Machine Learning for Functional Connectomics in *Caenorhabditis elegans*

Andrew Warrington

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary Term, 2021

For Grandma

Abstract

Santiago Ramón y Cajal first traced the microscopic intricacies of individual neurons in the late 19th century. His work revolutionised neuroscience, and earned him the moniker "the father of modern neuroscience." Cajal was a major contributor to the newly established *neuron doctrine*, outlining that even the most complex organism behaviours are simply the result of the interactions between networks of small, discrete units, called neurons. Since then, neuroscience research has measured, codified, and modelled these neural interactions in progressively more detail.

However, these neural models are rarely applied at whole-brain scales, and are even less frequently used as mechanistic models of neural activity in which Bayesian inference is performed. This offers an exciting and truly ground-breaking opportunity. Application of Bayesian inference to whole-connectome neural models would allow previously unobservable physiological states and parameter values to be inferred from readily observable data at an unprecedented scale. It would also allow interpretable neural models to be refined, tested and even learned directly from observed data without manual input. Finally, tuned whole-brain simulators promise *in silico* neuroscience experiments, where hypotheses can be investigated without the need for biological specimens. Such experiments are not hindered by the logistical requirements of obtaining measurements *in vivo*, and can be easily, cheaply and reliably reproduced.

In this thesis we develop tools to perform this estimation. Specifically, we study performing this estimation in the nematode roundworm *Caenorhabditis elegans*, the only organism for which the entire connectome has been fully mapped. At its core, the work we present is the first attempt to infer, at whole-connectome scale, neural states and parameters from limited observations. We introduce the machine learning and computational neuroscience tools required to make this tractable, and explore extensions arising directly from this work. We conclude by suggesting further extensions of this work that are within reach, and discussing more wide-reaching ideas and implications of this work.

Acknowledgements

A PhD is remarkably similar to an iceberg. The tip of an iceberg contributes just ten percent of the total mass, with the full extent of the supporting structure hidden from view. Similarly, this thesis is just the tip, and hides the immeasurable time and effort of those who have supported me during its conception and completion.

Foremost I thank my parents and my grandma. Without hesitation or question you gave me your time, energy and support: from reading incomprehensible and esoteric paper drafts, to giving me the confidence to venture halfway around the world, to simply listening when I needed to talk. For this, and for so much more, I am eternally grateful.

I also extend a debt of immense gratitude to my supervisor, Frank Wood. I would never have guessed, when I dropped into an undergraduate project you were running, we would end up in the Pacific Northwest, studying a worm, more than six years later. Even when my self-confidence was at rock bottom, your confidence in me never wavered. It's been a real blast so far, and we certainly aren't done yet.

I must also thank my collaborators: Arthur Spencer, Ryan Fayyazi, Wilder Lavington, Saeid Naderiparizi, Neil Dhir, Adam Ścibior and Tom Rainforth, whose creativity and sheer effort underpin much of this thesis. I also thank M. Pawan Kumar, who volunteered his time and services in a moment of need. More generally, I thank my colleagues, both in Oxford and UBC, for their invaluable support, guidance and friendship along the way.

Finally, I thank those outside of research that I have had the pleasure of sharing the last four years with. You are too numerous to name individually, but PlayToFeet, HistoryBeers, Doctors, CoffeeClub, and the three incarnations of Jeune Street covers almost everyone. Coffee breaks, bleary-eyed breakfasts after all-nighters, home-made packed lunches delivered to me during deadlines, beers after deadlines, days skiing and international adventures are amongst my most cherished memories. Thank you all, you are truly friends for life.

Contents

1 Introduction

I used to live in a high-rise apartment block. Occasionally, another tenant would be waiting for the elevator with their pet dog. I would ponder while commuting: "how do dogs rationalise elevators?" Dogs have a sophisticated notion of object permanence and are capable of constructing highly accurate world maps. They show confusion, even distress, when either their world map or notion of object permanence is challenged or violated.[1] Despite this, each dog would calmly walk from the lobby into the windowless elevator, wait in the carriage, and when the doors reopened, would happily trot out of the elevator on the appropriate level. Despite the abrupt and gross violation of object permanence, the dog would remain calm and content throughout. Therefore, it is reasonable to conclude that the dog has a cognitive process sophisticated enough to rationalise this violation. As a scientist, the obvious question is therefore: "how can we measure, quantify and understand this cognitive process?" Answering questions of this nature is arguably *the* primary goal of neuroscience.

A foundational principle of neuroscience is that all behaviours are simply the result of the predictable and reproducible interactions within the vast web of interconnected neurons, hormone and chemical signalling pathways that make up the brain. By understanding these interactions, we can understand even the most complex behaviours. Experimental designs are continually updated to isolate and codify specific behaviours, and techniques for measuring neural activity are developed to allow researchers to more precisely examine the neural activity underlying each behaviour. Non-invasive measurement techniques, such as functional magnetic resonance imaging (fMRI) [Glover, 2011; Kwong et al., 1992] and electroencephalography [Coenen et al., 2014; Lindsley, 1952], allow brain-wide neural activity to be measured with minimal alteration to the organism. However, they yield comparatively little information about the activity of individual neurons. While knowing that the hippocampus (the brain region responsible for spatial memory) is active as the dog

[1] See the "What the fluff challenge."

steps into the elevator may be sufficient to answer high-level questions such as "*where* does processing happen?" it is unlikely to be sufficient to answer "*how* does processing happen?"

Consequently, high-fidelity measurement techniques have been developed. These techniques allow single-cell neural state dynamics and properties to be measured under controlled experimental conditions. Such techniques include electrode arrays [Spira & Hai, 2013] and patch clamps [Neher & Sakmann, 1992]. However, these techniques are highly invasive and can only directly measure a small number of neurons. Typically the brain of a dog has on the order of half a billion neurons, and as such, brain-wide measurement of individual neurons, or, observing the change in behaviour after ablation of individual neurons, is ethically questionable at best, and totally infeasible at worst.

This shortcoming highlights a fundamental contradiction in neuroscience. At its most granular, neuroscience studies low-level cellular interactions. Over millions of neurons, these individually simple and mechanistic interactions combine to give rise to complex and intelligent behaviour. However, it is impossible to measure all of the pertinent interactions at the fidelity required. This limits our ability to quantify the role of individual interactions in generating intelligent behaviour, and, limits our absolute understanding of how information is processed by flowing throughout the brain.

Instead, therefore, consider the scenario where we are able to digitally simulate the dog and its neural activity [Einevoll et al., 2019; Eliasmith et al., 2012; Eliasmith, 2013]. This simulation generates neural activity and behaviours *in silico* (in simulation) that are indistinguishable from that of a real dog. In this scenario, we put a digital dog, into a digital elevator, and simply read off the values of the digital neurons. These cellular-resolution values can be interpreted and analysed as though they were observations obtained from real neurons, and can be used to finally answer: "how do dogs rationalise elevators?"

More generally, we can use such synthetic data to examine any neural function, behaviour, or hypothesis, spanning multiple spatial and organisational levels, that we are capable of simulating. For instance, we can record and analyse the aggregate dynamics and high-level function of entire cortical columns, while still retaining the ability to evaluate the contribution of an individual neuron. We can generate counterfactual data, evaluating the change in behaviour resulting from an alteration to the connectome. We can also conduct experiments *in silico* that it may be logistically impossible or even unethical to investigate *in vivo*.

This transforms the question into: "how can we simulate an entire brain?" We have already suggested that organism behaviour is merely the emergent result of the interactions of the atomic units of the brain. This is a foundational principle of the neuron doctrine, and more specifically, the sub-discipline of *connectomics* [Seung, 2011, 2012]. Connectomics hypothesises that an organism's behaviours, memories, and even identity, are defined exclusively by the neurons and connections that make up the brain. Connectomics aims to build complete maps of the connections in brains, or *connectomes*, in the hope that this structural information will provide information about the function of the brain. While it is unlikely that the function of the brain is *exclusively* a function of the physical connections between neurons, it is not unreasonable to hypothesise that analysing and simulating the temporal dynamics of networks of neurons, or whole connectomes, will provide an unparalleled insight into neural function.

Therefore, the question becomes: "how can we simulate networks of neurons?" In the simplest of terms, neurons can be considered as electrical wires, connected to one another through a series of weighted junctions. Numerous models of the electrical dynamics of neurons have been created by considering the neuron as a series of electrical components [Abbott & Kepler, 1990]. Most famously, the Hodgkin-Huxley model reduces individual neurons to a circuit of conductances, capacitances and inductances [Hodgkin & Huxley, 1952]. This approach converts simulating the dynamics of networks of neurons into specifying an equivalent electrical circuit, specifying the parameters of this circuit, and then forward-simulating the low-dimensional equivalent circuit using basic circuit theory.

However, understanding organism-level behaviour using this approach requires the simulation of *all* the neural interactions, and hence requires a *complete* connectome. The only fully mapped connectome is that of the nematode roundworm *Caenorhabditis elegans* (*C. elegans*) [Brenner & Wood, 1988; Hope, 1999]. *C. elegans* is a translucent one-millimetre long roundworm, studied widely by neuroscientists as a *model organism* [Nigon & Félix, 2018; Riddle et al., 1997b]. The connectome of *C. elegans* was first mapped by White et al. [1986], and is one of the smallest known to science, consisting of just 302 neurons. Despite this, reconstructing the connectome took over a decade. Critically, this connectome is fixed for all *C. elegans* specimens of the same type, and hence we know the *entire* neural hardware on which all neural processing is performed *for all specimens*.

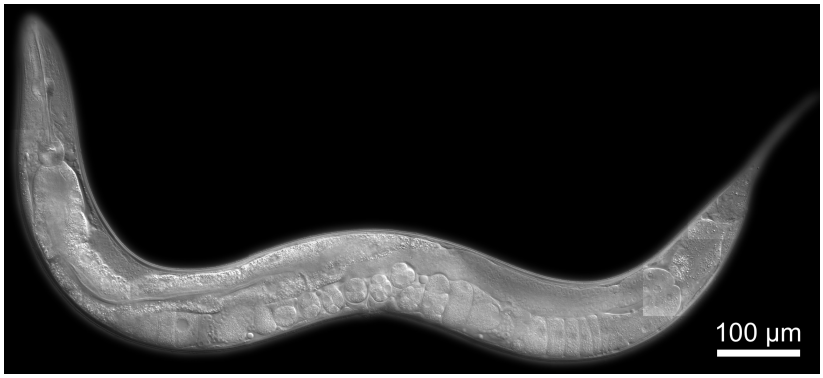


Figure 1.1: The *C. elegans* roundworm. Reproduced with permission from the Chin-Sang laboratory.

Despite its small size and comparatively simple physiology, *C. elegans* exhibits surprisingly complex behaviours, including multiple locomotion patterns [Berri et al., 2009], sexual reproduction, *chemotaxis* [Bargmann, 2006], and even elements of agency [Gordus et al., 2015]. Collectively this means *C. elegans* is an ideal organism for us to study.

Unfortunately, the *structural* connectome, such as the *C. elegans* connectome recovered by White et al. [1986], only conveys the *existence* of connections between neurons. It does not convey the functional properties of the network. For instance, each connection is, in fact, a weighted, non-linear, and highly variable mechanism that modulates the transmission of neural impulses between neurons. This variation and modulation is a critical component of computation in the brain. Furthermore, analysis based on the structural connectome alone neglects the influence of additional mechanisms not captured by the structural connectome, such as neuropeptides or neuroglia, as well as local variations in the functional properties of neurons and connections. Therefore, considering the structural connectome alone may not adequately describe the neural mechanisms than underpin particular phenomena. What we require instead is an accurate model of the *functional* connectome [Seung, 2011], describing the temporal dynamics of the brain.

The model of the functional connectome is specified through the equivalent circuit *and* the parameters of this equivalent circuit. Crucially, these parameters cannot be accurately inferred from the structural connectome alone. Establishing the strength of a connection from purely structural information is an outstanding challenge [Bhatt et al., 2009; Loewenstein et al., 2015]; and while these strengths can be *approximated* from the number and size of connections between two neurons [Branco et al., 2010; Meyer et al., 2014], it is not a perfect predictor, and requires each new specimen to be destroyed and painstakingly reconstructed.

Some electrophysiological parameters can be measured using invasive techniques, but again, this destroys the specimen, or, assumes that these parameters are constant across specimens. Using inaccurate parameter values renders the simulation, at best, a coarse approximation of the true system, and at worst, uninformative, even spurious [Eliasmith & Trujillo, 2014]. Therefore, the original question is reduced to "how can we infer representative parameter values of a whole-connectome simulator from readily obtainable data?"

Accordingly, the gargantuan task of understanding an entire brain has been reduced to estimating the parameter values of a single-neuron fidelity simulator, and then leveraging this refined simulator to analyse behaviour. Developing methods to perform this estimation and analysis is the central focus of this thesis.

An Aside on Dogs Before we move on, we return to our motivating example. We note that dogs have senses far more sensitive than our human senses. As a result a dog is far more acutely aware of the acceleration and deceleration of the carriage, the sound of the cables moving, and the slight changes in smell as the elevator changes floors. These combine to allow the dog to rationalise the elevator by recognising that it is moving, even without visual confirmation of this. However, the ability of the dog to combine these senses, and to use this as a rationalisation of the discontinuous spatial regions reachable via the elevator, highlights the sophistication of the cognitive processes of our canine companions.

1.1 It's Only a Worm: Why Bother?

An important question for us to ask is "why is this exercise valuable?" Not only does answering this question motivate both neuroscience and machine learning audiences to read this thesis, but also focuses our effort into building tools that are of practical use to the neuroscience community. Despite the comparatively simple physiology and connectome of *C. elegans*, simulating the entire organism has only recently come into focus [Larson et al., 2018; Stiefel & Brooks, 2019]. As such, developing methods for performing inference in *C. elegans* presents an exciting opportunity, not only to contribute to the understanding of *C. elegans*, but to wider neuroscience. We suggest several such opportunities and motivations, ranging from highly practical considerations to field-level objectives.

The first observation is that observing every time-varying variable of interest is often practically infeasible. For instance, we may wish to study the membrane potentials of the

motor neurons during locomotion, but are only capable of gathering measurements from a small fraction of these neurons. Possession of an accurate simulator facilitates inferring the unobserved variables of interest from *in vivo* observations. This enhances the data far beyond conventional data cleaning methods, and allows previously unobtainable neural activity responsible for generating a specific observed behaviour to be estimated from readily observable covariates by exploiting the covariances implied by the model. The inferred activities can then be inspected and analysed to further understand the behaviour.

Beyond this, this approach also provides the opportunity to estimate physiologically pertinent static parameter values that are impractical to measure, or, cannot be measured without significantly altering or destroying the organism. Such parameter values may include the connection strengths between neurons and the electrophysiological properties of neurons. More generally, we can aim to recover the *distributions* over these parameters, instead of simply a point estimate. Possession of the joint distribution over parameters facilitates the assessment of the covariances and relative importances of particular parameter values to the overall neural function. These distributions can also be recovered and compared across multiple specimens. The *variation* of these parameters between specimens can then be examined to further understand the neural function.

A simulator that generates simulations that are indistinguishable from true data also facilitates *in silico* experimentation using *unconditional generation*. Classical *in vivo* techniques typically modify a system by altering or inhibiting its function, using physical modifications or chemical treatments, and recording the changes in behaviour. Preparing and conducting these experiments requires significant technical and logistical overhead, and severely limits the scope of data that can be collected. In contrast, we can perform these experiments *in silico* simply by varying the model parameters, running a series of simulations, and outputting the change in behaviour. For instance, we can patch clamp an arbitrary number of neurons *in silico* simply by fixing the value of selected neural potentials to a particular value, and reading off the resulting dynamics of the rest of the neural state. These experiments are perfectly controllable and observable, as each physiological state is simply a variable in memory. Critically, the reproduction of experimental results by third parties becomes trivial through the distribution of source code and random number generator seeds, and only

requires standard computational hardware, instead of a specialist wet lab, highly trained technicians, and highly variable specimen preparation procedures.

Experiments can also be iterated on and refined more quickly. Suppose the results of an *in vivo* experiment highlights the importance of a neuron for which data was not acquired. Performing the follow-up experiment *in vivo* may require obtaining a new phenotype, patch-clamping neurons, and then physically conducting the experiment. In simulation, however, re-configuring the experiment to run may be as simple as changing a configuration file and running the simulation. These experiments can exploit massive parallelisation, essentially allowing as many experiments to be run simultaneously as there are CPU cores available, and, without user interaction. As a result, the experimental cycle is faster, cheaper, less technically demanding, and releases the experimentalist to perform other tasks while simulations run, instead of continuously monitoring an *in vivo* experiment.

While recovering latent states and continuous parameter values undoubtedly presents an exciting opportunity, these results are still implicitly conditioned on a fixed model. Therefore we should also perform inference over the model structure. More sophisticated experiments may *automatically* propose and test alternative models of neural function [Nagy et al., 2019; Schaechtle et al., 2017], drawing on themes from AutoML [Ghahramani, 2016; Feurer & Hutter, 2018] and program synthesis [Biermann, 1985]. Maybe the most well-known example of this is the *automated statistician* [Ghahramani, 2016; Hutter et al., 2019; Janz et al., 2016], where data analysis is automated by automatically searching over a set of composable operations to produce an interpretable model of the data. Transferring this reasoning to neuroscience, this approach promises to automate generating and testing new models of neural function and, in the absolute best-case scenario, automating considerable swathes of the mechanical elements of neuroscience research. This idea is extended by *conditional generation*, where simulations or modifications are forced such that specific criteria are met.

While understanding *C. elegans* alone is not the primary goal of neuroscience, *C. elegans* often provides a tractable experimental subject. Likewise, if we are able to achieve all of these outcomes in *C. elegans*, it represents a huge advancement in our understanding of how intelligent behaviour is created. It also serves as proof-of-concept for whole-organism computational modelling, and is a huge methodological leap forward towards understanding the brains of larger organisms. Efforts to map the connectome of *Drosophila melanogaster*

(*Drosophila*, common fruit fly) are already underway [Chiang et al., 2011]. *Drosophila* exhibits more complex behaviours than the *C. elegans*, such as mating [Dickson, 2008] and grooming [Berman et al., 2016]. The connectome of a healthy adult fly is estimated to contain 135,000 neurons, and hence represents a significantly more complex connectome than that of *C. elegans*. Therefore, analysis of *C. elegans* is a necessary a springboard towards the analysis of more complex organisms.

To perform the analyses described above we apply cutting-edge machine learning and probabilistic inference methodologies. This provides us with an opportunity to develop and apply machine learning methodologies in a new context. Crucially, however, it provides an opportunity to deploy machine learning to perform and automate neuroscientific analysis that is far beyond what is possible with traditional methods. As such, this work provides both a proof-of-concept, and an exposition, of the opportunities presented by combining mechanistic whole-connectome neural models with machine learning techniques.

The most high-level consideration is whether or not simulations are artificially intelligent organisms. Although it is questionable as to whether a biological worm is truly intelligent, they are still capable of displaying complex behaviours, including associative and non-associative learning [Ardiel & Rankin, 2010; Rose & Rankin, 2001] and elements of agency [Gordus et al., 2015]. If the simulated data is indistinguishable from real worm data, it could be argued that the simulation has passed a "worm Turing test." By definition, the simulation *is* as artificially intelligent as the real worm is intelligent. Most strikingly, this line of reasoning can be extended all the way to examining the very origin of consciousness. For instance, Koubeissi et al. [2014] found a region of the human brain that when stimulated resulted in "disrupted consciousness." The subject becomes completely unresponsive during the stimulation, but with no lasting side-effects once the stimulation is removed. Performing this stimulation *in silico* is trivial, and if similar disruptions are observed, would provide compelling evidence that the simulation is conscious, even sentient.

1.2 Purpose of Thesis

While contemplating science-fiction-worthy results is both important and thought provoking, the aims of this thesis are far more immediate. Our primary purpose is to tackle statistical estimation in computational models of *C. elegans*. This requires techniques and understanding from neuroscience, machine learning, statistics, and systems engineering. While we do not

intend for this thesis to be an exhaustive tutorial on each of these topics, we introduce each topic sufficiently such that any reader, regardless of prior experience, can engage with the material presented in this thesis, access more advanced literature, and is presented with an organised and unified set of ideas and techniques that might play a role in achieving our overall objectives. Although ultimately we do not complete all of the objectives outlined above, this work represents an ambitious first step, and provides a single jumping-off point for follow-up work. We also, therefore, record and disseminate the ideas that guided each step in the development of our work, and the specific choices made during our research, so that they can be understood, critiqued, and then built upon. We also provide several extensions that are within reach and follow directly from this work, and reflect on the high-level themes that arose as a result of this work. Clearly presenting these will hopefully enable new researchers to quickly grasp the critical ideas and concepts, and to expedite further research.

1.3 Organisation of Thesis

Figure 1.2 shows a high-level schematic of the layout of this thesis. We begin in Chapters 2 and 3 by introducing the neuroscience and machine learning background required for the material presented later in this thesis. These sections are not intended to be an exhaustive discussion of all the facets and rich histories of these large fields. They are instead intended to introduce the reader to the key aspects of these topics, and to front-load all existing theoretical concepts, definitions and explanations used in this thesis. We are then able to move on to discuss the technical material without needing to repeatedly reintroduce these foundational concepts. Any reader who considers themselves competent in either of these fields can skip the respective sections or subsections.

In Chapter 4 we introduce the first major body of work in this thesis, which we describe as the "virtual patch clamp" (VPC) [Warrington et al., 2019]. This work tackles performing inference in a biologically motivated model of *C. elegans*. We introduce a parametrised brain-body *C. elegans* simulator, designed and implemented with computational scalability as a priority. We then apply particle-based posterior inference methods to estimate distributions over unobserved physiological states and estimate global parameter values, conditioned on partial calcium fluorescence data. We demonstrate the efficacy of our method on a demonstrative autoregressive problem, and then apply these methods to synthetic *C. elegans* data. We critically evaluate the methods presented, and discuss opportunities for scaling

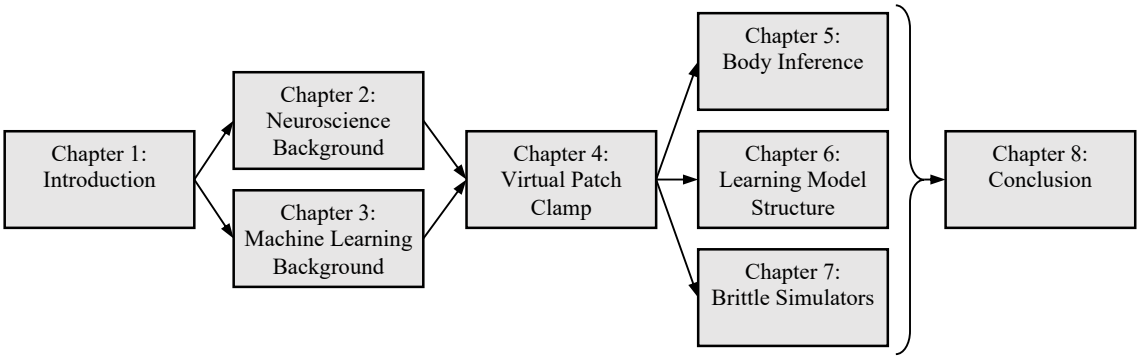


Figure 1.2: Schematic of the layout of this thesis. This introduction (Chapter 1) is followed by chapters containing the background neuroscience (Chapter 2) and machine learning (Chapter 3) material required for this thesis. We then introduce the central piece of work in this thesis, "the virtual patch clamp," tackling inference and estimation in a physiologically grounded model of *C. elegans* (Chapter 4). As part of this work, we identify three smaller extensions which we subsequently present: performing inference conditioned on body position (Chapter 5), automatically learning the structure and functional form of the model from data (Chapter 6), and performing inference in brittle simulators (Chapter 7). We then conclude by discussing the work presented, the central themes, possible extensions and some final thoughts (Chapter 8).

this method up to large-scale, real *C. elegans* data. As part of this work we highlight three extensions, which we explore in Chapters 5, 6 and 7.

We tackle the first extension in Chapter 5. We observed in Chapter 4 that conditioning on the initial body pose greatly improved the fidelity of the reconstructions, to the point of practically being a requirement. This chapter extends this observation to condition on the body pose throughout the entire time series. We propose that body pose data is a complementary data modality to calcium fluorescence data. We present a pipeline for processing raw video footage of *C. elegans* such that it is amenable for inference, and then show the ability to impute neural states and parameters from synthetic body pose time series. We then apply the methodology to real data, although with less success. This work also represented somewhat of an inflection point, both highlighting the need, and providing an opportunity, to apply more sophisticated inference tools going forwards. We therefore dedicate significant effort to discussing the extensions arising from this work.

The second extension, presented in Chapter 6, is the opportunity to learn the discrete structure of the model directly from data. Specifically, we consider recovering the update rule governing synaptic plasticity. We introduce a flexible class of synaptic plasticity rules into a restricted neural model. We then show on a simple synthetic example that the correct learning rule can be recovered without human intervention, while simultaneously recovering the static

parameter values using gradient descent. We find that using the incorrect learning rule leads to the recovery of incorrect parameter values. As such, learning the model may be necessary to obtain representative parameter estimates and latent state imputations. This short experiment also highlights that the methods used in the preceding chapters are not the only method for recovering parameter values from data. Finally, this demonstrates how cutting-edge machine learning techniques can be deployed to automate neuroscience research.

We present the final extension in Chapter 7, building on the observation made in Chapters 4 and 5 that perturbing highly structured models is often non-trivial. We observed that the body simulator crashed for all-but-minuscule perturbations to state, a property we describe as *brittle*. These crashes can remove the majority of the particles at each time step, causing crippling particle degeneracy and increasing the variance of any subsequent inference result. We therefore develop a method for selecting the perturbation applied to state prior to iteration to minimise the number of particle failures, but that does not change the model. This increases the efficiency and reduces the variance of subsequent inference methods. We are able to train an amortised artefact capable of efficiently proposing perturbations that have a high probability of succeeding, leading to markedly less particle degeneracy and lower variance inference results. Much of this work was first presented in Warrington et al. [2020b].

We conclude by discussing the broader themes presented in this thesis. The technical work and results presented are only the first step on a longer road towards realising the ambition of fully synthetic brain models and *in silico* neuroscience. We therefore discuss several research opportunities that follow directly from this work. We conclude by reflecting on the high-level observations and learnings that can be taken from this work, and the opportunities for pushing *C. elegans* modelling[2] to the forefront of neuroscience and machine learning research.

[2]Or understanding dogs in elevators.

2

Neuroscience Background

Neuroscience is the study of neurons, the cellular building blocks of the brain, and how neurons combine to create intelligent behaviours. Neuroscience, on one hand, has an incredibly practical motivation. By understanding the brain more deeply, the underlying causes of neurological disorders, such as Alzheimer's disease, Parkinson's disease and epilepsy, can be discovered. Using this knowledge we can then develop more effective treatments and cures for these most debilitating of diseases. Beyond this, however, neuroscience is motivated by the basic desire to understand and rationalise our own lived experience. Neuroscience therefore provides evidence for more philosophical questions, such as why and how we feel pain, why social creatures seek out the company of others, and why we are innately motivated to learn and discover. These questions, ultimately, bridge the gap between the physical sciences and philosophy, providing evidence for questions pertaining to ethics, the human condition, and nothing less than consciousness itself.

Much of the technical effort in neuroscience is dedicated to codifying observed behaviours and measuring the cellular-level neural circuitry responsible for creating these behaviours. However, there exists a curious dichotomy that does not manifest as strongly in other sciences. We are intimately aware of our own mind and the experience of consciousness, and yet, for centuries, the tools required to examine and analyse the origins of that consciousness simply did not exist. Therefore, over its history, neuroscience has ranged from very pragmatic studies, to wild conjecture and speculation (retrospectively, at least). Many fanciful hypotheses were proposed, from albeit very limited observations, that were more heavily influenced by romance, politics, religion, and, often, pure convenience, than by rigorous scientific experimentation.

However, even the most fanciful of hypotheses spurred the development of new experimental techniques to test these hypotheses. New data and insight, garnered by using these new techniques would, in turn, spur the generation of new hypotheses. Although the hypotheses, tools, practices, and ethics have changed dramatically throughout the history

of neuroscience, this theme remains constant. Neuroscientists develop and use tools and procedures, often on the cutting edge of science and technology, to observe more deeply than ever into how our brain works. Hypotheses about the function of aspects of the system that cannot be observed are then proposed, drawing on intuitions, prior beliefs, and concepts from other fields.

This is the general observation from which this thesis takes direction. We attempt to develop a new generation of tools, such that more sophisticated theories can be developed, refined, quantitatively tested, and ultimately, accepted or rejected. Instead of more direct chemical, optical or biological methods, we predominantly discuss and develop the scientific instrument of information processing. Like the techniques before, the methods we present in this thesis are inevitably not the final incarnation: they will be revised, enhanced, surpassed, and eventually discarded to replaced by new methods. However, the hope is that the methods we present, at least in part, enable the next iteration of neuroscience research.

In this chapter we introduce the reader to the core neuroscience background required for this thesis. While much of the technical content in this dissertation can be considered as purely a statistical inference problem, the true scientific interest lies in the interpretation and opportunities presented when considered as a neuroscience exercise. We begin by providing a short history of neuroscience, orientated specifically for grounding the terminology and motivation for the topics we discuss later. We then introduce basic neural anatomy and the specimen that we predominantly consider in this thesis, *Caenorhabditis elegans*. We then conclude by introducing current data acquisition methods, and the mathematical models we use to analyse this data in later chapters.

2.1 A Brief History

The earliest recorded mention of the brain is contained in an Egyptian scroll dating from the 17^{th} century BC [Finger, 2001]. The Egyptians recognised that the brain was the source of a number of pathologies and illnesses, but believed that the heart was the seat of consciousness. This *cardiocentricism* was the primary belief of many classical Greek philosophers as late as Aristotle. In 4^{th} century BC the *cephalocentric* hypothesis, championed by Hippocrates and Plato, predominantly replaced cardiocentricism, instead identifying the brain as the seat of consciousness. Curiously however, Aristotle, a student of Plato, maintained throughout

that the brain could not be the centre of consciousness, citing that some organisms could continue moving after their brain was removed.

At the time, dissection was forbidden in Greece, and hence many early Greek cephalocentric theories were not rigorously backed by observation. Herophilos of Chalcedon, a 3^{rd} century BC Greek anatomist working in Alexandria, Egypt, performed some of the first rigorously documented dissections on living patients. These experiments effectively disproved cardiocentricism in favour of cephalocentrism. Herophilos hypothesised that the brain was responsible for conducting and processing *pneuma*, the physical manifestation of thoughts and motor commands. Neurological disorders were then caused by an imbalance of the four humours (bile, black bile, phlegm and blood) preventing pneuma from flowing correctly.

The ideas of Herophilos were advanced by Galen, who theorised that different parts of the brain performed different functions. Galen hypothesised that the much firmer *cerebellum* must control the much firmer musculature, and that the softer *cerebrum* must be where thoughts, feelings and sensations were processed. Although Galen was ultimately correct, it was certainly not for the right reasons. Further, and taking guidance from Plato, Galen believed that Herophilos' pneuma was in fact animal spirits, that roamed through the ventricles of the brain and gave rise to different thoughts and emotions. This theory would be expanded later to include the movement of animal spirits through the nervous system, and that the movement of animal spirits into muscles caused muscular contraction.

The state of neuroscience then remained relatively constant for a period of centuries. Islamic scholars in Persia and the Iberian peninsula began to identify and codify various neurological disorders, and concluded that these pathologies originated in the brain. More detailed anatomical drawings of the brain were being created, and sub-structures of the brain were being described in progressively more detail. Descartes declared that the pineal gland specifically was the seat of the soul [Lokhorst & Kaitaro, 2001], (mistakenly) believing the gland was located in the middle of the ventricles and hence was in constant contact with the animal spirits proposed by Galen and Plato. This, however, was to be the last incarnation of the spirit-based beliefs of Plato, and marked the beginning of the transition towards "modern neuroscience."

In the early 19^{th} century, Luigi Galvani applied electrical current to the legs of frogs, observing that each application resulted in muscular contraction [Piccolino & Bresadola, 2013]. He further observed that the strength of contraction increased as the strength of the stimulus applied increased. From this, Galvani concluded that electricity was the medium of neural communication, and not animal spirits. In 1815, Jean Pierre Flourens demonstrated that animals with lesions in specific regions of the brain lost specific and predictable mental faculties [Pearce, 2009]. He hypothesised that regions of the brain were responsible for certain faculties and was able to identify these regions with remarkable accuracy. These findings, along with the work of the English polymath Francis Galton and German physiologist Franz Joseph Gall, were pivotal in the development of *phrenology* [Simpson, 2005]. Phrenology hypothesised that particular regions of the brain were responsible for certain characteristics, and that the *size* of these regions of the brain, and by proxy the head, was predictive of their relative acuity. Phrenologists drew correlations between the size and shape of heads with personality types, traits, skills and disabilities – even hypothesising that regions of the head were predictive of qualities such as benevolence and destructiveness. This culminated in the publication of many phrenology maps, describing the *functional* layout of the brain.

The final wrong turn of pre-modern neuroscience was *reticular theory*, proposed by Joseph von Gerlach, and corroborated by chemist and neuroanatomist Camilo Golgi [Cimino, 1999]. Reticular theory hypothesised that the brain was made up of a continuous and interconnected series of tubes through which information flowed. This was in stark contrast to the newly developed cellular theory, which hypothesised that organic tissue was a collection of small, individual and discrete cells. Golgi became convinced by this theory during the development of the *Golgi stain*; a silver-chromate solution that could be used to dye regions of tissue such that individual cells (what would later be referred to as *neurons*) became visible under a microscope [Golgi, 1873]. Critically however, the Golgi stain did not provide a uniform and high-fidelity stain, and only coloured roughly one in one hundred thousand cells. Golgi therefore agreed that the faintly visible tree-like cells were interconnected tubes. He suggested that the brain had a special place as an organ and hence did not need to abide by cellular theory. Reticular theory would shortly be disproved, in turn marking the end of pre-modern neuroscience and the beginning of modern neuroscience.

While many of the theories presented in this section were ultimately disproved, they highlight how neuroscience has always had an element of informed guesswork. Although much of this guesswork was influenced by wayward or historic beliefs, it was always founded on what could be measured. By using "models" derived from their beliefs about the underlying system or from other fields, "inferences" were drawn about what could not be observed.

2.2 The Neuron Doctrine & Connectomics

Modern neuroscience begins at the turn of the 20^{th} century, ushered in by Santiago Ramón y Cajal [DeFelipe, 2002]. Cajal, a Spanish neuroscientist, learned of the Golgi stain while working as a professor in Barcelona. Cajal refined the Golgi staining protocol by instead using two short perfusions instead of one long perfusion as originally suggested by Golgi. This improved the colour, contrast, and detail the stain provided. Cajal also had access to new and improved microscopes, which, coupled with his improved staining method, meant he could study neural tissue in unprecedented detail. The sketches and diagrams made by Cajal, such as those shown in Figure 2.1, were the most accurate depictions of neural tissue at the time.

Cajal discovered that neural tissue was comprised of millions of discrete, individual cells, in stark contrast to the continuous system of tubes hypothesised by reticular theory. These cells would later be called *neurons* [Finger, 2001]. Cajal's discovery of an atomic unit effectively disproved reticular theory (much to the ire of Golgi), provided strong links to existing cellular theory, and was the foundation of what would go on to become known as the neuron doctrine [Shepherd, 2015]. Cajal and Golgi shared the 1906 Nobel Prize in Physiology or Medicine for their work advancing the understanding of the nervous system. However, Cajal and Golgi had a turbulent relationship, with Golgi remaining a staunch believer in reticular theory, and (allegedly) irked that Cajal had modified and improved his staining method. Golgi even disputed Cajal's findings in his acceptance speech for the Nobel Prize they shared. As a footnote, the efforts of Fridtiof Nansen in developing foundational neuron theory are often downplayed or omitted, as indeed they were by Cajal [Edwards & Huntford, 1998].

Cajal further hypothesised that the function of the brain was the result of simple interactions across many individual neurons. Cajal noticed there are visibly two "sides" to a neuron – now known as axonal and dendritic arbors, visible in Figure 2.1. Using the knowledge that neurons are discrete, contiguous units, he concluded that one side accepts

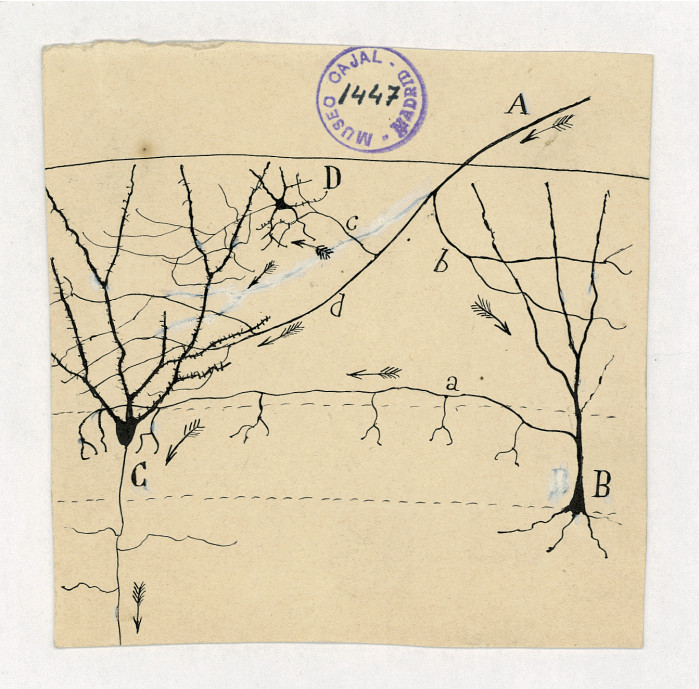
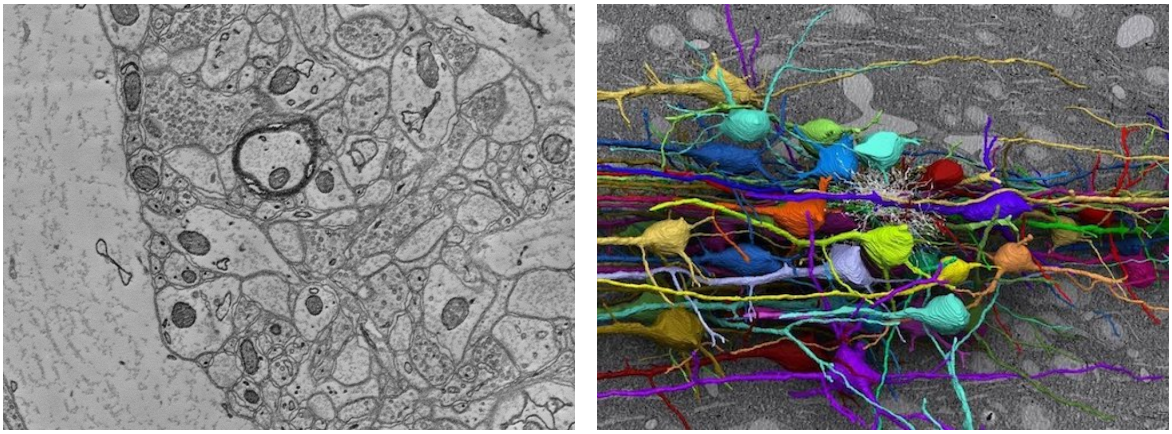


Figure 2.1: Hand drawn diagram of four neurons by Cajal [Cajal, 1899]. The direction of information flow in each neuron is indicated by arrows. Information flows out of the neurons through the axon, and is received as input by the dendrites. Dendritic spines are visible on neuron $\mathbb{C}$ as small black protrusions, and are the site of the dendritic side of a neural connection. This image (M1447) is reproduced with courtesy of the Cajal Institute, Cajal Legacy, Spanish National Research Council (CSIC), Madrid, Spain.

"inputs" to the neuron, while the other conveys "outputs." To test his hypothesis, Cajal painstakingly traced the branches of neurons from the retina all the way to the visual cortex. Noting that information must originate in the eye before being transmitted to the cortex, Cajal was able to trace the circuit, and at each neuron, note which "side" was responsible for receiving inputs and transmitting information onwards. Cajal correctly determined that axons transmit signals from a neuron, which in turn act as the inputs to the dendrites of other neurons. The discovery of this polarity, known as the *law of dynamic polarisation*, is one of the cornerstones of the neuron doctrine [Shepherd, 2015].

For his efforts, Cajal is often cited as the "father of modern neuroscience." Cajal would discover many of the pillars of neuron theory. His work, and the earlier work of Galvani, effectively mark the end of pre-modern neuroscience, and the influences of Plato and his disciples. It also marked a turning point, where neuroscientists were able to observe the fundamental atomic units of the brain in sufficient detail to draw evidenced conclusions about function from anatomical structure. The neuron doctrine considers neurons to be discrete units, each one classifiable as part of a hierarchy (just as the rest of the cells in the



(a) **(b)**

Figure 2.2: Figure 2.2a shows a scan of neural tissue using an electron microscope. Thousands of scans of neural tissue slices are processed to generate dense neural segmentations and connectivity maps, such as is shown in Figure 2.2b. Data and images are reproduced with permission from Daniel Berger [Kasthuri et al., 2015].

body are), as having finite and quantifiable inputs and outputs, and a specific polarisation for behaviour. Armed with this knowledge, neuroscientists are able to reduce the function of the brain to the net behaviour of a collection of discrete units, each of which acting as an "absolutely autonomous physiological canton" (Cajal, as quoted in Finger [2001]), or, as individual units with known properties and behaviours.

The logical extension of this is that even the most high-level behaviours can be explained by synthesizing the function of successively smaller components of the brain. This reasoning is the driving force behind much of neuroscience, particularly systems neuroscience, and is taken to the logical extreme by the field of *connectomics* [Seung, 2012]. Connectomics attempts to demystify the deepest functioning of neurobiological tissue by building and analysing the *wiring diagram* – or *connectome* – of volumes of tissue at unprecedented scale and accuracy, just as Cajal did in the optic tract. Connectomics posits that these connectomes, when constructed at the scale of entire brains, can be used to link the function of individual neurons, or even individual connections, to high-level behaviours.

However, the early drawings by Cajal under-represent the complexity of neural tissue. The Golgi stain only stains one in every $1,000$ to $100,000$ neurons, and the light microscopes used by Cajal were only capable of imaging the larger branches of neurons. In the "white space" of these drawings (such as in Figure 2.1) there are potentially tens of thousands of neurons and many metres of neural process that are unstained, and hence are invisible. The true complexity of neuropil tissue was only truly captured with the development of

high resolution electron microscopy (EM) methods [Bogner et al., 2007; Leighton, 1980]. Electron microscopes use a beam of electrons, instead of visible light photons, to construct images. The wavelength of electrons is orders of magnitude shorter than visible light, and hence can resolve structures in much greater detail. An example of an EM image is shown in Figure 2.2a. However, the images produced by electron microscopy are dense, noisy, and can contain trillions of voxels for even a modestly sized volume of tissue. This means that analysing electron microscopy images is not as simple as sketching individual neurons as Cajal did. High-fidelity connectome reconstructions were first created through painstaking manual annotation [Kasthuri et al., 2015; White et al., 1986]. The reconstruction published by Kasthuri et al. [2015] took several years to complete. Therefore, sophisticated machine-learning reconstruction pipelines have been developed to automate this process [Jain et al., 2010], albeit at the expense of requiring immense computational firepower.

The process for producing these reconstructions generally starts by first cooling the brain (to reduce cell atrophy) and extracting the brain after death. The brain is then perfused with chemicals to dehydrate the tissue, prevent proteins from breaking down, and fix macro-structures to increase rigidity and durability [Newman & Hobot, 1999]. The tissue is then stained to increase contrast in EM images [Mikula et al., 2012]. The brain is sectioned into ultra-thin slices ($\approx$ 50nm) [Knott et al., 2008; Schalek et al., 2011] and are then scanned using an electron microscope to create images, such as is shown in Figure 2.2a [Denk & Horstmann, 2004; Helmstaedter et al., 2008; Kasthuri et al., 2015]. These steps can also be combined through the use of FIB-SEM [Xu et al., 2017]. These images are then segmented to create continuous regions representing each cell, sub-cellular structure, and connection point between neurons [Jain et al., 2010; Lichtman & Denk, 2011], as shown in Figure 2.2b. Finally, these regions are analysed to create connectivity graphs, such as the connectome shown in Figure 2.3 [Gray Roncal et al., 2014; White et al., 1986; Varshney et al., 2011]. The result of this is a segmented and annotated volume, gigabytes or terrabytes in size, and a matrix, defining the connectivity of each neuron in the region. These connectomes can then be analysed, compared and contrasted to understand the structural organisation of neural tissue.

However, a critical assumption (often implicitly) made by connectomics is that one can determine "function from form," where it is possible to recover the function of neural tissue from its structure, or even more ambitiously, the connectivity graph alone. For instance, the

most numerous type of connection between neurons, *chemical synapses*, are microscopic chemical junctions that transmit and modulate the electrical impulses that neurons conduct. There is strong evidence that synapses encode memories [Bruel-Jungerman et al., 2007; Mayford et al., 2012], and that a reduction in the density of these connections contributes to neurodegenerative diseases such as Alzheimer's disease [Sheng et al., 2012] and Parkinson's disease [Bellucci et al., 2016]. Therefore, according to connectomics, being able to map and analyse structural connectomes from both healthy and diseased brains would describe the causes of these neuropathies.

However, there are numerous neural mechanisms not described by just the binary existence of connections that contribute to such neuropathies, and organisms behaviours more generally. These include, for instance, the *efficiency* with which synapses operate [Picconi et al., 2012; Sheng et al., 2012], and even functionality of glial cells [Dzamba et al., 2016], which are often completely discarded by structural connectomes. Therefore, to fully understand these neuropathies we must analyse *functional connectomes* [Seung, 2011], describing the temporal dynamics of the brain (*including* the mechanisms mediated by cells other than neurons), instead of analysing structural connectomes. As such, recovering the functional connectome is of critical importance to furthering neuroscientific understanding. Possession of a functional connectome, expressed as a parametric probabilistic or computational model of neural dynamics, would facilitate precise predictions of the function of arbitrary sub-volumes of tissue and the response to external stimuli. This culminates in being able to use whole-brain, biologically-grounded simulators modelling the cellular-level neural interactions, informed by the structural and functional connectome, to simulate and analyse emergent, organism-level, intelligent behaviours. However, estimating the functional connectome from the structural connectome is an outstanding challenge. Driving towards solving this challenge is a major motivation behind this thesis.

A further open question is the scale and level of detail required in the connection map for effective analysis, either structurally or functionally. The sheer scale of the images produced by electron microscopes often leads analysis to be performed on approximate connectomes constructed from low-resolution images and bulk connectivity statistics [Markram, 2006; Markram et al., 2015]. These synthetic networks offer limited information on the actual low-level computation being performed by a particular neuron in a real organism [Eliasmith

& Trujillo, 2014]. However, the logistical effort and computational cost of gathering and analysing data with sufficient detail to be informative functionality may be prohibitive. We may also only be able to reconstruct fragments of larger connectomes in reasonable timeframes, and hence we may have to accept low-resolution reconstructions or approximate connectomes for large swathes of tissue, and restrict high-resolution analysis to small subvolumes of tissue. Balancing (or eliminating) these trade-offs and developing techniques to leverage complete, partial or approximate structural connectomes to reveal the functional connectome is a grand challenge of connectomics moving forward [Seung, 2011].

While the field of connectomics, whether studying structural or functional connectomes, is still in its infancy, it is merely an extension of the ideas first posited by Cajal. The critical difference between the work of Cajal and modern connectomics is the vast technological innovations and person-power now available. Automated scanning electron microscopes and supercomputer-scale processing algorithms enable large regions of tissue to be reconstructed at nanometre precision in short time frames. Indeed, the main "field-level" contribution of connectomics may be, in fact, expanding the purview of classical neuroscientific analysis from considering individual neurons or small assemblies of neurons, to instead analysing millions of neurons and billions of connections at sub-cellular resolution.

2.3 Neural Anatomy

We now give a brief introduction to the key aspects of neural anatomy. This section is intended only to familiarise non-neuroscientists with the basic terminology and concepts required to engage with the neuroscience focused discussions presented later in this thesis. For more information we refer the reader to more advanced neuroscience-specific texts [Dayan & Abbott, 2001; Shepherd, 1988, 2015].

