Generative Models

2022 年 2 月 27 日

Contents

Section 1

Variational Inference

Subsection 1

Statistical Inference

Statistical inference

统计推断（Statistical Inference）：通过样本推断总体的统计方法。

统计推断两大学派：

- **频率学派（Frequentist）**

  最大似然估计（Maximum Likelihood Estimation, MLE）

- **贝叶斯学派（Bayesians）**

  最大后验概率（Maximum A Posterior，MAP）

Frequentist VS. Bayesians

- **频率学派**：

  认为存在唯一的真实常数参数，观察数据都是在这个参数下产生的。

- **贝叶斯学派**：

  认为参数本身存在一个概率分布，参数空间中的每个值都可能是真实模型使用的参数，只是概率不同。

Problem Definition

给定样本数据 $D = \{x_1, x_2, \ldots, x_n\}$，统计推断的任务就是通过已有的数据估计某个未知的参数 θ。

$$p(\theta|D) = \frac{p(\theta, D)}{p(D)} = \frac{p(D|\theta)p(\theta)}{p(D)}$$

$$Posterior = \frac{Likelihood \times Prior}{Evidence}$$

Frequentist & MLE

频率学派认为存在唯一的真实常数参数。那么这个参数 θ 取概率最大的参数即可。这就是最大似然估计（MLE）。

$$\hat{\theta}_{map} = \arg\max_{\theta} p(\theta|D) = \arg\max_{\theta} \frac{p(D|\theta)p(\theta)}{p(D)}$$

对于频率学派的最大似然估计，$p(\theta)$ 和 $p(D)$ 可以看作常数。相当于找到一个使抽样出当前样本集概率最大的 θ。

$$\begin{aligned}\arg\max_{\theta} p(D|\theta) &= \arg\max_{\theta} \prod_{i=1}^{n} p(x_i|\theta) \\ &= \arg\max_{\theta} \sum_{i=1}^{n} log(p(x_i|\theta))\end{aligned}$$

Bayesians & MAP

贝叶斯学派则是需要求出 θ 的分布，而不是只有一个 θ，而且 θ 服从一个先验的分布 $p(\theta)$。

$$
\begin{aligned}
p(\theta|D) &= \frac{p(D|\theta)p(\theta)}{p(D)} \\
&= \frac{(\prod_{i=1}^{n} p(x_i|\theta))p(\theta)}{\int_\theta \prod_{i=1}^{n} p(x_i|\theta)p(\theta)d\theta}
\end{aligned}
$$

可以看出这个过程是需要求积分的，而且这个积分通常情况下是 intractable 的。

Bayesians & MAP

为了避免求积分，一种可行的思路就是在贝叶斯估计中，采用极大似然估计的思想。考虑后验分布极大化而求解参数 θ，这样就变成了最大后验估计（MAP）。

$$\begin{aligned}\hat{\theta}_{map} &= \arg\max_{\theta} p(D|\theta)p(\theta) \\ &= \arg\max_{\theta}(log(p(D|\theta)) + log(p(\theta)))\end{aligned}$$

与最大似然估计最大的区别就是后面的 $log(p(\theta))$ 而这一项的本质就是正则化项。

Subsection 2

Approximate Inference

Approximate Inference

MAP 最终得到的也是只有一个 θ。如果仍然需要得到分布 $p(\theta|D)$，需要计算积分，可能代价很大，或者根本不可积，无法进行精确推断。现在钱我不想给，货我又想要，怎么办？

答案就是：**近似推断（Approximate Inference）**

两个重要近似推断方法：

- 变分推断（Variational Inference，VI）
- 马尔科夫蒙特卡洛采样（Markov Chain Monte Carlo，MCMC）

Approximate inference

Probabilistic model: $p(x, \theta) = p(x \mid \theta)p(\theta)$

Variational Inference

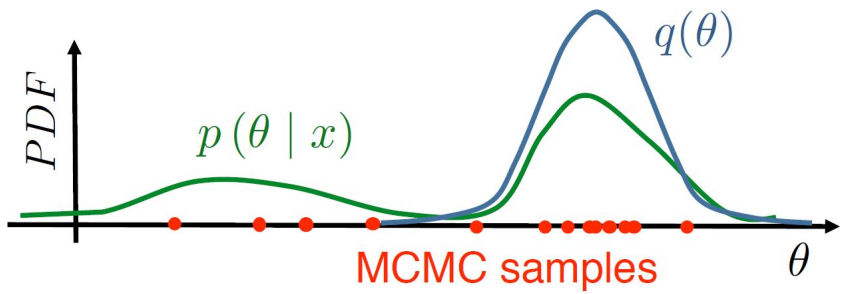
Approximate $p(\theta \mid x) \approx q(\theta) \in \mathcal{Q}$

- Biased
- Faster and more scalable

MCMC

Samples from unnormalized $p(\theta \mid x)$

- Unbiased
- Need a lot of samples



Variational Inference

变分推断的思想就是从已知的一族分布 Q，中找到一个与后验分布 $p(\theta|D)$ 最接近的分布来近似 $p(\theta|D)$ 。这样就把求积分换成了优化问题。

Variational Inference

于是用 KL 散度衡量两个分布的相似性可以得到如下优化目标：

$$
\begin{aligned}
q^*(\theta) &= \arg\min_{q(\theta)\in Q} D_{KL}[q(\theta)||p(\theta|D)] \\
&= \arg\min_{q(\theta)\in Q}\{E_{\theta\sim q(\theta)}[log(q(\theta))] - E_{\theta\sim q(\theta)}[log(p(\theta|D))]\} \\
&= \arg\min_{q(\theta)\in Q}\{E_{\theta\sim q(\theta)}[log(q(\theta))] - E_{\theta\sim q(\theta)}[log(p(\theta,D))] + E_{\theta\sim q(\theta)}[log(\\
&= \arg\min_{q(\theta)\in Q}\{E_{\theta\sim q(\theta)}[log(q(\theta))] - E_{\theta\sim q(\theta)}[log(p(\theta,D))] + log(p(D))\} \\
&= \arg\min_{q(\theta)\in Q}\{E_{\theta\sim q(\theta)}[log(q(\theta))] - E_{\theta\sim q(\theta)}[log(p(\theta,D))]\}
\end{aligned}
$$

Variational Inference

$$D_{KL}[q(\theta)||p(\theta|D)] = E_{\theta\sim q(\theta)}[log(q(\theta))] - E_{\theta\sim q(\theta)}[log(p(\theta, D))] + log(p(D$$

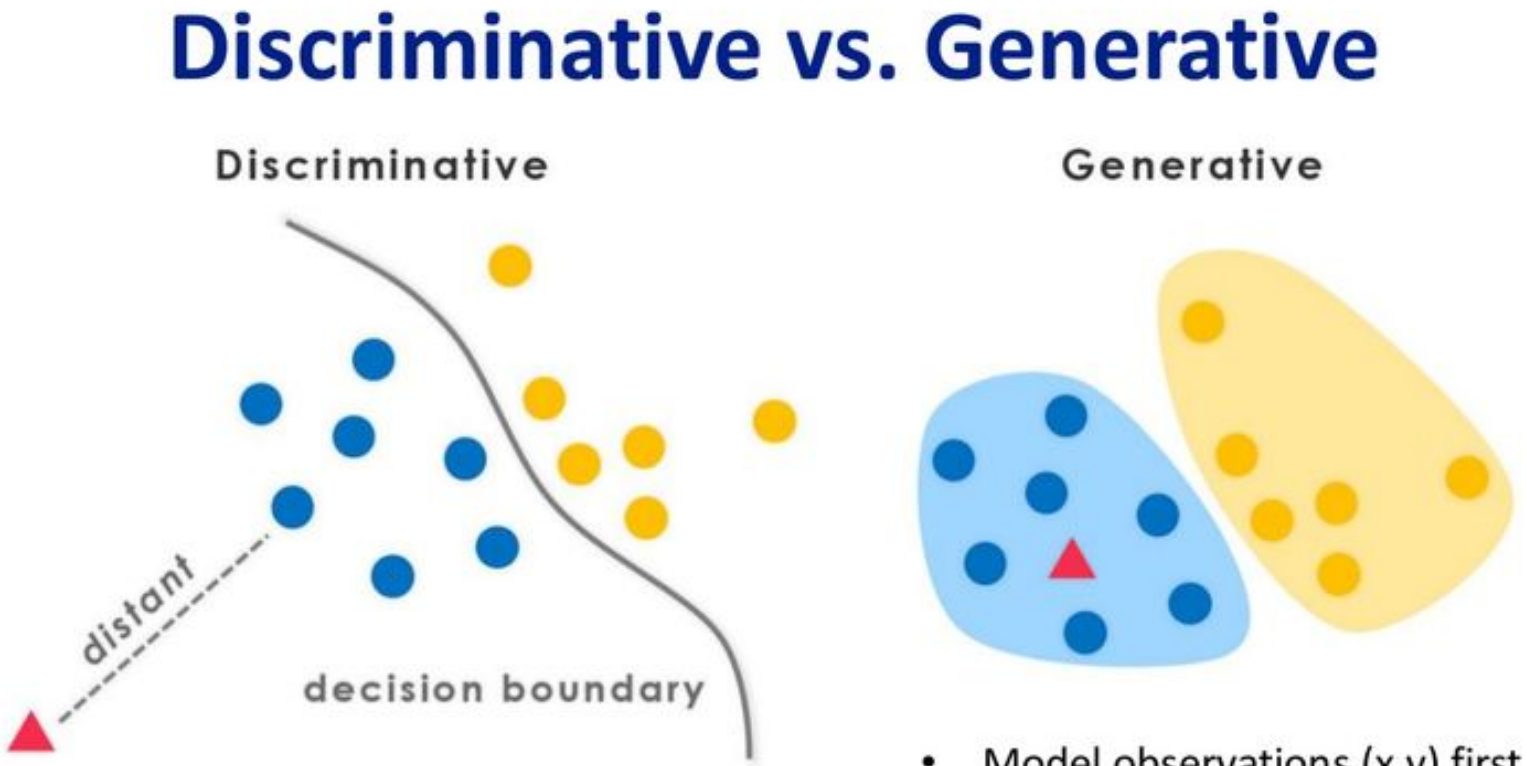
$$log(p(x)) \geq E_{\theta\sim q(\theta)}[log(p(\theta, D))] - E_{\theta\sim q(\theta)}[log(q(\theta))]$$

