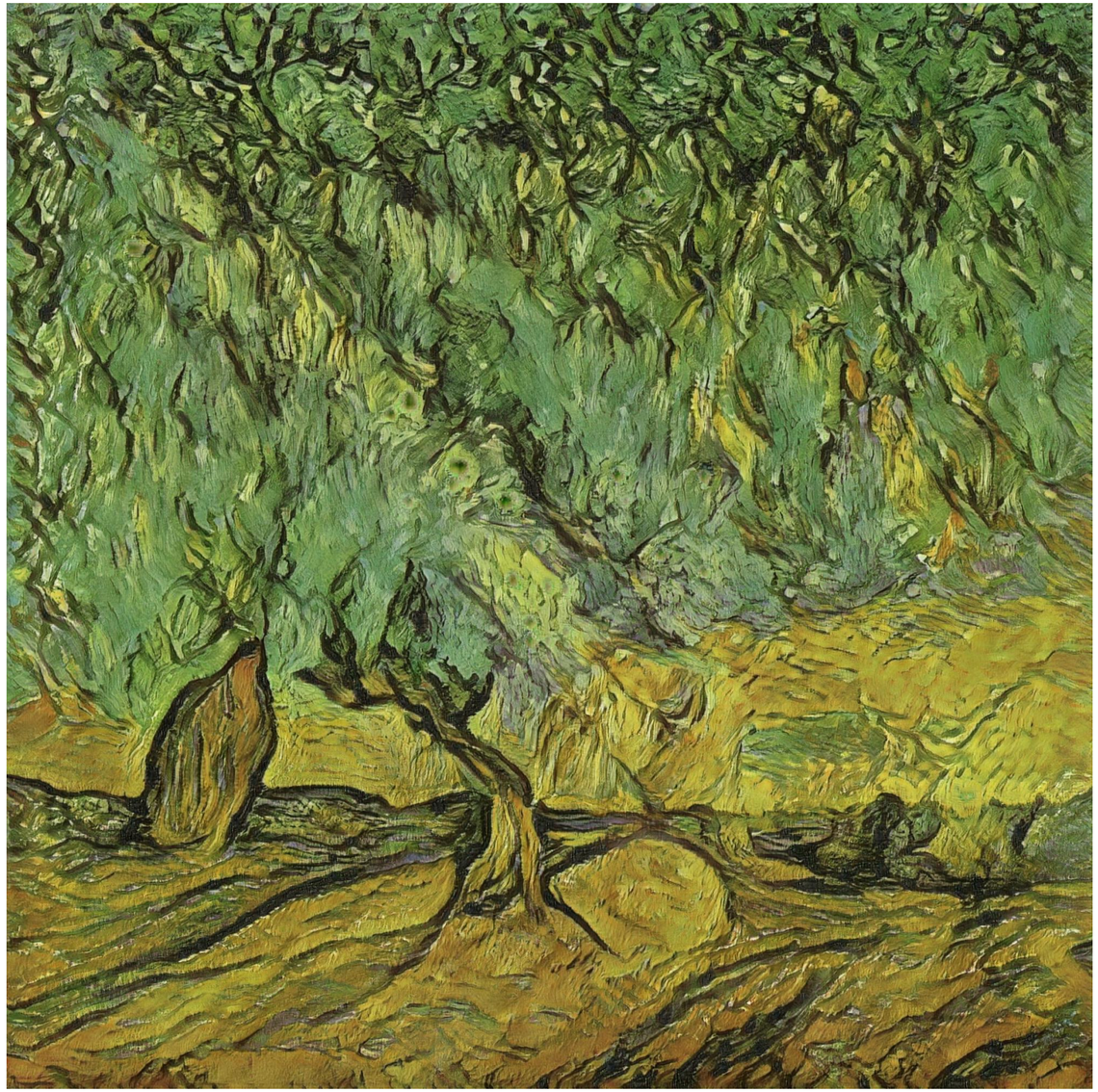
Art & Design

Please pay attention to the signature:

$$\min_G \max_D \left[\mathbb{E}_x[\log(D(x))] + \mathbb{E}_z[\log(1 - D(G(z)))]\right]$$

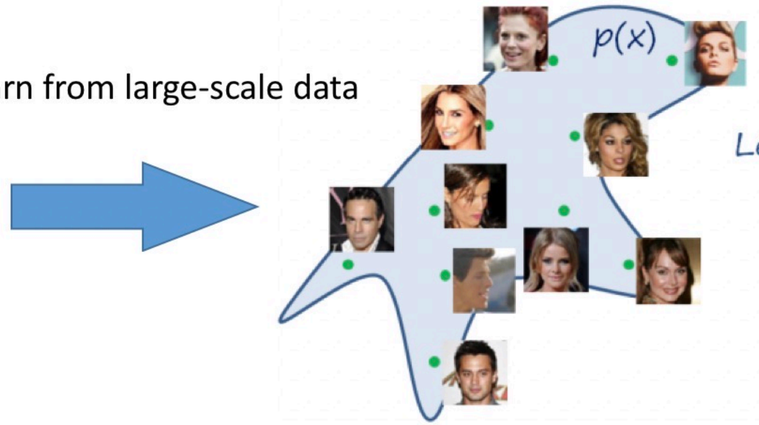
Edmond de Belamy, sold on Oct 25, 2018

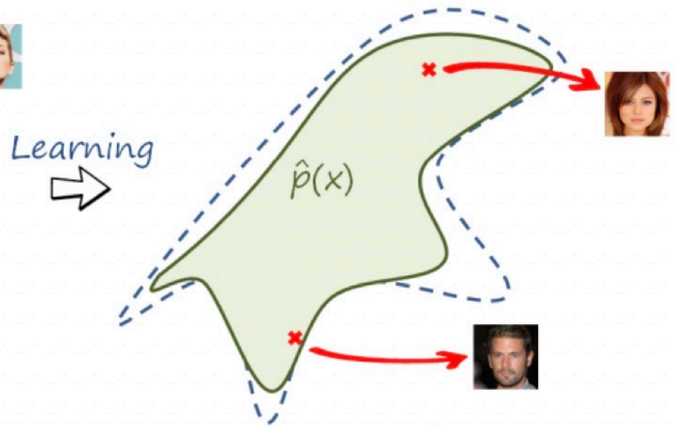
Mathematical Art at HKUST



Leveraging Big Data

learn from large-scale data





GAN: Basic Ideas

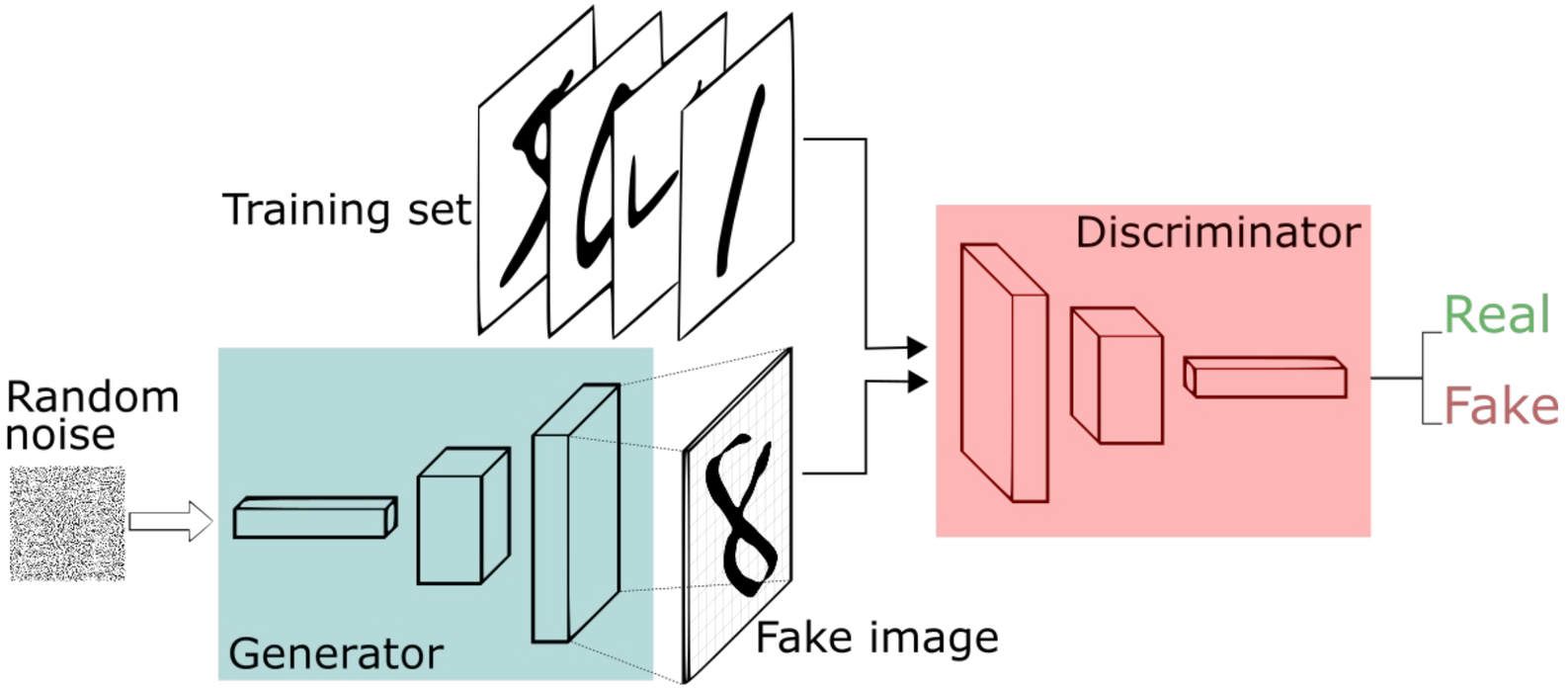
- Given a training set *X*, e.g. *X* = set of dog images, we shall assume X contain samples drawn from an *unknown distribution* $p_X(x)$
- We would like to learn $p_X(x)$ from samples in *X*. But how?
- Starting off with random noise $z \sim N(0, I)$, we try to find a function G so that the distribution q_G of $G(z)$ approximates the distribution p_X
- Once we find G, from any random sample $z \sim N(0,1)$ we obtain a generated sample $G(z)$ that should look like something from *X*

