UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

Path Integral Optimiser: Global Optimisation via Neural Schrödinger-Föllmer Diffusion

Max McGuinness

Queens' College

June 2023

Submitted in partial fulfillment of the requirements for the
Computer Science Tripos, Part III

Total page count: 48

Main chapters (excluding front-matter, references and appendix): 40 pages (pp 1–40)

Main chapters word count: 11912

Methodology used to generate that word count:

```
pdftotext -f 8 -l 48 'main.pdf' - | egrep '\w{4}' | wc -w
```

Declaration

I, Max McGuinness of Queens' College, being a candidate for the Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: Max McGuinness

Date: October 10, 2024

Abstract

Most tasks in machine learning can be characterised as *high-dimensional non-convex stochastic optimisation problems*, in which an optimiser seeks the parameters that minimize some loss incurred by a particular model over a dataset. Stochastic gradient descent (SGD) [1] is the archetypical approach to such optimisation in differentiable settings, but it neglects to consider second-order gradient information. Many extensions have been proposed to incorporate such information—such as Adam [2], Adagrad [3], and momentum [4, 5].

While these approaches are effective in the early stages of training, in practice they struggle to generalise as broadly as SGD [6], and their theoretical convergence properties remain poorly understood for non-convex global minimisation [7, 8]. These limitations inspire an ongoing research mission to find a global optimiser that is general, theoretically motivated, and makes complete use of gradient information.

One promising nascent area of machine learning is neural-approximated diffusion. Rooted in almost century-old stochastic theory, diffusion models can solve machine learning problems by transforming samples from a simple distribution towards a target distribution over time. Neural approximations of diffusion processes, such as OpenAI's DALLE·2 [9] have recently proven highly effective at sampling from high-dimensional structured distributions [10, 11].

This paper thus proposes the first use of diffusion for machine learning optimisation, in the form of the **Path Integral Optimiser**: a neural Schrödinger-Föllmer diffusion process trained on-the-fly to generate the optimal parameters for a target network. I derive novel theoretical guarantees for this new optimiser, present an empirical study comparing it to state-of-the-art optimisers at classic machine learning tasks (demonstrating superior performance in some cases), and recommend future avenues of improvement in diffusion-based optimisation.

In addition, a key contribution of this paper is a complete description of the stochastic theory behind Schrödinger-Föllmer processes, which makes what is otherwise an intractable but promising research area accessible to computer science undergraduates.

Acknowledgements

I firstly thank my supervisor, Francisco Vargas Palomo, for his continued support and incredible responsiveness this year; cheers to Eirik Fladmark for the stimulating conversation; and kudos to Charles Li for a few last-minute math corrections.

On a different note, I would like to take a moment to acknowledge the Queens' Computer Science community for playing such a huge role in my life at Cambridge over the last 4 years. I am immensely grateful to everyone at Queens', including my DoSes Andrew Rice, Neil Lawrence, Ramsey Faragher, and particularly Alastair Beresford for putting me here in the first place. I have absolutely loved it.

Lastly, thank you to my friends, housemates (not mutually exclusive), and family for your vital love and support. I hope you will enjoy reading 40 pages of dense stochastic theory.

Contents

Chapter 1

Introduction

This project investigates, for what I believe to be the first time, the application of diffusion models to the challenging task of *machine-learning optimisation* (**D5.1.2**). This is approached by adapting recent work on the Schrödinger-Föllmer diffusion process (**D3.1.1**), particularly Zhang et al.'s *Path Integral Sampler* [10] (**D3.2.2**), to construct a novel optimiser which I call the **Path Integral Optimiser**.

1.1 Motivation

Most tasks in machine learning can be characterised as **high-dimensional non-convex stochastic optimisation problems**, in which an optimiser seeks the optimal parameters θ_* that minimise some loss V incurred by a particular model $\mathscr{F}$ over a dataset $\mathcal{D}$, solving $\theta_* \in \arg\min_{\theta\in\mathbb{R}^p} \sum_{x\in\mathcal{D}} V\left(\mathscr{F}_\theta(x)\right)$. Stochastic gradient descent (SGD) [1] is the archetypical approach to such optimisation in differentiable settings, but it neglects to consider second-order gradient information. Many extensions have been proposed to incorporate such information—such as Adam [2], Adagrad [3], and momentum [4, 5].

However, while these approaches are effective in the early stages of training, in practice they struggle to generalise as broadly as SGD [6], and their theoretical convergence properties remain poorly understood for non-convex global minimisation [7, 8]. These limitations inspire an ongoing research mission to find a global optimiser that is general, theoretically motivated, and makes complete use of gradient information.

One promising nascent area of machine learning is **neural-approximated diffusion**. Rooted in almost century-old stochastic theory, these models can solve machine learning problems by transforming samples from a simple distribution towards a target over time. Neural approximations of diffusion processes, such as DALLE·2 [9], have recently proven highly effective at sampling from high-dimensional structured distributions [10, 11].

I believe that recent work in the field of diffusion may enable an effective global optimiser. The basic idea is that training a diffusion model to learn a non-convex parameter space

(i.e. `SGD` $\rightarrow$ `Diffusion` $\rightarrow$ `Parameters`) may be easier than learning that parameter space outright (i.e. `SGD` $\rightarrow$ `Parameters`). Ha et al. [12] refer to this form of optimiser as a *HyperNetwork*, and found that a recurrent neural network (RNN) HyperNetwork was able to achieve near-state-of-the-art results at LSTM sequence modelling tasks in 2016.

Specifically, this project focuses on utilising the **Schrödinger-Föllmer** diffusion process (SFP) as a HyperNetwork. SFP drives samples from a point $X_0 = 0$ towards a target distribution $X_1 \sim \mathbb{P}$ in finite time by a stochastic process of the form $dX_T = b(X_t, t)dt + dW_t$, where W is a Wiener process and b is a drift term with an analytic form (**D3.1.1**). Computing the process revolves around identifying an approximation of the drift term b, which one can achieve with a sufficiently parameterised neural network [13, 11, 10].

In 2021, Huang et al. derived compelling bounds for sampling with an SFP in finite time [14]; that same year, Dai et al. found that sampling from a Boltzmann distribution with an SFP allows efficient global optimisation for low-dimensional problems [15]; and in 2022, Zhang et al. found that a neural-approximated SFP, which they call a **Path Integral Sampler**, can outperform state-of-the-art sampling methods, even at high dimensions [10]. Together, these recent results imply that one may be able to achieve efficient global optimisation with a neural-approximated SFP.

Notable alternatives include *Langevin dynamics* of the form $dX_T = -\nabla\frac{1}{2}\nabla \log f(X_t, t)dt + dW_t$ for target density f, which tend to require a far longer mixing time than SFP [15]; and *denoising diffusion* with a time-reversal of the form $dY_t = -\beta_{T-t}(Y_t + 2\sigma^2\nabla \log p_{T-t}(Y_t))dt + \sigma\sqrt{2\beta_{T-t}}dW_t$ from $Y_0 \sim \mathcal{N}_{\text{approx}}(0, \sigma^2)$, which may be more numerically stable than SFP but incurs an additional approximation error for starting from an approximate gaussian rather than a point [16].

1.2 Contributions

This paper thus proposes the first use of diffusion for machine learning optimisation, in the form of the **Path Integral Optimiser (PIO)**: a neural Schrödinger-Föllmer diffusion process trained on-the-fly to generate the optimal parameters for a target network.

My key contributions include:

- **Design and implementation** of PIO (Chapter 4).
- **Novel theoretical guarantees** for PIO (Section 4.2).
- **An empirical study** comparing PIO to popular optimisers at classic machine learning tasks, in some cases matching or exceeding their performance (Chapter 5).
- **A from-first-principles description** of the theory behind Schrödinger-Föllmer processes, a domain otherwise inaccessible to many computer scientists (Chapter 2).
- Recommendations for future research in diffusion-driven optimisation (Chapter 6).

Chapter 2

Background

2.1 Measure Theory

In this section, I will briefly recap probability theory formalisms, in particular the notion of "*measures*", a generalisation of volume necessary to reason about higher-dimension processes probabilistically.

Definition D2.1.1. *A measure is simply a function* $\mu : \mathcal{F} \to \mathbb{R}$ *for which:*

- μ ***is non-negative****:* $\forall A \in \mathcal{F} : \mu(A) \geq 0$.
- μ ***is 0 for the empty set****:* $\mu(\emptyset) = 0$.
- μ ***has countable additivity****:* $((\bigcup_{k=1}^{\infty} A_k) \in \mathcal{F}) \ \wedge \ (i \neq j \implies A_i \cap A_j = \emptyset) \implies \mu\left(\bigcup_{k=1}^{\infty} A_k\right) = \sum_{k=1}^{\infty} \mu\left(A_k\right)$.

2.1.1 Measure Spaces

Measures are defined within **measure spaces** to establish their domain.

Definition D2.1.2. *A* ***measure space*** *is a triple* $(\Omega, \mathcal{F}, \mu)$*, in which* $(\Omega, \mathcal{F})$ *is a measurable space and* $\mu : \mathcal{F} \to \mathbb{R}$ *is a measure.*

Definition D2.1.3. *A measurable space is a pair* $(\Omega, \mathcal{F})$*, such that:*

- *"****Sample space****"* Ω *is a set.*
- *"****Event space****"* $\mathcal{F}$ *is a* σ*-algebra on* Ω.

Definition D2.1.4. *A* σ*-algebra on a set* Ω *is a collection* $\mathcal{F}$ *of subsets of* Ω *satisfying the following properties:*

- $\Omega \in \mathcal{F}$.
- *If* $A \in \mathcal{F}$*, then its complement* $A^c = \Omega \setminus A \in \mathcal{F}$.

- *If $A_i \in \mathcal{F}$ for each $i \in \mathbb{N}$, then the countable union $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.*

The σ-algebra for a measure space is typically constructed as a **Borel algebra** $\mathcal{B}(\Omega)$, which is the smallest σ-algebra to contain all open sets in the topology of Ω. For the purposes of this paper, it is sufficient to know that $\mathcal{B}(\mathbb{R})$ equates to the smallest σ-algebra containing all intervals in $\mathbb{R}$.

Borel algebras are chosen due to satisfying the properties of containing outcome set Ω, being closed under complements, and being closed under infinite unions of elements, while providing a natural framework for defining measurable functions and measurable sets.

2.1.2 Principal Measures

Perhaps the most familiar measure is the **probability measure**, which is simply a measure normalised to sum to 1. When a measure space involves the probability measure it is known as a *probability space* $(\Omega, \mathcal{F}, \mathbb{P})$. I also refer to probability measures as "distributions".

However, the standard form of measure is the **Lebesgue measure**, a generalisation of volume to higher dimensions. Informally, the Lebesgue measure can be thought of as answering "*what proportion of our (real) domain is in this (real) set?*".

I summarise these types and their purposes in table 2.1.2.

Measure	**Definition**	**Purpose**
Probability Measure $\mathbb{P}$	Measure confined by $\mu(\Omega) = 1$.	Calculating probability of sets in Ω.
Lebesgue Measure λ	Measure defined by $\Omega := \mathbb{R}$, and $\mu(A) := \inf\{\sum_{k=1}^{\infty}(\ell(I_k)) : A \subset \bigcup_{k=1}^{\infty}(I_k)\}$, where $A \subseteq \Omega$ and $\ell(I) := \max(I) - \min(I)$. *Intuition: $\mu(A)$ is min sum of lengths that fills A.*	Extending "volume" to higher dimensions by calculating "volume" of intervals in $\mathbb{R}$.

2.1.3 The Lebesgue Integral

The standard method of integration is the familiar **Riemann integral**, which calculates the integral of a function by dividing the domain into small intervals and summing the product of the function value with the width of each interval. In other words, we think of the Riemann integral as calculating the "*area under the curve*":

$$\int_a^b f(x)dx.$$

However, the Riemann integral fails when dealing with functions that do not respect the "*area under the curve*" analogy, such as those with discontinuities or unbounded

oscillations. In these cases, the **Lebesgue integral** will be more suitable.

Intuitively, the Lebesgue integral can be thought of as summing over subsets of the domain, whose size is calculated by a Lebesgue measure (defined in Table 2.1.2), rather than using fixed step size dx. The Lebesgue integral is thus written as:

$$\int_A f(x)d\mu(x),$$

where A is a set, and $d\mu(x)$ is a measure capable of being applied to A and f.

2.2 Stochastic Theory

2.2.1 Random Variables

I believe that the measure-theoretic definition of a random variable would seem unintuitive to those accustomed to thinking of a random variable as a mapping from outcomes to probabilities. In reality, a random variable should be considered a means of *describing outcomes using real numbers*, which allows a distribution to be more easily manipulated and enables the calculation of statistical properties such as expectation.

Definition D2.2.1. *A random variable is a $(\mathbb{R}, \Sigma)$-valued random element, meaning a (S, Σ)-valued random element in which the observation space S is the real space $\mathbb{R}$.*

Definition D2.2.2. *A (S, Σ)-valued random element is a $(\mathcal{F}, \Sigma)$-measurable function $X : \Omega \to S$ that maps from probability space $(\Omega, \mathcal{F}, \mathbb{P})$ to measurable space (S, Σ).*

Definition D2.2.3. *A function is measurable for a particular event space and state space iff the event space includes every pre-image of the state space. Formally: given measurable spaces $(\Omega, \mathcal{F})$ and (S, Σ), the function $X : \Omega \to S$ is $(\mathcal{F}, \Sigma)$-measurable iff*

$$\forall B \in \Sigma : X^{-1}(B) \in \mathcal{F}.$$

For shorthand, we can also denote measurable function X with $X : (\Omega, \mathcal{F}) \to (S, \Sigma)$.

Definition D2.2.4. *Given a measurable function $X : (\Omega, \mathcal{F}) \to (S, \Sigma)$, we have inverse:*

$$X^{-1}(B) = \{a \in \Omega \mid X(a) \in B\}.$$

Finally, to construct and reason about the distribution of a random variable in practice, we require the concept of a *pushforward measure* (**D2.2.5**). At this point, it may be informative to those unfamiliar with measure theory to walk through a simple example using a pushforward measure; while it is not directly required for the narrative of this paper, I provide such an example in Appendix **A**.

Definition D2.2.5. *Given a measurable function $X : (\Omega, \mathcal{F}) \to (S, \Sigma)$ and measure space*

$(\Omega, \mathcal{F}, \mu)$, the pushforward measure $\mu_X : \Sigma \to \mathbb{R}$ is a measure on (S, Σ), defined as

$$\mu_X(B) = \mu(X^{-1}(B)), \quad \forall B \in \Sigma.$$

2.2.2 Stochastic Processes

At the most basic level, a **stochastic process** is simply a random element whose distribution depends on time; this concept opens a connection between the fields of *differential calculus* and *probability theory.*

Definition D2.2.6. *A stochastic process is a collection of (S, Σ)-valued random elements indexed by time. In probability space $(\Omega, \mathcal{F}, \mathbb{P})$, we can denote a stochastic process X as*

$$\{X_t \mid t \in T\},$$

where each $X_t : \Omega \to S$ is a (S, Σ)-valued random element.

2.2.3 Trajectories

A trajectory is the observation of a stochastic process over time. To gain intuition, it may be instructive to compare the role of outcome $\omega \in \Omega$ in determining the trajectory $\{X_t(\omega) \mid t \in T\}$ with that of a *random seed* in shaping the outputs of a pseudorandom number generator.