$p(D)$ 被称作 Evidence，上式又被称作 Evidence Lower Bound (ELBO)，也就是变分下界。

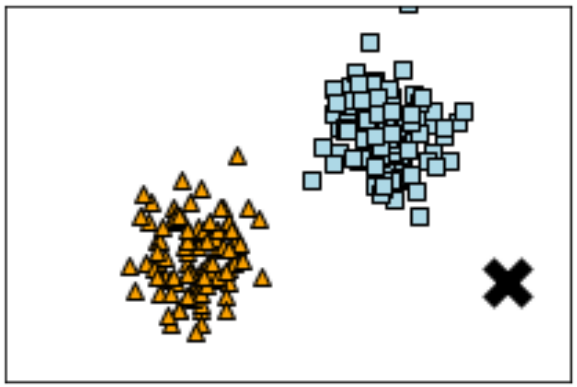
Section 2

Generative Models

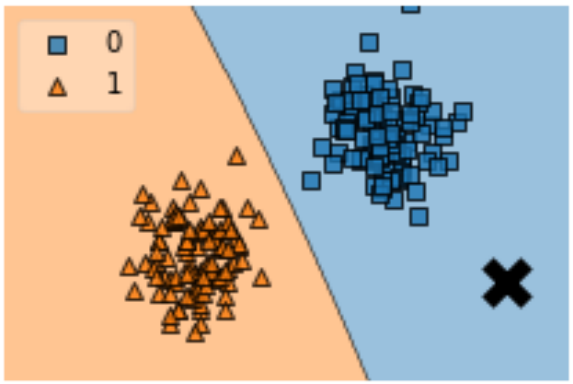
Generative VS. Discriminative



Generative VS. Discriminative

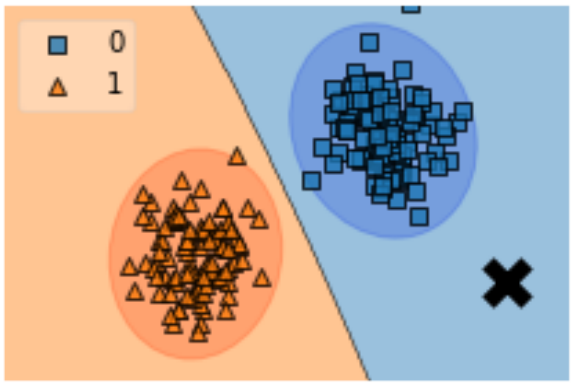


Data



$p(y|\mathbf{x})$

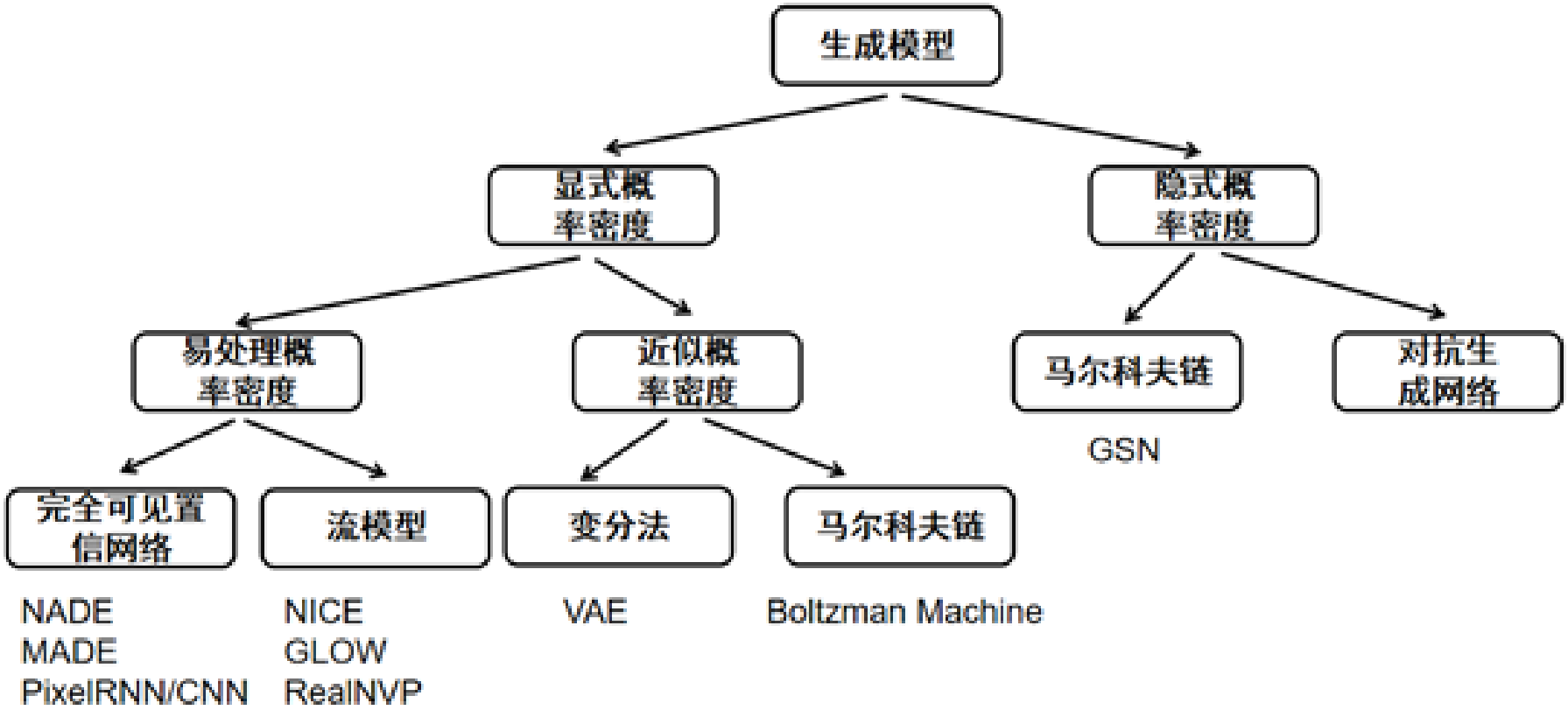
$p(blue|\mathbf{x})$ is high
= certain decision!



$p(\mathbf{x}, y) = p(y|\mathbf{x})\ p(\mathbf{x})$

$p(blue|\mathbf{x})$ is high
and $p(\mathbf{x})$ is low
= uncertain decision!

