Stochastic Dynamics and Machine Learning

— Notes for a Seminar Course

Yu CAO

Shanghai Jiao Tong University

Spring, 2025

Abstract:

This material is part of a seminar course/mini-course designed for undergraduate students in the School of Mathematical Sciences at Shanghai Jiao Tong University in Spring 2025. A basic understanding of ordinary differential equations (ODEs), partial differential equations (PDEs), and probability theory is assumed. Familiarity with stochastic processes is preferable, but key concepts will be briefly reviewed in Chapter 1.

Over the course of these four weeks, the lectures will cover the following topics:

- Chapter 1: A review of basic stochastic processes and key preliminary results.
- Chapters 2: An exploration of neural dynamics in control problems. As a remark, this chapter does not aim to discuss the mathematics of control theory, but rather a discussion of backpropagation, together with its continuous-time and stochastic generalizations.
- Chapter 3: An exploration of neural dynamics in generative problems, including discussions on normalizing flows and probability flows, along with simple applications.
- Chapters 4: Fundamental results in Langevin dynamics, its origin from open classical systems and applications in sampling.
- Chapter 5: Discuss how one can use Langevin dynamics for generative models, specifically, the diffusion model.

The source code of some examples will be available at `https://gitee.com/yucaoyc/Spring25_StocDynML`.

If you spot any errors or have suggestions for improvement, please don't hesitate to email me at yucao@sjtu.edu.cn.

Contents

Chapter 1

Introduction and Preliminaries

1.1 Introduction

The study of dynamical systems and machine learning has been very closely connected [8]. This is a seminar course for stochastic dynamics (including deterministic dynamics) and recent machine learning techniques. Before we delve into topics, we shall review results about dynamical systems, e.g, notions like evolution operator, and review basic concepts from Stochastic Differential Equations (SDEs). Then we will provide a very brief introduction to machine learning. As a remark, this mini-course does not aim to be an introductory course for machine learning.

1.2 Deterministic Dynamics and Evolution Operators

Suppose we consider a deterministic dynamics,

$$\partial_t \rho(t) = \mathcal{L}(t)\rho(t),$$

where the state space of $\rho(t)$, denoted as $\mathcal{S}$, can be a finite-dimensional vector space, or a proper functional space. The operator $\mathcal{L}(t) : \mathcal{S} \to \mathcal{S}$ can be time-dependent. Then the solution can be written as

$$\rho(t) = U(t,s)\ \big(\rho(s)\big), \qquad U(t,s) = \mathcal{T} e^{\int_s^t \mathcal{L}(r)\ \mathrm{d}r},$$

where $\mathcal{T}$ is the time chronological ordering operator, namely

$$\mathcal{T}\big(A(t_1)A(t_2)\big) = \begin{cases} A(t_1)A(t_2) & \text{if } t_1 \geq t_2 \\ A(t_2)A(t_1) & \text{if } t_1 < t_2 \end{cases}$$

for appropriate time-dependent operators $t \mapsto A(t)$. $U(t,s)$ can be approximated by

$$U(t,s) \approx e^{\Delta t \mathcal{L}(t_{N-1})} e^{\Delta t \mathcal{L}(t_{N-2})} \cdots e^{\Delta t \mathcal{L}(t_0)},$$

where $t_k = s + k\Delta t$ and $\Delta t = \frac{t-s}{N}$. This is well-known as Lie's first-order algorithm. Generalizations include Strang splitting scheme (2 nd order), Lie-Trotter-Suzuki splitting schemes (arbitrary order), and et al.

These abstract results are valid for both general ODEs and PDEs. We remark that if $\mathcal{L}(t)$ is anti-Hermitian (as in the case of Schrödinger's equation in quantum mechanics), then $U(t,s)$ represents unitary dynamics.

Lemma 1.1.

$$\frac{\mathrm{d}}{\mathrm{d}t} U(t,s) = \mathcal{L}(t)U(t,s), \qquad \frac{\mathrm{d}}{\mathrm{d}s} U(t,s) = U(t,s)\big(-\mathcal{L}(s)\big). \tag{1.1}$$

A common situation is that $\mathcal{L}$ can be split into two parts:

$$\mathcal{L}(t) = \mathcal{L}_A(t) + \mathcal{L}_B(t).$$

A useful tool is called the Duhamel's principle:

Theorem 1.2 (Duhamel's principle)**.**

$$\rho(t) = U_A(t,s)\rho(s) + \int_s^t U_A(t,r)\ \mathcal{L}_B(r)\ \rho(r)\ \mathrm{d}r, \tag{1.2}$$

where $U_A(t,s) := \mathcal{T}\Big(e^{\int_s^t \mathcal{L}_A(r)\ \mathrm{d}r}\Big)$ is the evolution operator for $\mathcal{L}_A$ only.

Proof. Hint: Duhamel's principle can be proved by introducing integrating factors; consider estimating $\frac{\mathrm{d}}{\mathrm{d}t}\big(U_A(s,t)\rho(t)\big)$ and write it in the integral form. □

1.3 Stochastic Differential Equations and Fokker-Planck Equations

This section is prepared by partially referring to the textbook [12].

1.3.1 Brownian motion

Back into 1820s, Robert Brown (a Scottish botanist) discovered the motion of pollen particle in fluid[1]. Later Albert Einstein discovered the equation of Brownian motion in 1905 (the other two most famous results in the same year by Einstein is special relativity and light quanta) [32]. The mathematical theory of Brownian motion was later formulated by Wiener [12]. Brownian motion is arguably the most famous continuous-time stochastic process. But what is a stochastic process then?

Definition 1.3 (An informal definition of stochastic process)**.** Suppose there is a probability space $(\Omega, \mathcal{F}, \mathcal{P})$, where Ω is the event space, $\mathcal{F}$ is the filtration, and $\mathcal{P}$ is the probability measure [2]. A stochastic process $\big\{X(t)\big\}_{t\in\mathfrak{T}}$ is a time-parameterized family of random variables, where $\mathfrak{T}$ is the time index, which can be either discrete or continuous.

For instance, the location of a pollen inside the fluid at time t is a random variable, which can be modeled via $X(t)$; for the event $\omega \in \Omega$, $X(t,\omega)$ is the location of pollen particle if ω "happens". The stock price of amazon in year 2030 is random, which can be modeled via a random variable $X(t = 2030)$; the stock price of Tencent in year 2030 is also random, which can be modeled via a random variable $\widetilde{X}(t = 2030)$; clearly, these stock prices can be imagined to share the same probability space Ω (which can be thought of as all possibilities of stocks on earth, even though we cannot literally list all outcomes on a paper). This example explains that for the same space Ω, we can have different stochastic processes.

Definition 1.4 (Brown motion)**.** Brownian motion is a stochastic process with the following properties:

1. $B(t) - B(s)$ $(t > s)$ is independent of all histories up to time s;
2. $B(t) - B(s)$ is a normal random variable $\mathcal{N}(0, t-s)$;
3. $B(t)$ is continuous in time t.

Therefore, the transition probability

$$P(y,t|x,s) = P\big(B(t) = y | B(s) = x\big) = \frac{1}{\sqrt{2\pi(t-s)}} \exp\Big(-\frac{(y-x)^2}{2(t-s)}\Big).$$

As you may observe in Figure 1.1, the Brownian motion path is a bit "rough".

[1] See e.g., https://www.britannica.com/science/Brownian-motion

[2] $\mathcal{F}$ is called the filtration, which can be regarded as a technical part to adapt this notion into the rigorous measure theory; we will disregard the details for simplicity.

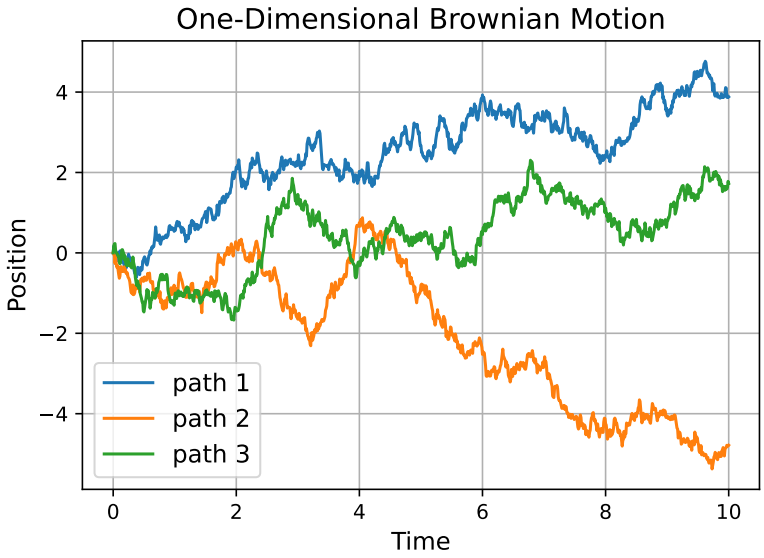


Figure 1.1: Example of Brownian motion trajectories

Theorem 1.5 (Properties of Brownian motion; see [12, Chp. 3, page 64])**.**

(i) Brownian motion trajectories are continuous but not differentiable.

(ii) Brownian motion trajectories have infinite total variation.

1.3.2 Itô and Stratonovich integrals

Now we need to make sense of stochastic calculus in a heuristic way. For any given continuous function f, the integral of Brownian motion can be clearly approximated as follows:

$$\int_0^T f\big(B(s)\big)\,\mathrm{d}s = \lim_{N\to\infty} \frac{1}{N}\sum_{k=0}^{N-1} f\big(B(s_k)\big), \qquad (s_k = k\Delta s,\ \Delta s = \frac{T}{N}).$$

As Brownian motion $s \mapsto B(s,\omega)$ is a continuous function (also uniformly continuous), the above definition (in the pointwise limit sense) is mathematically valid.

Remark. To understand why we need this, imagine the following artificial situation. Suppose a certain stock price exactly follows the Brownian motion, and someone bet with you that for every minute in the next hour, for each outcome $B(s)$ you will earn $f\big(B(s)\big)\Delta s$ dollars (of course if the value is negative, it means you lose dollars). The time interval is $\frac{1}{60} = \Delta s$, and your total asset is exactly $\frac{1}{N}\sum_{k=0}^{N-1} f\big(B(s_k)\big)$. Of course, it is reasonable to model this problem using continuous time as time interval Δs is small. This will motivates the definition of the integral $\int_a^b f\big(B(s)\big)\,\mathrm{d}s$ on the interval $\big[a,b\big]$.

Exercise 1.6. Compute the probability of events $\Big\{\omega : \int_0^1 B(s,\omega)\,\mathrm{d}s > 1\Big\}$.

Proof. Hint: first prove that $\int_0^1 B(s,\omega)\,\mathrm{d}s$ is a normal random variable with mean 0 and variance $1/3$. □

However, things becomes tricky when we want to take the integral of "$\int_0^T X(t)\ \times\ \mathrm{d}B(t)$". In this case,

$\mathrm{d}B(t)$ kind of plays the role as time-rescaling. Suppose that $G(t)$ is a given differentiable function, then

$$\begin{aligned}\int_0^T X(t)\ \mathrm{d}G(t) &= \int_0^T X(t)\ G'(t)\ \mathrm{d}t \approx \sum_{k=0}^{N-1} X(s_k) \int_{s_k}^{s_{k+1}} G'(s)\ \mathrm{d}s \\ &= \sum_{k=0}^{N-1} X(s_k)\Big(G\big(s_{k+1}\big) - G\big(s_k\big)\Big).\end{aligned} \tag{1.3}$$

Similarly, we can also approximate the integral via

$$\begin{aligned}\int_0^T X(t)\ \mathrm{d}G(t) &= \int_0^T X(t)G'(t)\ \mathrm{d}t \approx \sum_{k=0}^{N-1} \frac{X(s_k) + X\big(s_{k+1}\big)}{2} \int_{s_k}^{s_{k+1}} G'(s)\ \mathrm{d}s \\ &= \sum_{k=0}^{N-1} \frac{X(s_k) + X\big(s_{k+1}\big)}{2}\ \Big(G\big(s_{k+1}\big) - G\big(s_k\big)\Big).\end{aligned} \tag{1.4}$$

Exercise 1.7. If G has finite total variation (e.g., when G is continuously differentiable), then the approximations in (1.3) and (1.4) are the same in the large N limit.

However, the above two approximations are distinct for Brownian motions (as Brownian motion has infinite total variation).

$$\begin{aligned}\text{Itô integral}: \int_0^T X(t)\ \mathrm{d}B(t) &= \lim_{N\to\infty} \sum_{k=0}^{N-1} X(s_k)\ \Big(B\big(s_{k+1}\big) - B\big(s_k\big)\Big) \\ \text{Stratonovich integral}: \int_0^T X(t) \circ \mathrm{d}B(t) &= \lim_{N\to\infty} \sum_{k=0}^{N-1} \frac{X\big(s_k\big) + X\big(s_{k+1}\big)}{2}\ \Big(B\big(s_{k+1}\big) - B\big(s_k\big)\Big)\end{aligned} \tag{1.5}$$

Using the above definitions, we can verify the following two results.

Example 1.8. For Itô and Stratonovich integrals, we have the following results respectively:

$$\mathbb{E}\Big(\int_0^T B(t)\ \mathrm{d}B(t)\Big) = 0, \qquad \mathbb{E}\Big(\int_0^T B(t) \circ \mathrm{d}B(t)\Big) = \frac{T}{2}.$$

Therefore, these two definitions are clearly different.

Lemma 1.9 (Properties of Itô integral)**.** Suppose that $X(t)$ is $\mathcal{F}(t)$-adapted, then

- **(Zero mean)** $\mathbb{E}\left(\int_0^T X(t)\ \mathrm{d}B(t)\right) = 0.$
- **(Itô isometry)** $\mathbb{E}\left(\int_0^T X(t)\ \mathrm{d}B(t)\right)^2 = \int_0^T \mathbb{E}\big(X(t)^2\big)\ \mathrm{d}t.$

These can be proved (at least formally) directly using definitions (1.5). The proofs are left as exercises.

1.3.3 Stochastic differential equations (SDEs)

Definition 1.10 (Stochastic Differential Equations, SDEs)**.** Suppose a stochastic process $X(t)$ satisfies the following:

$$X(t) = X(0) + \int_0^t b\big(X(s), s\big)\ \mathrm{d}s + \int_0^t \sigma\big(X(s), s\big)\ \mathrm{d}B(s) \qquad \text{with probability one,} \tag{1.6}$$

where $b : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$, $\sigma : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^{d\times d}$, [3] then $X(t)$ is called the solution to the following SDE

$$\mathrm{d}X(t) = b\big(X(t), t\big)\ \mathrm{d}t + \sigma\big(X(t), t\big)\ \mathrm{d}B(t). \tag{1.7}$$

If the integral inside (1.6) is replaced by Stratonovich integrals, then the corresponding SDE is denoted as

$$\mathrm{d}X(t) = b\big(X(t), t\big)\ \mathrm{d}t + \sigma\big(X(t), t\big)\ \circ \mathrm{d}B(t). \tag{1.8}$$

[3] Diffusion coefficient σ does not necessarily needs to be a square matrix, but we use simpler cases without loss of generality.