The vanilla GAN by Goodfellow et al achieves this via:

- Approximate G using a neural network
- Introducing another neural network D to "compete against" G so that G will continue to improve
- The "competition" between G and D is in the form of a minimax optimization of a loss function

Vanilla GAN

$$\min_G \max_D V(D,G)$$
$$= \min_G \max_D E_{x\sim p_X}[\log(D(x))] + E_{w\sim N(0,I)}[\log(1 - D(G(w))]$$



Generative Adversarial Networks. Source: Google Images

Training the Vanilla GAN

Algorithm: Alternating the following two steps

1. Given the generator G, optimize the discriminator D:

$$\max_D E_{x\sim p_X}[\log(D(x))] + E_{z\sim N(0,I)}[\log(1 - D(G(z))]$$

2. Given the discriminator D, optimize the generator G :

$$\min_G E_{x\sim p_X}[\log(D(x))] + E_{z\sim N(0,I)}[\log(1 - D(G(z))]$$

$$\Longleftrightarrow \quad \min_G E_{z\sim N(0,I)}[\log(1 - D(G(z))]$$

- **Log-D trick:** replaces Step 2 by

$$\min_G E_{z\sim N(0,I)}[-\log D(G(z))]$$

Mathematics of GAN: Minimizing Divergence

We are essentially trying to find a function G so that the distribution q_G of $G(z)$ is close to the unknown target distribution p_X, where $z \sim N(0, I)$.

There are two important questions here:

- What do we mean by two distributions p_X and q_G being close to each other?
- We have no explicit expression of p_X, how do we quantify how close is q_G to p_X?

Divergence: A metric for probability distributions

Let $p(x)$ and $q(x)$ be two probability distributions (for simplicity they are density functions)

- Kullback-Leibler Divergence: $D_{KL}(p\|q) = E_{x\sim p}[\log\left(\frac{p(x)}{q(x)}\right)] = \int_{\mathbb{R}^n} \log\left(\frac{p(x)}{q(x)}\right)p(x)dx$
- Jensen-Shannon Divergence: $D_{JS}(p\|q) = D_{KL}(p\|M) + D_{KS}(q\|M) \quad M = \frac{1}{2}(p+q)$

There are other divergences

Mathematics of GAN: Minimizing Divergence

It is proved that for vanilla GAN

$$\min_G \max_D E_{x\sim p_X}[\log(D(x))] + E_{z\sim N(0,I)}[\log(1 - D(G(z))]$$

is equivalent to

$$\min_G D_{JS}(p_X \| q_G)$$

The beauty of the vanilla GAN objective is that it has converted the minimization of JS-divergence into a minimax involving expectations.

- Minimizing JS-divergence (or KL-divergence) directly without knowing p_X and G explicitly is **highly nontrivial**
- The expectations, however, can be estimated by (mini-batch) sample averages. E.g. given $D(\mathrm{x})$

$$E_{x\sim p_X}[\log(D(x))] \approx \frac{1}{N}\sum_{i=1}^{K} \log(D(x_i))$$

where x_i are samples drawn in the training dataset X

f-Divergence

While vanilla GAN minimizes the Jensen-Shannon divergence, there are many other divergences one can use for GANs. A more general framework is f-divergence.

Let $f(t)$ be any strictly convex function with $f(1) = 0$. The f-divergence between two probability distributions p and q is

$$D_f(q\|p) = E_{x\sim p}[f\left(\frac{q(x)}{p(x)}\right)] = \int f\left(\frac{q(x)}{p(x)}\right)p(x)\,dx$$

- Why is it a divergence? It follows from Jensen's Inequality

$$D_f(q\|p) \geq f\left(E_{x\sim p}[\frac{q(x)}{p(x)}]\right) = f\left(\int \frac{q(x)}{p(x)}\cdot p(x)\,dx\right) = f(1) = 0$$

f-GAN: Minimizing f-Divergence

GANs can be trained by minimizing f-divergence (f-GAN)

$$\min_G D_f(p_X \| q_G)$$

Like for the Jensen-Shannon divergence for vanilla GAN, there is an equivalent dual version of this minimization

$$\min_G \max_T E_{x\sim p_X}[f(T(x))] - E_{z\sim N(0,I)}[f^*(T(G(z)))]$$

where f^* is the convex dual (Fenchel dual) of f.

- With the dual version, expectations can now be estimated by (mini-batch) sample averages.
- Both $G(x)$ and $T(x)$ will be modeled by neural networks, thus we write $G(x)$ and $T(x)$ as $G_\theta(x)$ and $T_\omega(x)$, where θ and ω are neural network parameters.
- The parameters are optimized by SGD.

f-GAN: Minimizing f-Divergence

- The dual model solves the minimax optimization problem

$$\min_{\theta}\max_{\omega} E_{x\sim p_X}[f(T_\omega(x))] - E_{z\sim N(0,I)}[f^*(T_\omega(G_\theta(z)))]$$

- Actually we often encounter the same problems (i.e. vanishing gradient, mode collapse, etc) in training for f-GAN as we do with vanilla GAN.
- So is there a better way to train GAN so it is more stable?
- Turns out there is a way to directly solve the primal minimization problem

$$\min_\theta D_f(p_X \| q_\theta)$$

where q_θ is the distribution of $G_\theta(z)$, with $z\sim N(0,I)$.

- Our Variational Gradient Flow (Vgrow) method does just that, and it proves to offer a much more stable method for training GAN.

GAN and *f*-divergence (2018)

PMLR Proceedings of Machine Learning Research

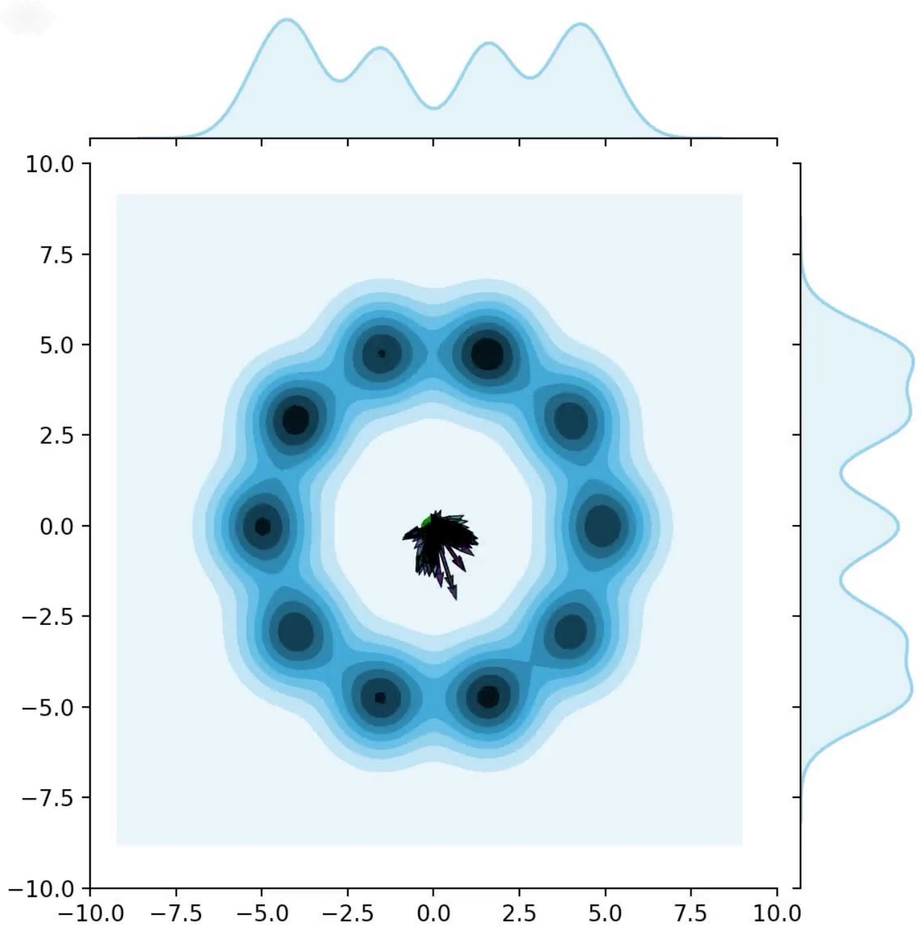
Volume 97 All Volumes JMLR MLOSS FAQ Submission Format RSS

Deep Generative Learning via Variational Gradient Flow

[edit]

Yuan Gao, Yuling Jiao, Yang Wang, Yao Wang, Can Yang, Shunkang Zhang ; ***Proceedings of the 36th International Conference on Machine Learning, PMLR 97:2093-2101, 2019.***

f-DIV	$f(u)$	$f''(u)$
KL	$u \log u$	$\frac{1}{u}$
JS	$-(u+1)\log\frac{u+1}{2} + u\log u$	$\frac{1}{u(u+1)}$
LOGD	$(u+1)\log(u+1) - 2\log 2$	$\frac{1}{u+1}$
JEFFREY	$(u-1)\log u$	$\frac{u+1}{u^2}$



Primal Minimization --Variational Gradient Flow (VGrow)

Goal: Directly solve $\min_\theta D_f(p_X \| q_\theta)$

Idea: Use variational gradient wrt q_θ, where recall that q_θ is the distribution of $G_\theta(z)$, with $z \sim N(0, I)$.