Generative Models



Section 3

Flow-Based Models

Background

变量替换公式：设 $f: \mathbb{R}^D \to \mathbb{R}^D$ **为可逆变换，** $x = f(z), z = f^{-1}(x)$，X **的概率密度为** $p_x(x)$，z **的概率密度为** $p_z(z)$。

$$p_x(x) = p_z(z = f^{-1}(x))|\frac{\partial f^{-1}(x)}{\partial x}|$$

$$|\frac{\partial f^{-1}(x)}{\partial x}| = |det\ \mathbf{J}_{f^{-1}}(x)| = |det\ \mathbf{J}_f(x)|^{-1}$$

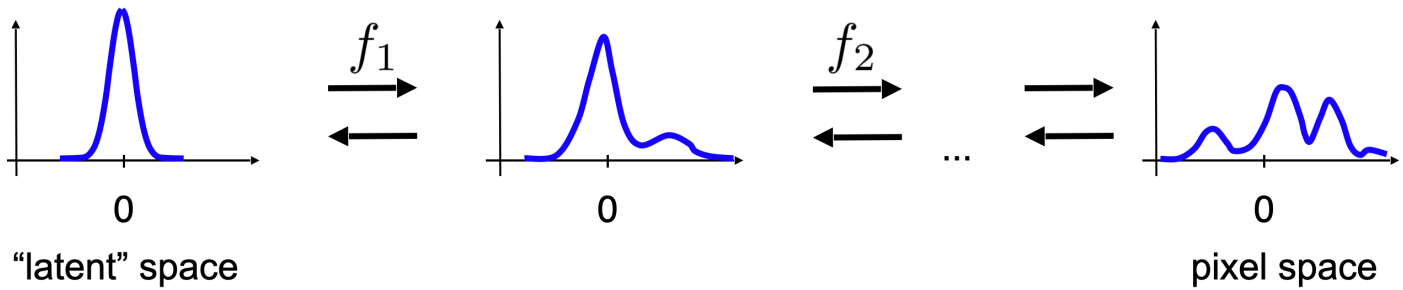
$$p_x(x) = p_z(z = f^{-1}(x))|det\ \mathbf{J}_f(x)|^{-1}$$

Flow-Based Model

Flow 模型的思想正是变量替换公式。通过一系列可逆变换，先将未知的分布 $p_x(x)$ 变换到一个已知的分布 $p_{z_0}(z_0)$ 上，那么通过逆变换就可以将 $p_z(z)$ 变换回 $p_x(x)$ 。

$$
\begin{aligned}
p_x(x) &= p_{z_0}(z_0 = f^{-1}(x)) \prod_{i=1}^{K} |det \frac{\partial f_i(z_{i-1})}{\partial z_{i-1}}|^{-1} \\
&= p_{z_0}(z_0 = f^{-1}(x)) \prod_{i=1}^{K} |\mathbf{J}_{f_i}(z_{i-1})|^{-1}
\end{aligned}
$$

Flow-Based Model



Maximum Likelihood

Flow 模型属于似然模型，其优化目标：

$$\arg\max_{\theta} P(D|\theta) = \arg\max_{\theta} \sum_{n} log(P(\boldsymbol{x_n}|\theta)) + Regularization$$

$$log(P(x|\theta)) = log(P_z(z = f^{-1}(x)) \prod_{i=1}^{K} |\mathbf{J}_{f_i}(z_{i-1})|^{-1}))$$

$$= log(P_z(z = f^{-1}(x))) - \sum_{i=1}^{K} (log(|\mathbf{J}_{f_i}(z_{i-1})|))$$

目标前半部分就是约束 x 变换后的 z 要与先验分布一致。后半部分就是尽量使得样本空间经过可逆变换能够覆盖足够多的正态分布的空间。

Invertible Function: Coupling Layer

Coupling Layer：设输入的 D 维向量，进行拆分即

$x = [x_a, x_b] = [x_{1:d}, x_{d+1:D}]$ ，coupling layer 执行如下转换：

$$y_a = x_a$$

$$y_b = exp(s(x_a)) \odot x_b + t(x_a)$$

相应的逆变换：

$$x_a = y_a$$

$$x_b = (y_b - t(y_a)) \odot exp(-s(y_a))$$

Invertible Function: Coupling Layer

Coupling Layer 的 Jacobian 矩阵为:

$$J = \begin{bmatrix} I_{d\times d} & 0_{d\times(D-d)} \\ \frac{\partial y_b}{\partial x_a} & diag(exp(s(x_a))) \end{bmatrix}$$

$$det(J) = \prod_{j-1}^{D-d} exp(s(x_a))_j$$

$$= exp(\sum_{j-1}^{D-d} s(x_a)_j)$$

An Example: Additive Coupling Layer

Coupling Layer 的一个例子，Additive Coupling Layer：

$$y_a = x_a$$

$$y_b = x_b + m(x_a)$$

其对应的 Jacobian 矩阵：

$$J = \begin{bmatrix} I_{d\times d} & 0_{d\times(D-d)} \\ \frac{\partial y_b}{\partial x_a} & I_{(D-d)\times(D-d)} \end{bmatrix}$$

$$det(J) = 1$$

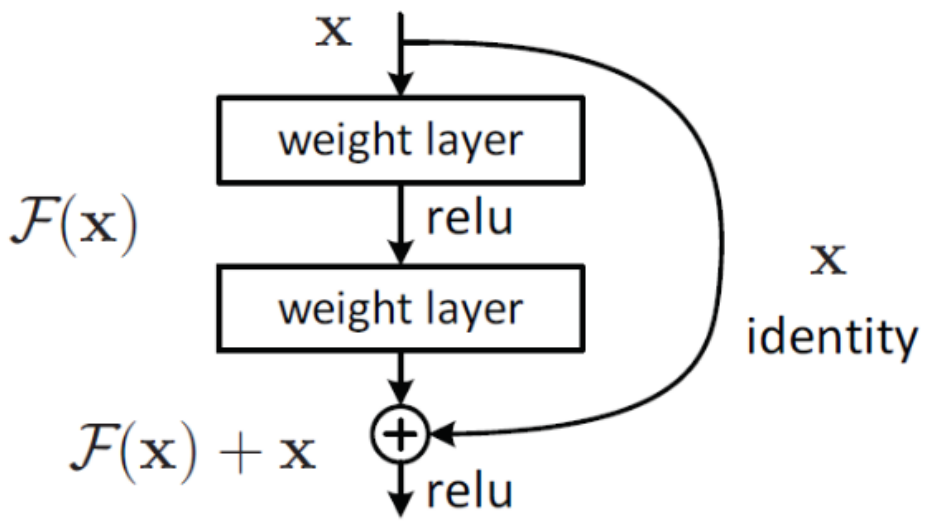
Section 4

Neural ODE

Subsection 1

Neural ODE

ResNet Block



残差网络可以写成如下形式：

$$z_{t+1} = z_t + f(z_t, \theta_t)$$

From Res to ODE

残差网络可以看作建模每一步变化量：

$$\triangle z = z_{t+1} - z_t = f(z_t, \theta_t)$$

换一个角度，也可看作建模的是变化率，其中 dt=1：

$$\frac{dz(t)}{dt} = \frac{z_{t+dt} - z_t}{dt} = f(z_t, \theta_t)$$

如果 dt 能够无穷小，就变成了一个常微分方程（Ordinary Differential Equation，ODE）：

$$\frac{dz(t)}{dt} = \frac{z_{t+dt} - z_t}{dt} = f(z(t), t, \theta)$$

Neural ODE

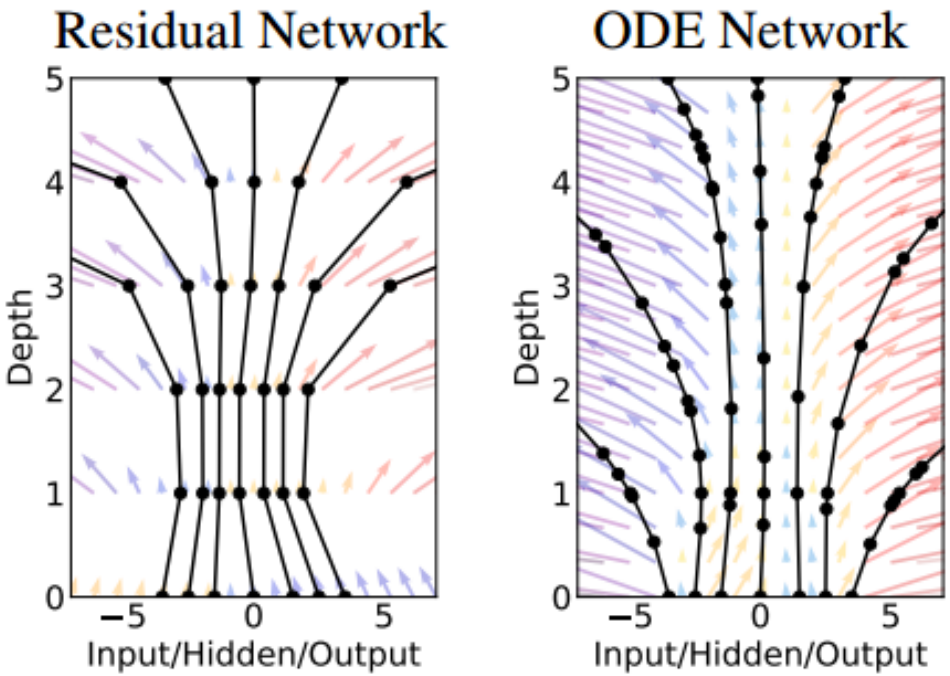


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

Forward Process

Neural ODE 的前向过程就是求解一个常微分方程。这个求解的方式可以是一个黑盒操作。

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt = ODESolve(z(t_0), f, t_0, t_1, \theta)$$

Backward Process

反向过程不能直接通过计算图反向传播 loss。需要另辟蹊径。设 loss 函数为

$$
\begin{aligned}
L\Big(z(t_1)\Big) &= L\left(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt\right) \\
&= L\left(ODESolve\Big(z(t_0), f, t_0, t_1, \theta\Big)\right)
\end{aligned}
$$

令 $a(t) = \frac{\partial L}{\partial z(t)}$ 称为 adjoint。

$$
\begin{aligned}
\frac{da(t)}{dt} &= -a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial z} \\
\frac{dL}{d\theta} &= -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt
\end{aligned}
$$

