Deep Generative Models

Tijin Yan

BIT

June 3, 2022

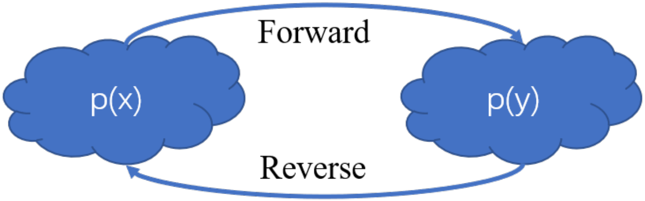
目录

Generative vs Discriminative

Observed data: $\mathbf{x}$. Distribution of the observed data: $\mathbf{p(x)}$.
Target data: $\mathbf{y}$. Distribution of the target data: $\mathbf{p(y)}$.
Ultimate target: $\mathbf{p(y|x)}$

Discriminative Models: Models of the conditional probability $P(Y|X = x)$ of the target y and observation x. $\boldsymbol{p(y|x) = f(x)}$

Generative Models: Statistical models of the joint probability distribution $P(X, Y)$ on given observable variable x and target variable y. $\boldsymbol{p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)}}$.



Overview

Table 1: Overview of some generative models

		steps	transition types	example
VAE	Forward process	1	Stochastic	p(y|x)=f(x)
	Reverse process	1	Stochastic	p(x|y)=g(y)
Flow	Forward process	K	Bijective	p(y|x)=$f_1(f_2 \cdots f_K(x))$
	Reverse process	K	Bijective	p(x|y)=$f_K^{-1}(f_{K-1}^{-1} \cdots f_1^{-1}(y))$
DDPM	Forward process	N	Stochastic	p(y|x)=Markov Chain(x)
	Reverse process	N	Stochastic	p(x|y)=Reverse Markov Chain(y)
Neural ODE	Forward process	∞	Bijective	$p(y\|x) = \int_a^b f(x)$
	Reverse process	∞	Bijective	$p(x\|y) = \int_b^a f(y)$
Neural SDE	Forward process	∞	Stochastic	p(y|x)=SDE(x)
	Reverse process	∞	Stochastic	p(x|y)=SDE(y)
score-sde	Forward process	∞	Stochastic	p(y|x)=SDE(x)
	Reverse process	∞	Stochastic	p(x|y)=Inverse SDE(y)
SurVAE Flow	Forward process	1	Surjective	p(y|x)=surjective f(x)
	Reverse process	1	Surjective	p(x|y)=surjective g(y)
GAN	Reverse process	1		
	Adversarial process	1		

Basic derivation process.

There are two ways for understanding VAE[1]. (1) Introduce $q(y|x)$ to approximate $p(y|x)$.

$$\begin{aligned} D_{KL}(q(y|x) \| p(y|x)) &= \int q(y|x) \log \frac{q(y|x)}{p(y|x)} dx \\ &= \mathbb{E}_{q(y|x)}[\log q(y|x) - \log p(y|x)] \\ &= \mathbb{E}_{q(y|x)}[\log q(y|x) - \log p(x,y)] + \log p(x) \\ &= -\mathcal{L}(x) + \log p(x) \end{aligned} \tag{1}$$

$$\log p(x) = \mathcal{L}(x) + D_{KL}(q(y|x) \| p(y|x)) \geq \mathcal{L}(x)$$

(2) Maximum likelihood estimation.

$$\begin{aligned} \log p(x) &= \log \int p(x|y)p(y)dy \\ &= \log \int \frac{p(x|y)p(y)}{q(y|x)} q(y|x)dy \\ &= \log \mathbb{E}_{q(y|x)}[p(x,y)/q(y|x)] \geq \mathbb{E}_{q(y|x)} \log p(x,y)/q(y|x) = \mathcal{L}(x) \end{aligned} \tag{2}$$

Basic derivation process

$$\begin{aligned}\mathcal{L}(x) &= \mathbb{E}_{q(y|x)} \log p(x,y)/q(y|x)\\ &= \mathbb{E}_{q(y|x)} \log \frac{p(x|y)p(y)}{q(y|x)}\\ &= -\mathbb{E}_{q(y|x)} \log \frac{q(y|x)}{p(y)} + \mathbb{E}_{q(y|x)} \log p(x|y)\\ &= -D_{KL}(q(y|x)|p(y)) + \mathbb{E}_{q(y|x)} \log p(x|y)\end{aligned} \tag{3}$$

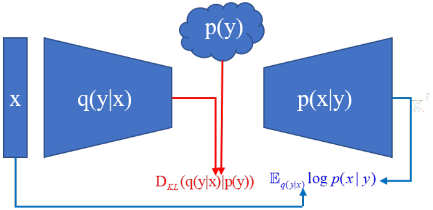


Figure 1: Overall structure of VAE.

Table 2: Examples for various term in VAE.

PDF	Dist	Formulas
p(x|y)	Gaussian	$\mathcal{N}(\mu(y),\sigma(y))$
q(y|x)	Gaussian	$\mathcal{N}(\mu(x),\sigma(x))$
p(y)	Gaussian	$\mathcal{N}(\mathbf{0},\mathbf{I})$
p(x|y)	Bernoulli	$\mathcal{B}(p=g(y))$
q(y|x)	Bernoulli	$\mathcal{B}(p=f(x))$

Noted that parameters of the conditional PDF are obtained by NN when the form of the distribution is determined. The outputs are sampled from the parameterized distribution. Obviously, the transformations of forward and reverse process are stochastic.

Basic derivation process

Assume x, y are D-dimensional vectors. Then given an invertible function f such that f and f^{-1} are both differentiable, $y = f(x) \longrightarrow x = f^{-1}(y)$. With simple derivation, we can get

$$p(y) = p(x) \left| \det \frac{\partial f}{\partial x} \right|^{-1} \tag{4}$$

Similarly, for multiple invertible functions $f_1, f_2, \cdots, f_K$, the transformation can be formulated as

$$\log p(x_K) = \log p(x_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial x_k} \right| \tag{5}$$

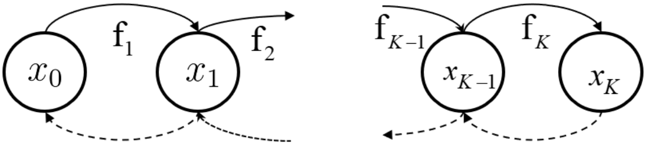


Figure 2: Overall structure of Flow.

Classification

Classification of flow-based generative models[2-3]:

1. Coupling Flows
2. Linear Flows.
3. Autoregressive Flows.
4. Residual Flows.
5. Planar and Radial Flows.
6. Continuous Normalizing Flows.

Coupling flows

Assume x_1, x_2 are partition of data x, g is an invertible map with respect to the first argument given the second, then the general coupling layer[4] can be formulated as

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= g(x_2, m(x_1)) \\ x_1 &= y_1 \\ x_2 &= g^{-1}(y_2, m(y1)) \end{aligned} \tag{6}$$

$$\det \frac{\partial y}{\partial x} = \det \begin{bmatrix} I_d & 0 \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} = \det \frac{\partial y_2}{\partial x_2}$$

Planar and Radial Flows[6]

Planar flows.

$$\begin{aligned} f(z) &= z + uh(w^T z + b) \\ \psi(z) &= h'(w^T z + b)w \\ \det|\frac{\partial f}{\partial z}| &= |\det(I + u\psi(z)^T)| = |1 + u^T \psi(z)| \end{aligned} \tag{7}$$

Sylvester planar flows[5].