Step 1. Update $G_{\theta_n}(z)$ via functional gradient $\overline{G_n} = G_{\theta_n} + s \cdot h_n$, where $s > 0$ and

$$h_n = -\frac{\partial}{\partial q_\theta} D_f(p_X \| q_\theta)\Big|_{\theta=\theta_n} = -f''(r)\nabla r$$

Step 2. Update θ of the generator G_θ via

$$\theta_{n+1} = \text{argmin}_\theta \sum_{i=1}^{N} \left\| G_\theta(z_i) - \overline{G_n}(z_i) \right\|^2$$

where $z_i \sim N(0, I)$.

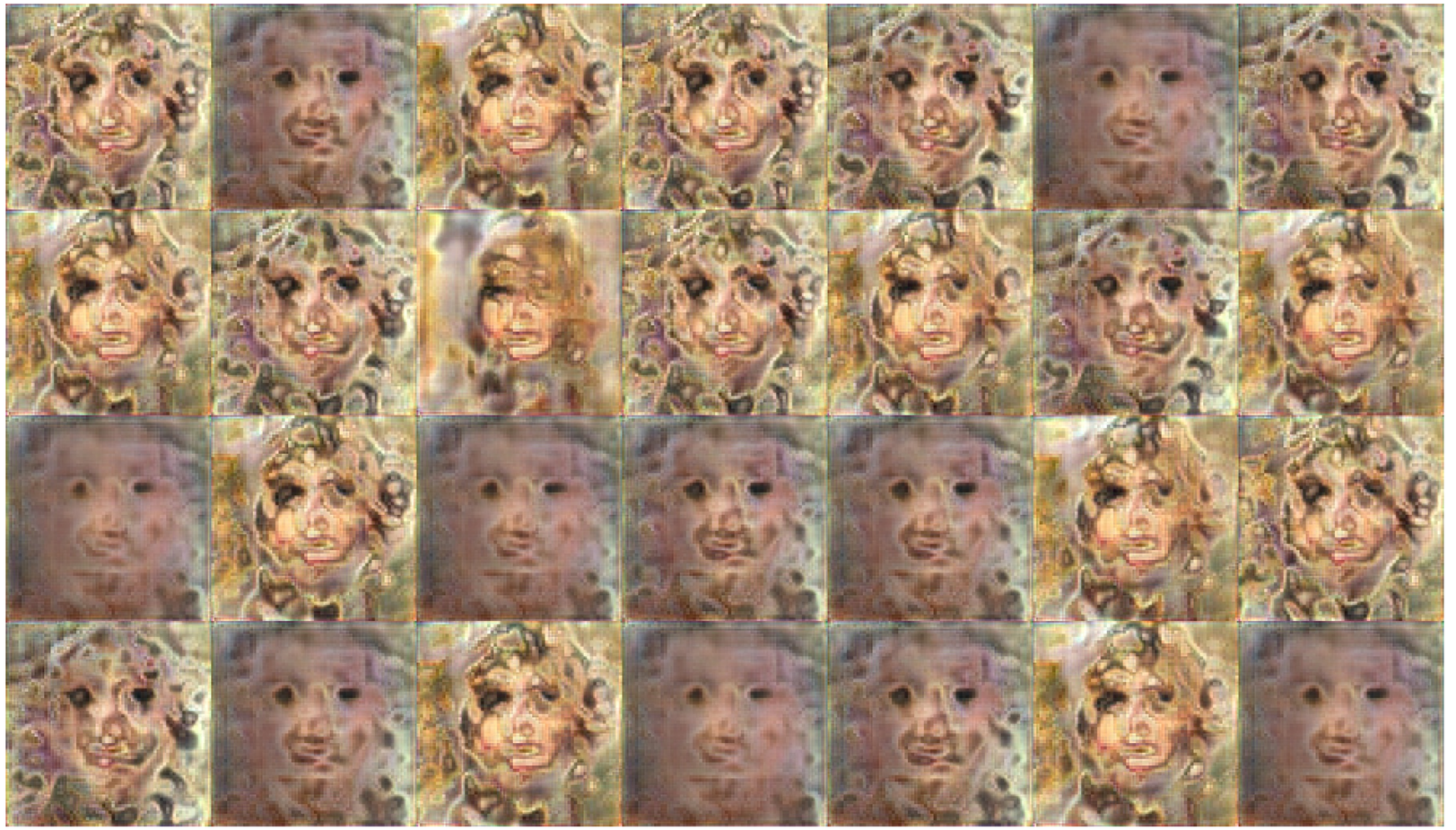
$$r(x) = \frac{q_\theta(x)}{p_X(x)}$$

Density ratio estimator

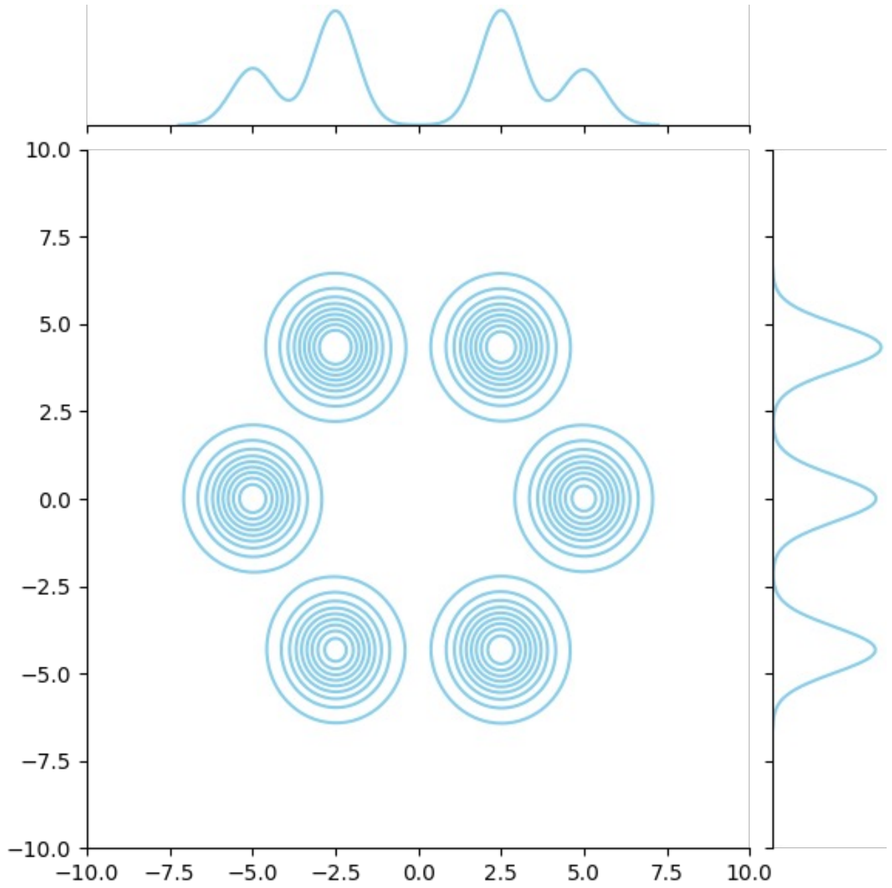
AI Art (2019)