Theorem 1.11 (Relationship between Itô and Stratonovich SDEs)**.**

(i) Suppose $X(t)$ follows the Stratonovich SDEs

$$\mathrm{d}X(t) = b\big(X(t),t\big)\ \mathrm{d}t + \sigma\big(X(t),t\big)\ \circ \mathrm{d}B(t),$$

then $X(t)$ also follows the following Itô SDE (with a different drift function):

$$\mathrm{d}X(t) = \big(b(X(t),t) + \theta(X(t),t)\big)\ \mathrm{d}t + \sigma(X(t),t)\ \mathrm{d}B(t), \tag{1.9}$$

where $\theta_i\big(X(t),t\big) := \frac{1}{2}\sum_{j,k}\partial_{x_k}\sigma_{i,j}\big(X(t),t\big)\sigma_{k,j}\big(X(t),t\big)$.

(ii) For 1D case ($d=1$), one has the following simplified expression:

$$\mathrm{d}X(t) = \Big(b\big(X(t),t\big) + \frac{1}{2}\partial_x\sigma\big(X(t),t\big)\sigma\big(X(t),t\big)\Big)\ \mathrm{d}t + \sigma\big(X(t),t\big)\ \mathrm{d}B(t).$$

Proof. For Stratonovich SDEs,

$$\begin{aligned}
\Delta X(t) &= b(X(t),t)\Delta t + \frac{\sigma(X(t+\Delta t),t+\Delta t)+\sigma(X(t),t)}{2}\Delta B(t) + o(\Delta t)\\
&= b(X(t),t)\Delta t + \sigma(X(t),t)\Delta B(t) + \frac{\sigma(X(t+\Delta t),t+\Delta t)-\sigma(X(t),t)}{2}\Delta B(t) + o(\Delta t)\\
&= b(X(t),t)\Delta t + \sigma(X(t),t)\Delta B(t) + \theta(X(t),t) + o(\Delta t),
\end{aligned}$$

where

$$\begin{aligned}
\theta_i(X(t),t) =& \sum_j \frac{\sigma_{i,j}(X(t+\Delta t),t+\Delta t)-\sigma_{i,j}(X(t),t)}{2}\big(\Delta B(t)\big)_j\\
=& \frac{1}{2}\sum_{j,k,\ell}\partial_{x_k}\sigma_{i,j}\big(X(t),t\big)\sigma_{k,\ell}\big(X(t),t\big)\big(\Delta B(t)\big)_\ell\big(\Delta B(t)\big)_j + o(\Delta t)\\
=& \frac{1}{2}\sum_{j,k}\partial_{x_k}\sigma_{i,j}\big(X(t),t\big)\sigma_{k,j}\big(X(t),t\big)\Delta t + o(\Delta t)
\end{aligned}$$

where in the last line, we used the fact that $\big(\mathrm{d}B_j(t)\mathrm{d}B_k(t)\big) = \delta_{j,k}\mathrm{d}t$ (which can be more rigorously established using the law of large numbers). The proof is complete after simplifying notations.

□

Theorem 1.12 (Itô's formula and Stratonovich's formula)**.** Suppose $f \in C^2(\mathbb{R}^d\times\mathbb{R})$.

(i) Suppose $X(t)$ follows the Itô SDE (1.7), then

$$\begin{aligned}
\mathrm{d}f\big(X(t),t\big) &= \partial_t f\big(X(t),t\big)\ \mathrm{d}t + \nabla_x f\big(X(t),t\big)\cdot\mathrm{d}X(t) + \frac{1}{2}(\sigma\sigma^\mathsf{T})\big(X(t),t\big) : \nabla_x^2 f\big(X(t),t\big)\ \mathrm{d}t\\
&\equiv \partial_t f\big(X(t),t\big)\ \mathrm{d}t + \nabla_x f\big(X(t),t\big)\cdot b\big(X(t),t\big)\ \mathrm{d}t + \big(\nabla_x f\big(tX(t),t\big)\big)^\mathsf{T}\sigma\big(X(t),t\big)\ \mathrm{d}B(t)\\
&\quad + \frac{1}{2}(\sigma\sigma^\mathsf{T})\big(X(t),t\big) : \nabla_x^2 f\big(X(t),t\big)\ \mathrm{d}t.
\end{aligned} \tag{1.10}$$

We will denote $D(x,t) := \frac{1}{2}(\sigma\sigma^\mathsf{T})(x,t)$ and the notation $D:\nabla^2 f$ means $\sum_{i,j=1}^d D_{i,j}\nabla_{x_i,x_j}f$.

(ii) Suppose $X(t)$ follows the Stratonovich SDE (1.8), then

$$\begin{aligned}
\mathrm{d}f\big(X(t),t\big) &= \partial_t f\big(X(t),t\big)\ \mathrm{d}t + \nabla_x f(X(t),t)\cdot\mathrm{d}X(t)\\
&\equiv \partial_t f\big(X(t),t\big)\ \mathrm{d}t + \nabla_x f\big(X(t),t\big)\cdot b\big(X(t),t\big)\ \mathrm{d}t + \Big(\nabla_x f\big(X(t),t\big)\Big)^\mathsf{T}\sigma\big(X(t),t\big)\circ\mathrm{d}B(t).
\end{aligned} \tag{1.11}$$

Proof. Below are heuristic proofs for illustrating main ideas.

Proof of Part (i). By Taylor's expansion,

$$\begin{aligned}
&f\big(X(t+\Delta t),t+\Delta t\big)-f\big(X(t),t\big)\\
&=\partial_t f\big(X(t),t\big)\Delta t+\nabla_x f\big(X(t),t\big)\cdot\big(X(t+\Delta t)-X(t)\big)\\
&\quad+\frac{1}{2}\big(X(t+\Delta t)-X(t)\big)^{\mathsf{T}}\nabla_x^2 f\big(X(t),t\big)\big(X(t+\Delta t)-X(t)\big)+o\big(\Delta t\big)\\
&=\partial_t f\big(X(t),t\big)\Delta t+\nabla_x f\big(X(t),t\big)\cdot\big(X(t+\Delta t)-X(t)\big)\\
&\quad+\frac{1}{2}\Delta B(t)^{\top}\sigma\big(X(t),t\big)^{\mathsf{T}}\nabla_x^2 f\big(X(t),t\big)\big(\sigma(X(t),t)\Delta B(t)\big)+o\big(\Delta t\big).
\end{aligned}$$

The cross product terms $\big(\Delta B(t)\big)_i\big(\Delta B(t)\big)_j$ $(i\neq j)$ has negligible contributions. Hence,

$$\begin{aligned}
&f\big(X(t+\Delta t),t+\Delta t\big)-f\big(X(t),t\big)\\
&=\partial_t f\big(X(t),t\big)\Delta t+\nabla_x f\big(X(t),t\big)\cdot\big(X(t+\Delta t)-X(t)\big)\\
&\quad+\frac{1}{2}\sum_{j,\ell}\Big(\nabla^2 f\big(X(t),t\big)\Big)_{\ell,j}\Big(\sigma\sigma^{\mathsf{T}}\big(X(t),t\big)\Big)_{\ell,j}\Delta t+o\big(\Delta t\big).
\end{aligned}$$

By writing the l.h.s as $\mathrm{d}f\big(X(t),t\big)\Delta t$, and by matching terms, one can obtain (1.7).

Proof of Part (ii). Suppose $X(t)$ follows the Stratonovich SDE (1.8), then it follows the SDE (1.9). By part (i), we have

$$\begin{aligned}
\mathrm{d}f\big(X(t),t\big)=&\partial_t f\big(X(t),t\big)\ \mathrm{d}t+\nabla_x f\big(X(t),t\big)\cdot\Big(b\big(X(t),t\big)+\theta\big(X(t),t\big)\Big)\ \mathrm{d}t\\
&+\big(\nabla_x f\big(X(t),t\big)\big)^{\mathsf{T}}\sigma\big(X(t),t\big)\ \mathrm{d}B(t)\\
&+\frac{1}{2}(\sigma\sigma^{\mathsf{T}})\big(X(t),t\big):\nabla_x^2 f\big(X(t),t\big)\ \mathrm{d}t.
\end{aligned}\tag{1.12}$$

Let us transform this back into Stratonovich SDE. Imagine that the corresponding Stratonovich SDE has the form

$$\begin{aligned}
\mathrm{d}f\big(X(t),t\big)=&\partial_t f\big(X(t),t\big)\ \mathrm{d}t+\nabla_x f\big(X(t),t\big)\cdot\Big(b\big(X(t),t\big)+\theta\big(X(t),t\big)\Big)\ \mathrm{d}t\\
&+\frac{1}{2}(\sigma\sigma^{\mathsf{T}})\big(X(t),t\big):\nabla_x^2 f\big(X(t),t\big)\ \mathrm{d}t+\kappa\big(X(t),t\big)\ \mathrm{d}t\\
&+\Big(\nabla_x f\big(X(t),t\big)\Big)^{\mathsf{T}}\sigma\big(X(t),t\big)\ \circ\mathrm{d}B(t),
\end{aligned}\tag{1.13}$$

where κ is a term to be determined. By following the same calculation as in Theorem 1.11, we know that the relevant Ito's SDE for the last equation is

$$\begin{aligned}
\mathrm{d}f\big(X(t),t\big)=&\partial_t f\big(X(t),t\big)\ \mathrm{d}t+\nabla_x f\big(X(t),t\big)\cdot\Big(b\big(X(t),t\big)+\theta\big(X(t),t\big)\Big)\ \mathrm{d}t\\
&+\frac{1}{2}(\sigma\sigma^{\mathsf{T}})\big(X(t),t\big):\nabla_x^2 f\big(X(t),t\big)\ \mathrm{d}t+\kappa\big(X(t),t\big)\ \mathrm{d}t\\
&+\Big(\nabla_x f\big(X(t),t\big)\Big)^{\mathsf{T}}\sigma\big(X(t),t\big)\ \mathrm{d}B(t)\\
&+\frac{1}{2}\sum_{i,j,k}\partial_{x_k}\big(\partial_{x_i}f\sigma_{i,j}\big)\sigma_{k,j}\Big|_{\big(X(t),t\big)}\mathrm{d}t.
\end{aligned}\tag{1.14}$$

By matching terms in (1.12) and (1.14), we have

$$\kappa\big(X(t),t\big)+\frac{1}{2}\sum_{i,j,k}\partial_{x_k}\big(\partial_{x_i}f\sigma_{i,j}\big)\sigma_{k,j}\Big|_{\big(X(t),t\big)}=0,$$

which means,

$$\kappa = \frac{1}{2}\sum_{i,j,k}\Big(- \partial_{i,k}f\sigma_{i,j}\sigma_{k,j} - \partial_i f\partial_k\sigma_{i,j}\sigma_{k,j}\Big) = -D : \nabla^2 f - \big(\nabla_x f\big)\cdot\theta.$$

Therefore, (1.13) can be simplified to (1.11). □

Example 1.13. A widely used example is the Ornstein-Uhlenbeck process:

$$\mathrm{d}X(t) = aX(t)\ \mathrm{d}t + \sigma\ \mathrm{d}B(t), \tag{1.15}$$

whose solution can be explicitly written as

$$X(t) = e^{at}X_0 + \sigma\int_0^t e^{a(t-s)}\ \mathrm{d}B(s).$$

The mean is $\mathbb{E}\big[X(t)\big] = e^{at}\mathbb{E}\big[X_0\big]$, the variance is

$$\mathrm{Var}\big(X(t)\big) = e^{2at}\mathrm{Var}\big(X_0\big) + \sigma^2\int_0^t e^{2a(t-s)}\ \mathrm{d}s. \tag{1.16}$$

Proof. By Lemma 1.9 and Theorem 1.12. □

Exercise 1.14. Write code to simulate the above Ornstein-Uhlenbeck (1.15) and verify (1.16) numerically.

Exercise 1.15 (1D Geometric Brownian Motion)**.** The geometric Brownian motion is the SDE behind Black–Scholes equation which is widely used as a typical model in finance:

$$\mathrm{d}X(t) = \alpha X(t)\ \mathrm{d}t + \sigma X(t)\circ\mathrm{d}B(t). \tag{1.17}$$

(i) Prove that the SDE (1.17) is equivalent to the following Itô's SDE form

$$\mathrm{d}X(t) = \big(\alpha + \frac{1}{2}\sigma^2\big)X(t)\ \mathrm{d}t + \sigma X(t)\ \mathrm{d}B(t).$$

(ii) Prove that the explicit solution of the above SDE (1.17) is

$$X(t) = X(0)\exp\big(\alpha t + \sigma B(t)\big) \equiv X(0)\exp\Big(\int_0^t \alpha + \sigma\circ\mathrm{d}B(s)\Big).$$

1.3.4 Fokker-Planck equation

The probability density function of $X(t)$, denoted as $p(x,t)$, satisfies a certain type of PDEs, well-known as Fokker-Planck equations. Let us consider the Itô SDE for example. Suppose g is any compactly supported smooth function, then

$$\int g(x)p(x,t)\ \mathrm{d}x = \mathbb{E}\Big(g\big(X(t)\big)\Big).$$

Let us take derivatives on both sides,

$$\begin{aligned}
&\Big(\int g(x)\partial_t p(x,t)\ \mathrm{d}x\Big)\Delta t + o(\Delta t) = \ \mathbb{E}\ \Big(\Delta g\big(X(t)\big)\Big)\\
&= \mathbb{E}\ \Big(\nabla_x g\big(X(t)\big)\cdot b\big(X(t),t\big) + \frac{1}{2}(\sigma\sigma^{\mathsf{T}})\big(X(t),t\big) : \nabla_x^2 g\big(X(t)\big)\ \Big)\ \Delta t + o(\Delta t)\\
&= \Big(\int \nabla_x g(x)\cdot b(x,t)p(x,t) + D_{i,j}(x,t)\nabla_{x_i,x_j}g(x)\ p(x,t)\ \mathrm{d}x\Big)\ \Delta t + o(\Delta t)\\
&= \Big(\int -g(x)\nabla_x\cdot\big(b(x,t)p(x,t)\big) + g(x)\nabla_{x_i,x_j}\big(D_{i,j}(x,t)p(x,t)\big)\ \mathrm{d}x\Big)\ \Delta t + o(\Delta t).
\end{aligned}$$

In the limit of small Δt and since g is arbitrary, one has:

Theorem 1.16. The evolution of $p(t,\cdot)$ for Itô SDE (1.7) satisfies the following PDE

$$\begin{aligned}\partial_t p(x,t) &= -\nabla_x \cdot \big(b(x,t)p(x,t)\big) + \sum_{i,j} \nabla_{x_i,x_j}\big(D_{i,j}(x,t)p(x,t)\big) \\ &\equiv -\nabla_x \cdot \big(b(x,t)p(x,t)\big) + \nabla_x^2 : \big(D(x,t)p(x,t)\big).\end{aligned} \tag{1.18}$$

Example 1.17. When $\sigma(x,t) = \sqrt{2}\ \mathbf{I}_d$ is independent of time and space, then the Fokker-Planck equation can be simplified as

$$\partial_t p(x,t) = -\nabla_x \cdot \big(b(x,t)p(x,t)\big) + \Delta p(x,t).$$

Exercise 1.18. Verify that when $b(x,t) = -\nabla U(x)$ and $\sigma(x,t) = \sqrt{2}\mathbf{I}_d$, where U is an appropriate potential function, one stationary solution of the Fokker-Planck equation is $p_\infty(x) = e^{-U(x)}/\mathcal{Z}$ where $\mathcal{Z} = \int_{\mathbb{R}^d} e^{-U(x)}\ \mathrm{d}x$ is a normalizing constant. This Itô SDE is well-known as the overdamped Langevin dynamics, which has been widely used for sampling in statistics (see Chapter 4).