Lemma

For $A \in R^{D \times M}, B \in R^{M \times D}$, *we have* $\det(I_D + AB) = \det(I_M + BA)$

Therefore, for the system in Eq. 7,

$$\det|\frac{\partial f}{\partial z}| = |\det(I + u\psi(z)^T)| = \det(I_M + \mathrm{diag}(h')Wu) \tag{8}$$

Besides, QR decomposition is used to accelerate the calculation of det.

Planar and Radial Flows

$$\begin{aligned} z' &= z + QR^A h(R^B Q^T z + b) \\ \det(\frac{\partial z'}{\partial z}) &= \det(I_M + diag(h')R^B R^A) \end{aligned} \tag{9}$$

Radial flows.

$$\begin{aligned} f(z) &= z + \beta h(\alpha, r)(z - z_0) \\ \det \|\frac{\partial f}{\partial z}\| &= [1 + \beta h(\alpha, r)]^{D-1}[1 + \beta h(\alpha, r) + h'(\alpha, r)r] \end{aligned} \tag{10}$$

where $r = |z - z_0|, h(\alpha, r) = 1/(\alpha + r), \alpha, \beta \in R$.

Autoregressive Flows[2]

For a D-dimensional data x, the generalized form of autoregressive flows can be formulated as

$$x'_i = \tau(x_i; h_i), \quad \text{where} \quad h_i = c_i(x_{<i}) \tag{11}$$

where τ is termed the *transformer* and c_i the i-th *conditioner*. $\tau_i$is strictly monotonic function w.r.t. z_i and therefore invertible.

$$J_{f_\phi}(x) = \begin{bmatrix} \frac{\partial \tau}{\partial x_1}(x_1; h_1) & & 0 \\ & \ddots & \\ L(x) & & \frac{\partial \tau}{\partial x_D}(x_D; h_D) \end{bmatrix} \tag{12}$$

Therefore, we can get

$$\log|\det J_f(x)| = \log|\Pi_{i=1}^{D} \frac{\partial \tau}{\partial x_i}(x_i; h_i)| = \sum_{i=1}^{D} \log|\frac{\partial \tau}{\partial x_i}(x_i; h_i)| \tag{13}$$

Autoregressive Flows.

Classification of transformers[2]:

1. Affine AF[4,7-10].
2. Non-affine neural transformers.[11-13]
3. Integration-based transformers.[14-15]
4. Neural spline flows.[16-17]

Classification of conditioners:

1. Recurrent AF.
2. Masked AF.
3. Coupling layers.

Residual Flows

$$y = x + g(x) \tag{14}$$

1. Make the system invertible $\longrightarrow \mathrm{Lip}(g) < 1$.
2. Obtain the inverse function.

$$\begin{gathered} J_F = I + \frac{\partial g}{\partial x} = I + J_g \\ \log|\det(J_F)| = \log\det(I + J_g) = \mathrm{Tr}(\log(I + J_g)) \end{gathered} \tag{15}$$

The second term is based on the identity that $\log\det(A) = \mathrm{Tr}(\log A)$. Then we can the truncated version of Eq. 16 to estimate $\log(I + J_g)$.

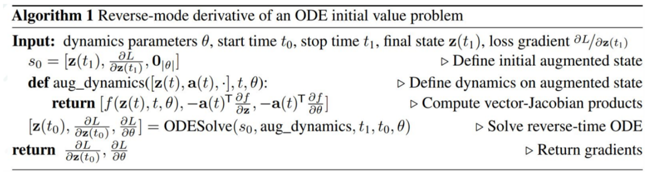
$$\log(I + J_g) = \sum_{n=1}^{\infty} (-1)^{(n-1)} \frac{J_g^n}{n} \tag{16}$$

Continuous Normalizing Flows

Neural ODE [18-19]

$$
\begin{aligned}
\frac{\partial h}{\partial t} &= f(h(t), t, \theta) \\
\frac{d \log p(h(t))}{dt} &= -\mathrm{Tr}\left(\frac{df}{dz(t)}\right) = -\mathbb{E}_{p(\epsilon)}\left[\epsilon^T \frac{df}{dz(t)} \epsilon\right]
\end{aligned}
\tag{17}
$$

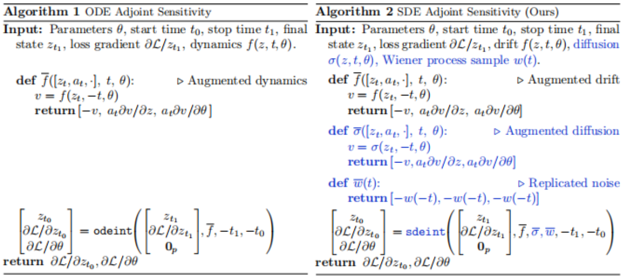
Compared with discrete NF models, the forward process and reverse process are continuous. Besides[18] proposes adjoint methods for parameter training instead of BP.



```
Algorithm 1 Reverse-mode derivative of an ODE initial value problem
Input: dynamics parameters $\theta$, start time $t_0$, stop time $t_1$, final state $\mathbf{z}(t_1)$, loss gradient $\partial L/\partial \mathbf{z}(t_1)$
  $s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$                ▷ Define initial augmented state
  def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta$):          ▷ Define dynamics on augmented state
      return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \theta}]$       ▷ Compute vector-Jacobian products
  $[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}]$ = ODESolve($s_0$, aug_dynamics, $t_1, t_0, \theta$)      ▷ Solve reverse-time ODE
return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$                                        ▷ Return gradients
```

Figure 3: Training process of Neural ODE.

[20-21]



```
Algorithm 1 ODE Adjoint Sensitivity
Input: Parameters $\theta$, start time $t_0$, stop time $t_1$, final
  state $z_{t_1}$, loss gradient $\partial\mathcal{L}/z_{t_1}$, dynamics $f(z,t,\theta)$.

  def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$:        ▷ Augmented dynamics
      $v = f(z_t, -t, \theta)$
      return $[-v,\ a_t \partial v/\partial z,\ a_t \partial v/\partial\theta]$

  $\begin{bmatrix} z_{t_0} \\ \partial\mathcal{L}/\partial z_{t_0} \\ \partial\mathcal{L}/\partial\theta \end{bmatrix} = \texttt{odeint}\left(\begin{bmatrix} z_{t_1} \\ \partial\mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, -t_1, -t_0\right)$
return $\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta$
```

```
Algorithm 2 SDE Adjoint Sensitivity (Ours)
Input: Parameters $\theta$, start time $t_0$, stop time $t_1$, final
  state $z_{t_1}$, loss gradient $\partial\mathcal{L}/z_{t_1}$, drift $f(z,t,\theta)$, diffusion
  $\sigma(z,t,\theta)$, Wiener process sample $w(t)$.

  def $\overline{f}([z_t, a_t, \cdot],\ t,\ \theta)$:        ▷ Augmented drift
      $v = f(z_t, -t, \theta)$
      return $[-v,\ a_t \partial v/\partial z,\ a_t \partial v/\partial\theta]$

  def $\overline{\sigma}([z_t, a_t, \cdot],\ t,\ \theta)$:        ▷ Augmented diffusion
      $v = \sigma(z_t, -t, \theta)$
      return $[-v, a_t \partial v/\partial z, a_t \partial v/\partial\theta]$

  def $\overline{w}(t)$:        ▷ Replicated noise
      return $[-w(-t), -w(-t), -w(-t)]$

  $\begin{bmatrix} z_{t_0} \\ \partial\mathcal{L}/\partial z_{t_0} \\ \partial\mathcal{L}/\partial\theta \end{bmatrix} = \texttt{sdeint}\left(\begin{bmatrix} z_{t_1} \\ \partial\mathcal{L}/\partial z_{t_1} \\ \mathbf{0}_p \end{bmatrix}, \overline{f}, \overline{\sigma}, \overline{w}, -t_1, -t_0\right)$
return $\partial\mathcal{L}/\partial z_{t_0}, \partial\mathcal{L}/\partial\theta$
```

Figure 4: Comparison between adjoint and SDE adjoint sensitivity.