Training data: 11,170 portrait paintings at 256×256 resolution

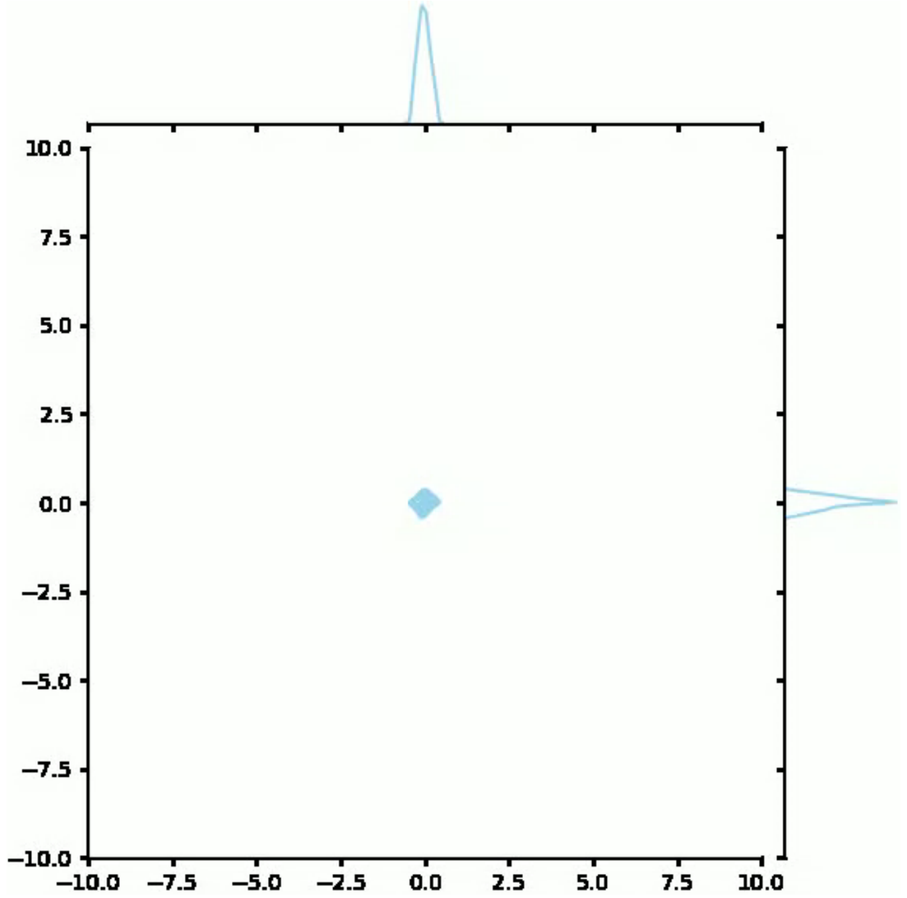
Mode Collapse in Training GAN



Demo of mode collapse



Ground truth



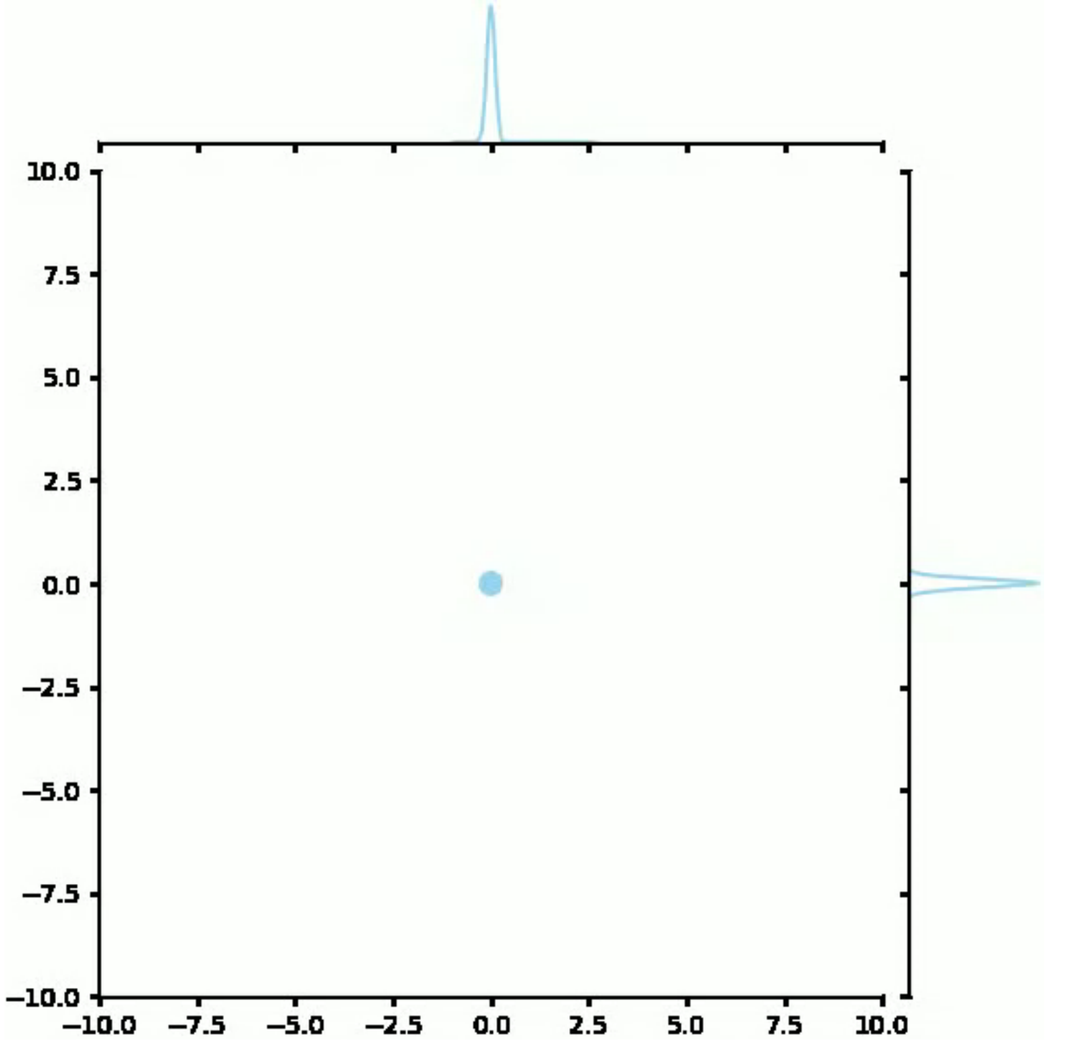
Vanilla GAN

Deep Generative Learning via Schrödinger Bridge (2021)

PMLR **Proceedings of Machine Learning Research**

Deep Generative Learning via Schrödinger Bridge

Gefei Wang, Yuling Jiao, Qian Xu, Yang Wang, Can Yang *Proceedings of the 38th International Conference on Machine Learning*, PMLR 139:10794-10804, 2021.



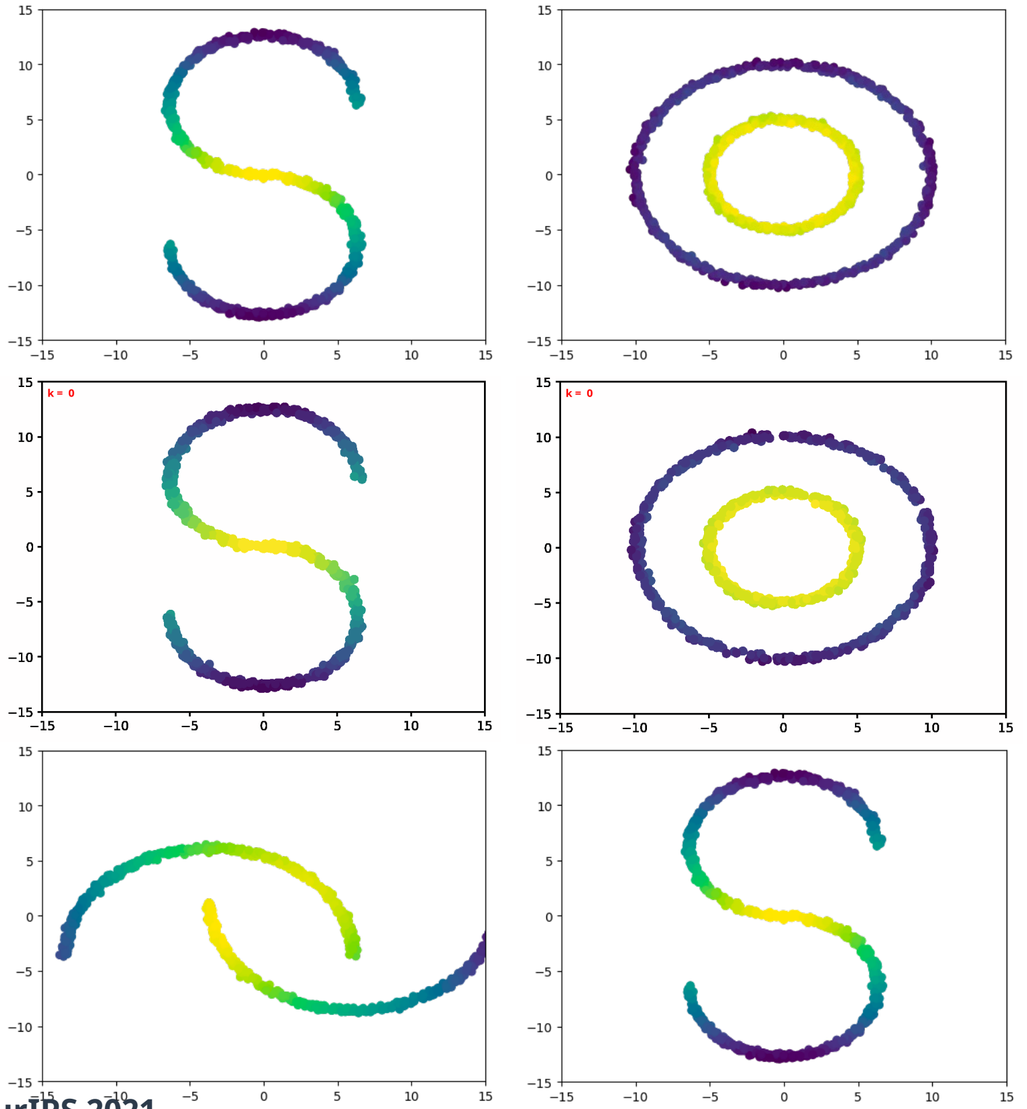
Schrödinger Bridge Problem (SBP)