1.4 A Heuristic Introduction to Machine Learning

Machine learning problems are typically formulated in terms of an optimization problem

$$\min_\theta\ L(\theta) = \sum_{i=1}^N \omega_i f_i(\theta). \tag{1.19}$$

where $\omega_i \in \mathbb{R}$ is the weight, f_i is a family of functions, and θ is the parameter to find.

The success of machine learning relies on at least the following:

- high-quality data x_i; the data can come from either realistic problems (for many AI products), or comes from generative process via some existing processes (perhaps more common in AI for Science);
- a good way to turn the original problems into an optimization problem with effective parameterization (e.g., a neural network like below); this means a good choice of f overall;
- an efficient optimization algorithm.

We will go through these ingredients for different tasks in the following chapters, as there is no universal practical way to design f and find x_i.

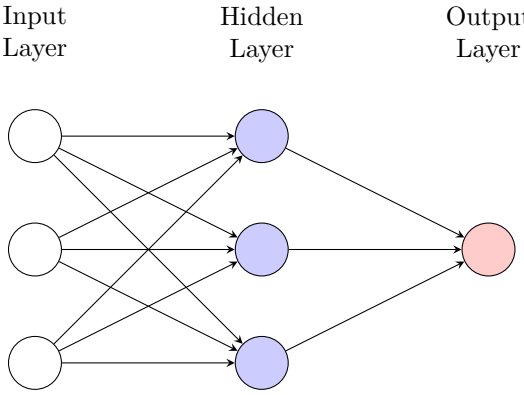


Figure 1.2: A basic structure of a two-layer neural network.

1.4.1 Discussion on the origins of data and loss functions

The data can come from either the realistic problems or from generated data.

Example 1.19 (Classification Problems)**.** Suppose x_i represents image data, and suppose $x \mapsto g(x,\theta)$ is a parameterized mapping that decides the probability of certain events to occur, e.g., the probability that the medical image contains a cancer. Suppose we have some labeled data $y_i \in \{0,1\}$ that represent whether an image contains a cancer (this is where the data comes from). Therefore, to minimize the gap between prediction and the actual data, one may want to minimize the mean-square error

$$\frac{1}{N}\sum_{i=1}^{N} \underbrace{|g(x_i,\theta)-y_i|^2}_{\text{denoted as } f_i(\theta)}.$$

Of course, one may use any monotone function φ, and design a loss function as

$$\frac{1}{N}\sum_{i=1}^{N} \underbrace{\varphi\big(g(x_i,\theta)-y_i\big)}_{\text{denoted as } f_i(\theta)}.$$

Example 1.20 (Deep Ritz [9])**.** Suppose one wants to solve the Poisson's equation

$$\begin{cases} -\Delta u = \phi & \text{in } \Omega; \\ u = 0, & \text{on } \partial\Omega. \end{cases}$$

Then one can show that it is equivalent to solve the following optimization problem

$$\min_u \ \frac{1}{2}\int_\Omega |\nabla u(x)|^2 \,\mathrm{d}x - \int_\Omega u(x)\phi(x)\,\mathrm{d}x.$$

This PDE model finds wide applications, e.g., in electrostatics. [4] To incorporate the boundary conditions, we may minimize the following in practice

$$\min_u \ \frac{1}{2}\int_\Omega |\nabla u(x)|^2 \,\mathrm{d}x - \int_\Omega u(x)\phi(x)\,\mathrm{d}x + \lambda\int_{\partial\Omega} |u(x)|^2\,\mathrm{d}x.$$

where λ is a hyper-parameter for penalty term arising from boundary conditions. Then we can parameterize the solution via a class of functions u_θ, and also deterministically or randomly pick sample points $x_i \in \Omega$ and $z_i \in \partial\Omega$ to approximate the integral via

$$\min_\theta \ \sum_i \omega_i\Big(\frac{1}{2}\big|\nabla u_\theta(x_i)\big|^2 - u_\theta(x_i)\phi(x_i)\Big) + \lambda\sum_i \overline{\omega}_i\big(u_\theta(z_i)\big)^2,$$

where $\omega_i, \overline{\omega}_i$ is the weight in the quadrature. This again falls into the above form (1.19). [5]

Example 1.21 (Physics-Informed Neural Network [25])**.** Suppose one wants to solve the differential equation

$$\begin{cases} \mathcal{L}(u) = 0, & \text{in } \Omega; \\ u = 0, & \text{on } \partial\Omega. \end{cases}$$

Imagine that we pick sample points $\{x_i\}_{i=1}^N$ from the domain Ω and $\{z_i\}_{i=1}^M$ from the boundary $\partial\Omega$, then one may minimize the following

$$\min_\theta \ \frac{1}{N}\sum_{i=1}^{N}\big(\mathcal{L}(u_\theta)(x_i)\big)^2 + \lambda\frac{1}{M}\sum_{i=1}^{M}\big(u_\theta(z_i)-0\big)^2,$$

and λ is a hyper-parameter. This again falls into the above form (1.19). [6]

[4] https://en.wikipedia.org/wiki/Poisson%27s_equation.

[5] See https://github.com/yucaoyc/math6008_num_pde/blob/main/chp08-1D-Poisson.ipynb for illustrative codes.

[6] see the above footnote.

1.4.2 Examples of architecture: residual neural network and ODEs

There are many architectures, from the simplest fully connected neural network, to convolutional neural network (CNN), to Recurrent Neural Network (RNN), to Residual Network (ResNet), and to more recent ones like Transformers. The study of their performances and their expressibility is an active research field.

Though perhaps not a one-to-one correspondence, there is a tight connection between dynamical systems and neural network architectures. We follow the discussion in [8] below. The deep Residual network takes the following form

$$y_\ell = h(z_\ell) + \mathcal{F}(z_\ell, W_\ell), \qquad z_{\ell+1} = g(y_\ell), \qquad \text{or in the form } z_{\ell+1} = g\big(h(z_\ell) + \mathcal{F}(z_\ell, W_\ell)\big),$$

where z_l are intermediate layers, and y_ℓ are auxiliary variables. Suppose $\mathcal{F}$ has a small magnitude, and $g = h$ are identity mapping, then

$$z_{\ell+1} = z_\ell + \mathcal{F}(z_\ell, W_\ell),$$

can be regarded as Euler-discretization of a certain ODE dynamics. There are other connections — normalizing flow can be regarded as discretization of ODE/probability flows; see later chapters.

1.4.3 Optimization algorithms: SGD

The Gradient Descent (GD) is perhaps one of the easiest algorithms to solve an optimization problem. However, GD algorithm is well-known to fail to find a good estimates of global minimum when the loss function is not convex. For machine learning problems, typically, we use Stochastic Gradient Descent (SGD), or other more complicated algorithms [26]. We shall simply explain SGD for illustration.

Suppose the loss function is (1.19) and suppose the weight $\omega_i = 1/N$ for simplicity. The SGD refers to the following iterative algorithms

$$\theta_{k+1} = \theta_k - h_k \nabla_\theta f_{i_k}(\theta_k),$$

where the index i_k is randomly drawn with probability $\omega_i = 1/N$, and $h_k > 0$ is called the learning rate. Note that $\nabla f_{i_k}(\theta)$ is random variable with unbiased mean $\nabla_\theta L(\theta)$:

$$\mathbb{E}\big[\nabla_\theta f_{i_k}(\theta)\big] = \nabla_\theta L(\theta).$$

Of course, there is an intermediate region between GD and SGD, which is called Mini-batch gradient descent — which is less fluctuation than SGD, and has more exploring power than GD [26].

Theorem 1.22 (See e.g., [27])**.** Assume that

- The loss function L is bounded from below by L^* (the global minimum value);
- The gradient of loss function ∇L is L-Lipschitz;
- $\mathbb{E}\big[\|\nabla f_i(\theta)\|^2\big] \le \sigma^2$ for any θ.

Then

$$\min_{k=0,1,\cdots,K-1} \mathbb{E}\|\nabla L(\theta_k)\|^2 \le \frac{L(\theta_0) - L^*}{\sum_{k=0}^{K-1} h_k} + \frac{\mathsf{L}\sigma^2}{2} \frac{\sum_{k=0}^{K-1} h_k^2}{\sum_{k=0}^{K-1} h_k}.$$

If $h_k = \mathsf{h}$ is independent of k, then the error scales like $\mathcal{O}(\frac{1}{K} + \mathsf{h})$.

Proof. See slides 3-6 from [27]. □

Exercise 1.23. When $h_k = \mathsf{h}/k$ and $h_k = \mathsf{h}/\sqrt{k}$, please find the scaling of error with respect to K. Answers can be found in [27].

Theorem 1.24 (See e.g., [27])**.** Under the same assumption as Theorem 1.22, if we further assume that $\nabla^2 L(\theta) \geq \mu\, \mathbf{I}$ for any θ (namely, the strongly convex assumption holds) and assume that $h_k = \mathsf{h} < \frac{2}{\mu}$ (small enough), then

$$\mathbb{E}\Big[\|\theta_k - \theta^*\|^2\Big] \leq (1-2\mathsf{h}\mu)^k \|\theta_0 - \theta^*\|^2 + \frac{\mathsf{h}\sigma^2}{2\mu}, \qquad \forall k.$$

This suggests that for a large enough k, θ_k is close to θ^* with distance $\sqrt{\mathsf{h}\sigma^2/(2\mu)}$. By Markov inequality,

$$\mathbb{P}\big(\|\theta_k - \theta^*\| \geq \epsilon\big) \leq \frac{\mathbb{E}\big[\|\theta_k - \theta^*\|^2\big]}{\epsilon^2} \approx \frac{\mathsf{h}\sigma^2}{2\mu\epsilon^2}.$$

For instance, when $\epsilon = 10\sqrt{\mathsf{h}\sigma^2/(2\mu)}$, one has $\mathbb{P}\big(\|\theta_k - \theta^*\| \geq \epsilon\big) \leq 1\%$ for large enough k.

Proof.

$$\begin{aligned}
\mathbb{E}\big[\|\theta_{k+1} - \theta^*\|^2\big] &= \mathbb{E}\big[\|\theta_k - \theta^* - \mathsf{h}\nabla_\theta f_{i_k}(\theta_k)\|^2\big] \\
&= \mathbb{E}\Big[\|\theta_k - \theta^*\|^2 - 2\mathsf{h}(\theta_k - \theta^*)\cdot \nabla_\theta f_{i_k}(\theta_k) + \mathsf{h}^2 \|\nabla_\theta f_{i_k}(\theta_k)\|^2\Big] \\
&\leq \mathbb{E}\big[\|\theta_k - \theta^*\|^2 - 2\mathsf{h}\mu\|\theta_k - \theta^*\|^2\big] + \mathsf{h}^2\sigma^2 \\
&\leq (1-2\mathsf{h}\mu)\mathbb{E}\big[\|\theta_k - \theta^*\|^2\big] + \mathsf{h}^2\sigma^2.
\end{aligned}$$

By the iterative relation,

$$\begin{aligned}
\mathbb{E}\big[\|\theta_k - \theta^*\|^2\big] &\leq (1-2\mathsf{h}\mu)^k \|\theta_0 - \theta^*\|^2 + \mathsf{h}^2\sigma^2\big((1-2\mathsf{h}\mu)^0 + (1-2\mathsf{h}\mu)^1 + \cdots + (1-2\mathsf{h}\mu)^{k-1}\big) \\
&\leq (1-2\mathsf{h}\mu)^k \|\theta_0 - \theta^*\|^2 + \frac{\mathsf{h}\sigma^2}{2\mu}.
\end{aligned}$$

□

1.5 Notations

By default, we take the column-based vector form rather than row-based vector form. Suppose $z \in \mathbb{R}^d \mapsto f(z) \in \mathbb{R}^d$, then $\nabla_z f(z)$ is a matrix whose $(i,j)^{\text{th}}$ element is $\partial_{z_j} f_i(z)$. Suppose $z \in \mathbb{R}^d \mapsto F(z) \in \mathbb{R}^{d\times d}$, then $\nabla_z F(z)$ is a 3-fold tensor whose $(i,j,k)^{\text{th}}$ element is $\partial_{z_k} F_{i,j}(z)$. For any arbitrary vector v, we use notation

$$\nabla F v := \Big[\sum_k \partial_{x_k} F_{i,j} v_k\Big]_{i,j}$$

which is a matrix.

1.6 Further Readings

- **Theory of Stochastic Calculus**: For further details on fundamental results of stochastic calculus, one may refer to e.g., [12, 24].
- **Numerical Methods of Stochastic Calculus**: For further details on numerical methods for stochastic dynamics, one may refer to [13].
- **Deep Learning:** For background on machine learning (in particular, deep learning), one may refer to e.g., [4]. For more optimization algorithms and practical challenges, one may refer to e.g., [26]. One may also refer to course materials e.g., by Mark Schmidt (UBC) at https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/, and by Martin Jaggi and Nicolas Flammarion (EPFL) at https://github.com/epfml/OptML_course.
- **Learning to Code a ML Project:** 深入浅出 PyTorch. https://datawhalechina.github.io/thorough-pytorch/ This is a Chinese book that introduce you to code in PyTorch for machine learning problems.

Chapter 2

Neural Dynamics for Control Problems

2.1 Introduction

Problem: For many control problems, our goal is to find a certain dynamics, such that the resulting dynamics at a certain time T satisfies a certain condition, either matching a certain value, or minimizes a certain target value (namely, loss function).

The dynamics itself may have additional constraint itself.

- **(Deterministic Network).** Suppose we consider a neural network with layer depth ℓ and width N_d and let us consider a simple setup:

  $$f = f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1$$

  where each $f_i : \mathbb{R}^{N_d} \to \mathbb{R}^{N_d}$ are parameterized functions. For simplicity, let us denote

  $$y_1 = f_1(x), \qquad y_2 = f_2(y_1), \qquad \cdots, \qquad y_\ell = f_\ell\big(y_{\ell-1}\big).$$

  For the control problem, one wants the final output state to satisfy certain criterions, e.g., one may want to minimize

  $$\min_\theta \; L(y_\ell) \tag{2.1}$$

  where L is a certain loss function, and θ are parameters inside functions f_i.

- **(Neural ODE).** If each layer takes the form of Residual Network and each $f_i \approx \mathbf{I}$, then one may regard this network as an approximation of an ordinary differential equation (ODE). Suppose we consider $z(t, \theta) : \mathbb{R} \times \mathbb{R}^{N_p} \to \mathbb{R}^{N_d}$:

  $$\frac{\mathrm{d}z(t,\theta)}{\mathrm{d}t} = f\big(z(t,\theta), t, \theta\big), \tag{2.2}$$

  where θ is the control parameter, and $(z, t, \theta) \mapsto f(z, t, \theta)$ is the ODE flow for consideration. The target is to find such a function f such that the output state $z(T, \cdot)$ satisfies certain conditions.

- **(Neural SDE).** We can further extend the setup to the probability space Ω. Suppose we consider $X(t, \theta, \omega) : \mathbb{R} \times \mathbb{R}^{N_p} \times \Omega \to \mathbb{R}^{N_d}$:

  $$\mathrm{d}X(t,\theta) = b\big(X(t,\theta), t, \theta\big)\ \mathrm{d}t + \sigma\big(X(t,\theta), t, \theta\big)\ \circ \mathrm{d}B(t), \tag{2.3}$$

  where θ is the control parameter, and $(z, t, \theta) \mapsto b(z, t, \theta)$ is the drift and $(z, t, \theta) \mapsto \sigma(z, t, \theta)$ is the diffusion coefficient. The target is again to find such a function b and σ, such that the output state $X(T, \cdot)$ satisfies certain conditions.