Backward Process

总之一句话，反向过程也可以通过求解一个 ODE 来实现 loss 回传，更新参数。

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, 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

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

Advantages of NODE

Neural ODE 用一个网络来建模变化率，那么一个 Neural ODE 网络可以达到任意多层残差网络堆叠的效果。

带来的弊端也就是更高的计算复杂度。

Subsection 2

Continuous Normalizing Flow

Continuous Normalizing Flow

任何形如 $\triangle z = z_{t+1} - z_t = f(z_t, \theta_t)$ 的式子都可以用 Neural ODE 来解决。

有一类流模型，表达式形如

$$z_1 = z_0 + f(z_0)$$

$$log\Big(p(z_1)\Big) = log(p(z_0)) - log\Big|det(I + \frac{\partial f}{\partial z_0})\Big|$$

就可以使用 Neural ODE 来解决。

Continuous Normalizing Flow

以 Planar Flow 为例，$\boldsymbol{u}, \boldsymbol{w} \in R^D, b \in R, h$ 为激活函数。:

$$z(t+1) = z(t) + \boldsymbol{u} h(\boldsymbol{w}^T z(t) + b)$$

$$log\Big(p(z(t+1))\Big) = log\Big(p(z(t))\Big) - log\Big|1 + \boldsymbol{u}^T \frac{\partial h}{\partial z}\Big|$$

可以利用 Neural ODE 学到每一时刻的参数，类似于超网来实现 Continuous Flow。

$$\frac{dz(t)}{dt} = \boldsymbol{u} h(\boldsymbol{w}^T z(t) + b)$$

$$\frac{\partial log p(z(t))}{\partial t} = -\boldsymbol{u}^T \frac{\partial h}{\partial z(t)}$$

Subsection 3

Time-Series Model

Time-Series Model

另一种与 Neural ODE 契合的模型是时间序列模型。

作者基于 VAE 的思想提出一种 Generative Latent Function Time-Series Model：

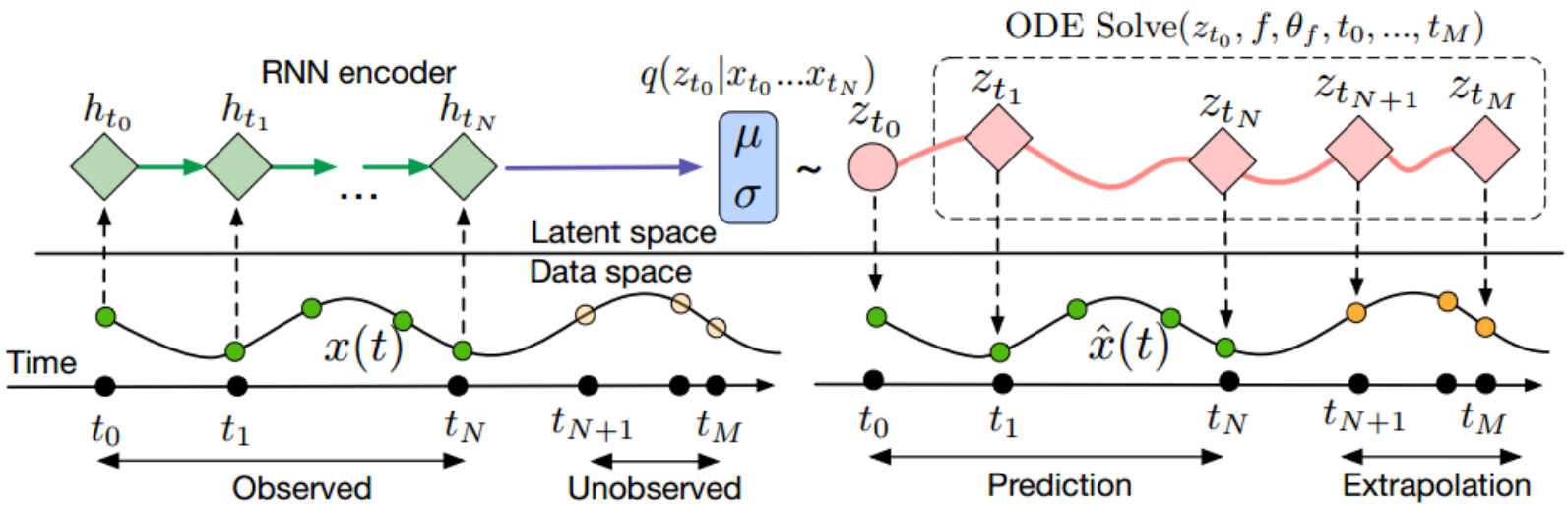


Figure 6: Computation graph of the latent ODE model.

Time-Series Model

预测的过程就可以用求解 ODE 来表示:

$$\mathbf{z}_{t_0} \sim p(\mathbf{z}_{t_0}) \tag{11}$$

$$\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_N} = \text{ODESolve}(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_N) \tag{12}$$

$$\text{each} \quad \mathbf{x}_{t_i} \sim p(\mathbf{x}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}}) \tag{13}$$

Neural CDE

如何让 ODE 能有更强的时序性，达到像自回归模型 (Autoregressive Model，ARM)，比如 RNN 一样的效果？

现在的 ODE 是对时间 t 进行积分。而 RNN 每一步的结果依赖于之前所有的结果，相当于对 x 求积分。

这也就是 Neural Controlled Differential Equation（Neural CDE）

Neural CDE

通过对离散的 x 进行三次样条插值（Natural Cubic Spline），将其连续化，然后对其进行积分。

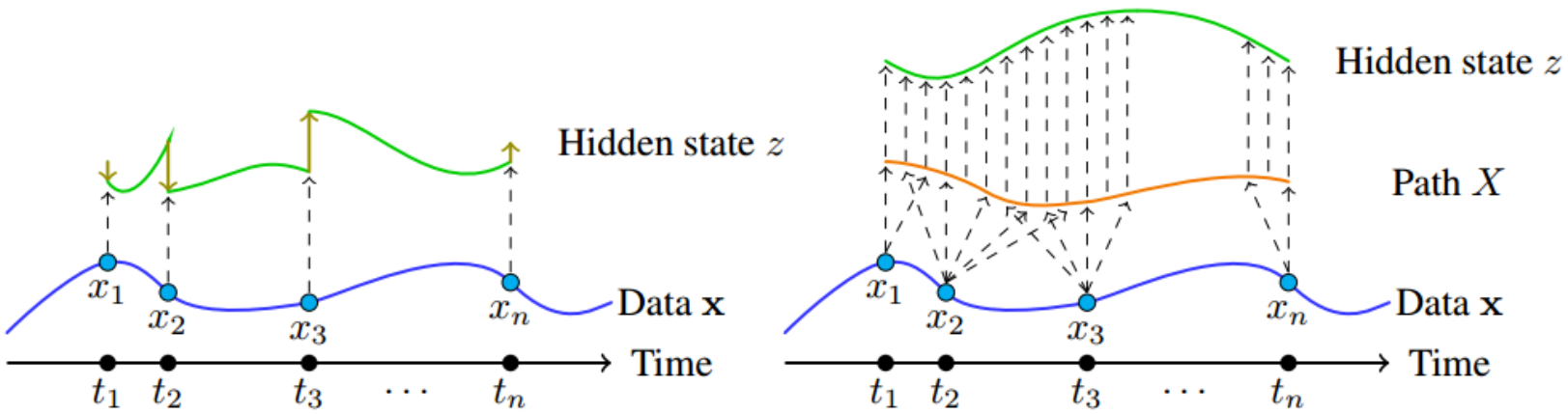


Figure 1: Some data process is observed at times $t_1, \ldots, t_n$ to give observations $x_1, \ldots, x_n$. It is otherwise unobserved. **Left:** Previous work has typically modified hidden state at each observation, and perhaps continuously evolved the hidden state between observations. **Right:** In contrast, the hidden state of the Neural CDE model has continuous dependence on the observed data.