- A large number of independent particles in $\mathbb{R}^d$ are observed.
- $t = 0, \mu(\mathbf{x}) = q(\mathbf{x})\mathrm{d}\mathbf{x}; t = 1, \upsilon(\mathbf{x}) = p(\mathbf{x})\mathrm{d}\mathbf{x}.$
- SBP aims to find the one which is closest to the Brownian motion.
- **(Dai Pra, 1991)**:

$$\mathbf{u}_t^*(\mathbf{x}) \in \operatorname{argmin}_{\mathbf{u}\in\mathcal{U}} \mathbb{E}\left[\int_0^1 \frac{1}{2}\|\mathbf{u}_t\|^2 \,\mathrm{d}t\right],$$

$$\text{s.t.} \begin{cases} \mathrm{d}\mathbf{x}_t = \mathbf{u}_t \mathrm{d}t + \sqrt{\tau}\mathrm{d}\mathbf{w}_t, \\ \mathbf{x}_0 \sim q(\mathbf{x}), \qquad \mathbf{x}_1 \sim p(\mathbf{x}). \end{cases}$$

- $\mathcal{U}$ is the set of controls with finite energy, that satisfying the above condition.

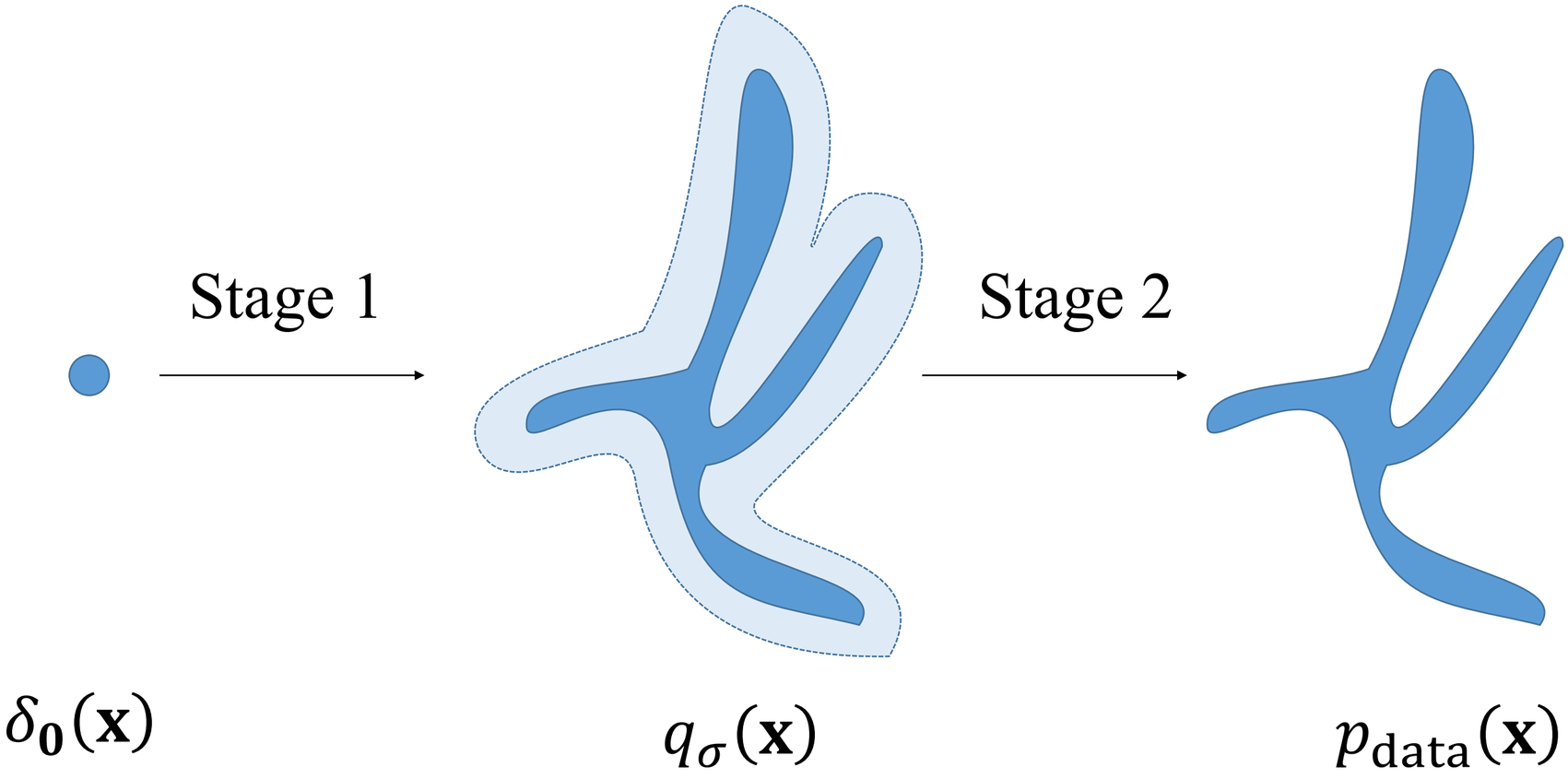


Valentin De Bortoli, James Thornton, Jeremy Heng, Arnaud Doucet

Diffusion Schrödinger Bridge with Applications to Score-Based Generative Modeling, NeurIPS 2021

https://vdeborto.github.io/publication/schrodinger_bridge/

Deep Generative Learning via Schrödinger Bridge



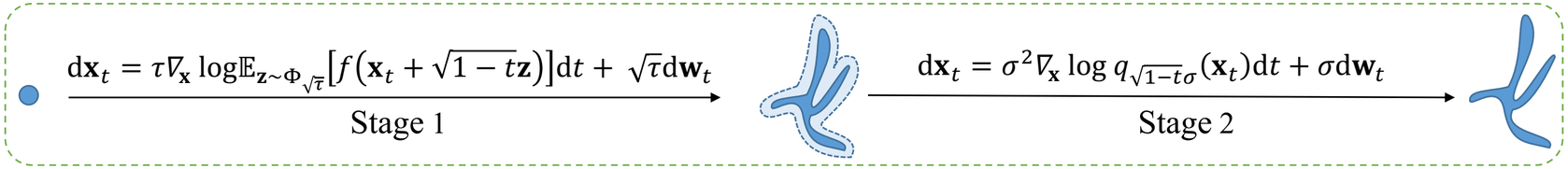
As the underlying distribution is difficult to learn, we first smooth it with $q_\sigma(\mathbf{x})$, $q_\sigma(\mathbf{x}) = \int p_{\text{data}}(\mathbf{y})\Phi_\sigma(\mathbf{x}-\mathbf{y})\mathrm{d}\mathbf{y}$ is a smoothed distribution of $p_{\text{data}}(\mathbf{x})$, where $\Phi_\sigma(\mathbf{x})$ is the density of $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$

Sampling algorithm

- **Stage 1**: $\mathrm{d}\mathbf{x}_t = \tau \nabla_{\mathbf{x}} \log \mathbb{E}_{\mathbf{z}\sim\boldsymbol{\Phi}_{\sqrt{\tau}}}\left[f\left(\mathbf{x}_t + \sqrt{1-t}\mathbf{z}\right)\right]\mathrm{d}t + \sqrt{\tau}\mathrm{d}\mathbf{w}_t$, $t \in [0,1]$. Let $\mathbf{x}_0 \sim \delta_{\mathbf{0}}(\mathbf{x})$, then $\mathbf{x}_1 \sim q_\sigma(\mathbf{x})$. $\left(f(\mathbf{x}) = \frac{q_\sigma(\mathbf{x})}{\boldsymbol{\Phi}_{\sqrt{\tau}}(\mathbf{x})}\right)$.

- **Stage 2**: $\mathrm{d}\mathbf{x}_t = \sigma^2 \nabla_{\mathbf{x}} \log q_{\sqrt{(1-t)}\,\sigma}(\mathbf{x}_t)\mathrm{d}t + \sigma \mathrm{d}\mathbf{w}_t$, $t \in [0,1]$. Let $\mathbf{x}_0 \sim q_\sigma(\mathbf{x})$, then $\mathbf{x}_1 \sim p_{\mathrm{data}}(\mathbf{x})$.