2.2 Backpropagation

Our goal is to find a parameter to solve (2.1). In order to use many gradient-based optimization algorithm, we have to be able to compute the gradient of $L(y_\ell)$ with respect to the parameters inside the layer. One widely used efficient method is called the backpropagation [4, Chapter 8], which we will review below. Suppose f_i has parameters $\theta_j^{(i)}$ where the index j characterize the index of parameters for this layer.

By the chain rule, when we compute, e.g.,

$$\frac{\mathrm{d}L(y_\ell)}{\mathrm{d}\theta_\lambda^{(1)}} = \sum_{j_1,j_2,\cdots,j_\ell=1,\cdots,N_d} \frac{\partial L}{\partial (y_\ell)_{j_\ell}}(y_\ell) \cdot \frac{\partial (y_\ell)_{j_\ell}}{\partial (y_{\ell-1})_{j_{\ell-1}}}(y_{\ell-1}) \cdots \frac{\partial (y_1)_{j_1}}{\partial \theta_\lambda^{(1)}}(x). \tag{2.4}$$

For the forward process, one compute y_j in the forward order; to compute the gradient, one may start with $\frac{\partial L}{\partial (y_\ell)_{j_\ell}}(y_\ell)$ and backwardly update the gradient value in a backward manner.

Example 2.1. Let us consider the fully-connected neural network without the bias term for simplicity:

$$y_k = f_k(y_{k-1}) \equiv \varsigma\big(W^{(k)} y_{k-1}\big),$$

where $W^{(k)}$ is the weight matrix with size $N_d \times N_d$ and ς is an activation function. Then

$$\frac{\partial (y_k)_{j_k}}{\partial (y_{k-1})_{j_{k-1}}}(y_{k-1}) = \varsigma'(W^{(k)} y_{k-1})_{j_k} W^{(k)}_{j_k, j_{k-1}}.$$

- As one needs to save all intermediate steps $\{y_k\}_{k=1,2,\cdots,\ell}$, one requires $\mathcal{O}(N_d \ell)$ total memory cost. As we also need to store $\left[\frac{\mathrm{d}L(y_\ell)}{\mathrm{d}\theta_\lambda^{(1)}}\right]_{\lambda=1,2,\cdots,N_d^2}$, the memory cost is $\mathcal{O}\big(N_d^2 \ell\big)$, which is essentially the number of parameters. In any case, the memory clearly scales linearly with respect to the depth ℓ.
- The total computational cost for the backpropagation of (2.4) is

  $$\mathcal{O}\big(N_d^2 \ell\big) = \mathcal{O}(\text{total number of parameters}),$$

  where the cost of N_d^2 comes from matrix-vector multiplication for each layer and inside (2.4).

Exercise 2.2 (Comparison with FDM)**.** Please compare the cost of backpropagation with the central finite difference method, e.g.,

$$\frac{\mathrm{d}L(y_\ell)}{\mathrm{d}\theta_\lambda^{(1)}} = \frac{L\Big(y_\ell\big(\theta_\lambda^{(1)} + \epsilon\big)\Big) - L\Big(y_\ell\big(\theta_\lambda^{(1)} - \epsilon\big)\Big)}{2\epsilon} + \mathcal{O}\big(\epsilon^2\big).$$

In particular, prove that to compute the gradient of the loss with respect to all parameters in weight, one requires $\mathcal{O}\big(N_d^4 \ell^2\big) = \mathcal{O}\big(\text{total number of parameters}^2\big)$; see the discussion e.g., in [4, Chapter 8].

Automatic Differentiation

Automatic differentiation is a well-implemented computer technique to compute gradient automatically based on the above chain rule. The forward-mode automatic differentiation computes (2.4) from right to left, and the reverse-mode automatic differentiation computes (2.4) from left to right. Whether one should use forward-mode or reverse-mode automatic differentiation depends on the dimension of input and output variables. For machine learning problems, one typically has a large number of parameters where the output dimension is small (or perhaps one). Then generally, reverse-mode automatic differentiation is preferred. But for a specific problem, it is better to analyze it case-by-case.

2.3 Neural ODEs

Our goal is to find the parameter θ such that

$$\min_{\theta} \; L\big(z(T,\theta)\big),$$

where L is a scalar-valued function. Suppose that we use a gradient-based optimization method, e.g., gradient descent method, then we need to estimate $\frac{\mathrm{d}L\left(z(T,\theta)\right)}{\mathrm{d}\theta}$. A core ingredient in Neural ODE [5] is to find an efficient way to compute this quantity.

Theorem 2.3 (Results in [5])**.** We consider the ODE in (2.2) with the initial time T_0 and the final time T. Suppose that $z(T_0,\theta)=z_0$ is fixed, and $L(\cdot),\ f(\cdot,\cdot,\cdot)$ are continuously differentiable. Then

$$\begin{cases} \big(\nabla_\theta L(z(T,\theta))\big)^{\mathsf{T}} = \displaystyle\int_{T_0}^{T} a(s)^{\mathsf{T}}\ \partial_\theta f\big(z(s,\theta),s,\theta\big)\ \mathrm{d}s \\ \qquad \dfrac{\mathrm{d}}{\mathrm{d}s}a(s) = -\big(\partial_z f(z(s,\theta),s,\theta)\big)^{\mathsf{T}} a(s), \qquad s\in[T_0,T]; \\ \qquad\quad a(T) = \nabla L\big(z(T,\theta)\big). \end{cases} \tag{2.5}$$

where $\partial_\theta f(z(s,\theta),s,\theta) = \begin{bmatrix}\partial_{\theta_1} f(z(s,\theta),s,\theta) & \cdots & \partial_{\theta_{N_p}} f(z(s,\theta),s,\theta)\end{bmatrix} \in \mathbb{R}^{N_d\times N_p}$, and $a(s)\in\mathbb{R}^{N_d\times 1}$. Alternatively, one could compute $\nabla_\theta L(z(T,\theta))$ via running the following system of ODEs for time $s\in[T_0,T]$,

$$\begin{cases} \dfrac{\mathrm{d}y(s)}{\mathrm{d}s} = -\big(\partial_\theta f(z(s,\theta),s,\theta)\big)^{\mathsf{T}} a(s), \qquad y(T) = 0_{N_p\times 1}; \\ \dfrac{\mathrm{d}}{\mathrm{d}s}a(s) = -\big(\partial_z f(z(s,\theta),s,\theta)\big)^{\mathsf{T}} a(s), \qquad a(T) = \nabla L\big(z(T,\theta)\big). \end{cases} \tag{2.6}$$

The output $y(T_0) = \nabla_\theta L\big(z(T,\theta)\big)$.

Exercise 2.4. How does the storage resource and computational time scale with respect to N_d and N_p?

The following proof is presented in a way different from the original paper [5]. It fully uses the language of differential equations, echoing the proposal in [8]. It avoids to discuss any derivatives along the propagation of (discretized) deep neural networks. In [8, Section 1.1], this problem was also briefly discussed in the language of functional derivatives.

2.3.1 Proof of Theorem 2.3

By direct computation,

$$\partial_{\theta_j} L\big(z(T,\theta)\big) = \sum_i \nabla_i L\big(z(T,\theta)\big)\frac{\partial z_i(T,\theta)}{\partial\theta_j}. \tag{2.7}$$

Therefore, we need to compute $\frac{\partial z_i(T,\theta)}{\partial\theta_j}$. Let us take derivative in (2.2) with respect to θ_j,

$$\frac{\mathrm{d}^2 z(t,\theta)}{\mathrm{d}t\ \mathrm{d}\theta_j} = \partial_{\theta_j} f\big(z(t,\theta),t,\theta\big) + \nabla_z f(z(t,\theta),t,\theta)\frac{\mathrm{d}z(t,\theta)}{\mathrm{d}\theta_j}.$$

Let us denote $h(t,\theta)=\frac{\mathrm{d}z(t,\theta)}{\mathrm{d}\theta_j}$. Then one has

$$\frac{\mathrm{d}}{\mathrm{d}t}h(t,\theta) = \partial_{\theta_j} f\big(z(t,\theta),t,\theta\big) + \nabla_z f(z(t,\theta),t,\theta)h(t,\theta).$$

Let us denote the evolution/solution operator $U(t_f,t_i) = e_{\mathcal{T}}^{\int_{t_i}^{t_f}\partial_z f(z(s,\theta),s,\theta)\ \mathrm{d}s}$ where $\mathcal{T}$ is the chronological time-ordering operator. Then one can show that

$$\frac{\mathrm{d}}{\mathrm{d}t}\big(U(T_0,t)h(t,\theta)\big) = U(T_0,t)\Big(-\nabla_z f(z(t,\theta),t,\theta)\Big)h(t,\theta)$$

$$
\begin{aligned}
&\quad + U(T_0,t)\Big(\partial_{\theta_j} f(z(t,\theta),t,\theta) + \nabla_z f\big(z(t,\theta),t,\theta\big)h(t,\theta)\Big) \\
&= U(T_0,t)\ \partial_{\theta_j} f(z(t,\theta),t,\theta).
\end{aligned}
$$

By rewriting this in the integral form, one has

$$
U(T_0,t)h(t,\theta) - U(T_0,T_0)h(T_0,\theta) = \int_{T_0}^{t} U(T_0,s)\ \partial_{\theta_j} f\big(z(s,\theta),s,\theta\big).
$$

Since we considered $z(T_0,\theta) = z_0$ is fixed, thus $h(T_0,\theta) = 0$, and therefore

$$
h(t,\theta) = \int_{T_0}^{t} U(t,s)\partial_{\theta_j} f\big(z(s,\theta),s,\theta\big)\ \mathrm{d}s.
$$

One can also obtain this result by applying the Duhamel's principle stated in Theorem 1.2.

By plugging this equation into (2.7), one has

$$
\begin{aligned}
\partial_{\theta_j} L\big(z(T,\theta)\big) &= \sum_i \nabla_i L\big(z(T,\theta)\big)\frac{\partial z_i(T,\theta)}{\partial \theta_j} \\
&= \nabla_i L\big(z(T,\theta)\big)\cdot h(T,\theta) \\
&= \int_{T_0}^{T} \nabla_i L\big(z(T,\theta)\big)\cdot\Big(U(T,s)\partial_{\theta_j} f\big(z(s,\theta),s,\theta\big)\Big)\ \mathrm{d}s.
\end{aligned}
$$

Let us denote $a(s)^\mathsf{T} = \nabla L\big(z(T,\theta)\big)^\mathsf{T} U(T,s)$ (an $1\times N_d$ time-dependent row vector), then

$$
\partial_{\theta_j} L\big(z(T,\theta)\big) = \int_{T_0}^{T} a(s)^\mathsf{T}\ \partial_{\theta_j} f(z(s,\theta),s,\theta)\ \mathrm{d}s.
$$

If we adopt the matrix form,

$$
\big(\nabla_\theta L(z(T,\theta))\big)^\mathsf{T} = \int_{T_0}^{T} a(s)^\mathsf{T}\ \partial_\theta f(z(s,\theta),s,\theta)\ \mathrm{d}s,
$$

where $\partial_\theta f(z(s,\theta),s,\theta) = \begin{bmatrix}\partial_{\theta_1} f(z(s,\theta),s,\theta) & \cdots & \partial_{\theta_{N_p}} f(z(s,\theta),s,\theta)\end{bmatrix} \in \mathbb{R}^{N_d\times N_p}$.

The dynamics of $a(s)$ can be easily verified as follows:

$$
\frac{\mathrm{d}}{\mathrm{d}s}a(s)^\mathsf{T} = \nabla L(z(T,\theta))^\mathsf{T} U(T,s)\big(-\partial_z f(z(s,\theta),s,\theta)\big) = -a(s)^\mathsf{T}\partial_z f(z(s,\theta),s,\theta).
$$

Or in a column form as in (2.5). The terminal condition of a can be easily checked, as well as (2.6).

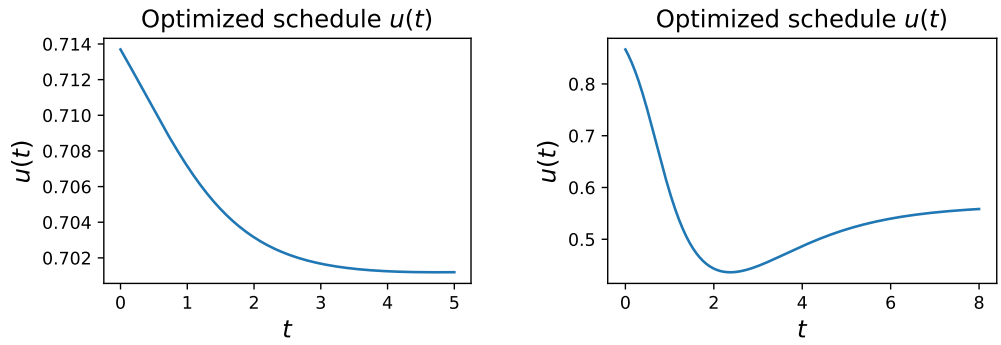
Example 2.5. For Neural ODE in (2.2), if we consider a special family $f(z,t,\theta) = -iH(t,\theta)z$, where $H(t,\theta)$ is a θ-parameterized time-dependent Hermitian operator, then for the forward process, one has

$$
\frac{\mathrm{d}z(s,\theta)}{\mathrm{d}t} = -iH(s,\theta)z(s,\theta).
$$

For illustration, let us consider the following pedagogical example:

$$
H(t,\theta) = \frac{\omega}{2}\sigma_Z + u_\theta(t)\sigma_X \equiv \begin{bmatrix}\frac{\omega}{2} & u_\theta(t)\\ u_\theta(t) & \frac{\omega}{2}\end{bmatrix}
$$

with the constraint $\|u_\theta(t)\| \le u_{\max}$ and with initial condition $z(0,\theta) = \begin{bmatrix}1\\0\end{bmatrix}$. We want to find a control u_θ such that $z(T,\theta) \approx \begin{bmatrix}0\\1\end{bmatrix}$ as much as possible. We choose $\omega = 1$, $u_{\max} = 2$, and parameterize the u_θ via fully-connected neural network. The training results for $T = 5$ and $T = 8$ are visualized in Figure below. For the case $T = 8$, the final target state is almost identical to the target state (up to a constant rescaling) with Fidelity as high as 0.999.



(a) $T = 5$, Fidelity of final state is 0.729 (b) $T = 8$, Fidelity of final state is 0.999

Figure 2.1: Training result for Example 2.5. Fidelity measures the similarity between the final state $z(T, \theta)$ and the target state. If the number of Fidelity is close to 1, it means the similarity is very high and the control is very efficient.

2.4 Neural SDEs

Lastly, we will discuss how to work on backpropagation for an SDE. We will mainly focus on the most fundamental mathematical facts without getting deep into various applications. In this section, we focus on the strong solution of SDE. The following notes is prepared partially based on [17]. In order to extend the backpropagation (adjoint equation) from ODEs to SDEs, it is easier to work with the Stratonovich SDE. For the Itô SDE, one may convert it into Stratonovich SDE by Theorem 1.11, and then apply the following procedures.

In the adjoint equation of ODEs, one needs to run dynamics backwardly. The first result below is to show how one can reverse the dynamics of Stratonovich SDE, provided that one is given access to the Brownian motion paths.