Definition D2.2.7. *A trajectory, or path, is a collection of observations for a stochastic process for a particular outcome $\omega \in \Omega$ over time:*

$$X(\omega) = \{X_t(\omega) \mid t \in T\}.$$

Definition D2.2.8. *A path space is the collection of all possible paths for a stochastic process $X : \Omega \to S$:*

$$\{\{X_t(\omega) \mid t \in T\} \mid \omega \in \Omega\}.$$

Definition D2.2.9. *We use the notation $C(T, S)$ to represent the path space of all continuous paths in interval T over all stochastic processes X with observation space S:*

$$C([0, 1], S) = \{\{\{X_t(\omega) \mid t \in T\} \mid \omega \in \Omega\} \mid X_t(\omega) : S\}.$$

One can use a path measure over a path space to compute the probability of particular events occurring within a stochastic process, such as a trajectory remaining within a particular range.

Definition D2.2.10. *A path measure is a probability measure defined on the path space*

of a stochastic process. Given a stochastic process X, its path space Ω, and a σ-algebra $\mathcal{F}$ on Ω, the path measure P_X is a probability measure on the measurable space $(\Omega, \mathcal{F})$:

$$P_X : \mathcal{F} \to [0, 1].$$

Definition D2.2.11. *We use the notation $\mathcal{D}(\mu_1, \mu_2)$ to represent the set of path measures associated with stochastic processes satisfying marginal distributions μ_0 and μ_1:*

$$\mathcal{D}(\mu_1, \mu_2) = \{P_X \mid X_0 \sim \mu_1 \wedge X_1 \sim \mu_2\}.$$

2.2.4 Filtration Adaptation

A useful notion in manipulating stochastic processes is the idea of **filtration adaptation**.

While this is typically described in literature as "*preventing a process from seeing into the future*", I believe a more suitable description would be to say that filtration adaptation "*introduces a notion of time in a process's event space*". Naturally, it is only once this notion is introduced that we can talk in terms of a process being able to see events across time, and naturally, our definition of time should only permit it to see events up to the present timestamp.

Definition D2.2.12. *A stochastic process is filtration-adapted iff its state space can be totally explored by the filtration schedule. Formally: a stochastic process X is adapted for a filtration $\mathbb{F} := (\mathcal{F}_t)_{t \in T}$ iff X_t is $(\mathcal{F}_t, \Sigma)$-measurable for all $t \in T$.*

Definition D2.2.13. *A filtration $\mathbb{F} := (\mathcal{F}_t)_{t \in T}$ of a σ-algebra $\mathcal{F}$ is a totally ordered collection of subsets of $\mathcal{F}$, such that*

$$\forall s \leq t : \mathcal{F}_s \subseteq \mathcal{F}_t \subseteq \mathcal{F}.$$

In this paper, we consider only stochastic processes that are $\mathbb{F}$-adapted. This property is essential for Itô-integration, as defined in **D2.3.2**.

2.3 Applying Stochastic Theory

2.3.1 Brownian Motion

One of the most fundamental stochastic processes is **Brownian motion**, also known as the *Wiener process*, which is a continuous stochastic process with independent, Gaussian-distributed increments. I provide a visualisation of the process in Figure 2.1.

Definition D2.3.1. *A stochastic process $W = \{W_t \mid t \in T\}$ is said to be a Brownian motion on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ if it satisfies the following conditions:*

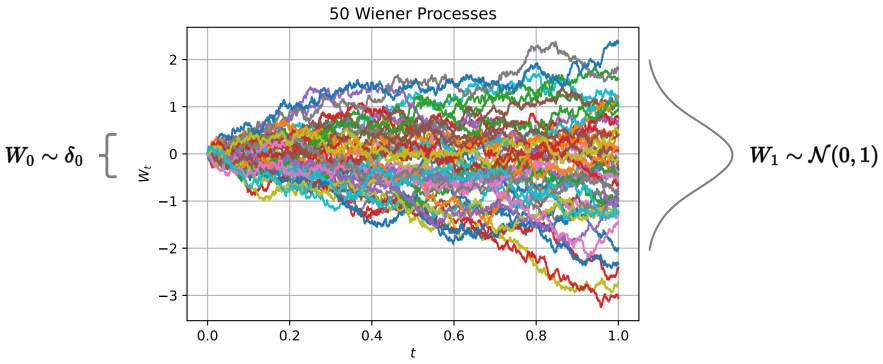

Figure 2.1: 50 Wiener processes (Brownian motion) simulated using NumPy's `np.random.normal(0, 1, (N, M))` command from time $t = 0$ to 1. Wiener processes always transport the distribution $W_0 \sim \delta_0 = 0$ to a normal distribution $\mathcal{N}(0, \cdot)$.

1. $W_0 = 0$.
2. *Increments are Gaussian-distributed such that* $W_t - W_s \sim \mathcal{N}(0, t - s)$*, for all* $0 \leq s < t < \infty$.
3. *Increments are independently distributed such that* $W_t - W_s$ *is independent of* W_r *for all* $0 \leq r < s < t < \infty$.
4. $W_t(\omega)$ *is continuous in* t.

We can use the notation P_{W_t} *to denote the distribution of* W_t *as a pushforward probability measure.*

2.3.2 Itô Integral

An Itô integral is an integral in which the integrator is a stochastic process (typically Brownian motion). This can be used, for example, to calculate a property of a process with respect to all possible trajectories over time. Informally, the Itô integral can be defined as the sum of Riemann integrals over a limit $n \to \infty$.

Definition D2.3.2. *The* ***Itô integral*** *of stochastic process* X *with respect to Brownian motion* W *over the interval* $[0, T]$ *is denoted by*

$$\int_0^T X_t dW_t,$$

and can be defined as the limit in probability of the Riemann-Stieltjes sums:[1]

$$\lim_{n \to \infty} \sum_{i=1}^{n} (X_{t_{i-1}})(W_{t_i} - W_{t_{i-1}}),$$

[1]This definition is an abstraction of the more general Itô integral, which involves a number of steps too complicated to cover here; the reader is advised to investigate these resources on stochastic theory if they wish to learn more: `https://vittoriasilvestri.files.wordpress.com/2016/11/lecture_notes_201725.pdf` and `https://www.statslab.cam.ac.uk/~james/Lectures/stochcalc.pdf`.

where $0 = t_0 < t_1 < \cdots < t_n = T$ is a sequence of partition points that converge to a continuous-time process as $n \to \infty$. X must be adapted to W's filtration $(\mathcal{F}_t)_{t \in T}$, such that $X_{t_{i-1}}(\omega)(W_{t_i}(\omega) - W_{t_{i-1}}(\omega))$ always exists.

Note that the Itô integral is itself a stochastic process, indexed by T.

2.3.3 Itô Processes

An Itô process can effectively be thought of as an evolving Brownian motion, in which each Gaussian step has an expectation determined by μ and variance determined by σ.

Definition D2.3.3. *An **Itô process**, also known as drift-augmented Brownian motion, is a continuous stochastic process that can be expressed as the sum of a Lebesgue integral and an Itô integral:*

$$X_T = X_0 + \int_0^T \mu_t dt + \int_0^T \sigma_t dW_t,$$

where coefficients μ ("drift") and σ ("diffusion") are deterministic functions which may depend on current and past values of X as well as current and past values of other processes. The Itô process can also be expressed as a stochastic differential equation (SDE):

$$dX_t = \mu_t dt + \sigma_t dW_t.$$

Definition D2.3.4. *A **stochastic delay process** is an Itô process in which the drift and diffusion terms depend only on the process's own present and past values:*

$$X_T = X_0 + \int_0^T \mu(X_{(0:t)}, t) dt + \int_0^T \sigma(X_{(0:t)}, t) dW_t,$$

where μ ("drift") and σ ("diffusion") are deterministic functions, using the trajectory notation

$$X_{(0:t)} = \{X_s \mid 0 < s \leq t\}$$

to emphasise dependence on both current and past values of X. As an SDE, this takes the form:

$$dX_t = \mu(X_{(0:t)}, t) dt + \sigma(X_{(0:t)}, t) dW_t.$$

Definition D2.3.5. *A **diffusion process** is an Itô process in which the drift and diffusion terms depend only on the current value of X:*

$$X_T = X_0 + \int_0^T \mu(X_t, t) dt + \int_0^T \sigma(X_t, t) dW_t.$$

As an SDE, this takes the form:

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t.$$

2.3.4 Euler-Mayurama Discretisation

To compute properties of a stochastic process in practice, one may wish to consider its *time discretisation*. This reformulates the SDE of the process in terms of discrete intervals, thus offering a computable (approximate) solution when taking non-infinitesimal timesteps.

Definition D2.3.6. *The **Euler-Mayurama (EM) discretisation** can be used to create a numerical solution to a **diffusion process** (**D2.3.5**). Assuming $X_0 = \delta_0 = 0$, we solve:*

$$\lim_{K\to\infty} Y_T = \int_0^T \mu(X_t, t)dt + \int_0^T \sigma(X_t, t)dW_t,$$

by recursively defining Y such that

$$\begin{aligned} Y_0 &= 0, \\ Y_{i+1} &= Y_i + \mu(Y_i, \tau_i)\Delta t + \sigma(Y_i, \tau_i)\Delta W_i, \end{aligned}$$

using the K discrete time intervals,

$$0 = \tau_0, \qquad \tau_{i+1} = \tau_i + \Delta t, \qquad \Delta t = T/K,$$

and discrete Wiener process steps $\Delta W_i = W_{\tau_{i+1}} - W_{\tau_i}$.

For an example of Euler-Mayurama discretisation applied to a specific diffusion process (a *neural Schrödinger-Föllmer process*), see **D3.2.1**.

2.4 Function Analysis and Inequalities

There are many mechanisms to describe what I informally call the "regularity" of continuous functions, which are important in deriving theoretical optimisation guarantees. This section addresses several properties that relate to "regularity", both for probability measures and for arbitrary continuous functions.

2.4.1 Smoothness

Smoothness is a property that a function possesses if it is *infinitely differentiable* — i.e., we can take derivatives of the function to any order, and the result is always a well-defined and continuous function.

Definition D2.4.1. *A function $f : \mathbb{R}^p \to \mathbb{R}$ is said to be **smooth**, or **C-Infinity (C^∞)**, iff for all $x \in \mathbb{R}^p$, for any $k \in \mathbb{Z}^+$, the kth derivative $\nabla^k f(x)$ exists and is continuous.*

Note that "smoothness" is a stronger condition than both Lipschitz continuity (**D2.4.2**) and convexity (**D2.4.3**); a function can be Lipschitz continuous or convex without being smooth. For example, the absolute value function $|x| = x\,\mathrm{sgn}(x)$ is 1-Lipschitz continuous and convex but not smooth, since it is non-differentiable at $x = 0$.

2.4.2 Lipschitz Continuity

Intuitively, a function is *Lipschitz Continuous* if its rate of change can be limited by some constant L. We tend to abbreviate this description to simply *L-Lipschitz.*

Definition D2.4.2. *A function $f : \mathbb{R}^p \longmapsto \mathbb{R}^q$ is **Lipschitz Continuous**, also known as **L-Lipschitz**, iff there exists some constant ("Lipschitz constant") $L \in \mathbb{R}$ such that:*

$$\|f(x_1) - f(x_2)\| \leq L\|x_1 - x_2\| \quad \textit{for all } x_1, x_2 \in \mathbb{R}^p, \tag{2.1}$$

*and this function can be called **bi-Lipschitz continuous** iff its inverse f^{-1} is also Lipschitz continuous.*

2.4.3 Convexity

We can describe a continuous function as *convex* when the values of the function between any two points on its graph lie beneath the line segment connecting those two points. To put it formally:

Definition D2.4.3. *Function $f : \mathbb{R}^p \to \mathbb{R}$ is convex iff we have that, for all $\theta \in [0, 1]$ and $x, x' \in \mathbb{R}^p$,*

$$f(\theta x + (1 - \theta)x') \;\leq\; \theta f(x) + (1 - \theta)f(x').$$

This notion of convexity can be extended to **sets** of values by the intuitive notion that one should be able to interpolate between any two points of a convex set without leaving said set:

Definition D2.4.4. *Set $C \subseteq \mathbb{R}$ is convex iff we have that, for all $\theta \in [0, 1]$ and $x, x' \in C$,*

$$\theta x + (1 - \theta)x' \in C.$$

Our two definitions of convexity **D2.4.3** and **D2.4.4** can be consolidated by considering the epigraph of f,

$$\mathrm{epi}(f) = \{(x, y) | f(x) \leq y\},$$

giving us the equivalence:

$$\text{Function } f \text{ is convex} \iff \mathrm{epi}(f) \text{ is convex.}$$

2.4.4 Logarithmic Sobolev Inequality

The logarithmic Sobolev inequality (LSI) highlights the relationship between the *entropy* (expected surprisal) and *gradient* of functions under a particular probability measure. In simple terms, it quantifies the idea that functions with small gradients should not concentrate too much of the measure.

Definition D2.4.5. *Probability measure $\mu : \mathbb{R}^p \to \mathbb{R}$ satisfies the **log-Sobolev inequality** (LSI) with constant $C > 0$ if for any smooth function f we have:*

$$H_\mu(f^2) \leq C \int |\nabla f|^2 d\mu,$$

where $H_\mu(f^2) = \int f^2 \log \left[f^2 / (\int f^2 d\mu) \right] d\mu$ is the entropy of f^2 in μ.

2.5 Density Problems

2.5.1 The Sampling Problem

The **sampling problem** is the task of generating samples according to a target distribution; for example, motivation for this could be to estimate properties like variance and expectation. Put simply:

$$\texttt{density} \to \texttt{samples}.$$

I found that the general task is often defined only informally in literature, as it can be seen more as an algorithmic task than an abstract, mathematical one.

To reach a formal definition, I take the approach of assuming that we have a computer capable of sampling from a uniform distribution (such as via a pseudorandom number generator like the *Mersenne Twister* or the *Xorshift* family of algorithms), at which point one can clarify the problem from a more mathematical perspective:

Definition D2.5.1. *The $\mathbb{P}$-density sampling problem is the task of mapping a uniform probability space $(\Omega, \mathcal{F}, \lambda)$, where λ is the Lebesgue measure, to a "target" probability space $(\Omega^*, \mathcal{F}^*, \mathbb{P})$, assuming we have access to an analytical form of $\mathbb{P}$.*

It is solved by finding a measurable function $X : (\Omega, \mathcal{F}) \to (\Omega^*, \mathcal{F}^*)$ that minimises the distance between the pushforward measure λ_X and the target measure $\mathbb{P}$:

$$X \in \{\arg\min_X d(\lambda_X, \mathbb{P})\},$$

where $d(\cdot, \cdot)$ is a distance metric between probability measures, such as the total variation distance, Kullback-Leibler divergence, or Wasserstein distance.

2.5.2 The Density Estimation Problem

The **density estimation problem** is the task of deriving a probability measure from a given dataset of samples; for example, motivation for this could be to perform classification, or to estimate prior and likelihood terms for Bayesian inference. This can be seen as the *inverse* of the sampling problem. Put simply:

$$\texttt{samples} \to \texttt{density}.$$

Definition D2.5.2. *The $\mathcal{D}$-density estimation problem is the task of translating a given dataset $\mathcal{D} = \{x_1, x_2, \ldots, x_n\} \in \Omega^n$ to the underlying probability measure $\mathbb{P}$ from which $\mathcal{D}$ was sampled in $(\Omega, \mathcal{F})$-measurable space.*

It is solved by finding a measure $\mathbb{Q} : \mathcal{F} \to \mathbb{R}$ that minimises the distance from target measure $\mathbb{P}$:

$$\mathbb{Q} \in \{\arg\min_{\mathbb{Q}} d(\mathbb{Q}, \mathbb{P})\},$$

where $d(\cdot, \cdot)$ is a distance metric between probability measures, such as the total variation distance, Kullback-Leibler divergence, or Wasserstein distance.