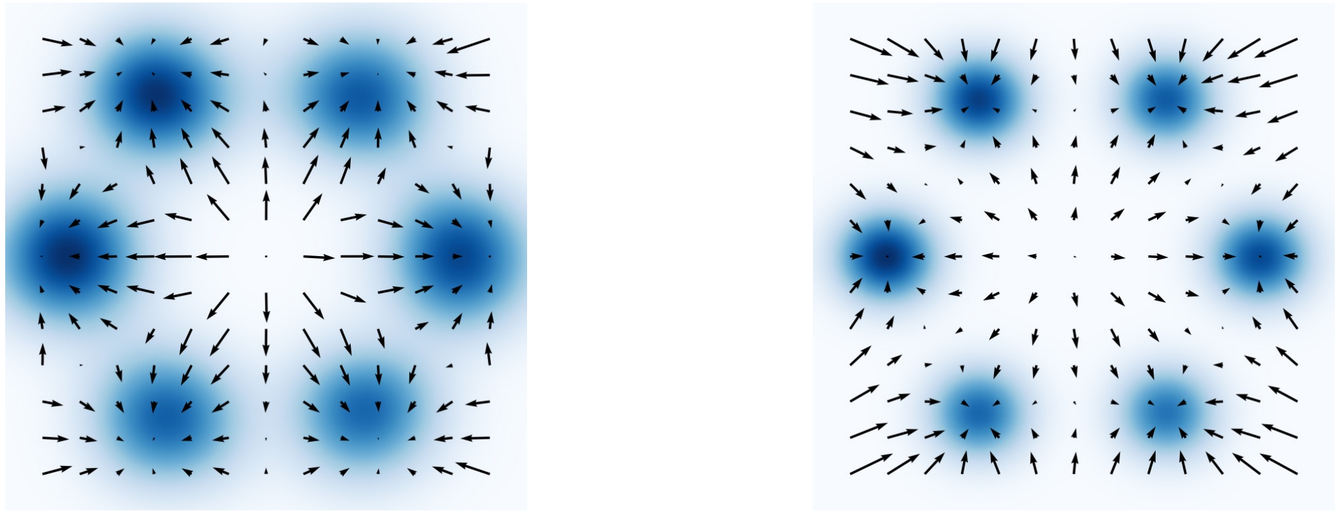
- With the two estimators $\hat{f}(\mathbf{x}) \approx \log \frac{q_\sigma(\mathbf{x})}{\Phi_{\sqrt{\tau}}(\mathbf{x})}$ and $\hat{\mathbf{s}}(\mathbf{x};\tilde{\sigma}) \approx \nabla_{\mathbf{x}} \log q_{\tilde{\sigma}}(\mathbf{x})$, $\tilde{\sigma} \in [0,\sigma]$, we can use the Euler-Maruyama method to solve the SDEs.



Theoretical results

Theoretical results: we proved that

- Drift terms can be estimated consistently.



Estimated drift terms of stage 1 and stage 2 in 2D example.

Theoretical results

- **Theorem (Consistency of the density ratio estimator)**

Assume that the support of $p_{\text{data}}(\mathbf{x})$ is contained in a compact set, and density ratio $f(\mathbf{x}) = \frac{q_\sigma(\mathbf{x})}{\Phi_{\sqrt{\tau}}(\mathbf{x})}$ is Lipschitz continuous and bounded. Set the depth $\mathcal{D}$, width $\mathcal{W}$ and size $\mathcal{S}$ of $\mathcal{NN}_\phi$ as

$$\mathcal{D} = O\big(\log(n)\big), \mathcal{W} = O\left(n^{\frac{d}{2(2+d)}} \log(n)^{-1}\right), \mathcal{S} = O\left(n^{\frac{d-2}{d+2}} \log(n)^{-3}\right).$$

Then $\mathbb{E}\left[\left\|\hat{f}(\mathbf{x}) - f(\mathbf{x})\right\|_{L^2(p_{\text{data}})}\right] \to 0$ as $n \to \infty$, where d is the dimensionality of data, n is the number of sample used to train the estimator.

Technical theoretical results

- **Theorem (Consistency of the score estimator)**

Assume that the support of $p_{\text{data}}(\mathbf{x})$ is differentiable with bounded support, and $\nabla_{\mathbf{x}} \log q_{\tilde{\sigma}}(\mathbf{x})$ is Lipschitz continuous and bounded for $(\tilde{\sigma}, \mathbf{x}) \in [0, \sigma] \times \mathbb{R}^d$. Set the depth $\mathcal{D}$, width $\mathcal{W}$ and size $\mathcal{S}$ of $\mathcal{NN}_\theta$ as

$$\mathcal{D} = O\big(\log(n)\big), \mathcal{W} = O\left(\max\left\{n^{\frac{d}{2(2+d)}} \log(n)^{-1}, d\right\}\right),$$
$$\mathcal{S} = O\left(dn^{\frac{d-2}{d+2}} \log(n)^{-3}\right).$$

Then $\mathbb{E}\left[\left\| \|\hat{\mathbf{s}}_\theta(\mathbf{x}; \tilde{\sigma}) - \nabla_{\mathbf{x}} \log q_{\tilde{\sigma}}(\mathbf{x})\|_2 \right\|_{L^2(q_{\tilde{\sigma}})}\right] \to 0$ as $n \to \infty$, where d is the dimensionality of data, n is the number of sample used to train the estimator.

Theoretical results

Theoretical results: we proved that

- **(Consistency of proposed algorithm, informal)** Under some mild smoothness assumptions of the target distribution,
  $\mathbb{E}\left[\mathcal{W}_2\left(\mathrm{Law}\left(\mathbf{x}_{\mathrm{output}}\right), p_{\mathrm{data}}\right)\right] \to 0,$

as the **number of samples** for training and estimating drift terms, width, depth and size of **neural networks** and **discretization steps** of the Euler-Maruyama method go to ∞**.**

Technical theoretical results

- **Theorem (Consistency result of our method)**

Let $D_1(t,\mathbf{x}) = \nabla_{\mathbf{x}} \log \mathbb{E}_{\mathbf{z}\sim\boldsymbol{\Phi}_{\sqrt{\tau}}}\left[f\left(\mathbf{x}+\sqrt{1-t}\mathbf{z}\right)\right]$, $D_2(t,\mathbf{x}) = \nabla_{\mathbf{x}} \log q_{\sqrt{(1-t)}\,\sigma}(\mathbf{x})$.

Denote $h_{\sigma,\tau}(\mathbf{x}_1,\mathbf{x}_2) = \exp\left(\frac{\|\mathbf{x}_1\|^2}{2\tau}\right) p_{\text{data}}(\mathbf{x}_1+\sigma\mathbf{x}_2)$.

Under the following **assumptions**:

1. $\text{supp}(p_{\text{data}})$ is contained in a ball with radius R, and $p_{\text{data}} > c > 0$ on its support;

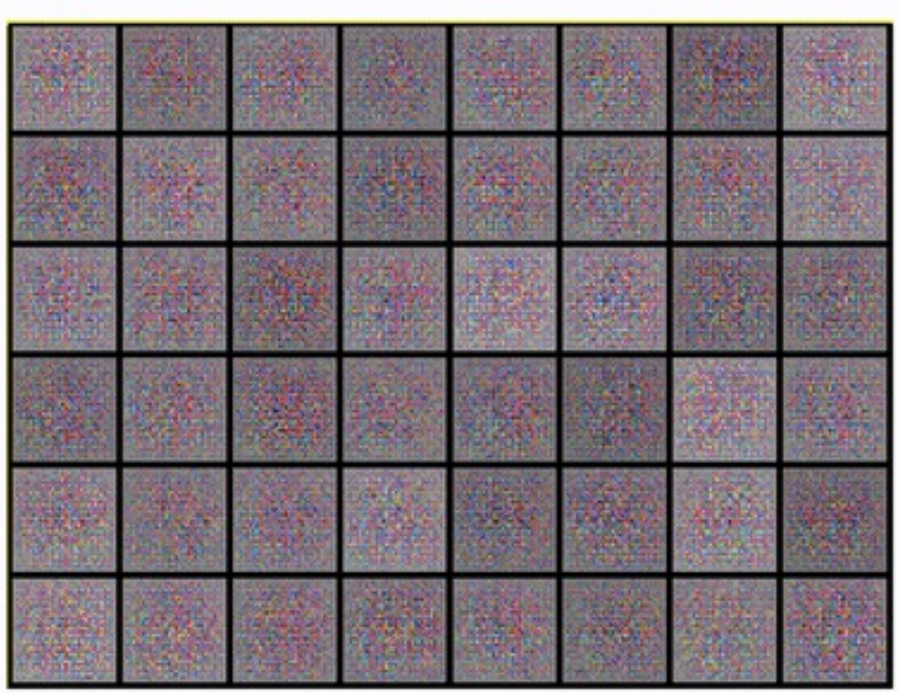
2. $\|D_i(t,\mathbf{x})\|^2 \le C_1(1+\|\mathbf{x}\|^2)$, $\forall \mathbf{x} \in \text{supp}(p_{\text{data}}), t \in [0,1]$, where C_1 is a constant, $i = 1,2$;

3. $\|D_i(t_1,\mathbf{x}_1) - D_i(t_2,\mathbf{x}_2)\|^2 \le C_2(\|\mathbf{x}_1-\mathbf{x}_2\| + |t_1-t_2|^{1/2})$, $\forall \mathbf{x}_1,\mathbf{x}_2 \in \text{supp}(p_{\text{data}}), t_1,t_2 \in [0,1]$, where C_2 is another constant;