The mammalian brain is composed of billions of individual cells, blood vessels and fluid tracts. However, the fundamental unit of computation in the brain is the neuron. Neurons are tree-like structures, as shown in Figure 2.1. The branches of neurons, referred to as *neural processes*, can measure just microns in diameter and can extend as far as centimetres from the cell soma. As noted by Cajal, neurons have two types of process: *axonal* and *dendritic*. Axons are typically larger, longer and less branched than dendrites, and are the above-ground half of the tree in the analogy. Axons carry the output of the neuron, transmitting signal

to downstream neurons. Dendrites are generally shorter, thinner, and bushier, and are the "roots" of the neurons, collecting inputs from neighbouring axons.

Information is processed in the brain by passing and modulating signals through chains of potentially millions of neurons. Each neuron aggregates the inputs from the neurons it is connected to and passes on signals to the downstream neurons. Each neuron acts as a non-linear operator, only selectively responding to inputs. This allows complex functions, such as detecting particular patterns in visual stimuli [Hubel & Wiesel, 1968] or generating complex movement patterns [Arber, 2012], to be implemented by composing many simple individual units. Most famously, Quiroga et al. [2005] discovered a neuron that activated *exclusively* when the subject was shown an image of actress Jennifer Anniston. On a high level, the upstream neural circuitry has developed a method of identifying meaningful features for identifying a particular person. This single neuron then aggregates these features to establish if a particular person is present. The downstream neurons can then use this higher-level binary existence or non-existence to perform even more sophisticated processing.

Neural signals are processed and transmitted by changes in the electrical and chemical state of the neuron. These dynamics are mediated by the movement of charged ions across the cell membrane, along the branches of the neuron and between neurons. Calcium, sodium and potassium are amongst the most important ions in this process. The "activity" in a neuron is described most generally by the fluctuations in the difference in electrical potential between the interior and exterior of the cell across the cell membrane, referred to as the *membrane potential*. The *resting potential* or *equilibrium potential* is the membrane potential when there is no neural activity, and when there is no net flow of ions across the cell membrane. This is often considered as a fixed datum by which to analyse the relative fluctuations of the membrane potential. The resting potential is negative relative to the extracellular space (referred to as a negative membrane potential) due to a lower concentration of positively charged potassium ions inside the cell compared to positive sodium ions outside of the cell. These chemical gradients are maintained by potential controlled *ion gates* and *ion pumps*. Gates allow specific ions to diffuse into and out of the cell, and pumps allow ions to be forcibly pulled into or ejected from the cell. When the cell is at the resting potential, most gates are shut and the ion pumps are ejecting excess sodium ions from the cell and pulling in potassium ions into the cell.

When the membrane potential rises, generally as a result of external input from other neurons, potential-controlled ion gates that permit the flow of sodium ions open. This allows positively charged sodium ions to flow into the cell, further increasing the potential of the cell, referred to as *depolarisation*. This in turn causes more sodium gates to open. If there are insufficient ions to sustain this rise or other factors preventing this rise, the potential will cease increasing and will return to equilibrium. This is referred to as a *failed firing*. If the potential rises sufficiently, above the *threshold potential*, the positive feedback loop will cause the membrane potential to rise sharply. This event is referred to as an *action potential* (AP) and the neuron is said to have *fired*, i.e. the neuron identified by Quiroga et al. [2005] fires when the subject is shown a picture of Jennifer Anniston, which is a discrete signal indicating the existence of a particular person. Once the membrane potential rises above a certain value, the sodium gates shut and the potassium gates open, allowing positively charged potassium ions to diffuse out of the cell, and ion pumps act to eject sodium ions from the cell. This causes the potential to fall rapidly, referred to as *repolarisation*. The potential often falls below the resting potential, referred to as *hyperpolarisation*. After firing, there is a period known as the *refractory period* where the neuron is hyperpolarised and is unable to fire again. Over a period of milliseconds, the neuron will equilibrate, returning to the resting potential, gate configuration and ion concentrations, at which point it is ready to fire again. Neurons firing in this way is the basic unit of computation and signalling in the brain.

The initial rise in membrane potential generally originates at the connection between neurons, referred to as *synapses*. Synapses most often occur between the axon of the *presynaptic* neuron and the dendrite of the *postsynaptic* neuron. Critically, these connections allow neurons to modify the electrical and chemical state of the neurons they connect to. This allows information to be injected from one neuron into another neuron. There are two main types of connection: *chemical synapses* and *electrical gap junctions* (also referred to as *electrical synapses*).

Chemical synapses are more numerous than electrical synapses. A chemical synapse most frequently occurs at the contact point between an axonal *bouton* and a dendritic *spine*. The bouton is a small bulge or protrusion from the axon that stores *neurotransmitter*. The bouton acts as the sender in communication, and neurotransmitter is the modality of this communication. The spine is a longer protrusion from the dendrite that approaches the

bouton and acts as the receiver. There is a small gap between the bouton and spine, referred to as the *synaptic cleft*. When the presynaptic neuron fires, potential-controlled chemical gates on the axonal bouton open and neurotransmitter-filled bubbles, called *vesicles*, are released into the cleft. These vesicles flow through chemical channels on the dendritic spine and trigger ion gates to open in the cell membrane of the postsynaptic neuron. This causes the membrane potential to rise in the postsynaptic neuron, which, as described previously, can lead to an action potential.

Electrical synapses are markedly simpler. These junctions are simply open channels between two adjacent neurons through which charged ions can flow. When the relative potential in one neuron increases (or decreases), ions passively diffuse through the channel, increasing (or decreasing) the potential in the second neuron. If this flow is sufficiently large, an action potential can be triggered.

There are several key functional differences between chemical and electrical synapses beyond simply their physical differences. Gap junctions are *bidirectional*. Ions can flow through the junction in either direction, unlike chemical synapses where the signal can only flow from the presynaptic to the postsynaptic neuron. Gap junctions therefore have a diminished notion of presynaptic and postsynaptic identity and simply equilibrate two connected neurons, whereas chemical synapses enable one neuron to activate a second neuron, and not vice versa. Gap junctions also operate nearly instantaneously and continuously, whereas chemical synapses take in the order of milliseconds to transmit signals and become ready to transmit again. Finally, gap junctions do not require the potential to exceed the threshold potential to transmit information and hence are more sensitive to sub-threshold potential changes.

Thus far we have discussed *excitatory* chemical synapses, where a presynaptic release *increases* the probability of a postsynaptic action potential. A small fraction of synapses are *inhibitory*, where a presynaptic release actually *reduces* the probability of a postsynaptic action potential. Whether a synapse is excitatory or inhibitory is determined by the type of neurotransmitter, and all synapses on a single neuron are generally the same type.

Synapses, both electrical and chemical, are not simply on/off mechanisms. They are instead *weighted* junctions, where the weight of a given junction is broadly referred to as the *synaptic strength* or the *synaptic weight* [Glasgow et al., 2019]. Often, “weight”

is generically used to simply indicate the notion of the relative level of amplification of a signal across the junction, although this definition is somewhat imprecise. In firing neurons, the synaptic strength for (excitatory) chemical synapses can be defined as the probability that an action potential in the presynaptic neuron elicits an action potential in the postsynaptic neuron [Branco et al., 2010]. In graded neurons (discussed later), strength can be defined as the change in postsynaptic potentiation for a unit change in the presynaptic potentiation [Manor et al., 1997]. The strength of a chemical synapse is dependent on factors such as the efficiency of uptake on the postsynaptic neuron, probability of vesicle release given a certain activity level in the presynaptic neuron and the number of vesicles available close to the cleft [Glasgow et al., 2019]. These determinants are themselves defined by factors such as the geometry of the synapse and the concentration of ion channels on both sides of the cleft. The strength of an electrical synapse is determined by the ability of ions to flow through the junction in response to a change in potential, or, the electrical coupling coefficient [Haas, 2015]. In general, however, stronger synapses transmit a stronger signal, either through the release of more neurotransmitter, more effective permeation of neurotransmitter into the postsynaptic terminal, or, less restricted flow of ions between neurons.

However, a single synaptic transmission is unlikely to generate the potential change required to trigger an action potential. A neuron therefore only responds to input from a single neuron if the connection is exceptionally strong, or more importantly, there are simultaneous action potentials and synaptic transmissions in several presynaptic neurons. This allows neurons to respond selectively. For instance, long hair alone is not sufficient to identify Jennifer Anniston. However, the simultaneous presence of long hair, light skin, blue eyes and the absence of a beard (which would trigger *inhibitory* synapses) may be sufficiently predictive. Hence, the complex computational task of identifying a person is broken down into many simpler tasks that are then composed to perform higher-level cognitive tasks.

The precise mechanism through which information is encoded by synaptic activity is an area of active research. The prevailing hypothesis being that information is conveyed through both the frequency and timing of these discrete events [Brette, 2015], although alternative hypotheses that consider individual spikes as continuous signals are gaining popularity [Maley, 2018; Tee & Taylor, 2020]. The (binary) existence of a connection

between two neurons and the continuously valued strength are large factors in how networks of neurons transmit and process information.

The connection strength of synapses changes over time in a process called *plasticity* [Dayan & Abbott, 2001; Huttenlocher, 2009]. Plasticity is one of the most important and widely studied topics in neuroscience [Mateos-Aparicio & Rodríguez-Moreno, 2019]. The adaptation of synaptic strength allows the relative coupling between pairs of neurons to be modified, allowing circuits of neurons to collectively adapt their function such that they can perform a particular cognitive task. Plasticity plays an important role in learning and memory. The change in neural strength is generally modelled as *activity dependent*, where the instantaneous and relative activity in presynaptic and postsynaptic neurons determines the change in strength. Most famously, *Hebbian learning* [Hebb, 1949] states that the rate of change of synaptic weight is proportional to the product of the presynaptic and postsynaptic firing rate, summarised in his infamous quote: "neurons that fire together, wire together." Many more complex rules describing the weight change exist, as well as quantifying the role of reward mechanisms in this adaptation [Hyman et al., 2006].

Finally, there are numerous classifications of neurons. We have already introduced that a neuron is typically excitatory or inhibitory and with a specific neurotransmitter. However, there are several other descriptors used to define the functional and structural properties of a neuron. Most broadly, the functional properties of neurons are further delineated as *afferent neurons*, *efferent neurons* and *interneurons*. Afferent neurons, also referred to as *sensory neurons*, collect and convey information from non-neural tissue. Sensory neurons are specialised for their respective task, including olfaction (smell), mechanosensitivity (pressure or stretch reception), and temperature sensitivity. Efferent neurons, also referred to as *motor neurons*, carry impulses from neural tissue to muscles to induce muscular contraction. Finally, interneurons process and transmit information, connecting exclusively with other neurons. Interneurons are by far the most common class of neurons in most brains. A rich hierarchy of structural cell types and morphologies have been developed, such as the iconic pyramidal neuron, and the intricate, beautiful, almost fractal Purkinje neurons. Each of these cell types are evolved to have particular connectivity patterns, connection range and broad functional purpose.

This concludes our general introduction to neuroscience and neural anatomy. We presented here a brief history of neuroscience, the development of connectomics and the neuron doctrine, and introduced the basic anatomical terms and concepts that we will reference throughout this thesis. The critical take away from these sections however is that organism behaviour is simply a function of the atomic units of the brain. Scaling this granular, low-level understanding to system-level understanding is an outstanding grand challenge in neuroscience. In the following sections we introduce material that we use directly in the technical part of this thesis, beginning by introducing our test subject, *Caenorhabditis elegans*.

2.4 *Caenorhabditis elegans*

Caenorhabditis elegans (*C. elegans*) is a translucent nematode roundworm that has been the subject of significant scientific study for over fifty years [Ankeny, 2001; Frézal & Félix, 2015; Nigon & Félix, 2018]. Despite being just one millimetre in length, three Nobel prizes have been shared by eight researchers for work involving *C. elegans*. Scientific interest in *C. elegans* increased dramatically in the 1960s, when Sydney Brenner advocated for increased neuroscientific research into the worm. Brenner recognised the potential of *C. elegans* as a *model organism* [Brenner, 2003]. A model organism is any organism that is widely and easily studied, typically due to the ease of maintenance of the organism, the biological similarity to more complex organisms such that conclusions are generalisable (i.e. mice to humans), and having desirable and measurable behavioural, genetic or reproductive properties. Other common model organisms are *Drosophila melanogaster* (common fruit fly), zebrafish and mice. Brenner isolated a particular *C. elegans* specimen and distributed genetic clones of this specimen to laboratories worldwide [Sterken et al., 2015]. This specimen was named *N2*, and is frequently referred to as *wild-type*. Genetic clones of N2 are still available to study today, and is often considered a "baseline" *C. elegans* specimen to which the behaviours of altered *C. elegans* specimens are compared.

C. elegans possesses many properties that combine to make it an ideal candidate for neuroscientific study. Foremost, *C. elegans* can reproduce asexually. This means offspring are genetic clones, allowing specific gene lines to be preserved over generations of worms. *C. elegans* specimens are also able to reproduce sexually, meaning that new gene lines and strains, referred to as *variants*, can be introduced without direct genetic manipulation.

C. elegans specimens are also able to endure harsh conditions, such as those required for cryogenic storage [Stiernagle, 1999]. Specimens can be frozen and defrosted with little-to-no damage to the organism and with no discernible change in behaviour [Vita-More & Barranco, 2015]. This allows particular strains to "backed up" for future experiments, or, for distribution to other laboratories for reproduction and validation of experimental results. Compared with other species, maintaining *C. elegans* specimens requires markedly simpler conditions and less space due to their small size and simple nutritional requirements [Stiernagle, 1999]. *C. elegans* can be maintained in laboratory conditions on agar, or in a fluidic environment, with *E. coli* bacteria as a food source. Furthermore, *C. elegans* has a short life and reproductive cycle, and is amenable to genetic manipulation [Chen et al., 2016]. Iterating experiments using genetic modifications can therefore be carried out in much shorter time frames and with higher success rates than in other organisms.

C. elegans has a relatively simple physiology. The worm is often discussed with reference to three anatomical planes: anterior-posterior (head to tail), dorsal-ventral (top to bottom) and left-right. Fully developed worms are approximately one millimetre in length and approximately fifty microns in diameter. The worm has a smooth and unsegmented body, and is approximately rotationally symmetric around the anterior-posterior axis. The worm has a head, with an opening that serves as a mouth, and a tail, where waste products are ejected roughly every fifty seconds [Riddle et al., 1997a]. The internal structure of the worm is two longitudinal concentric cylinders. The outermost cylinder is bounded by the cuticle, and contains primarily neurons and muscles for locomotion. The interior cylinder, separated by the pseudocoelomic space, contains the digestive system and reproductive organs. *C. elegans* does not possess a respiratory or circulatory system. On high-viscosity surfaces the worm tends to lie on either the left or right side, and bends in the dorsal-ventral plane. *C. elegans* has four main muscle bands responsible for locomotion, described as left, right, dorsal and ventral muscles. Additional muscles for defecation and reproduction exist in the posterior of the worm. Approximately 99.9% of *C. elegans* specimens are *hermaphrodite*, and are capable of reproducing asexually. The remaining worms are *male* and are only capable of sexual reproduction with a hermaphrodite.

C. elegans also displays a remarkably diverse range of behaviours for such a simple organism. *C. elegans* exhibits simple behaviours such as eating [Avery & You, 2012],

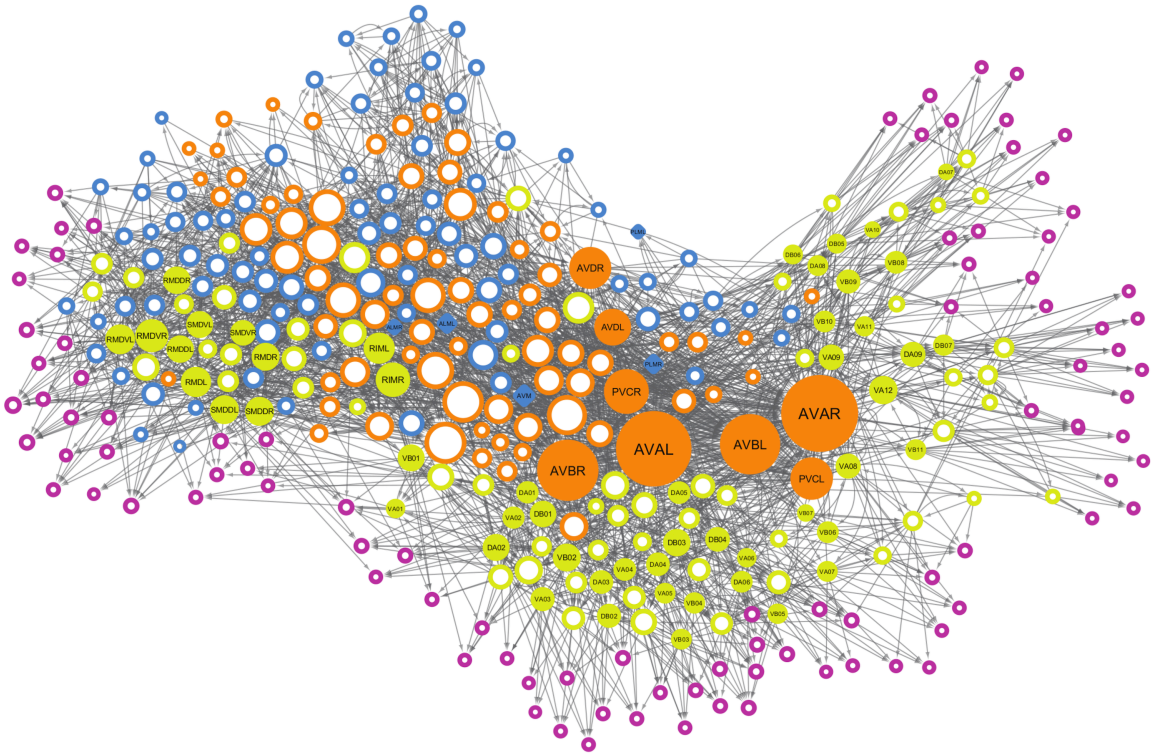


Figure 2.3: The connectome obtained from the *C. elegans* roundworm. Vertices represent neurons, edges represent synapses. Image reproduced courtesy of Emma Towlson [Yan et al., 2017].

defecating [Thomas, 1990], touch sensitivity [Bounoutas & Chalfie, 2007], and avoiding areas of high heat [Wittenburg & Baumeister, 1999] or adverse osmotic conditions [Hart, 2006]. Specimens display a continuous range of locomotion patterns dependent on the viscosity of the environment [Fang-Yen et al., 2010]. On high-viscosity surfaces *C. elegans crawls*, characterised by low lateral velocities and short wavelengths. In low-viscosity environments, such as water, the worm is said to *swim* or *thrash*, characterised by high lateral velocities and longer wavelengths. Beyond this, *C. elegans* also exhibits markedly more complex behaviours such as mating [Gruninger et al., 2006], group behaviours [Ding et al., 2019; Yoshimizu et al., 2018], and even learning [Ardiel & Rankin, 2010; Rose & Rankin, 2001]. These behaviours can be studied under a range of conditions and specimen alterations to investigate hypotheses. Despite its comparatively simple anatomy, the understanding of the physiological, genetic and neural processes that gives rise to many of these behaviours remains incomplete.

Most importantly for this thesis, *C. elegans* is the only organism to have its entire connectome mapped. In 1986, White et al. [1986] successfully mapped the connectome of *C. elegans*, shown in Figure 2.3. This was the result of over ten years of painstaking histography and manual labelling. The connectome was subsequently been updated by

Varshney et al. [2011]. The connectome consists of 302 neurons, with approximately $5,000$ chemical synapses, $2,000$ neuromuscular junctions, and 900 gap junctions [Richmond, 2007]. During reconstruction, White noted that similar patches reconstructed from different worms share the same connectivity structure. It was therefore concluded that all hermaphrodite N2 *C. elegans* worms possess the same connectome. Critically, this means that the neural hardware, on which all cognitive processes are executed, is identical for all N2 specimens. This observation is arguably the genesis of the ideas and methods we develop in this thesis.

While there is evidence that certain types of neurons in *C. elegans* exhibit action potentials [Liu et al., 2011; Lockery & Goodman, 2009; Shindou et al., 2019], it is believed that many of the neurons in *C. elegans* use *graded potentials* [Goodman et al., 1998; Lockery & Goodman, 2009]. Graded potential neurons do not produce the all-or-nothing action potentials, instead producing a response proportional to the amount of stimulus while the stimulus is applied. Graded potentials can also be highly localised within the neuron, changing the potential of the neuron only in the vicinity of the input, unlike the whole-cell action potentials. The molecular mechanism through which potentials are controlled and transmitted between neurons is however largely the same as introduced in Section 2.3.

The neurons in the *C. elegans* connectome are not arranged into a single neuropil region. Instead there is a cluster of neurons in the head of the worm, there are then motor and interneurons throughout the length of the worm, and a small cluster of neurons at the tail of the worm. The position and morphology of neural processes, as well as the positions of synapses, are highly repeatable across *C. elegans* specimens. Neurons can be roughly classified into four categories: *motor neurons* which connect to and innervate the muscles, *sensory neurons* which receive a range of sensory inputs including proprioception, mechanosensation, and chemosensation, *interneurons* which transmit signals between neurons and perform intermediate processing, and a catch-all *polymodal neuron* class, for neurons that perform a range of functions or do not fall into one of these categories. Each neuron is named, with names being a structured string of three to five characters. Neurons that are rotationally symmetric have an additional `R`, `L`, `V` or `D` indicating right, left, ventral and dorsal positioning respectively. For instance, `AVAR` and `AVAL` are rotationally symmetric interneurons. Neurons, typically motor neurons, may additionally have a number after the name indicating their relative position along the worm. For instance, `VB1` is anterior to `VB2` on the ventral side.

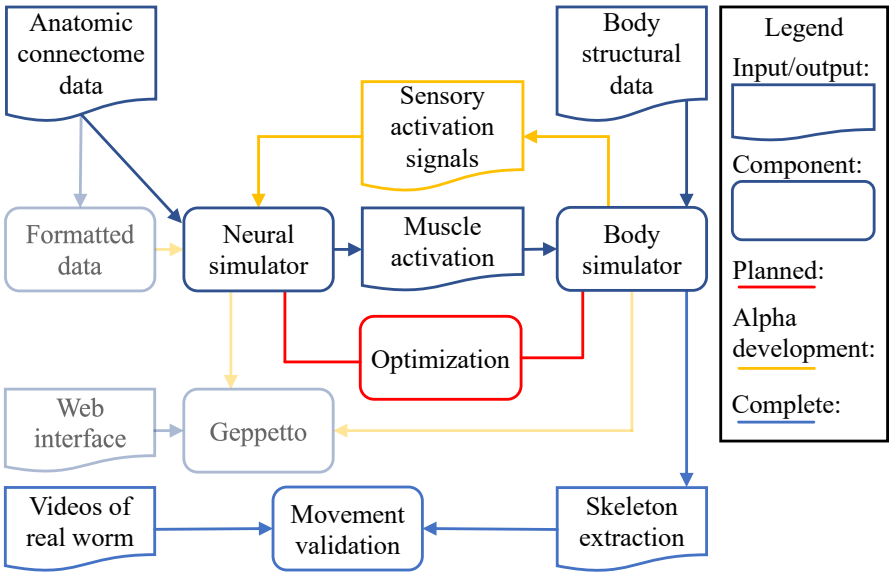


Figure 2.4: Diagram (adapted from Sarma et al. [2018]) reflecting the community planned development pipeline for *C. elegans* simulation. Greyed out components are not generally considered in this thesis. The coloured status of components are as categorised by OpenWorm [Sarma et al., 2018; Szigeti et al., 2014]. OpenWorm discuss c302 [OpenWorm, 2018] and Sibernetic [Palyanov & Khayrulin, 2015] as the neural and body simulators. In this thesis we consider the neural simulators presented by Wicks et al. [1996] and Rahmati et al. [2016], and the body simulator WormSim [Boyle et al., 2012]. We discuss this selection more in Section 4.1.

***C. elegans* Community** There is a large community of researchers focused on *C. elegans*, and has been studied in wet laboratories around the world for decades. However, there is a growing number of researchers studying *C. elegans* computationally, in so-called "web laboratories" [Keshavan & Poline, 2019]. Recently, an entire journal series has been dedicated to computational modelling of *C. elegans* [Sarma et al., 2018]. This thesis is part of that collective effort, and aims to bring modern machine learning tools into the conversation. Figure 2.4 shows a flowchart, adapted from Sarma et al. [2018], outlining the organisation of topics currently being considered by the *OpenWorm* group [Szigeti et al., 2014]. We tackle many of these topics as part of this thesis.

2.5 Data & Acquisition

Data are at the core of any experimental investigation, and hence so to are the methods and techniques to acquire data. Since the early work of Galen, neuroscientists have designed experiments to isolate certain behaviours, such that any changes to the brain can be analysed by analysing any changes in these behaviours. In contrast, Cajal used high-resolution image

data to investigate the structure brain and draw conclusions from the known function of the neural circuits. Both methods quantify the dependencies between physiological variables that can be observed, and hypothesises about the underlying dependencies between what cannot be observed. Similarly, data lies at the heart of Bayesian statistics, refining our prior beliefs about the world. These models and inferences are then used to estimate and reason about what cannot be observed. This similarity is at the heart of this thesis. In this section, we introduce several data modalities. This data, denoted $\mathbf{y}$, will ultimately appear to the left of the conditioning bar in the likelihood of a given model or hypothesis, $p(\mathbf{y}|\mathcal{M})$.

2.5.1 Neural Data

The most straightforward method is to directly observe the membrane potential dynamics of every neuron. If we could obtain directly time-series observations of the whole neural state, analysing the brain becomes a regression task. Therefore, we start by discussing how we can observe the neural state directly.

2.5.1.1 Electrodes & Clamps

Galvani discovered that it was possible to activate muscles by injecting electrical current into the nerves of frogs legs. The first record of the reverse operation, measuring the current or potential inside a neuron, is attributed to Edgar Adrian, who was awarded the 1932 Nobel Prize in Physiology for his work [Adrian, 1926; Garson, 2015]. Adrian measured the electrical dynamics within a single neuron by recoding neural potentials using a *Lippmann electrometer*. Over the following decades glass, platinum and stainless steel *microelectrodes* were developed [Kita & Wightman, 2008]. A microelectrode is a thin piece of conductive material that is inserted into the body of a neuron that measures the potential in the cell body relative to the extracellular potential. This signal is then amplified through an infinite impedance amplifier such that the probe draws negligible current, minimally affecting the electrical properties of the neuron. Using such electrodes, Alan Lloyd Hodgkin and Andrew Fielding Huxley gathered the data used to develop the Hodgkin-Huxley model of neural dynamics [Hodgkin & Huxley, 1952]. Arrays of extracellular microelectrodes can be used to monitor thousands of electrodes simultaneously [Obien et al., 2015; Spira & Hai, 2013]. However, these arrays can be imprecise, recording the aggregate dynamics of volumes of tissue as opposed to individual neurons.

A more complex method for investigating the potential dynamics of neurons is *patch clamping* [Neher & Sakmann, 1992; Zhao et al., 2008]. A clamp differs from an electrode as it allows the voltage or current in the cell to be fixed, or clamped, to a specific value. The other quantity is then free to vary and be recorded. For instance, many ion channels are only active in a narrow region of membrane potentials, and hence the voltage clamp fixes the membrane potential to this value and the current in different parts of the cell can be recorded. This is done using three electrodes: an intracellular electrode for measuring the control variable, an extracellular electrode for measuring the reference potential, and a third electrode to supply current or voltage such that the controlled variable takes the target value [Molleman, 2003]. All of these measurements can then be analysed. Clamps allow the electrophysiological properties of neurons to be manipulated and measured under specific conditions.

However, electrodes and clamps are highly invasive, and must be manually inserted into the organism. This is a highly technical, time-consuming, error-prone and resource-intensive procedure. Further, it may alter or limit the range of behaviours that can be expressed by the specimen. While experiments using these methods may permit limited model development, refinement and parameter estimation; brain-wide measurement using these methods is not logistically feasible.

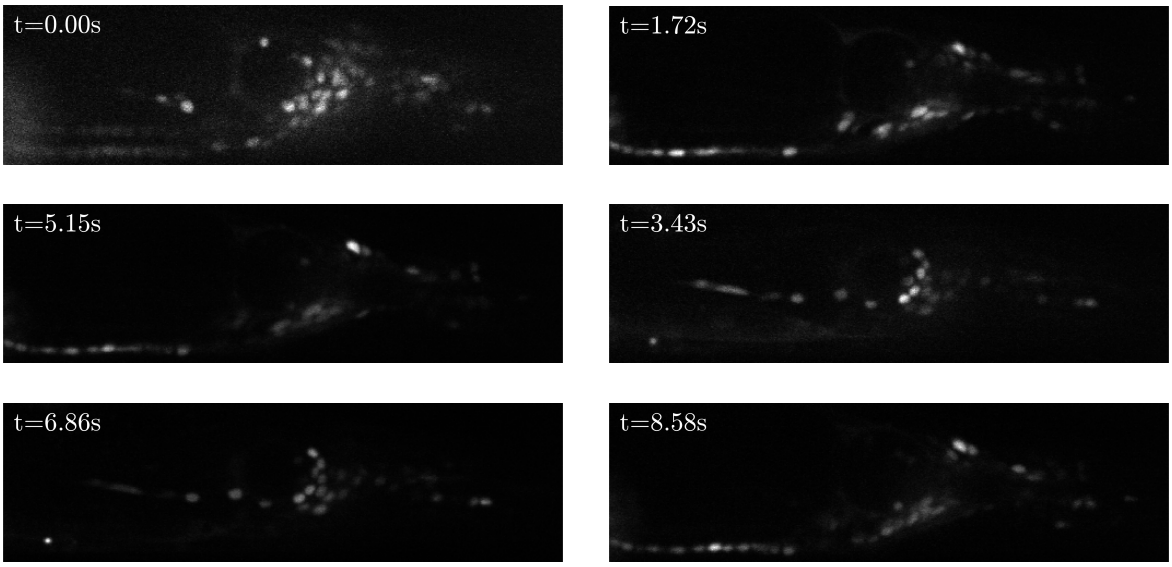
2.5.1.2 Calcium Fluorescence Imaging

An alternative acquisition method is *calcium fluorescence imaging*, often referred to as just calcium imaging [Chung et al., 2013; Stosiek et al., 2003]. Calcium imaging promises high-frequency, single-neuron activity recordings by exploiting the close relationship between ion and potential dynamics [Brini et al., 2014]. Crucially, in small, transparent organisms, such as *C. elegans*, calcium imaging can be performed non-invasively. Action and graded potentials are primarily driven by the controlled flow of charged ions, such as calcium, through the cell membrane mediated by voltage-dependent gates, in turn changing the intracellular ion concentration. The deviations in concentration may be as large as four orders of magnitude. The change in calcium concentration is therefore a covariate of membrane potential. Observing the intracellular calcium concentration (or a correlate thereof) is therefore one of the most important tools for inspecting neural function [Lin & Schnitzer, 2016].

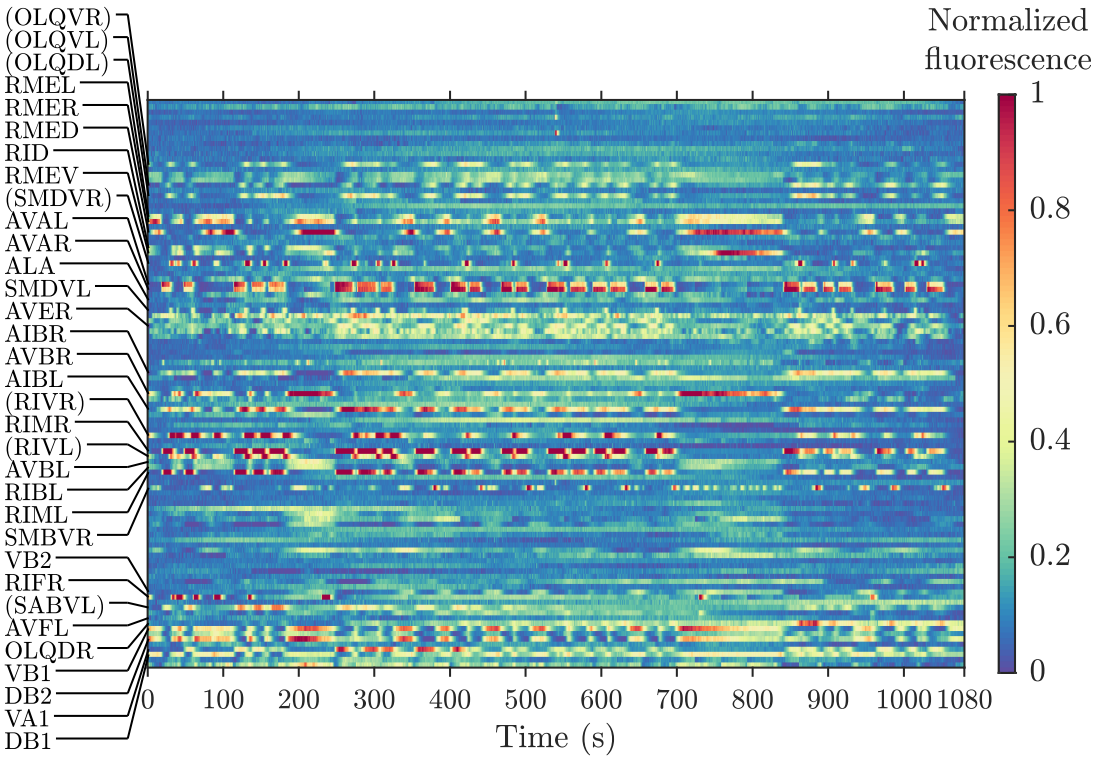
The core method behind calcium imaging is the introduction of a fluorescent molecule or protein into the cell. These molecules release a photon when binding to a calcium ion with a known wavelength, typically in the visible light range [Grienberger & Konnerth, 2012; Stosiek et al., 2003]. Increases in calcium concentration cause an increase in the amount of binding, and hence an increase in the level of fluorescence. Neurons with increased neural activity have spikes and troughs in calcium concentration, and therefore the level of fluorescence rises and falls, causing neurons to *flash*. These flashes can be recorded using a microscope as a bright region, shown in Figure 2.5a. The relative intensity can then be recorded and translated into a time-series, and can be analysed as a surrogate of neural activity. Research in calcium imaging is dedicated to developing molecules that fluoresce more brightly, increasing the signal-to-noise ratio, reducing the time delay between changes in calcium concentration and fluorescence [Dana et al., 2019; Inoue et al., 2019; Zarowny et al., 2020], and developing more sophisticated, higher-resolution, and microscopes with a wider field of view [Sofroniew et al., 2016; Terada et al., 2018].

Calcium indicating chemicals or proteins can be introduced into an organism via chemical infiltration or genetic alteration [Russell, 2011]. Chemical infiltration is the simplest method for introducing the indicator, injecting the dye into the region of interest and then naturally infiltrating into cells, or, directly injecting into the cells of interest. However, this process is time consuming, difficult, error-prone, is not specific to a particular neuron type, and produces variable results. A more reliable method is genetic alteration, and is the preferred method for many *C. elegans* researchers. The organism is transgenically modified such that the fluorescent protein is produced inside the cell. This can be done easily, reliably, and in such a way that the protein is only produced in a selected type of cell. The genetic mutant can then be preserved, and genetic clones that also express the required proteins can be produced. This improves experimental reproducibility and reduces the overhead of conducting experiments. Commonly used genetically encoded calcium indicators (GECIs) include `GCaMP6` and `FIP-CB`$_{\texttt{SM}}$. Often, they are paired with measurements of a second red fluorescent protein (RFP), such as `mCherry`, that is insensitive to calcium concentration, and provides a stable baseline signal for tracking and normalization.

Converting raw calcium fluorescence observations into usable fluorescence traces requires a non-trivial amount of post-processing, however. Fluorescent neurons are visible as bright



(a)



(b)

Figure 2.5: Raw and processed calcium florescence imaging data [Kato et al., 2015]. Figure 2.5a shows the raw florescence acquisitions, where neurons are shown as florescent "blobs." Figure 2.5b then shows this data after post-processing. Plotted is the deviation of the fluorescence from a baseline level. The neurons listed on the left indicate those neural traces that were positively identified by an expert annotator. Neurons in brackets indicate that the annotator was confident but not sure of the neural identity. These annotations are used to define the permutation matrix mapping neural identities to calcium traces. Data reproduced with permission from Manuel Zimmer [Kato et al., 2015].

blobs, shown in Figure 2.5a. Therefore, to create fluorescence time series, as in Figure 2.5b, we must post-process these video frames using a computer vision pipeline [Giovannucci et al., 2019; Stringer & Pachitariu, 2019]. Each frame is first segmented into fluorescent regions. These regions are then tracked between frames to create a time series. Either the average or peak intensity from each segmented region of interest in each frame is then extracted to create a scalar-valued fluorescence time series. As this fluorescence is only a surrogate for calcium, to convert back into a calcium concentration the emission process must be calibrated against known calcium concentrations. Furthermore, on a high-level, calcium concentration transients, and hence the fluorescence, are a time-smoothed covariate of the potential (reflected in the calcium dynamics model presented in (2.16) and (2.17)). Therefore, fluorescence measurements are less sensitive to short, sharp changes in potential. This is a further motivation behind the use of Bayesian inference in physiologically plausible models, as this allows highly structured high-frequency dynamics to be estimated from lower frequency and intermittent observations.

Finally, expert annotators are then required to identify the specific neuron in the *C. elegans* connectome that a fluorescence trace corresponds to. This identification is guided by the organism behaviour, and the expected activity and position of the fluorescent blob. Only a subset of the total traces, typically in the order of fifty, can be positively identified. Crucially, as the number of neurons imaged increases, or the identity of more neurons is assumed, the possibility of incorrectly identifying a neuron increases. These false identifications could lead to spurious results, or, require additional post-processing to detect and correct [Gauthier et al., 2018]. To automatically identify the remaining traces requires solving a challenging non-square permutation problem, probabilistically assigning latent neural identities to unlabelled fluorescent traces [Linderman et al., 2017; Mena et al., 2018].

Calcium imaging provides us with a non-invasive method of estimating the intracellular calcium concentration. However, only a subset of the neurons in an organism can be imaged or positively identified by expert annotations. Hence there is inherently an inference problem, inferring the activity of the unobserved or unidentified neurons. Furthermore, since it is ion *gradients* that generate and transmit signals the calcium concentration is not constant across the neuron, and therefore, the estimated fluorescence is simply a point estimate of the calcium concentration in the soma of the cell [Ali & Kwan, 2019]. However, these

estimates have been well correlated with behaviours [Kato et al., 2015; Scholz et al., 2018] and hence are likely to be sufficiently informative for our needs. Wider field of view microscopes [Sofroniew et al., 2016; Terada et al., 2018] and high-performance multicolour calcium stains [Akerboom et al., 2013; Inoue et al., 2019; Sands et al., 2018] promise to partially alleviate these issues. It is also important to note here that *voltage fluorescence imaging* techniques are being developed [Peterka et al., 2011; Knöpfel & Song, 2019]. Voltage imaging produces a fluorescence signal as a function of the instantaneous neural potential. This means potential imaging can detect potential transients with higher frequencies than calcium imaging. Furthermore the need to explicitly model calcium is removed, and observations can be succinctly integrated into the methods we discuss later. Although voltage imaging is not currently developed enough to be deployed at the scale required, it offers an promising opportunity going forwards.

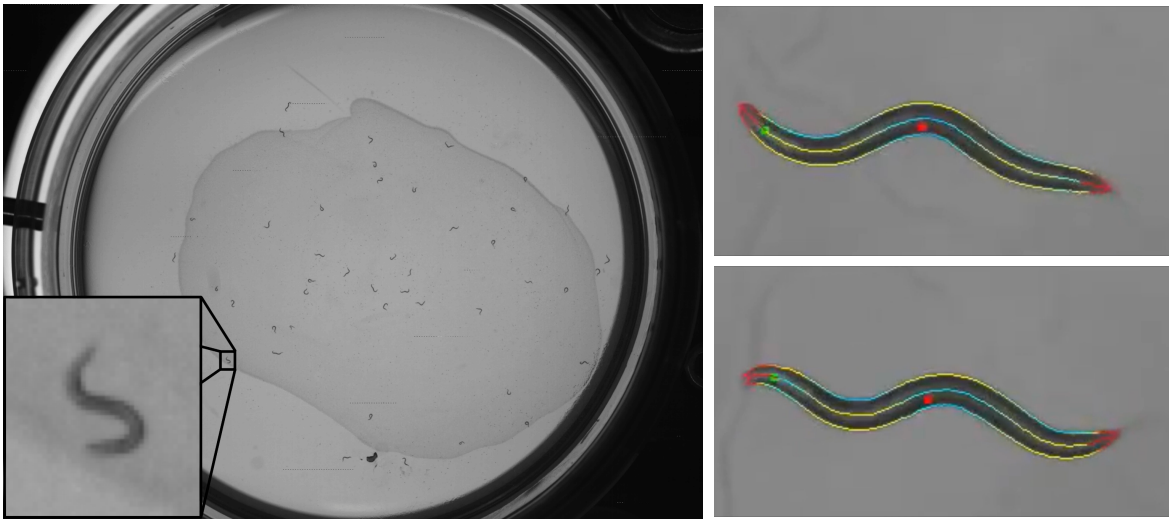
2.5.2 Organism-Level Data

We conclude this section by considering organism-level sources of data. While we are ultimately interested in modelling the neural dynamics, there are many other informative observations that can be recorded simultaneously with calcium imaging. These data modalities may be less directly informative of the underlying neural state of the worm, however, they may provide critical and complementary information about the regions of the brain that were not imaged using calcium fluorescence imaging.

2.5.2.1 Morphology Information

The first additional source of data is the morphology and locomotion of *C. elegans* over time. It is estimated that 113 of the 302 *C. elegans* neurons are directly involved in locomotion [Altun & Hall, 2011], and hence it is reasonable to hypothesise that the locomotion will provide at least a degree of information about the state of these neurons. Moreover, many of the neurons involved with locomotion, including the majority of motor neurons, are positioned throughout the body of the worm and hence are not recorded using calcium imaging. Therefore the physical configuration of the worm is the most direct modality for obtaining information about these neurons.

The locomotion of the worm is used throughout neuroscientific analysis of *C. elegans*. Changes in locomotive patterns are used as a feature to analyse the response to neural modification or drug treatment. Measuring the physical configuration of the worm does not



(a) **(b)**

Figure 2.6: Different recordings of *C. elegans* body poses. Figure 2.6a shows low-resolution, multi-worm recordings. Image reproduced with permission from Alex J. Yu [Ardiel et al., 2017]. Figure 2.6b shows high-resolution, single-worm recordings, made publicly available as part of Yemini et al. [2013]. Annotations indicate the centreline, edges, head (green dot) and ventral side (red dot).

require any specimen preparation and only requires a standard desktop light microscope. As many as 100 individual specimens can be recorded simultaneously using relatively inexpensive and widely available hardware and software [Husson et al., 2005; Javer et al., 2018; Ramot et al., 2008; Swierczek et al., 2011]. The raw video data is then passed into a processing pipeline that extracts the time-evolution of the shape of each worm. From the low-dimensional shape representation, features such as the forward velocity and curvature can be extracted and analysed between worm phenotypes, treatments or stimulus patterns.

To acquire this data, specimens are placed on an agar-filled petri dish under a video-recording equipped microscope or high-resolution video recording device. There are two common imaging configurations. Wide field of view microscopes can be used to record the whole petri dish and many specimens simultaneously, shown in Figure 2.6a. However, these recordings are often at low spatial and temporal resolutions. Alternatively, high-fidelity, narrow field of view microscopes can be used to record a single worm in more detail, such as shown in Figure 2.6b. A motorised stage is then used to keep the worm within the field of view. The type of data required often determines which type of acquisition is used.

The raw video feed is then passed into the image processing software. The video frames are first segmented and specimens are tracked between adjacent frames. Segmentation and tracking is generally an easy operation, as the worms have regular morphology and are

dark in colour compared to the uniform and bright background. If a motorised stage is being used, the centroid of the worm is computed and the stage is automatically moved. From these segmentations, the worm is then skeletonised, reducing each frame to a series of control points representing the centreline of the worm. These control points are then analysed to produce estimates of the desired features, such as forward velocity or curvature. The low-dimensional skeletons and features are then saved to disk for later analysis.

These pipelines do much of the data analysis "heavy lifting," providing low-dimensional, structured, interpretable and highly informative data at each time step. Unlike calcium imaging, data can be gathered for hundreds of worms simultaneously with little input from the experimentalist. Hence, morphology data is a promising modality for analysing the interactions of multiple organisms in response to complex external stimuli. It also offers a cheap and flexible acquisition modality that may be more accessible to experimentalists and theoreticians than full calcium imaging apparatus. We discuss using this modality at length in Chapter 5.

2.5.2.2 High-level Behavioural Features

The final data modality we discuss is high-level behavioural features. High-level here refers to behavioural motifs that are only identifiable at an organism level and are non-trivial to identify. Such features include omega turns, reversals, or group interactions. Identifying and labelling many of these motifs can be performed automatically by worm tracking software, however, more complex motifs may require expert annotation. These behaviours are identified at discrete time points, and hence are a *very* coarse encoding of the neural state. However, for critical behaviours, matching simulated behaviours to real behaviours may provide a more flexible and informative analysis method. For instance, a simulated worm may only reverse once every minute on average, whereas the real worms reverse three times per minute, indicating that the model is poor. This provides a more qualitative measure of how different the simulated behaviour is to real worms. Although this behavioural information is unlikely to be sufficient in isolation for meaningful inference, it is a highly informative feature that will help analyse and refine models in a more qualitative and understandable way than other modalities.

2.6 Neurophysical Models

Scientific investigation often relies on developing and analysing mathematical models of the system of interest. Specifying a model makes a concrete statement about the relationship between observed and unobserved quantities of interest. These models are often expressed as mathematical equations, or, are often more elegantly specified for complex systems through programmatic simulators. In this thesis we consider models defining the relationship between physiological variables, such as the unobserved neural membrane potential and observed calcium fluoresce. Using mathematical models will allow us to reduce the task of analysing the intimidatingly vast and complex connectome, to analysing a simple parametrised model. The central theme in this thesis is that automating the analysis of observed data and performing inference in these models provides a more effective method for performing neuroscience research.

In this section we introduce, in detail, a number of neurophysical models and simulators that we will use repeatedly. We introduce the necessary background material on each model here as we will use different combinations of components in Chapters 4, 5 and 6. We then discuss in Chapter 3 the tools provided by statistics, probability and machine learning that we use to analyse these models.

2.6.1 Single Compartment Neural Model

The first model we introduce is a model of neural potential dynamics for networks of neurons. This model was first introduced by Wicks et al. [1996], and we refer to this model herein as the *Wicks model*. As we are primarily interested in how the neurons of the connectome combine to create behaviours, this model will form the core of much of the material we present in this thesis.

The Wicks model formulates the neural membrane potential dynamics as a set of coupled ordinary differential equations, and specifies the state-dependent time derivative of neural membrane potential. This model simplifies analysis by ignoring the spatial variation over the volume of the neuron by modelling neurons as *single compartments*. This is a relatively strong assumption as it is known that neural states vary spatially and equilibrate over time scales that are not negligible compared to the bulk neural dynamics. However, simulating this spatial variation dramatically increases the complexity of simulating neural dynamics (see the NEURON package [Hines & Carnevale, 2006]). For further information on each

of the terms, expressions and constants introduced here, we refer the reader to the original work [Wicks et al., 1996].