Applications

1. Irregularly TS forecasting[22-29].

Definition

Assume $y = f_\theta(x)$, then we can define a Energy function $E(x, y)$, which obtains minimum/maximum value when $y - f_\theta(x) \to 0/\infty$. EBMs usually obtain the PDF $p(x)$ based on the Energy function.

$$p(x) = \frac{\exp[f_\theta(x)]}{z_\theta} = \frac{\exp[-E_\theta(x)]}{z_\theta}, z_\theta = \int \exp[f_\theta(x)]dx \tag{18}$$

As shown in Eq. 18, the PDF involves a typically intractable integral. Thus directly optimization with maximum likelihood method is difficult. There are now several methods for training EBM[30]:

1. Maximum likelihood training with MCMC.
2. Score matching.
3. Noise contrastive estimation.
4. Other methods (To be explained further).

ML Training with MCMC

Maximizing likelihood is equivalent to minimizing the KL divergence between $p(x)$ and $p_\theta(x)$, because

$$E_{x\sim p(x)}[\log p_\theta(x)] = D_{KL}[p(x)\|p_\theta(x)] - \mathbb{E}_{x\sim p(x)}\log p(x) \tag{19}$$

The second term is a constant. Although directly compute the likelihood is difficult, we can estimate the gradient of the log-likelihood with MCMC.

$$\begin{aligned}\nabla_\theta \log p_\theta(x) &= -\nabla_\theta E_\theta(x) - \nabla_\theta \log z_\theta \\ &= -\nabla_\theta E_\theta(x) - \mathbb{E}_{x\sim p_\theta(x)}[-\nabla_\theta E_\theta(x)]\end{aligned} \tag{20}$$

The first term can be obtained by automatic differentiation and the second term can be estimated with Monte Carlo method.

Score matching

Hyvarinen[31] minimize the Fisher divergence between $p(x)$ and $p_\theta(x)$.

$$
\begin{aligned}
&D_F(p(x)\|p_\theta(x)) = E_{p(x)}[\frac{1}{2}\|\nabla_x \log p(x) - \nabla_x \log p_\theta(x)\|^2] \\
&= \frac{1}{2}\int p(x)[(\nabla_x \log p(x))^2 - 2\nabla_x \log p(x)\nabla_x \log p_\theta(x) + (\nabla_x \log p_\theta(x))^2] \\
&= \mathrm{C} - \int p(x)\frac{1}{p(x)}\nabla_x p(x)\nabla_x \log p_\theta(x) + \frac{1}{2}\int p(x)(\nabla_x \log p_\theta(x))^2 \\
&= \mathrm{C} - p(x)\nabla_x \log p_\theta(x)|_{-\infty}^{\infty} + \int p(x)\nabla_x^2 \log p_\theta(x) + \frac{1}{2}\int p(x)(\nabla_x \log p_\theta(x))^2 \\
&= E_{p(x)}[\frac{1}{2}(\nabla_x \log p_\theta(x))^2 + \nabla_x^2 \log p_\theta(x)]
\end{aligned}
\tag{21}
$$

Score matching

Denoising score matching

The objective in Eq. 21 requires $\log p(x)$ should be continuously differentiable and finite everywhere. However, these conditions may not hold in practice. To alleviate the difficulty, we can add a little bit noise to each data point $\tilde{x} = x + \epsilon$. As long as $p(\epsilon)$ is smooth, $q(\tilde{x}) = \int q(\tilde{x}|x)p(x)dx$ is also smooth.

$$\begin{aligned} D_F(q(\tilde{x})\|p_\theta(\tilde{x})) &= \mathbb{E}_{q(\tilde{x})}[\frac{1}{2}\|_x \log q(\tilde{x}) - \nabla_x \log p_\theta(\tilde{x})\|_2^2] \\ &= \mathbb{E}_{q(x,\tilde{x})}[\frac{1}{2}\|_x \log q(\tilde{x}|x) - \nabla_x \log p_\theta(\tilde{x})\|_2^2] + C \end{aligned} \tag{22}$$

Score matching
Sliced score matching

Although DSM avoids the expensive computation of second-order derivatives, the optimal EBM that minimizes the DSM objective corresponds to $p(\tilde{x})$, not the $p(x)$. Sliced score matching[32] randomly samples a project vector v, takes the inner product between v and two scores, and thencompare the resulting two scalars. The objective is to optimize the sliced Fisher divergence

$$D_{SF}(p(x)\|p_\theta(x)) = \mathbb{E}_{p(x)}\mathbb{E}_{p(v)}[\frac{1}{2}(v^T\nabla_x \log p(x) - v^T\nabla_x \log p_\theta(x))^2] \qquad (23)$$

Score matching

Score matching has difficulty in estimating the relative weights of two modes separated by large low-density regions[33]. Song[33-34] overcomes this difficulty by perturbing training data with different scales of noise and learn a score model for each scale. As for sampling, they use annealed Langevin dynamics or leveraging reverse diffusion processes[35].

Score matching

NCSN & NCSN v2

Score matching with Langevin dynamics. $p_\sigma(\tilde{x}|x) = \mathcal{N}(x, \sigma^2 I)$ is the perturbation kernel. Consider a noise sequence with ascending order $\sigma_{1:T}$. σ_1 is small enough that $p_{\sigma_1} \approxeq p(x)$, σ_T is big enough that $p_{\sigma_T} \approx \mathcal{L}(0, \sigma_T^2 I)$. The loss is a weighted sum of denoising score matching objective.

$$\arg\min_\theta \sum_{i=1}^{T} \sigma_i^2 E_{p(x)} E_{p_{\sigma_i}(\tilde{x}|x)} \left[\| s_\theta(\tilde{x}, \sigma_i) - \nabla_{\tilde{x}} \log p_{\sigma_i}(\tilde{x}|x) \|_2^2 \right] \tag{24}$$

The sampling process depends on Langevin MCMC,

$$x_i^m = x_i^{m-1} + \epsilon_i s_{\theta^*}(x_i^{m-1}, \sigma_i) + \sqrt{2\epsilon_i} z_i^m, m = 1, \cdots, M \tag{25}$$

NCSN v2 gives the settings of noise scales, step size and the number of sampling steps per noise scale. It also propose the way that s_θ incorporates information of σ.