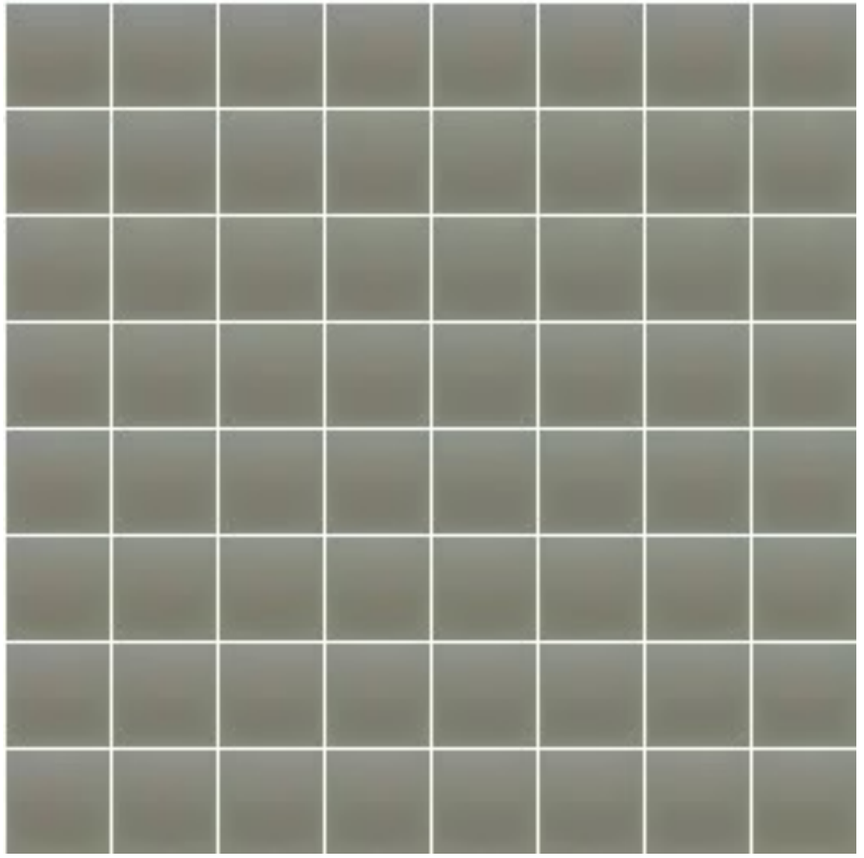
4. $h_{\sigma,\tau}(\mathbf{x}_1,\mathbf{x}_2)$, $\nabla_{\mathbf{x}_1} h_{\sigma,\tau}(\mathbf{x}_1,\mathbf{x}_2)$, p_{data} and ∇p_{data} are L-Lipschitz functions;

$\mathbb{E}\left[\mathcal{W}_2\left(\text{Law}\left(\mathbf{x}_{\text{output}}\right), p_{\text{data}}\right)\right] \to 0$, as $n, N_1, N_2, N_3 \to \infty$, where n is the number of samples used to train the estimators, N_1 and N_2 are number of discretization steps in the Euler-Maruyama method of two stages, N_3 is the sample size for estimating the expectation in the drift term in stage 1.

Experimental results (CIFAR-10)



Oscillation in GANs
Source: https://theaisummer.com/gan-computer-vision/



Our sampling procedure

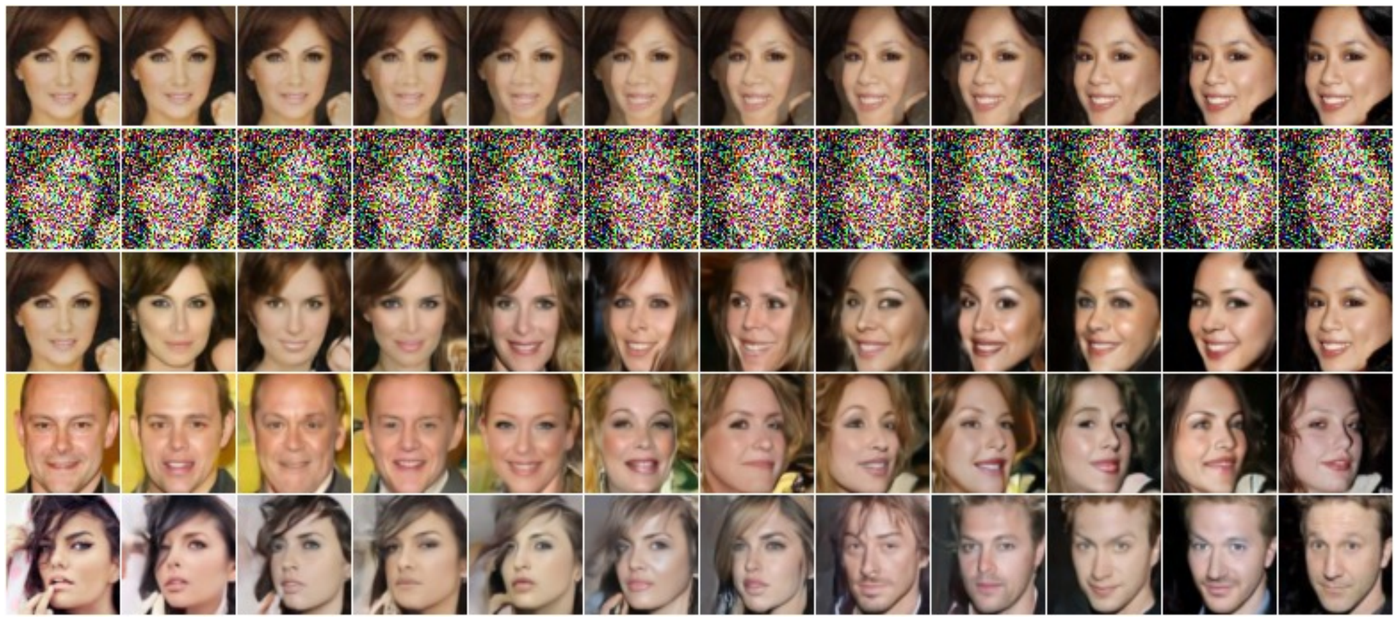
Experimental results (CIFAR-10, CelebA)

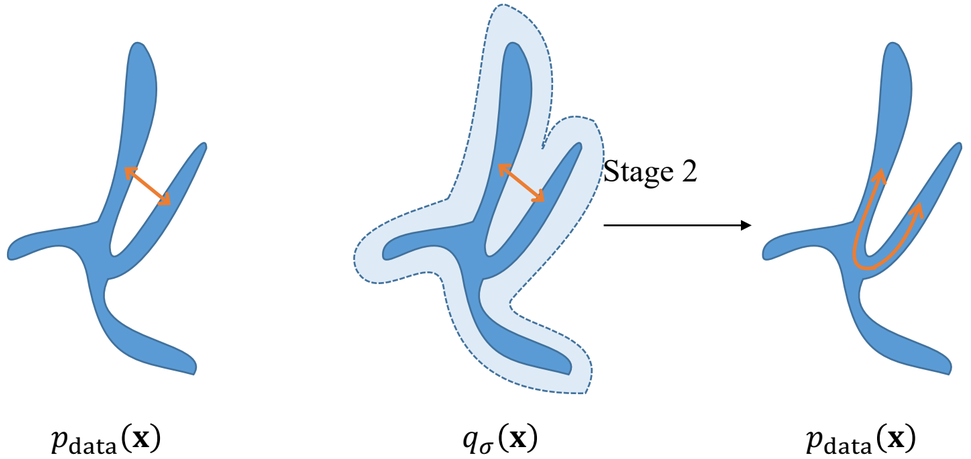
Models	FID	IS
WGAN-GP	36.4	7.86±0.07
SN-SMMDGAN	25.0	7.3±0.1
SNGAN	21.7	8.22±0.05
NCSN	25.32	8.87±0.12
Ours	**12.32**	**8.14±0.07**

FID and Inception Score on CIFAR-10.
FID: lower is better;
IS: higher is better

Some recent diffusion models have achieved comparable or even better performance.