Neural CDE

用公式来表示：

$$z_t = z_{t_0} + \int_{t_0}^{t} f_\theta(z_s) dX_s \qquad for \quad t \in (t_0, t_n]$$

$$g_{\theta,X}(z, s) = f_\theta(z) \frac{dX}{ds}(s)$$

$$z_t = z_{t_0} + \int_{t_0}^{t} g_{\theta,X}(z_s, s) ds$$

Section 5

Neural ODE Processes

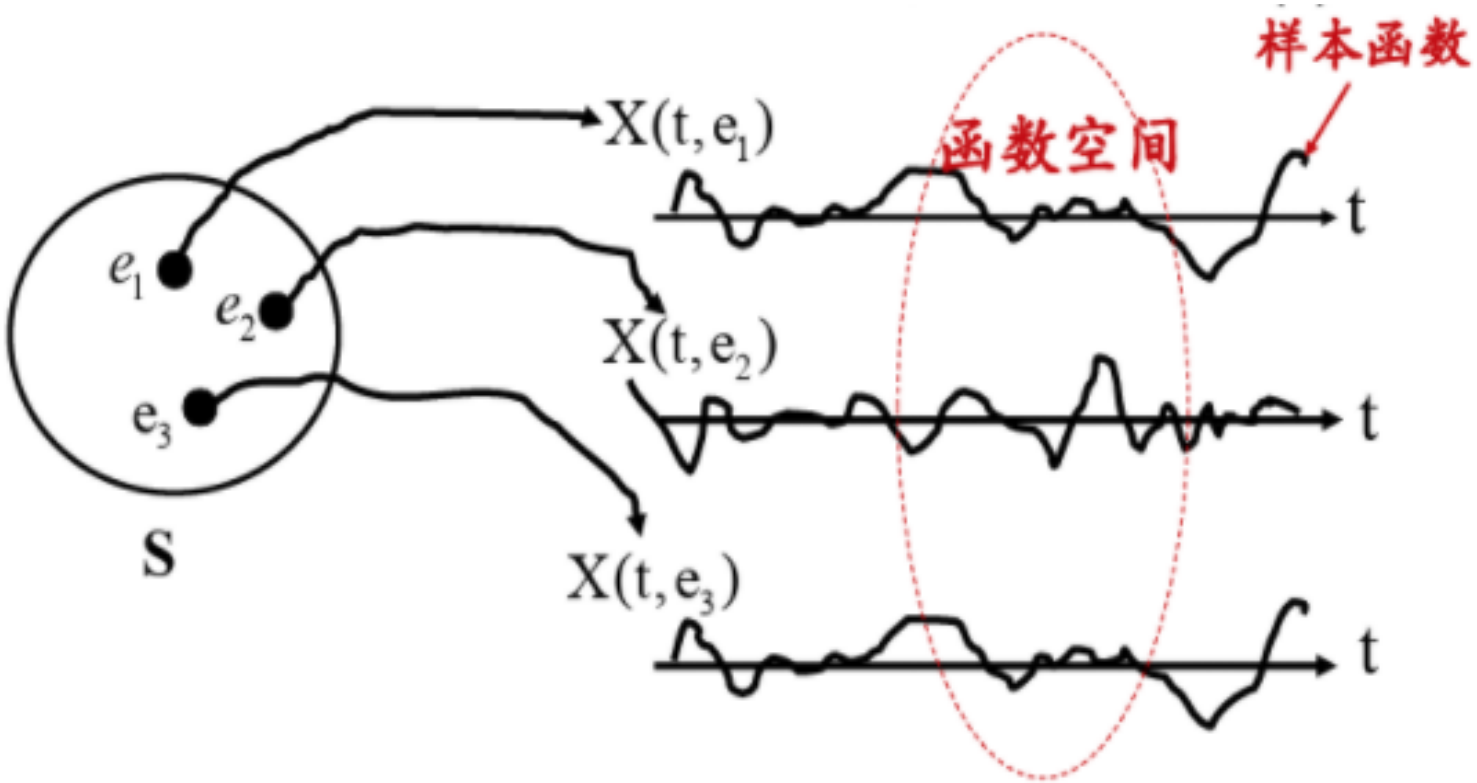
Subsection 1

Gaussian Processes

Stochastic Processes

随机过程（Stochastic Processes）：设 E 是随机试验，它的样本空间是 S，若对于每一个 $e_i \in S$，总有一个确定的时间函数与之对应 $X(t, e_i)$ 与之对应，这样对于所有的 e，就可以得到一族时间 t 的函数 $X(t, e_1), X(t, e_2), ..., X(t, e_n)$，称为随机过程。族中的每一个函数称为这个随机过程的样本函数。

Stochastic Processes



Markov Processes

马尔科夫性：

$$P(X(t) \leq x | X(t_n) = x_n, ..., X(t_1) = x_1) = P(X(t) \leq x | X(t_n) = x_n)$$

也称作无后效性或无记忆性。

马尔科夫过程（Markov Processes）：若随机过程满足马尔科夫性，则称为马尔科夫过程。

Brouwn Motion

布朗运动也称维纳过程：

如果随机过程 $W(t), t \geq 0s$ 满足下列条件：

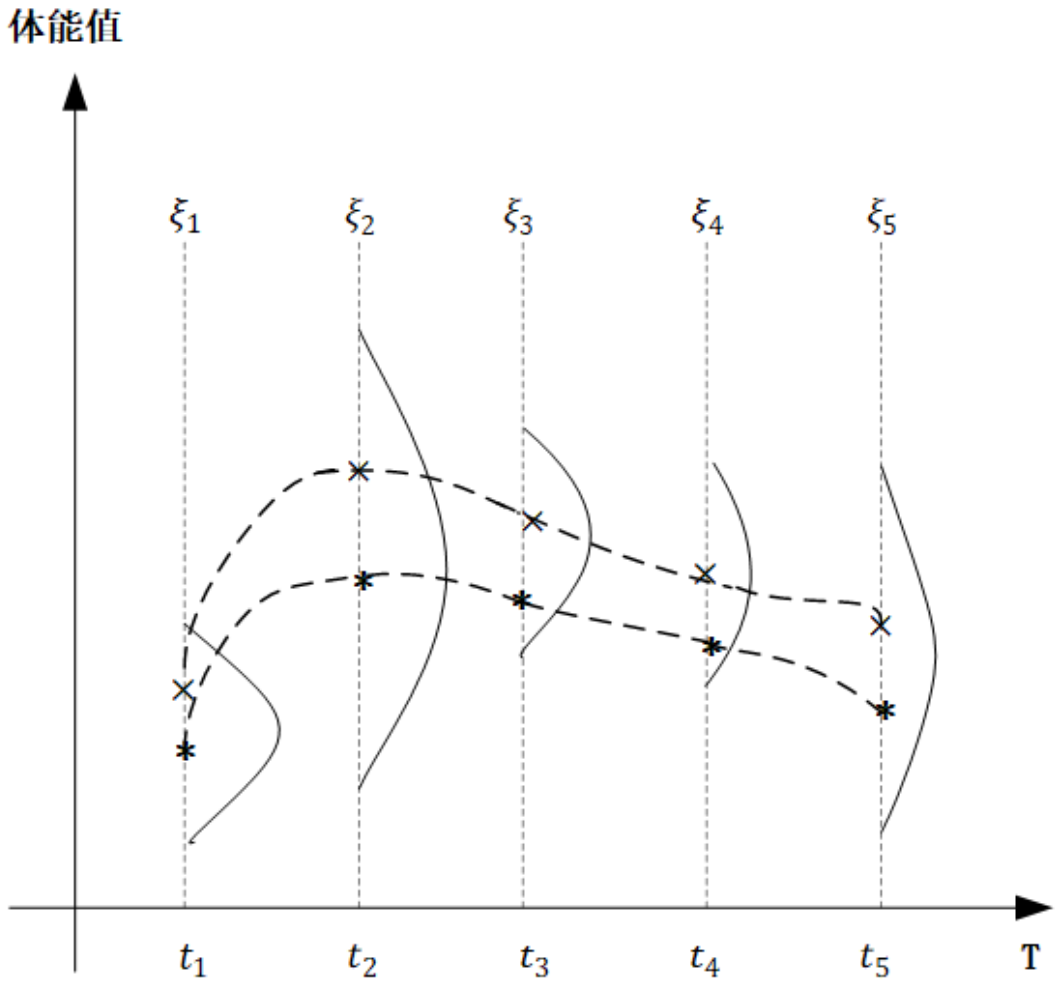
- 1. $W(0) = 0$
- 2. $E[W(t)] = 0$
- 3. 具有平稳独立增量
- 4. $t > 0, W(t) \sim N(0, \sigma^2 t), \sigma > 0$

称随机过程 $W(t), t \geq 0s$ 是参数为 σ^2 的维纳过程。（若 $\sigma = 1$，则称为标准维纳过程）维纳过程属于马尔科夫过程。

Gaussian Processes

给定随机过程 $\{X(t), t \in T\}$，若对任意的正整数 $n \geq 1$ 及任意的 $t_1, t_2, ..., t_n \in T$，随机变量 $X_{t_1}, X_{t_2}, ..., X_{t_n}$ 的联合分布是 n 维正态分布。则称 $X(t), t \in T$ 是正态过程（高斯过程）。

高斯过程的一个例子：人的体能与年龄



Gaussian Processes

p 维高斯分布，需要确定的参数是每一个维度的均值构成的向量 μ，以及不同维度之间的协方差矩阵 $\sum_{p\times p}$。

连续域内，不同维度的均值就是每一个时间 t 对应的的均值，是一个关于时间的函数 $m(t)$ 。相应的不同维度之间的协方差就是不同时间 s, t 之间的协方差函数 $k(s, t)$ ，这个函数也称为核函数。