The model can analyse an arbitrary number of neurons, denoted $N \in \mathbb{Z}_+$. The vector of neural potentials, $\mathbf{v}_t \in \mathbb{R}^N$, is the state of the model at time t. The membrane potential of the n^{th} neuron is individually denoted as $v_{t,n}$. The derivative term for each individual potential can be written by modelling the neuron as a resistor-capacitor electrical circuit:

$$\frac{\mathrm{d}v_{t,n}}{\mathrm{d}t} = \frac{1}{R_n C_n}(v_{\text{leak}} - v_{t,n}) + \frac{1}{C_n} I_{\text{ext},t,n} + \frac{1}{C_n} \sum_{k=1}^{N} I_{\text{syn},t,n,k}. \tag{2.1}$$

The three terms in this expression represent the passive leakage current through the cell wall driven by the membrane potential, the external current injected into the cell, and the current injected into the cell through junctions with other cells. Note here that the n^{th} neuron is the *postsynaptic* neuron, while the k^{th} neuron is the *presynaptic* neuron. While this is somewhat counter-intuitive, it is easiest to consider this model as simulating *incoming* current, and that current flows from positive to negative potential. We now describe each of these terms in more detail.

The first term represents the passive leakage current of ions into and out of the cell by considering the cell membrane as a resistor-capacitor circuit. The electrical potential outside of the cell is denoted v_{leak}, and is assumed throughout to be equal to -35mV. The term $(v_{\text{leak}} - v_{t,n})$ is therefore the membrane potential. The rate of change of neural potential is equal to the membrane potential divided by the *membrane resistance*, R_n, and the *membrane capacitance*, C_n. This term is the homogeneous component of the differential equation, and admits an exponential decay solution returning the intracellular potential to the extracellular potential.

The remaining terms are then the inhomogeneous components that drive the differential equation. The second term is external current injected into the neuron or drawn out of the neuron. This term provides a mechanism for stimulating the network, or, connecting the network to other non-neural systems. We will use this term later when considering proprioceptive feedback and muscular stimulation.

The final term is the current injected into the cell through synaptic connections. This term decomposes as the sum of the current through electrical and chemical synapses between the n^{th} and k^{th} neurons:

$$I_{\text{syn},t,n,k} = I_{\text{syn},t,n,k,\text{gap}} + I_{\text{syn},t,n,k,\text{chem}}. \tag{2.2}$$

The self-stimulation is defined as zero, $I_{\text{syn},t,n,n,\text{chem}} = I_{\text{syn},t,n,n,\text{gap}} = 0$. The synaptic transmission model is graded transmission model [Lockery & Sejnowski, 1992] with reversal potentials [Wicks et al., 1996].

Gap junctions are assumed to behave as Ohmic resistors, and as such the current through parallel gap junctions can be defined as:

$$I_{\text{syn},t,n,k,\text{gap}} = \hat{g}_{n,k}\hat{w}_{n,k} \times (v_{t,k} - v_{t,n}), \tag{2.3}$$

where $\hat{g}_{n,k}$ is the conductance of a single gap junction and $\hat{w}_{n,k}$ is the number of parallel gap junctions between the n^{th} and k^{th} neurons. This model assumes that the conductance of gap junctions in parallel is additive. The conductive properties of individual junctions are assumed to be constant, with a conductance of $\hat{g}_{n,k} = 5\text{nS}$. For neurons that are not connected, $\hat{w}_{n,k} = 0$, and hence there is no current.

Chemical synapses cannot be accurately modelled as Ohmic resistors, and are instead non-linear operators that zero-out low signals, saturate at high signals, and are approximately linear in the midrange. The current between the n^{th} and k^{th} is defined as:

$$I_{\text{syn},t,n,k,\text{chem}} = w_{n,k}g_{n,k}(v_{t,k}) \times (E_{\text{syn},n,k} - v_{t,n}), \tag{2.4}$$

where $w_{n,k}$ is the number of parallel chemical synapses. The term E_{syn} is the *reversal potential*, and defines the potential at which there is no net current through the synapse. This quantity is assumed to be 0mV for excitatory synapses, and -45mV for inhibitory synapses. As with gap junctions, the conductance of parallel chemical synapses are assumed to be additive. The model assumes that the conductivity of the postsynaptic membrane is a function of the *presynaptic* potential as [Lockery & Sejnowski, 1992]:

$$g_{n,k}(v_{t,k}) = \frac{\bar{g}_{n,k}}{1 + \exp K\left(\frac{v_{t,k} - v_{\text{eq},n}}{v_{\text{range}}}\right)}, \tag{2.5}$$

where v_{range} is the typical range of pre-synaptic potentials, and $\bar{g}_{n,k}$ is the maximum possible synaptic conductivity. The term v_{eq} is the presynaptic *equilibrium potential*, and is assumed to be the potential at which the synapse is most sensitive, and is therefore the potential at which the rate of change of the synaptic conductivity is steepest. We discuss the equilibrium potential in more detail shortly. The constant K is chosen as:

$$K = 2\ln\left(\frac{0.1}{0.9}\right) = -4.3994. \tag{2.6}$$

such that the synaptic conductivity, $g_{n,k}(v_{t,k})$, changes from $0.1\bar{g}_{n,k}$ to $0.9\bar{g}_{n,k}$ over v_{range} (as can be seen by substitution into (2.5)). However, stipulating this value is an assumption, and K is a free parameter that could be learned from data. The expression in (2.4) therefore drives the neural potential to the reversal potential with a strength determined by the presynaptic activation.

We are now able to fully define the potential derivative as:

$$\frac{\mathrm{d}v_{t,n}}{\mathrm{d}t} = \frac{1}{R_n C_n}\left(v_{\text{leak}} - v_{t,n}\right) + \frac{1}{C_n} I_{\text{ext},n} + \frac{1}{C_n}\hat{w}_{n,k}\hat{g}_{n,k}(v_{t,k} - v_{t,n}) + \tag{2.7}$$

$$\frac{1}{C_n}\sum_{k=1}^{N} \frac{w_{n,k}\bar{g}_{n,k}}{1 + \exp K\left(\frac{v_k - v_{\text{eq},n}}{v_{\text{range}}}\right)}(E_{\text{syn},n,k} - v_{t,n}). \tag{2.8}$$

This expression appears intimidating due to the number of terms and operations. However, it is important to note that this is simply a parameterised function of the current neural potential and can be simply implemented as a vectorised operation on the current neural potential.

The only outstanding quantity to calculate is the equilibrium potential, v_{eq}. This potential represents the neural potential at which there is no net current into or out of the neuron, and hence can be considered as a resting potential. The equilibrium potential is a per-neuron constant that is a function of the whole network connectivity. It is an assumption above however that this resting potential defines the most sensitive point of the synapse. The equilibrium potential can be computed analytically by noting that the derivative is zero at equilibrium, $\mathrm{d}v_{t,n}/\mathrm{d}t = 0$, assuming there is no input current, $I_{\text{ext}} = 0$, and setting $\mathbf{v}_t = \mathbf{v}_{\text{eq}}$, implying that the chemical synapses are also at their midrange conductivity, $\bar{g}_{n,k}/2$.

To begin, we first substitute these quantities into (2.8):

$$0 = v_{\text{leak}} - v_{\text{eq},n} + R_n \sum_{k=1}^{N} \frac{w_{n,k}\bar{g}_{n,k}}{2}(E_{\text{syn},n,k} - v_{\text{eq},n}) + \hat{w}_{n,k}\hat{g}_{n,k}(v_{t,k} - v_{\text{eq},n}). \tag{2.9}$$

Pulling the time-variable terms to the left we obtain:

$$v_{\text{eq},n} + R_n \sum_{k=1}^{N}\left(\frac{w_{n,k}\bar{g}_{n,k}}{2} v_{\text{eq},n} + \hat{w}_{n,k}\hat{g}_{n,k} v_{\text{eq},n} - \hat{w}_{n,k}\hat{g}_{n,k} v_{\text{eq},k}\right) = v_{\text{leak}} + R_n \sum_{k=1}^{N} \frac{w_{n,k}\bar{g}_{n,k}}{2} E_{\text{syn},n,k}. \tag{2.10}$$

We can then form this expression into a matrix-vector product, noting that $\mathbf{v}_t$ terms are

only multiplied by on the right hand side:

$$\mathbf{A}\mathbf{v}_{\text{eq}} = \mathbf{b}, \tag{2.11}$$

$$b_n = v_{\text{leak}} + R_n \sum_{k=1}^{N} \frac{w_{n,k}\bar{g}_{n,k}}{2} E_{\text{syn},n,k}, \tag{2.12}$$

$$A_{n,n} = 1 + R_n \sum_{k=1}^{N} \left(\frac{w_{n,k}\bar{g}_{n,k}}{2} + \hat{w}_{n,k}\hat{g}_{n,k} \right), \tag{2.13}$$

$$A_{n,k} = -R_n \hat{w}_{n,k}\hat{g}_{n,k}, \quad n \neq k. \tag{2.14}$$

Noting that $\mathbf{A}$ and $\mathbf{b}$ are constant in time, we conclude (somewhat reassuringly) that the equilibrium potential is indeed constant, and can be solved for by simply pre-multiplying both sides by $\mathbf{A}^{-1}$. This inverse only needs to be computed once prior to execution, as it is constant and is not a function of state. As a result, $\mathbf{v}_{\text{eq}}$ is not a free parameter.

By using (2.8) we can simulate arbitrary networks of neurons as a function of the electrophysical constants (such as conductances and permeabilities) and the inter-neuron connectivity (through the specification of the synaptic weight matrices, or connectomes, $\mathbf{w}$ and $\hat{\mathbf{w}}$). It also makes clear the modelling assumptions that reduce a complex biophysical system to a set of coupled ODEs. The network evolution can then be simulated by using Euler integration or a numerical ODE integrator. For *C. elegans*, N is set to 302, and hence we compute 302 derivatives at each integration step. The free parameters of the model are: $R_n, C_n, v_{\text{leak}}, \hat{g}_{n,k}, \bar{g}_{n,k}, \hat{w}_{n,k}, w_{n,k}, K, v_{\text{range}}, E_{\text{syn},n,k}$, and are all scalar values. The connectivity map for *C. elegans* defines the elements of $\mathbf{w}$ and $\hat{\mathbf{w}}$ that are non-zero [White et al., 1986]. Crucially, it also provides us with a coarse estimate of the overall conductivity by assuming all synapses are equal in weight. The equilibrium potential is computed from these. These parameters are collectively denoted $\boldsymbol{\theta}_{\text{neural}}$. These quantities are typically measured through invasive experimentation, however, we propose later that they can be estimated from non-invasively gathered data.

2.6.2 Tap-Withdrawal Circuit

The neural model introduced by Wicks et al. [1996] is actually introduced for a subset of *C. elegans* neurons, referred to as the *tap-withdrawal circuit*, originally identified in Wicks & Rankin [1995]. This circuit is the subset the connectome that is responsible for the *tap-withdrawal* behaviour. When the petri dish is tapped, specimens reverse locomotion direction for a short period, turn slightly, and then resume forward locomotion. This behaviour is

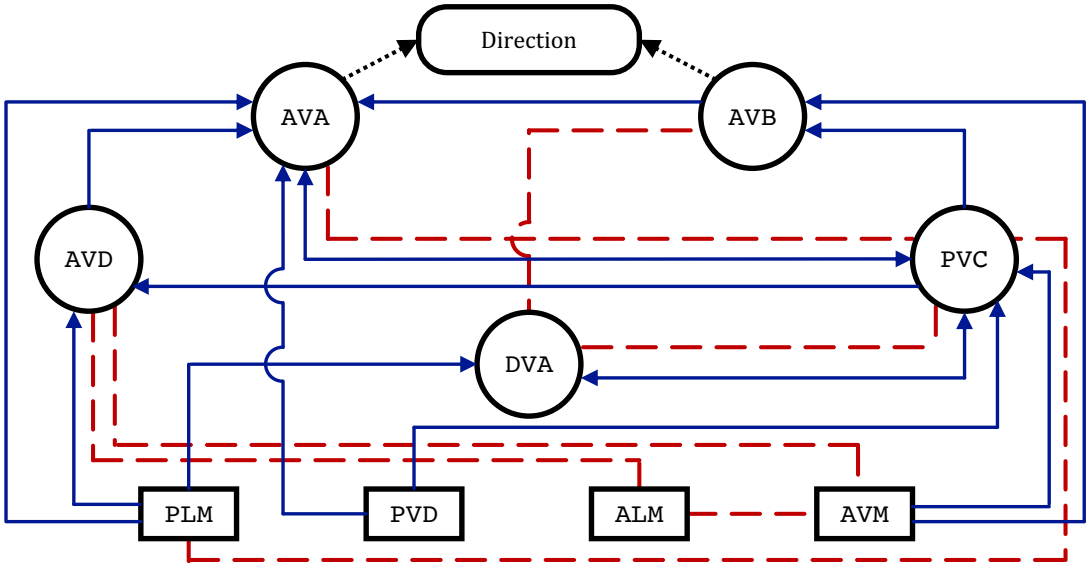


Figure 2.7: Simplified tap withdrawal circuit as defined by Wicks et al. [1996]. Sensory neurons are shown as rectangles and interneurons are shown as circles. Bidirectional gap junctions are shown as red dashed lines and directional chemical synapses are shown as solid blue arrows (neurons pairs with a chemical synapse in either direction are shown using a double headed arrow). We do not picture chemical synapses with low strength. The "gearbox," determining the direction of locomotion, shown as a pill, is a function of the AVA and AVB interneurons.

often used as an informative biomarker for comparing different specimens [Amano et al., 1999; Bozorgmehr et al., 2013; Wicks & Rankin, 1995].

The full tap withdrawal circuit consists of a subset of *C. elegans* neurons that were identified as being active during the tap withdrawal response, and consists of seven sensory neurons and nine interneurons, and is organised into dorsal and ventral sides, with each side consisting of seven neurons, and two neurons centrally orientated neurons. The network is simulated using the neural circuit equations introduced above, where the strengths of each connection are selected through a mixture of hand-tuning and the prior published connectome. A simplified circuit is then created by exploiting the (approximately) symmetrical nature of this circuit to reduce it to an idealised nine-neuron circuit, shown in Figure 2.7. Herein, we only consider the simplified circuit.

The idealised circuit is comprised of four sensory neurons, five interneurons, seven gap junctions and 26 chemical synapses. The tap withdrawal response is encoded by what is described as the *gearbox*, corresponding to the relative potentials of the two main interneurons, AVA and AVB. When AVA is higher than AVB, the worm moves forward. When AVB drops below AVA, the worm is more likely to reverse. The greater the differential, the more likely

the worm is to change from its current locomotion direction. The propensity of the worm is therefore generically expressed as:

$$\text{Propensity to reverse} \propto \int_{t \in \text{stimulus time}} v_{\text{AVB}} - v_{\text{AVA}} \mathrm{d}t. \tag{2.15}$$

This provides a simple metric of how likely a reversal event is. Taps are simulated by providing a short burst of current into the sensory neurons, through the I_{ext} variable. By using known reversal propensities measured from assays of worms, the authors are able to estimate the polarity of synaptic connections (inhibitory or excitatory). This simple relationship provides an idealised link between the neural dynamics and the high-level behaviours of the worm. We will use this simplified circuit in Chapter 6.

2.6.3 Calcium Dynamics

We also highlighted previously the potential to use calcium fluorescence imaging as a non-invasive, high-fidelity method for inspecting neural activity. A model of the temporal intracellular calcium ion concentration dynamics is presented by Rahmati et al. [2016]. This model describes the change in intracellular calcium ion concentration as a function of neural membrane potential. We denote the calcium concentration of the n^{th} at time t as $c_{t,n}$. The net flow of calcium into neurons is defined by the following inhomogeneous ODE:

$$\frac{\mathrm{d}c_{t,n}}{\mathrm{d}t} = -\kappa_n I_{Ca,t,n} - \frac{c_{t,n} - c_{\text{base}}}{\tau_{Ca}}. \tag{2.16}$$

The quotient term is the homogeneous part of the differential equation, representing the diffusion process that equilibrates the intracellular concentration with the extracellular concentration, assumed to be an infinite reservoir of calcium ions with concentration c_{base}. The time constant of this diffusion process is defined as τ_{Ca}.

The inhomogeneous component of the differential equation is the flux of calcium ions driven by the potential difference across the cell membrane:

$$I_{Ca,t,n} = g_{Ca,n} s_{\infty,n}(v_{t,n}) \times (v_{t,n} - E_{\text{Ca}}), \tag{2.17}$$

where v_{Ca} is the reversal potential, $g_{Ca,n}$ is the maximum conductivity of calcium through the cell wall, and $s_{\infty,n}$ is the instantaneous, potential-dependent conductivity multiplier. As with the neural model, changes in conductivity are assumed to be instantaneous. The potential-dependence of membrane conductivity is then represented by multiplying the

maximum conductivity by the fraction:

$$s_{\infty,n}(v_{t,n}) = \left(1 + \exp\left(-(v_{t,n} - v_{1/2})/\rho\right)\right)^{-1} \tag{2.18}$$

where ρ is analogous to v_{range} and $v_{1/2}$ analogous to v_{eq}. The scaling constant κ_n then converts the flux to a change in concentration and is assumed to be constant across neurons.

These equations provide us with a flexible model of calcium dynamics in the neurons as a function of the membrane potential. The fixed constants of these expressions are: $\kappa, \tau_{Ca}, v_{1/2}, \rho, g_{Ca,n}$. These equations can be simulated in a similar fashion to the neural potentials using vectorised computations. We generally refer to the "neural dynamics" and "neural state" as the complete set of neural potentials and intracellular calcium concentrations and their functional relationship.

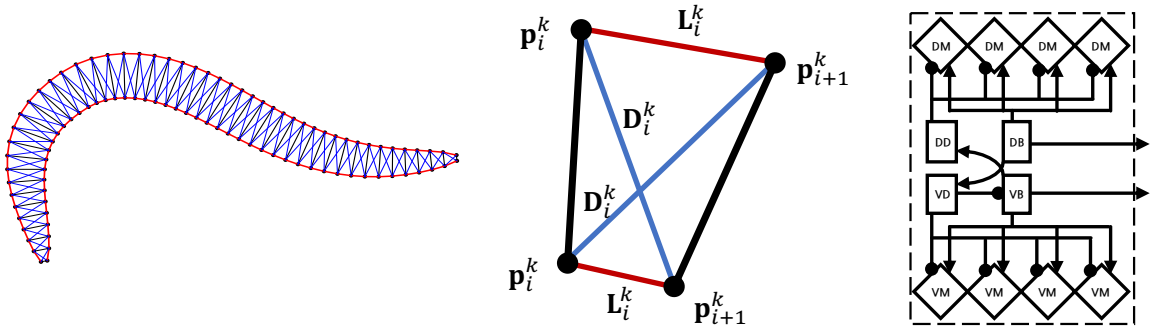
There are drawbacks to this model however. Most importantly, and much like the Wicks model, neurons are modelled as single compartments. Spatial variation in the intracellular calcium concentration is neglected, and the ion concentration in the soma is assumed to be representative of the ion concentration throughout the neural processes. While this limits the absolute fidelity of the model, simulating multiple compartment neurons or individual ion mechanisms would dramatically increase the computational complexity, and hence would decrease inference performance for a reasonable computational budget.

2.6.4 Calcium Fluorescence Observations

The second contribution of Rahmati et al. [2016], also suggested by Grienberger & Konnerth [2012]; Yasuda et al. [2004], is to introduce a model relating the intracellular calcium concentration and the calcium fluorescence intensity. This relationship is described by a saturating Hill-type function [Hill, 1938], a parametrized non-linear function:

$$y_{t,n} = \kappa_F \frac{c_{t,n}}{c_{t,n} + K_d} + d_F, \tag{2.19}$$

were $\mathbf{y}_t$ is the raw n-dimensional observed calcium fluorescence signal at each time, κ_F is the scale and d_F is the offset of the observation process, and K_d is the *dissociation constant*, which is a property of the chemical dye used. This equation relates the intracellular calcium concentration to the observed fluorescence trace. The observed value is then often modelled as being Gaussian distributed around this mean value. We will use this model throughout this thesis as a likelihood term to compare and score simulated calcium fluorescence data to observed calcium fluorescence data.



(a) **(b)** **(c)**

Figure 2.8: Schematic diagram of WormSim, adapted from Boyle et al. [2012]. Muscular units are shown in red, rigid bars are shown in black, and passive springs are shown in blue. Figure 2.8a shows the overall shape of the worm, with the 48 repeating physical units. Figure 2.8b shows the an individual physical unit. The control points are located at the centre of the bars, which, coupled with the angle and the predefined profile of the worm, define the end points of the bars to which muscles and passive springs are attached. Figure 2.8c shows the repeating neural units. Interneurons are shown as squares, and motor neurons are shown as diamonds. Excitatory connections are shown as arrows and inhibitory connections are shown with round edges. There are four physical units per neural unit, and hence there are twelve neural units in the worm.

2.6.5 WormSim

The final model we discuss is a model of the locomotion and physical dynamics of *C. elegans*. The simulator, *WormSim*, was introduced by Boyle et al. [2012] as a computationally tractable simulator for studying the effect of proprioceptive feedback and environment viscosity on the gait and locomotive patterns of free-roaming *C. elegans* specimens. On a high level, WormSim simulates the locomotion of *C. elegans* using an assembly of damped rods and springs, as shown in Figure 2.8a and 2.8b. Contractile forces, representing muscles, can be applied to certain elements. This contraction is driven by a simplified neural architecture.

The simulator represents the body using 48 repeating *physical units*, shown in Figure 2.8a. The worm is further organised into 12 repeating *neural units*, an example of which is shown in Figure 2.8c. Each neural unit contains four physical units, each driven by an idealised pair of dorsal and ventral motor neurons. The worm is constrained to move in two dimensions, and hence best reflects the motion of *C. elegans* on high-viscosity surfaces such as agar. There are *many* parameters, functional relationships and assumptions as part of WormSim, and hence we do not list them all exhaustively here. Instead, we introduce the core ideas of the model, and refer the reader to the original text for more detail [Boyle et al., 2012].

The physical units are an assembly of rigid rods, springs, dampers, and contractile elements. Each rod is defined by the extrinsic x and y position of the centre of the rod, and

the angle of the rod, denoted θ, and the velocity of each of these terms. The rods are orientated transversely to the centreline of the worm. These rods recreate the cell tension and internal pressure in real specimens, and serve to maintain the overall shape of the worm body. The 49 rods (bounding the 48 physical units) are regularly spaced and centred on the centerline of the worm. There is then a pre-defined sinusoidal worm cross-sectional profile, which defines the width of each rod as a function of its position along the centreline of the worm.

Between the ends of adjacent rods are 48 muscles, on both dorsal and ventral sides, shown as red lines in Figure 2.8b. The muscles are represented as damped springs that are able to exert a contractile force. The spring force exerted on the endpoints of the rods is then a function of the extension or compression of the muscle from a base length. Additional contractile force can be applied representing muscular activation, where the activation level is a Hill-type function of the potential at the neuromuscular junction [Hill, 1938]. Finally, springs cross each physical unit providing passive body forces that resist bending, shown in blue in Figure 2.8b. Muscles are driven by 96 values representing the instantaneous potential in each of the 48 dorsal and ventral motor neurons. The muscular potential is a leaky, low-pass filtered neural signal, discussed below. This defines the musculature and passive body forces in the worm, and together with the kinetic components, defines the *physical model*.

Each neural unit drives four physical units using four interneurons: one `DB` and one `VB` class excitatory motor neurons, and one `DD` and one `VD` class inhibitory neuron. Each neuron is modelled as a binary variable, such that the neuron is either "high" or "low," and assumes instantaneous transitions between states. The four dorsal and ventral muscles are innervated by the `DB` and `VB` class motor neuron respectively. The `DD` and `VD` class neurons track the opposite `B` class potential exactly and inhibit ventral and dorsal motor neurons respectively. This is described as *contralateral inhibition*, such that when one side is contracting the other side is relaxing. All other motor neurons are not simulated. Each neuron flips state when the net current into the cell (provided by proprioceptive feedback and neural input) exceeds a state-dependent threshold, such that each neuron exhibits *hysteresis*, which was found to reduce the amount of small oscillations. Each `B` class neuron also excites the `B` class neurons in the next posterior neural unit. This means that excitation originates in the head and flows down the worm, generating the undulations seen during locomotion. A single `AVB` neuron is simulated, with a constant "on," to drive forward locomotion.

Stretch receptive, also referred to as proprioceptive, feedback in *C. elegans* is currently believed to enter the neural circuit though B class motor neurons [Schafer, 2015; Wen et al., 2012]. This feedback is calculated as a function of the extension (or compression) of muscular units above (or below) their resting length. The amount of feedback contributed by each muscle is computed as a linear function of the extension or compression of the muscle from its baseline length. The total stretch receptive feedback provided to each B class neuron is then a weighted average of the adjacent muscles, where the weight applied decays to zero over N_{SR} muscular units (where $N_{\mathrm{SR}} = 6$). The stretch receptive feedback is converted to a current that is injected into each B class neuron, along with any driving AVB signal, contralateral inhibition and excitation from the preceding unit. We denote the proprioceptive feedback at each time as $\mathbf{r}_t$, computed from the body state and is input into the neural model.

Collectively, the position and velocity of the rods, the muscular activation levels, and neural potentials define the state of the worm. The total physical or body state is denoted $\mathbf{b}_t \in \mathcal{B} = \mathbb{R}^{49\times3\times2} \times \mathbb{R}^{48\times2}$, representing the rods and muscular potentials. The neural state is represented as a 24 dimensional vector (as the D class neurons exactly track the B class neurons), denoted $\mathbf{v}_t \in \mathbb{R}^{24}$. The model dynamics described above define the evolution of body and neural state, denoted $p(\mathbf{b}_t, \mathbf{v}_t|\mathbf{b}_{t-1}, \mathbf{v}_{t-1}, \mathbf{r}_{t-1}, \boldsymbol{\theta}_{\mathrm{body}})$, and is dependent on both the previous state of the body, $\mathbf{b}_{t-1}$, the neural state at the previous timestep, $\mathbf{v}_{t-1}$, and the proprioceptive feedback, r_{t-1}, and the parameters of the simulator, $\boldsymbol{\theta}_{\mathrm{body}}$. The physical model also returns proprioceptive feedback, denoted $p(\mathbf{r}_t|\mathbf{b}_t)$, to the neural model. We note that in WormSim all transitions are deterministic, and hence all the distributions above are Dirac-δ functions.

2.6.6 An Incomplete Picture

Before we proceed it is important to note that these are only *models* of the true process. As such, there are inevitably modelling errors and unexplained physical phenomena. The classic quote, generally attributed to George Box, "all models are wrong, but some are useful" [Box, 1979] suggests that any model used is an imperfect representation of the underlying system, but that this does not preclude useful inferences from being drawn. This is certainly true of the models we discuss in this thesis. Put simply, a neuron is not a single point, potentials and concentrations are not uniform throughout the neural processes, and the physical form of *C. elegans* is not 48 perfectly repeating assemblies of rods and springs.

Therefore, it is important to remain mindful that the models we present and analyse are only approximations of the true processes. These models are designed to trade off the usability and interpretability of the model, with the faithfulness to the true generative procedure. As new interactions that are unmodelled by the current generation of models are discovered, the models we use to analyse *C. elegans* must be updated.

For instance, calcium is only one of several ions used in neurons to create and propagate neural potentials. It is well known that in many cells sodium acts either independently of, or jointly with, calcium ions to generate action potentials [Geduldig & Junge, 1968]. Therefore, only observing calcium means only observing part of the transmission mechanism. However, sodium inspection methods are not as sophisticated as calcium measuring methods, and hence observing sodium (and other more minor ions) in the manner we require remains practically impossible [Rong et al., 2018].

Recent research has also reinforced that the connectome-driven and calcium-driven elements of the brain are just two of a number of equally important, interacting communication mechanisms. A study by McDiarmid et al. [2015] suggests that neuropeptides modulate behaviours and neural dynamics in *C. elegans*. Therefore, we could improve the faithfulness of our model by including a model of the "peptome" as well. While research into the peptome is in its relative infancy, over the coming years new information, data and (parametric) models of peptide behaviour and peptome-connectome interaction will likely be created.

It is therefore important that we remain aware that any model is an incomplete picture of the underlying mechanism through which behaviours originate. This will limit the absolute fidelity of any inferences. However, there is a significant and growing collection of additional layers of complexity continually being discovered that will reduce this modelling error. Hence, there is a bidirectional relationship: the methods and models we present here will help develop a more sophisticated understanding of neural function, which, in turn, will help develop more precise models of neural mechanisms, in turn improving the range of inferences that can be made.

2.7 Discussion

In this chapter we have introduced the neuroscience background required for this thesis. We outlined a brief history of neuroscience, connectomics, basic neural anatomy, and introduced the specimen we discuss throughout this thesis, *C. elegans*. These sections

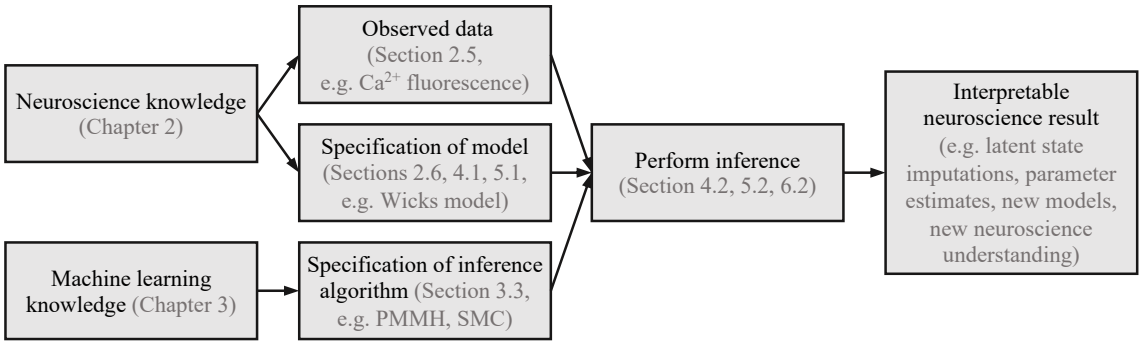


Figure 2.9: Schematic outlining, on a very high-level, how the topics introduced in this chapter and the next chapter combine. In this chapter, we have discussed the neuroscience knowledge, observed data, and mechanistic models that we wish to analyse. In the next chapter, we introduce machine learning, and formally define the inference algorithms required to perform this analysis. In subsequent chapters we then present examples of this analysis. The results of this analysis can then be interpreted to garner new neuroscience knowledge. We could therefore draw a further arrow to feed this new knowledge back into the first box, in what is often referred to as "Box's Loop" [Blei, 2014].

were focused in providing the core intuitions and knowledge that underpin the broad ideas discussed in the introduction, and that are essential for the work we present later. The central theme we drive towards in this thesis is combining interpretable, mechanistic models of neural dynamics, at the scale of whole-connectomes, with Bayesian inference techniques, to provide more powerful analysis techniques to further the understanding of neural systems. These techniques allow us to condition simulations on observed data to infer unobserved physiological quantities of interest, parameter values, and refine models, all from readily observable data. We therefore also introduced the types of data that we will condition on and the neurophysical models which we analyse. In the next chapter we introduce the machine learning and Bayesian inference techniques that we will use to perform this analysis. This is shown schematically in Figure 2.9. In Chapters 4, 5 and 6 we then present our work, performing inference and estimation in neurophysical models of *C. elegans*, using the material introduced in this chapter and in the following chapter.

3

Machine Learning Background

This thesis is primarily dedicated to developing the machine learning approaches required to perform the types of neuroscientific analysis outlined in the introduction. In this chapter we present the necessary background machine learning material. We formally define the foundational principles, techniques and algorithms that we will use throughout this thesis. While this chapter is intended to be an introduction to machine learning for any novice readers, it is not intended to be an exhaustive machine learning tutorial. There are *many* topics and fields that we do not cover here. We therefore instead refer the reader to more complete textbooks for a general and wide-reaching introduction to machine learning [Barber, 2012; Bishop, 2006; Boyd & Vandenberghe, 2004; LeCun et al., 2015].

In Section 3.1 we introduce the basic probability theory that underpins the sophisticated machine learning algorithms we discuss later. In Section 3.2 we give a broad overview of the aims and objectives of inference, discuss the variants of inference and estimation, and define some of the terminology used to describe different methods. We then dedicate a significant proportion of this chapter to introducing sample-based *Monte Carlo* inference methods in Section 3.3. We will directly use many of the algorithms introduced in this section in Chapters 4 and 5 to perform inference in neural simulators. We briefly discuss optimisation and variational methods in Section 3.4. These methods provide both contrasting and complementary tools to the full inference methods discussed prior. We conclude this chapter in Section 3.5 by introducing neural networks and flow-based approximators.

3.1 Introduction to Probability

We begin our review of machine learning literature with an introduction to the fundamentals of probability. The calculus of probability provides us with a set of tools which we will use to analyse neural models. A good understanding of how we denote probabilities, their interpretation, and basic probability manipulations is therefore vital to engage with the more advanced material presented later in this thesis. Any reader who is already comfortable with

probability can safely skip this section. We note that many of the foundations of probability theory are rooted in *measure theory* [Halmos, 2013; Tao, 2011]. However, measure theory is beyond the purview of this thesis. Therefore, we note that there are alternative, more precise definitions for much of what we discuss. The definitions and derivations we include are intended to trade-off accessibility and compatibility with the vast majority of machine learning literature, and more advanced, measure-theoretic definitions.

3.1.1 Basic Probability

Probability refers generally to a set of tools for quantifying uncertainty. The value of each roll of a fair dice is uncertain. However we can agree that the dice lands on any one of the six faces approximately one-sixth of the time, or, that each value is rolled with probability one-sixth. The value of a hypothetical roll is referred to as a *random variable*, while each possible value that could be rolled is referred to as an *outcome*.[1] Each individual roll of a dice is described to as an *event*. More generally, an event is where one or more of the random variables is assigned an outcome. Each random variable takes exactly one value during a single event, a property referred to as *mutual exclusivity*. Denoting the value rolled as the random variable X, the probability that a 1 is rolled is denoted $P(X = 1) = 1/6$. Denoting an outcome as x, the *distribution* over outcomes, $P(X = x)$, describes the probability of every possible outcome, as opposed to simply the probability of a particular outcome. If the context is clear this notation is frequently shortened to simply $P(1) = 1/6$ and $P(x)$. The set of outcomes a random variable can take is defined as the *space* of the variable. We will use a calligraphic typeface to indicate the space of a random variable. For the dice example, the set of possible of outcomes is $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$, and $X \in \mathcal{X}$ indicates that the random variable X takes *exactly* one of these values.

Probability, as we know it today, was first formally tackled in the mid 17^{th} century by Blaise Pascal and Pierre de Fermat, as they discussed ways to fairly allocate winnings in a prematurely ended game of chance [Ore, 1960]. The uncertainty over the outcome of the remaining rounds meant that allocating all winnings to the player with the highest score currently was unsatisfactory. Therefore, players should instead be allocated winnings in

[1] As an example of where measure theory goes beyond this definition, a measure-theoretic definition would state that a random variable is a measurable function mapping from outcome space to a real number. This level of precision is not required in this thesis or to engage with much of current machine learning literature.

proportion to the fraction of possible game endings where they go on to win. This notion was the genesis of the probability calculus.

Over the following centuries, many different philosophers and mathematicians proposed and advocated for different formulations, or *axioms*, to explain and make rigorous the concept of probability, to quantify thought experiments, such as the statement "will the ice caps melt within fifty years?" or to analyse the practical aspects of operating under uncertainty, such as the games discussed by Pascal and Fermat. The primary distinction that arises is *Bayesian* and *frequentist* interpretations of probability. Bayesians consider probabilities as subjective statements of belief, and was axiomatised by Cox [Cox, 1946; Jaynes, 1995]. The Bayesian approach prescribes that one is only able to predict the roll of a dice with a confidence of one-sixth. In contrast, frequentists view probability as the long-run average occurrence of an event, under the axioms of Kolmogorov [Kolmogorov & Bharucha-Reid, 2018]. A frequentist would therefore say that, on average, any given side of a die is rolled on one-sixth of trials. This difference, although subtle, leads to dramatic differences in the conclusions that can be drawn under each formulation. We return to discuss further in Section 3.2.

The basic laws governing the manipulations of probability terms are generally referred to as *the probability axioms*. Although there are differences in the origins, interpretations, and specific forms of the axioms presented by Kolmogorov, Cox and others, the mechanistic outcome from both treatments is broadly the same. Nearly all of the probability theory required for machine learning builds on combinations of these rules and their corollaries. We base our exposition here on the Kolmogorov axioms [Kolmogorov & Bharucha-Reid, 2018].[2] The *first axiom* states that the probability of each event is a real number greater than or equal to zero, $P(X) \geq 0$. An impossible outcome is indicated by a probability of zero. The *second axiom* states that the probability a random variable takes any outcome from the set of possible outcomes is one, $P(X \in \mathcal{X}) = 1$ or $\sum_{x \in \mathcal{X}} P(X = x) = 1$. This defines that guaranteed outcomes are described by a probability of one. The *third axiom* extends this by defining that the probability of one of a set of mutually exclusive outcomes occurring is the sum of the probabilities of the individual outcomes. For the dice example, the probability of rolling a one or a two is the probability of rolling a one plus the probability of rolling a two: $P(x = 1 \text{ or } x = 2) = P(x = 1) + P(x = 2)$.

[2] Again, measure-theoretic definitions of these axioms can be defined, but are not necessary for our aims.

The simplest corollary of the axioms presented is that the probability of one outcome is one minus the probability of every other outcome:

$$P(X = x) = 1 - \sum_{x' \in \mathcal{X} \setminus x} p(X = x'), \tag{3.1}$$

where $\mathcal{X} \setminus x$ is the set $\mathcal{X}$ with the value x removed, sometimes referred to as a *set minus* or *set difference* operation, formally denoted $\mathcal{X} \setminus x = \{x' \in \mathcal{X} \mid x' \neq x\}$. The relationship in (3.1) is one of the rules derived by Cox, as opposed to merely a corollary.

The next construct we introduce is a *conditional probability*, denoted as $P(X|Y)$, using a *conditioning bar*, $|$. A conditional distribution is the distribution over the random variable X *given* the value of the random variable Y. Suppose a six is not rolled, denoted $Y = \text{False}$, and hence only five equally weighted options remain. The probability that X takes the value of 1, knowing that X has not taken the value of 6, is therefore $P(X = 1|Y = \text{False}) = 1/5$.

However, we have used a sleight of hand here. Conditioning on $X \neq 6$ changes the probability of $X = 6$ to zero, and so, naively speaking, the distribution over events is instead therefore $\hat{P}(X|X \neq 6) = \text{Cat}\,(1/6, 1/6, 1/6, 1/6, 1/6, 0)$, which does not sum to one, and is described as the distribution being *unnormalised*. Unnormalised distributions are often indicated using a hat, $\hat{P}$. It is easy to *normalise* the distribution by simply rescaling the probability of the outcomes such that the sum is one:

$$P(X = 1|X \neq 6) = \frac{\hat{P}(X = 1|X \neq 6)}{\sum_{x' \in \mathcal{X}} \hat{P}(X = x'|x' \neq 6)} \tag{3.2}$$

$$= \frac{1/6}{1/6 + 1/6 + 1/6 + 1/6 + 1/6 + 0} = \frac{1}{5}, \tag{3.3}$$

matching the result when using our intuitions earlier. The denominator in this fraction is referred to as the *normalising constant*. As we discuss later, computing the normalising constant is rarely straightforward, and much of probabilistic machine learning is dedicated to computing the normalisation constant, or, developing methods that avoid computing the normalisation constant altogether.

Conditional distributions define the *dependence* between random variables, where dependence indicates that the distribution over X changes as Y changes. The opposite of dependence is *independence*. Independence is a property of two random variables, where knowing the outcome of one of the events does not alter the distribution over the random variable. The independence of two random variables, X and Y, is expressed mathematically

as $P(X|Y) = P(X)$, and is sometimes denoted $X \perp Y$. For instance, the probability of rolling a 1 on a fair die is unchanged by learning that a 6 was rolled previously. Independent variables cannot be mutually exclusive.

Independence is extended by the notion of *conditional independence*. Conditional independence states that given the outcome of an event Z, the distribution of X is independent of Y: $P(X|Z,Y) = P(X|Z)$. If this condition holds, it is said that X is conditionally independent of Y given Z. For instance, if a subject is a child ($Z = \text{True}$) they tend to be shorter ($X = \text{True}$) and have less developed vocabularies ($Y = \text{True}$) compared to adults. Given that someone is short, there is an increased probability their vocabulary is also poor, as there is an increased probability they are a child. However, given that the person is a child, knowing their height does not provide additional information as to whether their vocabulary is more developed or not. Therefore, height and vocabulary are conditionally independent given whether the subject is a child. Exploiting independences and conditional independences is critical to probabilistic analysis, as it allows certain sets of variables to be considered separately from other sets.

The distribution over multiple random variables is referred to as a *joint distribution*. For two outcomes, X and Y, the joint distribution is denoted $P(X = x, Y = y)$, and conveys the probability that both X and Y take outcomes x and y. The joint distribution is not ordered, i.e. $P(X,Y) = P(Y,X)$. The joint distribution is most often defined over *every* random variable being considered and hence is the most general definition of a model, leading the joint to be colloquially referred to as "the probability of everything." As we go on to see, all other distributions are simply manipulations of the joint, and hence sampling from and evaluating the joint distribution is of paramount importance in machine learning.

Joint distributions can be decomposed using the *product rule of probability*, written as $P(X = x, Y = y) = P(X = x|Y = y)P(Y = y)$. This is read as the "the probability of outcomes x and y is equal to the probability of outcome y multiplied by the probability of outcome x occurring given that outcome y occurred." These decompositions allow *causality* to be implied, i.e. the age of the subject logically precedes the probability of literacy given the age of the subject. The joint can, of course, be decomposed the other way, $P(X = x, Y = y) = P(Y = y|X = x)P(X = x)$, implying different causality. It is therefore a matter of choice or convenience how the joint is decomposed. Each decomposition

of a joint distribution is referred to as *factorisation*, and many different factorisations may exist, as is discussed further below.

A complementary rule is the *sum rule of probability*, written as $P(X = x \text{ or } Y = y) = P(X = x) + P(Y = y) - P(X = x, Y = y)$. The sum rule defines the probability that *at least* one of random variables X or Y takes the specified value. The form of the sum rule is more difficult to interpret than the product rule. Consider two of the possible eventualities that satisfy the condition: x occurs irrespective of Y, or, y occurs irrespective of X. These two terms are represented by $P(X = x)$ and $P(Y = y)$, and are additive as these events are independent. However, the possibility that *both* x and y occur is expressed in *both* of these terms. We must therefore subtract the probability of one of these subcases, represented by $-P(X = x, Y = y)$. Hence the form of the sum rule.

It is worth noting that we have thus far discussed probability in terms of discrete or categorical variables. The probability of a continuous variable taking a particular value is zero. For instance, no one is *exactly* six feet tall, whereas the value rolled by a dice can be exactly six. It is possible however to define the probability that a continuous variable falls within a set of bounds as a discrete event, i.e. probability you are less than x centimetres tall, $P(X \leq x)$. By the third probability axiom, we can see that the curve traced out by $P(X \leq x)$ rises monotonically as x increases. We can therefore appeal to basic calculus and write the derivative of this density as:

$$p(X = x) = \lim_{\delta x \to 0} \frac{P(X \leq x + \delta x) - P(X \leq x)}{\delta x} \tag{3.4}$$

where $p(X = x)$ is defined as the *probability density of* x. The density is therefore the amount of extra probability mass added per δx slice. However, the density is often referred to and practically used as the "probability" of a continuous variable without further consideration. Densities are often denoted with a lower-case p, although we do not rigorously differentiate density and distribution and their respective notations going forwards.

The *expectation* of a random variable, X, under the density $p(X)$ is defined as:

$$\mathbb{E}_{x \sim p(X)}[x] = \int_{x \in \mathcal{X}} x p(x) \mathrm{d}x. \tag{3.5}$$

and as a finite summation for discrete distributions, $\mathbb{E}_{x \sim P(X)}[x] = \Sigma_{x \in \mathcal{X}} x P(x)$. This integral or summation is defined over the whole domain of x and weights each value of x by the probability of x. This can also be viewed as the average value of x if an infinite number of

samples were drawn. This latter interpretation underpins *Monte Carlo methods*, discussed at length in Section 3.3. If the distribution from which x is drawn is obvious, often notation is simplified to just $\mathbb{E}[x]$. More generally, we can consider expectations of functions of random variables. The expectation of a function $f(x)$ under $p(X)$ is defined as:

$$\mathbb{E}_{x\sim p(X)}[f(x)] = \int_{x\in\mathcal{X}} f(x)p(x)\mathrm{d}x, \tag{3.6}$$

and similarly for discrete distributions. For instance, the points received by a player in a game may be equal to the square of the value on the dice. Here, $f(x) = x^2$, and the expected value is $15\frac{1}{6}$. An expectation is a linear operator, and hence obeys superposition, $\mathbb{E}[\alpha X + \beta Y] = \alpha\mathbb{E}[X] + \beta\mathbb{E}[Y]$, and is non-multiplicative, $\mathbb{E}[XY] \neq \mathbb{E}[X]\,\mathbb{E}[Y]$ (unless X and Y are independent). We can also construct conditional expectations, usually denoted $\mathbb{E}[f(X,y)|Y=y]$ or $\mathbb{E}_{x\sim p(X|Y=y)}[f(X,y)]$, to condition on certain outcomes.

Marginalisation is taking an expectation over one or more random variables in a joint distribution to yield a distribution over the remaining variables. For example, given a joint distribution $P(A,B,C,D)$, we marginalise over A and C to yield the *marginal distribution* $P(B,D)$:

$$P(B=b,D=d) = \mathbb{E}_{a',c'\sim P(A,C|B=b,D=d)}\left[P(A=a',B=b,C=c',D=d)\right], \tag{3.7}$$

noting that this result actually follows from the third probability axiom. Marginalisation allows us to average over *nuisance variables* in the joint distribution, and obtain the marginal distribution over only the variables of interest.

We now have everything required to derive the final fundamental rule in probabilistic analysis: Bayes rule. Bayes rule is easy to derive by writing both factorisations of the joint:

$$P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X), \tag{3.8}$$

and manipulating the second and third expressions to write Bayes rule as:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}. \tag{3.9}$$

Bayes rule therefore states that we can compute the conditional distribution over X given Y, by computing the probability of X irrespective of the value Y takes, multiplying this by the conditional probability of Y given X, and dividing by the probability of Y irrespective of X.

These distributions are discussed using specific terminology and convey particular ideas. The *prior* distribution, denoted $P(X)$, encodes a belief about the variable before data is

observed. The prior is often a design choice, relying on the prior conceptions of the modeller. The *marginal distribution*, denoted $P(Y)$, is the resulting distribution after marginalising all other random variables in a joint distribution. Referring to a distribution as a marginal implies the distribution is obtained by marginalising over a joint distribution, as opposed to being specified *a priori* as the prior distribution is, i.e. $P(Y)$ is subtly different to a prior. This marginal, also referred to as the *marginal likelihood* or *evidence*, is also the aforementioned normalising constant, ensuring that the posterior is correctly normalised. The *likelihood* of X given Y, denoted $P(Y|X)$, defines the conditional probability that the particular Y value was generated by a particular X value. However, it is important to note that the likelihood is actually used as a function of the variable *after* the conditioning bar, i.e. it is a *function* of the parameters X and not a distribution over the random variables X or Y. The likelihood, when viewed as a function of X, returns a real number indicating the relative probability that different X values generated Y, irrespective of how probable or likely that X is itself. As a result, the raw likelihood values over X are not guaranteed to yield a correctly normalised distribution. Finally, the *posterior distribution* over X given Y, denoted $p(X|Y)$, is the (normalised) conditional distribution over X given Y *and* the modellers prior held beliefs about the values X takes.

Quantifying the posterior is often the target of inference, and the Bayesian paradigm builds on this idea. We often denote variables that we cannot measure, also referred to as *unobserved* or *latent variables*, as X, and variables that we can measure, referred to as *observed variables*, as Y. Suppose as a modeller we have some prior belief over the values that X takes, expressed as the prior $P(X)$. Through the application of Bayes rule in (3.9), we can refine our prior belief about X in light of observed data Y (the posterior over X given Y) by simply multiplying the prior probability of X by the likelihood of X under the observed data and dividing by the probability of Y. We already eluded that the marginal is difficult to compute. However, the marginal, $P(Y)$, is just the normalising constant required to ensure the posterior is correctly normalised, and, critically, is independent of X. We can therefore express Bayes rule as a proportionality:

$$P(X|Y) \propto P(Y|X)P(X) = P(X, Y). \tag{3.10}$$

The probability of an unobserved latent variable, X, is proportional to the probability of the latent variable X *and* the observed data Y. This means that given data Y, and the ability

to evaluate the joint density, we can calculate *any* posterior distribution over subsets of unobserved variables up to a constant of proportionality. As a result, much of probabilistic and Bayesian machine learning is dedicated to devising more general and efficient ways to evaluate, estimate or generate samples from joint distributions. We discuss the wider implications of Bayesianism later.