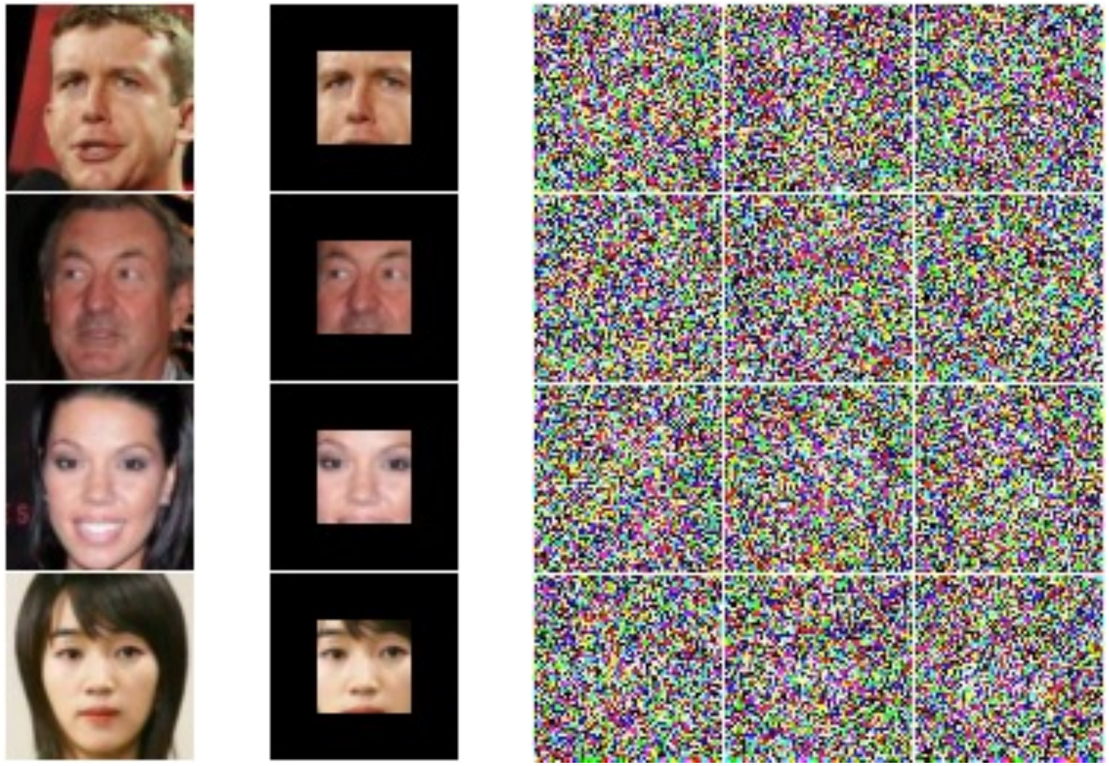
Experimental results (Image interpolation)

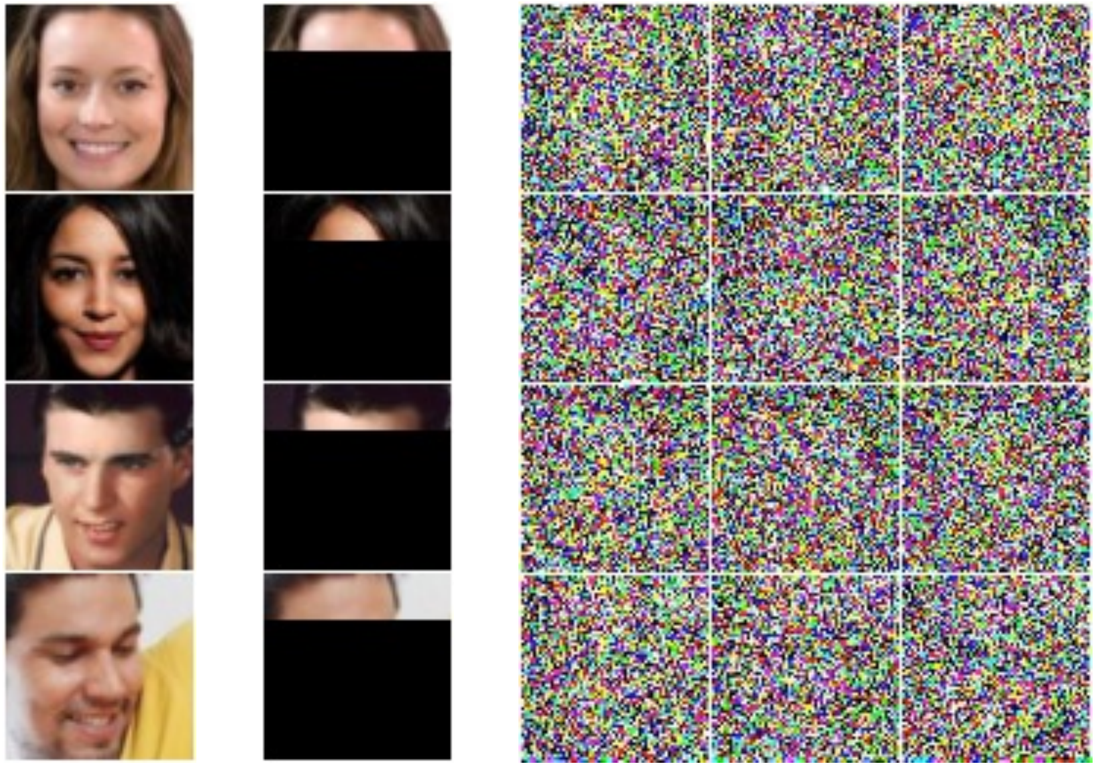




$p_{\text{data}}(\mathbf{x})$ $q_{\sigma}(\mathbf{x})$ $p_{\text{data}}(\mathbf{x})$

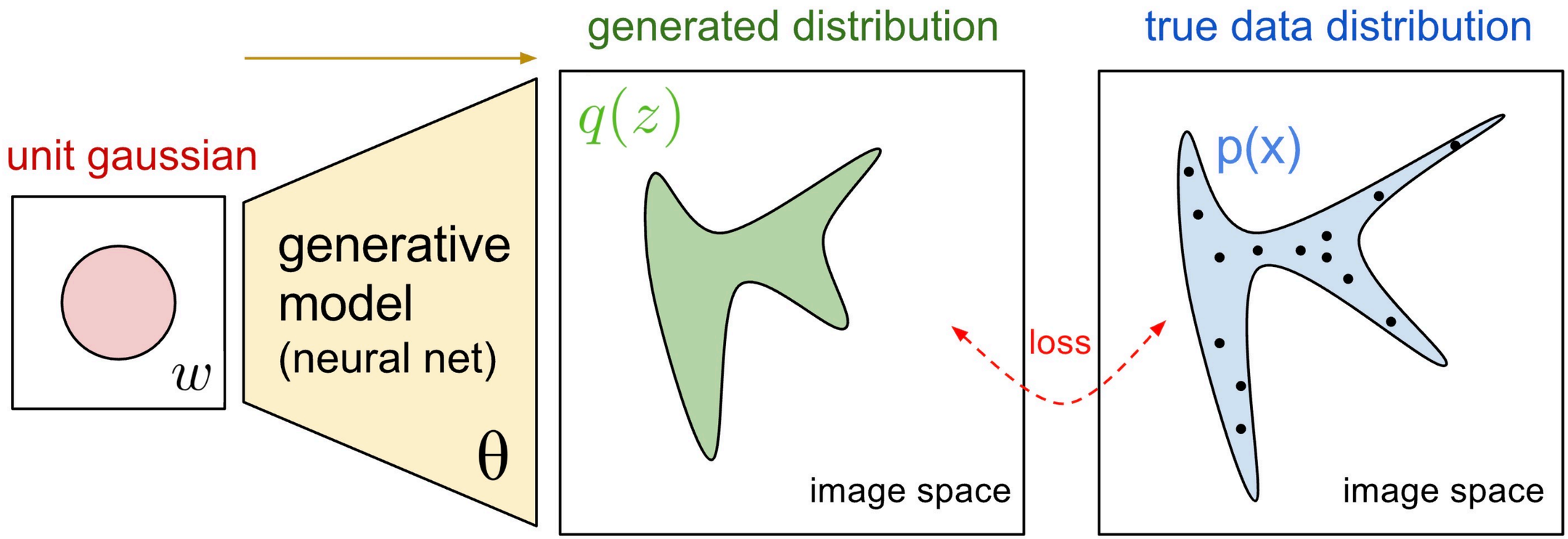
Experimental results (Image inpainting)





Thank you!

Understanding of GAN



$$\min_G D(q(z)||p(x))$$

Variational Gradient Flow (VGrow)

- Consider a batch of particles $\{z_i\}, i = 1, \ldots, n$ with distribution $q(z)$
- Update these particles $\{z_i\}$ by a small amount (preserve continuity),

$$T(z) = z + s \cdot h(z)$$

$$h(x) = -f''(r(x))\nabla r(x)$$

such that the distribution of $\{T(z_i)\}$, denoted as $\tilde{q}(z)$, is closer to $p(x)$, the distribution of $\{x_i\}$

$$D_f(\tilde{q}(z)||p(x)) \leq D_f(q(z)||p(x))$$

Training GAN (Log-D trick)

- Alternating the following two steps:

1. Given the generator G, optimize the discriminator D:

$$\max_D E_{x\sim p_X(x)}[\log D(x)] + E_{w\sim N(0,I)}[\log(1 - D(G(w))]$$

2. Given the discriminator D, optimize the generator G:

$$\min_G E_{x\sim p_X(x)}[\log D(x)] + E_{w\sim N(0,I)}[\log(1 - D(G(w))]$$
$$= \min_G E_{w\sim N(0,I)}[\log(1 - D(G(w))]$$

- Log-D trick:

$$\min_G E_{w\sim N(0,I)}[-\log D(G(w))]$$

Mathematics of GAN: Minimizing Divergence

It is proved that for vanilla GAN

$$\min_G \max_D E_{x\sim p_X}[\log(D(x))] + E_{w\sim N(0,I)}[\log(1 - D\big(G(w)\big)]$$

is equivalent to

$$\min_G D_{JS}(p_X \| q_G)$$

- **WHY does the log-D trick works?**