Gaussian Processes

高斯过程通常可以用来表示一个函数的分布。

通常如果我们要学习一个函数，首先定义函数的参数，然后根据训练数据来学习这个函数的参数。也就是说学习到的是一个函数。比如线性回归，学习这样一个函数就相当于训练回归参数（权重，偏置）。这种方法叫参数化方法。这种做法把可学习的函数的范围限制死了，无法学习任意类型的函数。而非参数化方法就没有这个缺点，用高斯过程来建模函数就是一种非参数方法。用高斯过程，学到的是一族函数，我们对每个时间点采样得到的 path，就是这一族函数之中的一个。

Subsection 2

Conditional Neural Processes

Background

- 与 meta-learning 类似。通常神经网络能很好的拟合函数，但是对于一个新的任务就要从头开始训练，而且需要数据集需要有大量的数据。
- 高斯过程（GP）则利用先验的信息，从而能够在测试的时候很快的拟合一个新的函数。
- 但是 GP 计算代价很大，而且设计一个合适的先验并不容易。

Conditional Neural Processes

与 meta-learning 相似，Conditional Neural Processes(CNP) 的训练集和测试集都包含两部分的数据，观测数据 Observation（也称作 Context）和目标 Target，也就是说 CNP 的训练集和测试集是数据集的集合。观测数据 $O = \{(x_i, y_i)_{i=0}^{n-1} \subset X \times Y\}$，目标数据 $T = \{x_i\}_{i=n}^{n+m-1} \subset X$。那么 CNP 的训练集和测试集都包含着观测数据和目标数据的组合，即形如 $train - set = \{(O, T)_{j=1}^{J}\}$

Conditional Neural Processes

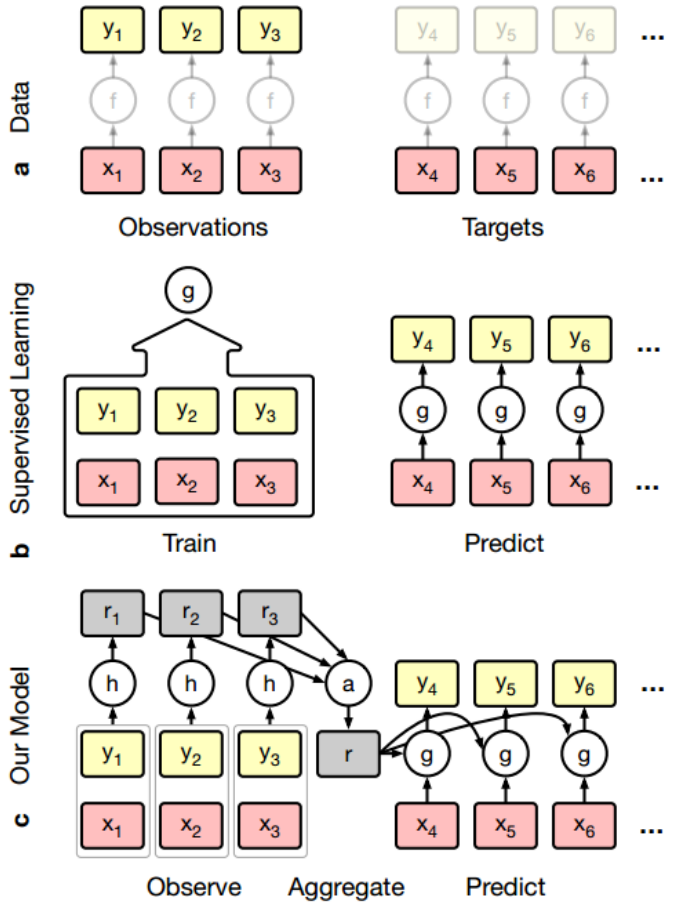


Figure 1. **Conditional Neural Process.** a) Data description b) Training regime of conventional supervised deep learning models c) Our model.

Conditional Neural Processes

CNP 包含三个部分：

- encoder(Observe) 部分，根据观测集 O 中的数据得到中间信息 r。
- aggregator 部分，将中间信息 r 进行整合。
- decoder(Predict)，在目标集 T 上测试结果。

Conditional Neural Processes

用公式可以简洁的表示成如下形式：

$$r_i = h_\theta(x_i, y_i), \qquad \forall (x_i, y_i) \in O$$

$$r = r_1 \oplus r_2 \oplus \ldots r_{n-1} \oplus r_n$$

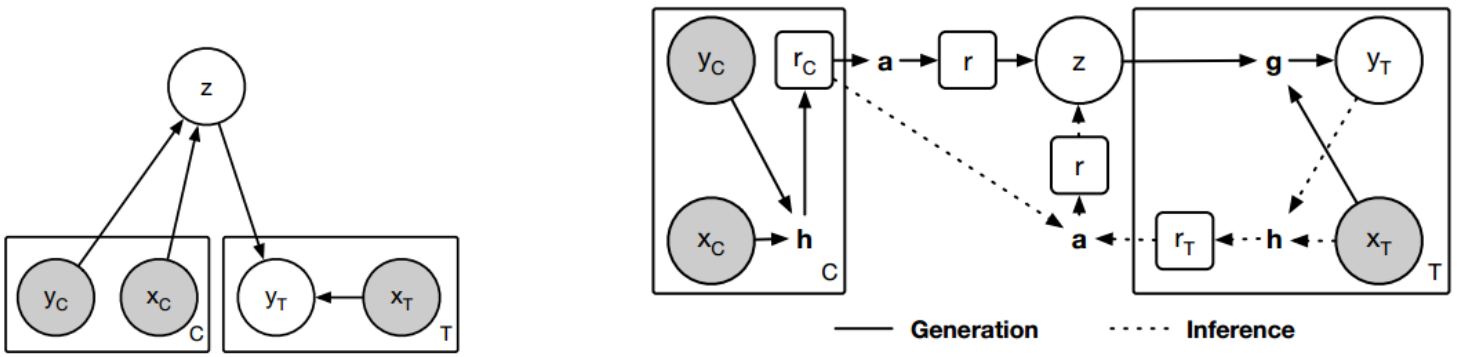
$$\phi_j = g_\theta(x_j, r) \qquad \forall (x_i) \in T$$

Subsection 3

Neural Processes

Neural Processes

Neural Processes（NP）只是在 CNP 的基础上加入了一步变分推断。从而使得结果具有随机性。



(a) Graphical model

(b) Computational diagram

Figure 1. **Neural process model. (a)** Graphical model of a neural process. x and y correspond to the data where $y = f(x)$. C and T are the number of context points and target points respectively and z is the global latent variable. A grey background indicates that the variable is observed. **(b)** Diagram of our neural process implementation. Variables in circles correspond to the variables of the graphical model in (a), variables in square boxes to the intermediate representations of NPs and unbound, bold letters to the following computation modules: h - encoder, a - aggregator and g - decoder. In our implementation h and g correspond to neural networks and a to the mean function. The continuous lines depict the generative process, the dotted lines the inference.

Neural Processes

NP 与 CNP 一样也分为三个部分：

- encoder
- aggregator
- decoder

而区别就在于在 aggregator 中计算出 r 之后多加了一步计算均值方差并采样 z 的过程。

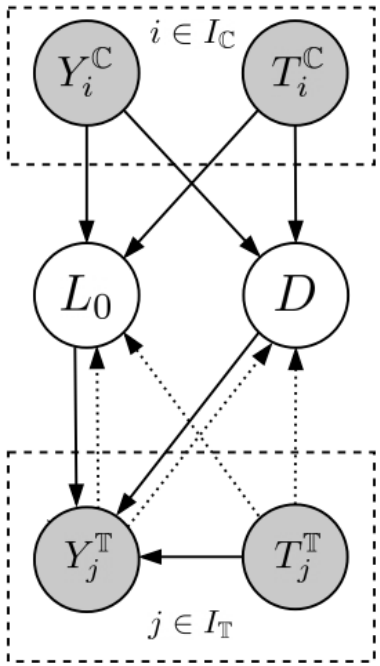
Subsection 4

Neural ODE Processes

Neural ODE Processes

Neural ODE Processes(NDP) 则是将 Neural ODE 用于 Neural Processes。

NDP 借鉴了了 NP 的三部分的结构，并且稍作改动。



Encoder & Aggregator

第一部分：Encoder & Aggregator

Neural ODE 需要初始值，另外 NDP 还借鉴了 NP 中的隐变量 z。

于是 encoder 部分需要学到 NODE 的初始状态 $L_0 \sim q_L(l(t_0)|C)$ ，

以及对于 ODE 的控制信息 D，相当于 NP 中的 z，C 为 Context 集也就是观测集（Observation）。

$D \sim q_D(d|C) = N(d|\mu_D(r), diag(\sigma_D(r)))$

Latent ODE

第二部分：Latent ODE

求解常微分方程

$$l(t_i^T) = l(t_0) + \int_{t_0}^{t_i^T} f_\theta(l(t), d, t)dt$$

Decoder

第三部分：Decoder 通过第二步的微分方程得到的 $I(t_i^T)$，来计算任意时刻的 $Y_{t_i}^T \sim N(y_i^T | g(I(t_i^T), t_i^T))$