A final relationship, applicable only to continuous densities, is a *change of variables*. The change of variables rule defines the distribution over a variable, y, that is a function of a second variable, x, as $y = f(x)$ where f is a one-to-one function, in terms of a distribution over x:

$$p(Y = y) = p(X = f^{-1}(y)) \left| \frac{\mathrm{d}}{\mathrm{d}y} f^{-1}(y) \right|_y , \tag{3.11}$$

where $f^{-1}(y)$ denotes the inverse of the function f. The first term is a distribution over x, and the second term is the absolute value of the derivative of the inverse function evaluated at y. This is a function of y. The first term translates the y value into the corresponding x value through the inverse function and evaluates the density. The second term then applies a *warp*, representing the observation that an infinitesimal δy slice in y-space is scaled by a factor of the derivative of the inverse of f when mapping into x-space. If x and y are vector valued, this term becomes the determinant of the Jacobian matrix, and hence the change of variables is only computable when a valid Jacobian exists, i.e. the function is bijective.

3.1.2 Specifying Models

We have talked at length about probability distributions and basic manipulations. We now flip to discussing specifying distributions. When defining a *probabilistic model* of a system we are often (explicitly or implicitly) defining the joint distribution over the random variables. This joint distribution defines your belief, as a modeller, of the functional relationship between all of the random variables in the system, both observed and latent. Crucially, a model is just a probability distribution. Any analysis of the model can then therefore be posed as manipulations of the distribution using the rules and axioms defined in the previous section.

In this thesis we most often discuss *generative models*. A generative model can be interpreted as an approximation of the sequence of relationships required to generate all random variables. As such, there is a strong sense of causality in generative models. For instance, whether or not it is raining and whether or not the sprinkler is on logically precedes whether or not the grass will be wet. Therefore, the generative model of the system would first

sample whether it is raining, then whether the sprinkler is on, and then whether the lawn is wet conditioned on whatever two values were sampled first. To define a generative model, one must define the existence and form of these relationships. Performing *inference* (particularly Bayesian inference) in a specified model is then the process of recovering the distribution over unobserved variables that led to the observed values, weighted by the probability that these latent outcomes co-occurred. To infer the probability that it is raining outside, given that the grass is wet, we must also consider the possibility that the sprinkler is on.

It is critical to understand here that there is not necessarily a *single* unique model for a given system. In fact, there are often many comparable models capable of explaining the same set of observed data. Different models may have different latent variables and entail different dependency structures between those random variables. For instance, one may model the dependency between the height and weight of a population as linear. The model has two parameters, or, if we are being fully Bayesian, random variables: the slope and offset of the linear fit. However, one may also consider a quadratic model defined by three parameters. Specifying the model structure relies on expert domain knowledge to define a set of meaningful latent variables and probabilistic relationships capable of describing the system. Deciding which of these models best represents the data is referred to as *model selection*, discussed later. However, designing a model must balance representing the system accurately, the complexity of the model, and what relationships are actually tractable to express and analyse. Designing and choosing the best model is therefore part science, part art.

In this thesis we consider exclusively *directed acyclic models*. Here, directed implies that there is a conveniently defined one-way dependency between two variables, i.e. the weighting of a coin determines the probability that a head is flipped. Acyclic then specifies that there are no closed loops in the model. A directed acyclic is a model that can be *topologically sorted*, where the variables are sorted such that the first variable is independent, the second variable is dependent only on the first, and so on for all variables. Directed acyclic models (later referred to as graphs, or DAGs) are by far the most common type of model studied, and hence "models" often refers exclusively to DAGs. There are also *undirected models*, also referred to as *fields*, which do not have directed probability relationships, and instead specify a *potential function* defining the probability of a particular setting of *both* variables, i.e. there is no inherent notion of causality and cannot be topologically sorted. We do not

use undirected models in this thesis, and so we refer the reader to Koller & Friedman [2009] and Murray & Ghahramani [2012] for more information.

3.1.2.1 Factorisations

The most straightforward way of specifying a joint distribution is through direct specification of the *factors* of the joint distribution. The factors of a joint distribution are individual probability distributions defining the dependencies between subsets of the variables being considered. The product of all the factors, as defined by the product rule, define the joint distribution. For instance, the lawn is more likely to be wet ($W = T$) when the timed sprinkler is on ($S = T$) or it is raining ($R = T$). The joint distribution can therefore be factorised as:

$$P(W, S, R) = P(W|S, R)P(S)P(R). \tag{3.12}$$

If, however, the sprinkler is fitted with a rain sensor that turns the sprinkler off, then the status of the sprinkler and rain are dependent and the joint distribution is instead represented as:

$$P(W, S, R) = P(W|S, R)P(S|R)P(R). \tag{3.13}$$

In going from (3.12) to (3.13) we used domain knowledge that the system was most accurately modelled by including the dependency between two of the variables, highlighting how different models with different dependency structures can be constructed to reflect different beliefs about the underlying system. We note that the dependency structure of two seemly identical joint distributions can be different, i.e. the distributions defined by (3.12) and (3.13) are different, despite the joint distribution being written as $P(W, S, R)$ for both. To deduce equality, the factors must be inspected. We also note there is not a unique factorisation of the joint distribution, as described above with the vocabulary and height example. Often however, some factorisations are more intuitive than others. This equivalence can also be derived as an implication of Cox's original postulates [Jaynes, 1995].

We have also chained together simple relationships to define the more complex joint distribution over all of the random variables. Each of these factors must be separately defined to define the whole system. To evaluate the probability of a particular configuration, say $p(S = T, W = T, R = T)$, we simply multiply the probability of each factor. When performing inference, we are often simply decomposing and manipulating a joint distribution into factors that we can evaluate or sample from. As a result, the dependencies in certain

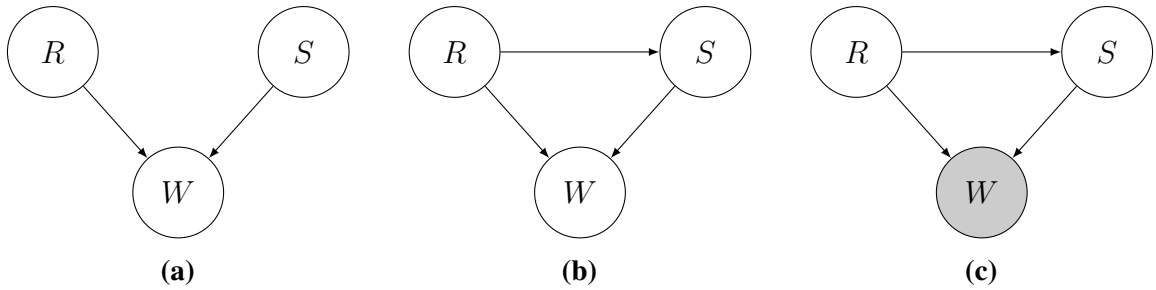


Figure 3.1: Graphical models for the models introduced in Section 3.1.2. Figures 3.1a and 3.1b show the graphical model associated with (3.12) and (3.13). Random variables are shown as circles and dependencies are shown as arrows. Figure 3.1c shows the graphical model where we have observed whether the grass is wet, indicated by a shaded node. Whether it was raining or whether the sprinkler is on, are latent variables, indicated by being unshaded.

factorisations may be more straightforwardly defined using domain knowledge, or, have more favourable properties for performing inference.

3.1.2.2 Graphical Models

Graphical models are a convenient method of representing the structure of a model in a more interpretable way than simply specifying the factors. Somewhat confusingly, graphical model refers to nothing more than an agreed convention for visualising a particular factorisation – it has nothing to do with the study of graphics. Examples of graphical models are shown for the sprinkler example in Figure 3.1.

There are two core components of graphical models: *nodes* and *edges*. Random variables are represented as nodes and are visualised as circles. Variables that are observed are indicated by shading the interior of the node, whereas latent variables are unfilled. Edges indicate a dependency between two random variables, and are visualised as an arrow joining the two nodes. The variables that a variable depends on is defined by the incoming arrows, i.e. an arrow from node S to node W indicates that W is conditioned on S in the chosen factorisation. Much like a factorisation, the graphical model only indicates the *structure* of the model. The type and dimension of each random variable and functional form of these relationships is not specified on the graphical model itself and must be stated elsewhere.

3.1.2.3 Programs

The direct specification of joint distributions through factorisations and graphical models is a convenient way of representing the dependency structure of variables. However, the functional relationship between two random variables must still be specified. For instance, let A denote the voxels of a volumetric MRI scan and let B denote the location of a cancerous

tumour in the scan. It is trivial to define that B depends on A. However, expressing the probability that a particular voxel is cancerous from raw voxel values using a closed-form probability distribution is likely to be impossible. Similarly, A may be the entire neural and body state of an organism at time t, and B is the entire neural and body state at time $t + 1$. Again, it is trivial to state that B is conditionally dependent on A. However, this relationship may require numerically solving a complex system of differential equations for which no convenient closed-form solution exists. Therefore, the dependency between A and B may be most easily expressed implicitly as a *program*. This program may be a segmentation or object detection algorithm that provides a probability distribution over the location of the tumour, or, through a multi-compartment neural simulator (such as NEURON [Hines & Carnevale, 2006]). The distribution $P(B|A)$ is therefore implicitly defined through the source code of the program. Specifying a program defines a joint distribution by implicitly specifying a particular factorisation and dependency structure between the internal variables of the program (random variables) *and* the stochastic and deterministic operations applied to these variables (dependencies).

Specifying distributions through programs allows a more diverse range of functional relationships to be compactly encoded. However, analysing and performing inference when specified as programs is notably more challenging. Evaluating the density of $P(B|A)$ when specified as a program may be intractable, and we may only be able to sample from the distribution by executing the program. Further, the range of models that can be specified by programs far outstrips those expressible through other methods, including, for instance, a different number of random variables per execution. Automating such inference in arbitrary programs is at the heart of *probabilistic programming* [Pfeffer, 2016; Rainforth, 2017; Tolpin et al., 2016; van de Meent et al., 2018]. Probabilistic programming aims to develop a range of language and inference tools to separate the process of defining models as programs from performing inference in the resulting model. Probabilistic programming has found success in domains as diverse as inverse reasoning in generative models of images [Ritchie et al., 2015; Le et al., 2017], planning in epidemiological models [Wood et al., 2020], and ecology [Dhir et al., 2016]. Much of this thesis will specify distributions as programs, and hence we must design inference methodologies that accommodate the limitations of this representation.

3.1.3 Commmon Models

Models may be specified directly by writing the factors of the joint, through a graphical model, or, through the specification of program code. However, there are a small set of common models that are used throughout machine learning, due to their generality in representing systems, favourable properties for performing inference, and ease of conveying the implications and assumptions of the model. Systems are therefore often approximated using one of these models, or, the representation used is chosen such that the system can be represented using one of these models. We discuss three common models: *Markov chains*, *hidden Markov models* (HMMs) and *parametric hidden Markov models*. These models are used as theoretical constructs for analysing systems, but also as convenient generative models for analysing systems. Much of the analysis we present later in this thesis relies on performing inference in one of these three models.

3.1.3.1 Markov Chain

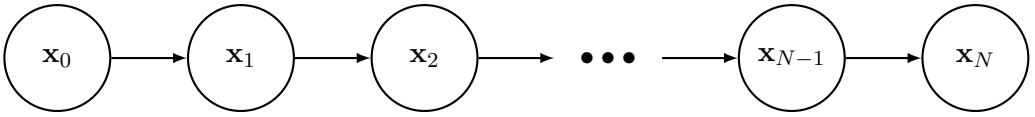
A Markov chain, shown graphically in Figure 3.2a, defines a sequence of random variables, indexed by n, individually denoted $\mathbf{x}_n \in \mathcal{X}$ and often referred to as the *state* of the chain. Each state is only dependent on the previous value in the sequence:

$$p_n(\mathbf{x}_n|\mathbf{x}_{0:n-1}) = p_n(\mathbf{x}_n|\mathbf{x}_{n-1}), \quad \forall n \in 1:N. \tag{3.14}$$

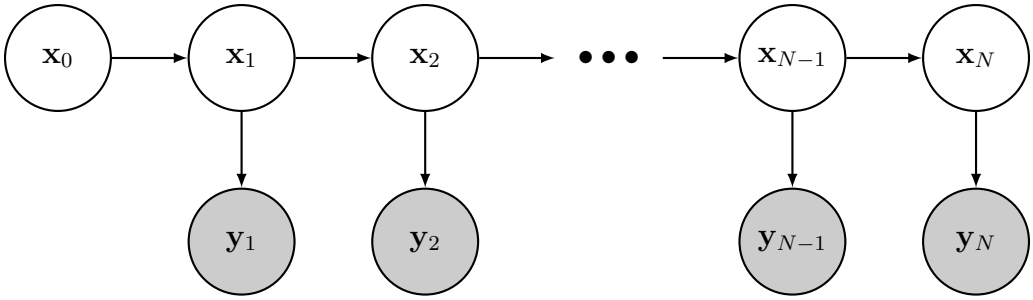
The distribution $p_n(\mathbf{x}_n|\mathbf{x}_{n-1})$ is referred to as the *transition probability*, *transition kernel*, the *transition density* in continuous state-spaces, or *plant model* in many engineering disciplines. The transition kernel describes the distribution over the next state conditioned on the current state. Therefore, Markov chains are often used to define the time-evolution of the state of a system, and are extremely convenient for analysis, as all future states are conditionally independent of all previous states given the current state. This means that storing and processing long histories of states is not required. The joint distribution factorises as:

$$p(\mathbf{x}_{0:N}) = p_0(\mathbf{x}_0) \prod_{n=1}^{N} p_n(\mathbf{x}_n|\mathbf{x}_{n-1}) \tag{3.15}$$

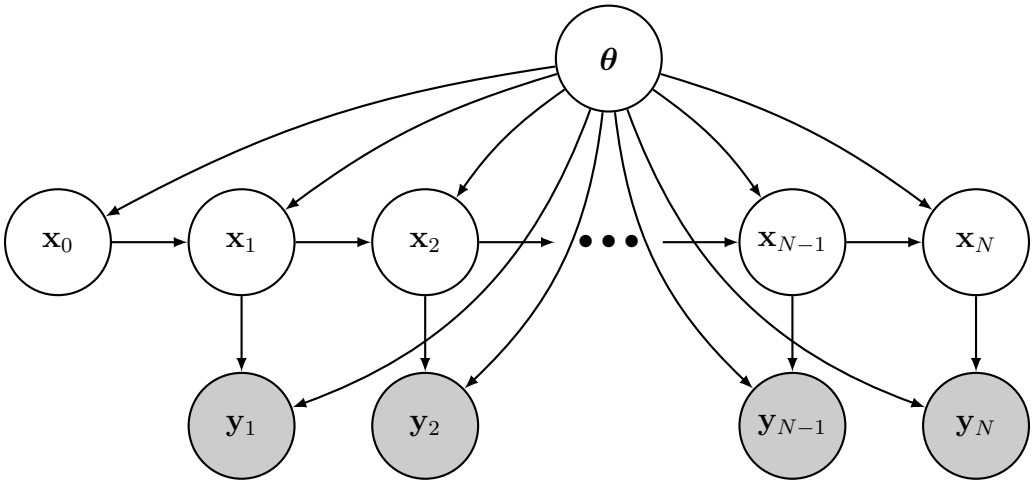
where the chain is of length $N+1$. The distribution $p(\mathbf{x}_0)$ is the distribution over state at time zero. Often this distribution will represent a prior over states. A Markov chain is entirely specified by the transition kernel and initial density. Later in this chapter we will denote to the transition kernel of a Markov chain as $T(\mathbf{x}_{n-1} \to \mathbf{x}_n)$, as is standard in certain areas of literature.



(a) Markov chain.



(b) Hidden Markov model (HMM).



(c) Parametric hidden Markov model.

Figure 3.2: Graphical models for a Markov chain, hidden Markov model (HMM), and parametric HMM introduced in Section 3.1.3, with joint distributions specified mathematically in (3.15), (3.17) and (3.20).

Other variants of Markov chains also exist. Throughout this thesis, we actually discuss discrete-time, finite-horizon, homogeneous Markov chains. Homogeneous Markov chains are where the transition distribution is not a function of time, i.e.

$$p_n(\mathbf{x}_n|\mathbf{x}_{n-1}) = p_n(\mathbf{x}_{n+1}|\mathbf{x}_n), \quad \forall n \in 1:N, \tag{3.16}$$

and hence the subscript in p_n is often dropped. Homogeneous chains are easier to analyse, are often a good approximation of the underlying system if there is no inherent time-dependency in the dynamics, and will allow us to make more powerful statements about the distribution over state implied by the chain later on. While inhomogeneous chains are allowed, we will not consider such scenarios, and assume homogeneity herein. *Infinite-horizon* chains are analysed to establish the convergence properties of general chains, and are often studied in the

context of Markov decision processes [Sutton & Barto, 2018]. In practice, however, all chains are *finite-time* chains, that is, the transition kernel will be applied N times, generating $N+1$ $\mathbf{x}_n$ values. Most chains are studied as *discrete time* chains, where variables are indexed by a discrete variable, $n \in \mathbb{Z}$, and are iterated by application of the transition kernel to go from one discrete time point to the next. In *continuous time* chains the state is indexable only by a continuous variable, $n \in \mathbb{R}$, and hence the chain must be numerically integrated. An example of a continuous time Markov chains is a stochastic ordinary differential equation. Often, continuous time chains are approximated as discrete time chains by using a small-but-finite integration timestep, δt. We do not study continuous time systems in detail in this thesis.

3.1.3.2 Hidden Markov Model

A hidden Markov model (HMM) is an extension of the Markov chain where the true state, $\mathbf{x}_n$, is not observable. Instead, only noisy or partial measurements, $\mathbf{y}_n$, dependent on $\mathbf{x}_n$ can be obtained at each time step. For instance, measuring the speed of a car using a speed gun returns a noisy estimate of the true speed of the car, whereas an image of a car reveals the position but not the speed of the car. In both cases, the *true* speed of the car is unknown, or is hidden. The graphical model of a HMM is shown in Figure 3.2b.

Much like a Markov chain, a HMM is defined using an initial state distribution, $p(\mathbf{x}_0)$, and a transition kernel, $p(\mathbf{x}_n|\mathbf{x}_{n-1}),\ n \in 1:N$. The HMM adds to this an *observation model*, sometimes referred to as the *emission density*, $p(\mathbf{y}_n|\mathbf{x}_n)$, describing the distribution over observed variables given the latent state. Throughout this document we adopt the convention that the latent state is indexed from $n=0$ to $n=N$, and that observations are available for $n=1$ to $n=N$. HMMs also exploit conditional independences for analysis. Observations are conditionally independent of all other random variables given the corresponding latent states:

$$p(\mathbf{y}_n|\mathbf{x}_n) = p(\mathbf{y}_n|\mathbf{x}_{0:N}, \mathbf{y}_{1:N\setminus n}), \tag{3.17}$$

and the distribution over the next state is conditionally independent of all previous latent and observed variables given the current state:

$$p(\mathbf{x}_n|\mathbf{x}_{n-1}) = p(\mathbf{x}_n|\mathbf{x}_{0:n-1}, \mathbf{y}_{1:n-1}). \tag{3.18}$$

The joint distribution of the HMM is therefore:

$$p(\mathbf{x}_{0:N}, \mathbf{y}_{1:N}) = p(\mathbf{x}_0) \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n)p(\mathbf{x}_n|\mathbf{x}_{n-1}). \tag{3.19}$$

As with Markov chains we again assume discrete time, finite-time horizon, homogeneous HMMs throughout this document. Observations may also not be available for every timestep, normally due to the interval between acquisitions being higher than the simulation timestep. This is often omitted to avoid notational clutter but is cleanly handled by most inference algorithms. *Selective perception* is a topic in robotics addressing selecting when to observe such that uncertainty over position or additional constraints are met [Warrington & Dhir, 2018; Ondrúśka et al., 2015]. We define two further distributions which are widely discussed in the context of HMMs: *filtering* and *smoothing* distributions. The filtering distribution is the distribution over the current latent variable, given all previous observations: $p(\mathbf{x}_n|\mathbf{y}_{1:n})$. In contrast, the smoothing distribution is the distribution over a previous latent state, given the future observations: $p(\mathbf{x}_{n'}|\mathbf{y}_{1:n})$, where $n' \in 0 : n-1$.

3.1.3.3 Parametric Markov Models and HMMs

Both Markov chains and HMMs can have unknown global parameters. A parametric HMM is shown in Figure 3.2c. In a parametric HMM, there is a set of fixed and latent global parameters upon which all transitions and emissions depend. An example of a global parameter may be the unknown mass of the vehicle being modelled as a HMM. The joint distribution is similar to the HMM, except for the addition of the prior probability and conditioning on parameters:

$$p(\mathbf{x}_{0:N}, \mathbf{y}_{1:N}, \boldsymbol{\theta}) = p(\boldsymbol{\theta})p(\mathbf{x}_0|\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})p(\mathbf{x}_n|\mathbf{x}_{n-1}, \boldsymbol{\theta}). \tag{3.20}$$

We can perform joint inference over the parameters of the model *and* the states of the model. Much of the core technical content we present in Chapters 4, 5 and 6 reduces to inference in parametric HMMs.

3.2 Estimation & Inference

In the previous section we discussed, at length, the specification and manipulation of complex probability distributions. We then introduced the idea that these distributions can be used as models of reality, or, more mathematically convenient approximations of a generative procedure. These models are ultimately deployed to make predictions about the world. Most often, we wish to estimate the distribution over a set of unobserved random variables

of interest, conditioned on a set of observed random variables and a particular model. For instance, we may wish to predict the location of a vehicle from an intermittent and noise corrupted series of latitude and longitude coordinates, or, we may wish to predict the distribution over pixel values for a region of an image that has been occluded (referred to as *image inpainting* [Xie et al., 2012]).

Such tasks are often referred to under the broad umbrella terms of *estimation* and *inference*. Inference, in statistics, refers to deducing a set of desired properties of a distribution from data. Estimation is then used generally to refer to where only a single value, or *point estimate*, of a variable is recovered. We will often refer to the distribution of interest as the *target distribution*, denoted $\pi(\mathbf{x})$, where $\mathbf{x}$ is the set of variables of interest, and the functional form of π is unavailable. Crucially, the target distribution may be either an objective statement about the system or data, or, a subjective statement of belief. The chosen property (or properties) targeted may be a single random value distributed according to the target distribution, the probability of a particular outcome, a closed-form expression defining the distribution, an approximate representation of the target distribution, or, exact recovery of the full target distribution.

While the scope of estimation and inference are very broad, our objective is nearly always inferring the posterior distribution over unobserved variables conditioned on observed variables. We may wish to condition on data that has been observed, such as inferring if a particular transaction is fraudulent given the customers transaction history. In this scenario, the transactional history of the customer, including the transaction in question, is observed and denoted $\mathbf{y}$. We wish to infer the probability that the transaction is fraudulent, denoted $\mathbf{x} = \text{True}$. We therefore wish to recover the posterior density $\pi(\mathbf{x}) = p(\mathbf{x}|\mathbf{y})$. However, we do not only need to condition on data that has actually been observed. We may wish to condition on particular future outcomes, such as quantifying the relative probability that an environmental policy decision, $\mathbf{x}$, will prevent a certain rise in global average temperatures, $\mathbf{y}$.

To foreshadow much of the material presented later in this thesis and to provide a concrete example, consider the scenario where we wish to infer the latent neural dynamics of a network of N neurons, given noisy recordings of potential. The potential of each neuron at each time is the latent state we wish to recover, $\mathbf{x}_t \in \mathbb{R}^N$. We leverage the Wicks neural model as our approximation of the true underlying dynamics, defining the

time-evolution of state, $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$, parameterised by static parameters $\boldsymbol{\theta}$, consisting of connection weights, electrophysiological parameters etc. We observe a noisy measurement of the potentials at each time, $\mathbf{y}_t$, using a microelectrode. The observed voltage is dependent on state by a known emission distribution, $p(\mathbf{y}_t|\mathbf{x}_t)$. Hence, the system can be modelled as a parametric hidden Markov model (Section 3.1.3.3). Suppose we know the model parameters, i.e. $\boldsymbol{\theta}$ is fixed and known, the inference task reduces to recovering the posterior distribution over the neural state at each time point, given the model and observed data, denoted $\pi(\mathbf{x}) = p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}, \boldsymbol{\theta})$. However, we may then instead assume that the parameters of the model are unknown and must also be inferred, and hence the target distribution becomes $\pi(\mathbf{x}, \boldsymbol{\theta}) = p(\mathbf{x}_{0:T}, \boldsymbol{\theta}|\mathbf{y}_{1:T})$. Inference is then the task of manipulating the probabilistic relationships such that this distribution can be quantified.

Actually performing this inference requires judicious application of the laws of probability introduced above. We introduced these laws as a series of mechanistic or axiomatic statements. However, what we did not discuss was how these manipulations and models are combined and applied to achieve practical objectives, such as estimating vehicle positions or identifying fraudulent transactions. We also only briefly discussed the conceptual differences underpinning the various applications of these methods. Different methods for solving a problem implicitly make different assumptions about the underlying system and solution space. In this section we aim to give a high-level overview of the terms and concepts used throughout machine learning and statistics. A working knowledge of these terms and approaches, and the implications and differences between these approaches, is critical to understanding, comparing and contrasting different methods for solving a given problem. In subsequent sections we then introduce standard methods, that build directly on these probability laws and principals, that are used throughout machine learning to perform inference.

It is worth noting here however that the choice of method is more often driven by the practical constraints of the problem, the type of solution required and the prior expertise of the modeller, as opposed to a deep-seated conceptual or philosophical preference.

3.2.1 Bayesian & Frequentist Methods

The first, and most notorious, distinction is between *Bayesian* and *frequentist* methods [Barber, 2012; Bishop, 2006; Jordan, 2009]. This distinction is closely related to the different

interpretations of probability offered by Cox and Kolmogorov [Cox, 1946; Kolmogorov & Bharucha-Reid, 2018]. The debate about the merits of these philosophies has raged for decades [Bayarri & Berger, 2004; Gelman et al., 2008; Orloff & Bloom, 2014; Popper, 1959; Samaniego, 2010; Wagenmakers et al., 2008], and discusses topics from the highly practical, such as predicting class labels, to the abstract, tackling epistemology and what it means to "know" [Talbott, 2001; Tarantola, 2006]. As alluded to in the previous section, the principal distinction between Bayesian and frequentist philosophies is a difference in what is implied by a probability. In frequentist statistics, a probability is viewed as the *frequency* of an event. For instance, if one flips a coin a number of times, the probability is interpreted as the relative frequency of each side being face up. In contrast, the Bayesian philosophy views probability as a measure of *uncertainty*, i.e. a Bayesian can only predict the outcome of a coin flip with a confidence of one-half.

This distinction is rooted in what is held constant by each paradigm. A Bayesian considers the data to be fixed, and considers a range of hypotheses, models or parameters. The probability then expresses the relative confidence that the data were generated by a particular model. Observing more data concentrates our belief as to which is the most probable model. This is viewed by proponents of the Bayesian method as the most natural framing of the problem. The same data can be generated by a range of models and hence we should have some inbuilt notion of representing, and then aggregating over, the possible models. As we see more data, we should become more confident in which of the candidate models best explains the underlying generative process.

In contrast, the frequentist approach considers the underlying generative process as fixed, and that it is instead the data that are variable. The frequentist therefore searches for the single model that minimises the error across all possible datasets that could have been generated. Increasing the amount of data does not necessarily change the inference result, but may make the result more robust to noise in the data. As a result, frequentist methods are more closely related to the repeatability of the experiment, leading to concepts such as binary acceptance or rejection of hypotheses, p-values and cross-validation.

The Bayesian interpretation has several benefits. Most practically, considering probabilities as statements about belief or uncertainty allow prior beliefs to be neatly incorporated. Data are then used to *update* or *refine* these beliefs through Bayes rule. This allows us,

as scientists, to impart additional information beyond what is simply observed. This is particularly important in the low-data regime. For instance, suppose we have flipped a coin exactly once, returning heads. A pure frequentist can only interpret this data as the coin only ever landing heads, $p(H) = 1$. In contrast, a Bayesian would consider a range of underlying models, such as the coin is not weighted, slightly waited heads, or slightly weighted tails, and use the data to update the belief ascribed to each model. In this case, the Bayesian would increase the probability that the coin is slightly weighted to heads, but not to the point of rejecting the possibility that the coin is fair or weighted slightly towards tails. Maintaining a distribution over models retains more information than simply accepting a single model, and allows more powerful inferences and predictions to be made. As a result, Bayesian methods often allow multiple weighted explanations to be generated, e.g. there is a distribution over the location of the vehicle, or, multiple different inpaintings can be generated (although this is also dictated by methodological choices, discussed below). In contrast, frequentist methods typically produce a single point estimate.

Bayesianism also encapsulates more neatly events that cannot be repeated multiple times to establish a frequency. For instance, we may wish to predict the probability that the polar ice caps melt within fifty years [Bishop, 2006]. We have no data, and hence the purely frequentist approach is limited. However, we can define our beliefs about the meteorological, geological and societal time-evolution of the world to predict the probability that the ice caps melt. Our predictions can then be refined under whatever data becomes available. Crucially, Bayesian approaches inherently handle uncertainty over parameters and models. Frequentist methods have no such in-built notion of uncertainty over parameters, and must be manipulated to generate ad-hoc estimates of such parameter values.

Critics of the Bayesian method cite that the model, and particularly the prior, is subjective. Two people may have different prior beliefs, and hence receive different results from interpreting the same data. If the model specified is *mismatched* to the underlying system, the results of Bayesian inference may be spurious. Beyond the subjectivity of the prior, specification of a full generative prior is non-trivial for many systems, and actually performing Bayesian inference is often far more challenging than frequentist methods, to the point where solving a problem in a fully Bayesian manner is intractable. Often, the simpler frequentist solution is sufficient, and hence solving the more complex Bayesian task is unnecessary.

Finally, there is a slight inconsistency in the Bayesian method, that the model itself is defined using information acquired by the modeller that is not internalised or represented in the inference process, and hence has access to "extra" information.

3.2.2 Discriminative & Generative Methods

Inference methods are further separated as *discriminative* and *generative* [Jebara, 2012; Ng & Jordan, 2002]. The principal difference between the two methods is the level of consideration of how the data were created. Discriminative methods do not attempt to model this process, simply learning a mapping from the observed data to the unobserved quantity of interest. Examples of discriminative methods are decision tree classifiers [Ho, 1995; Kulkarni & Sinha, 2012; Rainforth & Wood, 2015] and logistic regression [Bishop, 2006]. In contrast, generative methods attempt to model the underlying process that generated the data. Examples of generative methods include Gaussian mixture models [Reynolds, 2009; Bishop, 2006], naive Bayes classifiers [Rish et al., 2001] and variational autoencoders [Kingma & Welling, 2013]. Discriminative methods often have parallels with more frequentist philosophies, whereas generative methods are almost always more strongly linked with Bayesian philosophies.

Generative methods begin by quantifying the joint distribution over observed data, $\mathbf{y}$, and the latent variables, $\mathbf{x}$, denoted $p(\mathbf{y}, \mathbf{x})$. Generative methods are nearly always *unsupervised*, where we only have access to data, $\mathbf{y}$, and we rely on our prior beliefs and specified models to draw conclusions about the target latent variables, $\mathbf{x}$. The joint distribution decomposes as $p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$. Generative methods require that the modeller specify a generative process or distribution over the unobserved variables, $p(\mathbf{x})$, and the procedure for how observations are generated from these unobserved variables, $p(\mathbf{y}|\mathbf{x})$. Once the joint has been quantified, application of Bayes rule yields the posterior distribution, $p(\mathbf{x}|\mathbf{y})$, which can be generated for any $\mathbf{x}$ or $\mathbf{y}$ value.

In contrast, discriminative methods do not attempt to model the underlying generative process, instead directly targeting (usually) the posterior distribution, $p(\mathbf{x}|\mathbf{y})$, or simply providing an estimate of the latent random variable, $\mathbf{x} \leftarrow f(\mathbf{y})$. This is generally done by using a large corpus of $(\mathbf{x}, \mathbf{y})$ pairs, and learning a function that directly maps from $\mathbf{y}$ to $\mathbf{x}$, described as *supervised* learning. When unseen $\mathbf{y}$ values are provided at deployment time, denoted $\mathbf{y}'$, these values are simply input into this function to produce an estimate of the unobserved latent variables $\mathbf{x}'$. Discriminative methods are generally easier to implement, as

they do not require a generative model to be specified, and can make easy use of black-box function approximators such as neural networks.

Suppose in an image inpainting task, $\mathbf{y}$ is an image with an occluded region, and $\mathbf{x}$ are the occluded pixels we wish to predict. Using a generative method here would be difficult (although not impossible) as we would need to be able to specify the generative process for images, $p(\mathbf{x})$. In contrast, a discriminative method simply learns a mapping from the image to the missing pixels from training examples. Hence, in domains where the latent space is highly structured, but the structure is difficult to specify and there is an abundance of data, discriminative methods are often more readily applicable.

However generative methods allow us to impart domain knowledge through the specification of the model. This reduces the reliance on data compared to discriminative methods by instead using our prior beliefs as to how the random variables are interdependent, and critically, allowing us to model systems for which supervised data cannot be gathered. Generative methods tend to allow more effective uncertainty estimates and tend to be more interpretable. Hence, generative methods are typically used in safety-critical applications, or, to study systems where we cannot simply measure the target. Generative methods are, however, solving a more complex problem, requiring the definition of, and performing inference in, the entire latent structure for what might only be a simple regression task. Therefore, generative models can be less flexible and more computationally demanding.

Models can further be broken down as *parametric* and *non-parametric* models. The difference between these two is how the complexity of the inference result changes with the amount of data. Parametric models have fixed complexity. An example of a parametric model is linear regression. While adding more training data may increase the accuracy and robustness of the regressor to noise at training and test time, the regressor is still a linear function and cannot represent more complex relationships regardless of how much data are provided. Parametric models normally have a fixed and finite number of parameters, regardless of the size of the dataset being analysed. Non-parametric models on the other hand can increase their complexity as the dataset grows larger. This means that as more data are acquired, the solution space grows. As a result, non-parametric models can represent more complex systems, often requiring a hypothetically infinite number of parameters if one were

to represent the model using a parametric model. Examples of non-parametric models include Gaussian processes [Rasmussen, 2003] and nearest neighbours classifiers [Altman, 1992].

3.2.3 Point Estimators

We are generally concerned with answering a question given some data: Is the pedestrian going to step into the road? Which products should we recommend to the customer next? What is the best model of observed data? Will the ice caps melt in fifty years? Therefore, often, the dominant factor in deciding which approach to use is whether a full distribution is required, whether a point estimate of the value is sufficient, what can be controlled and should be averaged over, and if it is possible to specify and perform the required inference in the model. Often, a decision must be taken, and hence the distribution must be reduced to a single, actionable value. Alternative formulations are therefore often used over full Bayesian inference.

Maximum likelihood estimation (MLE) is arguably the most widespread estimation tool. Given data $\mathbf{y}$, the MLE estimator searches for the latent values, $\boldsymbol{\theta}$, that maximise the likelihood of the observed data under a given model: $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\Theta} p(\mathbf{y}|\boldsymbol{\theta})$. The probability of the latent variables themselves are not considered. This is in contrast to *maximum a posteriori* (MAP) estimation, which instead maximises the posterior probability: $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\Theta} p(\boldsymbol{\theta}|\mathbf{y}) = \arg\max_{\boldsymbol{\theta}\in\Theta} p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. Although MAP estimators return a single value, they are inherently Bayesian as a prior distribution is considered. MLE is invariably the easiest operation, but can be prone to overfitting, especially when using small datasets. Accordingly, Bayesians may assert that MLE is a Bayesian method assuming a uniform prior, whereas frequentists may assert that MAP is simply a regularised MLE estimator. Therefore, MLE sits between Bayesian and frequentist inference, whereas MAP is inherently Bayesian. Both methods are only applicable when a point estimate, as opposed to a full distribution, is sufficient.

Marginal maximum a posteriori (MMAP) estimation [Doucet et al., 2002], similar to *type two maximum likelihood* [Bishop, 2006], is an extension of MAP estimation to when nuisance latent states are marginalised over: $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\Theta} \int_{\mathbf{x}\in\mathcal{X}} p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta})\mathrm{d}\mathbf{x}p(\boldsymbol{\theta})$. The integral term is often referred to as the *model evidence*. This estimation problem is common when $\boldsymbol{\theta}$ refers to the static parameters of a model, and $\mathbf{x}$ refers to individual events that contribute to $\mathbf{y}$. Here, the latent variables $\mathbf{x}$ are dependent on $\boldsymbol{\theta}$, but we do

not have direct control over them. For instance, we may wish to maximise the probability that we win a hand of poker, $\mathbf{y}$, over the available strategies, $\boldsymbol{\theta}$, while marginalising over the unknown strategies of our opponents, $\mathbf{x}$.

MMAP is frequently used in Bayesian model selection [Linhart & Zucchini, 1986; Wasserman et al., 2000]. Given a set of candidate models, $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_N$, or a continuum of models defined by continuously valued parameters, $\boldsymbol{\theta} \in \mathbb{R}$, we wish to select the model that explains the data the best, while marginalising over all the random variables that we could not observe. Therefore, maximising the model evidence is a common criterion for choosing the best performing model. We will use MMAP in Chapter 4 to select the model of neural dynamics by estimating the parameters from observed data, marginalising over the unknown stochastic latent neural dynamics.

Finally, *posterior predictive inference* is then a specific computation that uses the *current* posterior distribution to estimate the distribution over *future* random variables [Gelman et al., 2013]. The posterior predictive inference task is compactly defined as:

$$p(\mathbf{x}|\mathbf{y}) = \int_{\boldsymbol{\theta}\in\Theta} p(\mathbf{x}|\boldsymbol{\theta}, \mathbf{y}) p(\boldsymbol{\theta}|\mathbf{y}) \mathrm{d}\boldsymbol{\theta}, \tag{3.21}$$

where $\mathbf{y}$ is a corpus of data from which the posterior over model parameters or states, $p(\boldsymbol{\theta}|\mathbf{y})$, has been solved for. PPI then integrates over the uncertainty in the estimate of $\boldsymbol{\theta}$ to predict the new random variable, $\mathbf{x}$. If a point estimate of $\boldsymbol{\theta}$ has been obtained, either by frequentist, MLE or MAP methods, this integral reduces to the density under the single parametrised model and no integration is required. In contrast, if a fully Bayesian treatment is used, uncertainty over $\boldsymbol{\theta}$ is propagated through the system to provide more expressive predictions about $\mathbf{x}$. PPI is particularly common in state-space models, where we wish to predict the evolution of state conditioned on our current estimate of the state. Suppose the data are all observations to date, $\mathbf{y} = \mathbf{y}_{1:t}$, and the latent state of the state-space model is denoted $\boldsymbol{\theta}$, then the posterior predictive inference task is predicting the *future* states, denoted $\mathbf{x}$.

3.2.4 Integration and the Normalising Constant

We introduced Bayesian inference above. At the heart of Bayesian inference is Bayes' rule:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})} = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{\int_{\mathbf{x}\in\mathcal{X}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})\mathrm{d}\mathbf{x}}, \tag{3.22}$$

where we use observed data, $\mathbf{y}$, to refine our prior beliefs about an unobserved variable, $\mathbf{x}$. The denominator, $p(\mathbf{y})$, referred to as the marginal likelihood, is the normalising constant

that ensures that $p(\mathbf{x}|\mathbf{y})$ integrates to one as is required by the probability axioms. Inspecting the form of the marginal likelihood, we see that it is the probability of the data $\mathbf{y}$ weighted by our belief of the values that $\mathbf{x}$ can take. Suppose $\mathbf{y} = \{H, H, T\}$ is the result of flipping a coin several times, and $\mathbf{x} \in [0, 1]$ is the weight of the coin. To compute the normalised posterior probability that a particular coin is weighted, we must compute $p(\mathbf{y})$, and hence we must integrate over the probability of this outcome over *all possible* weighted coins. Analytically performing this integration, even for this simple example, is intractable for all but carefully designed models. Monte Carlo integration or quadrature may be possible as the problem is low-dimensional. Consider instead that $\mathbf{y}$ is the hand you have been dealt in poker, and that $\mathbf{x}$ is the ordered set of cards left in the pack. There are more possible orderings than there are atoms on Earth, and hence even estimating this summation is likely impossible. This is why computing the normalisation constant is, often reflexively, referred to as "difficult" or "intractable" to compute. As a result, exactly computing the posterior density in Bayesian inference directly is often challenging.

More generally, performing *any* integration often analytically is intractable, and is practically challenging to even approximate. This is, in part, the reason why MLE and MAP are so widespread, since these methods do not need to compute integrals. Much of Bayesian inference research is dedicated to developing more efficient ways to estimate integrals, and hence enable the computation of the normalising constant. However, very broadly speaking, most of Bayesian inference and probabilistic machine learning research relies on methods that simply avoid performing integrals wherever possible, generally through a combination of clever design and specific assumptions. As we will see shortly, we can often exactly evaluate the joint density, $p(\mathbf{x}, \mathbf{y})$, as this does not require computation of the normalising constant. We can then operate on this joint to recover distributions or values of interest without explicitly computing the normalising constant.

3.2.5 Summary

In this short section we aimed to give a high-level overview of inference. Fully exploring all of the terms introduced here is well beyond the scope of this thesis, and, to fully understand these principals and their most deep-rooted and wide-reaching implications often takes an entire career working with these ideas. These ideas are also deeply intertwined with each other, and are not always clearly delineated in literature or conversation. In this thesis we take

a predominantly generative and Bayesian approach to analysing *C. elegans*. This allows us to succinctly incorporate the decades of neuroscience research and accumulated knowledge into our models and inferences, and ameliorate the inherent unobservability of the system. This approach also fundamentally places the onus onto us as the designer to define meaningful models, and abstracts the inference and estimation to theoretically grounded procedures operating on this model. While there is no guarantee that any numerical approximation of such an algorithm actually recovers the correct solution, it strongly divorces the specification of the model from inference. Much of this thesis reduces to recovering the joint density of the data, time-varying latent variables and static global parameters in a parametric HMM. From this joint we can estimate the posterior density over the latent variables, or, maximise the joint density over latent variables to recover the single set of values that are most likely given the data and our prior beliefs. We concluded by suggesting that manipulating and performing inference on closed-form definitions of probability distributions, although mathematically well-posed, is often practically or computationally intractable. We therefore introduce the most powerful and flexible workhorse of statistical inference: *Monte Carlo* methods.

3.3 Monte Carlo Methods

We suggested in the previous section that many mathematically simple inference tasks are practically or computationally intractable, either due to the inability to combine probability distributions, or, the difficulty of performing the required integrals. We now introduce what is arguably the workhorse of inference: *Monte Carlo* methods [Kroese et al., 2014]. The etymological origin of the term Monte Carlo refers to the Monaco gambling area, and the observation that the distribution over outcomes of inherently chance-based gambling can be approximated by averaging over many individual trials.[3]

Monte Carlo methods use large sets of discrete points, referred to as *samples* or *particles*, to approximate distributions. As the number of particles used becomes large, the approximation becomes exact. The discrete representation used by Monte Carlo is often more flexible and easier to manipulate than analytic or programmatic representations of distributions. This flexibility and asymptotic correctness means that a vast range of Monte Carlo inference algorithms have been developed, and are used in practice to solve a vast array of problems. Each of these algorithms place different assumptions on the family of models

[3] In the long-run, the house *always* wins.

that can be analysed, excel at particular types of inference, and have different operational requirements. Although not without limitations and idiosyncrasies, Monte Carlo methods provide us with the probabilistic and algorithmic tools required to analyse complex systems.

We begin by giving an overview of the Monte Carlo approach, before examining the specific algorithms we will use repeatedly throughout this thesis. Much of the material in this section is explored in more detail in Owen [2013].

3.3.1 Foundations

Monte Carlo methods represent distributions using discrete points:

$$\pi(\mathbf{x}) = \lim_{K\to\infty} \frac{1}{K} \sum_{k=1}^{K} \delta\left(\mathbf{x} - \mathbf{x}^{(k)}\right), \quad \mathbf{x}^{(k)} \sim \pi(\mathbf{x}), \tag{3.23}$$

where individual particles, denoted $\mathbf{x}^{(k)}$, are *independent and identically distributed* (i.i.d.) samples from the distribution being approximated, $\pi(\mathbf{x})$. This representation uses K particles to approximate the distribution. As K tends to infinity, the approximation becomes exact. For discrete distributions, the origin of this form is easy to see. For continuous probability densities, this representation can be derived by considering the cumulative distribution, similar to how we derived the continuous density in Section 3.1. Importantly, it is the *density* of samples in Monte Carlo that equates to the distribution. Hence, histograms of samples are frequently plotted, as these visualise the density of samples, or, kernel density estimation is used to aggregate the discrete samples into a smooth approximation of the target density [Sköld & Roberts, 2003].

Often, we can only evaluate an unnormalised distribution, $\gamma(\mathbf{x}) = Z\pi(\mathbf{x})$. However, it is critical to note that samples distributed according to an unnormalised density are also distributed according to the corresponding normalised distribution. As a result sampling will often allow us to avoid explicitly computing normalising constants. Much of the core of Monte Carlo methods are then dedicated to actually performing the sampling, $\mathbf{x}^{(k)} \sim \gamma(\mathbf{x})$, more efficiently and under fewer restrictions.

A further benefit of particle-based representations is highlighted by considering compositions of probability distributions. Suppose we have specified a joint distribution over two variables using the product rule, $\pi(\mathbf{x}) = p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1)$. We have a closed-form representation of $p(\mathbf{x}_1)$ that we can sample from and evaluate easily, such as a Gaussian

distribution. The variable $\mathbf{x}_2$ is then dependent on $\mathbf{x}_1$. However we do not have a closed-form mathematical representation of the density and we are not able to evaluate the density. However, we can draw samples of $\mathbf{x}_2$ for a particular $\mathbf{x}_1$ value (i.e. $p(\mathbf{x}_2|\mathbf{x}_1)$ is a *generative procedure* for *forward-only model*). Multiplying the distributions is intractable and hence computing or sampling from the joint density directly is not tractable. However, using Monte Carlo principals we can sample from the joint by first sampling $\mathbf{x}_1^{(k)} \sim p(\mathbf{x}_1)$ and then sampling $\mathbf{x}_2^{(k)} \sim p(\mathbf{x}_2|\mathbf{x}_1^{(k)})$, to yield a single $(\mathbf{x}_1, \mathbf{x}_2)^{(k)}$ pair. This is a single i.i.d. sample from the joint distribution, and hence this process can be repeated many times to obtain a discrete representation of the joint distribution as per (3.23). This can be performed for any joint distribution by sampling from each conditional distribution in a topological order. Since models are typically specified through the factors, this is often immediately tractable.

Discrete representations also make evaluating integrals more tractable. Computing the expectation of a function, $f(\mathbf{x})$, under the distribution, $\pi(\mathbf{x})$, can be performed easily using *Monte Carlo integration* by taking the empirical mean of the evaluations of the function at the discrete points:

$$A = \mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f(\mathbf{x})\right] = \int_{\mathbf{x}\in\mathcal{X}} \pi(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x}, \tag{3.24}$$

$$= \lim_{K\to\infty} \frac{1}{K} \sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right), \quad \mathbf{x}^{(k)} \sim \pi(\mathbf{x}). \tag{3.25}$$

This allows $f(\mathbf{x})$ and $\pi(\mathbf{x})$ to take any form so long as we can evaluate the function and generate samples from $\pi(\mathbf{x})$. This is unlike if we attempted to directly integrate, as this would impose practical constraints of the compatibility of $f(\mathbf{x})$ and $\pi(\mathbf{x})$. Most significantly for our purposes, this allows $f(\mathbf{x})$ and $\pi(\mathbf{x})$ to be specified as a program which we can only sample from or run forward.