Lemma 2.6 (Reverse of Stratonovich SDE)**.** Suppose $\{X(t)\}_{t\in[0,T]}$ satisfies the Stratonovich SDE in (1.8) (also copied below herein):

$$\mathrm{d}X(t) = b\big(X(t), t\big)\ \mathrm{d}t + \sigma\big(X(t), t\big)\ \circ \mathrm{d}B(t).$$

Let $\widetilde{X}(t) := X(T - t)$, then $\widetilde{X}$ satisfies the following Stratonovich integral

$$\mathrm{d}\widetilde{X}(t) = -b\big(\widetilde{X}(t), T - t\big)\ \mathrm{d}t + \sigma\big(\widetilde{X}(t), T - t\big)\ \circ \mathrm{d}B(T - t). \tag{2.8}$$

A direct proof will be provided in § 2.4.2.

Similar to Theorem 2.3 for Neural ODEs, one can show the following:

Theorem 2.7 (Gradient estimates for neural SDE; see also [17])**.** Suppose that the initial time is T_0 and the final time is T. Suppose that $X(t, \theta)$ follows the SDE (2.3) (also copied below herein)

$$\mathrm{d}X(t,\theta) = b\big(X(t,\theta), t, \theta\big)\ \mathrm{d}t + \sigma\big(X(t,\theta), t, \theta\big)\ \circ \mathrm{d}B(t), \qquad X(T_0, \theta) = x_0 \text{ is fixed.} \tag{2.9}$$

Suppose that $L(\cdot), b(\cdot,\cdot,\cdot), \sigma(\cdot,\cdot,\cdot)$ are continuously differentiable. For simplicity, let us consider the case dimension $N_d = 1$. Then

$$\begin{cases} \nabla_\theta L\big(X(T,\theta)\big) = \displaystyle\int_{T_0}^{T} a(s)\nabla_\theta b\big(X(s,\theta), s, \theta\big)\mathrm{d}s + a(s)\nabla_\theta \sigma\big(X(s,\theta), s, \theta\big) \circ \mathrm{d}B(s), \\ \mathrm{d}a\,(s) = -a(s)\partial_x b\big(X(s,\theta), s, \theta\big)\ \mathrm{d}s - a(s)\partial_x \sigma\big(X(s,\theta), s, \theta\big) \circ \mathrm{d}B(s). \end{cases} \tag{2.10}$$

Alternatively, one could compute $\nabla_\theta L\big(X(T,\theta)\big)$ via running the following system of SDEs for time $s \in [T_0, T]$,

$$\begin{cases} \mathrm{d}y(s) = -a(s)\nabla_\theta b\big(X(s,\theta), s, \theta\big)\mathrm{d}s - a(s)\nabla_\theta \sigma\big(X(s,\theta), s, \theta\big) \circ \mathrm{d}B(s), & y(T) = 0_{N_p} \\ \mathrm{d}a\,(s) = -a(s)\partial_x b\big(X(s,\theta), s, \theta\big)\ \mathrm{d}s - a(s)\partial_x \sigma\big(X(s,\theta), s, \theta\big) \circ \mathrm{d}B(s), & a(T) = L'\big(X(T,\theta)\big) \end{cases} \tag{2.11}$$

The output $y(T_0) = \nabla_\theta L\big(X(T,\theta)\big)$.

Exercise 2.8. Generalize Theorem 2.7 to a general dimension N_d.

2.4.1 Proof of Theorem 2.7

The proof is essentially the same as that for Theorem 2.3. As mentioned above, for simplicity, we shall only consider the 1D case. By direct computation,

$$\partial_{\theta_j} L\big(X(T,\theta)\big) = L'\big(X(T,\theta)\big)\frac{\partial X(T,\theta)}{\partial \theta_j}. \tag{2.12}$$

Therefore, we need to compute $\frac{\partial X(T,\theta)}{\partial \theta_j}$. For the time being, let us fix j, denote $h(t,\theta) = \frac{\mathrm{d}X(t,\theta)}{\mathrm{d}\theta_j}$, and let us take derivative in (2.9) with respect to θ_j. Then

$$\begin{aligned}\mathrm{d}h(t,\theta) =& \Big(\partial_{\theta_j} b\big(X(t,\theta),t,\theta\big) + \partial_x b\big(X(t,\theta),t,\theta\big)h(t,\theta)\Big)\ \mathrm{d}t \\ &+ \Big(\partial_{\theta_j}\sigma\big(X(t,\theta),t,\theta\big) + \partial_x\sigma\big(X(t,\theta),t,\theta\big)h(t,\theta)\Big)\ \circ \mathrm{d}B(t).\end{aligned} \tag{2.13}$$

Since the path of Stratonovich SDE is reversible for any given Brownian motion path, let us denote the evolution/solution operator $U(s_f, s_i)$ by the following: suppose we fix θ (and thus fix $X(t,\theta)$), and then for any h_0, we define $U(s_f,s_i)h_0$ as the solution at time s_f of the following SDE

$$\mathrm{d}\widetilde{h}(t) = \partial_x b\big(X(t,\theta),t,\theta\big)\widetilde{h}(t)\ \mathrm{d}t + \partial_x\sigma\big(X(t,\theta),t,\theta\big)\widetilde{h}(t)\ \circ \mathrm{d}B(t), \qquad \widetilde{h}(s_i) = h_0.$$

Lemma 2.9. For arbitrary $s_i, s_f \in [T_0, T]$, the evolution operator $U(s_f, s_i)$ has the following form:

$$U(s_f,s_i) = \exp\left(\int_{s_i}^{s_f} \partial_x b\big(X(t,\theta),t,\theta\big)\ \mathrm{d}t + \partial_x\sigma\big(X(t,\theta),t,\theta\big)\circ \mathrm{d}B(t)\right). \tag{2.14}$$

A direct proof will be provided in § 2.4.2.

Remark. This result is essentially a time-dependent generalization of Exercise 1.15.

Lemma 2.10. For a fixed s_i and consider any $t \in [T_0, T]$, then

$$\mathrm{d}U(t,s_i) = \partial_x b(X(t,\theta),t,\theta)U(t,s_i)\ \mathrm{d}t + \partial_x\sigma(X(t,\theta),t,\theta)U(t,s_i)\circ \mathrm{d}B(t).$$

For a fixed s_f, and consider any $t \in [T_0, T]$, then

$$\mathrm{d}U(s_f,t) = -U(s_f,t)\partial_x b\big(X(t,\theta),t,\theta\big)\mathrm{d}t - U(s_f,t)\partial_x\sigma\big(X(t,\theta),t,\theta\big)\circ \mathrm{d}B(t).$$

Proof. The proof of Lemma 2.9 can be easily modified to prove the conclusion. A detailed proof is skipped. □

Remark. This lemma resembles Lemma 1.1.

Then one can directly show that

$$\mathrm{d}\big(U(T_0,t)h(t,\theta)\big) = U(T_0,t)\partial_{\theta_j} b\big(X(t,\theta),t,\theta\big)\mathrm{d}t + U(T_0,t)\partial_{\theta_j}\sigma\big(X(t,\theta),t,\theta\big)\circ \mathrm{d}B(t).$$

By rewriting this in the integral form, one has

$$U(T_0,t)h(t,\theta) - h(T_0,\theta) = \int_{T_0}^{t} U(T_0,s)\ \partial_{\theta_j} b\big(X(s,\theta),s,\theta\big)\mathrm{d}s + U(T_0,s)\partial_{\theta_j}\sigma\big(X(s,\theta),s,\theta\big)\circ \mathrm{d}B(s).$$

Since we considered $X(T_0,\theta) = X_0$ is fixed, thus $h(T_0,\theta) = 0$, and therefore

$$h(t,\theta) = \int_{T_0}^{t} U(t,s)\partial_{\theta_j} b\big(X(s,\theta),s,\theta\big)\mathrm{d}s + U(t,s)\partial_{\theta_j}\sigma\big(X(s,\theta),s,\theta\big)\circ \mathrm{d}B(s).$$

By plugging this equation into (2.12), one has

$$\begin{aligned}
&\partial_{\theta_j} L\big(z(T,\theta)\big)\\
&= L'\big(X(T,\theta)\big)\frac{\partial X(T,\theta)}{\partial \theta_j}\\
&= L'\big(X(T,\theta)\big)h(T,\theta)\\
&= \int_{T_0}^{T} L'\big(X(T,\theta)\big)\Big(U(T,s)\partial_{\theta_j} b\big(X(s,\theta),s,\theta\big)\mathrm{d}s + U(T,s)\partial_{\theta_j}\sigma\big(X(s,\theta),s,\theta\big)\circ \mathrm{d}B(s)\Big).
\end{aligned}$$

Let us denote $a(s) = L'\big(X(T,\theta)\big)U(T,s)$, then

$$\partial_{\theta_j} L\big(z(T,\theta)\big) = \int_{T_0}^{T} a(s)\partial_{\theta_j} b\big(X(s,\theta),s,\theta\big)\mathrm{d}s + a(s)\partial_{\theta_j}\sigma\big(X(s,\theta),s,\theta\big)\circ \mathrm{d}B(s)$$

If we adopt the matrix form, we can obtain the formula of $\nabla_\theta L\big(z(T,\theta)\big)$ as in (2.10). The dynamics of $a(s)$ in (2.10) can be easily verified by Lemma 2.10.

2.4.2 Proof of lemmas

Proof of Lemma 2.6. Let us divide the time interval into N equally spaced grid points and let $t_k = k\Delta t$ where $\Delta t = T/N$.

$$\begin{aligned}
&\widetilde{X}(t_{k+1}) - \widetilde{X}(t_k)\\
=&X(T-t_{k+1}) - X(T-t_k)\\
=&-b\big(X(T-t_{k+1}),T-t_{k+1}\big)\Delta t - \frac{\sigma\big(X(T-t_{k+1}),T-t_{k+1}\big)+\sigma\big(X(T-t_k),T-t_k\big)}{2}\big(B(T-t_k)-B(T-t_{k+1})\big)\\
&+o(\Delta t)\\
=&-b\big(\widetilde{X}(t_k),T-t_k\big)\Delta t + \frac{\sigma\big(\widetilde{X}(t_{k+1}),T-t_{k+1}\big)+\sigma\big(\widetilde{X}(t_k),T-t_k\big)}{2}\big(B(T-t_{k+1})-B(T-t_k)\big)+o(\Delta t),
\end{aligned}$$

which leads into (2.8). □

Proof of Lemma 2.9. For simplicity of notations, we shall suppress the dependence on θ. Let us consider an arbitrary h_0 and by definition,

$$\begin{aligned}
&U(s_f+\Delta t,s_i)h_0 - U(s_f,s_i)h_0\\
=&U(s_f+\Delta t,s_f)U(s_f,s_i)h_0 - U(s_f,s_i)h_0 \qquad (\text{Denote } \widetilde{h}(s_f) = U(s_f,s_i)h_0)\\
=&\,\partial_x b\big(X(s_f),s_f\big)\widetilde{h}(s_f)\;\Delta t + \frac{1}{2}\Big(\partial_x\sigma(X(s_f),s_f)\widetilde{h}(s_f) + \partial_x\sigma\big(X(s_f+\Delta t),s_f+\Delta t\big)\widetilde{h}(s_f+\Delta t)\Big)\Delta B(s_f) + o(\Delta t)\\
=&\partial_x b\big(X(s_f),s_f\big)\widetilde{h}(s_f)\;\Delta t + \frac{1}{2}\Big(\partial_x\sigma\big(X(s_f),s_f\big)\widetilde{h}(s_f) + \partial_x\sigma\big(X(s_f+\Delta t),s_f+\Delta t\big)\widetilde{h}(s_f)\Big)\Delta B(s_f)\\
&+\frac{1}{2}\partial_x\sigma\big(X(s_f+\Delta t),s_f+\Delta t\big)\Delta B(s_f)\big(\widetilde{h}(s_f+\Delta t)-\widetilde{h}(s_f)\big) + o(\Delta t)\\
=&\partial_x b\big(X(s_f),s_f\big)\widetilde{h}(s_f)\;\Delta t + \frac{1}{2}\Big(\partial_x\sigma(X(s_f),s_f) + \partial_x\sigma(X(s_f+\Delta t),s_f+\Delta t)\Big)\Delta B(s_f)\widetilde{h}(s_f)\\
&+\frac{1}{2}\Big(\partial_x\sigma\big(X(s_f),s_f\big)\Delta B(s_f)\Big)^2\widetilde{h}(s_f) + o(\Delta t).
\end{aligned}$$

Let us denote the expression on the right hand side of (2.14) as $\overline{U}$ to distinguish expressions:

$$\begin{aligned}
&\overline{U}(s_f+\Delta t,s_i)h_0 - \overline{U}(s_f,s_i)h_0\\
=&\exp\left(\partial_x b\big(X(s_f),s_f\big)\Delta t + \frac{1}{2}\partial_x\sigma\big(X(s_f),s_f\big)\Delta B(s_f) + \frac{1}{2}\partial_x\sigma\big(X(s_f+\Delta t),s_f+\Delta t\big)\Delta B(s_f) + o(\Delta t)\right)\overline{U}(s_f,s_i)h_0
\end{aligned}$$

$$
\begin{aligned}
&\quad - \overline{U}(s_f, s_i)h_0 \\
&= \Big(\partial_x b\big(X(s_f), s_f\big)\Delta t + \frac{1}{2}\partial_x \sigma\big(X(s_f), s_f\big)\Delta B(s_f) + \frac{1}{2}\partial_x \sigma\big(X(s_f + \Delta t), s_f + \Delta t\big)\Delta B(s_f)\Big)\overline{U}(s_f, s_i)h_0 \\
&\quad + \frac{1}{2}\Big(\partial_x \sigma\big(X(s_f), s_f\big)\Delta B(s_f)\Big)^2 \overline{U}(s_f, s_i)h_0 + o(\Delta t)
\end{aligned}
$$

As the above two dynamics have the same discretization and match with each other, we can obtain (2.14) as a limit. □

2.5 Further Readings

Below we list a few related topics for possible exploration:

- [15] gave a more detailed discussion on the relation between control theory and deep learning in lecture notes for a summer school at Peking University (2020).
- When the problem involves taking higher-order derivatives, e.g.,

  $$\min_\theta \mathbb{E}_x \; g\big(\nabla_x p(\theta, x), \nabla_x^{(2)} p(\theta, x), \cdots, \nabla_x^{(k)} p(\theta, x)\big),$$

  and g is some non-linear non-negative function, then an efficient way to take derivatives becomes important. An example is to solve PDEs using PINNs (cf. Example 1.21). For Laplacian operator, the forward method can provide speed-up over backpropagation [16].
- The above results can be further generalized to Neural Jump Process, and Levy process. This is an interesting topic for exploration on your own.

Chapter 3

Neural Dynamics for Generative Problems: (I) Normalizing Flows

3.1 Background

Generative problem, in a mathematical way, means to generate samples according to a certain probability measure p_X (which could be extremely complex). Here we list two examples among many:

- Each black/white image (with n pixels) can be represented by a data point in $\mathbb{R}^n$. A task in machine learning is to find a way to generate samples according to p_X, provided that one is given a certain amount of data samples $\{x_i\}_{i=1}^N$. This is a reminiscent of the first type of data source, namely, data directly coming from realistic problems.
- For N interacting particles in statistical physics with potential U, the equilibrium distribution (also known as the Boltzmann distribution) takes the following form $p_X(x) = e^{-U(x)}/\mathcal{Z}$ where $\mathcal{Z} = \int e^{-U}$ is known as the partition function or normalization constant (depending on which research field that you come from). One goal is to find a way to generate samples according to such a distribution, which could enable the estimation of many physical quantities for equilibrium dynamics. This is a reminiscent of the second type of data source: the data is not provided in general and needs to be generated artificially in order to formulate the generative problem as a machine learning problem.