2.5.3 Data-Based Generative Modelling

The concepts of density estimation and density sampling can be merged to develop a dataset-based *generative* model. Put simply:

$$\texttt{samples} \to \texttt{density} \to \texttt{samples}.$$

Here, generative model X can extrapolate a given dataset $\mathcal{D}$ with underlying unknown probability measure $\mathbb{P}$, such that we solve:

$$\mathbb{Q} \in \{\arg\min_{\mathbb{Q}} d(\mathbb{Q}, \mathbb{P})\},$$
$$X \in \{\arg\min_{X} d(\lambda_X, \mathbb{Q})\}.$$

For instance, a Generative Adversarial Network (GAN) can be seen as an embodiment of this framework, where a critic network essentially carries out density estimation using the provided samples, while the generator network conducts sampling based on the critic's estimation. The innovative aspect of the GAN lies in the fact that the generator's output is employed to refine the critic, establishing a dynamic interplay between the two components.

2.6 The Schrödinger Bridge Problem

2.6.1 Generalised Form

Earlier, we defined the *sampling problem* as finding a mapping $X : (\Omega, \mathcal{F}) \to (\Omega^*, \mathcal{F}^*)$ from a uniform distribution to a target distribution, for which there exists a set of solutions.

We can generalise this problem by considering mapping between two arbitrary probability measures $\mathbb{Q}$ and $\mathbb{P}$,

$$X \in \{\arg\min_X d(\mathbb{Q}_X, \mathbb{P})\},$$

in which $\mathbb{Q}_X$ is the pushforward measure (**D2.2.5**) produced by X. We can also constrain the solution set $\mathcal{X}$ to a single solution by minimising the distance between the joint distribution described by X and a prior joint distribution $Y \otimes Y$, so long as $\mathbb{Q}$ and $\mathbb{P}$ have the same state space:

$$\begin{aligned}\mathcal{X} &= \{\arg\min_X d(\mathbb{Q}_X, \mathbb{P})\},\\ X &= \arg\min_{X \in \mathcal{X}} d(\mathbb{Q} \otimes \mathbb{Q}_X, Y \otimes Y),\end{aligned}$$

in which where $\mathbb{Q} \otimes \mathbb{Q}_X$ denotes the joint distribution of $\mathbb{Q}$ and $\mathbb{Q}_X$.

We have now generalised the sampling problem to the problem of transforming one distribution into another, for which only one solution is available according to a prior Y. I call this problem the *generalised Schrödinger bridge problem*.

Definition D2.6.1. *The **generalised Schrödinger bridge problem** for probability measures $\mathbb{Q} : \Omega \to \mathbb{R}$ and $\mathbb{P} : \Omega \to \mathbb{R}$ and prior joint distribution $Y \otimes Y$ is that of finding a mapping $X : (\Omega, \mathcal{F}) \to (\Omega, \mathcal{F})$ to transform $\mathbb{Q}$ into $\mathbb{P}$, such that*

$$\begin{aligned}\mathcal{X} &= \{\arg\min_X d(\mathbb{Q}_X, \mathbb{P})\},\\ X &= \arg\min_{X \in \mathcal{X}} d(\mathbb{Q} \otimes \mathbb{Q}_X, Y \otimes Y),\end{aligned}$$

for some distance metric d.

2.6.2 Static Form

The more typical instance of the Schrödinger bridge problem is the formulation in the real domain ($\Omega = \mathbb{R}^n$) that uses a Wiener proces prior W for Y, and KL divergence for d. This is known as the *static Schrödinger bridge problem*.

Definition D2.6.2. *The **static Schrödinger bridge problem** for probability measures $\mathbb{Q} : \mathbb{R}^n \to \mathbb{R}$ and $\mathbb{P} : \mathbb{R}^n \to \mathbb{R}$ is that of finding a mapping $X : (\mathbb{R}^n, \mathcal{F}) \to (\mathbb{R}^n, \mathcal{F})$ to*

transform $\mathbb{Q}$ into $\mathbb{P}$ while remaining close to a Wiener prior, such that

$$\mathcal{X} = \{\arg\min_X D_{KL}(\mathbb{Q}_X \parallel \mathbb{P})\},$$
$$X = \arg\min_{X \in \mathcal{X}} D_{KL}(\mathbb{Q} \otimes \mathbb{Q}_X \parallel P_{W_0} \otimes P_{W_1}),$$

where P_{W_t} is the distribution of random variable W_t from the Wiener process W^n at t.

Definition D2.6.3. *The **Kullback–Leibler (KL) divergence** between two probability measures for probability spaces $(\Omega, \mathcal{F}, \mathbb{Q})$ and $(\Omega, \mathcal{F}, \mathbb{P})$ is defined as*

$$D_{KL}(\mathbb{Q} \parallel \mathbb{P}) = \int_\Omega \log\left(\frac{\mathbb{Q}(dx)}{\mathbb{P}(dx)}\right) \mathbb{Q}(dx).$$

2.6.3 Dynamic Form

In the dynamic formulation, we seek a path measure $P_X : C([0,1], \mathbb{R}^n)$ over a stochastic process X, rather than assuming X to simply be a measurable function.

Note that, for convenience, at this point most authors make the assumption that the target distribution $\mathbb{P}$ can be reached by P_{X_1} exactly, allowing us to drop the target-matching $\mathcal{X}$ term and summarise the problem as finding a path measure $P_X : C([0,1], \mathbb{R}^n) \to \mathbb{R}$ over the set $\mathcal{D}(\mathbb{Q}, \mathbb{P})$:

Definition D2.6.4. *The **dynamic Schrödinger bridge problem** for probability measures $\mathbb{Q} : \mathbb{R}^n \to \mathbb{R}$ and $\mathbb{P} : \mathbb{R}^n \to \mathbb{R}$ is that of finding a path measure $P_X : C([0,1], \mathbb{R}^n) \to \mathbb{R}$ and associated stochastic process $X = \{X_t \mid t \in [0,1]\}$, $X_t : (\mathbb{R}^n, \mathcal{F}) \to (\mathbb{R}^n, \mathcal{F})$ to transform $\mathbb{Q}$ into $\mathbb{P}$, such that*

$$P_X = \arg\min_{P_X \in \mathcal{D}(\mathbb{Q}, \mathbb{P})} D_{KL}(P_X \parallel P_W),$$

where P_W is path measure for the Wiener process W^n.

Explicit Form

For the sake of this paper, it will still be useful to consider cases in which the target distribution $\mathbb{P}$ cannot be reached exactly and thus include a target-matching $\mathcal{X}$ term. I refer to this as the "explicit" form of the dynamic Schrödinger bridge problem:

Definition D2.6.5. *The **explicit dynamic Schrödinger bridge problem** for probability measures $\mathbb{Q} : \mathbb{R}^n \to \mathbb{R}$ and $\mathbb{P} : \mathbb{R}^n \to \mathbb{R}$ is that of finding a stochastic process*

$X = \{X_t \mid t \in [0,1]\}$, $X_t : (\mathbb{R}^n, \mathcal{F}) \to (\mathbb{R}^n, \mathcal{F})$ *to transform* $\mathbb{Q}$ *into* $\mathbb{P}$*, such that*

$$\mathcal{X} = \underset{X \in \{X \mid X_0 \sim \mathbb{Q}\}}{\arg\min} D_{KL}(P_{X_1} \parallel \mathbb{P}),$$
$$X = \underset{X \in \mathcal{X}}{\arg\min}\, D_{KL}(P_X \parallel P_W),$$

where P_{X_t} *is the probability measure for random variable* X_t*, and* P_W *is the path measure (probability measure) for Wiener process* W^n*.*

2.6.4 Dynamic Form with Stochastic Control

To approach the problem with stochastic control, we assume the underlying stochastic process X to be an identity-variance Itô process (i.e. one for which diffusion term $\sigma = 1$).

Like in Section 2.6.3, we will start by defining the non-explicit form that assumes one has access to $\mathcal{D}(\mathbb{Q}, \mathbb{P})$ to match target distribution $\mathbb{P}$ exactly. Thus the problem is reduced to minimising distance to a Wiener process, which, importantly, can be formulated as *minimising the process's drift term*, as this allows the diffusion-driven Wiener term dW_t to take over.

The specific form in which the drift term is minimised ($\mathbb{E}\left[\frac{1}{2}\int_0^1 \|\mu_t\|^2 dt\right]$) can be derived as equivalent to $D_{KL}(P_X \parallel P_W)$ through the Girsanov formula.

Definition D2.6.6. *In* ***stochastic control terms****, the* ***dynamic Schrödinger bridge problem*** *for probability measures* $\mathbb{Q} : \mathbb{R}^n \to \mathbb{R}$ *and* $\mathbb{P} : \mathbb{R}^n \to \mathbb{R}$ *is that of finding the minimal drift term* μ *for an identity-variance Itô process* $X = \{X_t \mid t \in [0,1]\}$*,* $X_t : (\mathbb{R}^n, \mathcal{F}) \to (\mathbb{R}^n, \mathcal{F})$ *to transform* $\mathbb{Q}$ *into* $\mathbb{P}$*, such that given the Itô process,*

$$X_T = (X_0 \sim \mathbb{Q}) + \int_0^T \mu_t dt + \int_0^T dW_t,$$

and set of viable drifts from $\mathbb{Q}$ *to* $\mathbb{P}$*,*

$$U = \{\mu \mid P_X \in \mathcal{D}(\mathbb{Q}, \mathbb{P})\},$$

we seek the minimal drift:

$$\mu = \underset{\mu \in U}{\arg\min}\, \mathbb{E}\left[\frac{1}{2}\int_0^1 \|\mu_t\|^2 dt\right].$$

Explicit Form

As with Section 2.6.3, it is also helpful to consider the "explicit" case in which the exact solution set $\mathcal{D}(\mathbb{Q}, \mathbb{P})$ is not directly accessible, requiring us to alter the drift term's minimisation goal to reach $\mathbb{P}$ at X_1. However, reaching this form requires further assumptions on the bridge problem.

In particular, when assuming the initial distribution $\mathbb{Q}$ of the Schrödinger bridge problem to be the Dirac distribution $\delta_0 = 0$, the drift that solves the problem is known as a **Föllmer** drift. This assumption enables the problem to be expressed explicitly by adding a counterweight $\log \frac{P_{W_1}(X_1)}{\mathbb{P}(X_1)}$ to the drift goal [17, 14, 10], which I will provide intuition for below.

Definition D2.6.7. *In **stochastic control terms**, the **explicit dynamic Schrödinger bridge problem** for probability measures $\delta_0 : \mathbb{R}^n \to \mathbb{R}$ and $\mathbb{P} : \mathbb{R}^n \to \mathbb{R}$ is that of finding the minimal drift term μ for an identity-variance Itô process $X = \{X_t \mid t \in [0, 1]\}$, $X_t : (\mathbb{R}^n, \mathcal{F}) \to (\mathbb{R}^n, \mathcal{F})$ to transform δ_0 into $\mathbb{P}$, such that given the Itô process,*

$$X_T = (X_0 \sim \delta_0) + \int_0^T \mu_t dt + \int_0^T dW_t,$$

we seek the minimal drift to reach $\mathbb{P}$:

$$\mu = \arg\min_{\mu} \mathbb{E}\left[\frac{1}{2}\int_0^1 \|\mu_t\|^2 dt + \log \frac{P_{W_1}(X_1)}{\mathbb{P}(X_1)}\right].$$

Definition D2.6.8. *The **Dirac distribution** δ_0 is defined by*

$$\delta_0(x) = \begin{cases} 1 & x = 0 \\ 0 & \textit{otherwise.} \end{cases}$$

To gain insight, consider how one might alter the drift in Definition **D2.6.6** to reach $\mathbb{P}$ at X_1. Intuitively, we can exclude the X_1 distribution from our minimisation-towards-Wiener goal by additionally minimising $P_{W_1}(X_1)$, and we can encourage X_1 towards $\mathbb{P}$ by additionally maximising $\mathbb{P}(X_1)$.

The idea here is that the Wiener term dW_t of our Itô process should prevent μ from trivially taking $X_1 = \arg\max_{X_1} \mathbb{P}(X_1)$, instead forcing X_1 to approximate the entire distribution of $\mathbb{P}$. Föllmer [17] found that this can be achieved by inserting the logarithms of the two terms into our drift goal, as shown in Definition **D2.6.7**.

Zhang et al. [10] note that this causes the process to minimise

$$D_{KL}(P_X(\tau) \parallel P_W(\tau \mid X_1)\mathbb{P}(X_1))$$

over trajectories $\tau \in C([0, 1], \mathbb{R})$, the solution to which is equivalent to solving the dynamic Schrodinger bridge problem defined in **D2.6.4**.

Chapter 3

Related work

In recent years, several authors have explored tractable stochastic control solutions to the Schrödinger-Bridge problem using a Föllmer-type flow [13, 11, 14], which solves the explicit Dirac-prior form of the SBP as defined in **D2.6.7**. The aim of this paper is to utilise this work for *global optimisation*.

Specifically, this chapter builds towards the Path Integral Optimiser (PIO) by describing Föllmer-type solutions to the Schrödinger-Bridge problem in Section 3.1, and practical solutions the global optimisation problem in Section 3.3. The former motivates the structure of PIO and explains why neural network optimisation has not previously been attempted with a Schrödinger-Föllmer flow, while the latter places this work in the context of state-of-the-art optimisers used in practical machine learning.

3.1 The Schrödinger-Föllmer Process

The Schrödinger-Föllmer process is a diffusion process that solves the Schrödinger bridge problem for initial Dirac distribution $\mathbb{Q} = \delta_0$, as problemised in Definition **D2.6.7**. In other words, the Schrödinger-Föllmer process guides a point to a target distribution while remaining as close as possible to a Wiener process.

3.1.1 Analytic Solution

Definition D3.1.1. *The **Schrödinger-Föllmer process** is an identity-variance diffusion process that solves the dynamic Schrödinger-Föllmer problem from Dirac starting point $X_0 \sim \delta_0 = \mathbb{Q}$ to arbitrary target distribution $X_1 \sim \mathbb{P}$. The process is defined as:*

$$X_T = \delta_0 + \int_0^T \frac{\mathbb{E}_{Z\sim N(0,I_n)}[\nabla f(X_t + \sqrt{1-t}Z)]}{\mathbb{E}_{Z\sim N(0,I_n)}[f(X_t + \sqrt{1-t}Z)]} dt + \int_0^T dW_t,$$

in which we take $f : \mathbb{R}^n \to \mathbb{R}^n$ *to be the ratio*

$$f(x) = \frac{d\mathbb{P}}{dN\left(0, \mathbf{I}_n\right)}(x).$$

3.1.2 Monte-Carlo Approximation

In practice, calculation of the Schrödinger-Föllmer process faces a problem: even though one may use Euler-Maruyama discretisation (**D2.3.6**) to approximate the integrals, the drift's expectation terms rarely have an analytic closed-form, except in specific cases of the target distribution (such as the distribution being a Gaussian mixture).

Huang et al. [14] proposed that we find a closed form by instead approximating the drift up to arbitrary precision using the *Monte-Carlo method*.