A crucial property is that the summation in (3.25) is also *unbiased* for finite K. Unbiased means that the expected error between the true value and the value computed using a finite Monte Carlo summation is zero. This is critical as we must ensure that the tractable estimator does not introduce any systematic *bias*, or error, into our inference result. For notational convenience, we will define the result of a single finite Monte Carlo integration as $\bar{A} = \frac{1}{K}\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right)$, where all K particles are independently distributed according to $\pi(\mathbf{x})$, denoted $\mathbf{x}^{(1:K)} \sim \pi(\mathbf{x})$, and hence $\bar{A}$ is itself a random variable. For an estimator

to be unbiased we require the following equality to hold:

$$A = \mathbb{E}\left[f(\mathbf{x})\right] = \mathbb{E}\left[\bar{A}\right] = \mathbb{E}_{\mathbf{x}^{(1:K)}\sim\pi(\mathbf{x})}\left[\frac{1}{K}\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right)\right]. \tag{3.26}$$

As summations and expectations are linear operators, we can rearrange:

$$\mathbb{E}\left[\bar{A}\right] = \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}_{\mathbf{x}^{(k)}\sim\pi(\mathbf{x})}\left[f\left(\mathbf{x}^{(k)}\right)\right]. \tag{3.27}$$

However, the term inside the summation is now an infinite-sample expectation, and hence has a fixed value equal to A, therefore:

$$\mathbb{E}\left[\bar{A}\right] = \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}_{\mathbf{x}^{(k)}\sim\pi(\mathbf{x})}\left[f\left(\mathbf{x}^{(k)}\right)\right] = \frac{1}{K}\sum_{k=1}^{K} A = A. \quad \square \tag{3.28}$$

Therefore the expected value of the finite-sample estimator is equal to the expected value of the infinite-sample estimator, and hence the expected error in the estimate is zero.

However, there is no guarantee that the error for any single finite-sample estimate is zero. The expected squared error in any single estimate of A is referred to as the *variance* of the estimator. A critical identity for deriving the variance of the estimator is that the variance of a sum of independent samples is the sum of the variances: $\mathrm{Var}\left[\sum_{k=1}^{K}\mathbf{x}^{(k)}\right] = \sum_{k=1}^{K}\mathrm{Var}\left[\mathbf{x}^{(k)}\right]$. Defining σ^2 as the variance of a single function evaluation:

$$\mathrm{Var}\left[f\left(\mathbf{x}^{(k)}\right)\right] = \sigma^2, \quad \mathbf{x}^{(k)} \sim \pi(\mathbf{x}), \tag{3.29}$$

then the variances of a summation is expressible as:

$$\sum_{k=1}^{K}\mathrm{Var}\left[f\left(\mathbf{x}^{(k)}\right)\right] = \mathrm{Var}\left[\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right)\right] = K\sigma^2. \tag{3.30}$$

Expanding the second variance term using the standard expression for variance:

$$\mathbb{E}\left[\left(\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right)\right)^2\right] - \mathbb{E}\left[\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right)\right]^2 = K\sigma^2, \tag{3.31}$$

and dividing by K^2 (noting that both terms on the left hand side are square terms):

$$\mathbb{E}\left[\left(\frac{1}{K}\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right)\right)^2\right] - \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right)\right]^2 = \frac{\sigma^2}{K}. \tag{3.32}$$

Then, recognising that $\frac{1}{K}\sum_{k=1}^{K} f\left(\mathbf{x}^{(k)}\right) = \bar{A}$, we see that the left hand side is simply the variance of the finite-sample Monte Carlo integration, $\bar{A}$:

$$\mathrm{Var}\left[\bar{A}\right] = \frac{\sigma^2}{K}. \quad \square \tag{3.33}$$

This shows that the variance of the Monte Carlo integrator reduces reciprocally with the number of samples taken. Critically, when coupled with the fact that the estimator is unbiased,

this verifies that as $K \to \infty$ the error in the Monte Carlo integrator also tends to zero. This property is referred to as the estimator being *consistent*. Furthermore, the convergence rate is not a function of the dimensionality of $\mathbf{x}$, i.e. the convergence rate is the same regardless of the size of the latent variable. While it may take longer in absolute terms to converge for higher dimensional latent variables, doubling K will always half the variance of the error. This is in contrast to the scaling performance of other evaluation-based methods, such as quadrature, that can scale extremely poorly with dimension [Dick et al., 2013; Owen, 2013]. We note, however, this behaviour is not necessarily preserved when considering nested expectations [Rainforth et al., 2018].

The final benefit of discrete representations is the ease with which marginalisation and conditioning can be performed. Suppose we wish to marginalise $\mathbf{x}_2$ in the joint distribution $p(\mathbf{x}_1, \mathbf{x}_2)$ to yield the marginal distribution over $\mathbf{x}_1$. Marginalisation can be expressed as an integration, and hence, for the reasons outlined above, performing this marginalisation analytically may be intractable. However, given Monte Carlo samples from the joint, $(\mathbf{x}_1, \mathbf{x}_2)^{(k)} \sim p(\mathbf{x}_1, \mathbf{x}_2)$, marginalising a random variable from the joint is trivial, simply dropping the variable that is being marginalised over, i.e. if $(\mathbf{x}_1, \mathbf{x}_2)^{(k)} \sim p(\mathbf{x}_1, \mathbf{x}_2)$ then $(\mathbf{x}_1)^{(k)}$ is distributed according to $p(\mathbf{x}_1)$. This means that we can trivially obtain samples from any marginal once we have samples from the joint. To condition, we simply take those samples that meet the criteria. Suppose we wish to recover $p(\mathbf{x}_1|\mathbf{x}_2 = a)$. We can obtain $(\mathbf{x}_1)^{(k)} \sim p(\mathbf{x}_1|\mathbf{x}_2 = a)$ by discarding those $(\mathbf{x}_1, \mathbf{x}_2)^{(k)}$ pairs for which $\mathbf{x}_2^{(k)} \neq a$. For the remaining pairs, $(\mathbf{x}_1, a)^{(k)}$, the $\mathbf{x}_1$ values are distributed according to $p(\mathbf{x}_1|\mathbf{x}_2 = a)$. While this approach is sometimes used for discrete variables, where equality conditions can hold, it is not immediately useful for continuous variables, as the equality condition will never hold. Many methods therefore exploit notions of smoothness to leverage nearby samples.

3.3.1.1 Why Expectations?

An important point to reflect on is why we discuss "performing integrals" or "evaluating expectations" on such a regular basis. Since we can draw samples from a joint distribution, marginalise by dropping dimensions, and condition by discarding samples; we can therefore obtain samples from any marginal, conditional or posterior from these samples. These operations do not require integration, and hence the reverence with which expectations are discussed can seem unfounded. The explanation of simply "they just are important"

can be deeply unsatisfactory. While there are no cut-and-dry reasons why integrals are important, we offer some simple intuitions as to why performing integrals and computing expectations is unavoidable.

The simplest, although the most unsatisfactory, reason is that it is the form of the sum rule. Probabilities and probability densities sum to one. Therefore summation or integration, both of which are expressible as expectations, are fundamentally an operation of paramount importance. Any time a posterior is being computed, the normalisation constant must be computed, which is computed as an integral. Posterior predictive inference, one of the most powerful tools available, marginalises over the parameters, which is expressed as an integral. As a result, integration and expectations are found throughout probabilistic machine learning.

Following from this, a slightly more compelling answer is that marginalisation is inherently an integration, and hence aggregating over the variables we cannot control is vital. Moreover, removing nuisance variables requires a marginalisation. The particular value of these variables is irrelevant to whatever analysis we wish to perform, or cannot be controlled, and hence marginalising over these variables allows us to concentrate effort on the variables of interest. This will be important when we consider larger systems where reducing the number of variables improves the performance of inference. Therefore, integrating over uncertainty can be used to produce *summary statistics*, to simplify subsequent analysis. Furthermore, expectations allow us to consider the long-run or asymptotic properties of a system, aggregating over many individual random choices. For instance, the value drawn from a distribution is stochastic, but in expectation (as per (3.23)), the value of individual draws are aggregated over to analyse the system. Analysing systems by considering individual samples is often intractable, and hence we can make more powerful statements by considering the long-run behaviour of a system, expressed as an expectation.

The final reason we offer, and maybe the most compelling for scientists, is that we inherently take an expectation when we are forced into making a decision. Possessing the distribution over the position and velocity of a car and a nearby pedestrian is obviously important, but at some point we must process this distribution to decide whether or not to apply the brakes to avoid hitting the pedestrian. In this scenario, we are taking a conditional expectation over the relative position and velocities, conditioned on a particular action, to evaluate the probability that we hit the pedestrian. This probability is then minimised over

actions. Similarly, there are many experiments we could conduct to isolate a particular behaviour, however, we must actually pick an experiment to conduct, and hence we should pick the experiment that is expected to yield the most information. Alternatively, computing the full posterior predictive distribution may be prohibitively expensive at runtime. As a result, we may decide to instead compute the single parameter value that performs the best across potential outcomes, or, fit an approximate distribution that is more tractable to operate on. This single parameter value is computed as the expectation over all parameters. Therefore, a reasonably compelling answer as to why expectations are important is because we are often making a single choice to minimise the error, loss, or harm caused as a result of the uncertainty inherent in any given system.

3.3.1.2 Weighted and Unweighted Particles

In subsequent sections we will discuss using weighted and unweighted particles to approximate distributions. Unweighted particles are represented by the just value, $\mathbf{x}^{(k)} \in \mathcal{X}$ (although it could be considered as simply having a fixed weight of $1/K$), and distributions are represented as in (3.23). A weighted particle is instead a pair of values, one corresponding to the *normalised weight* of the particle, $W^{(k)} \in [0, 1]$, where $\sum_{k=1}^{K} W^{(k)} = 1$, and the actual value of the particle, $\hat{\mathbf{x}}^{(k)} \in \mathcal{X}$, as before. The weight represents the relative significance of that particle in the set of K particles in the finite-dimensional representation. We can construct a discrete representation of the distribution $\pi(\mathbf{x})$ as a sum of weighted delta functions:

$$\pi(\mathbf{x}) = \lim_{K \to \infty} \sum_{k=1}^{K} W^{(k)} \delta \left(\mathbf{x} - \hat{\mathbf{x}}^{(k)} \right). \tag{3.34}$$

Often, only *unnormalised weights*, $w^{(k)}$, can be computed. We (self-)normalise weights as:

$$W^{(k)} = \frac{w^{(k)}}{\sum_{k=1}^{K} w^{(k)}}, \tag{3.35}$$

where we discuss self-normalisation in more detail later. To produce unweighted samples from weighted particles we perform *resampling*, by sampling particles from a categorical distribution according to the normalised weights:

$$\mathbf{x}^{(k)} \leftarrow \hat{\mathbf{x}}^{\left(A^{(k)}\right)}, \quad \text{where} \quad A^{(k)} \sim \text{Discrete}\left(W^{(1:K)} \right), \tag{3.36}$$

where the index $A^{(k)}$ is often referred to as the *ancestor* of the particle, such that the k^{th} particle in the resampled particle set, denoted $\mathbf{x}^{(k)}$, is equal to $\hat{\mathbf{x}}^{\left(A^{(k)}\right)}$. These new particles, $\mathbf{x}^{(k)}$, are now unweighted samples (denoted by absence of the hat). Using weighted

samples when composing distributions is generally intractable, as appropriately modifying the weight as the particle passes through the system is often intractable. However, unweighted particles do not have a weight, and hence can be flowed through distributions without further consideration. Therefore, we will frequently generate weighted particles, then apply resampling to generate unweighted particles, which are then used in subsequent inference steps. It is worth noting however that resampling discards information, and hence should be avoided where possible.

There exist several algorithms for performing resampling. In (3.36), resampling reduces to simply sampling the ancestor from the discrete distribution, referred to as *multinomial resampling*. More sophisticated resampling techniques include *systematic* [Kitagawa, 1996], *stratified* and *residual* resampling [Doucet & Johansen, 2010]. These techniques can all be shown to produce lower-variance results than multinomial resampling. Practically, systematic resampling is nearly always used, owing to the relative ease of implementation and performance [Doucet & Johansen, 2010].

3.3.1.3 Accepting and Rejecting Samples

The algorithms we present are predicated on generating a discrete set of points that represent the target distribution. A common technique throughout these algorithms is to define a method for proposing samples independently from the target distribution, and then deriving and applying a rule to either *accept* or *reject* the sample, such that the set of accepted samples is distributed according to the target distribution. Rejected samples are permanently discarded. The differences between methods are often in the specific rule that is used, and how accepted and rejected samples are processed. However, we often say "the sample is accepted" without much recourse as to what actually happens thereafter. Most often at the start of inference an empty set of samples is defined, $X \leftarrow \emptyset$. When a proposed sample, denoted $\mathbf{x}'$, is accepted the value is added to the set, $X \leftarrow X \cup \mathbf{x}'$. If the sample is rejected, either no sample is added to the set, or, the previous value, denoted $\mathbf{x}$, is added to the set, $X \leftarrow X \cup \mathbf{x}$. The set X is then returned, where the constituent values are distributed according to the target distribution.

3.3.1.4 Why Aren't Evaluations Enough?

A slight idiosyncrasy of Monte Carlo methods is that we frequently evaluate the target density (at least up to a normalising constant) as part of generating samples, but instead use the density of samples as our estimate of the target density. The reasons as to why this is the

case are relatively simple, but are not always immediately apparent. Foremost, we can rarely actually evaluate the target density exactly. Instead, we are only able to evaluate the target density up to a normalising constant, which is often insufficient for comparing distributions conditioned on different values. However the discrete representation can be used to generate a normalised representation of the target density, by considering the density of samples, and hence the normalisation constant does not need to be computed.

Furthermore, as discussed previously, passing probability values through successive distributions is often intractable due to the complexity of the required mathematical manipulations. However, we can instead "flow" particles through distributions to construct more complex distributions. As we will see in later chapters, we are unable to evaluate densities in the neural models we employ, and, mathematical manipulation of these models is impossible. Therefore, instead of evaluating and operating on densities, we will use sets of particles to approximate the distributions of interest.

Finally, to obtain a characterisation of the target density using evaluations requires the density to be evaluated at regular intervals for each random variable. Such *quadrature* methods break down very quickly as the dimensionality of the system increases. While Monte Carlo methods may be less efficient than quadrature methods in low dimensions, the complexity of Monte Carlo methods are independent of dimension and hence they are observed to scale more effectively [Owen, 2013].

3.3.2 Basic Monte Carlo Methods

The properties discussed in the preceding section make Monte Carlo principles both exceptionally powerful and flexible. Hence, methods using Monte Carlo form much of the backbone of statistical inference. Methods that generate samples for Monte Carlo integrations are referred to as *samplers*. Samplers are designed to be task-agnostic algorithms that can be readily modified and tuned for specific applications. The core objective of these samplers is to generate samples from the target distribution, $\mathbf{x}^{(k)} \sim \pi(\mathbf{x})$, as per the right-hand side of (3.23), when sampling directly from this distribution is assumed to be intractable. Therefore, the *de facto* Monte Carlo approach is to draw samples from a more tractable distribution, referred to as a *proposal distribution*, and then "correcting" these samples such that they are distributed according to the target distribution. Once samples from the target distribution have been obtained, all the implications and freedoms afforded by Monte Carlo integration

automatically apply. We begin by introducing two foundational sampling methods: rejection sampling and importance sampling. While these methods are rarely used in isolation, they form the foundation upon which more complex methods build upon.

3.3.2.1 Rejection Sampling

The simplest sampling algorithm is *rejection sampling*. Each rejection sampling step generates a single unweighted sample distributed according to the target distribution, here denoted $\pi(\mathbf{x})$, and only requires that the unnormalised target density can be evaluated. A rejection sampler is specified through a proposal distribution, $q(\mathbf{x})$, which can be sampled from and evaluated exactly, and a strictly positive scaling parameter, $M \in \mathbb{R}_+$. The rejection sampling process is shown in Algorithm 3.1 and is shown graphically in Figure 3.3.

An individual rejection sampling step first samples a value from the proposal distribution, $\mathbf{x}' \sim q(\mathbf{x})$, and evaluates the densities of the (unnormalised) target density, $\pi(\mathbf{x}')$, and proposal, $q(\mathbf{x}')$, at this value. A scalar value, α, is then uniformly sampled between 0 and $Mq(\mathbf{x}')$. If $\alpha < \pi(\mathbf{x}')$ the sample is accepted. This sample is distributed according to the target distribution, and the sampler exits, returning $\mathbf{x}' \sim \pi(\mathbf{x})$. If $\alpha > \pi(\mathbf{x}')$, the sample is rejected. This sample is discarded and a new sample is drawn from the proposal. This process is repeated until a sample is accepted. The only requirement for the algorithm to be correct is that the proposal distribution multiplied by the scaling constant must be greater than or equal to the unnormalised target density for all state space:

$$Mq(\mathbf{x}) \geq \pi(\mathbf{x}), \quad \forall \mathbf{x} \in \mathcal{X}. \tag{3.37}$$

To show why this condition is required, and why rejection sampling yields correctly distributed samples, we first define a Bernoulli variable, a, indicating whether the sample is accepted. For a rejection sampler to yield correct samples we require the following condition to be true:

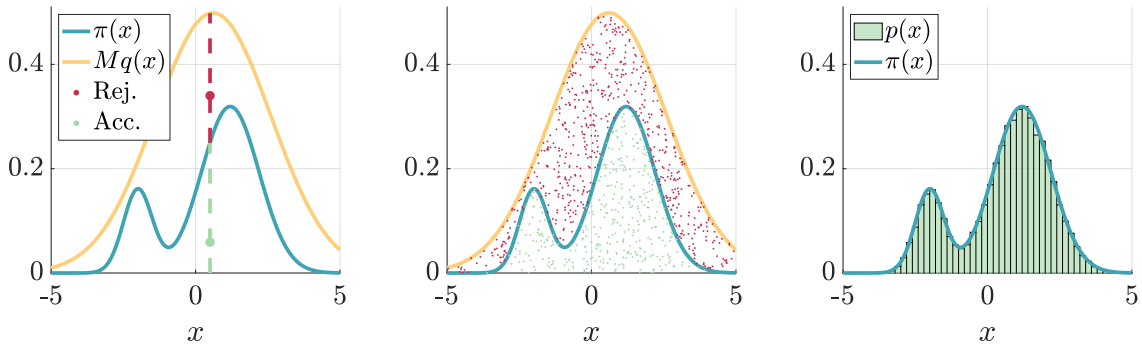
$$p(\mathbf{x}|a = 1) = \pi(\mathbf{x}). \tag{3.38}$$

Application of Bayes rule to the left-hand side yields:

$$\frac{p(a = 1|\mathbf{x})p(\mathbf{x})}{p(a = 1)} = \pi(\mathbf{x}), \tag{3.39}$$

and application of the law of total probability, noting that $p(\mathbf{x}) = q(\mathbf{x})$ yields:

$$\frac{p(a = 1|\mathbf{x})q(\mathbf{x})}{\int_{\mathbf{x}' \in \mathcal{X}} p(a = 1|\mathbf{x}')q(\mathbf{x}')\mathrm{d}\mathbf{x}'} = \pi(\mathbf{x}). \tag{3.40}$$



(a) **(b)** **(c)**

Figure 3.3: Illustration of rejection sampling. Figure 3.3a shows the target distribution in blue and a Gaussian proposal in orange. Rejection sampling proceeds by sampling an x value from the proposal, indicated by a dashed line, and then uniformly sampling a value between zero and the scaled proposal. Values above the target function are rejected (i.e. the red dot) whereas values under the target distribution are accepted (i.e. the green dot). Figure 3.3b shows $1,000$ proposed values, where accepted samples are shown in green and rejected samples shown in red. Figure 3.3c shows the histogram of the accepted samples are a good approximation of the target distribution.

For a particular $\mathbf{x}$ value, assuming that (3.37) is satisfied, we can define the probability of acceptance as:

$$p(a = 1|\mathbf{x}) = \frac{\pi(\mathbf{x})}{Mq(\mathbf{x})}. \tag{3.41}$$

Substituting this in:

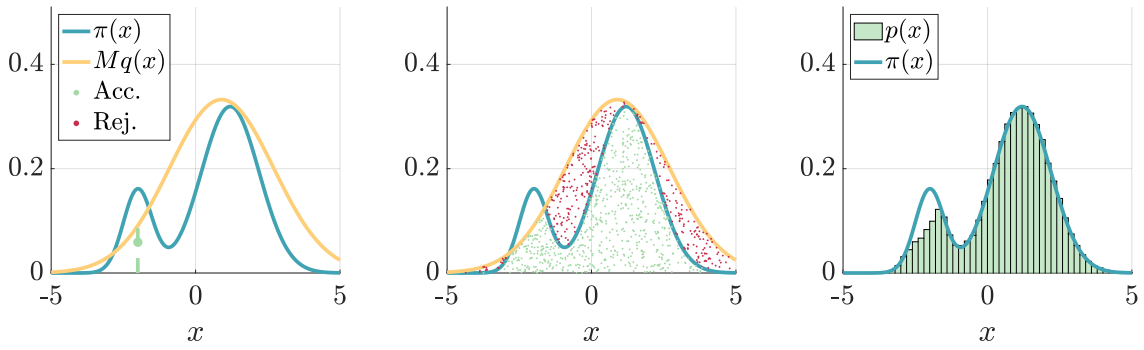
$$\frac{\frac{\pi(\mathbf{x})}{Mq(\mathbf{x})}q(\mathbf{x})}{\int_{\mathbf{x}\in\mathcal{X}}\frac{\pi(\mathbf{x})}{Mq(\mathbf{x})}q(\mathbf{x})\mathrm{d}\mathbf{x}} = \pi(\mathbf{x}). \tag{3.42}$$

Cancelling M and $q(\mathbf{x})$ yields:

$$\frac{\pi(\mathbf{x})}{\int_{\mathbf{x}\in\mathcal{X}}\pi(\mathbf{x})\mathrm{d}\mathbf{x}} = \pi(\mathbf{x}). \quad \square \tag{3.43}$$

If the distribution $\pi(\mathbf{x})$ is correctly normalised, the integral integrates to one. If the distribution $\pi(\mathbf{x})$ is not correctly normalised, the constant M must be picked such that $Mq(\mathbf{x})$ is greater than the unnormalised density at all $\mathbf{x}$ values. The desired result follows accordingly. As the number of samples drawn tends to infinity, the accepted samples perfectly approximate the target distribution.

This proof verifies that the rejection sampling algorithm generates a sample distributed according to the target distribution. To generate N samples, the rejection sampler is called N times. A useful, and somewhat unique, benefit of rejection sampling is that an independent and identically distributed (i.i.d.) sample is generated every time it is called. This is unlike



(a) **(b)** **(c)**

Figure 3.4: Figure showing the breakdown of rejection sampling by specifying a M value that is too low. We see in Figure 3.4a that the scaled proposal is not greater than the target at all points. Hence there is a region of the target distribution where samples are not generated, as can be seen in Figure 3.4b. This results in the incorrect distribution being recovered, as can be seen in Figure 3.4c.

other techniques that require large particle batches to be generated, before values resampled from within this batch can be considered as i.i.d.

However, rejection sampling is rarely used in practice for a number of reasons. If M is too low, the condition in (3.37) is not satisfied and the samples are not correctly distributed, as seen in Figure 3.4. Specifying M such that (3.37) is satisfied is a non-trivial task. Selecting a M too high will drive the rejection rate up (as can be seen from (3.41)), reducing the computational efficiency of the sampler. Rejection sampling also scales poorly to high-dimensions. Suppose the proposal distribution is uniform over a unit hypercube, and the target density is uniform over a unit hypersphere, centred in the hypercube. In one dimension, the acceptance rate is one, as both proposal and target are a unit-length line. In two dimensions the proposal is a square and the target is a circle centred in the square. Samples outside the circle, in the corners of the square are rejected, corresponding to a rejection probability of approximately 0.21. In three dimensions, the target is a sphere centred in a cube, and has a rejection probability of 0.47. Hence, as the number of dimensions increases, the acceptance rate actually tends to zero. Rejection samplers with a high rejection rate are computationally demanding, as the target distribution must be evaluated many times for just a single sample. This scaling behaviour prevents rejection sampling from being deployed efficiently in high-dimensional problems.

3.3.2.2 Importance Sampling

Importance sampling builds on rejection sampling by making use of samples that would otherwise be rejected. Instead of rejecting samples, importance sampling *weights* samples.

Algorithm 3.1 Rejection Sampling

Inputs: Target distribution $\pi(\mathbf{x})$, proposal distribution $q(\mathbf{x})$, scaling constant M.

Outputs: Single sample from target distribution $\mathbf{x} \sim \pi(\mathbf{x})$.

while True **do**
  $\mathbf{x}' \sim q(\mathbf{x})$
  $\alpha \sim \mathcal{U}(0, Mq(\mathbf{x}'))$
  if $\alpha \leq \pi(\mathbf{x}')$ **then**
    return $\mathbf{x}'$
  end if
end while

Algorithm 3.2 SNIS + Resamp

Inputs: Target distribution $\pi(\mathbf{x})$, proposal distribution $q(\mathbf{x})$, particle count K.

Outputs: Samples from target distribution $\mathbf{x}^{(1:K)} \sim \pi(\mathbf{x})$, norm. cost. Z.

$\hat{\mathbf{x}}^{(k)} \sim q(\mathbf{x})$
$w^{(k)} \leftarrow \pi(\hat{\mathbf{x}}^{(k)})/q(\hat{\mathbf{x}}^{(k)})$
$Z \leftarrow (1/K)\sum_{k=1}^{K} w^{(k)}$
$W^{(k)} \leftarrow w^{(k)}/Z$
$A^{(k)} \leftarrow \text{Discrete}(W^{(1:K)})$
$\mathbf{x}^{(k)} \leftarrow \hat{\mathbf{x}}^{(A^{(k)})}$
return $\mathbf{x}^{(1:K)}$, Z

Samples that would have been rejected receive a lower weight, but critically, still contribute to the estimation. An importance sampler is solely defined by a proposal distribution, $q(\mathbf{x})$, from which we can draw samples and evaluate. We begin by considering the case where we can evaluate the normalised target distribution, $\pi(\mathbf{x})$. Multiplying and dividing the target distribution by a proposal distribution, $q(\mathbf{x})$, and rearranging yields:

$$\pi(\mathbf{x}) = \pi(\mathbf{x})\frac{q(\mathbf{x})}{q(\mathbf{x})} = \frac{\pi(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x}). \tag{3.44}$$

We now define the weight function $w(\mathbf{x})$ as:

$$w(\mathbf{x}) = \frac{\pi(\mathbf{x})}{q(\mathbf{x})} \tag{3.45}$$

such that (3.44) becomes

$$\pi(\mathbf{x}) = w(\mathbf{x})q(\mathbf{x}). \tag{3.46}$$

Noting that for a given $\mathbf{x}$ we can evaluate $\pi(\mathbf{x})$, and $q(\mathbf{x})$ is a distribution from which we can both draw samples from *and* evaluate. We can therefore approximate $\pi(\mathbf{x})$ by sampling from $q(\mathbf{x})$, and then weighting the samples according to $w(\mathbf{x})$. The weight applied to each sample is referred to as an *importance weight*. This weight can be viewed as the relative weight of each of the N particles that best represents the target distribution:

$$\pi(\mathbf{x}) = \lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} w\left(\mathbf{x}^{(k)}\right)\delta\left(\mathbf{x} - \mathbf{x}^{(k)}\right), \quad \mathbf{x}^{(k)} \sim q(\mathbf{x}). \tag{3.47}$$

again noting the similarity between this expression and (3.34). Importance sampling is observed to scale more effectively to high dimensions and imposes fewer restrictions than

rejection sampling. Instead of requiring a (scaled) proposal that upper bounds the objective function, as in rejection sampling, IS only requires:

$$q(\mathbf{x}) > 0, \quad \forall \mathbf{x} \in \{\mathbf{x}' \in \mathcal{X} \mid \pi(\mathbf{x}') > 0\}. \tag{3.48}$$

For continuous variables, this is trivially satisfied by using a Gaussian proposal distribution. The elegance of importance sampling is again highlighted when considering expectations:

$$\begin{aligned}\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f(\mathbf{x})\right] &= \int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})\pi(\mathbf{x})\mathrm{d}\mathbf{x}, &(3.49)\\ &= \int_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})w(\mathbf{x})q(\mathbf{x})\mathrm{d}\mathbf{x}, &(3.50)\\ &= \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[f(\mathbf{x})w(\mathbf{x})\right]. &(3.51)\end{aligned}$$

This suggests that to evaluate integrals we simply sample from the proposal and weight the function evaluations by the importance weight under the chosen proposal and target distributions. The expectation itself can then be performed using Monte Carlo integration:

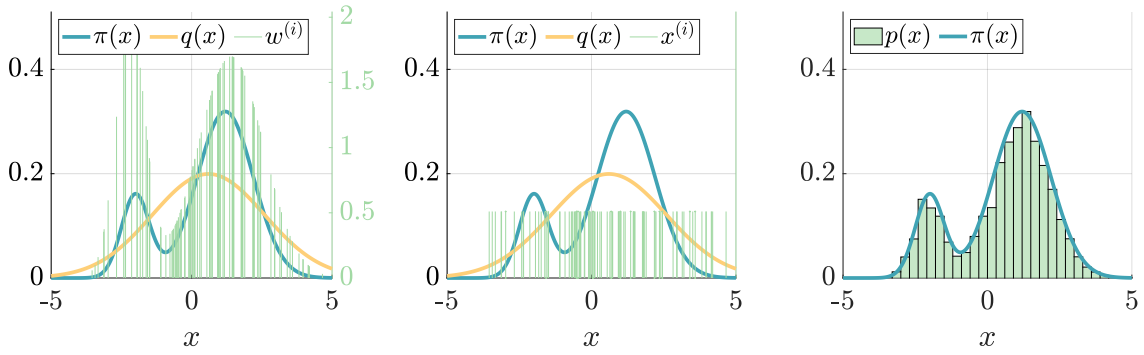
$$\mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[f(\mathbf{x})\right] = \lim_{K\to\infty}\frac{1}{K}\sum_{k=1}^{K} w\left(\mathbf{x}^{(k)}\right) f\left(\mathbf{x}^{(k)}\right), \quad \mathbf{x}^{(k)} \sim q(\mathbf{x}). \tag{3.52}$$

Crucially, importance sampling provides samples to compute unbiased Monte Carlo integrals. However, the convergence properties of IS are less straightforward than standard Monte Carlo integration. Many of the convergence properties are outlined in Owen [2013]. Importantly, satisfying the condition in (3.48) *does not* guarantee adequate performance in the finite sample regime, and hence designing more efficient proposals is an important part of using importance samplers. Furthermore, we assumed that we are able to evaluate the normalised target density, and does not natively provide us with a method for generating unweighed samples from the target density.

3.3.2.3 Normalisation Constants & Self-Normalized Importance Sampling

We assumed above that the target distribution was correctly normalised. However, as previously argued, the target distribution can often only be evaluated up to a normalising constant. Therefore, we now assume we can only evaluate an unnormalised target density, $\gamma(\mathbf{x}) = Z\pi(\mathbf{x})$, where Z is unknown. Hence, we can only compute the importance weight up to a normalisation constant, $w(\mathbf{x})/Z$. Substituting this into (3.46) and integrating with respect to $\mathbf{x}$ yields:

$$\int_{\mathbf{x}\in\mathcal{X}} \pi(\mathbf{x})\mathrm{d}\mathbf{x} = \int_{\mathbf{x}\in\mathcal{X}} \frac{w(\mathbf{x})}{Z} q(\mathbf{x})\mathrm{d}\mathbf{x}. \tag{3.53}$$



(a) **(b)** **(c)**

Figure 3.5: Graphical representation of importance sampling. Figure 3.5a shows the target and proposal distribution as in Figure 3.3. Discrete samples from the proposal are shown as vertical green bars, where the height corresponds to the importance weight of each sample. We can see that the samples are distributed according to the proposal, but the weight is lower for samples where the proposal density is higher relative to the target density. Figure 3.5b shows the equally weighted particles after applying self-normalisation and resampling, noting that many of the vertical bars represent multiple resampled particles. Figure 3.5c shows the histogram of the resampled particles. These particles are now distributed according to the target distribution, despite being sampled from the proposal distribution originally.

Noting that $\pi(\mathbf{x})$ is correctly normalized and hence the left of the equality must integrate to one. Moving Z through the integral yields:

$$Z = \int_{\mathbf{x}\in\mathcal{X}} w(\mathbf{x})q(\mathbf{x})\mathrm{d}\mathbf{x} = \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[w(\mathbf{x})\right]. \tag{3.54}$$

We can estimate this integral via Monte Carlo integration, drawing samples from $q(\mathbf{x})$:

$$Z = \lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K} w\left(\mathbf{x}^{(k)}\right), \quad \mathbf{x}^{(k)} \sim q(\mathbf{x}). \tag{3.55}$$

Somewhat remarkably, IS therefore provides a method to estimate the normalisation constant, Z, as the expected value of the unnormalised importance weights. This simple result will be critical to many of the more sophisticated algorithms we develop later, as it provides us with a tractable method for generating an unbiased and consistent estimate of the unknown normalising constant, similar to the result obtained in (3.28), using only evaluations of the unnormalised density.

We can therefore define the normalized weight of a particle as:

$$W^{(k)} = \frac{w(\mathbf{x}^{(k)})}{\sum_{j=1}^{J} w\left(\mathbf{x}^{(j)}\right)}, \; \mathbf{x}^{(k)} \sim q(\mathbf{x}), \; \mathbf{x}^{(j)} \sim q(\mathbf{x}), \tag{3.56}$$

where we have used a second set of samples to estimate the normalisation constant. Critically, however, it often suffices to *re-use* samples to estimate the normalising constant, contributing to the name *self-normalising*, i.e. the *same* samples are used in the computation in numerator

and denominator, removing the need to draw a second set of samples. This process above returns weighted samples, $(\mathbf{x}, W)^{(k)}$. Resampling, as introduced in Section 3.3.1.2, is then applied to generate unweighted samples by sampling (with replacement) particles from the multinomial distribution implied by the normalised importance weights. This produces unweighted samples distributed according to the target distribution.

It is worth noting that unless $K \to \infty$, re-using samples in this way introduces bias into Monte Carlo estimators. This bias is often neglected, as the bias reduces faster in N than the variance of the estimator, and hence is only a minor contributor to the error in the estimate [Doucet & Johansen, 2010]. While IS provides an asymptotically unbiased estimate of Z, it does not provide an unbiased estimate of $1/Z$. This can be shown simply through the application of Jensen's inequality. Self-normalised importance sampling allows us to generate samples distributed according to the target distribution in scenarios where only the unnormalised probability density can be evaluated, and, estimate the normalising constant. Self-normalised importance sampling with a resampling step is shown in Algorithm 3.2.

3.3.2.4 Limitations of Importance Sampling

Importance sampling is not without limitations. The proposal is independent of any observed data, and, must cover state-space such that the condition in (3.48) is satisfied. This necessarily entails that much of state-space has low probability mass under the proposal relative to any target density. This means that a small proportion of samples will dominate the normalised importance weights, increasing the variance of subsequent integrations, as just a few random samples are contributing to the estimation. This can be partially seen in Figure 3.5, where some particles have dramatically higher importance weights than other particles. In high dimensions, or with poorly tuned proposal distributions, this effect is amplified.

To demonstrate this, consider importance sampling in a HMM of length N, targeting the posterior $p(\mathbf{x}_{0:N}|\mathbf{y}_{1:N})$, using the true generative model as the proposal, $q(\mathbf{x}_{0:N}) = p(\mathbf{x}_{0:N})$, and weighting under the likelihood, $w(\mathbf{x}_{0:N}) = p(\mathbf{y}_{1:N}|\mathbf{x}_{0:N})$. The importance weights are computed as:

$$w\left(\mathbf{x}^{(k)}\right) = \prod_{n=1}^{N} p\left(\mathbf{y}_n|\mathbf{x}_n^{(k)}\right), \tag{3.57}$$

To compute the importance weight we multiply by a density at each time step. Hence the variance of this term grows linearly with time. For long (and therefore high-dimensional)

traces, this variance will be untenable and will allocate most of the probability mass to just a single particle after self-normalisation.

However, we have actually calculated the importance weight for the *whole* time series by calculating an importance weight at each step, and then multiplied these individual weights. If observations are available in real-time we can calculate individual weights as observations become available. We can then instead consider performing importance sampling *at each iteration*, discarding those particles with low importance weights and replicating those particles with high weight. This prevents samples that we already know are low weight from wasting the finite sample budget. This general approach is referred to as *sequential importance resampling*, or more commonly, *sequential Monte Carlo* (SMC).

3.3.3 Sequential Monte Carlo

We now introduce sequential Monte Carlo (SMC). The development of SMC has a somewhat protracted history, building towards what is now widely used and considered as the "default" SMC algorithm. Extensive tutorials on the history of SMC, the method itself and surveys of related work have been authored by Andrieu et al. [2010], Doucet & Johansen [2010] and Owen [2013]. The use of SMC is widespread, including performing inference in neural models [Warrington et al., 2019; Vogelstein et al., 2009], in programs with complex control flow [Wood et al., 2014; Ritchie et al., 2015], and in econometrics [Lopes & Tsay, 2011].

Sequential Monte Carlo can be considered as a generalization of importance sampling, exploiting the structure of the model to construct a series of more tractable intermediate importance samplers. This makes designing an efficient proposal distribution more straightforward by using observations *as they become available*. This moves the finite computational budget from regions of high probability *under the proposal* to regions of high probability *under the target distribution*. This shift reduces the variance of the estimator by reducing the number of particles with vanishing importance weight, in turn improving the efficiency of the sampler for a finite particle set size [Kantas et al., 2009]. Although intuitive, the complete proof of this is somewhat protracted and the result is sufficient for our purposes, and so we refer the reader to more advanced texts for more details [Del Moral, 2004; Del Moral et al., 2012; Doucet & Johansen, 2010; Kantas et al., 2009; Owen, 2013].

Most often SMC is used to perform posterior inference in state-space models. Indeed much of the work in this thesis reduces to using SMC to perform inference in parametric

HMMs. We use SMC throughout this thesis to estimate the posterior distribution over unobserved variables, $\mathbf{x}_{0:N}$, conditioned on the observed data, $\mathbf{y}_{1:N}$, and static global parameters of the model, $\boldsymbol{\theta}$. We also use SMC to estimate the *model evidence* for a given $\boldsymbol{\theta}$, $p(\mathbf{y}_{1:N}|\boldsymbol{\theta})$, defined as the density after marginalising over $\mathbf{x}$. Using SMC with a fixed $\boldsymbol{\theta}$ can then be leveraged to perform inference jointly over $\mathbf{x}_{0:N}$ and $\boldsymbol{\theta}$ conditioned on $\mathbf{y}_{1:N}$, referred to as *particle Markov chain Monte Carlo* (PMCMC), discussed more in Section 3.3.4.7. We present SMC generally at first, and then highlight how simplifications can be made that make SMC particularly applicable to HMMs.

3.3.3.1 Sequential Importance Sampling

We begin by reframing importance sampling in the context of SMC, before introducing additional elements to build up to full SMC. Most generally, we wish to construct a sampler drawing samples from the target density, $\pi_{0:N}(\mathbf{x}_{0:N})$, where $\mathbf{x}_n \in \mathcal{X}_n$. We assume that we can only evaluate the unnormalised density:

$$\gamma_{0:N}(\mathbf{x}_{0:N}) = Z_{0:N}\pi_{0:N}(\mathbf{x}_{0:N}), \tag{3.58}$$

where the unknown normalising constant is defined as:

$$Z_{0:N} = \int_{\mathbf{x}_{0:N}\in\mathcal{X}_{0:N}} \gamma_{0:N}(\mathbf{x}_{0:N})\mathrm{d}\mathbf{x}_{0:N}. \tag{3.59}$$

We note here that the densities $\pi_{0:N}(\mathbf{x}_{0:N})$ and $\gamma_{0:N}(\mathbf{x}_{0:N})$ are joint densities over all $N+1$ variables. We denote the factorisation of this densities as $\gamma_{0:N}(\mathbf{x}_{0:N}) = \gamma_0(\mathbf{x}_0)\prod_{n=1}^{N}\gamma_{0:n}(\mathbf{x}_n|\mathbf{x}_{0:n-1})$. We then denote the individual marginal distributions of the full distributions as $\pi_{0:N}(\mathbf{x}_n)$ and $\gamma_{0:N}(\mathbf{x}_n)$, where $0 \leq n \leq N$. We also assume that we can evaluate the unnormalised joint density of the first n variables, $\gamma_{0:n}(\mathbf{x}_{0:n})$, noting that this is *not* the marginal of the full joint distribution. Under certain additional restrictions introduced later, this assumption will be relaxed. We also note that $\mathcal{X}_n \neq \mathcal{X}_j$ for $n \neq j$ is permitted, as long as appropriate proposal distributions and weighting functions can be defined.

Importance sampling begins by defining a proposal over all latent states, $q_{0:N}(\mathbf{x}_{0:N})$. The condition on this distribution for an importance sampler to be valid is that this proposal places mass everywhere the target density places mass, $q_{0:N}(\mathbf{x}_{0:N}) > 0,\ \forall \mathbf{x}_{0:N} \in \{\mathbf{x}_{0:N} \in \mathcal{X}_{0:N} \mid \pi_{0:N}(\mathbf{x}_{0:N}) > 0\}$. Importance sampling then estimates the target density as:

$$\pi_{0:N}(\mathbf{x}_{0:N}) = \frac{w_{0:N}(\mathbf{x}_{0:N})q_{0:N}(\mathbf{x}_{0:N})}{Z_{0:N}}, \tag{3.60}$$

where $w_{0:N}$ is the unnormalised weight function:

$$w_{0:N}(\mathbf{x}_{0:N}) = \frac{\gamma_{0:N}(\mathbf{x}_{0:N})}{q_{0:N}(\mathbf{x}_{0:N})}, \tag{3.61}$$

and the normalising constant can be defined as:

$$Z_{0:N} = \int_{\mathbf{x}_{0:N} \in \mathcal{X}_{0:N}} w_{0:N}(\mathbf{x}_{0:N}) q_{0:N}(\mathbf{x}_{0:N}) \mathrm{d}\mathbf{x}_{0:N}, \tag{3.62}$$

$$= \mathbb{E}_{\mathbf{x}_{0:N} \sim q_{0:N}(\mathbf{x}_{0:N})} \left[w_{0:N} \left(\mathbf{x}_{0:N} \right) \right] \tag{3.63}$$

Importance sampling then draws samples from the proposal distribution and scores under the weight function to produce a weighted discrete representation of the target density. An unbiased estimate of the normalising constant can be computed by taking the empirical mean of the unnormalised importance weights. As the number of particles, K, tends to infinity, the approximation error tends to zero. We also assume that the proposal distribution has a tractable factorisation $q_{0:N}(\mathbf{x}_{0:N}) = q_0(\mathbf{x}_0) \prod_{n=1}^{N} q_n(\mathbf{x}_n | \mathbf{x}_{0:n-1})$. There are two types of proposal that can be combined in an SMC sampler: a *transition proposal* and a *rejuvenation proposal*. The former defines a distribution over $\mathbf{x}_n$ given $\mathbf{x}_{0:n-1}$. The latter is a distribution over an update to all $\mathbf{x}_{0:n}$ variables, used to improve sample diversity, and is often implemented as an MCMC update [Doucet & Johansen, 2010; Schuster et al., 2017]. Rejuvenation kernels are a more advanced SMC method and are less widely used. Hence, most SMC samplers use exclusively transition proposals. We do not discuss rejuvenation proposals here, and will refer to the transition proposal as simply "the proposal."

However, we recall that $\gamma_{0:N}(\mathbf{x}_{0:N})$ is a joint distribution over $N+1$ random variables, with factors $\gamma_{0:n}(\mathbf{x}_n | \mathbf{x}_{0:n-1})$, for $0 \leq n \leq N$. We can factorise the proposal once:

$$q_{0:N}(\mathbf{x}_{0:N}) = q_N(\mathbf{x}_N | \mathbf{x}_{0:N-1}) q_{0:N-1}(\mathbf{x}_{0:N-1}), \tag{3.64}$$

where $q_{0:N-1}(\mathbf{x}_{0:N-1})$ is a proposal distribution over the first N variables, and $q_N(\mathbf{x}_N | \mathbf{x}_{0:N-1})$ is a proposal over just the $N+1^{\text{th}}$ variable conditioned on the first N variables. We can then choose, somewhat arbitrarily, to rewrite (3.61) as

$$w_{0:N}(\mathbf{x}_{0:N}) = \frac{\gamma_{0:N-1}(\mathbf{x}_{0:N-1})}{q_{0:N-1}(\mathbf{x}_{0:N-1})} \times \frac{\gamma_{0:N}(\mathbf{x}_{0:N})}{\gamma_{0:N-1}(\mathbf{x}_{0:N-1}) q_N(\mathbf{x}_N | \mathbf{x}_{0:N-1})}, \tag{3.65}$$

$$= w_{0:N-1}(\mathbf{x}_{0:N-1}) \times \alpha_{0:N}(\mathbf{x}_{0:N}), \tag{3.66}$$

where $w_{0:N-1}(\mathbf{x}_{0:N-1})$ is the importance weight at the previous iteration, and $\alpha_{0:N}(\mathbf{x}_{0:N})$ is referred to as the *incremental importance weight*. Noting the recursive nature of (3.66),

we can fully factorise (3.61) as:

$$w_{0:N}(\mathbf{x}_{0:N}) = w_0(\mathbf{x}_0) \prod_{n=1}^{N} \alpha_{0:n}(\mathbf{x}_{0:n}). \tag{3.67}$$

This suggests a subtly different method for performing importance sampling, referred to as *sequential importance sampling* (SIS). Instead of proposing all $N+1$ latent variables at once, we iteratively propose the *next* $\mathbf{x}_n$ variable conditioned on $\mathbf{x}_{0:n-1}$ according to $\mathbf{x}_n \sim q_n(\mathbf{x}_n|\mathbf{x}_{0:n-1})$, and update the weight the of entire $\mathbf{x}_{0:n}$ sample by multiplying the preceding importance weight by the incremental importance weight. This formulation is simply a restructuring of IS, and hence does not "gain" anything over regular IS once all steps are complete. However, we do gain an estimate the distributions $\gamma_{0:n}(\mathbf{x}_{0:n})$ and the normalising constants $Z_{0:n}$ at each intermediate step. This will allow us to exploit techniques such as resampling (Section 3.3.1.2) and rejuvenation [Doucet & Johansen, 2010] to reduce the variance of the sampler.

It is convenient at this point to make a critical distinction. This distribution, $\gamma_{0:n}(\mathbf{x}_n)$, is *not* the marginal of the unnormalised target distribution, $\gamma_{0:N}(\mathbf{x}_n)$ (note the difference in the subscript of γ). The former is referred to as the *filtering* distribution, and is defined as the distribution over the n^{th} variable given the first n variables. The latter is referred to as the *smoothing* distribution, normally discussed specifically when $n = N$, and is defined as the distribution over any of the *previous* $N-1$ variables. Note however that "intermediate" smoothing distributions can also be constructed, i.e. the distribution over the first $n-1$ variables after the first n iterations, but this is rarely discussed in practice. Hence, SIS estimates all $N+1$ filtering and smoothing distributions. Crucially, it also computes the intermediate weight, $w_{0:n}(\mathbf{x}_{0:n})$, and hence estimates all intermediate normalisation constants, $Z_{0:n}$.

However, we noted that as the dimensionality of $\mathbf{x}$ becomes large, jointly proposing *all* $\mathbf{x}_{0:N}$ variables and then scoring *all* $\mathbf{x}_{0:N}$ is computationally inefficient, even when using SIS. This is because computation is focussed in regions where the *proposal distribution* places mass, as opposed to focussing computation where the *target density* places mass. However, we compute estimates of the intermediate distributions, and hence we can exploit this to improve efficiency.