A widely studied tool for sample generation is normalizing flows. When normalizing flows have infinitely many layers and for each layer, the update is infinitesimally small, the corresponding mathematical object is simply ODEs, and also refers to as probability flow in some contexts. The following notes are prepared mainly by referring to review papers [14, 23] and [4, Chapter 18]. This topic is also tightly connected to a field called optimal transport theory; one may refer to e.g., [2, 33].

For this mini-course, we will only cover some basic concepts about normalizing flows and probability flows, as well as a little from optimal transport theory. In this notes, we will not specifically distinguish the notation of probability measures and probability densities.

3.2 Connection to Optimal Transport

As mentioned above, the goal is to find a way to generate samples according to a target measure p_X. One widely used approach is to find a mapping f such that $f(Z)$ follows the target distribution p_X. Designing a function f to generate samples using $f(Z)$ is a very classical idea, with enormous applications. For instance, *Box–Muller transform* (a backend algorithm for generating normal random variables in our computers) uses the following algorithm: generate two independent U_1 and U_2 following uniform distributions on $(0, 1)$ and then let

$$Z_0 = \sqrt{-2\ln(U_1)}\cos(2\pi U_2);$$

$$Z_1 = \sqrt{-2\ln(U_1)}\sin(2\pi U_2).$$

Then (Z_0, Z_1) are independent standard normal random variables.[1]

The prior distribution p_Z contains certain degree of freedom, but it is typically chosen as simple distributions like uniform distributions or Gaussians. The mapping f is what we need to find. Such a mapping is also well-known as the *Monge map* in some literatures, and this name dated back to the problem of finding a shortest path to move a pile of sand into another pile of sand in 18th century.

In general, if we consider general probability measures (rather than probability densities), there are examples such that one cannot find such a mapping. For instance, let $p_Z(z) = \delta_0(z)$ (a Dirac delta distribution at 0), and $p_X(x) = \frac{1}{2}\delta_1(x) + \frac{1}{2}\delta_{-1}(x)$ (half probability at 1 and half probability at -1). It is easy to see that one cannot find such a mapping f. Of course, in real problems, life is not that tough and we have or we can assume the existence of distributions, which has the following result.

Theorem 3.1 (Existence of Monge map [2, Theorem 2.3])**.** Suppose that the initial measure $\mu_Z \in \mathcal{P}_2(\mathbb{R}^n)$ is absolutely continuous with respect to Lebesgue measure, and $\mu_X \in \mathcal{P}_2(\mathbb{R}^n)$, then there exists a Monge map f such that $\mu_X = f\#\mu_Z$. [2]

Due to the above existential theorem, and also for many realistic problems, we can assume the existence of probability densities, we shall stick with this setup from now on. Let us consider the 1D case.

Example 3.2 (1D case)**.**

- Let us assume that the support of p_X, p_Z is simply $\mathbb{R}$, which means their corresponding cumulative probability density functions P_X and P_Z are strictly monotone functions and is invertible. Suppose that f is monotone increasing, then

$$f(x) = P_X^{-1} \circ P_Z(x) \tag{3.1}$$

  is a mapping that satisfies the condition. This can be easily checked: with such a choice

$$\mathbb{P}\big(f(Z) \le a\big) = \int_{\mathbb{R}} \chi_{\{f(z)\le a\}} p_Z(z)\,\mathrm{d}z = \int_{\mathbb{R}} \chi_{\{P_Z(z)\le P_X(a)\}} p_Z(z)\,\mathrm{d}z = P_X(a) = P(X \le a),$$

  which verifies that $f(Z)$ and X have the same distribution.

- Conversely, if one is given a monotone increasing and differentiable mapping f and the prior distribution p_Z, the probability density function of $X := f(Z)$ can be computed

$$p_X(x) = p_Z\big(f^{-1}(x)\big)\big(f^{-1}\big)'(x).$$

  The proof is straightforward by (3.1).

Such a result can be generalized to arbitrary dimensions. If f is invertible, we can similarly compute the density of p_X:

Theorem 3.3. Suppose the mapping f is invertible, and let $X = f(Z)$, then

$$p_X(x) = p_Z\big(f^{-1}(x)\big)\ \big|J_{f^{-1}}(x)\big|, \qquad J_{f^{-1}}(x) := \det\big(\nabla f^{-1}(x)\big), \tag{3.2}$$

and we also have

$$p_X\big(f(z)\big)|J_f(z)| = p_Z\big(z\big). \tag{3.3}$$

[1] See https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform

[2] The notation $\mathcal{P}_2(\mathbb{R}^n)$ means the space of probability measures with finite second moment. The original statement of [2, Theorem 2.3] contains a lot more interesting results, and we only selectively pick what we need.

Proof. For an arbitrary test function $\phi \in C_c^\infty(\mathbb{R}^n)$, we have

$$\begin{aligned}\int \phi(x)p_X(x)\,\mathrm{d}x = \mathbb{E}\big[\phi(X)\big] = \mathbb{E}\big[\phi(f(Z))\big] &= \int \phi\big(f(z)\big)p_Z(z)\,\mathrm{d}z \\ &= \int \phi(x)p_Z\big(f^{-1}(x)\big)\big|J_{f^{-1}}(x)\big|\,\mathrm{d}x.\end{aligned}$$

Since ϕ is arbitrary, one has the above result. The second equation can be obtained similarly. □

Remark. (Monge-Ampere equation). If $p_X = f\#p_Z$, namely, (3.3) holds, and suppose that the mapping $f = \nabla\varphi$ with φ being convex, then

$$\det\big(\nabla^2\varphi\big)p_X\big(\nabla\varphi\big) = p_Z. \tag{3.4}$$

This is well-known as Monge-Ampere equation.

3.3 Normalizing Flows

The next task is how to design some architectures for us to learn such a mapping f.

3.3.1 General architecture

For normalizing flows, a core ingredient is to represent f as the composition of a sequence of mapping f_i:

$$f(z) = f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1(z),$$

in same way as deep neural networks. The second new ingredient is that if we also want to compute the density of p_X, we'd better require f to be invertible (cf Theorem 3.3), which becomes a requirement for each f_i to be invertible. If so,

$$f^{-1}(x) = f_1^{-1} \circ f_2^{-1} \circ \cdots f_\ell^{-1}(x).$$

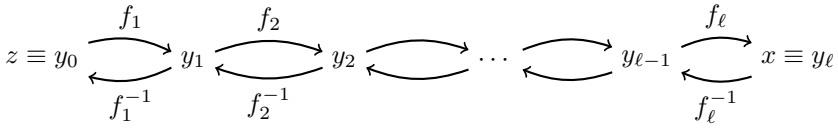


Figure 3.1: A visualization of the forward and backward directions in normalizing flows

Then we can compute the density, which involves computing the gradient of a composition of two invertible mapping.

Lemma 3.4. Suppose h_1 and h_2 are mappings, and $h = h_1 \circ h_2$, then

$$\det\big(\nabla h(x)\big) = \det\Big(\nabla h_1\big(h_2(x)\big)\Big)\det\Big(\nabla h_2(x)\Big) \implies J_{h_1\circ h_2}(x) = J_{h_1}\big(h_2(x)\big)J_{h_2}(x). \tag{3.5}$$

Proof. By chain rules, $\nabla h(x) = \nabla h_1\big(h_2(x)\big)\nabla h_2(x)$, which immediately leads into the above result. □

With such a result, one can show that

$$\begin{aligned}p_X(x) &\overset{(3.2)}{=} p_Z\big(f^{-1}(x)\big)\big|J_{f^{-1}}(x)\big| \overset{(3.5)}{=} p_Z\big(f^{-1}(x)\big)\Big|J_{f_1^{-1}}(y_1)J_{f_2^{-1}}(y_2)\cdots J_{f_\ell^{-1}}(x)\Big| \\ &= p_Z\big(f^{-1}(x)\big)\left|\prod_{i=1}^{\ell} J_{f_i^{-1}}(y_i)\right|.\end{aligned} \tag{3.6}$$

We have just shown that with such a mapping f available, we can surely generate samples according to $X = f(Z)$ and also compute the density of p_X.

The remaining questions become:

(Q1) how to find a parameterized invertible mapping?

(Q2) how to design such a mapping such that the determinant of Jacobian is easy to compute?

(Q3) the universality (expressiveness) and efficiency of such a design?

3.3.2 Coupling flow and RealNVP

A widely used idea to design invertible mapping is through the coupling flows. For a n dimensional problem, $x = (x_1, x_2) \in \mathbb{R}^d \times \mathbb{R}^{n-d}$, one defines such a mapping:

$$\begin{cases} y^{(1)} = h\big(x^{(1)}, \theta(x^{(2)})\big); \\ y^{(2)} = x^{(2)}. \end{cases}$$

where $h(\cdot, \theta)$ is an invertible function. The inverse is simply

$$\begin{cases} y^{(1)} = h^{-1}\big(y^{(1)}, \theta(y^{(2)})\big); \\ x^{(2)} = y^{(2)}. \end{cases}$$

Namely, we fix certain coordinates as conditioner, and we only update other coordinates, a bit like coordinate-wise gradient descent idea.

A widely used example is RealNVP [7] (standing for real-valued non-volume preserving):

$$\mathsf{f}(x) = \begin{cases} y^{(1)} = x^{(1)} \odot \exp\big(s(x^{(2)})\big) + t(x^{(2)}); \\ y^{(2)} = x^{(2)}. \end{cases}$$

$s, t : \mathbb{R}^{n-d} \times \mathbb{R}^d$ are parameterized functions, and $\odot$ is the element-wise product. The inverse function is

$$\mathsf{f}(y)^{-1} = \begin{cases} x^{(1)} = \big(y^{(1)} - t(y^{(2)})\big) \odot \exp\big(-s(y^{(2)})\big); \\ x^{(2)} = y^{(2)}. \end{cases}$$

In such a formalism, one does not require s, t to be invertible. The Jacobian

$$J_{\mathsf{f}}(x) = \det\big(\nabla \mathsf{f}(x)\big) = \exp\Big(\sum_j s(x^{(2)})_j\Big)$$

In order for the component $x^{(2)}$ to also evolve, one can change the partition (a collection of coordinates that remain unchanged) above so that all coordinates are overall "coupled".

3.3.3 Loss functions for learning without data

Given a parameterized density $X_\theta \sim p_{X,\theta} = f_\theta \# p_Z$, we want to match this with $X \sim p_X$. Therefore, our optimization problem can be setup as

$$\min_\theta \ \mathrm{Dist}\big(f_\theta \# p_Z, p_X\big).$$

A commonly used "metric" is KL divergence, which leads into

$$\begin{aligned} \min_\theta \ D_{\mathrm{KL}}\big(f_\theta \# p_Z, p_X\big) &= \int p_{X,\theta}(x) \log p_{X,\theta}(x) - p_{X,\theta}(x) \log p_X(x) \ \mathrm{d}x \\ &= \mathbb{E}_{X_\theta \sim f_\theta(Z)} \Big[\log p_{X,\theta}\big(X_\theta\big) - \log p_X\big(X_\theta\big)\Big]. \end{aligned}$$

Suppose we already know that our target distribution has a potential U with $p_X(x) = e^{-U(x)}/\mathcal{Z}$, then the optimization problem becomes

$$\min_\theta \ \mathbb{E}_{X_\theta \sim f_\theta(Z)} \Big[\log p_{X,\theta}\big(X_\theta\big) + U\big(X_\theta\big) \Big]. \tag{3.7}$$

The term involving $\mathcal{Z}$ can be dropped out as it is independent of θ. This formalism can help to learn a target distribution without given data. To discretize the above expectation, we can take samples

$$\min_\theta \frac{1}{M} \sum_{i=1}^{M} \log p_{X,\theta}\big(f_\theta(z_i)\big) + U\big(f_\theta(z_i)\big),$$

where z_i are *i.i.d.* samples drawn from the base distribution p_Z.

3.3.4 Loss functions for learning with data

Another possibility is to use

$$\min_\theta \ D_{\mathrm{KL}}\big(p_X, f_\theta \# p_Z\big) = \int p_X \log\big(p_X\big) - p_X \log\big(p_{X,\theta}\big).$$

As p_X is the true distribution, independent of θ, this is equivalent to estimate

$$\max_\theta \ \mathbb{E}_{X \sim p_X} \Big[\log\big(p_{X,\theta}(X)\big) \Big].$$

Suppose we are given data samples $\big\{x_i\big\}_{i=1}^N$ rather than the potential function, we can use

$$\max_\theta \ \frac{1}{N} \sum_{i=1}^{N} \log \ p_{X,\theta}(x_i) \overset{(3.2)}{=} \frac{1}{N} \sum_{i=1}^{N} \log p_Z\big(f_\theta^{-1}(x_i)\big) + \log \Big| J_{f_\theta^{-1}}(x_i) \Big|.$$

Then everything is computable. This formalism can also be interpreted as maximizing the log-likelihood for data conditioned on the model parameter θ.

3.4 ODE Flows

As mentioned above, the NFs structure can help to more easily compute the density functions. When the layers become infinitely deep, the normalizing flow structure can turn into an ODE system or say a "continuous normalizing flow". Moreover, ODE is naturally time-reversible (meaning invertible flow map).