Score matching

DDPM

Define the following Markov chains:

$$
\begin{aligned}
q(x_{1:T}|x_0) &= \Pi_{t=1}^{T} q(x_t|x_{t-1}) \\
p_\theta(x_{0:T}) &= p(x_T)\Pi_{t=1}^{T} p_\theta(x_{t-1}|x_t)
\end{aligned}
\tag{26}
$$

Given ascending noise scales $\beta_{1:T} \in [0, 1]$, $\alpha_{1:T} = 1 - \beta_{1:T}$, $\bar{\alpha}_{1:T} = \Pi_{s=1}^{t} \alpha_t$, the transition functions are defined as

$$
\begin{aligned}
q(x_t|x_{t-1}) &:= \mathcal{N}(\sqrt{1-\beta_t} x_{t-1}, \beta_t I) \\
q(x_t|x_0) &:= \mathcal{N}(\sqrt{\bar{\alpha}_t} x_0, (1-\bar{\alpha}_t) I) \\
q(x_{t-1}|x_t, x_0) &:= \mathcal{N}(\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I)
\end{aligned}
\tag{27}
$$

where $\tilde{\mu}_t = \frac{\sqrt{\bar{\alpha}_{t-1}\beta_t}}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t, \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. Similarly, we can construct an reverse Markov chain,

$$
\begin{aligned}
p_\theta(x_{t-1}|x_t) &:= \mathbb{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \\
\mu_\theta(x_t, t) &= \tilde{\mu}_t(x_t, (x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta)/\sqrt{\bar{\alpha}_t})
\end{aligned}
\tag{28}
$$

Score matching

DDPM

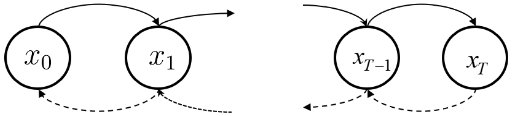
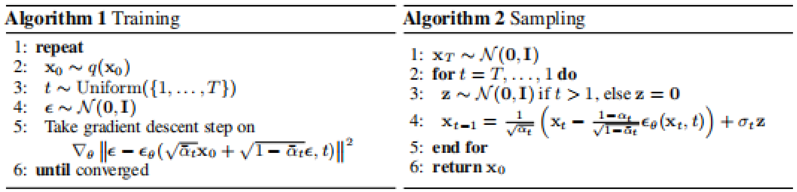


Figure 5: Overall structure of DDPM

$$L = \mathbb{E}\left[D_{KL}(q(x_T|x_0)\|p(x_T)) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t)) - \log p_\theta(x_0|x_1)\right]$$

The loss can be simplified as $\mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar\alpha_t}x_0 + \sqrt{1-\bar\alpha_t}\epsilon, t)\|^2\right]$.