3.3.3.2 SMC as Sequential Importance Sampling with Resampling

We now have everything required to define SMC, also referred to as *sequential importance resampling* or *particle filtering*. At its core, SMC simply interleaves SIS steps with a resampling step at each iteration. On a high-level, this resampling removes particles with low importance weight *early in the execution*. Those particles with high importance weight are then multiplicated. This is critical, as we only have a finite pool of particles, and hence concentrating the finite computational resources in regions that have high probability under the filtering distribution, and hence are more likely to have higher density under the target, as opposed to a high density under the proposal, will improve the performance of the sampler. This removal would occur at the end of regular IS/SIS, but, we can exploit the estimation of intermediate distributions during execution to preemptively remove these particles, and reallocate computational effort to higher probability regions of state space.

To show SMC generates correct samples, we first define how to generate samples distributed according to $\gamma_0(\mathbf{x}_0)$, using an importance sampler, as defined in Section 3.3.2.2. Particles are sampled from the initial proposal, $\hat{\mathbf{x}}_0^{(k)} \sim q_0(\mathbf{x}_0)$, and are self-normalised and resampled under the initial weight function, $w_0(\hat{\mathbf{x}}_0)$, to generate unweighted samples distributed according to $\gamma_0(\mathbf{x}_0)$.

We then consider how to propagate these unweighted samples, distributed as $\mathbf{x}_{0:n-1}^{(k)} \sim \gamma_{0:n-1}(\mathbf{x}_{0:n-1})$ (i.e. $n = 1$ in the previous step), to create unweighted samples distributed according to $\gamma_{0:n}(\mathbf{x}_{0:n})$. We can then apply this reasoning inductively to generate samples distributed according to $\gamma_{0:N}(\mathbf{x}_{0:N})$ as required.

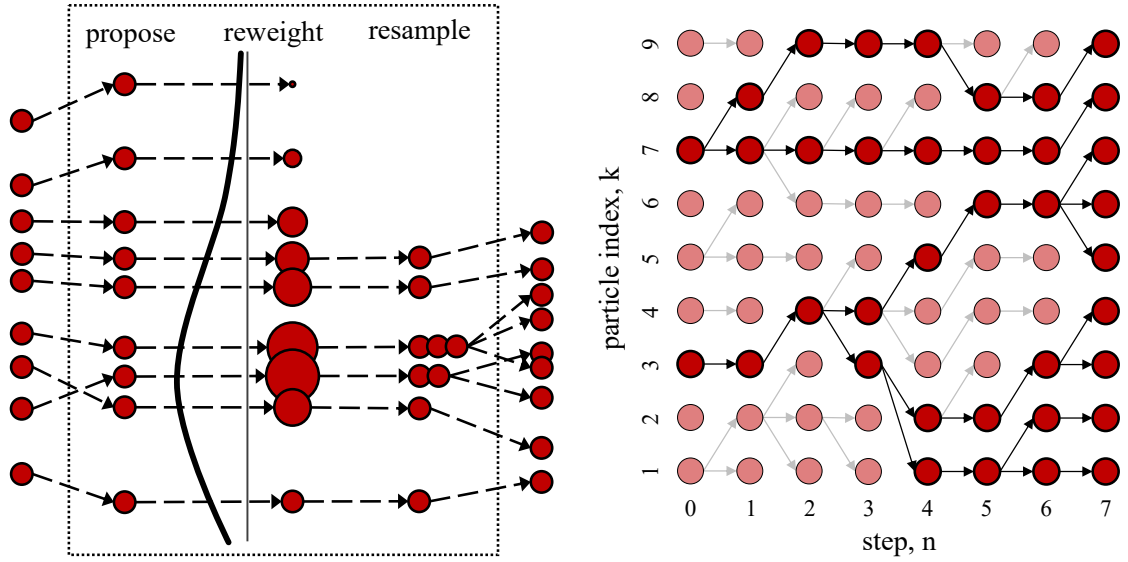
To do this, we again refer back to importance sampling. We construct an importance sampler, proposing particles from $q_n(\mathbf{x}_n|\mathbf{x}_{0:n-1})\gamma_{0:n-1}(\mathbf{x}_{0:n-1})$. The previous iteration has generated unweighted samples from the distribution $\mathbf{x}_{0:n-1}^{(k)} \sim \gamma_{0:n-1}(\mathbf{x}_{0:n-1})$. We can then pass these samples through the originally specified proposal distribution:

$$\hat{\mathbf{x}}_n^{(k)} \sim q_n\left(\mathbf{x}_n|\mathbf{x}_{0:n-1}^{(k)}\right), \quad \text{where} \quad \mathbf{x}_{0:n-1}^{(k)} \sim \gamma_{0:n-1}(\mathbf{x}_{0:n-1}), \tag{3.68}$$

$$\hat{\mathbf{x}}_{0:n}^{(k)} \leftarrow \left(\mathbf{x}_{0:n-1}^{(k)}, \hat{\mathbf{x}}_n^{(k)}\right). \tag{3.69}$$

We then compute the incremental importance weight of these particles using (3.66):

$$\alpha_{0:n}\left(\hat{\mathbf{x}}_{0:n}^{(k)}\right) = \frac{\gamma_{0:n}\left(\hat{\mathbf{x}}_{0:n}^{(k)}\right)}{\gamma_{0:n-1}\left(\hat{\mathbf{x}}_{0:n-1}^{(k)}\right) q_n\left(\hat{\mathbf{x}}_n^{(k)}|\mathbf{x}_{0:n-1}^{(k)}\right)}. \tag{3.70}$$



(a) (b)

Figure 3.6: Illustration of particle filtering. Figure 3.6a shows a single iteration of the particle filter. The unweighted particles from the previous step are used to propose the next set of particles. These particles are then reweighted under the weighting function (larger circles indicate higher weight). The particles are then resampled to yield unweighted particles. These particles are then used in the next iteration of the filter. Figure 3.6b shows a schematic of the ancestry of the particle filter. Particles without descendants at the last step are discarded (faded circle). The remaining particles, for each n, represent the smoothing distribution or marginal of the target distribution, $\pi_{0:N}(\mathbf{x}_n)$. We also see the effects of degeneracy, where the smoothing distribution at $n = 0$ is represented by just two particles. Note that the indexing of particles is arbitrary, and we separate the lineages for the purposes of illustration only. Figure 3.6a is an illustration of the process for going from step n to step $n + 1$ in Figure 3.6b.

Remembering that $\mathbf{x}_{0:n-1}^{(k)}$ are unweighted, we can compute the weight of each particle simply as the incremental importance weight:

$$w_{0:n}\left(\hat{\mathbf{x}}_{0:n}^{(k)}\right) = 1 \times \alpha_{0:n}\left(\hat{\mathbf{x}}_{0:n}^{(k)}\right). \tag{3.71}$$

We can then self-normalise these particles and apply resampling, sampling the ancestor index of each particle, $A_n^{(k)}$ from the distribution defined by self-normalised importance weights, to yield *unweighted* particles distributed according to $\gamma_{0:n}(\mathbf{x}_{0:n})$:

$$W_{0:n}^{(k)} = \frac{w_{0:n}\left(\hat{\mathbf{x}}_{0:n}^{(k)}\right)}{\sum_{i=1}^{K} w_{0:n}\left(\hat{\mathbf{x}}_{0:n}^{(i)}\right)}, \tag{3.72}$$

$$\mathbf{x}_{0:n}^{(k)} \leftarrow \hat{\mathbf{x}}_{0:n}^{\left(A_n^{(k)}\right)}, \quad \text{where} \quad A_n^{(k)} \sim \text{Discrete}\left(W_{0:n}^{(1:K)}\right). \tag{3.73}$$

Therefore, if we have an unweighted particle set at the $n-1^{\text{th}}$ step distributed according to $\gamma_{0:n-1}(\mathbf{x}_{0:n-1})$, we are able to use importance sampling to generate samples distributed

according to $\gamma_{0:n}(\mathbf{x}_{0:n})$. Hence, we have proven we are able to propagate particles as required above, and hence SMC provides a mechanism to produce the required samples. This process is illustrated in Figure 3.6.

We can also compute an unbiased estimate of the normalisation constant as the product of the expected incremental importance weights, where as the number of particles used tends to infinity the estimation becomes exact:

$$Z_{0:N} = \lim_{K\to\infty} \left(\frac{1}{K} \sum_{k=1}^{K} w_0 \left(\hat{\mathbf{x}}_0^{(k)} \right) \right) \prod_{n=1}^{N} \frac{1}{K} \sum_{k=1}^{K} \alpha_{0:n} \left(\hat{\mathbf{x}}_{0:n}^{(k)} \right), \tag{3.74}$$

where $\hat{\mathbf{x}}_{0:n}^{(k)}$ is sampled using the procedure outlined above. The full proof that this is unbiased in the finite-sample regime is somewhat involved, and so we refer the reader to Del Moral [2004] for full discussion of this. Many of the core SMC convergence results are also compactly presented in Chopin & Papaspiliopoulos [2020]. However, the important point is that SMC generates an unbiased estimate of the normalising constant by taking the product of the expectations of the unnormalised incremental importance weights at each step. This product can be performed as a summation in logarithmic space, significantly increasing numerical stability (and should always be implemented like this in practice).

Through this procedure of interleaving sequential importance sampling steps with resampling steps, we can generate samples distributed according to $\mathbf{x}_{0:N}^{(k)} \sim \pi(\mathbf{x})$, and, generate an asymptotically unbiased estimate of the normalising constant (as $N \to \infty$). SMC also scales more effectively to long state-space chains than regular IS by leveraging intermediate observations. This formulation is both powerful and general, as we have placed no restrictions on the functional form of any of the dependencies (other than requiring they construct a valid importance sampler), and have not restricted the space that each $\mathbf{x}_n$ variable exists in. Hence, SMC can be used to analyse complex combinations of discrete and continuous variables. To specify an SMC sampler all one has to define (in addition to the unnormalised target density) is the family of proposal distributions and the number of particles. The proposal distribution, and the weight function implied by the choice of proposal, is a powerful degree of freedom. The proposal can be different at different time steps, and hence "the" proposal distribution is actually a family of proposal distributions that can be tuned and optimised to improve inference performance. Designing more powerful proposals that automatically adapt to incorporate information from the observed data or

the current state to improve efficiency is an active area of research [Gu et al., 2015; Heng et al., 2020; Whiteley et al., 2014].

3.3.3.3 Vanilla SMC for Posterior Inference in Parametric HMMs

We explicitly define the most common application of SMC: performing posterior inference in HMMs. We make this clear as much of the work in this thesis reduces to using SMC to estimate the distribution over unobserved states and estimating the model evidence, using (3.74), in a HMM. The target distribution is the posterior distribution over unobserved, time-varying latent states, $\mathbf{x} = \mathbf{x}_{0:N}$, given observed data, $\mathbf{y} = \mathbf{y}_{1:N}$, and static global parameter values, $\boldsymbol{\theta}$. We assume that the model has the structure of a homogeneous parametric HMM, first introduced in Section 3.1.3, with joint distribution:

$$p(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = p(\boldsymbol{\theta})p(\mathbf{x}_0|\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{x}_{n-1}, \boldsymbol{\theta})p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}). \tag{3.75}$$

The posterior distribution over the time-varying latent state, assuming fixed parameters, is proportional to the joint distribution:

$$p(\mathbf{x}|\mathbf{y}, \boldsymbol{\theta}) = \frac{\gamma_{0:N}(\mathbf{x}_{0:N})}{Z_{0:N}} = \frac{p(\mathbf{x}, \mathbf{y}|\boldsymbol{\theta})}{p(\mathbf{y}|\boldsymbol{\theta})}, \tag{3.76}$$

$$= \frac{p(\mathbf{x}_0|\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{x}_{n-1}, \boldsymbol{\theta})p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})}{\int_{\mathbf{x}_{0:N} \in \mathcal{X}_{0:N}} p(\mathbf{x}_0|\boldsymbol{\theta}) \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{x}_{n-1}, \boldsymbol{\theta})p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}) \mathrm{d}\mathbf{x}_{0:N}}. \tag{3.77}$$

SMC in a homogeneous HMM is shown in Algorithm 3.3.

The particular variant of SMC we define here we describe as *vanilla SMC*. This particular variant is the most common formulation under which SMC is used. Vanilla SMC uses the transition kernel of the HMM as the proposal distribution:

$$q_{0:N}(\mathbf{x}_{0:N}) = p(\mathbf{x}_{0:N}|\boldsymbol{\theta}), \tag{3.78}$$

$$q_n(\mathbf{x}_n|\mathbf{x}_{0:n-1}) = q_n(\mathbf{x}_n|\mathbf{x}_{n-1}) = p(\mathbf{x}_n|\mathbf{x}_{n-1}, \boldsymbol{\theta}). \tag{3.79}$$

This choice has several key benefits. Firstly, we do not need to specify a further proposal distribution beyond what is already specified by the HMM. Secondly, this proposal is automatically *adapted* to the parameter values, $\boldsymbol{\theta}$. The third and most significant benefit, as we will show shortly, is that we do not actually need to be able to evaluate the proposal or the joint density. Instead, we only need to be able to sample from the proposal (and hence the transition kernel), and evaluate the likelihood. These are very weak requirements, as most often the HMM is defined through a transition kernel and emission function, from which sampling is easy and evaluating respectively is often easy.

Algorithm 3.3 SMC for posterior inference in homogeneous HMMs

Inputs: HMM transition kernel $p(\mathbf{x}_n|\mathbf{x}_{n-1})$, state prior $q_0(\mathbf{x}_0)$, proposal $q(\mathbf{x}_n|\mathbf{x}_{n-1})$, likelihood function $p(\mathbf{y}_n|\mathbf{x}_n)$, observed data $\mathbf{y}_{1:N}$, number of particles K.
Outputs: Samples from target distribution $\mathbf{x} \sim \pi(\mathbf{x})$, Log evidence estimate $Z_{0:N}$.

1: $\mathbf{x}_0^{(k)} \sim q_0(\mathbf{x}_0)$ // Initialize particles.
2: $Z_0 \leftarrow 0$ // Initialise log evidence.
3: **for** $n = 1 : N$ **do**
4: $\quad \hat{\mathbf{x}}_n^{(k)} \sim q(\hat{\mathbf{x}}_n|\mathbf{x}_{n-1}^{(k)})$ // Iterate particles.
5: $\quad w_n^{(k)} \leftarrow p(\mathbf{y}_n|\mathbf{x}_n^{(k)})p(\hat{\mathbf{x}}_n^{(k)}|\mathbf{x}_{n-1}^{(k)})/q(\hat{\mathbf{x}}_n^{(k)}|\mathbf{x}_{n-1}^{(k)})$ // Weight.
6: $\quad Z_n \leftarrow \frac{1}{K}\sum_{k=1}^{K} \log w_n^{(k)}$ // Log norm. constant.
7: $\quad Z_{0:n} \leftarrow Z_{0:n-1} + Z_n$ // Update log evidence.
8: $\quad W_n^{(k)} \leftarrow \log\left(w_n^{(k)}\right) - Z_n$ // Normalize weights.
9: $\quad A_n^{(k)} \sim \text{Discrete}\left(\exp W_n^{(1:K)}\right)$ // Sample ancestor.
10: $\quad \mathbf{x}_n^{(k)} \leftarrow \hat{\mathbf{x}}_n^{(A_n^{(k)})}$ // Resample particles.
11: $\quad \mathbf{x}_{0:n}^{(k)} \leftarrow \left(\mathbf{x}_{0:n-1}^{(A_n^{(k)})}, \mathbf{x}_n^{(k)}\right)$ // Append particles.
12: **end for**
13: **return** $\left\{\mathbf{x}_{0:N}^{(k)}\right\}_{k=1:K}$, $Z_{0:N}$

To see why using the transition kernel as the proposal is such a natural choice, both intuitively and mathematically, we substitute the proposal and target distribution into the definition of the incremental weight in (3.70) for the n^{th} step:

$$\alpha_{0:n}\left(\mathbf{x}_{0:n}\right) = \frac{p(\mathbf{x}_0|\boldsymbol{\theta})\prod_{n'=1}^{n} p(\mathbf{x}_{n'}|\mathbf{x}_{n'-1},\boldsymbol{\theta})p(\mathbf{y}_{n'}|\mathbf{x}_{n'},\boldsymbol{\theta})}{p\left(\mathbf{x}_n|\mathbf{x}_{n-1}\right)p(\mathbf{x}_0|\boldsymbol{\theta})\prod_{n'=1}^{n-1} p(\mathbf{x}_{n'}|\mathbf{x}_{n'-1},\boldsymbol{\theta})p(\mathbf{y}_{n'}|\mathbf{x}_{n'},\boldsymbol{\theta})}, \tag{3.80}$$

$$= p(\mathbf{y}_n|\mathbf{x}_n,\boldsymbol{\theta}). \tag{3.81}$$

Remarkably, using the transition kernel as the proposal means that the n^{th} incremental importance weight collapses to just the likelihood under the n^{th} observation. This means that we can trivially compute the weight required to resample in (3.71), or in Line 5 of Algorithm 3.3. Noting that the previous particle set is unweighted and distributed according to the previous intermediate distribution, this can be seen as re-directing samples to regions of state space that *also* explain the most recent observation. This also reduces computation of the normalising constant in (3.74) to the product of the average likelihoods of the particles prior to resampling. In this context, the normalising constant is the *model evidence*, $p(\mathbf{y}|\boldsymbol{\theta})$, a key quantity in model selection. Due to the flexibility in the specification of the model and ease of computing the model evidence, we will use this formulation throughout this thesis.

3.3.3.4 Summary

Before we proceed, we review what the mechanics of SMC provide. SMC builds on themes from importance sampling, and allows us to generate a discrete approximation of a target distribution. We placed no constraints on the form of the initial state distribution, transition kernel and observation model, and so these could be defined as complex distributions, programs, or simulators. However, computation of the normalising constants and weighting functions can be difficult. We therefore explicitly present so-called vanilla SMC, where we use the true generative model as the proposal. This particular choice is extremely common as it allows many factors to cancel, and hence only requires that we can sample from the generative model and evaluate the likelihood. As a result, the generative model can be specified through an arbitrary forward-only program, where computing densities is often intractable. It also provides, for negligible extra computational cost, an asymptotically unbiased estimate of the model evidence, which we will use extensively for evaluating the performance of the specified model.

SMC methods are not without drawbacks, however. The number of particles that need to be run may still be high, and hence SMC methods, particularly for expensive transition kernels, can be computationally demanding. Particle filters also suffer from *particle degeneracy*. As particles are necessarily removed at each time step, the smoothing distribution over early time steps can reduce to just a few particles, as is seen in Figure 3.6b. Further, there may be low diversity in the particles at each time step, if large numbers of particles are killed off, often leading to entire modes being missed. Critically, in simple SMC schemes, the proposal is independent of future data, and hence there may be large discrepancies between the true conditional probability and the proposal used, increasing the variance of the estimator. Ameliorating these drawbacks is a major component of SMC research. More sophisticated SMC algorithms are all contained under the framework we present. These algorithms often define more expressive proposal distributions, making use of the observed data in the proposal, or, rejuvenate particles at previous timesteps [Gilks & Berzuini, 2001; Whiteley et al., 2014; Gu et al., 2015; Le et al., 2018; Heng et al., 2020]. Despite the drawbacks of SMC, it is still one of the most widely used tools in Bayesian inference, owing to its flexibility, asymptotic correctness and ease of implementation.

3.3.4 Markov Chain Monte Carlo

We conclude this section on sampling methods by introducing a different framework for sampling from a target distribution: Markov chain Monte Carlo (MCMC). Self-normalised importance sampling and SMC generate a batch of samples from the target distribution. Instead, MCMC generates a sequence of samples, referred to as a *chain* [Robert & Casella, 2011]. The chain is designed such that as the length of the chain grows large, the states visited by the chain tend asymptotically to be distributed according to the target distribution [Metropolis et al., 1953].

The key observation in MCMC is that target distributions tend to be smooth. In IS, samples are generated independently from a fixed proposal, and hence samples are regularly proposed in regions of state-space where preceding samples yielded low probability evaluations. Instead, MCMC proposes new samples by making small perturbations to the current sample value. This yields samples that have a similar density under the target distribution as the current state. This allows computational effort to be focused in regions of state-space known to have significant density under the target distribution, as opposed to significant density under the fixed proposal distribution.

However, local perturbations mean that samples are inherently correlated, and hence successive states in the chain are not i.i.d. samples from the target distribution. Therefore the challenge is to design a method of proposing new states such that, as the chain grows long, the distribution over states visited by the chain tends towards the target density. Monte Carlo integrals can then be evaluated using the entire set of particles, or, individual particles can be resampled from the set of visited states to generate i.i.d. samples from the target distribution.

In this section we introduce MCMC methods. We begin by providing a brief introduction to ergodic theory, the mathematical framework that MCMC builds upon. We then introduce standard MCMC algorithms. We conclude by discussing extensions of the basic algorithms that improve efficiency and allow a wider class of problems to be solved. For more information on the theory that underpins MCMC, and more detailed proofs and convergence results, we direct the reader to more in-depth texts [Andrieu et al., 2003, 2010; Brooks, 1998; Neal, 1993; Owen, 2013].

3.3.4.1 Markov chains for Monte Carlo

We begin by first considering the desired outcome. We wish to develop a procedure for generating samples distributed according to the target distribution, denoted $\pi(\mathbf{x})$, where $\mathbf{x} \in \mathcal{X}$. We do not place restrictions on the type or dimensionally of the space $\mathcal{X}$. Unlike in importance sampling, we wish to instead use an operator that is repeatedly applied to generate a sequence of samples. Each state visited will be returned as an approximate sample from the target distribution. We denote the current state of the chain as $\mathbf{x}_{k-1}$, the next state of the chain as $\mathbf{x}_k$, and the length of the chain as K. Central to MCMC is the concept of an *invariant distribution*, denoted $p^*(\mathbf{x})$, and defined as:

$$p^*(\mathbf{x}_k) = \int_{\mathbf{x}_{k-1} \in \mathcal{X}} T(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_k) p^*(\mathbf{x}_{k-1}) \mathrm{d}\mathbf{x}_{k-1}. \tag{3.82}$$

The *transition kernel*, $T(\mathbf{x}_{k-1} \rightarrow \mathbf{x}_k)$, also referred to as a *transition operator*, defines the conditional distribution over the next state, $\mathbf{x}_k$, given the current state, $\mathbf{x}_{k-1}$, i.e. $p(\mathbf{x}_k|\mathbf{x}_{k-1})$. Intuitively, one can rationalise the invariant distribution by considering a large set of particles, $\mathbf{x}_{k-1}$, distributed according to $p^*(\mathbf{x})$. All of these particles are passed through the transition operator to yield a set of iterated particles, $\mathbf{x}_k$. According to (3.82), as the number of particles tends to infinity, the set of iterated particles is *also* distributed according to $p^*(\mathbf{x})$. This intuition is also why (under additional conditions introduced later) the invariant distribution is referred to as the *equilibrium* distribution [Neal, 1993].

The form of (3.82) is similar to the marginal distribution of a Markov chain:

$$p(\mathbf{x}_k) = \int_{\mathbf{x}_{0:k-1}} p(\mathbf{x}_k|\mathbf{x}_{0:k-1}) p(\mathbf{x}_{0:k-1}) \mathrm{d}\mathbf{x}_{0:k-1} = \int_{\mathbf{x}_{k-1}} p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}) \mathrm{d}\mathbf{x}_{k-1}, \tag{3.83}$$

first introduced in Section 3.1.3. Therefore, if we can design a Markov chain such that the invariant distribution is the target distribution, and given a set of samples distributed according to the target distribution, we can, in expectation, generate a new set of particles also distributed according to the target distribution. This suggests, in principal at least, we can simply iterate the chain to generate samples approximately distributed according to the target distribution, where the approximation becomes exact as $K \rightarrow \infty$:

$$\lim_{K \rightarrow \infty} \frac{1}{K} \sum_{k=1}^{K} \delta(\mathbf{x} - \mathbf{x}_k) = \pi(\mathbf{x}). \tag{3.84}$$

The invariant distribution of a Markov chain is a property of the chain itself, and not all Markov chains are guaranteed to have a well-defined invariant distribution. A Markov chain is

entirely defined by the transition operator and initial distribution. Therefore, we must design a transition operator and initial distribution such that an invariant distribution exists, and, is equal to the target distribution. Furthermore, we must also show that the distribution over states actually visited by a single Markov chain actually tends towards the target distribution as the chain is iterated. Designing bespoke transition and initial distributions that satisfy these conditions for a specific target density is extremely difficult, to the point of being practically impossible for all but the most trivial of target distributions. Therefore, it is often easier to design a generic transition operator that is guaranteed to work for all target densities and for any initial distribution.

Before continuing, we note that, crucially, the conditional independence of the Markov chain allows us to propose new samples conditioned on only the single preceding sample. Generating a new sample therefore has constant complexity. If this were not the case, as k grows large, the complexity would increase and rapidly become computationally intractable. Although there is no strict requirement for a Markov chain to be homogeneous, we assume homogeneity herein as this dramatically simplifies analysis, allowing us to consider expressions independent of time with identical transition dynamics. Analysing inhomogeneous chains is markedly more complex [Benaïm et al., 2017].

3.3.4.2 Designing Generic Chains

The first requirement is to design a transition kernel that renders the target distribution invariant. A sufficient (but not necessary) condition to ensure that this condition is satisfied is to show the operator satisfies *detailed balance* [Brooks, 1998]. Despite being a stricter condition than is required, proving the required invariance for a specific target distribution through other means is generally impossible, and hence detailed balance is often the only condition that is tractable to analyse. Detailed balance is a property of the transition kernel and target distribution, and is defined as:

$$T(\mathbf{x}_{k-1} \to \mathbf{x}_k)\pi(\mathbf{x}_{k-1}) = T(\mathbf{x}_k \to \mathbf{x}_{k-1})\pi(\mathbf{x}_k), \quad \mathbf{x}_{k-1} \in \mathcal{X}, \mathbf{x}_k \in \mathcal{X}. \tag{3.85}$$

Detailed balance can be interpreted as requiring the expected flow from $\mathbf{x}_{k-1}$ to $\mathbf{x}_k$ to be equal to the expected flow from $\mathbf{x}_k$ to $\mathbf{x}_{k-1}$. As a result, a chain that uses a transition operator that satisfies detailed balance is referred to as a *reversible chain*. Note that detailed balance does not determine the convergence properties of the chain.

To show that a transition operator that satisfies detailed balance renders the target distribution invariant we first integrate the left hand side of (3.85) with respect to the current state:

$$\int_{\mathbf{x}_{k-1}\in\mathcal{X}} T(\mathbf{x}_{k-1}\to\mathbf{x}_k)\pi(\mathbf{x}_{k-1})\mathrm{d}\mathbf{x}_{k-1} = \int_{\mathbf{x}_{k-1}\in\mathcal{X}} T(\mathbf{x}_k\to\mathbf{x}_{k-1})\pi(\mathbf{x}_k)d\mathbf{x}_{k-1}. \tag{3.86}$$

We can then move $\pi(\mathbf{x}_k)$ outside the integral as it is independent of $\mathbf{x}_{k-1}$:

$$\int_{\mathbf{x}_{k-1}\in\mathcal{X}} T(\mathbf{x}_{k-1}\to\mathbf{x}_k)\pi(\mathbf{x}_{k-1})\mathrm{d}\mathbf{x}_{k-1} = \pi(\mathbf{x}_k)\int_{\mathbf{x}_{k-1}\in\mathcal{X}} T(\mathbf{x}_k\to\mathbf{x}_{k-1})\mathrm{d}\mathbf{x}_{k-1}. \tag{3.87}$$

Noting that the transition operator is correctly normalized, we can immediately evaluate the second integral to be equal to one, yielding:

$$\int_{\mathbf{x}_{k-1}\in\mathcal{X}} T(\mathbf{x}_{k-1}\to\mathbf{x}_k)\pi(\mathbf{x}_{k-1})\mathrm{d}\mathbf{x}_{k-1} = \pi(\mathbf{x}_k). \tag{3.88}$$

This equation is identical to the definition of invariance in (3.82). Therefore, a transition operator that satisfies detailed balance, for a specific target distribution, is guaranteed to render that specific target distribution as the invariant distribution of the Markov chain. Crucially, the transition operator defined by composition of two or more transition operators, $T_1 \circ T_2 \circ \ldots \circ T_C$, that all satisfy detailed balance, *also* satisfies detailed balance.

To design a generic operator that renders *any* target distribution invariant, we therefore need to design a transition operator that satisfies detailed balance for any target distribution. The specific choice of transition operator is often the core of any particular MCMC algorithm. We introduce an example of such an operator in the next section.

A second property that is required for a generic chain to yield correctly distributed samples is for the chain to be *ergodic*. An ergodic chain is one that converges to a single invariant distribution regardless of the initial distribution or initial value, often surmised as the chain "forgetting its initialisation" [Robert & Casella, 1999]. An ergodic and reversible chain (that is reversible for any target distribution) is therefore guaranteed to converge to the target distribution. Crucially, this also means we can use any distribution to initialise the chain, and hence do not need to design a bespoke initial distribution. The invariant distribution of a reversible ergodic chain is described as the *equilibrium distribution* [Neal, 1993]. A chain is ergodic if it is *aperiodic* and *irreducible*.

A chain that is *periodic* is unable to reach certain states at certain iterations. Hence the k^{th} marginal over state is always dependent on the initial value. For instance, a chain that

alternates between odd and even numbers is periodic, and the equality in (3.82) cannot hold. Any chain can be made aperiodic trivially by introducing the possibility of a zero-move, i.e. the transition kernel places non-zero mass on the current state.

An irreducible chain is one that allows all states with non-zero probability under the target distribution to be reached from all other states in a finite number of steps. This means the chain, after being iterated for long enough, becomes independent of the initial value of the chain, and is therefore independent of the initial distribution used. In contrast, an irreducible chain will be trapped in a subregion of state-space determined by the initial value of the chain. A more precise requirement for irreducibility is that actually the chain must be able to reach any state with positive mass within a finite number of transitions:

$$\exists M \in \mathbb{Z}_+ \mid T^M(\mathbf{x}_{k-1} \to \mathbf{x}_k)/\pi(\mathbf{x}_k) > 0,\ \forall \mathbf{x}_k \in \{\mathbf{x} \in \mathcal{X} \mid \pi(\mathbf{x}) > 0\},\ \mathbf{x}_{k-1} \in \mathcal{X}, \tag{3.89}$$

where we denote M applications of the transition as T^M. The set of states that can be reached from a given state (and hence any other state also in that set) is called a *communicating space* [Andrieu et al., 2003]. An irreducible chain is therefore a chain where there is a single communicating space, equal to the entire state-space that has positive mass under the target distribution.

The *Fundamental Theorem* [Neal, 1993] states that an aperiodic homogeneous finite-state Markov chain that satisfies (3.89) is ergodic, and that the absolute error between the distribution defined by the samples gathered and the target distribution reduces by a constant factor for every step taken. The proof is somewhat involved, and so we refer the reader to Neal [1993] for the full proof. In short, this proof relies on the observation that the approximate distribution at the k^{th} iteration is representable as a mixture of the target distribution and an arbitrary second distribution. It is then possible to show that the contribution of the second (arbitrary and unknown) distribution decays by a constant multiplicative factor for every iteration. Therefore, the approximate distribution converges to the target distribution as $k \to \infty$. The rate of this convergence increases as the minimum value of the quotient in (3.89) over all of state-space increases. The movement of the chain around the state-space is referred to as *mixing*. Chains that mix quickly are typified by the quotient in (3.89) having a high value or a low valid M value, and hence converge to the target distribution more quickly. Designing chains that mix more effectively is a large part of MCMC research.

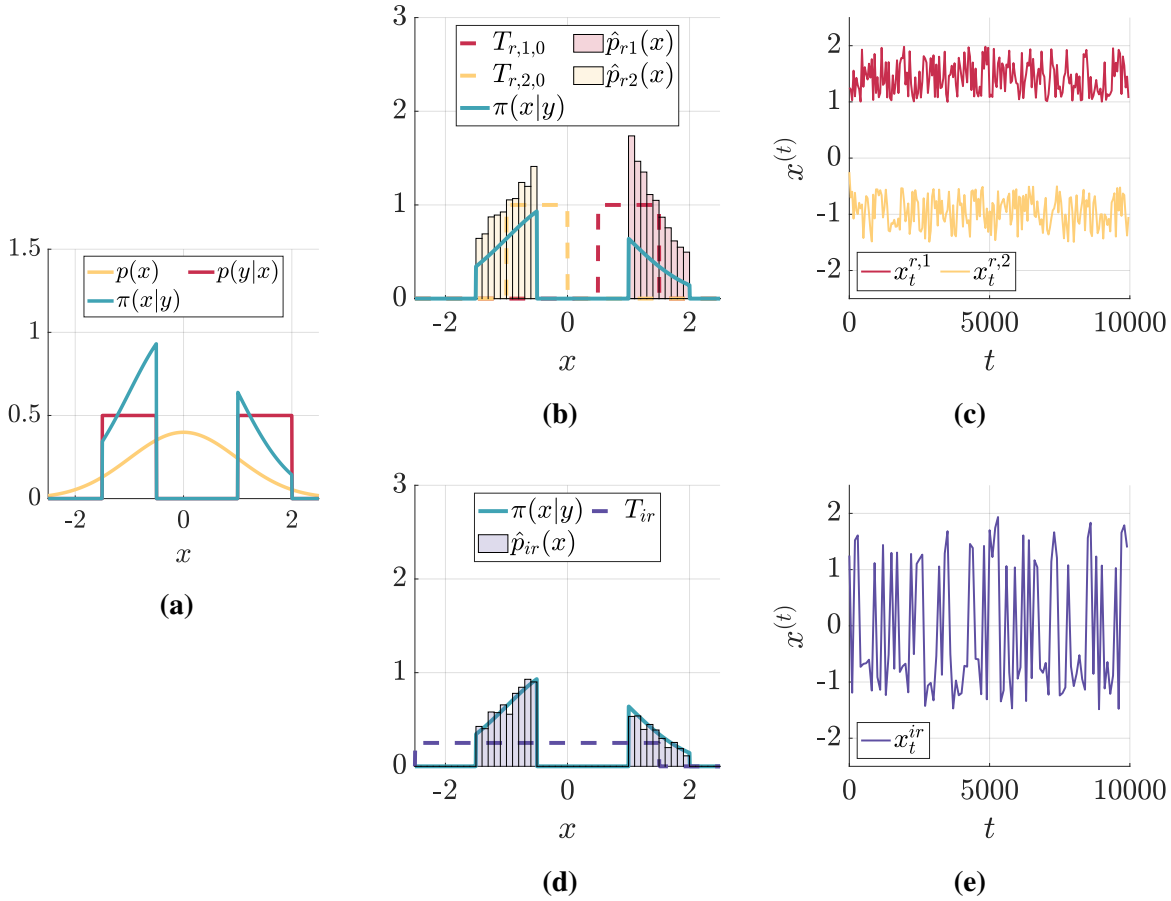


Figure 3.7: Irreducibility of MCMC chains. Figure 3.7a illustrates the model being studied. We wish to draw samples from the target density $\pi(x)$, which is proportional to the product of the prior, $p(x)$, and likelihood, $p(y|x)$. The likelihood takes the value 0.5 when x is in the interval $[-1.5, -0.5]$ or $[1.0, 2.0]$, and is 0 otherwise. This produces two regions of high-probability with a zero probability trough. Figure 3.7b and 3.7c show the results of a reducible chain. We show the results of running MCMC for two independent reducible chains, $r, 1$ in orange and $r, 2$ in red. By using a proposal distribution $q(x_k|x_{k-1}) = \mathcal{U}(x_{k-1} - 0.5, x_{k-1} + 0.5)$, shown as a dashed line for $x_{k-1} = -0.5$ and $x_{k-1} = 1$, the chain cannot cross between the models, and hence is reducible. Samples drawn from the chain are shown as a histogram. The first chain was initialized in the left partition and the second chain was initialized in the right partition. Neither chain is able to cross between modes and hence the communicating class is only one mode, shown by only mixing in a single mode in Figure 3.7c. The chain cannot "forget" in which partition it was initialized, is therefore reducible, cannot be ergodic, and hence does not produce correctly distributed samples. Figure 3.7d and 3.7e demonstrates the effects of widening the proposal distribution such that probability mass is placed on the other side of the partition, causing the chain to be irreducible (indicated by ir). This shows that in lieu of being a valid bespoke chain, a chain must be reversible *and* ergodic to yield correctly distributed samples.

A comparison of chains that are and are not irreducible is shown in Figure 3.7. Proving a chain is irreducible is not generally tractable, and hence is generally assumed. For continuous states, this condition is automatically met by using a Gaussian proposal distribution as the Gaussian places positive probability mass everywhere in its domain. Neal [1993] gives a more detailed discussion of the Fundamental Theorem and irreducibility. It is sufficient

however to know that a finite, aperiodic, irreducible, reversible and homogeneous Markov chain has exactly one invariant distribution that is equal to the target distribution.

Summary In this section we introduced a number of theoretical terms required to design and analyse Markov chains in the context of Monte Carlo sampling. We seek to create a Markov chain, where the states visited are distributed according to the target distribution. By exploiting locality the chain is able to focus samples, and hence computational effort, in regions proportional to probability under the *target* density. However, exploiting locality means that successive samples are not independent. Therefore, we instead require that as the number of samples grows large, the entire set of samples tends to the target distribution. We then define two conditions, reversibility and ergodicity. When these two conditions are satisfied, the set of states visited by the Markov chain are guaranteed to be distributed according to the target distribution as the length of the chain tends to infinity. What is outstanding is defining a generic transition operator that satisfies detailed balance for any target distribution. We therefore introduce the most widely used MCMC transition operator, the *Metropolis-Hastings* transition operator.

3.3.4.3 Metropolis Hastings

Metropolis-Hastings (MH) [Hastings, 1970; Metropolis et al., 1953] is by far the most widely used MCMC algorithm [Beichl & Sullivan, 2000; Chib & Greenberg, 1995]. At its core, MH simply specifies a particular form of the transition operator of a Markov chain, where this transition operator ensures the chain is aperiodic and detailed balance is satisfied *for any target distribution*. Crucially, MH only requires that the target density can be evaluated up to a normalising constant. This means that unnormalised distributions can be used as the target distribution. MH is illustrated in Figure 3.8.

On a high level, Metropolis-Hastings constructs a transition operator by proposing a local perturbation to the state of the chain, and then accepting or rejecting the proposed state. The probability with which the sample is accepted is determined by the *acceptance ratio*. If accepted, the next state of the chain is set to the proposed state. If rejected, the next state in the chain remains at the current state (a zero-move). The set of states visited by the chain, $\mathbf{x}_{0:K}$, are then unweighted samples from the target distribution as $K \to \infty$.

The MH transition operator begins by proposing a new state, $\mathbf{x}'_k$, conditioned on the current state, $\mathbf{x}_{k-1}$, from a proposal distribution, $q(\mathbf{x}'_k|\mathbf{x}_{k-1})$. The proposed state accepted with a probability given by the acceptance ratio:

$$A(\mathbf{x}_{k-1}, \mathbf{x}'_k) = \min\left(1, \frac{q(\mathbf{x}_{k-1}|\mathbf{x}'_k)\pi(\mathbf{x}'_k))}{q(\mathbf{x}'_k|\mathbf{x}_{k-1})\pi(\mathbf{x}_{k-1})}\right). \tag{3.90}$$

If a proposed value is accepted, denoted $a = T$, the next state, $\mathbf{x}_k$, is set to $\mathbf{x}'_k$. If the proposed value is rejected, denoted $a = F$, $\mathbf{x}_k$ is set to $\mathbf{x}_{k-1}$. The overall transition kernel for MH is therefore:

$$T(\mathbf{x}_{k-1} \to \mathbf{x}_k) = p(\mathbf{x}_k|\mathbf{x}_{k-1}) = p(\mathbf{x}_k, a = T|\mathbf{x}_{k-1}) + p(\mathbf{x}_k, a = F|\mathbf{x}_{k-1}), \tag{3.91}$$

$$= p(a = T|\mathbf{x}_{k-1}, \mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1}) + p(\mathbf{x}_k|\mathbf{x}_{k-1}, a = F)p(a = F|\mathbf{x}_{k-1}), \tag{3.92}$$

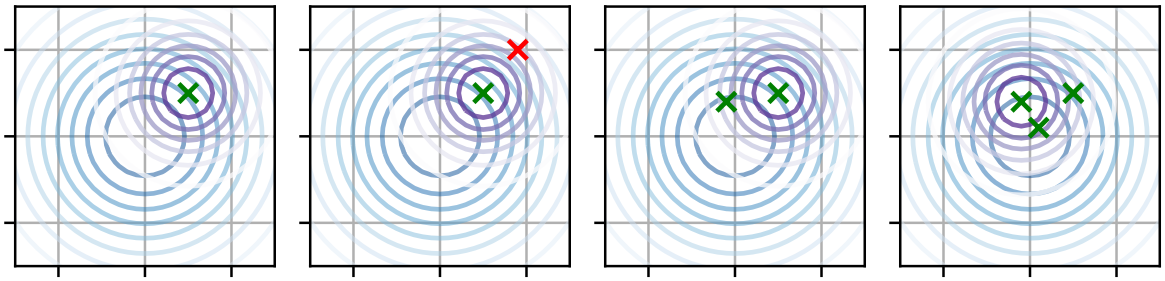
$$= A(\mathbf{x}_{k-1}, \mathbf{x}_k)q(\mathbf{x}_k|\mathbf{x}_{k-1}) + \delta(\mathbf{x}_{k-1} - \mathbf{x}_k)\Big[1 - \int_{\mathbf{x}\in\mathcal{X}} A(\mathbf{x}_{k-1}, \mathbf{x})q(\mathbf{x}|\mathbf{x}_{k-1})\mathrm{d}\mathbf{x}\Big]. \tag{3.93}$$

This transition operator therefore defines a mixture distribution over $\mathbf{x}_k$ conditioned on $\mathbf{x}_{k-1}$, fixed to the current state of the Markov chain. The first term describes the behaviour when the sample is accepted, and is equal to the probability of proposing a particular $\mathbf{x}_k$ conditioned on $\mathbf{x}_{k-1}$ *and* that the sample is accepted (which has probability $A(\mathbf{x}_{k-1}, \mathbf{x}_k)$). The second term describes when the sample is rejected. The square bracketed term is the probability that the step is rejected, calculated as one minus the probability that a step is accepted. This probability mass is then placed on the current state by multiplication with a delta function at the current state. It is important to note here that the transition operator is therefore a correctly normalised distribution over $\mathbf{x}_k$.

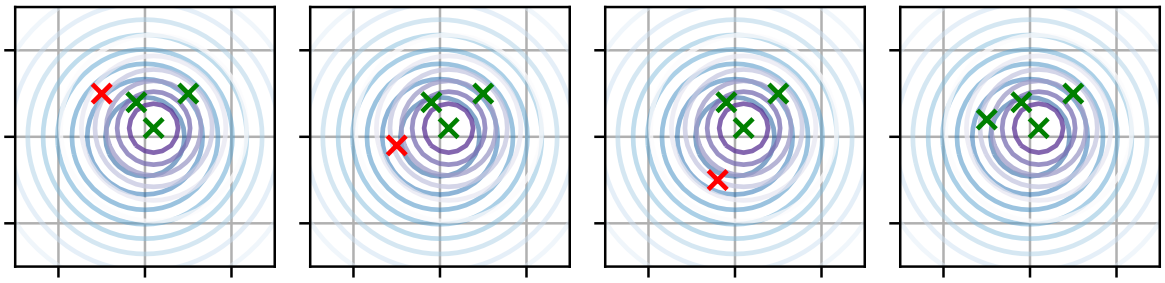
To show that Metropolis-Hastings satisfies detailed balance for any target distribution we consider the different cases in (3.93). If the step is rejected, detailed balance is satisfied trivially, as $\mathbf{x}_k = \mathbf{x}_{k-1}$. If the step is accepted we can consider just the the first term of (3.93) and set $\mathbf{x}'_{k-1}$ equal to $\mathbf{x}_k$:

$$T(\mathbf{x}_{k-1} \to \mathbf{x}_k) = A(\mathbf{x}_{k-1}, \mathbf{x}_k)q(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{3.94}$$

$$= \min\left(1, \frac{q(\mathbf{x}_{k-1}|\mathbf{x}_k)\pi(\mathbf{x}_k))}{q(\mathbf{x}_k|\mathbf{x}_{k-1})\pi(\mathbf{x}_{k-1})}\right) q(\mathbf{x}_k|\mathbf{x}_{k-1}). \tag{3.95}$$



(a) $k = 0$ **(b)** $k = 1,\ a = F$ **(c)** $k = 2,\ a = T$ **(d)** $k = 3,\ a = T$



(e) $k = 4,\ a = F$ **(f)** $k = 5,\ a = F$ **(g)** $k = 6,\ a = F$ **(h)** $k = 7,\ a = T$

Figure 3.8: Illustration of Metropolis-Hastings. The target distribution is shown in blue and the proposal distribution is shown in purple, centred on the current state of the chain, $\mathbf{x}_k$. The chain is initialised at $k = 0$ in Figure 3.8a. In Figure 3.8b a sample is proposed and is rejected (indicated by a red cross). The current state value is therefore the same in Figure 3.8c. A sample is proposed and accepted (indicated by a green cross and the proposal distribution shifting in Figure 3.8d). The algorithm continues until termination after eight steps. The green crosses in Figure 3.8h (some of which are duplicated) are (approximately) distributed according to the target distribution.

Substituting this into detailed balance yields:

$$\pi(\mathbf{x}_{k-1}) \min\left(1, \frac{q(\mathbf{x}_{k-1}|\mathbf{x}_k)\pi(\mathbf{x}_k))}{q(\mathbf{x}_k|\mathbf{x}_{k-1})\pi(\mathbf{x}_{k-1})}\right) q(\mathbf{x}_k|\mathbf{x}_{k-1}) = \tag{3.96}$$

$$\pi(\mathbf{x}_k) \min\left(1, \frac{q(\mathbf{x}_k|\mathbf{x}_{k-1})\pi(\mathbf{x}_{k-1}))}{q(\mathbf{x}_{k-1}|\mathbf{x}_k)\pi(\mathbf{x}_k)}\right) q(\mathbf{x}_{k-1}|\mathbf{x}_k). \tag{3.97}$$

Bringing the terms into the minimisation and rearranging inside the minimisation:

$$\min\left(\pi(\mathbf{x}_{k-1})q(\mathbf{x}_k|\mathbf{x}_{k-1}), q(\mathbf{x}_{k-1}|\mathbf{x}_k)\pi(\mathbf{x}_k))\right) = \min\left(\pi(\mathbf{x}_k)q(\mathbf{x}_{k-1}|\mathbf{x}_k), q(\mathbf{x}_k|\mathbf{x}_{k-1})\pi(\mathbf{x}_{k-1}))\right). \tag{3.98}$$

Since $\min$ is a commutative operation, $\min(a, b) = \min(b, a)$, the equality holds for *any* $\pi(\mathbf{x})$. This proof confirms that, as required, the MH transition operator admits *any* $\pi(\mathbf{x})$ as its invariant distribution. MH also guarantees that the chain is aperiodic, as there is a non-zero probability of a zero-move. If the chain then irreducible, and hence therefore also ergodic, then the set of samples produced by MH will converge to the target distribution as the number of samples tends towards infinity.

A crucial benefit of MH is that we are able to use unnormalised densities as the target

density. We are often unable to efficiently estimate the normalising constant and hence circumnavigating estimating this constant is a major benefit. Denoting the unnormalised density as $\gamma(\mathbf{x}) = Z\pi(\mathbf{x})$, and substituting this into the MH acceptance ratio:

$$A(\mathbf{x}, \mathbf{x}') = \min\left(\frac{q(\mathbf{x}|\mathbf{x}')\pi(\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})\pi(\mathbf{x})}, 1\right) = \min\left(\frac{q(\mathbf{x}|\mathbf{x}')\gamma(\mathbf{x}')/Z}{q(\mathbf{x}'|\mathbf{x})\gamma(\mathbf{x})/Z}, 1\right) \tag{3.99}$$

$$= \min\left(\frac{q(\mathbf{x}|\mathbf{x}')\gamma(\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})\gamma(\mathbf{x})}, 1\right), \tag{3.100}$$

we see that the normalising constants cancel. We can therefore evaluate the acceptance probability using just the unnormalised density. As a result, MH is particularly elegant when drawing samples from a posterior distribution. Representing the target posterior as $p(\mathbf{x}|\mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x})/Z$ we can write the MH acceptance probability as:

$$A(\mathbf{x}, \mathbf{x}') = \min\left(\frac{q(\mathbf{x}|\mathbf{x}')p(\mathbf{x}'|\mathbf{y})}{q(\mathbf{x}'|\mathbf{x})p(\mathbf{x}|\mathbf{y})}, 1\right) = \min\left(\frac{q(\mathbf{x}|\mathbf{x}')p(\mathbf{y}|\mathbf{x}')p(\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}, 1\right), \tag{3.101}$$

and hence avoids the often intractable task of estimating the marginal likelihood.