Definition D3.1.2. *The* ***Monte-Carlo approximation*** *to the* ***Schrödinger-Föllmer process*** *makes use of i.i.d.* $Z_i \sim N(0, I_n)$ *for all* $i \leq m \in \mathbb{Z}^+$*, defined as:*

$$X_T = \delta_0 + \int_0^T \frac{\sum_{i=1}^m [\nabla f(X_t + \sqrt{1-t} Z_i)]}{\sum_{i=1}^m [f(X_t + \sqrt{1-t} Z_i)]} dt + \int_0^T dW_t,$$

in which we take $f : \mathbb{R}^n \to \mathbb{R}^n$ *to be the ratio*

$$f(x) = \frac{d\mathbb{P}}{dN\left(0, \mathbf{I}_n\right)}(x).$$

Drawbacks

The Monte-Carlo approximation can be problematic in practice. Zhang et al. [10] highlight that the Monte-Carlo approach is analogous to importance sampling[1] of target distribution $X_1 \sim \mathbb{P}$ with proposal distribution $Z_i \sim N(0, I_n)$, which entails two major drawbacks encompassing excessively high variance:

- **Curse of Distribution Alignment**: The number of samples m to achieve a given accuracy increases exponentially with the KL divergence between proposed ($Z_i \sim N(0,\ I_n)$) and target ($\mathbb{P}$) distributions [18, 19].
- **Curse of Dimensionality**: The number of samples m to achieve a given accuracy increases exponentially with the dimension size n of our distributions [18, 19].

I also identify a third drawback of the Monte-Carlo approach, which would be particularly damaging to the use-case of neural network optimisation:

- **Re-Computation of Target Score**: The Monte-Carlo approach requires re-computing the ∇f target score m times at each timestep of the SDE, which will be

[1] To be specific, it may be more apt to compare the SF process to *annealed* importance sampling, which transforms an initial distribution into a target distribution by interpolating the geometric average of the two distributions.

impractical if the score is costly (e.g. for neural net optimisation) or inaccessible (e.g. for the density estimation problem per Definition **D2.5.2**).

Aside: Monte-Carlo Integration Error Bounds

It is worth noting that dimensionality does not appear explicitly in the standard error estimation for Monte-Carlo integration, because it instead affects the *variance* term σ_m.

Definition D3.1.3. *The error of Monte-Carlo approximation* $\frac{1}{m}\sum_{i=1}^{m} f(x_i)$ *for the integral* $\int_D f(x)dx$ *can be estimated by:*

$$E_m = \frac{\sigma_m}{\sqrt{m}},$$

where σ_m *is the standard deviation of the function* f *across the* m *samples, given by:*

$$\sigma_m = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(f(x_i) - \bar{f})^2},$$

where $\bar{f}$ *is the average value of the function* f *over the* m *samples, given by:*

$$\bar{f} = \frac{1}{m}\sum_{i=1}^{m} f(x_i).$$

3.1.3 Neural Approximation

Tzen et al. [13] found that, as an alternative to Monte-Carlo estimation, one can use a multilayer neural network to efficiently approximate the drift term of the Schrödinger-Föllmer process to arbitrary accuracy, so long as the network is capable of efficiently approximating the target density $\mathbb{P}$.

Definition D3.1.4. *The* ***neural approximation*** *to the* ***Schrödinger-Föllmer process*** *makes use of neural drift term* $\widehat{b}_\theta : (\mathbb{R}^n, [0,1]) \to \mathbb{R}^n$*, such that* $\widehat{b}$ *is a feed-forward multilayer neural net with parameters* θ*, and is defined as:*

$$X_T = \delta_0 + \int_0^T \widehat{b}_\theta(X_t, t)dt + \int_0^T dW_t.$$

Intuition

Intuitively, deep neural networks should be able to mitigate the two "curses" described in Section 3.1.2 due to their ability to perform hierarchical feature learning. While the Monte-Carlo method (Definition **D3.1.2**) must multiply m exponentially to explore the full volume of f as dimensionality increases (otherwise missing out on high-level features), a well-formed neural network can increase in depth *linearly* to increase expressivity exponentially [20].

Consider how the neural approach extracts high-level features by transforming the input through several layers of perceptrons; by contrast, we can imagine the MC approach to be analogous to a neural network with a single layer.

3.2 Path Integral Sampler

Though Tzen et al. [13] derived theoretical guarantees for the optimal (continuous) neural-approximated Schrödinger-Föllmer process (Section 2.6.3), they did not specify how exactly the network's parameters θ might be learned.

Zhang et al. [10] tackle this by proposing the **Path Integral Sampler (PIS)**, a complete description of a neural-approximated SFP that learns parameters θ by simply re-using the drift goal in Definition **D2.6.7** as the loss function for a standard gradient-based optimisation algorithm (i.e. Adam [2]) in discrete time.

As we are now moving into the realm of discretised neural processes, for clarity, I give the complete formula for an EM-discretised neural SFP below. This is simply a combination of Definitions **D2.3.6** and **D3.1.4**; it forms the basis of the Path Integral Sampler.

Definition D3.2.1. *The **EM-discretised neural approximation** to the **Schrödinger-Föllmer process** makes use of neural drift term $\widehat{b}_\theta : (\mathbb{R}^n, [0,1]) \to \mathbb{R}^n$, such that $\widehat{b}$ is a feed-forward multilayer neural net with parameters θ, and is defined for K timesteps as:*

$$X_T = \delta_0 + \sum_{i=0}^{T} \widehat{b}_\theta(X_{t_i}, t_i)/K + \sum_{t=0}^{T} (W_{t_{i+1}} - W_{t_i}).$$

with timesteps $t_i = i/K$.

(Note that this definition leaves times t_i explicit, as there may be scope for improving training by using non-uniform timesteps instead of i/K.)

3.2.1 Training Procedure

Re-using the drift goal in Definition **D2.6.7**, the loss term for PIS is thus defined as a random variable $\mathcal{L}(\theta)$ which drives X_1 towards $\mathbb{P}$ when minimised:

Definition D3.2.2. *The **Path Integral Sampler** approximates optimal parameters θ^* to solve the explicit dynamic Schrodinger Bridge problem **D2.6.7** using the **loss function** goal $\arg\min_\theta \mathcal{L}(\theta) \approx \theta^*$, with $\mathcal{L}(\theta) : (\mathbb{R}^n, \mathcal{F}) \to (\mathbb{R}, \Sigma)$ defined as:*

$$\mathcal{L}(\theta) = \frac{1}{2} \int_0^1 \|\widehat{b}_\theta(X_t, t)\|^2 dt + \log \frac{P_{W_1}(X_1)}{\mathbb{P}(X_1)}.$$

The assumption that target distribution $\mathbb{P}$ has an analytic form in the loss term characterises the process as solving the $\mathbb{P}$-*density sampling problem* (Definition **D2.5.1** supposing

our process X's state space is uniformly distributed), hence Zhang et al. referring to the overall solution as a "*sampler*".

Algorithm 1 provides an overview of the training procedure of PIS, performing M training steps making use of an Euler-Mayurama-based sampling procedure (detailed later in Algorithm 2). In this case, the integral $\int_0^1 \|\widehat{b}_\theta(X_t, t)\|^2 dt$ is returned by our sampling procedure in variable y.

Algorithm 1: PIS Training Procedure

Input: Initial Parameters θ_0, Target Distribution $\mathbb{P}$, EM Times $\{t_0, t_1, \ldots, t_K\}$.

for $i \leftarrow 0$ **to** M **do**

1 Sample (Algorithm 2) $x, y, z, w = \texttt{PISSamplingProcedure}(\theta_i, \mathbb{P}, t)$

2 Compute loss $\mathcal{L} = y + \log(P_{W_1}(x)/\mathbb{P}(x))$

3 Update parameters $\theta_{i+1} = \text{Adam}(\theta_i, d\mathcal{L}/d\theta_i)$

end

Result: Learned Parameters θ_M.

3.2.2 Importance Sampling

Unfortunately, the PIS training procedure (minimising $\mathcal{L}(\theta)$) is not guaranteed to tend towards the optimal drift. In other words, it will not guarantee P_{X_1} to approach $\mathbb{P}$ with arbitrary accuracy.

To calibrate the final distribution to better align with target distribution $\mathbb{P}$, Zhang et al. [10] suggest the use of importance sampling, whereby we assign each trajectory a *weight* measuring the alignment of its terminal state with $\mathbb{P}$. This enables *unbiased* sampling.

Definition D3.2.3. *The **PIS importance weight** can be calculated as*

$$w(\theta) = \exp\left(-\mathcal{L}(\theta) - \int_0^1 \frac{d\widehat{b}_\theta(X_t, t)}{dX_t} dW_t\right),$$

which, unpacking $\mathcal{L}(\theta)$, expands to

$$w(\theta) = \exp\left(-\int_0^1 \left[\frac{1}{2}\|\widehat{b}_\theta(X_t, t)\|^2 dt + \frac{d\widehat{b}_\theta(X_t, t)}{dX_t} dW_t\right] - \log\frac{P_{W_1}(X_1)}{\mathbb{P}(X_1)}\right).$$

Example

To give a brief demonstration of importance sampling, let us consider N samples $x_1, x_2, ..., x_N$ drawn from the terminal random variable X_1 of the PIS, and assume each sample x_i has an associated importance weight w_i calculated according to Definition **D3.2.3**.

Suppose we wish to estimate the expectation of a function $f(x)$ with respect to the target distribution $\mathbb{P}$. Instead of simply taking the average of $f(x)$ over the samples (as we would

if $X_1 \sim \mathbb{P}$), one can calculate a *weighted* average in which each sample's contribution is scaled by its importance weight:

$$\mathbb{E}_{\mathbb{P}}[f(X)] \approx \frac{\sum_{i=1}^{N} w_i f(x_i)}{\sum_{i=1}^{N} w_i}.$$

This weighted average corrects for the bias in the sample distribution P_{X_1}, providing an unbiased estimate of the expectation of $f(x)$ under the target distribution $\mathbb{P}$.

3.2.3 Network Choice

Zhang et al. [10] describe two possible formulations of PIS's neural drift:

1. A single feed-forward neural network parameterised by θ.

   Definition D3.2.4. ***PIS-NN*** *is a Path Integral Sampler which uses the following drift, where NN_θ is a neural network:*

   $$\widehat{b}_\theta(X_t, t) = NN_\theta(X_t, t).$$

2. The sum of two feed-forward neural networks, one of which is *time-conditioned* and multiplied by the score $\nabla \log \mathbb{P}(X_t)$. The incorporation of this gradient information was inspired by the improved convergence time it seems to offer in MCMC models [21, 22]. However, it does of course come with the penalty of re-computing score at every timestep.

   Definition D3.2.5. ***PIS-Grad*** *is a Path Integral Sampler which uses the following drift, where θ is split into (θ_1, θ_2), and $(NN_{\theta_1}, \widetilde{NN}_{\theta_2})$ are neural networks:*

   $$\widehat{b}_\theta(X_t, t) = NN_{\theta_1}(X_t, t) + \widetilde{NN}_{\theta_2}(\sin t, \cos t) \times \nabla \log \mathbb{P}(X_t).$$

3.2.4 Sampling Procedure

Algorithm 2 summarises the sampling procedure of PIS, which uses Euler-Maruyama discretisation (Definition **D2.3.6**) to simulate the neural Schrödinger-Föllmer process (Definition **D3.1.4**; discretised in **D3.2.1**) learned by PIS from $X_0 \sim \delta_0$ to X_1.

Using the weight formula in Definition **D3.2.3**, it also returns an *importance weight* which may be used to make the terminal X_1 sample unbiased with respect to $\mathbb{P}$.

Algorithm 2: PIS Sampling Procedure

Input: Parameters θ, Target Distribution $\mathbb{P}$, EM Times $\{t_0, t_1, \ldots, t_K\}$.

Initialise: Initial Sums $\mathbf{x}_0 = 0, y_0 = 0, z_0 = 0$.

for $i \leftarrow 0$ **to** K **do**

1	Compute time step	$\Delta t = t_{i+1} - t_i$
2	Sample Wiener step	$\Delta W \sim \mathcal{N}(0, \Delta t \mathcal{I})$
3	Update state	$x_{i+1} = x_i + \widehat{b}_\theta(x_i, t_i)\Delta t + \Delta W$
4	Update reg sum	$y_{i+1} = y_i + \frac{1}{2}\|\widehat{b}_\theta(x_i, t_i)\|^2 \Delta t$
5	Update grad sum	$z_{i+1} = z_i + (d\widehat{b}_\theta(x_i, t_i)/dx_i)\Delta W$

end

Result: Terminal Sample x_K, Sums y_K, z_K,
Importance Weight $w = \exp\left(-y_K - z_K - \log \frac{P_{W_1}(x_K)}{\mathbb{P}(x_K)}\right)$.

3.3 Optimisation

This section introduces the general problem of *optimisation* and common optimisation algorithms found in the literature. While this material may already be familiar to the reader, it will be helpful to define these optimisers in a consistent notation for the sake of creating clear comparisons.

Definition D3.3.1. *Given a function $f : \mathbb{R}^p \to \mathbb{R}$, one can define the problem of* ***f-optimisation***[2] *as being identifying any solution x_* such that:*

$$x_* \in \arg\min_{x \in \mathbb{R}^p} f(x).$$

Definition D3.3.2. *Given a stochastic function $f : \mathbb{R}^p \to (\Omega \to \mathbb{R})$, where $f(\cdot)$ is a random variable, one can define the problem of* ***stochastic f-optimisation*** *as being identifying any solution x_* such that:*

$$x_* \in \arg\min_{x \in \mathbb{R}^p} \mathbb{E}_{\omega \in \Omega}\left[f(x; \omega)\right],$$

which is often abbreviated to simply $x_* \in \arg\min_{x \in \mathbb{R}^p} f(x)$.

Per Definition **D3.3.2**, one can consider the task of *training a neural network* to simply be a stochastic $\mathcal{L}$-optimisation problem over parameters θ and loss function $\mathcal{L}$, but with two important characteristics differentiating it from the general case:

- **High Dimensionality**: The domain of θ may be extremely high dimensional (e.g. $p > 1000$).

[2] Global, continuous optimisation.

- **High Computational Cost**: The loss function $\mathcal{L}$ may be computationally expensive (as it requires a forward pass of the network).

3.3.1 Popular Algorithms

Stochastic Gradient Descent

The prototypical algorithm for solving stochastic optimisation over a neural network is *Stochastic Gradient Descent (SGD)* [1], an iterative gradient-based technique outlined in Algorithm 3.

Algorithm 3: Stochastic Gradient Descent (SGD)

Input: Parameter Initialisation θ_0, Learning Rate γ_k (default 10^{-3}).

for $k \leftarrow 0$ **to** N **do**

1 Compute loss gradient $g_k = \nabla \mathcal{L}(\theta_k)$

2 Update parameters $\theta_{k+1} = \theta_k - \gamma_k g_k$

end

Result: Learned Parameters θ_N.

Adam

Adam [2] is a popular extension of SGD. It accumulates gradient information on each parameter (specifically, the running average of first and second moments) and uses this information to determine the preconditioner by which we multiply γ_k.

The gradient information is forgotten exponentially over time according to β_1, β_2 parameters, effectively lending a *momentum* to the optimiser's updates. I outline Adam in Algorithm 4.

Algorithm 4: Adam

Input: Parameter Initialisation θ_0, Learning Rate γ_k (default 10^{-3}), Decay Rates β_1, β_2, Fuzz Factor ϵ (default 10^{-8}).

Initialise: Initial Moments $m_0 = 0, v_0 = 0$.

for $k \leftarrow 0$ **to** N **do**

1 Compute loss gradient $g_k = \nabla \mathcal{L}(\theta_k)$

2 Update biased first moment $m_{k+1} = \beta_1 m_k + (1 - \beta_1) g_k$

3 Update biased second moment $v_{k+1} = \beta_2 v_k + (1 - \beta_2) g_k^2$

4 Correct bias $\widehat{m} = m_{k+1} / (1 - \beta_1^k)$

5 Correct bias $\widehat{v} = v_{k+1} / (1 - \beta_2^k)$

6 Update parameters $\theta_{k+1} = \theta_k - \gamma_k \widehat{m} / \left(\sqrt{\widehat{v}} + \epsilon \right)$

end

Result: Learned Parameters θ_N.