3.4.1 Evolution of log-density function

More specifically, suppose we consider the ODE:

$$\frac{\mathrm{d}}{\mathrm{d}} Y(t) = F\big(Y(t), t\big), \qquad Y(0) \text{ is random.}$$

Though the initial condition is random, each trajectory is still deterministic. The time-t flow map $y \mapsto \phi(y,t)$ also solves the ODE

$$\frac{\mathrm{d}}{\mathrm{d}t} \phi(y,t) = F\big(\phi(y,t), t\big), \qquad \phi(y,0) = y.$$

The probability density function of $Y(t)$, denoted as $p(y,t)$, solves the following PDE

$$\partial_t p(y,t) \overset{(1.18)}{=} -\nabla_y \cdot \big(F(y,t)p(y,t)\big) = -\big(\nabla_y \cdot F(y,t)\big)p(y,t) - F(y,t) \cdot \nabla_y p(y,t). \tag{3.8}$$

This can be obtained by the formula of Fokker-Planck equation in (1.18) as ODE can be regarded as a special SDE without diffusion coefficient. Then by the chain rules, the log-density function can be computed as follows:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\log p\big(Y(t),t\big) &= \frac{\partial_t p\big(Y(t),t\big) + \nabla_y p\big(Y(t),t\big)\cdot\frac{\mathrm{d}Y(t)}{\mathrm{d}t}}{p\big(Y(t),t\big)} \\ &\overset{(3.8)}{=} \frac{-(\nabla_y\cdot F)\big(Y(t),t\big)p\big(Y(t),t\big) - F\big(Y(t),t\big)\cdot\nabla_y p\big(Y(t),t\big) + \nabla_y p\big(Y(t),t\big)\cdot F\big(Y(t),t\big)}{p\big(Y(t),t\big)} \\ &= -(\nabla_y\cdot F)\big(Y(t),t\big).\end{aligned}$$

Hence,

$$\log p\big(Y(T),T\big) - \log p\big(Y(0),0\big) = \int_0^T -\big(\nabla_y\cdot F\big)\big(Y(t),t\big)\ \mathrm{d}t. \tag{3.9}$$

3.4.2 Connection to (discrete) normalizing flows

Suppose the initial distribution is p_Z, we have

$$p\big(Y(T),T\big) = p_Z\big(Y(0)\big)\exp\Big(-\int_0^T \big(\nabla_y\cdot F\big)\big(Y(t),t\big)\ \mathrm{d}t\Big).$$

If we choose grid points $0 = t_0 < t_1 < \cdots < t_\ell = T$, grid size $\Delta t = \frac{T}{\ell}$ and suppose that $\ell \gg 1$. If we approximately choose $f_i^{-1}(x) = 1 - F(x,t_i)\Delta t + \mathcal{O}\big(\Delta t^2\big)$ and let mapping $f(x) = f_\ell\circ f_{\ell-1}\circ\cdots\circ f_1(x)$ and denote $\hat{y}_k = f_k\circ f_{k-1}\cdots f_1(x)$ for $k = 1,2,\cdots,\ell$. Then

$$\begin{aligned}\det\big(\nabla f_i^{-1}(x)\big) &= \det\Big(I - \nabla F(x,t_i)\Delta t\Big) + \mathcal{O}\big(\Delta t^2\big) \\ &= 1 - \mathrm{tr}\big(\nabla F(x,t_i)\big)\Delta t + \mathcal{O}\big(\Delta t^2\big) \\ &= \exp\Big(-\mathrm{tr}\big(\nabla F(x,t_i)\big)\Delta t\Big) + \mathcal{O}\big(\Delta t^2\big),\end{aligned}$$

and the last equation becomes

$$\begin{aligned}p\big(\hat{y}_\ell,T\big) &= p_Z\big(\hat{y}_0\big)\prod_{i=1}^{\ell}\exp\Big(-\Delta t(\nabla_y\cdot F)(\hat{y}_i,t_i)\Big) + \mathcal{O}(\Delta t) \\ &= p_Z\big(\hat{y}_0\big)\prod_{i=1}^{\ell}\det\big(\nabla f_i^{-1}(\hat{y}_i)\big) + \mathcal{O}(\Delta t).\end{aligned}$$

This reduces to (3.6) for the discrete-time case. In the above, we used a fact that $\det\big(I+\epsilon A\big) = 1 + \mathrm{tr}(A)\epsilon + \mathcal{O}\big(\epsilon^2\big)$ for any square matrix A and small enough ϵ.

3.4.3 Training of ODE flow for learning tasks without data

Suppose we consider the problem of learning probability distribution without data. Then similar to (3.7), we need to optimize:

$$\min_\theta\ \mathbb{E}\Big[\log p\big(Y(T,\theta),T,\theta\big) + U\big(Y(T,\theta)\big)\Big].$$

By (3.9), if we denote

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}t}Y(t,\theta) = F\big(Y(t,\theta),t,\theta\big), \qquad Y(0,\theta) = Y_0; \\ &\frac{\mathrm{d}}{\mathrm{d}t}J(t,\theta) = -\nabla_y\cdot F\big(Y(t,\theta),t,\theta\big), \qquad J(0,\theta) = \log p_Z\big(Y(0)\big),\end{aligned} \tag{3.10}$$

and denote $\overline{Y} = (Y, J) \in \mathbb{R}^{n+1}$, the optimization problem above can be written as

$$\min_{\theta} \ \mathbb{E}\left[V\big(\ \overline{Y}(T,\theta)\ \big)\right], \qquad V(\overline{Y}) := \overline{Y}_{n+1} + U(\overline{Y}_{1:n})$$

Finding an optimal parameter θ for the augmented dynamics (3.10) with terminal loss V is essentially the task for neural ODEs which had been revisited in Chapter 2.3.

3.4.4 Training of ODE flow for learning tasks with data

Similar to the discrete-time case, one can train the ODE flow with log-likelihood as loss (or equivalently the KL divergence as discussed above):

$$\max_{\theta} \ \mathbb{E}_{X\sim p_X}\ \big[\log p(X,T,\theta)\big] = \frac{1}{N}\sum_{i=1}^{N} \log p(x_i,T,\theta).$$

Using (3.9), we could simply solve the following system of ODEs:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}Y(t,\theta) = F\big(Y(t,\theta),t,\theta\big), \qquad Y(T,\theta) = x_i;\\
&\frac{\mathrm{d}}{\mathrm{d}t}J(t,\theta) = -\big(\nabla_y \cdot F\big)\big(Y(t,\theta),t\big), \qquad J(0,\theta) = \log p_Z\big(Y(0,\theta)\big).
\end{aligned}$$

This can also be formulated as a standard neural ODE problem with a little effort.

3.5 Further Readings

Below we list a few related topics for reading and possible exploration:

- Boltzmann Generator [22]: use normalizing flows to sample from the Boltzmann distributions of many-body particles.
- The Benamou-Brenier formalism [3] (or say a continuous-time dynamical perspective) in optimal transport, which connects to the ODE flow formalism.
- We only cover a few narrow aspects of normalizing flows. More can be found in review papers [14, 23] and references therein, as well as [4, Chapter 18], e.g., we haven't covered auto-regressive flow above.
- We haven't discussed the estimation of the complexity of computing Jacobian, which is left as exercises; see e.g., [4, Chapter 18].

Chapter 4

Langevin Sampling

The open system studies the interaction of the system with its environment. As the environment could have a huge amount of degrees of freedom (e.g., a large amount of water molecules), we won't want to, and we are unable to directly study all degrees of freedom of the environment. Therefore, this challenge leads into a topic called *open system*. For instance, in the example of the discovery of Brownian motion, the pollen particle is the system that we want to study, and all water molecules form an environment. The study of pollen particle without modeling all water molecules is a typical example for the study of open classical systems.

Among many models, *Langevin dynamics* is an important dynamical equation to characterize the interaction of system and environment, with applications in various research fields, in particular with recent applications in machine learning. Later we will show why compared with normalizing flows and ODE flows above (cf. Chp. 3.3.3), Langevin dynamics can be used as a sampling algorithm without any training. The study of Langevin dynamics and Langevin sampling is a very large topic and we will only very concisely touch the very basic concepts only.

4.1 Generalized Langevin Equation and Underdamped Langevin Dynamics

For those who are interested in its origin in physics, one may refer to classical papers [20, 34] in last 60-70s or [24, Chapter 8] for the introduction of deriving Langevin dynamics in open classical systems. The *Generalized Langevin equation (GLE)* takes the following form [34, Eq. (22)]

$$\begin{aligned}\dot{Q}(t) &= P(t)\\ \dot{P}(t) &= -\nabla U\big(Q(t)\big) - \int_0^t \mathrm{d}t'\ \zeta(t')P(t-t')\ + \mathcal{F}(t)\end{aligned}$$

where U is the potential function, ζ is the correlation function, the noise $\mathcal{F}(t)$ has mean zero and correlation $\langle \mathcal{F}(t)\mathcal{F}(t')\rangle = k_B\mathsf{T}\zeta(t-t')$, where k_B is the Boltzmann constant, T is the temperature.

Suppose that the correlation function decays sufficiently fast, e.g., $\zeta(t) \approx 2\gamma\frac{e^{-t^2/(2\delta^2)}}{\sqrt{2\pi\delta^2}}$ ($\delta \ll 1$) is approximately a delta measure, then we may model the above dynamics "equivalently" using the following SDE, also known as the *underdamped Langevin dynamics*:

$$\begin{aligned}\mathrm{d}Q\,(t) &= P(t)\ \mathrm{d}t\\ \mathrm{d}P\,(t) &= -\nabla U\big(Q(t)\big)\mathrm{d}t - \gamma P(t)\ \mathrm{d}t + \sqrt{2\gamma\beta^{-1}}\ \mathrm{d}W(t),\end{aligned} \tag{4.1}$$

where $\gamma > 0$ represents the damping coefficient, and $\beta = \frac{1}{k_B\mathsf{T}}$ means the inverse temperature. When the temperature T is large, $\beta^{-1} \gg 1$, and the fluctuation due to Brownian motion is more significant; when the temperature T is very low, $\beta^{-1} \ll 1$, and the influence of random noise due to environment is small. Hence, at least, the role of β is indeed compatible with thermodynamics.

For underdamped Langevin dynamics (4.1), the convergence analysis of the relevant Fokker-Planck equation, the limiting behavior, as well as the performances of various numerical discretization schemes have been active research topics in the last decade and there are still open questions along this line.

4.2 Over-damped Langevin Sampling

For simplicity, let us consider *over-damped Langevin dynamics* instead: [1]

$$\mathrm{d}X(t) = -\nabla U\big(X(t)\big)\ \mathrm{d}t + \sqrt{2\beta^{-1}}\ \mathrm{d}W(t). \tag{4.2}$$

The derivation of this SDE from underdamped Langevin dynamics (4.1) in the limit of $\gamma \to \infty$ could be found in e.g., [24, Chapter 6.5]. This is perhaps also where the name "overdamped" comes from.

The Fokker-Planck equation of this SDE is the following:

$$\partial_t p(x,t) \overset{(1.18)}{=} \nabla \cdot \big(\nabla U(x) p(x,t)\big) + \beta^{-1}\Delta p(x,t). \tag{4.3}$$

It can be easily checked that

$$p_\infty(x) \propto e^{-\beta U(x)},$$

is a stationary solution to the above PDE.

Remark. Of course, to reach such an equilibrium, one may use time-dependent potential $U(\cdot,t)$. Designing a dynamics (namely, designing drift and diffusion terms) so that it reaches a certain equilibrium faster is a topic of interest for research.

Due to such a convergence behavior, one may design a numerical scheme to simulate the above SDE to time T where $T \gg 1$, so that $p(\cdot,T) \approx p_\infty$. If so, then $X(T)$ are samples approximately from p_∞. To simulate the SDE, one can use a discrete time-step $\Delta t = \frac{T}{N}$ where $\Delta t \ll 1$, and discretize the SDE using Euler-Maruyama method:

$$\hat{X}_{k+1} = \hat{X}_k - \nabla U(\hat{X}_k)\ \Delta t + \sqrt{2\beta^{-1}\Delta t}\ \mathcal{N}(0,1), \tag{4.4}$$

where $\mathcal{N}(0,1)$ are standard random variables, e.g., generated by Box–Muller transform discussed in Chapter 3.2. Fortunately, most numerical packages have already provided reliable interface to a generator of normal random variables, like numpy in Python.

In practice, we don't have to re-draw samples every time. If one wants to estimate the expected value in the following form

$$\int_{\mathbb{R}^n} f(x)\ p_\infty(x)\ \mathrm{d}x = \lim_{T\to\infty} \frac{1}{T}\int_0^T f\big(X(t)\big)\ \mathrm{d}t$$

where the equality holds due to ergodicity. One may further approximate the integral via

$$\lim_{T\to\infty} \frac{1}{T}\int_0^T f\big(X(t)\big)\ \mathrm{d}t \approx \frac{1}{T}\int_0^T f\big(X(t)\big)\ \mathrm{d}t \approx \frac{1}{N}\sum_{i=1}^N f(\hat{X}_k). \tag{4.5}$$

where the time step is $\Delta t = \frac{T}{N}$ is supposed to be small.

Langevin dynamics, though originated from statistical physics, can be used for other statistical problems, and machine learning problems. A common task is that one wants to sample from a certain distribution q, one may define $U(x) := -\beta^{-1}\ \log q(x)$ and thus one can utilize such a dynamics above (4.2) to sample from q (provided that one is given an expression of q up to some multiplicative prefactor).

[1] the notation of the position Q has been changed to X below

4.3 Further Readings

- [24]: Chapter 6 and Chapter 8 provide a detailed mathematical description of Langevin dynamics and its Fokker-Planck equations.
- A more comprehensive introduction of this sampling topic can be found in [6].
- Algorithmic problem: Due to the hardness for Langevin dynamics to converge when a tall energy barrier exists under low temperature $\beta \gg 1$, it is still an active research topic to explore more efficient algorithms.

Chapter 5

Neural Dynamics for Generative Problems: (II) Diffusion Models

We will revisit the following problem:

Problem: given available data $\{x_i\}_{i=1}^N$ of an unknown p_X, how can we train a dynamical system to learn or generate samples from the distribution p_X, utilizing a base distribution (or say prior distribution) p_Z?

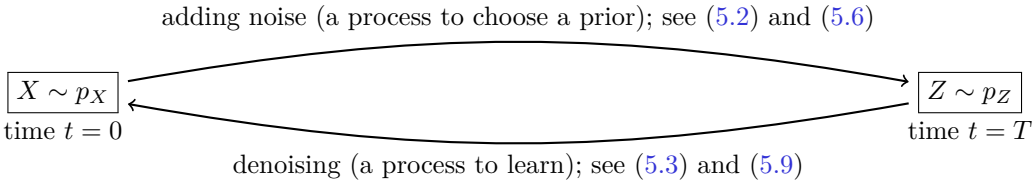


Figure 5.1: A visualization of the forward and backward directions in diffusion models

This is a problem that we have discussed in normalizing flows in Chapter 3. For normalizing flows, when one designs a structure and the loss function, only information for the prior distribution p_Z and the target distribution p_X are mainly counted. There are certain redundancy for the trajectory in the middle stages. For instance, for each trajectory in the probability space $p(t, \cdot)$, one can find a velocity field $v(x, t)$ such that

$$\partial_t p(x,t) = -\nabla_x \cdot \big(v(x,t)p(x,t)\big) \tag{5.1}$$

This can be realized e.g., via an ODE

$$\frac{\mathrm{d}}{\mathrm{d}t}X(t) = v\big(X(t), t\big).$$

As there are infinitely many possible trajectories, this degree of freedom may also come with the difficulty of training. The direct continuous normalizing flows (or say ODE flows) using the control perspective tastes a bit like a shooting method for ODEs.

One may impose additional criterions like

- minimizing the action of path (similar to Benamou-Brenier formalism in optimal transport [3]);
- to specify a certain "Fixed" trajectory in the space of probability distributions $\mathcal{P}_2(\mathbb{R}^n)$, which is perhaps a reminiscent of the annealing idea (e.g., [21]), a very powerful idea for many computational tasks beyond what has been described here.

Diffusion-based models and other variants mostly take the second approach (if not all). The dynamical system can of course be (a) deterministic mapping; (b) Markov Chains; (c) ODE flows; (d) SDEs. To give some background, in the earlier stage of developing diffusion models, the process of generating samples from p_X utilized discrete-time Markov process [28, 30, 10]. Later, an SDE-based formulation was available in [31] which unifies previous Markov process based models. Not too much time later, diffusion-like models based on ODEs become a popular alternatives, and frameworks like flow-matching [18] and rectified flow [19] are introduced; these frameworks can be further unified within stochastic interpolants [1]. More recently, for the consideration of faster generation, the advantage of discrete mapping becomes more appreciated and e.g., consistency model [29] was introduced partially under this background. So far, the diffusion-based models and many variants have become a powerful tool in data synthesis, e.g., text-to-image generation and et al.

In the following notes, for simplicity, we will not follow historical development, and one may check the review papers for an overarching understanding of this field and many other important works. Due to the time limitation in this mini-course, we will only cover the most basic mathematical objects behind these models and only focus on continuous-time dynamics.