On a practical note, MH samplers often need to be *burned-in*. The initial state is not necessarily distributed according to the target density, and may be in a region of low probability under the target distribution. These low-probability regions can be flat, and escaping these regions can take many steps. As a result, these low-probability regions may be over-represented in a finite set of samples as the chain randomly moves around the space until a region with non-negligible density under the target is found. Burn-in prevents this by discarding these samples, only accumulating samples when the state of the chain is distributed according to the target density. The number of samples to discard must be selected heuristically.

The MH algorithm is shown in Algorithm 3.4, and is illustrated in Figure 3.8. An MH sampler is specified by the (unnormalised) target distribution, $\gamma(\mathbf{x})$, a proposal distribution, $q(\mathbf{x}'|\mathbf{x})$, a distribution over the initial state of the chain, $q_0(\mathbf{x})$, the number of samples to generate, K, and the number of burn-in samples, B. If the chain is ergodic and K is sufficiently large, the choice of initial distribution in arbitrary. The algorithm returns the set of states visited by the chain, $\mathbf{x}_{B:B+K}$, approximating the target distribution. As $K \to \infty$ the approximation becomes exact. We reaffirm that any two consecutive states are not i.i.d. samples from the target distribution, but instead that the *entire set* of K samples is distributed according to the target distribution. This set of samples can be used to perform Monte Carlo

Algorithm 3.4 Metropolis Hastings (MH)

Inputs: Target distribution $\gamma(\mathbf{x})$, proposal distribution $q(\mathbf{x}'|\mathbf{x})$, initial distribution $q_0(\mathbf{x})$, number of samples K, burn-in steps B.

Outputs: Samples from target distribution $\mathbf{x} \sim \gamma(\mathbf{x})$.

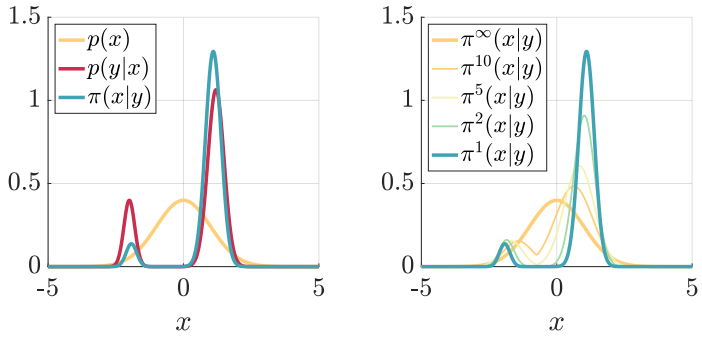
$\mathbf{x}_0 \sim q_0(\mathbf{x}_0)$ // Initialize chain.
for $k = 1 : (B + K)$ **do**
  $\mathbf{x}'_k \sim q(\mathbf{x}|\mathbf{x}_{k-1})$ // Propose next state.
  $A \leftarrow \min\left(1, \frac{q(\mathbf{x}_{k-1}|\mathbf{x}'_k)\gamma(\mathbf{x}')}{q(\mathbf{x}'_k|\mathbf{x}_{k-1})\gamma(\mathbf{x}_{k-1})}\right)$ // Calculate MH ratio.
  $\alpha \leftarrow \mathcal{U}(0, 1)$
  if $\alpha < A$ **then**
    $\mathbf{x}_k \leftarrow \mathbf{x}'_k$ // Sample is accepted.
  else
    $\mathbf{x}_k \leftarrow \mathbf{x}_{k-1}$ // Sample is rejected.
  end if
end for
return $\mathbf{x}_{B:(B+K)}$

integrals, construct marginals and conditional distributions, or be passed through further distributions to estimate more complex distributions. The set of states visited are unweighted particles, and hence i.i.d. samples distributed according to the target distribution can be sampled by uniformly drawing samples from this set.

One drawback of MH-based algorithms (and MCMC more generally) is the dependence on the proposal distribution. Often one can only define an *independent* proposal distribution, where individual dimensions of the state are perturbed independently. In high dimensions, jointly proposing perturbations from an independent proposal can lead to low acceptance rates. Therefore, a commonly used modification is to perform block-MH [Haario et al., 2005]. In block-MH, MH steps are taken on subsets or individual dimensions of the state. This is observed to increase mixing over independent proposals in high dimensional state-spaces [Andrieu et al., 2010]. However, if there is a strong correlation between the states (i.e. there is a sharp ridge in the target distribution), block samplers may result in a *lower* acceptance rate and slower mixing if correlated dimensions are not jointly proposed. Furthermore, updating individual dimensions can dramatically increase the cost of the chain, as each step still requires an evaluation of the target density.

3.3.4.4 Tempering

A method for improving acceptance rates and mixing of chains is to *temper* the target distribution [Angelopoulos & Cussens, 2008; Earl & Deem, 2005; Sambridge, 2013; Syed



(a) **(b)**

Figure 3.9: Tempering of the likelihood term in the posterior. Figure 3.9a shows a broad Gaussian prior distribution, and a comparatively narrow likelihood distribution. This creates an even narrower posterior distribution. Importance sampling using the prior distribution as the proposal will result in the majority of samples having very low weight, and a low effective sample size. MCMC may be ineffective as crossing between the modes is a low probably jump. Figure 3.9b shows the same prior and posterior distribution, but also shows a range of intermediate, tempered distributions. The prior corresponds to an "infinitely hot posterior," $\pi^{\infty}(\mathbf{x})$. The true posterior corresponds to a room temperature chain, denoted $\pi^{1}(\mathbf{x}|\mathbf{y})$. The tempering process slowly morphs the prior into the posterior as the temperature is reduced. These intermediate distributions can be exploited to enhance inference.

et al., 2020]. Tempering is the process of modifying the target distribution such that inference is more efficient. Most often, tempering smooths the target probability surface. This enables a Markov chain to move around the space more easily, leading to higher acceptance rates, better mixing and faster convergence. However, the tempered distribution is not equal to the original target distribution, and so must be used as a part of a larger inference algorithm where the tempering is eventually removed, and samples distributed according to the original target distribution are recovered. We therefore discuss tempering generally here, before introducing two specific algorithms that use tempering: *parallel tempering* and *annealed importance sampling*.

Any distribution can be tempered. The most straightforward way to temper a distribution is to raise it to a fractional power, $\pi^{1/T}(\mathbf{x}) = \sqrt[T]{\pi(\mathbf{x})}$, where $T \in \mathbb{R}_{\geq 1}$ is referred to as the *temperature*. For convenience we often use the *inverse temperature*, denoted $\beta = 1/T$. For unit temperatures, $T = 1$, the distribution is unmodified. As the temperature increases, the root term gradually flattens the distribution.

Tempering is particularly elegant when used in posterior inference. Our prior beliefs are often markedly more diffuse than the likelihood of the data, as illustrated in Figure 3.9. This can make inference using MCMC challenging as there can be deep probability troughs, where the probability of the chain crossing is small. A common form of tempering in posterior

inference is to instead raise the likelihood term to a power:

$$\pi^{1/T}(\mathbf{x}) = \pi^{\beta}(\mathbf{x}) = p^{\beta}(\mathbf{x}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{x})^{\beta} p(\mathbf{x}). \tag{3.102}$$

For infinite temperature systems ($T = \infty$) the tempered distribution is equal to the prior. For high, but not infinite, temperatures the contribution of the likelihood is attenuated relative to the contribution of the prior. For unit temperature systems ($T = 1$) we recover the original target distribution. This tempering effect is observed to smooth the ridges in the likelihood function, but retains our beliefs encoded by the prior.

An alternative method to temper posterior distributions is to subsample the data [Maclaurin & Adams, 2015; van de Meent et al., 2014]. The likelihood term is often a product over individual, independent likelihood terms for each data point. Each additional data point therefore further tightens the posterior distribution. Using less data reduces the amount of tightening, in turn tempering the distribution. Tempering in this way may also significantly reduce the cost of each evaluation if computing each likelihood term is independent.

3.3.4.5 Parallel Tempering

Parallel tempering (PT) is a method for accelerating MCMC by using tempering, while simultaneously producing samples according to the original target distribution, and excels particularly in systems with multiple modes [Altekar et al., 2004; Swendsen & Wang, 1986; Syed et al., 2020]. PT runs multiple parallel MCMC chains at different temperatures, referred to as *replicates*. At least one replicate is run with unit temperature. The high temperature chains can easily mix in the very smooth, high temperature surface. States with high probability under the (tempered) target distribution are then passed down into successively cooler chains, where samples eventually settle to be distributed according to the target distribution. This allows otherwise unreasonably large MCMC steps in each chain to be made, improving the mixing of cooler chains beyond what could be achieved with a single chain. Cooler chains can also use MCMC proposals with smaller length scales, yielding a higher acceptance rate and better mixing in the modes of the target distribution. At the end of inference, the higher-temperature chains are discarded and the unit temperature chain is retained as the samples from the originally specified target distribution. We illustrate the effects of parallel tempering on a bimodal distribution in Figure 3.10.

We denote the number of chains as $\mathcal{N}_r \in \mathbb{Z}_+$. Each chain uses a tempered target distribution, with temperatures $T_r \geq 1,\ r \in 1 : N_r$. Temperatures are almost always ordered

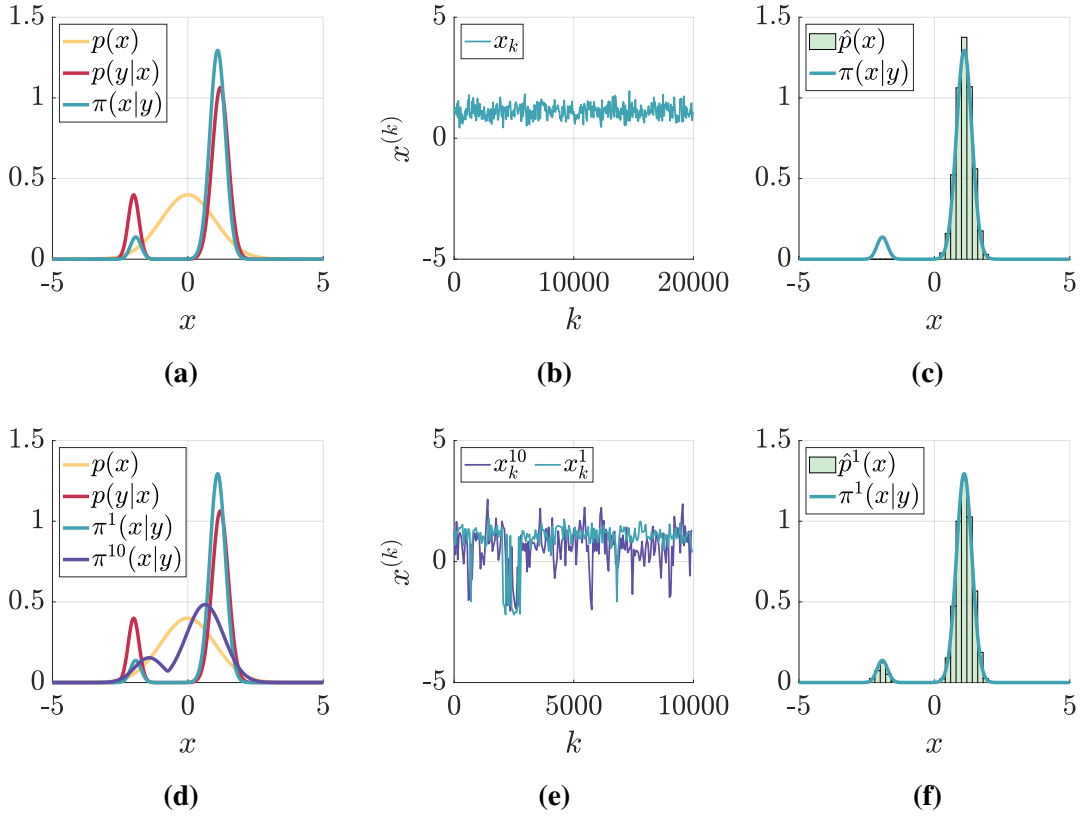


Figure 3.10: Parallel tempering in MCMC. *Top row*: In this example we target a bimodal posterior distribution, shown in Figure 3.10a in blue, by multiplying a Gaussian prior (shown in yellow), and a two component mixture of Gaussian likelihood (shown in red). We use a normal proposal distribution with variance 0.1. This narrow proposal may have been chosen because wider proposals resulted in an unacceptably low acceptance ratio. In Figure 3.10b and 3.10c we show the result of running a single MH chain. The chain has not mixed between the two modes, due to the narrow proposal distribution, and hence has poorly approximated the target distribution. *Bottom row*: We enhance the MCMC sampler using tempering. A replicate with the likelihood term is raised to the power $1/10$ (corresponding to a temperature of 10) is introduced, shown in purple in Figure 3.10d. Visibly, the depth of the trough between the two modes is reduced, meaning one would expect the tempered chain to mix more freely between the modes. This is verified in Figure 3.10e, where the samples from the tempered distribution (in purple) mix more, and, crucially, mix into the second mode. Occasionally, the unit temperature chain then swaps into the second mode, representing a transition with a distance nearly impossible in the original sampler. Samples from the coolest chain are then shown in Figure 3.10f, showing that the samples are still correctly distributed. Here, parallel tempering has solved the poor mixing experienced by the first chain, while also using the same number of function evaluations as the original sampler.

in descending order, such that $T_{r-1} \leq T_r \leq T_{r+1}$, and with $T_1 = 1$. When performing posterior inference, if $T_{\mathcal{N}_r} = \infty$, the replicates define a *ladder* of intermediate distributions that interpolate between the prior and target distribution, as shown in Figure 3.9b.

To show that PT defines a valid MCMC sampler, we first note that we can add additional random variables, often referred to as *auxiliary variables*, under the condition that the chain remains ergodic, reversible, and retains the original invariant distribution once the auxiliary

variables are marginalised over. This marginalisation can be performed trivially in Monte Carlo methods by simply discarding the auxiliary dimensions. We could therefore consider running $\mathcal{N}_r$ independent tempered MCMC chains, each performing reversible and ergodic MCMC updates, mixing in its own tempered space, and sampling from the corresponding tempered distribution with no interaction. The set of independent chains together define a single Markov chain on an extended space $\mathcal{X}^{\mathcal{N}_r}$. The overall target distribution of this extended Markov chain factorises as the product of the distributions targeted by each chain. To recover samples from the target density we then marginalise over the auxiliary states by simply discarding all but the $T = 1$ chain. This obviously brings no benefit however. The benefit in PT is in allowing information to pass between the chains.

We therefore introduce the possibility that replicates at different temperature can *swap* their states. Crucially, the composition of two reversible transition kernels is also reversible. Therefore, if we can define a reversible swap operator, and each replicate uses a reversible transition kernel, then the sampler operating on the extended space is also reversible. We can ensure each individual replicate is reversible by using the MH transition kernel. After each chain takes a tempered MH step updating its own state, we then apply a swap operator, also using a MH update. A swap between the i^{th} and j^{th} replicates is accepted with probability:

$$A(i,j) = \min\left(1, \frac{\pi^{\beta_i}(\mathbf{x}_k^j)\pi^{\beta_j}(\mathbf{x}_k^i)}{\pi^{\beta_i}(\mathbf{x}_k^i)\pi^{\beta_j}(\mathbf{x}_k^j)}\right). \tag{3.103}$$

To verify that this transition kernel is reversible we must show this transition kernel satisfies detailed balance. As in regular MH, we note that any rejected step trivially satisfies detailed balance. Hence we only need to consider the case when a swap is accepted. Assuming the swap operator that proposes the chains to swap is symmetric (e.g. a uniform distribution over chains), we can substitute (3.103) directly into (3.93) and then into detailed balance:

$$\pi^{\beta_i}(\mathbf{x}_k^i)\pi^{\beta_j}(\mathbf{x}_k^j)\min\left(1, \frac{\pi^{\beta_i}(\mathbf{x}_k^j)\pi^{\beta_j}(\mathbf{x}_k^i)}{\pi^{\beta_i}(\mathbf{x}_k^i)\pi^{\beta_j}(\mathbf{x}_k^j)}\right) = \pi^{\beta_i}(\mathbf{x}_k^j)\pi^{\beta_j}(\mathbf{x}_k^i)\min\left(1, \frac{\pi^{\beta_i}(\mathbf{x}_k^i)\pi^{\beta_j}(\mathbf{x}_k^j)}{\pi^{\beta_i}(\mathbf{x}_k^j)\pi^{\beta_j}(\mathbf{x}_k^i)}\right). \tag{3.104}$$

By again noting commutativity and bringing terms into the $\min$, we can show that this equality holds across all chains and for all target distributions. As a result, the swap operator is a reversible kernel, and hence can be composed with reversible MCMC replicates to yield a reversible sampler on the extended space. This means the set of replicates are still targeting the product of the marginal distribution of individual chains, and hence marginalising over

the auxiliary replicates yields the correct target distribution, as required. Similar to MH, unnormalised target distributions can be used as the normalising constants cancel.

A PT sampler therefore requires specification of an MCMC proposal per replicate, a method for proposing chains to swap, and a ladder of temperatures. While using PT requires a factor of N_r more computational effort to generate K samples compared to running a single chain, well tuned PT samplers are observed to improve inference performance over running N_r independent chains (or one chain for N_r times longer, as in Figure 3.10) [Earl & Deem, 2005], and hence are more efficient. Evaluating the set of target densities is also an *embarrassingly parallelisable* computation, defined as when parallelising the computation requires little (or no) effort and introduces negligible computational inefficiencies [Herlihy & Shavit, 2012]. Tasks that are embarrassingly parallelisable are therefore trivially amenable to efficient distribution over networked compute nodes. If the tempering is implemented by raising the likelihood to a power, evaluations of the likelihood (or model evidence) can be re-used to compute the swap ratio, and hence executing swaps is computationally inexpensive.

On a practical note, each replicate is itself a valid MCMC sampler and hence we have complete autonomy over the transition distribution used *within* that chain. Often, it is more efficient for hotter chains to have very wide proposal distributions to allow them to explore state-space effectively. Cooler chains then use narrower proposal distributions to mix more rapidly *within* local maxima. This allows parallel tempering to have favourable and tuneable exploit-explore characteristics, beyond what can be achieved with a single chain. Tuning PT algorithms can be difficult, exacerbated by high dimensions and when using many temperatures. This often leads to unknowingly wasting computational effort. Research into more sophisticated and adaptive tempering schemes aims to alleviate this design burden and further improve efficiency [Łącki & Miasojedow, 2016; Miasojedow et al., 2013; Syed et al., 2020].

3.3.4.6 Annealed Importance Sampling

A method that combines themes from MCMC, IS, tempering and SMC is *annealed importance sampling* (AIS) [Karagiannis & Andrieu, 2013; Neal, 2001]. Much like SMC, AIS defines a sequence of intermediate distributions. However, unlike SMC, the intermediate distributions are gradually cooler tempered target distributions, as shown in Figure 3.9b. Direct application of IS would simply sample from a proposal (normally the prior) and weight

the samples under the target. However, when the target distribution is high-dimensional, sharply peaked, or there is a large mismatch between the proposal and target distribution, many particles are assigned a low importance weight. Therefore, AIS exploits the fact that the difference between two successive tempered distributions is small, and hence each tempered distribution provides a good proposal distribution for the next tempered distribution. AIS applies an MCMC transition operator to each particle, scoring the particle under the *next* distribution in the temperature ladder, and then applies importance weighting. As the samples are cooled and passed down the temperature ladder, they gradually settle to be distributed according to the target distribution. On a high-level, this can be considered as batch-mode parallel tempering. We do not use AIS in this thesis, but note that it is an elegant method that has gained more attention recently [Brekelmans et al., 2020].

3.3.4.7 Particle Marginal Metropolis Hastings

The final MCMC algorithm we introduce is *particle marginal Metropolis-Hastings* (PMMH) [Andrieu et al., 2010; Kantas et al., 2015]. PMMH combines MCMC and SMC methods to efficiently target a joint distribution structured as $\pi(\boldsymbol{\theta}, \mathbf{x}) = p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. Often, $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ denotes the static and unknown global parameters of a generative model, and $\mathbf{x}$ represents a large and structured latent state, such as in a parametric HMM. Defining an MCMC proposal over such latent states can be difficult. Therefore, PMMH exploits our ability to structure this proposal as the true conditional $p(\mathbf{x}|\boldsymbol{\theta})$, from which we can sample using SMC. This results in a more readily defined, flexible and more efficient MCMC proposal distribution.

The proofs verifying the convergence characteristics of PMMH are very involved and are well beyond the scope of this thesis. We therefore provide a short derivation of PMMH, and direct the reader to Andrieu et al. [2010] for more rigorous discussion on the correctness of PMMH. However, it is sufficient for this thesis to understand that PMMH allows us to construct an MCMC sampler drawing samples from the joint distribution $\pi(\boldsymbol{\theta}, \mathbf{x})$, in an efficient and low-overhead manner, under only mild requirements and with little additional effort from the designer.

We begin by writing the MH acceptance ratio for the target distribution:

$$A(\boldsymbol{\theta}, \mathbf{x}, \boldsymbol{\theta}', \mathbf{x}') = \min\left(1, \frac{q(\mathbf{x}, \boldsymbol{\theta}|\mathbf{x}', \boldsymbol{\theta}')\pi(\mathbf{x}', \boldsymbol{\theta}')}{q(\mathbf{x}', \boldsymbol{\theta}'|\mathbf{x}, \boldsymbol{\theta})\pi(\mathbf{x}, \boldsymbol{\theta})}\right). \tag{3.105}$$

We are free to choose the form of our proposal distribution, but let us consider structuring the proposal distribution as:

$$q(\mathbf{x}', \boldsymbol{\theta}'|\mathbf{x}, \boldsymbol{\theta}) = q(\boldsymbol{\theta}'|\boldsymbol{\theta})\pi(\mathbf{x}'|\boldsymbol{\theta}'), \tag{3.106}$$

noting that $\mathbf{x}'$ is conditionally independent of $\mathbf{x}$ given either $\boldsymbol{\theta}$ or $\boldsymbol{\theta}'$. This proposal can be viewed of as first proposing a new parameter value conditioned only on the current parameter value, according to $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$, then proposing the latent variables conditioned on the proposed parameter value from the *true* conditional distribution. This latent state proposal is therefore *perfectly adapted* to the proposed parameter values (and any observed data) [Andrieu et al., 2010]. Substitution of this proposal into the acceptance ratio and simplifying yields:

$$A(\boldsymbol{\theta}, \mathbf{x}, \boldsymbol{\theta}', \mathbf{x}') = \min\left(1, \frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')\pi(\mathbf{x}|\boldsymbol{\theta})\pi(\mathbf{x}'|\boldsymbol{\theta}')\pi(\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})\pi(\mathbf{x}'|\boldsymbol{\theta}')\pi(\mathbf{x}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}\right) = \min\left(1, \frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')\pi(\boldsymbol{\theta}')}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}\right). \tag{3.107}$$

This justifies the somewhat arbitrary choice we made to propose values $\mathbf{x}'$ values independently from the current $\mathbf{x}$ value. Crucially, we do not need to be able to evaluate the density of a particular latent state. If the proposal is accepted, *both* the global parameter values and latent states are accepted. Hence, PMMH can be viewed as constructing an efficient MCMC sampler on an extended space, building a sequence of sample pairs, using a highly structured proposal. Showing that this acceptance ratio satisfies detailed balance is trivial.

PMMH constructs a Markov chain on a joint space, proposing new parameter values from a hand-crafted parameter proposal, and then proposing new latent states from the true conditional distribution. To use PMMH we therefore only need to define a proposal distribution over global parameters that we can sample from and evaluate, be able to *sample* from the true conditional distribution over latent states, and, generate an unbiased estimate of the marginal likelihood $\pi(\boldsymbol{\theta})$ for a given $\boldsymbol{\theta}$. These second two conditions are satisfied by performing a single SMC sweep conditioned on $\boldsymbol{\theta}$. The marginal likelihood approximation is used to compute the acceptance probability, and the surviving particles are used as samples from the proposal $\pi(\mathbf{x}'|\boldsymbol{\theta}')$. Defining a proposal over the global parameter values is typically easier than defining a proposal over the whole joint space, as the parameter values tend to be low-dimensional and on well-defined length scales.

As with MH, PMMH is also naturally suited to posterior inference, where the target distribution is the posterior distribution $p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y})$. New parameter values are proposed

independently of the data, $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\boldsymbol{\theta})$. A single SMC sweep is used to simultaneously estimate the model evidence used in the acceptance ratio, $p(\boldsymbol{\theta}|\mathbf{y})$, and to propose latent states from the conditional $p(\mathbf{x}|\mathbf{y}, \boldsymbol{\theta})$. The acceptance ratio is then computed as:

$$A(\boldsymbol{\theta}, \mathbf{x}, \boldsymbol{\theta}', \mathbf{x}') = \min\left(1, \frac{q(\boldsymbol{\theta}|\boldsymbol{\theta}')p(\mathbf{x}|\boldsymbol{\theta}, \mathbf{y})p(\mathbf{x}'|\boldsymbol{\theta}', \mathbf{y})p(\boldsymbol{\theta}'|\mathbf{y})/p(\mathbf{y})}{q(\boldsymbol{\theta}'|\boldsymbol{\theta})p(\mathbf{x}'|\boldsymbol{\theta}', \mathbf{y})p(\mathbf{x}|\boldsymbol{\theta}, \mathbf{y})p(\boldsymbol{\theta}|\mathbf{y})/p(\mathbf{y})}\right). \tag{3.108}$$

where the latent state densities and normalisation constants once again cancel, and hence do not need to be estimated. This can be extended more generally to allow PMMH to handle unnormalised target distributions.

Therefore, PMMH can be used to provide samples from the posterior over global parameter values and latent states conditioned on observed data. Most importantly, using PMMH separates the action of proposing latent states and global parameter values into sampling from a tractable and easily specified proposal over parameter values, and sampling from the true conditional using SMC. As a result, the application of PMMH is practically identical to alternating between independently using standard MH and SMC. The SMC procedure provides a highly efficient and perfectly adapted proposal over the latent states conditioned on the *proposed parameters and observed data*. As a result, the designer only has to define the required state-space proposals for the SMC sweep, and the parameter proposals for the MH updates, with no heed paid to the interaction of the two. This greatly reduces the burden on the designer and makes designing a valid sampler much more straightforward. We will use PMMH extensively in this thesis to efficiently sample from the posterior over global parameter values and latent states conditioned on observed data in parametric HMMs.

3.3.5 Summary

In this section we have introduced, in some detail, a range of methods for sampling from complex and arbitrary probability distributions, broadly referred to as Monte Carlo methods. We first introduced the benefits and intuitions behind Monte Carlo methods. We then introduced methods ranging from basic sampling, such as rejection and importance sampling, all the way through to advanced sampling methods, using tempering and multiple Markov chains to target complex joint distributions. We have also introduced some of the conditions for these algorithms to be provably correct. Rigorously re-proving many of these is outside the scope of this thesis and has been extensively covered in seminal textbooks [Bishop, 2006; Del Moral, 2004; Doucet & Johansen, 2010; Neal, 1993].

We will use these algorithms throughout this thesis to estimate distributions over unobserved latent variables given observed data in richly structured models. Crucially, these methods succinctly perform inference within a given generative probabilistic model, thereby clearly separating the task of specifying a model of a system (and the assumptions made as part of that model) from analysing the specified model. Performing such analysis by hand or through algebraic manipulation is almost inevitably intractable. Monte Carlo methods therefore provide a flexible and asymptotically unbiased framework to analyse the system. We can specify a probabilistic model of a system of interest, and then "hand-off" inference to a rigorously statistically defined procedure, to perform tasks such as inferring unobserved variables or quantitatively comparing hypotheses.

To both foreshadow the work presented shortly, and provide a concrete example, in Section 2.6 we define the Wicks neural model [Wicks et al., 1996] and calcium fluorescence model [Rahmati et al., 2016] using knowledge of the underlying system. These models define a parameterised state transition distribution, $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$, and emission distribution, $p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta})$. We can then define a prior over the physiological parameters, $p(\boldsymbol{\theta})$, using neuroscientific and domain-specific knowledge of the system. This model therefore defined a parametric HMM, where the emission distribution can be leveraged as the likelihood model. We can therefore straightforwardly use a vanilla SMC sampler to estimate the posterior distribution over unobserved neural potentials conditioned on observed calcium fluorescence traces and static parameter values, $p(\mathbf{x}|\mathbf{y}, \boldsymbol{\theta})$, and the model evidence, $p(\mathbf{y}|\boldsymbol{\theta})$. We can then use a PMMH sampler to draw samples from the joint distribution $p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y})$, to jointly estimate physiological parameters, physiologically important parameter values, and the interplay between these two. Performing this inference analytically is intractable, and therefore Monte Carlo methods provide us with a flexible and tractable method for estimating these distributions and quantities. We take this approach, or close derivatives thereof, to analyse neural systems in Chapters 4 and 5.

Before moving on we note however that simply satisfying the requirements of an algorithm does not necessarily guarantee that the algorithm will produce acceptable results in a reasonable time frame. Indeed, one can construct a valid Markov chain for any target distribution, run the chain for thousands of years, and still have poor characterisation of the

target density. Therefore, using the appropriate algorithm for a particular problem is often as practically important as ensuring the correctness of the algorithm itself.

3.4 Optimisation and Variational Methods

In this section, we briefly introduce optimisation and optimisation-based methods. Optimisation-based methods frame and approach problems in a fundamentally different way to the inference methods introduced previously. Indeed, inference and optimisation are often couched as competing frameworks, although, in practice, which framework is used is more often determined by the nature of the problem and the requirements of the solution, than as a result of any particular philosophical preference. Whereas inference aims to quantify the entire distribution, optimisation focuses solely on finding the maximum value of a function or distribution, and the inputs that yield this maximum value.

In this thesis we consider both inference and optimisation, and at least a working knowledge of optimisation is paramount for engaging with much of the machine learning literature. However, we only scratch the surface of optimisation, and hence we only introduce the high-level material and intuition needed for this thesis. We introduce optimisation and the accompanying terminology, and discuss the macroscopic differences between inference and optimisation, and the practical implications of each method, where there are close parallels to the discussion in Section 3.2. We then introduce *variational optimisation*, a particular optimisation algorithm we use in Chapter 4. We conclude by briefly introducing *variational inference*, which combines themes from inference and optimisation. We refer the reader to any one of the many seminal textbooks for more information [Boyd & Vandenberghe, 2004; Jain & Kar, 2017].

3.4.1 Optimisation Primer

Optimisation is one of the most important concepts in mathematics. Optimisation poses problems by searching for the *optimal* solution from the set of *feasible* solutions:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\Theta} f(\boldsymbol{\theta}), \tag{3.109}$$

where $\boldsymbol{\theta}$ is the variable we are searching over, often referred to as the *parameter*, and Θ is the space of all possible $\boldsymbol{\theta}$ values. The function $f : \Theta \to \mathbb{R}$, referred to as the *objective function*, maps a parameter value to a scalar value, $y = f(\boldsymbol{\theta})$. Optimisation is then the act of searching over the space of all possible $\boldsymbol{\theta}$ values for the single $\boldsymbol{\theta}$ value that yields the highest scalar

output value. The optimal parameter is often denoted $\boldsymbol{\theta}^*$, and the corresponding optimal value is denoted $\mathbf{y}^*$. The space of possible solutions, Θ, is often a single variable or set of variables. However, optimisation can be performed over anything that can be considered as an input, such as neural network architectures, combinatorial variables, or even the source code of programs. These inputs may have constraints placed on them (referred to as *constrained optimisation* [Bertsekas, 2014]). The objective function may also have additional terms added, referred to as *regularisers*, that modify the optimal solution to reflect certain desired properties, such as maximising speed while also minimising energy expenditure. The type of the variable being optimised and any additional regularisers or constraints often dictates the families of optimisation approaches that can be used. For all but the simplest of models, analytically solving for the optimal parameter is intractable, and hence many sophisticated numerical algorithms have been developed for performing this maximisation.

Optimisation methods can generally be grouped by their *order*. *Zeroth order methods* [Ruffio et al., 2011] only require evaluations of the objective function, $f(\boldsymbol{\theta})$. This is convenient as in many scenarios the gradient with respect to the input, $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$, cannot be easily computed. There is huge variety in zeroth order methods, from the simple and elegant *Nelder-Mead simplex method* [Nelder & Mead, 1965], to the physically inspired *simulated annealing* algorithm [Aarts & Korst, 1988; Kirkpatrick et al., 1983], to the sophisticated *Bayesian optimisation* [Mockus, 2012; Rainforth et al., 2015]. Zeroth order methods are flexible and easy to implement, although are generally limited in their ability to scale to high-dimensional input variables.

By far the most widespread optimisation methods are *first order methods*. These methods utilise the first derivative of the objective function with respect to the parameters, $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$. The gradient, which may be vector-valued, is the direction of steepest ascent in parameter space, i.e. the direction to travel in that yields the largest increase in the function value. This information is clearly highly informative when trying to maximise a function. As a result, *gradient ascent* (or descent), dating back to ideas first discussed by Cauchy [Cauchy, 1847; Lemaréchal, 2012] and Newton [Hildebrand, 1987; Newton, 1726], is the most widely used optimisation framework. The gradient is evaluated at the current value, and then a step is taken moving the current value in the direction of the gradient. This process is repeated until a computational budget is expended or the maxima is reached (signified by

$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = 0$). Gradient descent, and in particular its computationally favourable counterpart *stochastic gradient descent* [Robbins & Monro, 1951; Ruder, 2016], scales to much higher dimensions than zeroth order methods. Automatic differentiation [Baydin et al., 2017; Van Merriënboer et al., 2018], sometimes referred to as backpropagation, allows the first derivative of arbitrary functions with respect to the input to be computed with little user interaction. Gradient descent coupled with automatic differentiation has powered the deep learning revolution in machine learning. We will use stochastic gradient descent to train function approximators in Chapters 5 and 7.

Second order methods use second-derivative information, $\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$, to accelerate and stabilise learning [Battiti, 1992; Bottou et al., 2018]. These methods often have more favourable convergence properties, but require the computation and use of the second derivative matrix, which may be computationally demanding and unstable. Hence, second order methods are not as widely used as first other methods. Developing practical and scalable second order methods for machine learning is a growing area of research however [Brand et al., 2020; Dangel et al., 2019].

3.4.2 Optimisation versus Inference

Aside from the implementation and operational practicalities, inference and optimisation approach problems in fundamentally different ways. Optimisation searches a single *point estimate* of the variable value that maximises the objective function. In contrast, inference aims to quantify a target distribution over its entire support. Much like the difference between frequentist and Bayesian statistics, this difference is more than skin-deep. Hence, at least a cursory knowledge of the high-level and conceptual differences between these approaches is important.

The optimal parameter value conveys limited information about the *sensitivity* of the system to each parameter. Sensitivity is most simply defined as the derivative of the function value with respect to the parameters, $\mathrm{d}f(\boldsymbol{\theta})/\mathrm{d}\boldsymbol{\theta}$. In physical systems there is always uncertainty over the parameter values, uncertainty over the true system dynamics, or new data. As such, high sensitivity to parameters is often considered unsafe as small deviations from the optimal solution result in a dramatic reduction in function value (for instance, why we don't walk right on the edge of a pavement). As a result, simply recovering the simulator parameters that maximise the probability of the data may provide little-to-no

indication of how sensitive the overall system function is to each parameter. In contrast, inference quantifies the whole distribution, from which sensitivity to certain parameters can be identified as a sharply peaked distribution.

Extending this, the full inference result will quantify any multi-modality in the system. Multiple distinct modes that (approximately, if not exactly) explain the observed data suggests that the system has several distinct operating regimes. A simple point estimate will under represent the complexity of the system and preclude further insight into the system. Similarly, a point estimate of the parameters may lead to certain aspects of the system being unexplained. Maintaining a distribution over parameters or states allows uncertainty to be propagated through the system. Posterior inference, and particularly posterior predictive inference, can use the more expressive full distribution to encompass these corner cases. Finally, and similarly to frequentist and Bayesian methods, we may simply require a distribution over possible solutions, and not just a single solution.

However, optimisation is a fundamentally easier objective, often making problems computationally tractable when full inference is intractable. Optimisation is also generally more flexible due to the removal of the constraints that make the result of inference valid. This allows regularisers and constraints to be easily imposed to refine or guide the algorithm to a suitable solution. Imposing these constraints in inference may be difficult. Gradient information and tempering can be easily included in optimisation objectives, further accelerating the speed at which a good solution can be obtained. Finally, in many cases, a point estimate of parameters is sufficient, and, is often in fact easier to use.

3.4.3 Variational Optimisation

We motivated the use of first-order methods above. However, we also suggested that certain objective functions cannot be differentiated. This may be due to the presence of discrete variables or the prohibitive computational cost of estimating certain gradients. However, it is often possible to construct a Monte Carlo estimate of the gradient, and exploit the efficiency of a gradient ascent, without explicitly differentiating through the objective function. We now explore *variational optimisation* (VO) [Staines & Barber, 2012] that provides such an estimator. VO is illustrated in Figure 3.11. We will build on VO in Chapter 4.

VO maximises the objective function by maximising a lower bound on the objective function $f(\boldsymbol{\theta})$, with parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. To construct this bound, VO uses a parametrised

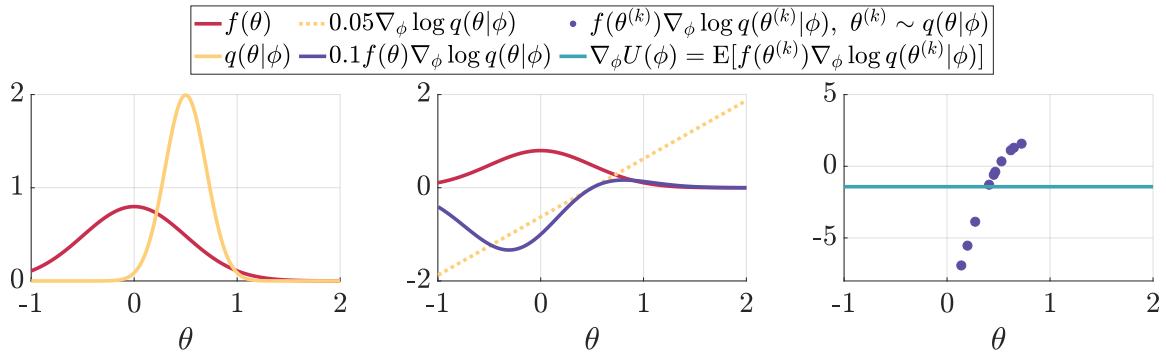


Figure 3.11: Illustration of the gradient operator in VO. The left plot shows the objective function, $f(\theta)$, and the proposal, $q(\theta|\phi)$. We use a Gaussian proposal with fixed variance (0.1), and that we are optimising just the mean of the proposal, i.e. ϕ is just the mean of the proposal. The mean is initialised to 0.5. To increase the value of $U(\phi)$, the proposal should move to the left, and hence we expect $\nabla_\phi U(\phi)$ to be negative. The middle plot shows the derivative of the logarithm of the proposal (scaled only for visualisation purposes), and the product of this logarithm with the objective function (again, scaled for visualisation purposes). In the right plot we then show evaluations of this product at points sampled from the proposal. Taking the expectation over these points returns a Monte Carlo estimate of the gradient, which, as expected, is negative. Hence, taking a gradient step using this gradient will move the proposal to the left.

proposal distribution over the variables being optimised, denoted $q(\boldsymbol{\theta}|\boldsymbol{\phi})$, parameterised by $\boldsymbol{\phi} \in \boldsymbol{\Phi}$. The gradient estimator then builds on the following inequality:

$$\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\boldsymbol{\theta}) \geq \mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\phi)}\left[f(\boldsymbol{\theta})\right] = U(\boldsymbol{\phi}), \tag{3.110}$$

where we define the bound as $U(\boldsymbol{\phi})$, equal to the expectation of the objective function under the proposal distribution, for notational convenience. The central observation of VO is that we can maximise this lower bound on the objective function by maximising $U(\boldsymbol{\phi})$ over $\boldsymbol{\phi}$. As the variance of q goes to zero, this bound becomes tight and the exact minimiser is recovered.

Although this inequality is trivial once understood, its exact origin can be opaque at first. We therefore offer a short "derivation" of this inequality. These arguments are not intended to be an absolutely rigorous derivation, but rather build intuition as to the relevance of this inequality and how it impacts the VO algorithm. Consider a set of parameter values, $\left\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(K)}\right\}$, where $\boldsymbol{\theta}^{(k)} \sim q(\boldsymbol{\theta}|\boldsymbol{\phi})$ for given value of $\boldsymbol{\phi}$. We denote the function evaluations at these values as $\left\{y^{(1)}, y^{(2)}, \ldots, y^{(K)}\right\}$, the optimal parameter as $\boldsymbol{\theta}^*$, and the corresponding optimal evaluation as y^*. By definition we can write:

$$y^* \geq \max(y^{(0)}, \ldots, y^{(K)}) \geq \frac{1}{K}\sum_{k=1}^{K} y^{(k)}, \tag{3.111}$$

where the first inequality states that the maximum function value is greater than or equal to the function value for all k. The second inequality then states that the maximum evaluation

from a set of values must be greater than or equal to the average of those evaluations. Both equality conditions only hold when $y^{(k)} = y^*$ for all k values. Recasting this inequality back to the original nomenclature yields, and as $K \to \infty$ yields:

$$y^* = \max_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \geq \max \left(f\left(\boldsymbol{\theta}^{(0)}\right), \ldots, f\left(\boldsymbol{\theta}^{(K)}\right) \right), \tag{3.112}$$

$$\geq \frac{1}{K} \sum_{k=1}^{K} f\left(\boldsymbol{\theta}^{(k)}\right) = \mathbb{E}_{\boldsymbol{\theta} \sim q(\boldsymbol{\theta}|\boldsymbol{\phi})} \left[f(\boldsymbol{\theta}) \right] = U(\boldsymbol{\phi}). \tag{3.113}$$

and hence the originally stated inequality. There are two conditions where the equality condition holds: when all sampled function evaluations are equal to the global function maximiser, $\boldsymbol{\theta}^*$. This means that if we allow the proposal distribution to collapse to a δ-function we are able to exactly minimise the objective function. The second condition is a generalisation of this to when the function has multiple global maxima. This condition can therefore be satisfied by q placing mass exclusively on any number of these maxima, yielding a family of q distributions that are *all* optimal. This second condition is somewhat of a corner-case, and only applies when using multi-modal proposal distributions. We do not explore this in this thesis.

Using this inequality we can recast the original optimisation to an optimisation of the lower bound, $U(\boldsymbol{\phi})$, over the parameters of the proposal distribution. We can then estimate the derivative of this bound by exploiting the log-derivative trick:

$$\nabla_{\boldsymbol{\phi}} U(\boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{\boldsymbol{\theta} \sim q(\boldsymbol{\theta}|\boldsymbol{\phi})} \left[f(\boldsymbol{\theta}) \right], \tag{3.114}$$

$$= \nabla_{\boldsymbol{\phi}} \int_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} q(\boldsymbol{\theta}|\boldsymbol{\phi}) f(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}, \tag{3.115}$$

$$= \int_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} f(\boldsymbol{\theta}) \nabla_{\boldsymbol{\phi}} q(\boldsymbol{\theta}|\boldsymbol{\phi}) \mathrm{d}\boldsymbol{\theta}, \tag{3.116}$$

$$= \int_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} f(\boldsymbol{\theta}) q(\boldsymbol{\theta}|\boldsymbol{\phi}) \nabla_{\boldsymbol{\phi}} \log q(\boldsymbol{\theta}|\boldsymbol{\phi}) \mathrm{d}\boldsymbol{\theta}, \tag{3.117}$$

$$= \mathbb{E}_{\boldsymbol{\theta} \sim q(\boldsymbol{\theta}|\boldsymbol{\phi})} \left[f(\boldsymbol{\theta}) \nabla_{\boldsymbol{\phi}} \log q(\boldsymbol{\theta}|\boldsymbol{\phi}) \right]. \tag{3.118}$$

We note here that we have assumed that we are able to pass the derivative under the integration, often referred to as *Leibniz's integral rule* [Flanders, 1973]. For this to be the case, we require that $f(\boldsymbol{\theta}) q(\boldsymbol{\theta}|\boldsymbol{\phi})$ is continuous in $\boldsymbol{\theta}$, the partial derivative with respect to $\boldsymbol{\phi}$ is also continuous, and that the integral limits and their first derivative are also continuous in the integral bounds (satisfied trivially by constant integration limits). We assume this throughout, noting that these conditions are often met without further consideration. Applying Monte Carlo

integration to estimate this expectation yields:

$$\nabla_{\boldsymbol{\phi}} U(\boldsymbol{\phi}) = \lim_{K\to\infty} \sum_{k=1}^{K} f\left(\boldsymbol{\theta}^{(k)}\right) \nabla_{\phi} \log q\left(\boldsymbol{\theta}^{(k)}; \boldsymbol{\phi}\right), \quad \text{where} \quad \boldsymbol{\theta}^{(k)} \sim q(\boldsymbol{\theta}|\boldsymbol{\phi}). \tag{3.119}$$

Therefore, to estimate the gradient of $U(\boldsymbol{\phi})$ with respect to the parameters of the proposal, we only need to define a parametrised and differentiable proposal distribution, $q(\boldsymbol{\theta}|\boldsymbol{\phi})$, and be able to evaluate the objective function, $f(\boldsymbol{\theta})$.

Using VO, we are able to construct a differentiable bound on an objective function without having to differentiate through the function. We can then use standard gradient ascent as before, using this estimator in-place of the exact gradient to maximise the lower bound, in turn maximising the function. The gradient estimator is a Monte Carlo approximation and hence evaluation of this gradient may be computationally expensive. However, the expectation in (3.118) is embarrassingly parallelisable, and hence we can leverage distributed supercomputer clusters with little overhead.

3.4.4 Variational Inference

We conclude this section by briefly introducing variational inference (VI) [Blei et al., 2017; Hoffman et al., 2013], which instead poses inference as optimising an approximate distribution such that it is as similar as possible to the true target distribution. VI is an incredibly powerful concept that has desirable properties for analysing real data, including lower computational demands and scaling to high dimensional problems more effectively than traditional inference methods, while retaining a notion of uncertainty unlike in pure optimisation. While we do not explicitly use VI in this thesis, many of the methods we go on to use make use of language and ideas from VI (and more broadly constrained optimisation [Bertsekas, 2014]) to make the assumptions made by the method explicit.

Instead of aiming to exactly quantify the target distribution, $\pi(\mathbf{x})$, VI sets out to recover the best approximation, referred to as the *variational distribution*, of the target distribution from a family of candidates, referred to as the *variational family*. The optimal variational distribution is defined as:

$$q^*(\mathbf{x}) = \arg\min_{q\in Q} \mathcal{D}\left(q(\mathbf{x}), \pi(\mathbf{x})\right), \tag{3.120}$$

where $q(\mathbf{x})$ is the variational distribution, Q is the variational family, and $\mathcal{D}$ is a divergence metric between two distributions. Assuming a parametric variational family, and under mild conditions, this optimisation can be performed using stochastic gradient descent. The

variational distribution that is closest to the true target distribution can then be used for downstream tasks as a convenient approximation of the true distribution. Solving for the variational distribution problem is often more computationally tractable compared to performing full inference and can exploit powerful automatic differentiation and stochastic gradient descent methods. However, unless the true posterior distribution is in the variational family (which is unlikely), there is a non-zero and unknown approximation error between the true distribution and the approximation. This approximation error may be arbitrarily large if the variational family is chosen poorly or the optimisation procedure is poorly tuned.