Algorithm 1 Training

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4:   $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\|\epsilon - \epsilon_\theta(\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\epsilon, t)\right\|^2$
6: until converged
```

Algorithm 2 Sampling

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

Figure 6: Training and sampling process of DDPM.

Score matching

DDIM[36] changes the Markov chain in DDPM and makes x_t depends on x_{t-1} and x_0.

$$\begin{aligned} q(x_{1:T}|x_t, x_0) &= q(x_T|x_0)\Pi_{t=2}^{T} q(x_{t-1}|x_t, x_0) \\ q(x_t|x_0) &= \mathcal{N}(\sqrt{\alpha_t}x_0, (1-\alpha_t)I) \\ q(x_{t-1}|x_t, x_0) &= \mathcal{N}(\sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}, \sigma^2 I) \end{aligned} \tag{29}$$

Noted that in DDIM, α_t is different from that of DDPM. Compared with DDPM, the forward process is the same while the reverse process can be formulated as

$$x_{t-1} = \sqrt{\alpha_{t-1}}(\frac{x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}}) + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\epsilon_\theta(x_t, t) + \sigma_t \epsilon_t \tag{30}$$

Score matching

score-sde [37]

Eq. 31a and Eq. 31b gives a SDE and the reverse-time SDE. $\bar{w}$ is a standard Wiener process when time flows backwards from T to 0.

$$dx = f(x,t)dt + g(t)dw \tag{31a}$$

$$dx = [f(x,t) - g(t)^2 \nabla_x \log p_t(x)]dt + g(t)d\bar{w} \tag{31b}$$

The function form in DDPM and NCSN can be regarded as the discrete form of stochastic differential equations.

$$\begin{aligned} q(x_t|x_{t-1}) &:= \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t I) \rightarrow dx = -0.5\beta(t)xdt + \sqrt{\beta(t)}dw \\ p_\sigma(\tilde{x}|x) &:= \mathcal{N}(x, \sigma^2 I) \rightarrow dx = \sqrt{\frac{d\sigma^2(t)}{dt}}dw \end{aligned} \tag{32}$$

The loss function can be unified as

$$\arg\min_\theta E_t \left\{\lambda(t) E_{x(0)} E_{x(t)|x(0)} [\| s_\theta(x(t),t) - \nabla_{x(t)} \log p_{0t}(x(t)|x(0)) \|_2^2]\right\} \tag{33}$$

Score matching

SGM_CLD[38]

$$\begin{bmatrix} dx_t \\ dv_t \end{bmatrix} = \begin{bmatrix} M^{-1}v_t \\ -x_t \end{bmatrix} \beta dt + \begin{bmatrix} 0 \\ -\Gamma M^{-1} v_t \end{bmatrix} \beta dt + \begin{bmatrix} 0 \\ \sqrt{2\Gamma\beta} dw_t \end{bmatrix} \tag{34}$$

When $\Gamma^2 = 4M$, an ideal balance is achieved and convergence to $\mathcal{N}(0, I)$ as fast as possible in a smooth manner. $u_t = (x_t, v_t)^T$, the score matching objective can be simplified as

$$\min_{\theta} E_{t\in[0,T]} E_{u_t \sim p_t(u_t)} [\lambda(t) \| s_{\theta(u_t,t)} - \nabla_{v_t} \log p_t(u_t) \|_2^2] \tag{35}$$

Noted that the objective requires only the velocity gradient of the log-density of the joint distribution. The sampling process can be formulated as

$$\begin{bmatrix} d\tilde{x}_t \\ d\tilde{v}_t \end{bmatrix} = \begin{bmatrix} -M^{-1}\tilde{v}_t \\ \tilde{x}_t \end{bmatrix} \beta dt + \begin{bmatrix} 0 \\ -\Gamma M^{-1} \tilde{v}_t \end{bmatrix} \beta dt + \begin{bmatrix} 0 \\ \sqrt{2\Gamma\beta} dw_t \end{bmatrix} + \begin{bmatrix} 0 \\ 2\Gamma[s(\tilde{u}, T-t) + M^{-1}\tilde{v}_t] \end{bmatrix} \beta dt \tag{36}$$

Score matching

Directions for existing works.

1. Acceleration for DDPM[39-43].
2. Change distributions.[44-47]
3. Continuous EBMs.[38,42,48]
4. Combine with other generative models.[42,49-52]
5. Various applications (To be explained further).

Noise contrastive estimation(NCE)

We can learn an EBM by contrasting it with another distribution with known density. For example, given target distribution $p(y) \sim \mathcal{N}(0,1)$ and a binary variable b, we can construct a mixture distribution as $p_{mix}(x) = p(b=0)p(y) + p(b=1)p(x)$. According to Bayes's rule,

$$p_{mix}(b=0|x) = \frac{p_{mix}(x|b=0)p(b=0)}{p_{mix}(x)} = \frac{p(y)}{p(y)+vp(x)} \tag{37}$$

, where $v = p(b=1)/p(b=0)$. Suppose the EBM $p_\theta(x)$ satisfies Eq. 18. Then similarly we can get

$$p_{mix,\theta}(b=0|x) = \frac{p(y)}{p(y)+vp_\theta(x)} \tag{38}$$

In NCE, we indirectly fit $p_\theta(x)$ to $p(x)$ by fitting $p_{mix,\theta}(b|x)$ to $p_{mix}(b|x)$ with a conditional ML objective:

$$\mathbb{E}_{p_{mix}(x)} D_{KL}[p_{mix,\theta}(b|x) \| p_{mix}(b|x)] \tag{39}$$

1

[1] NCE is equals that of sliced score matching[32].

Other methods for EBM.

1. Minimizing differences/derivatives of KL divergence. Minimum velocity learning[53], minimum probability flow[54], minimum KL contraction[55].
2. Minimizing the Stein Discrepancy.
3. Adversarial training.

Applications

1. Image generation[56-57].
2. Text to Speech[58-64].
3. Time series prediction/ anomaly detection[65-67].
4. Graph generation/prediction[68-69].

Derivation

GAN[70] is composed of a Generator and a Discriminator. The Generator is to establish connection between $p(y)$ and $p(x)$ while the Discriminator is used to judge whether the input is a real sample or a sample that generated from Generator.

$$\min_G \max_D V(D,G) = E_{x\sim p(x)}[\log D(x)] + E_{y\sim p(y)}[\log(1 - D(G(y)))] \tag{40a}$$

$$\min_G \max_D V(D,G) = E_{x\sim p(x)}[\log D(x)] - E_{y\sim p(y)}[\log(D(G(y)))] \tag{40b}$$

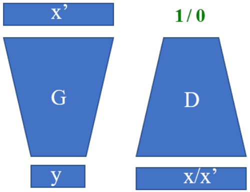


Figure 7: Overall structure of GAN

Derivation

Global optimum: $p_g(x) = p(x)$. We can get the optimal Discriminator $D_{G^*} = \frac{p(x)}{p(x)+p_g(x)}$ for a given Generator G^* (Obtain extreme value based on the derivative of Eq. 41). Then Eq. 40a can be reformulated as Eq. 42.

$$\begin{aligned} V(G,D) &= \int_x p(x)\log D(x)dx + \int_y p(y)\log(1-D(G(y)))dy \\ &= \int_x p(x)\log D(x) + p_g(x)\log(1-D(x))dx \end{aligned} \tag{41}$$

$$\begin{aligned} C(G) &= V(G,D) \\ &= E_{x\sim p(x)}[\log D_G^*(x)] + E_{x\sim p_g}[1-\log D_G^*(x)] \\ &= E_{x\sim p(x)}[\frac{p(x)}{p(x)+p_g(x)}] + E_{x\sim p_g}[\frac{p_g(x)}{p_{data}(x)+p_g(x)}] \\ &= -\log 4 + KL(p(x)||\frac{p(x)+p_g(x)}{2}) + KL(p_g(x)||\frac{p(x)+p_g(x)}{2}) \\ &= -\log 4 + 2JS(p(x)||p_g(x)) \end{aligned} \tag{42}$$

Stability
WGAN

Problems in GAN[71]:

1. Better Discriminator leads to vanishing gradient problem in Generator.
2. Optimize Eq. 40b is equal to optimize $KL(p_g|p) - 2JS(p|p_g)$, which leads to unstable training process.

WGAN[72] proposes to use Wasserstein Distance to replace original KL divergence.

$$W(P, P_g) = \inf_{\gamma \in \Pi(P, P_g} E_{(x,y)\sim\gamma}[||x - y|| \tag{43a}$$

$$W(p, p_g) = \frac{1}{K} \sup_{||f||_L \leq K} E_{x\sim p}[f(x)] - E_{x\sim p_g}[f(x)] \tag{43b}$$

Improvement compared with GAN:

1. Remove *sigmoid* in D.
2. Remove log in loss function.
   $\min_G \max_D V(D, G) = E_{x\sim p(x)}[D(x)] - E_{y\sim p(y)}[D(G(y))]$
3. Clip gradients of D.
4. Do not use momentum based optimizers.

Stability
WGAN-GP

Problems in WGAN:

1. Weight clipping pushes weights towards two values(the extremes of the clipping range)
2. Weight clipping method will cause gradients either explode or vanish, which depends on the threshold of weight clipping.

WGAN-GP[73] uses gradient penalty to satisfy the lipschitz constraint in D.

$$L(D) = -E_{x\sim p(x)}[D(x)] + E_{x\sim p_g}[D(x)] + \lambda E_{x\sim P_{\hat{x}}}[||\nabla_x D(x)||_p - 1]^2 \tag{44}$$

where $\hat{x} = \epsilon x_r + (1-\epsilon)x_g, x_r \sim p(x), x_g \sim p_g, \epsilon \sim U[0,1]$.

Stability

Spectral-Norm[74]

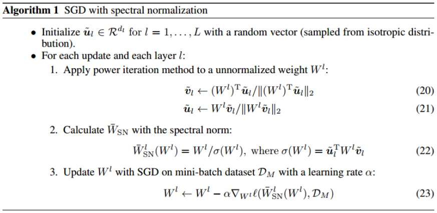
Lemma

The necessary and sufficient condition for a matrix satisfing the K-Lipschtiz is $\|Ax\| \leq K\|x\|$.

Lemma

The spectral-norm of matrix A *is the maximum eigenvalues of* $A^T A$.

Instead of using SVD, we use power iteration method to calculate the spectral norm. Then we add spectral-norm to each layer of D.