5.1 SDE-based Diffusion Models

Step 1: specify the path

In the following, we will adopt the continuous-time formalism based on [31] for easier illustration. In this case, we consider the following OU process

$$\mathrm{d}X(t) = b(X(t))\ \mathrm{d}t + \sqrt{2}\ \mathrm{d}W_t, \qquad b(x) = -x. \tag{5.2}$$

This is a very special Langevin dynamics with potential $U(x) = |x|^2/2$. Therefore, the limiting distribution is $p_\infty = \mathcal{N}(0, \mathbf{I}_n)$ (see Exercise 1.18). With a time T large enough, we know that $p(\cdot, T) \approx N(0, \mathbf{I}_n) := p_Z$. For the time being, let us worry not the gap between p_Z and $p(\cdot, T)$. What this process really did is to define a path in the space of probability distributions. The associated Fokker-Planck equation is

$$\partial_t p(x,t) = -\nabla \cdot \big(b(x)p(x,t)\big) + \Delta p(x,t),$$

and if we use $\widetilde{p}(x,t) := p(x, T-t)$ for the time-reversed process, then clearly

$$\partial_t \widetilde{p}(x,t) = \nabla \cdot (b(x)\widetilde{p}(x,t)) - \Delta \widetilde{p}(x,t).$$

Step 2: specify the reverse-time SDE

In order to reverse the process for data generation (driving a random variable following p_Z to a random variable following p_X), it is natural to image that one also uses an SDE

$$\mathrm{d}Y(t) = A(Y(t),t)\ \mathrm{d}t + \sqrt{2h}\ \mathrm{d}W_t, \qquad Y(0) \sim p(\cdot, T) \approx p_Z. \tag{5.3}$$

Since this SDE also need to follow the same path (in the sense of associated probability distributions), we have

$$\nabla \cdot (b(x)\widetilde{p}(x,t)) - \Delta \widetilde{p}(x,t) = \partial_t \widetilde{p}(x,t) = -\nabla \cdot \big(A(x,t)\widetilde{p}(x,t)\big) + h\ \Delta \widetilde{p}(x,t).$$

By matching terms, we need

$$b(x)\widetilde{p}(x,t) - \nabla \widetilde{p}(x,t) = -A(x,t)\widetilde{p}(x,t) + h\nabla \widetilde{p}(x,t)$$

which leads into

$$A(x,t) = -b(x) + (h+1)\frac{\nabla \widetilde{p}(x,t)}{\widetilde{p}(x,t)} = -b(x) + (h+1)\nabla \log \widetilde{p}(x,t).$$

Step 3: Training of the score function

Therefore, the unknown part is $\nabla \log p(x,t)$, which is also known as the *score function*. A natural idea is to parameterize $s_\theta(x,t) \approx \nabla \log p(x,t)$ and use the mean-square error

$$\begin{aligned}
\min_\theta \; L_{\mathrm{SDE}}(\theta) &:= \int_0^T \int_{\mathbb{R}^n} \left\| \nabla \log p(x,t) - s_\theta(x,t) \right\|^2 p(x,t) \, \mathrm{d}t \\
&= \int_0^T \int_{\mathbb{R}^n} \left\| s_\theta(x,t) \right\|^2 p(x,t) - 2 s_\theta(x,t) \cdot \nabla_x \log p(x,t) \; p(x,t) \, \mathrm{d}x \, \mathrm{d}t + C \\
&= \int_0^T \int_{\mathbb{R}^n} \left\| s_\theta(x,t) \right\|^2 p(x,t) - 2 s_\theta(x,t) \cdot \nabla_x p(x,t) \, \mathrm{d}x \, \mathrm{d}t + C \\
&= \int_0^T \mathrm{d}t \int_{\mathbb{R}^n} \mathrm{d}x \int_{\mathbb{R}^n} \mathrm{d}y \; \left\| s_\theta(x,t) \right\|^2 P(x,t|y,0) p(y,0) - 2 s_\theta(x,t) \cdot \nabla_x P(x,t|y,0) p(y,0) + C \\
&= \int_0^T \mathrm{d}t \; \mathbb{E}_{X_0 \sim p(\cdot,0)} \mathbb{E}_{X_t \sim P(\cdot,t|X_0,0)} \Big[\left\| s_\theta(X_t,t) \right\|^2 - 2 s_\theta(X_t,t) \cdot \nabla_x \log P(X_t, t \mid X_0, 0) \Big] + C \\
&= \int_0^T \mathrm{d}t \; \mathbb{E}_{X_0 \sim p(\cdot,0)} \mathbb{E}_{X_t \sim P(\cdot,t|X_0,0)} \Big[\left\| s_\theta(X_t,t) - \nabla_x \log P(X_t, t \mid X_0, 0) \right\|^2 \Big] + C
\end{aligned}$$

where "$+C$" means some terms independent of the parameter θ. Therefore, we only need to solve

$$\min_\theta \int_0^T \mathrm{d}t \; \mathbb{E}_{X_0 \sim p(\cdot,0)} \mathbb{E}_{X_t \sim P(\cdot,t|X_0,0)} \Big[\left\| s_\theta(X_t,t) - \nabla_x \log P(X_t, t \mid X_0, 0) \right\|^2 \Big], \tag{5.4}$$

where the transition probability for OU process is easy to obtain: for a fixed y, the conditional probability $P(x,t|y,0)$ follows the normal random variable with mean $e^{-t}y$ and covariance matrix $c(t)\mathbf{I}_n$ and the scalar-valued function

$$c(t) = 2 \int_0^t e^{-2(t-s)} \, \mathrm{d}s.$$

see Example 1.13; hence,

$$P(x,t|y,0) = \frac{1}{\sqrt{2\pi c(t)}^n} \; \exp\big(- \frac{\left\| x - e^{-t} y \right\|^2}{2c(t)} \big) \implies \nabla_x \log P(x,t|y,0) = - \frac{x - e^{-t} y}{c(t)},$$

which is easily computable.

Summary

One can train a parameterized function $(x,t) \in \mathbb{R}^n \times \mathbb{R} \to s_\theta(x,t) \in \mathbb{R}^n$ using the loss function in (5.4) and then use the SDE (5.3) to generate samples, also copied below

$$\begin{aligned}
\mathrm{d}Y(t) &= A_\theta(Y(t),t) \, \mathrm{d}t + \sqrt{2h} \, \mathrm{d}W_t, \qquad Y(0) \sim p(\cdot, T) \approx p_Z \\
A_\theta(x,t) &= -b(x) + (h+1) s_\theta(x, T-t).
\end{aligned} \tag{5.5}$$

When we pick $h = 0$ in (5.5), then the backward dynamics is simply an ODE.

5.2 ODE-based Models

Specify the loss and the ODE

Suppose that the probability distribution at time $t = T$ is (or very close to) the prior distribution p_Z (e.g., chosen as $p_Z = \mathcal{N}(0, \mathbf{I}_n)$ for simplicity), and at time $t = 0$, it is the target distribution p_X. As discussed earlier, there are infinitely many paths in the probability space to connect them, let us say a smooth path

$\{p(\cdot,t)\}_{t=0}^T$ in the functional space $\mathcal{P}_2(\mathbb{R}^n)$, then there exists a velocity field v satisfying (5.1). One only needs to parameterize the velocity field using $(x,t)\mapsto v_\theta(x,t)$ so that

$$\begin{aligned}\min_\theta\ L_{\text{ODE}}(\theta) &:= \int_0^T \int_{\mathbb{R}^n} \|v(x,t)-v_\theta(x,t)\|^2\, p(x,t)\,\mathrm{d}t \\ &= \int_0^T \mathbb{E}\left[\big\|v\big(X(t),t\big)-v_\theta\big(X(t),t\big)\big\|^2\right]\end{aligned}$$

where $X(t)\sim p(\cdot,t)$. Therefore, the next task becomes to find some examples of such $X(t)$ and also $v\big(X(t),t\big)$ is easily computable.

Specify the path

A widely used example is the following, studied in e.g., [18, 19, 1]:

$$X(t)=\alpha(t)X+\beta(t)Z \tag{5.6}$$

where $X\sim p_X$ and $Z\sim p_Z$, and $\alpha(0)=1,\alpha(T)=0,\ \beta(0)=0,\beta(T)=1$ are used to satisfy terminal conditions. It has been shown in [1] that:

Lemma 5.1 ([1, Theorem 2.6])**.** For (5.6), let $p(\cdot,t)$ be the associated probability distributions and thus specifies the velocity field $v(\cdot,t)$ to satisfy (5.1). Then

$$v(x,t)=\mathbb{E}\big[\dot X(t)|X(t)=x\big]\equiv\mathbb{E}\big[\dot\alpha(t)X+\dot\beta(t)Z\mid X(t)=x\big]. \tag{5.7}$$

Proof. For any test function $\phi\in C_c^\infty(\mathbb{R}^n)$,

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}[\phi\big(X(t)\big)]=\mathbb{E}\Big[\nabla\phi\big(X(t)\big)\cdot\big(\dot\alpha(t)X+\dot\beta(t)Z\big)\Big]$$

and in the weak form

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}[\phi\big(X(t)\big)] &= \frac{\mathrm{d}}{\mathrm{d}t}\int\phi(x)p(x,t)\,\mathrm{d}x=\int\phi(x)\partial_t p(x,t)\,\mathrm{d}x \\ &= -\int\phi(x)\nabla\cdot\big(v(x,t)p(x,t)\big)\,\mathrm{d}x=\int\nabla\phi(x)v(x,t)p(x,t)\,\mathrm{d}x \\ &= \mathbb{E}\big[\nabla\phi(X(t))v(X(t),t)\big]\end{aligned}$$

Since the test function ϕ can be arbitrary, this proves (5.7). □

Therefore, one can write the loss function as

$$\begin{aligned}L_{\text{ODE}}(\theta) &= \int_0^T \mathbb{E}\left[\big\|v_\theta\big(X(t),t\big)\big\|^2-2\Big\langle v\big(X(t),t\big),v_\theta\big(X(t),t\big)\Big\rangle\right]+C \\ &= \int_0^T \mathbb{E}\left[\big\|v_\theta\big(X(t),t\big)\big\|^2\right]-2\,\mathbb{E}\left[\Big\langle v\big(X(t),t\big),v_\theta\big(X(t),t\big)\Big\rangle\right]+C \\ &= \int_0^T \mathbb{E}\left[\big\|v_\theta\big(X(t),t\big)\big\|^2\right]-2\,\mathbb{E}\left[\mathbb{E}\left[\Big\langle v\big(X(t),t\big),v_\theta\big(X(t),t\big)\Big\rangle\mid X(t)\right]\right]+C \\ &= \int_0^T \mathbb{E}\left[\big\|v_\theta\big(X(t),t\big)\big\|^2\right]-2\,\mathbb{E}\left[\mathbb{E}\left[\Big\langle \dot\alpha(t)X+\dot\beta(t)Z,v_\theta\big(X(t),t\big)\Big\rangle\mid X(t)\right]\right]+C \\ &= \int_0^T \mathbb{E}\left[\big\|v_\theta\big(X(t),t\big)\big\|^2\right]-2\,\mathbb{E}\left[\Big\langle \dot\alpha(t)X+\dot\beta(t)Z,v_\theta\big(X(t),t\big)\Big\rangle\right]+C \\ &= \int_0^T \mathbb{E}\left[\Big\|v_\theta\big(X(t),t\big)-\big(\dot\alpha(t)X+\dot\beta(t)Z\big)\Big\|^2\right]+C.\end{aligned}$$

Summary

Therefore, one can solve the following optimization problem

$$\min_{\theta} \int_0^T \mathbb{E}_{X\sim p_X, Z\sim p_Z} \left[\left\| v_\theta\big(X(t),t\big) - \big(\dot\alpha(t)X + \dot\beta(t)Z\big) \right\|^2 \right], \tag{5.8}$$

and then simulate the following ODE systems (with time-already reversed):

$$\frac{\mathrm{d}}{\mathrm{d}t} Y(t) = -v_\theta\big(Y(t), T-t\big) \qquad Y(0)\sim p_Z, \tag{5.9}$$

for the data generation using the output $Y(T)$.

5.3 Further Readings

- [4, Chapter 20]: a textbook style introduction to this topic.
- Classifier-free Diffusion Guidance [11]: generating samples conditioned on certain label (or say class) information, which leads into the application like text-to-image generation.
- Stochastic Interpolants [1]: a very unified framework of many mathematical models.
- Consistency Models [29]: provide a one-shot generation (aka fast) for a data generation trajectory.

Bibliography

[1] Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic Interpolants: A Unifying Framework for Flows and Diffusions, 2023. arXiv:2303.08797 [cond-mat].

[2] Luigi Ambrosio and Giuseppe Savaré. Chapter 1 - Gradient Flows of Probability Measures. In C. M. Dafermos and E. Feireisl, editors, *Handbook of Differential Equations: Evolutionary Equations*, volume 3, pages 1–136. 2007.

[3] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.

[4] Christopher M. Bishop and Hugh Bishop. *Deep Learning: Foundations and Concepts*. Springer International Publishing, 2024.

[5] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 6572–6583. Curran Associates Inc., 2018.

[6] Sinho Chewi. *Log-Concave Sampling*. 2024. https://chewisinho.github.io/main.pdf.

[7] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations*, 2017.

[8] Weinan E. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[9] Weinan E and Bing Yu. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.

[11] Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[12] Fima C Klebaner. *Introduction to stochastic calculus with applications*. World Scientific Publishing Company, 2012.

[13] Peter E. Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, Berlin, Heidelberg, 1992.

[14] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021.

[15] Qianxiao Li. Dynamical Systems and Machine Learning. 2020. URL: https://www.math.pku.edu.cn/amel/docs/20200719122925684287.pdf.

[16] Ruichen Li, Haotian Ye, Du Jiang, Xuelan Wen, Chuwei Wang, Zhe Li, Xiang Li, Di He, Ji Chen, Weiluo Ren, and Liwei Wang. A computational framework for neural network-based variational Monte Carlo with Forward Laplacian. *Nature Machine Intelligence*, 6(2):209–219, 2024.

[17] Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable Gradients for Stochastic Differential Equations. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 3870–3882. PMLR, 2020. arXiv:2001.01328.

[18] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

[19] Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023.

[20] Hazime Mori. Transport, Collective Motion, and Brownian Motion. *Progress of Theoretical Physics*, 33(3):423–455, March 1965.

[21] Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

[22] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

[23] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

[24] Grigorios A Pavliotis. Stochastic processes and applications. *Texts in applied mathematics*, 60, 2014.

[25] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[26] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. arXiv:1609.04747.

[27] Mark Schmidt. CPSC 540: Machine Learning: SGD Convergence Rate, 2019. https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/L11.pdf.

[28] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, pages 2256–2265, 2015.

[29] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency Models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 32211–32252, 2023.

[30] Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[31] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*, 2021.

[32] Unknown. Einstein's random walk. *Physics World*, 2005. URL: https://physicsworld.com/a/einsteins-random-walk/.

[33] Cédric Villani. *Optimal Transport: Old and New*. Springer-Verlag, Berlin Heidelberg, 2009.

[34] Robert Zwanzig. Nonlinear generalized langevin equations. *Journal of Statistical Physics*, 9(3):215–220, 1973.

Appendix A

Erratum (which has been corrected)

- A missing argument (r) inside the Theorem (1.2).
- The function f in the original version really means the loss function L inside Section 1.4.3 SGD.
- Two typos inside Section 2.1 for the output state.
- The normalizing constant has been changed to $\mathcal{Z}$ to avoid notation conflicts.