Section 6

Score Matching

Subsection 1

Score-Based Models

Background

另一种求 P(X) 的方法就是积分。

$$P(X) = \int \nabla P(X) dX$$

这里涉及两个问题：

- 1. 这个积分怎么求？

  用朗之万动力学来避开求积分，或者说近似。

- 2. 导数怎么求？

  用 score-matching 来近似

Langevin Dyanmics

小惯性粒子的离散朗之万方程：

$$x_{t+\triangle t} = x_t - \triangle t \nabla E + \triangle t R_t$$

朗之万动力学的一个重要性质是随机过程 $x(t)$ 服从朗之万方程，则其扩散分布 $P(X)$ 收敛于平稳分布，即玻尔兹曼分布 (Boltzmann Distribution，BD)：

$$p(x) \propto e^{-E(x)/T}$$

Langevin Dyanmics

朗之万动力学应用到生成模型当中：

$$x_{t+\triangle t} = x_t - \triangle t \nabla log P_{data}(X) + \triangle t R_t$$

这样就可以通过不断采样的方式得到 $P_{data}(X)$，而不需要求积分。

Denoising Score Matching

DSM 建立在这样一种假设上：如果当前的 x_t 是带有噪声，那么下一步 x_{t+1} 就应该消除掉这个噪声，从而向没有噪声的 clean sample x 靠近。

事先对 x 进行扰动，比如加上一个高斯噪声，$q_\sigma(\tilde{x}|x) = N(0, \sigma^2)$ ，即 $\tilde{x} = x + z$ ，那么：

$$\frac{\partial log q_\sigma(\tilde{x}|x)}{\partial \tilde{x}} = \frac{1}{\sigma^2}(x - \tilde{x})$$

Denoising Score Matching

设置 DSM 的目标为：

$$J_{DSM} = E_{q_\sigma(x,\tilde{x})}[\frac{1}{2}||s_\theta(x) - \frac{\partial log q_\sigma(\tilde{x}|x)}{\partial \tilde{x}}||^2]$$

那么当 σ 足够小的时候，可以得到

$$s_{\theta^*}(x) = \nabla_x log q_\sigma(x) \approx \nabla_x log p(x)$$

Subsection 2

Noise Conditional Score Networks

Noise Conditional Score Networks

以 Noise Conditional Score Networks(NCSN) 为例。

NCSN 首先定义一系列的高斯噪声的标准差差 $\{\sigma_i\}_{i=1}^{L}$ ，这些标准差要严格递减。大的要足够大到让图像看起来就是个噪声，小的要足够小到几乎对图像没有影响。

Noise Conditional Score Networks

训练的过程：

- 1. 随机挑选一个 σ_i
- 2. 使用 σ_i 对数据进行 perturb，也就是 $\tilde{x} = x + z * \sigma_i$ ，z 服从标准正态分布。
- 3. 计算损失，更新网络参数
- 4. 重复以上过程直至收敛

Noise Conditional Score Networks

朗之万动力学采样过程：

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
  end for
  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

Noise Conditional Score Networks

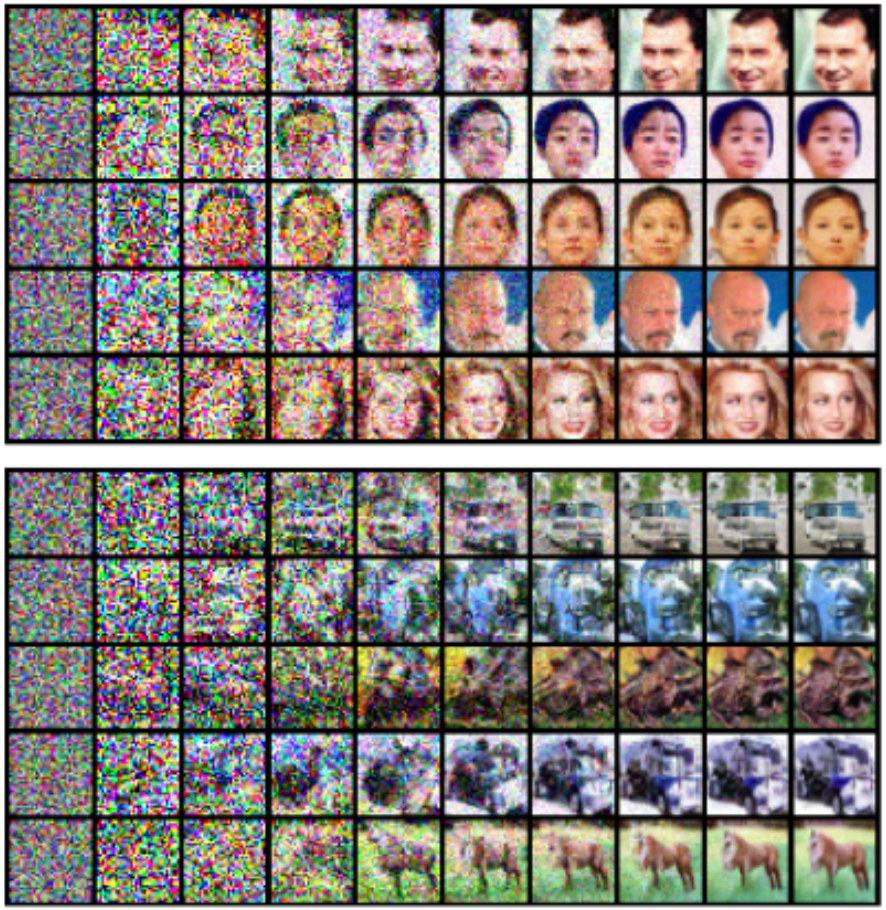


Figure 4: Intermediate samples of annealed Langevin dynamics.

Section 7

Diffusion

Subsection 1

Diffusion Model

Latent Variable Model

隐变量模型（Latent Variable Model）：

$$
\begin{aligned}
logp(x) &= \int p(x,z)dz \\
&= log \int p(x|z)p(z)dz \\
&= log \int \frac{q(z|x)}{q(z|x)} p(x|z)p(z)dz \\
&\geq E_{z\sim q(z|x)}[logp(x|z)] - E_{z\sim q(z|x)}[logq(z|x) - logp(z)]
\end{aligned}
$$

Diffusion Model

Diffusion 与 Flow 和 Score-Matching 类似，分为前向过程和反向过程。

- 前向过程：

  通过对原数据不断的加入固定分布的噪声，一直到整个数据近似于服从先验的噪声分布。

- 反向过程：

  通过网络学习每一步加入的噪声信息，然后不断减噪。

Diffusion Model

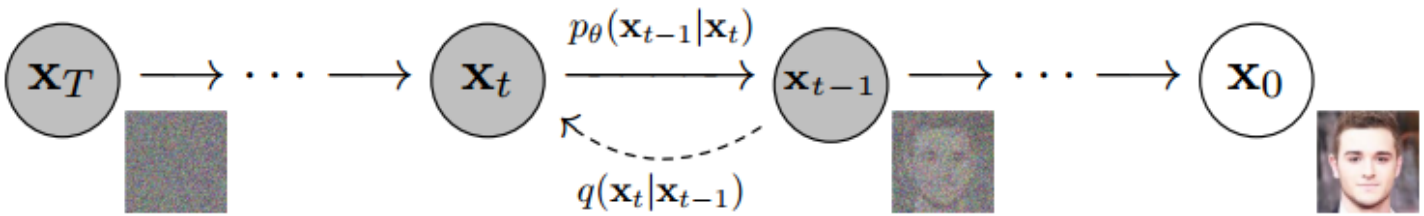


Figure 2: The directed graphical model considered in this work.

Diffusion Model

前向过程（Forward Processes）：

在 Diffusion 模型中，前向过程被建模成 Markov Chain：

$$q(z_{1:T}|x) = q(z_1|x)(\prod_{i=2}^{T} q(z_i|z_{i-1}))$$

并且中间的每一个条件分布都是高斯分布：

$$q(z_i|z_{i-1}) = N(z_i|\sqrt{1-\beta_i}z_{i-1}, \beta_i I)$$

即：

$$z_i = \sqrt{1-\beta_i}z_{i-1} + \sqrt{\beta_i} \odot \epsilon_i$$

其中 $\epsilon_i \sim N(0, I)$

Diffusion Model

反向过程（Reverse Processes）：

求积分的过程通过变分推断近似为优化的过程

$$p(x) = \int p(x, z_{1:T}) dz_{1:T}$$

Diffusion 同样把反向过程也建模为一个 Markov Chain：

$$p(x, z_{1:T}) = p(x|z_1)(\prod_{i=1}^{T-1} p(z_i|z_{i+1}))p(z_T)$$

Diffusion Model

那么 Diffusion 的优化目标就是反向过程网络学到的噪声参数要与前向过程固定的参数要尽可能的相同。

$$\mathcal{L}(\mathbf{x};\theta,\phi) = \mathbb{E}_{Q_\phi(\mathbf{z}_{1:T}|\mathbf{x})}\left[\ln p_\theta(\mathbf{x}|\mathbf{z}_1)\right] - \sum_{i=2}^{T-1}\mathbb{E}_{Q_\phi(\mathbf{z}_{-i}|\mathbf{x})}\left[KL\left[q_\phi(\mathbf{z}_i|\mathbf{z}_{i-1})||p_\theta(\mathbf{z}_i|\mathbf{z}_{i+1})\right]\right] - \mathbb{E}_{Q_\phi(\mathbf{z}_{-i}|\mathbf{x})}\left[KL\left[q_\phi(\mathbf{z}_T|\mathbf{z}_{T-1})||p_\theta(\mathbf{z}_T)\right]\right] - \mathbb{E}_{Q_\phi(\mathbf{z}_{-i}|\mathbf{x})}\left[KL\left[q_\phi(\mathbf{z}_1|\mathbf{x})||p_\theta(\mathbf{z}_1|\mathbf{z}_2)\right]\right]$$