```
Algorithm 1 SGD with spectral normalization
    • Initialize $\tilde{u}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
    • For each update and each layer $l$:
        1. Apply power iteration method to a unnormalized weight $W^l$:
               $\tilde{v}_l \leftarrow (W^l)^{\mathrm{T}}\tilde{u}_l / \|(W^l)^{\mathrm{T}}\tilde{u}_l\|_2$        (20)
               $\tilde{u}_l \leftarrow W^l\tilde{v}_l / \|W^l\tilde{v}_l\|_2$        (21)
        2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:
               $\bar{W}^l_{\mathrm{SN}}(W^l) = W^l/\sigma(W^l)$, where $\sigma(W^l) = \tilde{u}_l^{\mathrm{T}} W^l \tilde{v}_l$        (22)
        3. Update $W^l$ with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate $\alpha$:
               $W^l \leftarrow W^l - \alpha\nabla_{W^l}\ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M)$        (23)
```

Figure 8: spectral_normalization

Stability

DiracGAN[75]

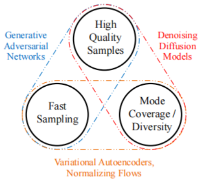
The optimization of GAN can be regard as a stochastic dynamic system. $\dot{\phi} = \nabla_\phi L_1(\phi, \theta), \dot{\theta} = -\nabla_\theta L_2(\phi, \theta)$.

If we have only one real data $\mathbf{0}$ while the fake data is θ. $D(x) = \sigma(x\phi)$. Then for vanilla GAN, $L_1 = -\log(1 - \sigma(\theta\phi)), L_2 = -\log\sigma(\theta\phi)$. The corresponding dynamic system can be formulated as

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \nabla_\phi \log(1 - \sigma(\theta\phi)) \\ \nabla_\theta - \log\sigma(\theta\phi) \end{bmatrix} = \begin{bmatrix} -\sigma(\theta\phi)\theta \\ (1 - \sigma(\theta\phi))\phi \end{bmatrix} \tag{45}$$

Eq. 45 can not converge to 0. Similarly, we can analyze the convergence of other GAN models. Details can be found at kexue.fm.

Denoising diffusion GAN



High Quality Samples
Generative Adversarial Networks
Denoising Diffusion Models
Fast Sampling
Mode Coverage / Diversity
Variational Autoencoders, Normalizing Flows

Figure 1: Generative learning trilemma.

Figure 9: Generative learning trilemma.

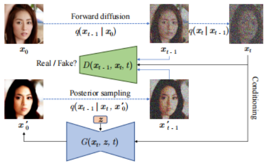


Figure 3: The training process of denoising diffusion GAN.

Figure 10: Training process of Denoising diffuion GAN.

SurVAE Flows [76]

$$\log p(x) = \log p(z) + \mathcal{V}(x,z) + \mathcal{E}(x,z), z \sim q(z|x) \tag{46}$$

where $\mathcal{V}(x,z), \mathcal{E}(x,z)$ are the likelihood contribution and bound looseness terms.

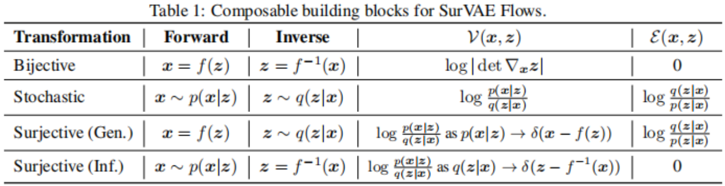


Table 1: Composable building blocks for SurVAE Flows.

Transformation	**Forward**	**Inverse**	$\mathcal{V}(\boldsymbol{x},\boldsymbol{z})$	$\mathcal{E}(\boldsymbol{x},\boldsymbol{z})$
Bijective	$\boldsymbol{x} = f(\boldsymbol{z})$	$\boldsymbol{z} = f^{-1}(\boldsymbol{x})$	$\log \lvert \det \nabla_{\boldsymbol{x}} \boldsymbol{z} \rvert$	0
Stochastic	$\boldsymbol{x} \sim p(\boldsymbol{x}\|\boldsymbol{z})$	$\boldsymbol{z} \sim q(\boldsymbol{z}\|\boldsymbol{x})$	$\log \frac{p(\boldsymbol{x}\|\boldsymbol{z})}{q(\boldsymbol{z}\|\boldsymbol{x})}$	$\log \frac{q(\boldsymbol{z}\|\boldsymbol{x})}{p(\boldsymbol{z}\|\boldsymbol{x})}$
Surjective (Gen.)	$\boldsymbol{x} = f(\boldsymbol{z})$	$\boldsymbol{z} \sim q(\boldsymbol{z}\|\boldsymbol{x})$	$\log \frac{p(\boldsymbol{x}\|\boldsymbol{z})}{q(\boldsymbol{z}\|\boldsymbol{x})}$ as $p(\boldsymbol{x}\|\boldsymbol{z}) \to \delta(\boldsymbol{x} - f(\boldsymbol{z}))$	$\log \frac{q(\boldsymbol{z}\|\boldsymbol{x})}{p(\boldsymbol{z}\|\boldsymbol{x})}$
Surjective (Inf.)	$\boldsymbol{x} \sim p(\boldsymbol{x}\|\boldsymbol{z})$	$\boldsymbol{z} = f^{-1}(\boldsymbol{x})$	$\log \frac{p(\boldsymbol{x}\|\boldsymbol{z})}{q(\boldsymbol{z}\|\boldsymbol{x})}$ as $q(\boldsymbol{z}\|\boldsymbol{x}) \to \delta(\boldsymbol{z} - f^{-1}(\boldsymbol{x}))$	0

Figure 11: Basic ideas of SurVAE flow.

SurVAE Flows

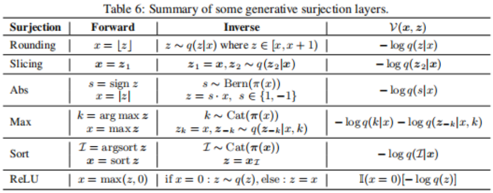


Table 6: Summary of some generative surjection layers.

Surjection	Forward	Inverse	$\mathcal{V}(\boldsymbol{x}, \boldsymbol{z})$
Rounding	$x = \lfloor z \rfloor$	$z \sim q(z\|x)$ where $z \in [x, x+1)$	$-\log q(z\|x)$
Slicing	$\boldsymbol{x} = \boldsymbol{z}_1$	$\boldsymbol{z}_1 = \boldsymbol{x}, \boldsymbol{z}_2 \sim q(\boldsymbol{z}_2\|\boldsymbol{x})$	$-\log q(\boldsymbol{z}_2\|\boldsymbol{x})$
Abs	$s = \text{sign } z$ $x = \|z\|$	$s \sim \text{Bern}(\pi(x))$ $z = s \cdot x, \ \ s \in \{1, -1\}$	$-\log q(s\|x)$
Max	$k = \arg\max \boldsymbol{z}$ $x = \max \boldsymbol{z}$	$k \sim \text{Cat}(\boldsymbol{\pi}(x))$ $z_k = x, \boldsymbol{z}_{-k} \sim q(\boldsymbol{z}_{-k}\|x, k)$	$-\log q(k\|x) - \log q(\boldsymbol{z}_{-k}\|x, k)$
Sort	$\mathcal{I} = \text{argsort } \boldsymbol{z}$ $\boldsymbol{x} = \text{sort } \boldsymbol{z}$	$\mathcal{I} \sim \text{Cat}(\boldsymbol{\pi}(\boldsymbol{x}))$ $\boldsymbol{z} = \boldsymbol{x}_{\mathcal{I}}$	$-\log q(\mathcal{I}\|\boldsymbol{x})$
ReLU	$x = \max(z, 0)$	if $x = 0 : z \sim q(z)$, else : $z = x$	$\mathbb{I}(x = 0)[-\log q(z)]$

Figure 12: Generative surjection layers.

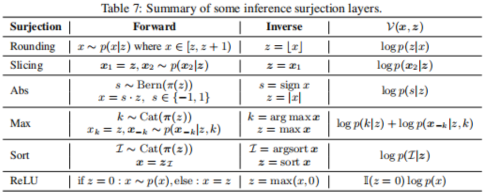
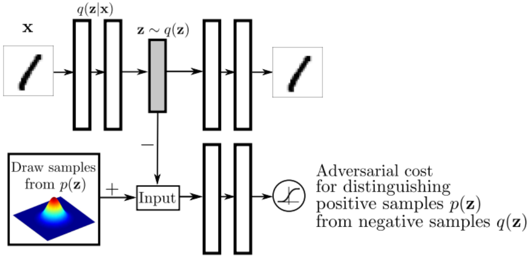


Table 7: Summary of some inference surjection layers.

Surjection	Forward	Inverse	$\mathcal{V}(\boldsymbol{x}, \boldsymbol{z})$
Rounding	$x \sim p(x\|z)$ where $x \in [z, z+1)$	$z = \lfloor x \rfloor$	$\log p(z\|x)$
Slicing	$\boldsymbol{x}_1 = \boldsymbol{z}, \boldsymbol{x}_2 \sim p(\boldsymbol{x}_2\|\boldsymbol{z})$	$\boldsymbol{z} = \boldsymbol{x}_1$	$\log p(\boldsymbol{x}_2\|\boldsymbol{z})$
Abs	$s \sim \text{Bern}(\pi(z))$ $x = s \cdot z, \ \ s \in \{-1, 1\}$	$s = \text{sign } x$ $z = \|x\|$	$\log p(s\|z)$
Max	$k \sim \text{Cat}(\boldsymbol{\pi}(z))$ $x_k = z, \boldsymbol{x}_{-k} \sim p(\boldsymbol{x}_{-k}\|z, k)$	$k = \arg\max \boldsymbol{x}$ $z = \max \boldsymbol{x}$	$\log p(k\|z) + \log p(\boldsymbol{x}_{-k}\|z, k)$
Sort	$\mathcal{I} \sim \text{Cat}(\boldsymbol{\pi}(\boldsymbol{z}))$ $\boldsymbol{x} = \boldsymbol{z}_{\mathcal{I}}$	$\mathcal{I} = \text{argsort } \boldsymbol{x}$ $\boldsymbol{z} = \text{sort } \boldsymbol{x}$	$\log p(\mathcal{I}\|\boldsymbol{z})$
ReLU	if $z = 0 : x \sim p(x)$, else : $x = z$	$z = \max(x, 0)$	$\mathbb{I}(z = 0) \log p(x)$