3.3.2 Boltzmann Transformation: Optimisation as Sampling

We can transform a *stochastic $\mathcal{L}$-optimisation problem* (Definition **D3.3.2**) into a $\mathbb{P}$-*sampling problem* (Definition **D2.5.1**) by carefully reformulating our loss function to act as a probability distribution over the set of optimal solutions. This enables us to solve the optimisation problem using a sampler.

The idea here is to use the parameter σ to make $\mathcal{L}$'s minimal points exponentially more probable than any other in its domain, and normalise the function into a probability measure by dividing by its integral. The resulting density describes what is known as a *Boltzmann distribution*, hence I refer to the conversion as a "Boltzmann transformation".

Definition D3.3.3. *Given loss function $\mathcal{L} : \mathbb{R}^p \to (\Omega \to \mathbb{R})$ and scalar $\sigma \in (0, 1]$, we define the* ***Boltzmann density*** *$\mathbb{P}$ as*

$$\mathbb{P}(\theta) = \frac{\exp(-\mathcal{L}(\theta)/\sigma)}{\int_{\mathbb{R}^p} \exp(-\mathcal{L}(x)/\sigma)dx}.$$

For sufficiently small σ, sampling from $\mathbb{P}$ is equivalent to identifying the optimal parameters $\theta_* \in \arg\min_{\theta\in\mathbb{R}^p} \mathcal{L}(\theta)$ such that $\lim_{\sigma\to 0} \mathbb{P}(\theta_*) = 1$.

3.3.3 Schrödinger-Föllmer Process Feasibility

Because Schrödinger-Föllmer processes (**D3.1.1**) are scale-invariant with regard to the target distribution—owing to the fact that constant factors in f are cancelled out by the $\nabla f/f$ division in the drift term—one can ignore the Boltzmann distribution's (**D3.3.3**) integral denominator $\int_{\mathbb{R}^p} \exp(-\mathcal{L}(x)/\sigma)dx$ and thus sample from a Boltzmann distribution with the more readily-computable formula $\mathbb{P}(\theta) = \exp(-\mathcal{L}(\theta)/\sigma)$.

Dai et al. [15] derived guarantees for the Monte-Carlo Schrödinger-Föllmer process (**D3.1.2**) at the task of non-convex deterministic optimisation (**D3.3.1**) using the such a Boltzmann transformation. To enable the drift term to appropriately match the scale dictated by σ, Dai et al. rescaled the process's terms using σ and $\sqrt{\sigma}$:[3]

$$X_1 = \delta_0 + \int_0^1 \sigma\widehat{b}_\theta(X_t, t)dt + \int_0^1 \sqrt{\sigma}dW_t.$$

Simulating the process targeting a Carrillo function, they found the sampler to be successful at global optimisation, surpassing the Langevin-dynamics sampling method.

However, while Dai et al.'s results on the Carrillo function are promising, there has not yet been an attempt to use a Schrödinger-Föllmer process for stochastic optimisation, let alone for machine-learning optimisation, which presents unique architectural problems.

[3] I note that this rescaling is not justified in the paper; it appears that Dai et al. may have been motivated by the rescaling that is necessary for *Langevin* dynamics, but I am not convinced that the same is necessary for Schrödinger-Föllmer processes.

Chapter 4

Path Integral Optimiser

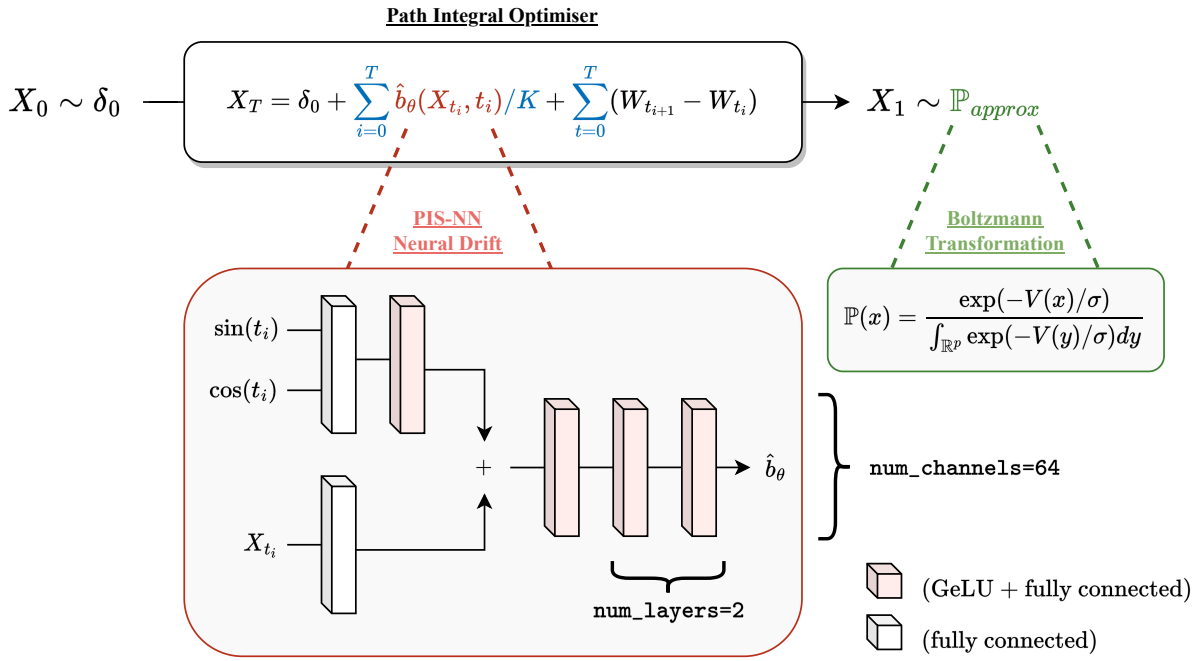

Figure 4.1: The **Path Integral Optimiser** is a neural-approximated Schrödinger-Föllmer process (**D3.2.1**) which minimises a loss function V by learning to generate samples from its Boltzmann distribution (green). The drift term is computed by a Fourier MLP (red), and the process as a whole is simulated with Euler-Maruyama discretisation (blue). With correct parameterisation of θ, σ, and K, this architecture is capable of global optimisation as proven in this paper's corollary (**C4.2.1**).

4.1 Design

To approach the problem of machine-learning optimisation, alleviating issues with Dai et al.'s monte-carlo optimiser (Section 3.3.3), this paper proposes a new Schrödinger-Föllmer optimiser: the **Path Integral Optimiser** (PIO).

PIO has the following characteristics:

- **Neural Approximation**: To account for the computationally expensive nature of high-dimensional loss functions, I use a neural approximation of the Schrödinger-Föllmer drift as per Definition **D3.2.1**, instead of a Monte-Carlo approximation as taken by Dai et al. [15]. This diminishes the "curses" associated with importance sampling (Section 3.1.2) and avoids multiple re-computation of target score.
- **PIS-Based Training**: To train the neural approximation, I adopt the drift loss proposed by Zhang et al.'s [10] PIS model as defined in **D3.2.2**, which is appropriate for sampling-based problems due to taking advantage of the availability of a target distribution $\mathbb{P}$ produced by a Boltzmann transformation (**D3.3.3**).

To further aid the model in optimisation, I suggest several experimental changes:

- **σ-Scheduling**: Under the observation that the optimisation-to-sampling (**D3.3.3**) parameter σ functions similarily to a learning rate—with lower σ producing a target distribution that is higher-variance yet closer to the global minima—there is motivation for designing a "σ schedule" that decreases during training, inspired by the convergence benefits of scheduling the learning rate [23].
  - Furthermore, it follows that it may be useful to decrease *both* σ and learning rate in lockstep, in order to match a higher-variance loss landscape with more fine-grained parameter updates.
- **Stochastic Initialisation-Matching**: Certain initial network configurations, such as Kaiming [24] or Xavier [25] initialisation, can enhance the optimisation process by balancing variance across the inputs and outputs of layers.[1] However, adjusting the PIO drift to generate this starting configuration at X_1 is a complex task, equivalent to solving a Schrödinger bridge problem targeting a Dirac distribution. To address this, I suggest a pre-training phase where in which PIO is trained with a special loss term designed to match the favorable initialisation.

4.1.1 Network Choice

For the drift-approximating network $\widehat{b}_\theta$, we take the *PIS-NN* approach put forward by Zhang et al. [10] (**D3.2.4**), in which a single time-conditioned feed-forward network $NN_\theta(X_t, t)$ approximates the entire drift. The *PIS-Grad* formulation (**D3.2.5**) would be impractical for general machine learning optimisation due to the cost of re-computing target score $\nabla \log \mathbb{P}$ at each timestep.

Specifically, we take the network $\mathrm{NN}_\theta(X_t, t)$ to be a time-conditioned fully-connected network which is augmented by a Fourier transformation of its input space, and utilises the GELU activation function $\mathrm{GELU}(x) = 0.5x\left(1 + \tanh\left[\sqrt{2/\pi}\,(x + 0.044715x^3)\right]\right)$. This is motivated by the results of Tancik et al. 2020 [26], who demonstrated that Fourier

[1] This helps maintain signal strength throughout the network, preventing issues like vanishing or exploding gradients.

transforms greatly assist fully-connected networks in learning high-frequency features from low dimensions (in our case, the time dimension). I provide an overview of the network in Figure 4.1.

4.2 Theoretical Guarantees

The aim of this section is to derive a core theorem for the usability of discretised *neural-approximated* Schrödinger-Föllmer processes (**D3.2.1**) for global optimisation, by building on the work of several recent authors [13, 15, 11]. The Path-Integral Optimiser is an instance of such a process.

I present the formal version of this new theorem (**T4.2.1**) in Section **4.2.3**, and then provide an informal corollary making use of big-$\mathcal{O}$ notation (**C4.2.1**) in Section **4.2.4**.

4.2.1 General Approach

We can build up our result by considering several Schrödinger-Föllmer processes:

- X: A continuous analytic SFP (**D3.1.1**).
- $\widehat{X}$: A continuous neural SFP (**D3.1.4**).
- $\widehat{Y}$: A discrete neural SFP (**D3.2.1**).

To evaluate $\widehat{Y}$'s proficiency at global optimisation for a given loss function V, the aim is to find a tight upper bound on the probability of $V(\widehat{Y}_1) > \tau$. In other words, to find the probability of our optimiser finding a solution with loss worse than τ.

Using the pushforward measure $P_{\widehat{Y}_1}$, one can write this as[2]

$$P_{\widehat{Y}_1}(V(X_1) > \tau).$$

To consider $\widehat{Y}$ a global optimiser, it should be possible to reduce this probabiliy to near-0 for any τ with correct choice of neural drift parameters θ as used in Definition **D3.2.1**.

4.2.2 Assumptions

To make use of the results of prior authors, we will first conglomerate their assumptions. To begin, to ensure that the SDE for X admits a unique and strong solution, Dai et al. [15] assume:

Assumption A4.2.1. *$V(x)$ is twice continuous differentiable on $\mathbb{R}$ and $V(x) = \|x\|_2^2/2$ outside a ball B_R, where B_R denotes the ball centred at origin with radius $R > 0$.*

[2]Note that this takes advantage of the fact that $V(X_1)$ and $V(\widehat{Y}_1)$ have the same event space, thus $V(X_1) > \tau$ is the same event as $V(\widehat{Y}_1) > \tau$.

Additionally, Tzen et al. [13] make several assumptions to ensure that the ratio f used in the analytic SFP definition **D3.1.1** can be efficiently approximated by a neural net:

Assumption A4.2.2. *For the analytic SFP definition (**D3.1.1**), f is differentiable, f and ∇f are L-Lipschitz (**D2.4.2**), and there exists a minima $c \in (0, 1]$ such that $f \geq c$ everywhere.*

Assumption A4.2.3. *For any $R > 0$ and $\hat{\epsilon} > 0$, there exists a feedforward neural net $\widehat{f}$ with size polynomial in $(1/\hat{\epsilon}, p, L, R)$, such that*

$$\sup_{x \in \mathrm{B}^p(R)} |f(x) - \widehat{f}(x)| \leq \hat{\epsilon} \quad \text{and} \quad \sup_{x \in \mathrm{B}^p(R)} \|\nabla f(x) - \nabla \widehat{f}(x)\| \leq \hat{\epsilon}.$$

Assumption A4.2.4. *For neural SFP $\widehat{X}$, the drift $\widehat{b}_\theta(x, t) = \widehat{v}(x, \sqrt{1-t})$ is determined by a neural net $\widehat{v} : \mathbb{R}^p \times [0, 1] \to \mathbb{R}^p$ with size polynomial in $(1/\hat{\epsilon}, p, L, c, 1/c)$, such that the activation function of each neuron is an element of the set $\{\mathrm{S}, \mathrm{S}', \mathrm{ReLU}\}$, where S is the sigmoid function.*

It is worth noting that none of our assumptions require V to be convex (**D2.4.3**) within the ball B_R; thus the global optimisation problem remains challenging.

4.2.3 Main Theorem: Neural SFP for Global Optimisation

My main theoretical contribution is as follows:

Theorem T4.2.1. *Suppose assumptions **A4.2.1** - **A4.2.4** hold. Then, for each $\epsilon \in (0, \tau), \sigma \in (0, 1), \widehat{\epsilon} \in (0, 16L^2/c^2), K \in \mathbb{Z}^+$, there exist constants $C_{\tau,\epsilon,p}$ and $C_{L'}$ (depending on τ, ϵ, p and L' respectively) such that*

$$P_{\widehat{Y}_1}(V(X_1) > \tau) \leq C_{\tau,\epsilon,p} \exp\left(-\frac{\tau - \epsilon}{\sigma}\right) + \sqrt{2\widehat{\epsilon}} + \sqrt{4L'^2(C_{L'}/K + 1/K^2)},$$

for Schrödinger Föllmer process $\widehat{Y}$ EM-discretised with step size $1/K$, whose neural drift $\widehat{b}_\theta(x, t) = \widehat{v}(x, \sqrt{1-t})$ is L'-Lipschitz in both parameters.

Remarks

Theorem **T4.2.1** may be more intuitively phrased as *the probability of discrete neural SFP $\widehat{Y}$ being a τ-global minimiser of function V:*

$$P_{\widehat{Y}_1}(V(X_1) < \tau) \geq 1 - \left(C_{\tau,\epsilon,p} \exp\left(-\frac{\tau - \epsilon}{\sigma}\right) + \sqrt{2\widehat{\epsilon}} + \sqrt{4L'^2(C_{L'}/K + 1/K^2)}\right). \quad (4.1)$$

We can trivially see that Theorem **T4.2.1** enables global optimisation by considering the fact that, by Dai et al.'s theorem, there exists a neural SFP drift for every $\widehat{\epsilon} > 0$; that by

Tzen et al.'s assumptions, we can set $\sigma \in (0, 1]$ arbitrarily; and that by EM-discretisation, we can set $1/K \in (0, 1]$ arbitrarily.