Subsection 2

Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models

以 Denoising Diffusion Probabilistic Models（DDPM）为例，看一个具体的 Diffusion 模型。

DDPM 采用了与 NCSN 几乎类似的思路，与其不同的是前向加噪和反向减噪的过程需要符合 Markov Chain。

Denoising Diffusion Probabilistic Models

前向过程：

令 $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$，就能得到：

$$q(x_t|x_0) = N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)I)$$

也就是说在固定噪声的情况下，我们可以直接得到任意 t 步高斯模糊的结果。

Denoising Diffusion Probabilistic Models

反向过程:

反向过程则是需要网络去学到每一步加入的噪声的均值和方差，然后去掉这个噪声。

$$\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},t)\right\|^2\right]$$

Denoising Diffusion Probabilistic Models

DDPM 训练和采样流程：

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** $\mathbf{x}_0$

Subsection 3

Neural SDE

Neural SDE

NCSN 和 DDPM 的前向过程都可以写作如下形式：

$$x_{t+1} = h(t)x_t + g(t) \odot w$$

类比 Neural ODE，把上式看作 dt=1 的情况就可以得到：

$$x_{t+1} - x_t = [h(t)x_t - x_t + g(t) \odot w]dt$$

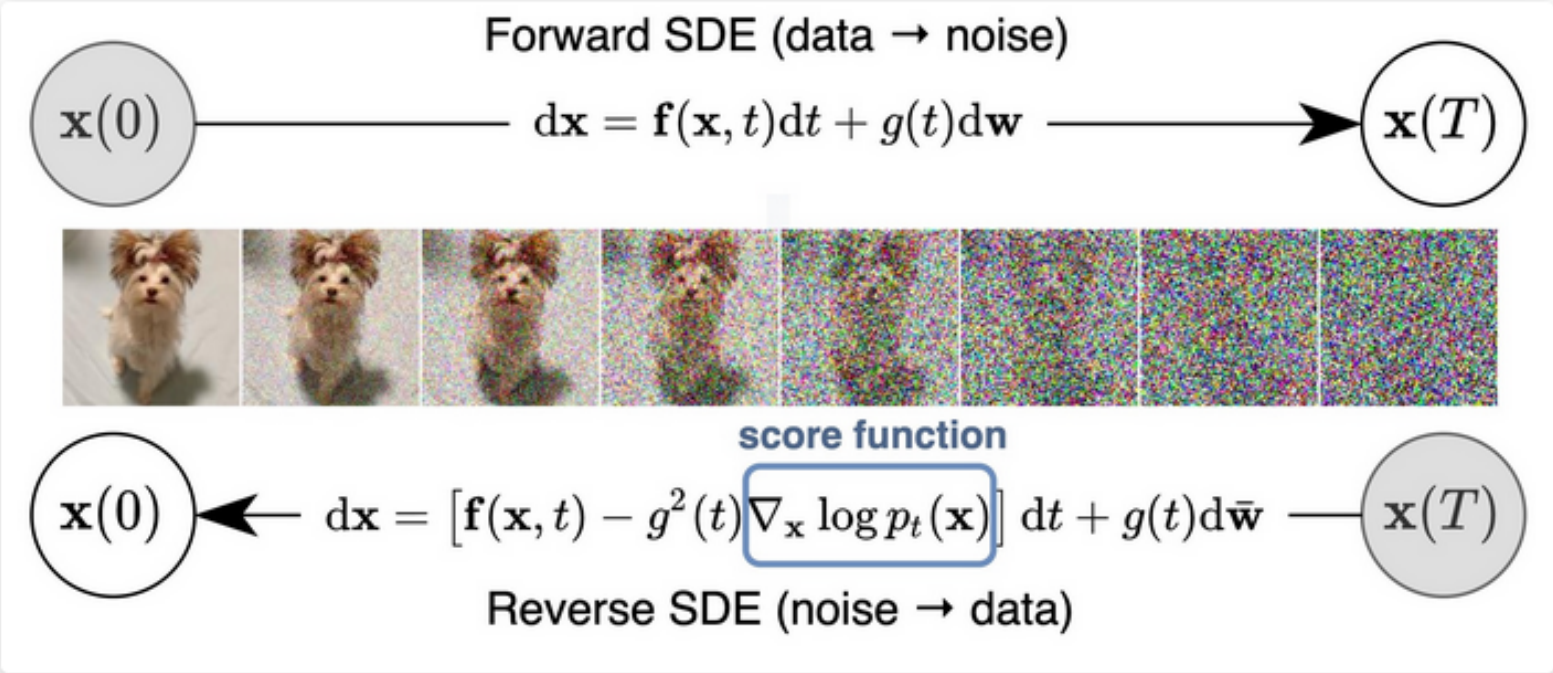
$$dx = f(x, t)dt + g(t)dw$$

这就是一个随机微分方程（Stochastic Differential Equation，SDE）

反向的过程也可以看作是一个随机微分方程：

$$dx = [f(x, t) - g(t)^2 \nabla_x log p_t(x)]dt + g(t)dw$$

Neural SDE



Solving a reverse SDE yields a score-based generative model. Transforming data to a simple noise distribution can be accomplished with an SDE. It can be reversed to generate samples from noise if we know the score of the distribution at each intermediate time step.

Neural SDE & Neural ODE

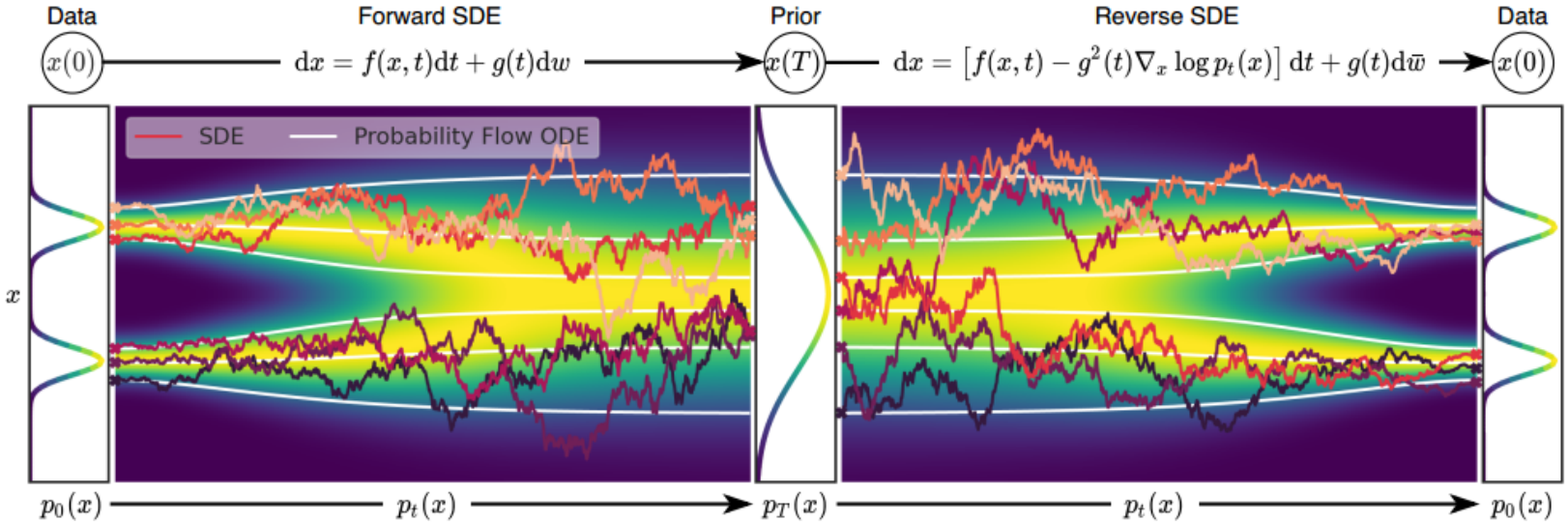


Figure 2: **Overview of score-based generative modeling through SDEs**. We can map data to a noise distribution (the prior) with an SDE (Section 3.1), and reverse this SDE for generative modeling (Section 3.2). We can also reverse the associated probability flow ODE (Section 4.3), which yields a deterministic process that samples from the same distribution as the SDE. Both the reverse-time SDE and probability flow ODE can be obtained by estimating the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ (Section 3.3).

Thanks!