Figure 13: Inference surjection layers.

AAE[77]



$\mathbf{x}$
$q(\mathbf{z}|\mathbf{x})$
$\mathbf{z} \sim q(\mathbf{z})$
$-$
Draw samples from $p(\mathbf{z})$
$+$
Input
Adversarial cost for distinguishing positive samples $p(\mathbf{z})$ from negative samples $q(\mathbf{z})$

Figure 14: Architecture of AAE.

References I

[1] Kingma D P, Welling M. Auto-encoding variational bayes[J]. ArXiv preprint arXiv:1312.6114, 2013.

[2] Papamakarios G, Nalisnick E, Rezende D J, et al. Normalizing flows for probabilistic modeling and inference[J]. Journal of Machine Learning Research, 2021, 22(57): 1-64.

[3] Kobyzev I, Prince S J, Brubaker M A. Normalizing flows: An introduction and review of current methods[J]. IEEE transactions on pattern analysis and machine intelligence, 2020, 43(11): 3964-3979.

[4] Dinh L, Krueger D, Bengio Y. Nice: Non-linear independent components estimation[J]. ArXiv preprint arXiv:1410.8516, 2014.

[5] Berg R v d, Hasenclever L, Tomczak J M, et al. Sylvester normalizing flows for variational inference[J]. ArXiv preprint arXiv:1803.05649, 2018.

[6] Rezende D, Mohamed S. Variational inference with normalizing flows[C] //International conference on machine learning. 2015: 1530-1538.

References II

[7] Kingma D P, Salimans T, Jozefowicz R, et al. Improved variational inference with inverse autoregressive flow[J]. Advances in neural information processing systems, 2016, 29.

[8] Dinh L, Sohl-Dickstein J, Bengio S. Density estimation using real nvp[J]. ArXiv preprint arXiv:1605.08803, 2016.

[9] Papamakarios G, Pavlakou T, Murray I. Masked autoregressive flow for density estimation[J]. Advances in neural information processing systems, 2017, 30.

[10] Kingma D P, Dhariwal P. Glow: Generative flow with invertible 1x1 convolutions[J]. Advances in neural information processing systems, 2018, 31.

[11] Huang C W, Krueger D, Lacoste A, et al. Neural autoregressive flows[C] //International Conference on Machine Learning. 2018: 2078-2087.

References III

[12] Ho J, Chen X, Srinivas A, et al. Flow++: Improving flow-based generative models with variational dequantization and architecture design[C]//International Conference on Machine Learning. 2019: 2722-2730.

[13] De Cao N, Aziz W, Titov I. Block neural autoregressive flow[C]// Uncertainty in artificial intelligence. 2020: 1263-1273.

[14] Jaini P, Selby K A, Yu Y. Sum-of-squares polynomial flow[C]// International Conference on Machine Learning. 2019: 3009-3018.

[15] Wehenkel A, Louppe G. Unconstrained monotonic neural networks[J]. Advances in neural information processing systems, 2019, 32.

[16] Müller T, McWilliams B, Rousselle F, et al. Neural importance sampling[J]. ACM Transactions on Graphics (TOG), 2019, 38(5): 1-19.

[17] Durkan C, Bekasov A, Murray I, et al. Neural spline flows[J]. Advances in neural information processing systems, 2019, 32.

References IV

[18] Chen R T, Rubanova Y, Bettencourt J, et al. Neural ordinary differential equations[J]. Advances in neural information processing systems, 2018, 31.

[19] Grathwohl W, Chen R T, Bettencourt J, et al. Ffjord: Free-form continuous dynamics for scalable reversible generative models[J]. ArXiv preprint arXiv:1810.01367, 2018.

[20] Li X, Wong T K L, Chen R T, et al. Scalable gradients for stochastic differential equations[C]//International Conference on Artificial Intelligence and Statistics. 2020: 3870-3882.

[21] Liu X, Xiao T, Si S, et al. Neural sde: Stabilizing neural ode networks with stochastic noise[J]. ArXiv preprint arXiv:1906.02355, 2019.

[22] Rubanova Y, Chen R T, Duvenaud D K. Latent ordinary differential equations for irregularly-sampled time series[J]. Advances in neural information processing systems, 2019, 32.

References V

[23] Lechner M, Hasani R. Learning long-term dependencies in irregularly-sampled time series[J]. ArXiv preprint arXiv:2006.04418, 2020.

[24] De Brouwer E, Simm J, Arany A, et al. Gru-ode-bayes: Continuous modeling of sporadically-observed time series[J]. Advances in neural information processing systems, 2019, 32.

[25] Kidger P, Morrill J, Foster J, et al. Neural controlled differential equations for irregular time series[J]. Advances in Neural Information Processing Systems, 2020, 33: 6696-6707.

[26] Chen R T, Amos B, Nickel M. Neural spatio-temporal point processes[J]. ArXiv preprint arXiv:2011.04583, 2020.

[27] Morrill J, Salvi C, Kidger P, et al. Neural rough differential equations for long time series[C]//International Conference on Machine Learning. 2021: 7829-7838.

References VI

[28] Kelly J, Bettencourt J, Johnson M J, et al. Learning differential equations that are easy to solve[J]. Advances in Neural Information Processing Systems, 2020, 33: 4370-4380.

[29] Morrill J, Kidger P, Salvi C, et al. Neural cdes for long time series via the log-ode method[J], 2020.

[30] Song Y, Kingma D P. How to train your energy-based models[J]. ArXiv preprint arXiv:2101.03288, 2021.

[31] Hyvärinen A, Dayan P. Estimation of non-normalized statistical models by score matching.[J]. Journal of Machine Learning Research, 2005, 6(4).

[32] Song Y, Garg S, Shi J, et al. Sliced score matching: A scalable approach to density and score estimation[C]//Uncertainty in Artificial Intelligence. 2020: 574-584.

[33] Song Y, Ermon S. Generative modeling by estimating gradients of the data distribution[J]. Advances in Neural Information Processing Systems, 2019, 32.

References VII

[34] Song Y, Ermon S. Improved techniques for training score-based generative models[J]. Advances in neural information processing systems, 2020, 33: 12438-12448.

[35] Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models[J]. Advances in Neural Information Processing Systems, 2020, 33: 6840-6851.

[36] Song J, Meng C, Ermon S. Denoising diffusion implicit models[J]. ArXiv preprint arXiv:2010.02502, 2020.

[37] Song Y, Sohl-Dickstein J, Kingma D P, et al. Score-based generative modeling through stochastic differential equations[J]. ArXiv preprint arXiv:2011.13456, 2020.

[38] Dockhorn T, Vahdat A, Kreis K. Score-Based Generative Modeling with Critically-Damped Langevin Diffusion[J]. ArXiv preprint arXiv:2112.07068, 2021.