Therefore, the three parameters can tend to 0 to reach a global optimiser:

$$\begin{aligned}
&\lim_{\sigma,\widehat{\epsilon},1/K\to 0}\left[C_{\tau,\epsilon,p}\exp\left(-\frac{\tau-\epsilon}{\sigma}\right)+\sqrt{2\widehat{\epsilon}}+\sqrt{4L'^2(C_{L'}/K+1/K^2)}\right]\\
&=\lim_{\sigma\to 0}C_{\tau,\epsilon,p}\exp\left(-\frac{\tau-\epsilon}{\sigma}\right)+\lim_{\widehat{\epsilon}\to 0}\sqrt{2\widehat{\epsilon}}+\lim_{1/K\to 0}\sqrt{4L'^2(C_{L'}/K+1/K^2)}\\
&\to 0+0+0\\
&\to 0. \qquad (4.2)
\end{aligned}$$

Proof of Main Theorem T4.2.1

We can phrase our optimisation probability in terms of its distance from the continuous neural SFP terminal distribution $\widehat{X}_1$:

$$\begin{aligned}
P_{\widehat{Y}_1}(V(X_1)>\tau) &= P_{\widehat{X}_1}(V(X_1)>\tau)+P_{\widehat{Y}_1}(V(X_1)>\tau)-P_{\widehat{X}_1}(V(X_1)>\tau)\\
&\leq P_{\widehat{X}_1}(V(X_1)>\tau)+|P_{\widehat{Y}_1}(V(X_1)>\tau)-P_{\widehat{X}_1}(V(X_1)>\tau)|. \qquad (4.3)
\end{aligned}$$

Next, consider the total variation norm as given in Definition **D4.2.1**.

Definition D4.2.1. *For probability measures P and Q in measurable space $(\Omega, \mathcal{F})$, the total variation norm is defined as $2\times$ the upper bound on the distance between P and Q:*

$$\|P-Q\|_{TV}=2\sup_{A\in\mathcal{F}}|P(A)-Q(A)|.$$

Naturally, as the total variation norm is an upper bound, we have $|P(A)-Q(A)| = |Q(A)-P(A)| \leq \|Q-P\|_{TV}$ for all events $A \in \mathcal{F}$. Considering $(V(X_1) > \tau)$ as an event in the event space of the probability measures for $\widehat{X}_1$ and $\widehat{Y}_1$, we can apply the bound to Equation 4.3, giving us:

$$\begin{aligned}
&P_{\widehat{X}_1}(V(X_1)>\tau)+|P_{\widehat{Y}_1}(V(X_1)>\tau)-P_{\widehat{X}_1}(V(X_1)>\tau)|\\
&\leq P_{\widehat{X}_1}(V(X_1)>\tau)+\|P_{\widehat{X}_1}-P_{\widehat{Y}_1}\|_{TV}. \qquad (4.4)
\end{aligned}$$

We can then move from the neural approximation $\widehat{X}$ to the analytic process X using the same trick:

$$\begin{aligned}
&P_{\widehat{X}_1}(V(X_1)>\tau)+\|P_{\widehat{X}_1}-P_{\widehat{Y}_1}\|_{TV}\\
&\leq P_{X_1}(V(X_1)>\tau)+\|P_{X_1}-P_{\widehat{X}_1}\|_{TV}+\|P_{\widehat{X}_1}-P_{\widehat{Y}_1}\|_{TV}. \qquad (4.5)
\end{aligned}$$

Next, we can obtain a KL divergence representation (**D2.6.3**) by applying Pinsker's inequality, which I define in **D4.2.2**.

Definition D4.2.2. *For probability measures P and Q, Pinsker's inequality states that*

$$\|P - Q\|_{TV} \leq \sqrt{2D_{KL}(P \parallel Q)}.$$

Thus, by Pinsker's inequality, we have:

$$\begin{aligned} &P_{X_1}(V(X_1) > \tau) + \|P_{X_1} - P_{\widehat{X}_1}\|_{TV} + \|P_{\widehat{X}_1} - P_{\widehat{Y}_1}\|_{TV} \\ &\leq P_{X_1}(V(X_1) > \tau) + \sqrt{2D_{KL}(P_{X_1} \parallel P_{\widehat{X}_1})} + \sqrt{2D_{KL}(P_{\widehat{X}_1} \parallel P_{\widehat{Y}_1})}. \end{aligned} \tag{4.6}$$

To continue, we will take assumptions **A4.2.1** - **A4.2.4** and apply three results from prior authors:

- *Dai et al.* [15] (Analytic optimisation bound 3.5):
  $P_{X_1}(V(X_1) > \tau) \leq C_{\tau,\epsilon,p} \exp\left(-\frac{\tau-\epsilon}{\sigma}\right)$ under assumption **A4.2.1**.

- *Tzen et al.* [13] (Neural approximation bound 3.1):
  $D_{KL}(P_{X_1} \parallel P_{\widehat{X}_1}) \leq \widehat{\epsilon}$ under assumptions **A4.2.2** - **A4.2.4**.

- *Vargas et al.* [11] (EM-Discretisation bound A6):
  $D_{KL}(P_{\widehat{X}_1} \parallel P_{\widehat{Y}_1}) \leq 2L'^2(C_{L'}/K + 1/K^2)$ under assumptions **A4.2.2** - **A4.2.4**.

Inserting these into Equation 4.6, we obtain:

$$\begin{aligned} &P_{X_1}(V(X_1) > \tau) + \sqrt{2D_{KL}(P_{X_1} \parallel P_{\widehat{X}_1})} + \sqrt{2D_{KL}(P_{\widehat{X}_1} \parallel P_{\widehat{Y}_1})} \\ &\leq C_{\tau,\epsilon,p} \exp\left(-\frac{\tau-\epsilon}{\sigma}\right) + \sqrt{2\widehat{\epsilon}} + \sqrt{4L'^2(C_{L'}/K + 1/K^2)}. \end{aligned} \tag{4.7}$$

In summary, we have obtained a bound for $\widehat{Y}$ as an optimiser, proving Theorem **T4.2.1**:

$$P_{\widehat{Y}_1}(V(X_1) > \tau) \leq C_{\tau,\epsilon,p} \exp\left(-\frac{\tau-\epsilon}{\sigma}\right) + \sqrt{2\widehat{\epsilon}} + \sqrt{4L'^2(C_{L'}/K + 1/K^2)}. \tag{4.8}$$

4.2.4 Main Corollary: Big-$\mathcal{O}$ Parameter Bounds

As a more informal result from Theorem **T4.2.1**, I present the following corollary:

Corollary C4.2.1. *By Theorem **T4.2.1**, $\forall 0 < \delta \ll 1$, with probability at least $1 - \sqrt{\delta}$, $\widehat{Y}_1$ is a τ-global minimiser of V, i.e., $V(\widehat{Y}_1) \leq \tau + \inf V(x)$, if for the scaling factor σ, neural error $\hat{\epsilon}$, and number of iterations K, we have:*

$$\mathcal{O}\left(\frac{\tau-\epsilon}{\ln(1/\delta)}\right) \geq \sigma \quad \wedge \quad \mathcal{O}(\delta) \geq \widehat{\epsilon} \quad \wedge \quad \mathcal{O}(L'/\sqrt{\delta}) \leq K.$$

Proof of Main Corollary C4.2.1

We can examine the terms of our core result **T4.2.1** in isolation to derive bounds on parameter σ, neural error $\widehat{\epsilon}$ and step count K for optimisation accuracy. In particular, suppose we wish to achieve τ-global minimisation with probability at least $1-\sqrt{\delta}$ for arbitrary δ, meaning

$$P_{\widehat{X}_1}(V(X_1) > \tau) \leq C_{\tau,\epsilon,p} \exp\left(-\frac{\tau-\epsilon}{\sigma}\right) + \sqrt{2\widehat{\epsilon}} + \sqrt{4L'^2(C_{L'}/K + 1/K^2)} \leq \sqrt{\delta}. \quad (4.9)$$

We can achieve this by dividing $\sqrt{\delta}$ into three parts using constants $C_1 + C_2 + C_3 = 1$, such that our bound in Equation 4.9 is equivalent to solving the system:

$$C_{\tau,\epsilon,p} \exp\left(-\frac{\tau-\epsilon}{\sigma}\right) \leq C_1\sqrt{\delta} \quad \wedge \quad \sqrt{2\widehat{\epsilon}} \leq C_2\sqrt{\delta} \quad \wedge \quad \sqrt{4L'^2(C_{L'}/K + 1/K^2)} \leq C_3\sqrt{\delta}.$$

I will now address each part individually; careful inequality handling can lead each term to a big-$\mathcal{O}$ bound, proving Corollary **C4.2.1**:

1. Firstly, manipulating the σ bound (corresponding to the gap between sampling and optimisation via Definition **D3.3.3**):

$$\begin{aligned}
C_{\tau,\epsilon,p} \exp\left(-\frac{\tau-\epsilon}{\sigma}\right) &\leq C_1\sqrt{\delta} \\
\ln\left(\frac{C_1}{C_{\tau,\epsilon,p}}\sqrt{\delta}\right) &\geq -\frac{\tau-\epsilon}{\sigma} \\
\frac{1}{2}\ln\left(\frac{C_1^2}{C_{\tau,\epsilon,p}^2}\delta\right)\sigma &\geq -(\tau-\epsilon) \\
-\frac{2(\tau-\epsilon)}{\ln\left(\frac{C_1^2}{C_{\tau,\epsilon,p}^2}\delta\right)} &\geq \sigma \\
\mathcal{O}\left(\frac{\tau-\epsilon}{\ln(1/\delta)}\right) &\geq \sigma.
\end{aligned}$$

2. Secondly, manipulating the $\widehat{\epsilon}$ bound (corresponding to the gap between the analytical SFP and its neural approximation):

$$\begin{aligned}
\sqrt{2\widehat{\epsilon}} &\leq C_2\sqrt{\delta} \\
\widehat{\epsilon} &\leq C_2^2\delta/2 \\
\mathcal{O}(\delta) &\geq \widehat{\epsilon}.
\end{aligned}$$

3. Finally, manipulating the K bound (corresponding to the gap between continuous and EM-discretised neural SFP):

$$\begin{aligned}
\sqrt{4L'^2(C_{L'}/K + 1/K^2)} &\leq C_3\sqrt{\delta} \\
(4L'^2)(C_{L'}/K + 1/K^2) &\leq C_3^2\delta \\
(4L'^2)(KC_{L'} + 1) &\leq K^2C_3^2\delta \\
(4L'^2)(C_{L'} + 1) &\leq K^2C_3^2\delta \\
\frac{(C_{L'}+1)(4L'^2)}{C_3^2\delta} &\leq K^2 \\
\mathcal{O}(L'/\sqrt{\delta}) &\leq K.
\end{aligned}$$

Chapter 5

Empirical Study

Evaluating a general optimiser's performance in a scientific manner poses considerable challenges. We are effectively asking for the expected performance over all ML-worthy tasks, target networks, and optimiser configurations, and the standards for such experiments tend to differ across optimisation papers.

Importantly, one should account for the fact that a practical application of the optimiser would involve some degree of hyperparameter tuning, and a fair comparison should offer the same tuning approach and budget across optimisers. In 2020, Choi et al. found that minor alterations to hyperparameter search space can **reverse classic optimisation results**, emphasising that hyperparameter relationships should not be neglected [27].

Thus, in this paper, I establish a framework through which to evaluate optimisers by describing the notation of a *task environment* in **D5.1.1**. I focus on three key task environments inspired by recent optimisation-related papers, outlined in Section 5.1.

For the comparison to be statistically sound, we also need to account for inherent stochasticity in both the diffusion model and dataset by re-running the optimisation over multiple seeds.

5.1 Tasks

In **D3.3.2**, I defined the optimisation problem as that of finding $x_* \in \arg\min_{x \in \mathbb{R}^p} f(x)$. To contextualise this problem to that of machine-learning optimisation, while keeping our notation distinct from the training of neural Schrödinger Föllmer models, I introduce the notion of a *task environment*:

Definition D5.1.1. *A **task environment (T.E.)** is a triple $(V, \mathscr{F}_\phi, \mathcal{D})$ consisting of:*

- ***Task loss** $V : \mathbb{R}^b \times \mathbb{R}^b \to \mathbb{R}$,*
- ***Task-solving model** $\mathscr{F}_\phi : \mathbb{R}^a \to \mathbb{R}^b$ with parameters $\phi \in \mathbb{R}^p$,*

- ***Task dataset*** $\mathcal{D} = \{(x_i, y_i) \mid x_i \in \mathbb{R}^a, y_i \in \mathbb{R}^b\}_{i=1}^N$.

To access elements of $\mathcal{D}$, one can use the notation $\mathcal{D}_i^0 = x_i$ and $\mathcal{D}_i^1 = y_i$ for datapoints and labels respectively.

The optimisation problem can then be recast as finding the model parameters ϕ:

Definition D5.1.2. *Given task environment $(V, \mathscr{F}_\phi, \mathcal{D})$, the ML optimisation problem is that of finding:*

$$\phi_* \in \arg\min_{\phi \in \mathbb{R}^p} f(\phi) = \arg\min_{\phi \in \mathbb{R}^p} \frac{1}{N} \sum_{i=1}^{N} V\left(\mathcal{D}_i^1, \mathscr{F}_\phi(\mathcal{D}_i^0)\right).$$

Here, the function $f : \mathbb{R}^p \to \mathbb{R}$ is the average loss over the dataset $\mathcal{D}$ incurred by the task-solving model $\mathscr{F}_\phi$, and V is a suitable task loss function that measures the discrepancy between the model's predictions and the true outputs. The optimisation objective is thus to find the model parameters ϕ that minimises this average loss.

5.1.1 Summary

In this evaluation, we will make use of three *task environments* inspired by recent optimisation-related papers. The goal here is to evaluate the optimiser at a variety of dimension sizes, for tasks that have a practical or theoretical basis. The selected tasks are summarised in Table 5.1; a more formal definition of the three, including motivation for their choice of model and loss function, can be found in Appendix B.

T.E.	Model $\mathscr{F}$	Parameters $\lvert\phi\rvert$	Connection
Carrillo (B.0.3)	ϕ	2	Dai et al. [15]
Moons (B.0.1)	MLP	41	Agnihotri et al. [28]
MNIST (B.0.2)	MLP	15,910	Andrychowicz et al. [29]

Table 5.1: Task environments considered in this study.

In brief, the **Carrillo** task is a low-dimension optimisation problem designed to test machine-learning optimisers; it is made challenging by possessing an infinity of local minima. The **Moons** task is a 2D binary classification problem from the `scikit` library, and **MNIST** is a classic ML problem involving the classification of 28×28px digits.

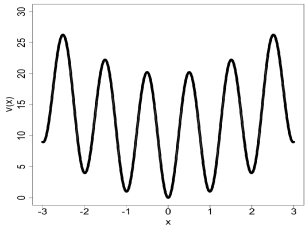

Figure 5.1: The Carrillo function as defined in **DB.0.4**, courtesy Dai et al. [15].

5.2 Results

5.2.1 Hyperparameter Sweeps

As mentioned at the start of this chapter, optimisers are considerably challenging to compare fairly; in 2020, Choi et al. found that even minor changes to the specifications of a hyperparameter sweep can reverse classic optimisation results [27]. For example, they found a tendency for researchers to underestimate Adam's performance by not including its full breadth of parameters in a sweep.

To that end, I strive to evaluate the **Path Integral Optimiser** robustly by performing two differently-configured hyperparameter sweeps and comparing their results individually.[1] This should also offer rare insights into the sensitivity of other popular optimisation algorithms.

In particular, I consider **two distinct sweeps**, each performed using the `BayesOpt` Bayesian optimisation algorithm included in Meta's Nevergrad library [30]. While these details are often relegated to the appendices in machine learning papers, they are relevant in our context of optimisation:

- **Narrow ×32**: a 32-run sweep, each run lasting 32 training steps, over the following hyperparameters:
  - PIO: `lr, grad_clip_val, batch_size, sigma`.
  - Others: `lr, grad_clip_val, batch_size`.