[39] Lam M W, Wang J, Huang R, et al. Bilateral Denoising Diffusion Models[J]. ArXiv preprint arXiv:2108.11514, 2021.

References VIII

[40] San-Roman R, Nachmani E, Wolf L. Noise estimation for generative diffusion models[J]. ArXiv preprint arXiv:2104.02600, 2021.

[41] Watson D, Ho J, Norouzi M, et al. Learning to efficiently sample from diffusion probabilistic models[J]. ArXiv preprint arXiv:2106.03802, 2021.

[42] Xiao Z, Kreis K, Vahdat A. Tackling the Generative Learning Trilemma with Denoising Diffusion GANs[J]. ArXiv preprint arXiv:2112.07804, 2021.

[43] Salimans T, Ho J. Progressive distillation for fast sampling of diffusion models[J]. ArXiv preprint arXiv:2202.00512, 2022.

[44] Nachmani E, Roman R S, Wolf L. Denoising Diffusion Gamma Models[J]. ArXiv preprint arXiv:2110.05948, 2021.

[45] Austin J, Johnson D D, Ho J, et al. Structured denoising diffusion models in discrete state-spaces[J]. Advances in Neural Information Processing Systems, 2021, 34: 17981-17993.

References IX

[46] Hoogeboom E, Nielsen D, Jaini P, et al. Argmax flows and multinomial diffusion: Learning categorical distributions[J]. Advances in Neural Information Processing Systems, 2021, 34.

[47] Lazaro-Gredilla M, Dedieu A, George D. Perturb-and-max-product: Sampling and learning in discrete energy-based models[J]. Advances in Neural Information Processing Systems, 2021, 34.

[48] Meng C, Song Y, Li W, et al. Estimating High Order Gradients of the Data Distribution by Denoising[J]. Advances in Neural Information Processing Systems, 2021, 34.

[49] Vahdat A, Kreis K, Kautz J. Score-based generative modeling in latent space[J]. Advances in Neural Information Processing Systems, 2021, 34.

[50] Kingma D P, Salimans T, Poole B, et al. Variational diffusion models[J]. ArXiv preprint arXiv:2107.00630, 2021.

References X

[51] Huang C W, Lim J H, Courville A C. A variational perspective on diffusion-based generative models and score matching[J]. Advances in Neural Information Processing Systems, 2021, 34.

[52] Song Y, Durkan C, Murray I, et al. Maximum likelihood training of score-based diffusion models[J]. Advances in Neural Information Processing Systems, 2021, 34.

[53] Wang Z, Cheng S, Yueru L, et al. A wasserstein minimum velocity approach to learning unnormalized models[C]//International Conference on Artificial Intelligence and Statistics. 2020: 3728-3738.

[54] Sohl-Dickstein J, Battaglino P, DeWeese M R. Minimum probability flow learning[J]. ArXiv preprint arXiv:0906.4779, 2009.

[55] Lyu S. Unifying non-maximum likelihood learning objectives with minimum KL contraction[J]. Advances in Neural Information Processing Systems, 2011, 24.

References XI

[56] Dhariwal P, Nichol A. Diffusion models beat gans on image synthesis[J]. Advances in Neural Information Processing Systems, 2021, 34.

[57] Meng C, Song Y, Song J, et al. Sdedit: Image synthesis and editing with stochastic differential equations[J]. ArXiv preprint arXiv:2108.01073, 2021.

[58] Kong Z, Ping W, Huang J, et al. Diffwave: A versatile diffusion model for audio synthesis[J]. ArXiv preprint arXiv:2009.09761, 2020.

[59] Chen N, Zhang Y, Zen H, et al. WaveGrad: Estimating gradients for waveform generation[J]. ArXiv preprint arXiv:2009.00713, 2020.

[60] Popov V, Vovk I, Gogoryan V, et al. Grad-tts: A diffusion probabilistic model for text-to-speech[C]//International Conference on Machine Learning. 2021: 8599-8608.

[61] Jeong M, Kim H, Cheon S J, et al. Diff-tts: A denoising diffusion model for text-to-speech[J]. ArXiv preprint arXiv:2104.01409, 2021.

References XII

[62] Chen Z, Tan X, Wang K, et al. Infergrad: Improving Diffusion Models for Vocoder by Considering Inference in Training[C]//ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2022: 8432-8436.

[63] Liu S, Su D, Yu D. DiffGAN-TTS: High-Fidelity and Efficient Text-to-Speech with Denoising Diffusion GANs[J]. ArXiv preprint arXiv:2201.11972, 2022.

[64] Tan X, Qin T, Soong F, et al. A survey on neural speech synthesis[J]. ArXiv preprint arXiv:2106.15561, 2021.

[65] Rasul K, Seward C, Schuster I, et al. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting[C]// International Conference on Machine Learning. 2021: 8857-8868.

[66] Yan T, Zhang H, Zhou T, et al. ScoreGrad: Multivariate Probabilistic Time Series Forecasting with Continuous Energy-based Generative Models[J]. ArXiv preprint arXiv:2106.10121, 2021.

References XIII

[67] Yan T, Zhou T, Zhan Y, et al. TFDPM: Attack detection for cyber-physical systems with diffusion probabilistic models[J]. ArXiv preprint arXiv:2112.10774, 2021.

[68] Luo S, Shi C, Xu M, et al. Predicting Molecular Conformation via Dynamic Graph Score Matching[J]. Advances in Neural Information Processing Systems, 2021, 34.

[69] Jo J, Lee S, Hwang S J. Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations[J]. ArXiv preprint arXiv:2202.02514, 2022.

[70] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[J]. Advances in neural information processing systems, 2014, 27.

[71] Arjovsky M, Bottou L. Towards principled methods for training generative adversarial networks[J]. ArXiv preprint arXiv:1701.04862, 2017.

References XIV

[72] Arjovsky M, Chintala S, Bottou L. Wasserstein generative adversarial networks[C]//International conference on machine learning. 2017: 214-223.

[73] Gulrajani I, Ahmed F, Arjovsky M, et al. Improved training of wasserstein gans[J]. Advances in neural information processing systems, 2017, 30.

[74] Miyato T, Kataoka T, Koyama M, et al. Spectral normalization for generative adversarial networks[J]. ArXiv preprint arXiv:1802.05957, 2018.

[75] Mescheder L, Geiger A, Nowozin S. Which training methods for GANs do actually converge?[C]//International conference on machine learning. 2018: 3481-3490.

[76] Nielsen D, Jaini P, Hoogeboom E, et al. Survae flows: Surjections to bridge the gap between vaes and flows[J]. Advances in Neural Information Processing Systems, 2020, 33: 12685-12696.

References XV

[77] Makhzani A, Shlens J, Jaitly N, et al. Adversarial autoencoders[J]. ArXiv preprint arXiv:1511.05644, 2015.