- **Wide ×64**: a 64-run sweep, each run lasting 64 training steps and making use of an LR-scheduling algorithm that drops LR by 50% on plateaus, and scales PIO's σ parameter to match. This is performed over the following hyperparameters:
  - PIO: `lr, grad_clip_val, batch_size, batch_laps, sigma, num_layers, K`.
  - Adam: `lr, grad_clip_val, batch_size, batch_laps, beta_1, beta_2`.
  - Others: `lr, grad_clip_val, batch_size, batch_laps`.

[1] As an aside, it is worth noting that a key advantage of a PIO compared to other optimisers is its ability to simulate multiple alternative yet viable parameter configurations in the form of stochastic *trajectories*. In hyperparameter sweeping and evaluation, we take the trajectory that produces the lowest validation loss.

The idea with **Wide ×64** is that we aim to fully exploit each optimiser by including a wide variety of parameters. The unusual `batch_laps` parameter entails re-using the same mini-batch over multiple consecutive training steps, which may improve numerical stability. Figure 5.2 exemplifies parameters for the **Narrow ×32** sweep.

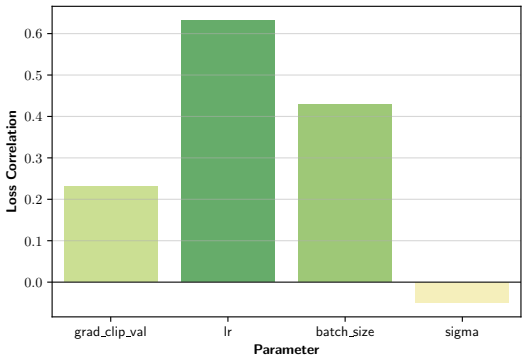

Figure 5.2: The correlation between PIO's hyperparameters and its validation loss at the Moons task for the ×32 sweep.

5.2.2 Core Results

Narrow ×32 sweep

×32 Optimiser	Carrillo	Moons	MNIST
Adagrad [3]	*0.549* ± 0.153	0.184 ± 0.036	**0.425** ± 0.031
Adam [2]	0.597 ± 0.536	**0.057** ± 0.047	0.476 ± 0.030
SGD [1]	**0.491** ± 0.125	0.356 ± 0.019	*0.427* ± 0.034
PIO (Ours)	1.458 ± 0.882	*0.129* ± 0.121	2.464 ± 0.026

Table 5.2: Final test loss $V(x)$ for various optimisers, tuned with a **Narrow ×32** hyperparameter sweep of 32 runs, at learning the *Carrillo*, *MNIST*, and *Moons* task environments as described in Section 5.1. Each optimiser was allowed to train their final task-solving model $\mathscr{F}$ for 100 steps. **Bold** denotes the best result in a column and *italics* the second-best. We find that PIO massively underperforms in these conditions, experiencing exceptionally high variance for Carrillo and MNIST tasks.

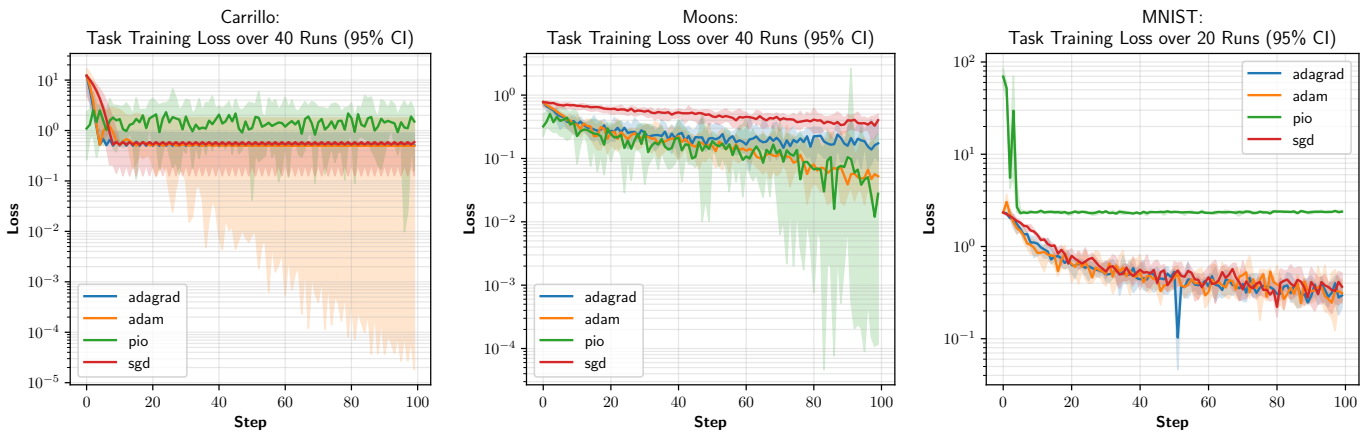

Figure 5.3: Training loss $V(x)$ for various optimisers, tuned with a **Narrow ×32** hyperparameter sweep of 32 runs, at learning three different task environments as described in Section 5.1. Confidence intervals are computed by re-running the best-found configuration across multiple seeds.

Figure 5.3 demonstrates the training loss for the best-found configurations, averaged over a number of runs, reduced for MNIST due to its computational cost.

LEFT: The Carrillo task. PIO appears stuck in a local minima, and Adam the only model seemingly able to converge to the global minima—though it does not do so reliably. Examining Figure 5.4, we see that PIO indeed became stuck at local minima.

MID: The Moons task. PIO performs impressively, beating Adam on training loss but narrowly missing out in test loss as per Table 5.2. While its variance is high, across seeds, there is a skew towards lower loss, which might imply the effectiveness of an *ensembling* technique. Figure 5.5 shows that PIO learned a basic linear boundary strategy to tackle the problem.

RIGHT: The MNIST task. PIO completely fails to learn in this task, rapidly becoming stuck at a loss of around 2.5. Figure 5.6 confirms that the optimiser only learned the basic strategy of producing the same number for each sample.

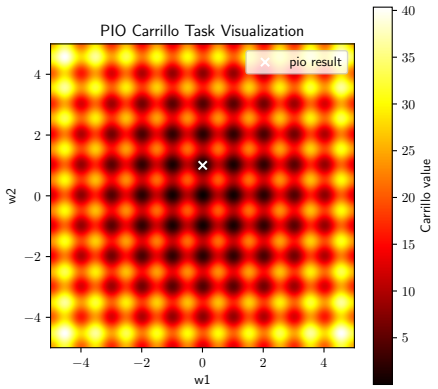

Figure 5.4: The Carrillo point found by the best PIO run in its ×32 hyperparameter sweep; the optimiser appears stuck in a local minima.

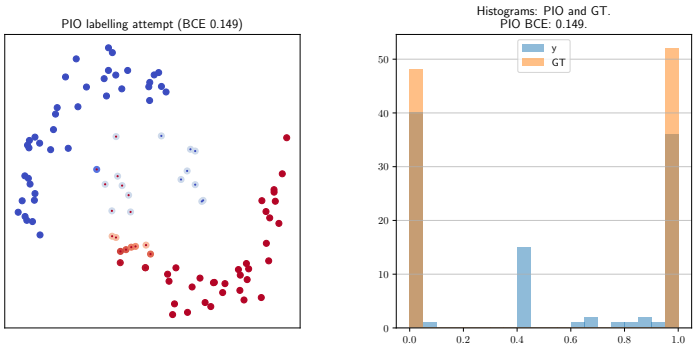

Figure 5.5: On the left: PIO's final attempt at the Moons labelling task after ×32 sweeping, with ground truth marked with a small dot in the center of each circle. On the right: a histogram of PIO's labels compared to the ground truth, demonstrating room for improvement.

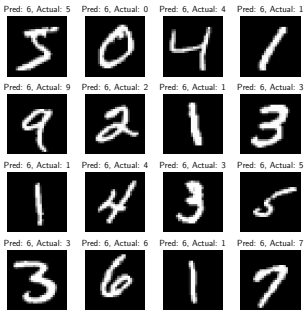

Figure 5.6: The MNIST labels predicted by PIO's best run in its ×32 hyperparameter sweep; every label is the same, implying that the optimiser is stuck in a very early local minima.

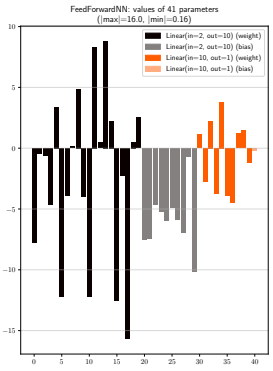

Figure 5.7: The parameters ϕ found by PIO for the Moons problem. All biases are negative, which may imply underutilisation.

Wide ×64 sweep

×64 Optimiser	Carrillo	Moons	MNIST
Adagrad [3]	0.647 ± 0.448	0.193 ± 0.038	*0.347* ± 0.020
Adam [2]	*0.498* ± 0.000	**0.022** ± 0.010	**0.346** ± 0.020
SGD [1]	0.553 ± 0.149	0.428 ± 0.026	0.385 ± 0.011
PIO (Ours)	**0.404** ± 0.417	*0.032* ± 0.049	2.424 ± 0.045

Table 5.3: Best loss $V(x)$ for various optimisers, tuned with a **Wide ×64** hyperparameter sweep of 64 runs, at learning the *Carrillo*, *Moons*, and *MNIST* task environments as described in Section 5.1. Each optimiser was allowed to train their final task-solving model $\mathscr{F}$ for 100 steps. **Bold** denotes the best result in a column and *italics* the second-best. We find that PIO's performance appears **significantly** stronger in these conditions compared to the **Narrow ×32** sweep, offering compelling results for Carrillo and Moons.

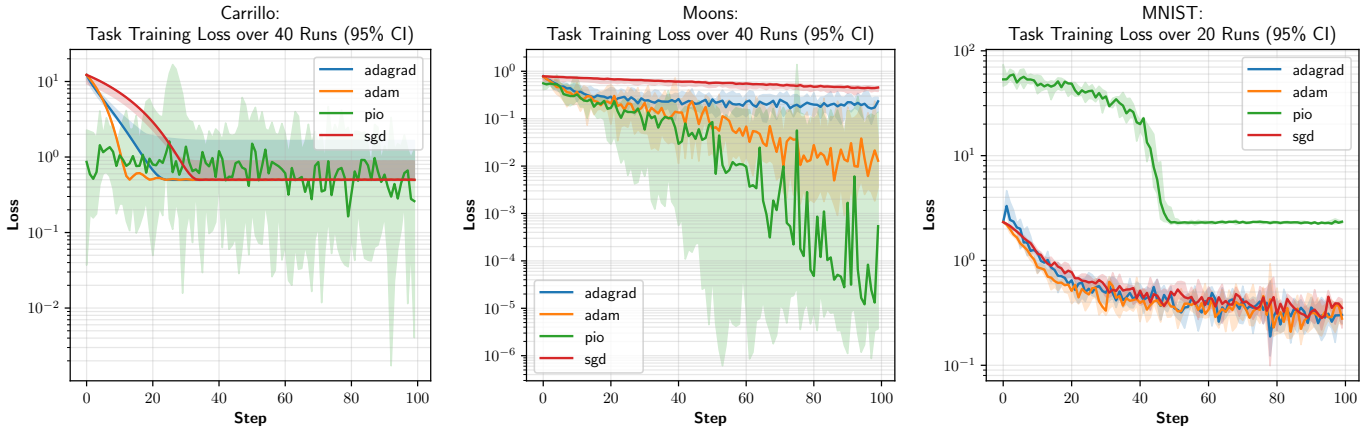

Figure 5.8: Training loss $V(x)$ for various optimisers, tuned with a **Wide ×64** hyperparameter sweep of 64 runs, at learning three different task environments as described in Section 5.1. Confidence intervals are computed by re-running the best-found configuration across multiple seeds. Notably, PIO still fails to learn the MNIST task.

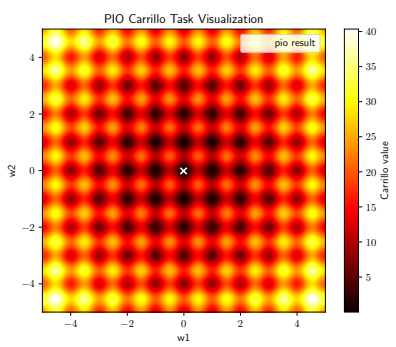

Figure 5.9: The Carrillo point found by the best PIO run in its ×64 hyperparameter sweep; the optimiser successfully approached a global minima, unlike in the ×32 case.

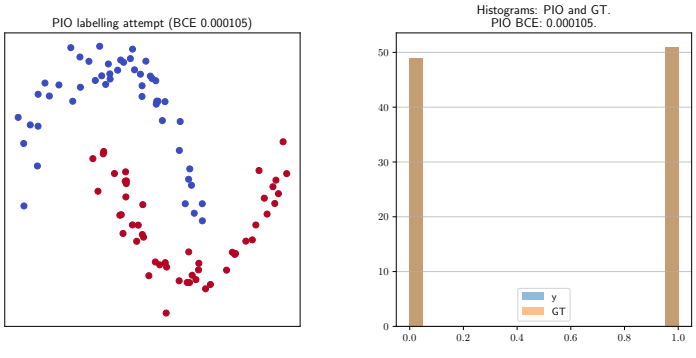

Figure 5.10: On the left: PIO's final attempt at the Moons labelling task after ×64 sweeping, with ground truth marked with a small dot in the center of each circle. On the right: a histogram of PIO's labels compared to the ground truth, demonstrating a perfect (in rounding error) result.

Chapter 6

Summary and conclusions

To summarise, this project presented a first venture into the application of *diffusion models* for optimisation, adapting the neural Schrödinger-Föllmer sampling described by Zhang et al. [10] to a new domain. By use of the Boltzmann transformation (**D3.3.3**), careful construction of the neural drift term as a Fourier MLP (Figure 4.1), and establishing the concept of a *task environment* (**D5.1.1**), I was apply to use PIO as an optimiser and glean insightful, albeit mixed, results.

Continuing the recent work in Föllmer-like diffusion models by the likes of Dai et al. (2021) [15], Zhang et al. (2021) [10], and Vargas et al. (2023) [16], I also derived the first theoretical guarantees for discretized neural Schrödinger-Föllmer processes at the task of optimization in Section 4.2. In Corollary **C4.2.1**, I put these guarantees in terms of big-$\mathcal{O}$ bounds, showing that—for example—the optimisation parameter σ should decrease logarithmically with δ to achieve a probability of global optimisation over $1 - \sqrt{\delta}$.

In Section 5.2, in evaluating PIO I re-affirmed Choi et al.'s [27] 2019 observation that comparisons of optimisers can be distorted by choice of hyperparameter sweep (see PIO's Carrillo performance reversing from Table 5.2 to Table 5.3). PIO as a whole exhibited promising performance on the relatively low-dimension tasks of Moons and Carrillo, but was unable to learn in higher dimensions for MNIST (see Figure 5.8).

For future work, I suggest:

- Improving PIO's starting point by matching an established initialization, such as Kaiming [24] or Xavier [25], in a pre-training phase.
- Taking advantage of PIO's ability to stochastically produce alternative trajectories by forming an ensemble model [31] from its output.
- Benchmarking other ML-based optimisers, such as Ha et al.'s Hypernetwork [12] and Andrychowicz et al.'s "Learning to Learn" meta-learner [29] at the same tasks.
    - In particular, to reproduce results across *multiple* hyperparameter sweeps, each with different time and space constraints—a condition often overlooked.

Bibliography

[1] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[2] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[3] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[4] Yurii Evgen'evich Nesterov. "A method of solving a convex programming problem with convergence rate O\bigl(k^2\bigr)". In: *Doklady Akademii Nauk*. Vol. 269. 3. Russian Academy of Sciences. 1983, pp. 543–547.

[5] Paul Tseng. "An incremental gradient (-projection) method with momentum term and adaptive stepsize rule". In: *SIAM Journal on Optimization* 8.2 (1998), pp. 506–531.

[6] Nitish Shirish Keskar and Richard Socher. "Improving generalization performance by switching from adam to sgd". In: *arXiv preprint arXiv:1712.07628* (2017).

[7] Soham De, Anirbit Mukherjee, and Enayat Ullah. "Convergence guarantees for RMSProp and ADAM in non-convex optimization and an empirical comparison to Nesterov acceleration". In: *arXiv preprint arXiv:1807.06766* (2018).

[8] Xiaoyu Li and Francesco Orabona. "On the convergence of stochastic gradient descent with adaptive stepsizes". In: *The 22nd international conference on artificial intelligence and statistics*. PMLR. 2019, pp. 983–992.

[9] Aditya Ramesh et al. "Hierarchical text-conditional image generation with clip latents". In: *arXiv preprint arXiv:2204.06125* (2022).

[10] Qinsheng Zhang and Yongxin Chen. "Path Integral Sampler: a stochastic control approach for sampling". In: *arXiv preprint arXiv:2111.15141* (2021).

[11] Francisco Vargas et al. "Bayesian Learning via Neural Schrödinger-Föllmer Flows". In: *arXiv preprint arXiv:2111.10510* (2021).

[12] David Ha, Andrew Dai, and Quoc V Le. "Hypernetworks". In: *arXiv preprint arXiv:1609.09106* (2016).

[13] Belinda Tzen and Maxim Raginsky. "Theoretical guarantees for sampling and inference in generative models with latent diffusions". In: *Conference on Learning Theory*. PMLR. 2019, pp. 3084–3114.

[14] Jian Huang et al. “Schrödinger-Föllmer sampler: sampling without ergodicity”. In: *arXiv preprint arXiv:2106.10880* (2021).

[15] Yin Dai et al. “Global Optimization via Schrödinger-Föllmer Diffusion”. In: *arXiv preprint arXiv:2111.00402* (2021).

[16] Francisco Vargas, Will Grathwohl, and Arnaud Doucet. “Denoising Diffusion Samplers”. In: *arXiv preprint arXiv:2302.13834* (2023).

[17] Hans Föllmer. “An entropy approach to the time reversal of diffusion processes”. In: *Stochastic Differential Systems Filtering and Control: Proceedings of the IFIP-WG 7/1 Working Conference Marseille-Luminy, France, March 12–17, 1984*. Springer. 2005, pp. 156–163.

[18] Sergios Agapiou et al. “Importance sampling: Intrinsic dimension and computational cost”. In: *Statistical Science* (2017), pp. 405–431.

[19] Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*. Vol. 2. Springer, 1999.

[20] Maithra Raghu et al. “On the expressive power of deep neural networks”. In: *international conference on machine learning*. PMLR. 2017, pp. 2847–2854.

[21] Zengyi Li, Yubei Chen, and Friedrich T Sommer. “A neural network mcmc sampler that maximizes proposal entropy”. In: *Entropy* 23.3 (2021), p. 269.

[22] Matthew D Hoffman, Andrew Gelman, et al. “The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo.” In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1593–1623.

[23] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures”. In: *Neural Networks: Tricks of the Trade: Second Edition* (2012), pp. 437–478.

[24] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[25] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*. Ed. by Yee Whye Teh and D. Mike Titterington. Vol. 9. JMLR Proceedings. JMLR.org, 2010, pp. 249–256. URL: `http://proceedings.mlr.press/v9/glorot10a.html`.

[26] Matthew Tancik et al. “Fourier features let networks learn high frequency functions in low dimensional domains”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7537–7547.

[27] Dami Choi et al. “On empirical comparisons of optimizers for deep learning”. In: *arXiv preprint arXiv:1910.05446* (2019).

[28] Apoorv Agnihotri and Nipun Batra. “Exploring bayesian optimization”. In: *Distill* 5.5 (2020), e26.

[29] Marcin Andrychowicz et al. “Learning to learn by gradient descent by gradient descent”. In: *Advances in neural information processing systems* 29 (2016).

[30] J. Rapin and O. Teytaud. *Nevergrad - A gradient-free optimization platform*. `https://GitHub.com/FacebookResearch/Nevergrad`. 2018.

[31] David Opitz and Richard Maclin. “Popular ensemble methods: An empirical study”. In: *Journal of artificial intelligence research* 11 (1999), pp. 169–198.

[32] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[33] Jenny Balfer, Jürgen Bajorath, and Martin Vogt. “Compound Classification Using the scikit-learn Library”. In: *Tutorials in Chemoinformatics* (2017), pp. 223–239.

[34] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[35] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998).

[36] José A Carrillo et al. “A consensus-based global optimization method for high dimensional machine learning problems”. In: *ESAIM: Control, Optimisation and Calculus of Variations* 27 (2021), S5.

Appendix A

Measure-Theoretic Example

Example: Two Dice

To calculate the expected sum of values of a pair of dice, one can start by defining a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ to represent the outcomes of the dice:

$$\begin{aligned}
&\Omega = \{(i,j) \mid i,j \in \{1,2,3,4,5,6\}\},\\
&\mathcal{F} = \mathcal{B}(\Omega),\\
&\mathbb{P}(\{(i,j)\}) = \frac{1}{36} \text{ for all } i,j \in \Omega,
\end{aligned}$$

and then defining a random variable $X : (\Omega, \mathcal{F}) \to (S, \Sigma)$ to represent the sum of the values of the two dice:

$$X(i,j) = i + j, \text{ for all } i,j \in 1,2,3,4,5,6.$$

This random variable now has a distribution that can be accessed with $\mathbb{P}_X$, which we call the *pushforward measure* (**D2.2.5**):

$$\mathbb{P}_X(n) = \mathbb{P}(X^{-1}(n)),$$

which finally enables the calculation of X's expectation in terms of $\mathbb{P}$:

$$\mathbb{E}(X) = \sum_{n \in S} n\mathbb{P}_X(n) = \sum_{n \in S} n\mathbb{P}(X^{-1}(n)) = \sum_{(i,j) \in \Omega} X(i,j)\mathbb{P}(i,j) = 7.$$

Appendix B

Task Environments

B.0.1 Moons T.E.

The **Moons** task is a binary classification problem based on a synthetic dataset supplied by the scikit-learn (sklearn) Python library [32]. This dataset encompasses two interleaving half circles, effectively generating two "moon" shapes to distinguish; its difficulty and interpretability make it popular in evaluating clustering and classification algorithms [33].

Dataset ($\mathcal{D}$)

Moons are generated with `sklearn.datasets.make_moons`. This includes a configurable Gaussian noise term e such that the instances from each class are scattered and not perfectly semicircular. In mathematical terms, for an instance A belonging to the first class (moon), and B belonging to the second class, their respective coordinates can be described as:

$$A = (r\cos(\theta), r\sin(\theta)) + e,$$
$$B = (r\cos(\theta + \pi), r\sin(\theta + \pi)) + e,$$

where θ is an angle uniformly sampled from $[0, \pi]$, r is the radius, and e is a Gaussian noise term with zero mean and a specified standard deviation. The moons task thus takes the form of dataset

$$\mathcal{D} = \{(x_i, y_i) \mid x_i \in \mathbb{R}^2, y_i \in \{0, 1\}\}_{i=1}^{N},$$

where x_i are the 2-dimensional data points and y_i is the binary label indicating membership of x_i to either A or B.

Loss Function (V)

An appropriate loss term for this task is *Binary Cross Entropy* (BCE), which is suited for learning binary class labels. In the context of a task environment (**D5.1.1**), we can

simply take $V = BCE$.

Definition DB.0.1. *Given the true label $y \in \{0, 1\}$, and that $y' \in [0, 1]$ is the predicted probability for the positive class (1), the **Binary Cross Entropy (BCE) loss** is defined:*

$$BCE(y, y') = -\log(P_{Ber}(y; y')) = -y \log(y') - (1 - y) \log(1 - y'),$$

where P_{Ber} is the Bernoulli likelihood function.

The BCE loss function is an appropriate choice because the predicted moon probability y' can be interpreted as the success probability in a Bernoulli trial; by minimising the BCE loss, we are effectively performing maximum likelihood estimation under the Bernoulli distribution assumption, which is statistically sound and interpretable.

Model ($\mathscr{F}$)

The Moons task can be efficiently solved with a small fully-connected neural network with at least one hidden layer.

A simple linear model would not be sufficient, as the two moon classes are not linearly separable; some non-linearity, such as in the form of a function $\mathrm{ReLU}(x) = \max(0, x)$ [34], must be introduced in order to find the optimal non-linear decision boundary. I found that a 10-neuron ReLU'd hidden layer is sufficient for optimisation by stochastic gradient descent.

As is standard for the binary classification tasks, we can use a sigmoidal activation function $\mathrm{S}(x) = 1/(1 + e^{-x})$ on the final layer to ensure the output is a probability and thus compatible with Binary Cross-Entropy loss (**DB.0.1**).

In total, using the `torch.nn` notation, we have a 41-parameter (30 weights + 11 biases) architecture:

`Sequential(Linear(2, 10), ReLU(), Linear(10, 1), Sigmoid()).`

B.0.2 MNIST T.E.

The **MNIST** task is a classic multiclass classification problem based on a dataset of handwritten digits, developed by Yann LeCun in 1998 [35]. This dataset is one of the most widely used in the field of machine learning, particularly for benchmarking classification algorithms.

Dataset ($\mathcal{D}$)

The MNIST dataset consists of 70,000 size-normalised grayscale images of handwritten digits, each of size 28x28 pixels. Each image is labelled with the digit it represents, from 0 to 9. In the context of a task environment (**D5.1.1**), the MNIST task takes the form

of a dataset

$$\mathcal{D} = \{(x_i, y_i) \mid x_i \in \mathbb{R}^{784}, y_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\}_{i=1}^{N},$$

where x_i are the 784-dimensional data points (flattened 28x28 pixel images) and y_i is the label indicating the digit that x_i represents.

In practice, we tend to *one-hot-encode* y_i to:

$$y_i \in \{0, 1\}^{10},$$

where each y_i is a 10-dimensional vector such that all components are zero except the k-th component, which is 1, where k is the digit that x_i represents.

Loss Function (V)

An appropriate loss term for this task, as a multiclass labelling task, is *Categorical Cross Entropy* (CCE). Like with BCE, in the context of a task environment (**D5.1.1**), we can simply take $V = CCE$.

Definition DB.0.2. *Given the true label $y = (y_1, \ldots, y_K)$, where $y_k = 1$ and $y_i = 0$ for all $i \neq k$, and the predicted probabilities $y' = (y'_1, \ldots, y'_K)$, the* ***Categorical Cross Entropy (CCE)*** *for K classes is defined as $CCE(y, y') = -\sum_{i=1}^{K} y_i \log(y'_i)$.*

Intuitively, this loss function strongly penalises predictions that are *confidently wrong*, since the log function approaches negative infinity as its argument approaches zero. In practice, to avoid numerical underflow after extracting probabilities using softmax, one tends to insert a *logarithmic* layer at the output of the model $\mathscr{F}$; in this case, we can simplify the CCE to:

Definition DB.0.3. *Given the true label $y = (y_1, \ldots, y_K)$, where $y_k = 1$ and $y_i = 0$ for all $i \neq k$, and the predicted log-probabilities $y' = (y'_1, \ldots, y'_K)$, the* ***Negative Log-Likelihood (NLL)*** *for K classes is defined as $NLL(y, y') = -\sum_{i=1}^{K} y_i y'_i$.*

Model ($\mathscr{F}$)

Being a visual task, a convolutional neural network would be well-suited to solving MNIST. However, due to the small nature of the problem, and motivated by consistency, we adopt the same architecture as Andrychowicz et al.'s *Learning to learn by gradient descent by gradient descent* [29], in which a 20-neuron fully connected neural network was employed.

Following the same notation as Section B.0.1, we define this 15910-parameter (15880 weights + 30 biases) model as:

`Sequential(Linear(784, 20), ReLU(), Linear(20, 10), LogSoftmax()).`

B.0.3 Carrillo T.E.

The **Carrillo** function refers to a non-convex (**D2.4.3**) smooth function (**D2.4.1**) used by Carrillo et al. to evaluate global optimisation techniques [36].

The idea is that this function serves as a cheaply-computed analogue to the objective of high-dimensional ML optimisation problems due to its challenging properties: importantly, while the function has only one global minima—of value C at point B—it has an infinity of *local* minima on all sides of B.

Definition DB.0.4. *For constants $B \in \mathbb{R}^d, C \in \mathbb{R}, d \in \mathbb{Z}^+$, we define the Carrillo function* $\text{Carrillo} : \mathbb{R}^d \to \mathbb{R}$ *as:*

$$\text{Carrillo}(y) = \frac{1}{d} \sum_{i=1}^{d} \left[(y_i - B)^2 - 10 \cos\left(2\pi \left(y_i - B\right)\right) + 10 \right] + C.$$

Task environment $(V, \mathscr{F}_\phi, \mathcal{D})$

The Carrillo definition **DB.0.4** allows one to evaluate the proficiency of an optimiser at the ML optimisation problem using the simple optimisation goal $y_* \in \arg\min_{y \in \mathbb{R}^d} \text{Carrillo}(y)$ as defined in **D3.3.1**, rather than within a *task environment* structure.

However, for the sake of defining a unified evaluation framework in both software and theory, it will still be useful to view the Carrillo optimisation problem through the lens of a task environment $(V, \mathscr{F}_\phi, \mathcal{D})$ as described in **D5.1.1**. This can be achieved with the contrivance of an empty dataset and a parameter-reflecting task model, producing the task environment:

$$(V(y, y') = \text{Carrillo}(y'), \quad \mathscr{F}_\phi(x) = \phi, \quad \mathcal{D} = \{(0,0)\}^N).$$

To see how solving this task environment, which is now an *ML optimisation problem* as defined in **D5.1.2**, is equivalent to the optimisation goal $y_* \in \arg\min_{y \in \mathbb{R}^d} \text{Carrillo}(y)$, consider the simple derivation:

$$\begin{aligned}
&\arg\min_{\phi \in \mathbb{R}^p} \frac{1}{N} \sum_{i=1}^{N} V\left(\mathcal{D}_i^1, \mathscr{F}_\phi(\mathcal{D}_i^0)\right) && \text{from } \mathbf{D5.1.2} \\
=\ &\arg\min_{\phi \in \mathbb{R}^p} \frac{1}{N} \sum_{i=1}^{N} V\left(0, \phi\right) && \text{by } \mathcal{D}_i^1 = 0 \text{ and } \mathscr{F}_\phi(x) = \phi \\
=\ &\arg\min_{\phi \in \mathbb{R}^p} V\left(0, \phi\right) && \text{by } \frac{1}{N} \sum_{i=1}^{N} c = c \text{ for const } c \\
=\ &\arg\min_{\phi \in \mathbb{R}^p} \text{Carrillo}\left(\phi\right) && \text{by } V(y, y') = \text{Carrillo}(y').
\end{aligned}$$

In practice, I initialize each optimiser with the starting parameters $\phi \sim \mathcal{N}(0, 1)$.