3.5 Neural Networks and Flows

We conclude our discussion of the machine learning material required for this thesis by introducing *artificial neural networks* (ANNs, or just neural networks) and *flows*.

Neural networks are, arguably, the single most used tool in machine learning today, providing a powerful and extensible toolbox for learning arbitrary functions. Networks that are able to identify cancerous tumours in medical images [Nasser & Abu-Naser, 2019; Stephan et al., 2002], generate photo-realistic images [Goodfellow et al., 2020; Zhu et al., 2017] and control bipedal robots [Xie et al., 2020] are now commonplace, and can be deployed on basic hardware. Although we do make use of neural networks in this thesis, knowledge of the low-level details are not central to engage with the work presented. We therefore include a short introduction, and refer the reader to LeCun et al. [2015] or Schmidhuber [2015] for a detailed history of the development and deployment of neural networks.

Flows are a more recent innovation that build on ideas from neural networks [Kobyzev et al., 2020; Rezende & Mohamed, 2015]. Flows restrict the flexibility of the network layers. However, these restrictions allow more powerful probabilistic statements to be made about the properties of the flow. Flows are less widely used than neural networks and have favourable characteristics for probabilistic modelling. We also use flows in Chapter 7, and therefore we include a more thorough introduction to flows.

3.5.1 Neural Networks

Artificial neural networks, historically referred to as multilayer perceptrons (MLP), or more generally referenced by the umbrella term *deep learning*, have achieved nearly cult status

in popular culture as being a standalone field[4] and the sole route to artificial intelligence. However, artificial neural networks are nothing more than a highly flexible parametric non-linear function, where the parameters can be learned efficiently using gradient descent. For instance, in the seminal work of Krizhevsky et al. [2012], a network, referred to as AlexNet, is presented that is capable of classifying images into one of 1000 possible classes. AlexNet far surpassed the performance of all hand-crafted classifiers previously developed, achieving an accuracy of 62.5%, and was learned using just the images and labels, with no user interaction beyond specifying the architecture and the optimiser, learning a function with hundreds of thousands of parameters automatically. This, arguably, was the beginning of the "deep learning revolution." Over the coming years, many state of the art, domain-specific solutions using hand-designed models and features (transformations of data), such as natural language processing [Devlin et al., 2018], object detection in images [Girshick et al., 2014], and detecting neural structures in EM tissue [Warrington & Wood, 2017; Ronneberger et al., 2015], were surpassed using "black-box" neural network approaches. The drawback of neural networks, typically, is that they require orders of magnitude more data than traditional approaches and have a large upfront computational cost to train. However, once trained, using the network is computationally cheap to use.

We do not delve into neural networks in more detail than this as part of this thesis. It is sufficient to understand that a neural network defines a parametrised function, where the parameters of the function can be learned to minimise a specified loss function quickly and efficiently using gradient descent given data. There is no guarantee that the function learned by the neural network perfectly represents the target function, and there is no guarantee that the function learned is even the best function representable by the neural network.

3.5.2 Normalizing Flows

As a direct result of their flexibility, statistical analysis of neural networks is notoriously difficult. Much of the difficulty analysing neural networks arises from the difficulty of inverting a neural network, or, evaluating densities that are parameterised by neural networks. Flows are a more recent innovation that aims to provide a flexible class of function and distribution approximators, using the same core mechanics and parlance as neural networks,

[4]Indeed, Mathworks, creators of MATLAB, published a course titled: "Deep Learning vs. Machine Learning: Choosing the Best Approach."

but avoiding the difficulty of analysing neural networks by imposing stricter constraints on the structure of individual layers [Kingma et al., 2016; Papamakarios et al., 2017; Rezende & Mohamed, 2015]. These constraints limit the diversity of layers, but, afford more powerful statistical analysis compared to the relatively unrestricted neural networks.

Flows learn a parametric mapping, denoted $f_{\boldsymbol{\phi}}(\boldsymbol{\epsilon})$, where $f \in \mathcal{F} : \mathcal{X} \times \boldsymbol{\phi} \rightarrow \mathcal{X}$ and $\boldsymbol{\epsilon} \in \mathcal{E} = \mathcal{X}$, that transforms samples distributed according to a fixed and simple *base* distribution, denoted $p_0(\boldsymbol{\epsilon}_0)$ and often assumed to be a unit Gaussian, through a series of N parameterised flow *layers*, individually denoted $f_{\boldsymbol{\phi}_n}$, into samples distributed according to a complex distribution. Each layer in the flow *warps* the distribution from the previous layer. Each individual warping function can be simple, but the composition of many warping layers leads to a complex transformation overall, as shown graphically in Figure 3.12. The learnable parameters of the flow, $\boldsymbol{\phi} \in \boldsymbol{\Phi}$, is the set of parameters used by each layer in the flow. A given flow architecture (and base distribution) therefore defines a parameterised family of distributions, $q_{\boldsymbol{\phi}}(\mathbf{x})$, by sampling from the base distribution, and passing the sampled value through the flow layers. The transformation defined by the flow, and hence the distribution, can be changed by varying the parameters of the flow.

Given a target distribution, $\pi(\mathbf{x})$, a flow is learned (by learning $\boldsymbol{\phi}$) to minimise the divergence between the target distribution and the distribution defined by the flow. Once trained, $q_{\boldsymbol{\phi}^*}(\mathbf{x})$ can be used as an approximation of $\pi(\mathbf{x})$ that can be sampled from and evaluated. Crucially, flows can still be learned if only a set of samples from the target density are available, $\left\{\mathbf{x}^{(k)}\right\}_{k=1:K}$, by maximising the probability of the observed points under $q_{\boldsymbol{\phi}}(\mathbf{x})$. This can be shown to equivalent to minimising the divergence between the unknown target $\pi(\mathbf{x})$ and the distribution defined by the flow.

Crucially, each individual layer is constrained to be invertible. Specifically, each layer in a flow implements a *change of variables*, and hence each layer yields a correctly normalised distribution (hence the name) that can be evaluated. This invertibility means that the inverse of the *entire* flow can be computed. We are therefore able to flow any $\mathbf{x}$ value back to an $\boldsymbol{\epsilon}_0$ value by inverting the flow, $\hat{\boldsymbol{\epsilon}}_0 \leftarrow f_{\boldsymbol{\phi}}^{-1}(\mathbf{x})$. Using the change of variables formula we can then compute the density of $\mathbf{x}$ under the base distribution. We begin by discussing this repeated change of variables.

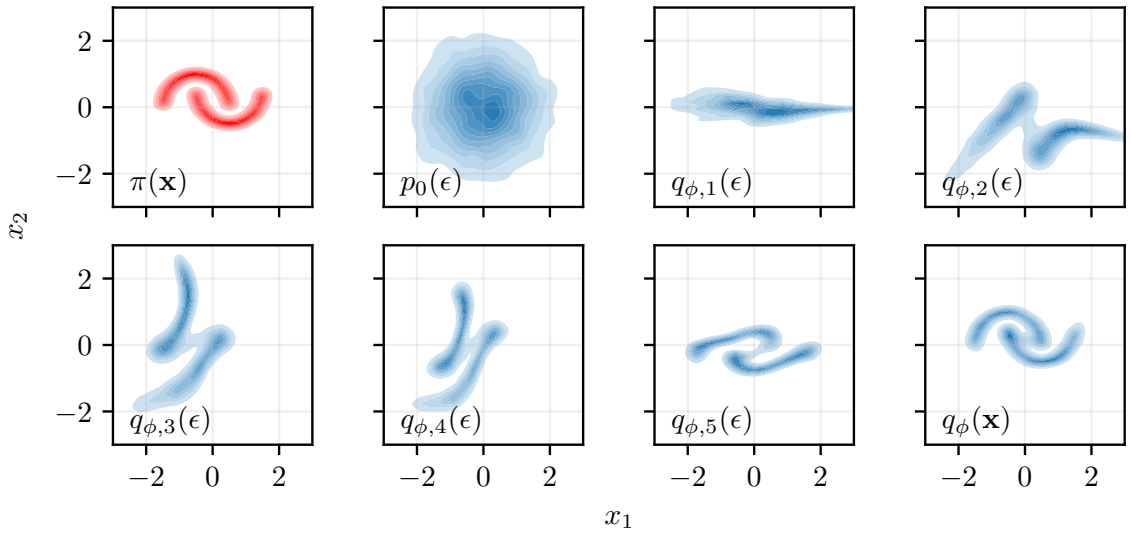


Figure 3.12: Example of predictions from a trained masked autoregressive flow [Papamakarios et al., 2017]. The observed data distributed according to $\pi(\mathbf{x}), \mathbf{x} \in \mathbb{R}^2$, shown top left as a density. The base distribution is a two-dimensional unit Gaussian, $p_0(\mathbf{x})$. This Gaussian is then transformed through a six-layer flow to approximate the target distribution, i.e. $q_{\phi,6}(\mathbf{x}) \approx \pi(\mathbf{x})$. Intermediate plots show the evolution of the latent state through the flow layers. We see that the flow has accurately approximated the target density and that we are still able to evaluate the density of arbitrary points under the flow.

3.5.2.1 Repeated Change of Variables

Flows are built on compositions of changes of variables. We introduced change of variables for deterministic functions in Section 3.1. The result for a deterministic, smooth, bijective mapping was:

$$\text{if } \boldsymbol{\epsilon}_1 = f_1(\boldsymbol{\epsilon}_0), \quad \text{then} \quad p_1(\boldsymbol{\epsilon}_1) = p_0(f_1^{-1}(\boldsymbol{\epsilon}_1)) \left| \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\epsilon}_1} f_1^{-1}(\boldsymbol{\epsilon}_1) \right|. \tag{3.121}$$

This expression means that if f_1 has a well-defined Jacobian, we can compute the density of the transformed variable by computing the density of the original variable, and multiplying by a contraction term. We can then apply a second transformation, $\boldsymbol{\epsilon}_2 = f_2(\boldsymbol{\epsilon}_1)$, and compute the density $p(\boldsymbol{\epsilon}_2)$, by applying the change of variables a second time:

$$p(\boldsymbol{\epsilon}_2) = p_0(f_1^{-1}(f_2^{-1}(\boldsymbol{\epsilon}_2))) \left| \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\epsilon}_2} f_1^{-1}(f_2^{-1}(\boldsymbol{\epsilon}_2)) \right|. \tag{3.122}$$

Noting that the Jacobian of a composite function is the product of the Jacobians:

$$J(f_2(f_1(a)) = J(f_1(a)) \times J(a), \tag{3.123}$$

and the determinant of a product of matrices the product of the determinants of the matrices:

$$|A \times B| = |A| \times |B|, \tag{3.124}$$

and using the fact that the determinant of the inverse of a function is the inverse of the determinant of the function, we can write (3.122) as:

$$p(\boldsymbol{\epsilon}_2) = p_0(f_1^{-1}(f_2^{-1}(\boldsymbol{\epsilon}_2))) \left| \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\epsilon}_1} f_2 \right|^{-1}_{f_2^{-1}(\boldsymbol{\epsilon}_2)} \left| \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\epsilon}_0} f_1 \right|^{-1}_{f_1^{-1}(f_2^{-1}(\boldsymbol{\epsilon}_2))} . \tag{3.125}$$

We can write this compactly as a product for N function applications, where we denote composition of functions $f_N \circ f_{N-1} \circ \cdots \circ f_1$ as $f_{N:1}$, with intermediate values $\boldsymbol{\epsilon}_n$, to evaluate the density of a variable, denoted $\mathbf{x}$, as:

$$p(\mathbf{x}) = p_0(f_{N:1}^{-1}(\mathbf{x})) \prod_{n=1}^{N} \left| \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\epsilon}_{n-1}} f_n \right|^{-1}_{\boldsymbol{\epsilon}_{n-1}} . \tag{3.126}$$

Taking logarithms converts the product to a summation:

$$\log p(\mathbf{x}) = \log p_0(f_{N:1}^{-1}(\mathbf{x})) - \sum_{n=1}^{N} \log \left| \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\epsilon}_{n-1}} f_n \right|_{\boldsymbol{\epsilon}_{n-1}} , \tag{3.127}$$

dramatically improving numerical stability, as naively multiplying probabilities can rapidly lead to numerical underflow, whereas summing log probabilities is less prone to underflow. This expression allows us to use the flow to estimate the density of an observed $\mathbf{x}$ value. To do this, we set $\boldsymbol{\epsilon}_N = \mathbf{x}$, and compute the sequence of intermediate values, $\boldsymbol{\epsilon}_{N-1:0}$, by successive application the inverse of each layer, accumulating the Jacobian term at each layer, and sum this with density of the final entry in this sequence under the base distribution (the first term in (3.127)).

By considering a flow as a series of change of variables, we can compute densities of observed values, and, generate new values by sampling from the flow. However, the success of flows hinges on the existence and efficient computation of the log-determinant of the Jacobian. Therefore, there are restrictions placed on the form of the warping functions f.

3.5.2.2 Structuring Layers

The structure of the warping functions, f_{ϕ_n}, is critical to the performance of flows. The Jacobian of these functions must be guaranteed to exist and must be cheaply computable. Early normalising flows work by Rezende & Mohamed [2015] derive two types of layers: planar flow and radial flow. These forms were chosen due to the ease of computing their determinants and that their composition yields reasonably expressive flows.

However, planar and radial flows are restrictive and require many layers to create a complex distribution. Therefore, instead of constraining the mathematical form of the *function*, we can instead impose constraints on the *parameters* of the function such that

the Jacobian remains tractable. By far the most common such constraint is to impose an *autoregressive* property [Larochelle & Murray, 2011]. Flows that use this property are described as *autoregressive flows* (AFs). Autoregressive implies that, for a particular ordering of random variables, each variable only depends on previous variables: $z_d \leftarrow f(z_{1:d-1})$. Therefore, each layer (of width D) is constructed by imposing that the first variable is independent, the second value is then dependent on the first value, and so on until all D variables have been generated:

$$\epsilon_{n,d} \leftarrow f_{n,d}(\boldsymbol{\epsilon}_{n-1,1:d-1}). \tag{3.128}$$

Assuming $f_{n,d}$ is a linear combination of the d variables, the whole function f_n can be written as a matrix multiplication where the weight matrix $\mathbf{W}$ is constrained to be upper-diagonal:

$$\boldsymbol{\epsilon}_n = \mathbf{W}_n \boldsymbol{\epsilon}_{n-1}, \tag{3.129}$$

and has Jacobian:

$$J(f_n) = \prod_{d=1}^{D} W_{n,d,d}. \tag{3.130}$$

As a result, the Jacobian of this autoregressive transformation can be computed in $O(D)$. An element-wise bijective non-linearity can then be introduced to generate more expressive flows. Autoregressive flows can be markedly more expressive than planar or radial flows, and hence can define a much more expressive flow with fewer layers and fewer parameters [Larochelle & Murray, 2011]. We note that the internal ordering of variables in the autoregressive function is arbitrary. Specific flow architectures, such as MADE [Germain et al., 2015], MAF [Papamakarios et al., 2017] and GLOW [Kingma & Dhariwal, 2018] structure f_n so that it adheres to the autoregressive property and is efficiently computable.

3.5.2.3 Training Flows

The flow layer is parameterised by $\boldsymbol{\phi}_n$. These parameters are learned by minimising the divergence of the distribution between the distribution defined by the flow, $q_{\boldsymbol{\phi}}(\mathbf{x})$, to the distribution of observed data, $\pi(\mathbf{x})$:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi} \in \boldsymbol{\phi}} \mathrm{KL}\left[\pi(\mathbf{x}) || q_{\boldsymbol{\phi}}(\mathbf{x})\right] \tag{3.131}$$

Algorithm 3.5 Training Flow Approximator, q_{ϕ}

procedure TRAINQ($\mathcal{D}$, K, S, q, $\boldsymbol{\phi}_0$, η)
  for $s = 1 : S$ **do**
    $\mathbf{x}^{(k)} \leftarrow \text{SampleMinibatch}(\mathcal{D})$ // Sample from dataset.
    $L \leftarrow \prod_{k=1}^{K} q_{\boldsymbol{\phi}^s}\left(\mathbf{x}^{(k)}\right)$ // Evaluate density.
    $\mathbf{G}^s \leftarrow \nabla_{\boldsymbol{\phi}^s} L$ // Do backprop.
    $\boldsymbol{\phi}^{s+1} \leftarrow \boldsymbol{\phi}^s + \eta\left(\mathbf{G}^s\right)$ // Apply update.
  end for
  return $\boldsymbol{\phi}^S$ // Return learned parameters.
end procedure

Expanding this and rearranging divergence:

$$\text{KL}\left[\pi(\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{x})\right] = \int_{\mathbf{x}\in\mathcal{X}} \pi(\mathbf{x}) \log\left(\frac{\pi(\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{x})}\right) \mathrm{d}\mathbf{x}, \tag{3.132}$$

$$= \int_{\mathbf{x}\in\mathcal{X}} \pi(\mathbf{x}) \log \pi(\mathbf{x}) \mathrm{d}\mathbf{x} - \int_{\mathbf{x}\in\mathcal{X}} \pi(\mathbf{x}) \log q_{\boldsymbol{\phi}}(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{3.133}$$

The first term is independent of $\boldsymbol{\phi}$, and so can be dropped from the minimisation:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}\in\boldsymbol{\phi}} - \int_{\mathbf{x}\in\mathcal{X}} \pi(\mathbf{x}) \log q_{\boldsymbol{\phi}}(\mathbf{x}) \mathrm{d}\mathbf{x} = \arg\max_{\boldsymbol{\phi}\in\boldsymbol{\phi}} \mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{x})\right], \tag{3.134}$$

such that we can easily write the gradient with respect to the parameters of the flow:

$$\nabla_{\boldsymbol{\phi}} \text{KL}\left[p(\mathbf{x})||q_{\boldsymbol{\phi}}(\mathbf{x})\right] = \mathbb{E}_{\mathbf{x}\sim\pi(\mathbf{x})}\left[\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{x})\right]. \tag{3.135}$$

The density of a particular data point under the flow can be computed using (3.127) by manually specifying the form of the Jacobian of each layer. The gradient can then be efficiently computed using autodifferentiation, and then use SGD to learn the flow parameters, shown in Algorithm 7.2. The training procedure only requires *samples* from the target distribution, $\mathbf{x}^{(k)} \sim \pi(\mathbf{x})$, (as opposed to evaluations of the target distribution) and hence can be trained directly from datasets.

3.5.2.4 Conditional Flows

Until now we have discussed unconditional distributions, simply sampling from and evaluating $\pi(\mathbf{x})$. However we wish to be able to generate and evaluate under conditional densities, $\pi(\mathbf{x}|\mathbf{y})$. There are two strategies for introducing conditioning into the flow. Firstly, one can simply concatenate the observed $\mathbf{y}$ variable to the input to each flow layer. However this can increase the dimensionality of each flow layer considerably to the detriment of performance. Alternatively, one can use hypernetworks [Ha et al., 2016] to introduce conditioning. Hypernetworks are simply neural networks that output the parameters of

a second neural network. Conditioning is introduced using the parameters output by the hypernetwork as the parameters of the flow. The flow is then learned by instead learning the parameters of the hypernetwork. Denoting the hypernetwork $g_{\phi}(\mathbf{y})$, with instead the parameters of the hypernetwork denoted ϕ, the conditional flow is denoted $p(\mathbf{x}|\mathbf{y}) = q_{g_{\psi}(\mathbf{y})}(\mathbf{x})$. The parameters of the hypernetwork do not feature in the Jacobian computation, and hence there are no restrictions placed on the structure of the hypernetwork or the type of the conditioning variables and can be learned using backpropagation as before.

Conditional flows define a more complex function than an unconditional flow, and hence the performance of a conditional flow cannot be expected to match the performance of an unconditional flow on each conditioning variable value [Oh & Valois, 2020]. However, only one flow needs to be trained for all $\mathbf{y}$ values, and hence is suitable for tasks with continuous conditioning variables.

3.5.2.5 Summary

In this section we introduced flow-based distribution approximators. These flows provide a flexible parametric function class that allows the density of an arbitrary data point to be evaluated, and, for new datapoints to be generated, either conditionally or unconditionally. We did not have to specify likelihoods, kernels or generative models, making flows a robust and easy-to-tune tool, that still affords powerful probabilistic interpretations and can be learned using efficient and GPU-amenable gradient descent. We will use flows in Chapter 7 to approximate distributions and efficiently generate random samples.

3.6 Discussion

In this chapter we introduced the statistical and machine learning techniques and algorithms that we will use throughout this thesis. As part of this, we briefly introduced some of the theory that underpins these approaches, discussed the relative merits and drawbacks of each approach, and gave a brief overview of how these approaches are different on a conceptual level. We heavily emphasised probabilistic and Bayesian methods. These methods allow us to succinctly define our belief over the probabilistic dependencies between variables including, crucially, unobserved variables. Bayesian inference then allows us to analyse these probabilistic models. This analysis typically includes estimating the distribution over unobserved latent variables for a given data point, estimating the global parameter values

of the system, and quantitatively comparing the probability of different hypotheses or data points under the specified model.

In this thesis we are particularly focused on estimating neurophysiological states and parameters from data. The principal (and somewhat reductionist) intention of the technical content presented herein is developing and combining neurophysiological models with machine learning methods to frame and perform neuroscientific analysis as inference and estimation. In Section 2.6 we introduced several parameterised models describing the neural potential, ion, locomotion and fluorescence dynamics of *C. elegans*. Broadly speaking, these models define our *approximation* of the underlying neural processes and observed data as a probabilistic generative model, $p(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$, where the neural activity is denoted $\mathbf{x}$, the observed data is $\mathbf{y}$, and the parameters of the model are denoted $\boldsymbol{\theta}$. Using this model, we can, for instance, estimate the posterior over neural states given a fixed model and the observed data, $p(\mathbf{x}|\mathbf{y}, \boldsymbol{\theta})$, or quantify how well a particular model describes the data, $p(\mathbf{y}|\boldsymbol{\theta})$.

Estimating these distributions and quantities analytically is practically infeasible, as these models are specified as highly complex systems of equations and programmatic simulators, and hence we appeal to Monte Carlo methods to approximate these distributions and quantities. Therefore, methods such as SMC and PMMH will play pivotal roles in the work we present. In cases where even Monte Carlo integration is intractable, or a pointwise estimate is sufficient, we will utilise optimisation-based techniques such as MLE and MAP. We will also use neural networks to approximate functions that are too difficult to express or where rigorous statistical treatment is not required, and flows where precise inference is difficult or intractable and approximate distributions are sufficient.

More broadly, however, the synthesis of mechanistic neuroscientific models and machine learning methodologies promises to facilitate more sophisticated, interpretable and general neuroscientific analysis, far beyond what simply can be measured. Machine learning promises to automate such analysis, whether this entails estimating unobserved variables from data, estimating static parameter values, deducing which model best represents the observed data, or even proposing new models directly from data. Crucially, we can clearly state our assumptions and the model in which we operate, and by considering the underlying probability manipulations, we can be statistically rigorous in our analysis and inferences.

4

The Virtual Patch Clamp

We now introduce the first technical component of this thesis. We motived in Chapter 1 how a central goal of neuroscience, connectomics, and even artificial intelligence, is to understand how intelligent behaviours emerge from the interactions between the atomic units of the brain [Doty, 1975; Kristan & Katz, 2006; Seung, 2011]. However, techniques for directly measuring neural interactions at cellular fidelity cannot be applied at whole-connectome scale, are highly invasive, and limit the range of behaviours. We therefore argued that an alternative approach is to study highly accurate whole-connectome computer *simulations* of neural interactions [Sarma et al., 2018; Einevoll et al., 2019]. We can use the simulator to conduct experiments *in silico*, without the logistical constraints of *in vivo* experiments. We can, for instance, perturb the system and gather complete and noise-free observations of the underlying neural state, without influencing the specimen. It is even possible to conduct experiments that are logistically impossible or ethically questionable *in vivo*. Moreover, by also leveraging the probabilistic model defined by the simulator to infer the distribution over unobserved latent states from data observed *in vivo*. This allows more informative neural data, such as membrane potentials, to be recovered from more easily observed data.

However, this approach presupposes the existence of an accurate neural simulator. Highly accurate, parameterised, mathematical models of individual neurons, and the interactions between neurons, have been developed [Gewaltig & Diesmann, 2007; Hines & Carnevale, 2006; Wicks et al., 1996]. However, the absolute accuracy of these simulators for a given specimen is uncertain. To make highly accurate cellular-level inferences, the simulated network structure must correspond exactly to the neural structure of a known specimen. This confines our analysis to organisms where the connectome structure is known. The only organism for which the entire connectome has been mapped is the nematode roundworm *Caenorhabditis elegans* (*C. elegans*). Crucially, this connectome is constant across wild-type specimens [Varshney et al., 2011; White et al., 1986], and hence we can define whole-

connectome simulators with anatomically correct structure [Gleeson et al., 2018; Hasani et al., 2017; Marblestone, 2016].

Furthermore, in addition to the network structure, the parameter values used by the simulator must also be correct. The parameter values used in these simulations are often derived from painstaking manual experimentation, or, are simply assumed to be equal to values from other specimens or organisms. However, it is a central tenant of neuroscience that the differences in behaviour between specimens can be attributed to structural changes in the brain, or, in the case of a static connectome, differences in the relative connection strengths and electrophysiological properties of the connections. This presents both a problem and an opportunity. To obtain accurate and faithful simulations of a particular specimen, these parameter values must be estimated. Some parameter values may be measurable, but through invasive or destructive methods. Therefore, an opportunity exists to learn these parameters from data gathered non-invasively and for a particular specimen. This also suggests a third application of this general approach. Given a parameterised neural model, we can estimate the physiologically meaningful parameters from readily obtainable and non-invasively gathered data, regularised by the structure and function of the whole connectome.

However, anatomically grounded *C. elegans* connectome models have never been conditioned on such data for brain-wide latent state imputation and parameter estimation. We present a method for performing joint posterior latent-state and parameter inference at whole-connectome scale, conditioning on the type of data that non-invasive calcium fluorescence measurement techniques are currently capable of capturing [Dana et al., 2019; Kato et al., 2015; Nguyen et al., 2016]. As part of this, we develop a *C. elegans* simulator, capable of simulating the neural dynamics of the whole connectome, the locomotion of the worm and the calcium fluorescence dynamics, by combining existing models [Wicks et al., 1996; Boyle et al., 2012; Rahmati et al., 2016]. We introduced these models in detail in Section 2.6. The guiding principle for this simulator is computational tractability and modularity, such that simulation and inference is at least computationally tractable.

We then develop and apply techniques to estimate the unobserved time-varying physiological quantities conditioned on observed data, using a fixed model. We pose this as performing posterior inference in a hidden Markov model (HMM, Section 3.1.3.2), where the transition kernel and emission function of the HMM are defined by the simulator. To

perform this inference we use particle filtering, the numerical approximation of sequential Monte Carlo (SMC, Section 3.3.3). We specifically consider the case where only a small fraction of the calcium fluorescence traces are observed.

We then explore performing joint state-space inference and parameter learning. This extends inference to consider the model itself by learning the parameter values of the model from data, and corresponds to instead performing full posterior inference in a parametric HMM (Section 3.1.3.3). This allows physiologically meaningful parameter values to be estimated from readily observable correlates, as opposed to requiring direct and potentially invasive measurement. This also allows more accurate inferences over the unobserved latent states to be made by refining the model given new data, and allows different dynamics models to be quantitatively compared. This is the first attempt we know of to "complete the circle," where an anatomically grounded whole-connectome simulator is used to impute a time-varying "brain" state and parameter values at single-cell fidelity from covariates that are measurable in practice.

However, the flexibility and control afforded by using simulation and inference in-place of a biological specimen and *in vivo* manipulation means that the reach of this principle and the opportunities that it presents extends beyond simply estimating latent states and parameters given observed data. Instead of "passively" estimating states, we are able to actively and arbitrarily manipulate the simulator and state to investigate particular facets of the model during inference or subsequent generation, such as controlling individual potential or current values. We therefore describe this general approach as a "virtual patch clamp" (VPC). This encompasses the use of a whole-connectome model to condition connectome-scale simulations on observed data to estimate unobserved values, to estimate physiological parameter values, performing unconditional generation, and performing posterior predictive inference or conditional generation for specific specimens, physiological conditions, or behavioural outcomes. In this thesis we primarily tackle the foundations of this general approach: model specification, conditional generation and parameter estimation.

Once the parameters have been adapted for a particular specimen, *in silico* patch clamp experiments can be performed by simply pinning the value of a potential or current in a particular neuron to a particular value, and inspecting the resulting simulations. To perform this *in vivo* would require sophisticated apparatus and skilled technicians. However, *in*

silico, this is as simple as pinning a particular neural potential to a particular value. Beyond this, structural alterations, such as ablating connections, or exposing the specimen to novel stimuli can be done simply by reconfiguring the connectome or environment. These *in silico* experiments enable rapid, wide-reaching, and fully observable screening and investigation of experimental hypothesis. Most significantly, experiments are inexpensive and easy to reproduce. Traditionally, reproduction of results would require a full wet laboratory, skilled technicians, often proprietary analysis software, and unique and specially prepared specimens. However, reproduction of VPC experiments requires just a computer, the correct version of the source code, the configuration file and the random number generator seed.

These objectives reflect our long-term ambitions for this approach. A simulator with high fidelity and low run-time cost would allow rapid *in silico* experimentation to be performed by neuroscientists on standard desktop hardware. These experiments could be run on the order of seconds, and allow rapid *in silico* screening of hypothesis or model investigation. Inference is then performed in a fixed model to inspect and augment data that has been gathered *in vivo*. This inference may take minutes, but would allow more relevant neural data to be obtained, allowing existing hypotheses to be investigated and the generation of new hypotheses. Finally, the effect of neural modifications (such as ablating individual neurons), drug treatments, or, new variants, can be quantified by estimating the distribution over parameter for the new specimens. These parameter estimation experiments may take on the order of hours or days on substantial computational hardware, but would allow the model to be adapted to new and previously unmodelled behaviour, and to quantify the difference between two specimens.

We begin in Section 4.1 by describing the simulator we assemble. Much of the background material for this was covered in Section 2.6, and hence much this section focuses on the rationale behind the inclusion of each component of the simulator and the low-level mechanics of how the components are integrated. In Section 4.2 we then discuss using this simulator as the transition and emission model in a particle-based SMC inference pipeline, first introduced in Section 3.3.3. We demonstrate recovery of the distribution over unobserved latent membrane potentials from synthetic partial calcium fluorescence data. In Section 4.3 we then discuss simultaneously performing parameter inference, using a range of the algorithms introduced in Chapter 3. In Section 4.4 we conclude by discussing the opportunities for extending this work. We tackle a number of these extensions in Chapters

5, 6 and 7. Each of these extensions in turn presents more insight into the work presented in this chapter. We then reflect more generally on the opportunities provided by this work, and the extensions identified, in Chapter 8. Much of the work presented in this chapter was originally presented in Warrington et al. [2019].

4.1 Simulating *C. elegans*

At the core of Bayesian inference is the model of the system. As such, the foundation of this work is a probabilistic model, or computational simulator, of *C. elegans*. This model defines not only the latent states we are concerned with modelling (such as membrane potentials and muscular activations), but it also defines the time-evolution of these states, how these unobserved states are dependent on one-another, how the observed variables depend on the unobserved variables, and the physiologically relevant parameters that determine the precise nature of these relationships. Therefore, the first component we must introduce is the model itself. The model we introduce here is the full *C. elegans* simulator. This simulator is intended to be as expressive and complete as possible, while remaining computationally tractable. Part of the motivation throughout this chapter is to be ambitious, and to drive towards our ultimate objective, outlined above. In subsequent chapters we will use components of this full model in isolation to study particular properties or extensions.

On a high level, the *C. elegans* simulator we construct is comprised of three separate models: a whole-connectome neural potential dynamics model, a locomotion and neuromuscular dynamics model, and model of calcium fluorescence. We introduced these models at length in Section 2.6, and hence we refer the reader back for information on the low-level details. Complete understanding of every facet of these models is not strictly necessary for the material presented here; simply understanding the high-level operation of each component is sufficient. The model defined by this simulator is representable as a parametric HMM (see Section 3.1.3). The graphical model defined by the simulator, with the latent structure and dependencies expanded, is shown in Figure 4.1.

The neural potential dynamics model defines the membrane potential time-evolution for all 302 *C. elegans* neurons, and is derived from the neural model first proposed by Wicks et al. [1996]. This model represents neurons as single compartments using a set of coupled ordinary differential equations (ODEs), and simulates both the inhibitory and excitatory interactions between neurons through gap junctions and chemical synapses. The

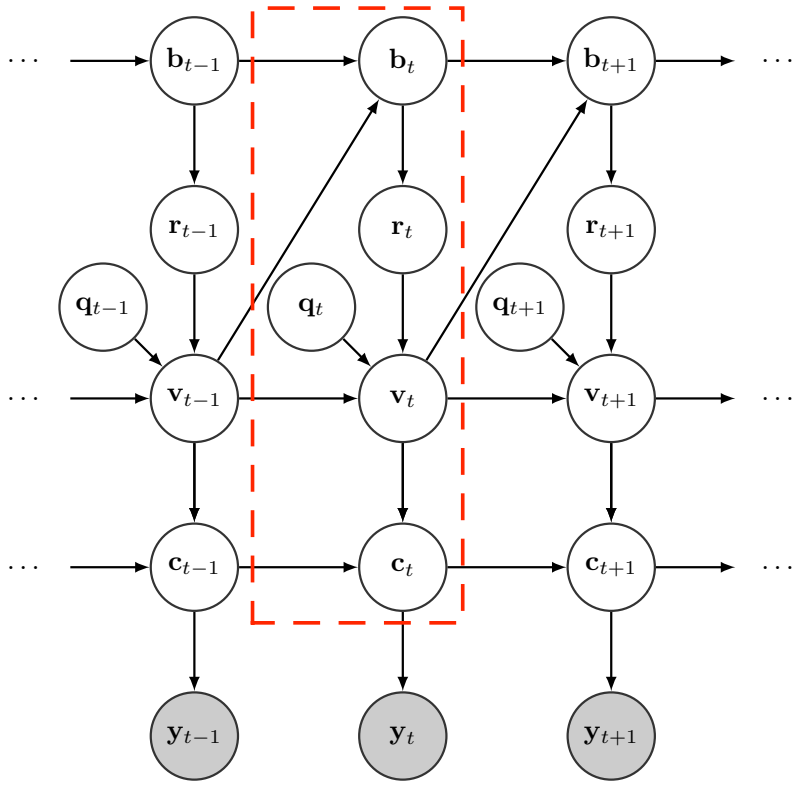


Figure 4.1: Graphical model describing the *C. elegans* simulator we assemble. The red dashed box indicates the simulator state, denoted $\mathbf{x}_t$, which corresponds to the hidden state in a parametric hidden Markov model. We do not show the global parameter values to reduce clutter, but note that every node in this graph is also conditioned on $\boldsymbol{\theta}$. The neural voltage is denoted by $\mathbf{v}_t \in \mathbb{R}^{302}$, intracellular calcium concentration by $\mathbf{c}_t \in \mathbb{R}_+^{302}$, body pose by $\mathbf{b}_t \in \mathbb{R}^{2\times3\times49} \times \mathbb{R}^{2\times48}$, and proprioceptive feedback by $\mathbf{r}_t \in \mathbb{R}^{12\times2}$. The external input current is denoted $\mathbf{q}_t \in \mathbb{R}^{302}$, however, we do not use this in this chapter. The observed calcium fluorescence trace as $\mathbf{y}_t \in \mathbb{R}_+^M$ where M is the number of observed neurons. As we do not explicitly consider $\mathbf{q}_t$, the time-varying latent space of the simulator is representable a 1018-dimensional vector.

connectivities and putative connection strengths are drawn directly from the *C. elegans* connectome [White et al., 1986; WormAtlas].

We have identified *in vivo* calcium fluorescence imaging as the foremost source of data on which to condition simulations. We therefore include a model of the potential-dependent intracellular calcium ion dynamics and observed fluorescence [Kato et al., 2015; Rahmati et al., 2016]. We then leverage this model to define the emission distribution under which we compute the likelihood of simulated latent states given observed fluorescence data. We subsume simulation of intracellular calcium into the potential dynamics model, collectively referred to as the *neural model*.

To the neural simulator we add a model of *C. elegans* locomotion. The model we use, WormSim, was first presented by Boyle et al. [2012] to study the role of proprioceptive

feedback in locomotion. Evidence suggests that this proprioception is an important element in capturing the neural dynamics of the worm [Boyle et al., 2012; Cohen & Sanders, 2014; Kunert et al., 2014, 2017; Wen et al., 2012]. The original simulator is driven by a simplified and idealised neural circuit, and so part of the technical effort is in integrating this model with the anatomically correct connectome and neural model. Inclusion of a body component is further motivated by the possibility of also conditioning on body pose and locomotion data, for which there exist large repositories [Yemini et al., 2013]. We explore this more in Chapter 5.

For the rest of this chapter we denote the number of neurons $N = 302$, with $n \in 1 : N$ indexing an individual neuron. The latent state of the simulator is denoted $\mathbf{x}_t \in \mathcal{X}$, and the time discretisation used in simulation is $\delta t = 0.01$ (discussed later). We condition simulations on observed data, denoted $\mathbf{y} \in \mathcal{Y} = \mathbb{R}_{\geq 0}^{T \times M}$, where the number of positively identified observed calcium traces is denoted $M \leq N$ (for example $M = 49$ in Kato et al. [2015]) and the length of the trace is denoted T, where $t \in 1 : T$ indexes discrete time steps. Observations are not necessarily available at every time step, caused by the interval between acquisitions being different to the simulation timestep used. We do not make accommodations for this in the notation, but note that is trivially accounted for it in the implementation. We now discuss individual components in more detail.

4.1.1 Neural Potential Simulation

The core component of our *C. elegans* model is a model of neural potential dynamics. The potential dynamics of individual neurons are well characterised by an ODE [Abbott & Kepler, 1990; Abbott, 1999; Hertz, 2018; Hodgkin & Huxley, 1952; Saarinen et al., 2006]. These equations can be numerically integrated over time to simulate the time-evolution of neural potential [Hines & Carnevale, 2006; Gewaltig & Diesmann, 2007]. Models of the conductive properties of electrical and chemical synapses can be integrated to extend the simulation to include the interactions between neurons [Hines & Carnevale, 2006; Wicks et al., 1996]. By simulating individual neurons and their connections, vast networks of neurons can be simulated as a set of coupled ODEs. Many different models have been developed across a range of fidelities and computational complexities, ranging from simple, single-compartment models [Hodgkin & Huxley, 1952; Wicks et al., 1996] to highly configurable, multi-compartment models [Hines & Carnevale, 2006; Gewaltig & Diesmann, 2007].

The potential model is a critical component as this model describes the time-evolution of the whole connectome. Studying *C. elegans* allows us to correctly structure this model by using the connectome published by White et al. [1986]. However the connectome only conveys the structure of the model and does not convey the physiologically grounded and interpretable parameter values, such as the relative strengths of synaptic connections or the electrophysiological properties of neural processes. Even though the connectome structure is constant across specimens, and the number of connections between neurons provides a coarse estimate of the overall connection strength, the values of these parameters vary from specimen to specimen, ultimately accounting for the differences in observed behaviours between specimens. Recovering such parameter values from observed data is part of the motivation of this work.

We compared two separate neural models, *c302* [Gleeson et al., 2018] and the *Wicks model* [Wicks et al., 1996]. c302, presented by Gleeson et al. [2018], acts as a wrapper to the NEURON simulation environment [Hines & Carnevale, 2006], configuring a NEURON simulation with the structure of the *C. elegans* connectome. NEURON can be configured to use a highly accurate electrophysioloigcal model, simulating individual neurons as multi-compartment differential equations, multiple ion mechanisms, and sophisticated models of synaptic conductance.

We also investigated the simpler model, presented by Wicks et al. [1996]. We discussed this model at length in Section 2.6. This model represents each of the 302 neurons as a single compartment, effectively assuming there are no spatial variations in the current and potential across the cell. The potential-dependent currents due to leakage, synaptic connections and external stimuli for each cell are defined by a parameterised differential equation. The complexity is further reduced by combining multiple connections of the same type between a pair of neurons into a single connection, where the strength of the combined connection is the sum of the individual connections. Current through synapses is not simulated using individual ion mechanisms, further reducing the computational complexity.

We ultimately chose the Wicks model over c302 for several reasons. Firstly, c302 is as much as two orders of magnitude more computationally expensive per second of simulation than the Wicks model. Although the modelling assumptions in the Wicks model fundamentally limit the absolute fidelity compared to c302, the computational burden of

performning inference in c302 would severely limit the number of particles that can be used, ultimately to the detriment of the final inference result, resulting in poorer inferences overall. Secondly, the Wicks model has fewer parameters than c302 and has a lower dimensional state representation, making the connectome-scale latent variables easier to interpret, and to form and test hypothesis on.

We introduced the Wicks neural model at length in Section 2.6, and hence we do not repeat the low-level detail here. We define the potential of all 302 neurons as $\mathbf{v}_t \in \mathbb{R}^N$. The model updates each potential as:

$$\mathbf{v}_t = f_{\text{neural}}(\mathbf{v}_{t-1}, \mathbf{r}_t, \boldsymbol{\theta}_{\text{neural}}), \tag{4.1}$$

where $\boldsymbol{\theta}_{\text{neural}}$ are the parameters of the neural model simulator, and the form of f_{neural} is given in (2.8). In the following section we introduce the locomotion model which provides proprioceptive feedback, $\mathbf{r}_t$, as direct stimulation (through I_{ext}) to the neural model. We provide more details about this interaction later. The initial estimates for the parameters $\boldsymbol{\theta}_{\text{neural}}$ are drawn from the original work [Wicks et al., 1996].

4.1.1.1 Direct Stimulation & Central Pattern Generation

The original model includes the ability to directly stimulate the network, denoted originally as I_{ext}. We use this facility to accommodate proprioceptive feedback from the physical model as additional current injected into specific neurons. These external stimuli also lays provision for sensory inputs [Bretscher et al., 2011; Izquierdo & Beer, 2013]. However, models of these processes are less developed currently. Once these mechanisms are better understood and quantified [Cohen & Denham, 2018; Izquierdo & Beer, 2013; Metaxakis et al., 2018], they can simply be added as additional random variables into the simulation. These additional models interact with the neural model through $\mathbf{q}_t$ in Figure 4.1. We therefore define I_{ext} as the sum of currents from proprioceptive feedback, $\mathbf{r}_t$, and direct stimulation, $\mathbf{q}_t$.

There is discussion in the *C. elegans* community as to the existence of a *central pattern generator* (CPG) in the *C. elegans* connectome [Boyle et al., 2012; Cohen & Sanders, 2014; Fouad et al., 2018; Gao et al., 2017; Olivares et al., 2018]. The CPG is often motivated as facilitating locomotion by providing a clock pulse for coordinating and activating the rest of the connectome. To produce sustained neural activity, *C. elegans* simulators are often driven by arbitrary or physiologically unrealistic periodic stimulus trains [Gleeson et al.,

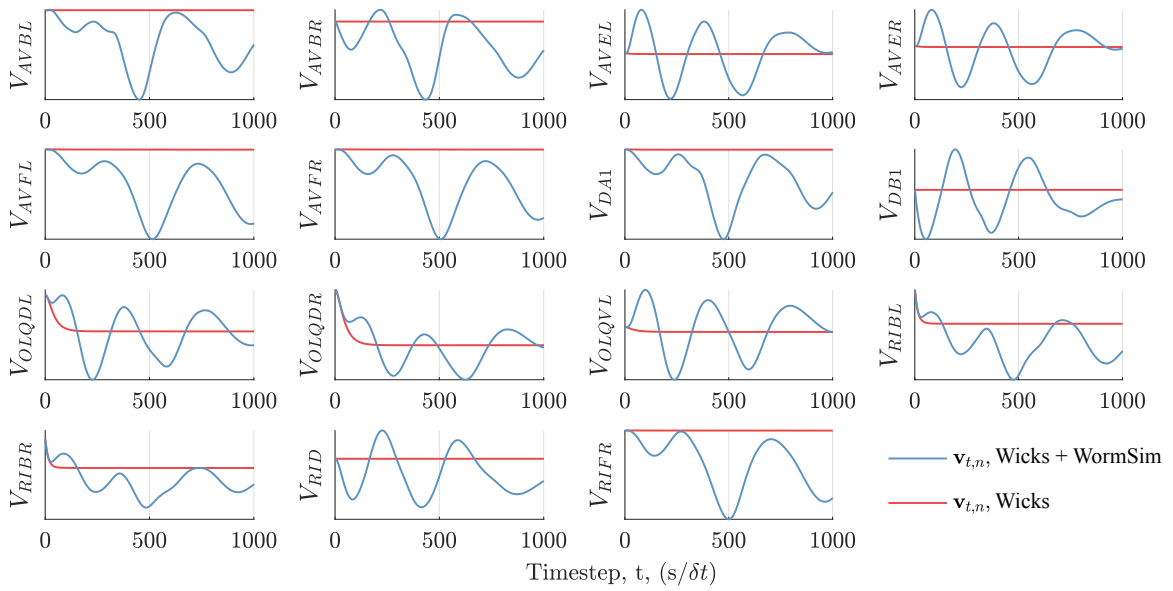


Figure 4.2: Experiment showing the success of stimulating the whole connectome with motor feedback provided by WormSim, compared to the Wicks model alone. Neither simulation uses a central pattern generator or artificial stimuli.

2018; Kunert et al., 2014, 2017; Marblestone, 2016], where this pulse is injected into the neural circuit through the external stimulation channel, $\mathbf{q}_t$.

However, Boyle et al. [2012] demonstrates in the WormSim simulator that the oscillating signals required to drive locomotion in *C. elegans* can be generated by a proprioceptive feedback circuit. The effect of this proprioceptive feedback mechanism is seen in Figure 4.2. We compare the original neural model without proprioception or direct stimulation, to the neural model we implement with proprioception provided by WormSim. We find that the Wicks model does not produce oscillatory behaviour without central pattern generation, while our simulation provides sustained simulation for approximately 10 seconds. Note that, even with proprioception, the oscillations eventually decay due to leakage currents in each neuron and muscle cell. We believe inclusion of sensory stimuli and refined parameter values can provide the additional neural stimulation to sustain activity, and therefore we do not consider non-physiological artificial stimulation here and set $\mathbf{q}_t = 0$.

To get sufficient excitement from the stretch-receptive feedback, we found that the membrane potentials of the most-excited neurons (predominantly motor neurons) are pushed into unphysiological ranges. We believe that this is a deficiency in both the original simulators (Wicks and WormSim) and the interface between the simulators through the use of poor parameter values. However, this shortcoming further motivates the use of

parameter estimation techniques, as manually specifying a self-sustaining model is non-trivial. Although undesirable, we do not believe this flaw fundamentally limits the functionality of our simulator, and rather provides an opportunity for further refinement and demonstration of the proposed methodology on *real data*.

4.1.2 Body Simulation

The next major component we incorporate is a simulator for the locomotion and neuromuscular interactions of the worm. *C. elegans* has 113 motor neurons [Altun & Hall, 2011] – over one-third of the connectome – and hence inclusion of muscular activation and proprioceptive feedback is pivotal for faithful simulation. We also suggest that the pose of the worm body is a valuable data source for conditioning simulations, and hence we must include a model of the body pose. We explore this theme more in Chapter 5.

We considered two body models. We first considered *Sibernetic*, presented by Palyanov et al. [2016]. Sibernetic is a highly accurate particle physics simulation, that is driven using the highly accurate neural model c302. Sibernetic can simulate the worm for a range of physical conditions in three dimensions. However, as with c302, we believe Sibernetic is too computationally intensive for inference to be feasible, taking minutes of computational effort per second of simulation.

Therefore, we opt to use the drastically simpler *WormSim*, presented by Boyle et al. [2012], which complements the experimental findings of Wen et al. [2012]. WormSim was developed initially to study the effect of proprioception on the generation of realistic *C. elegans* gaits [Cohen & Sanders, 2014; Lebois et al., 2012; Vidal-Gadea et al., 2011]. WormSim models the body shape and locomotion in two dimensions as a series of rigid rods, contractile units and damped springs, driven by impulses generated by a simplified neural network. We introduced WormSim in depth in Section 2.6, and so we refer the reader back and will not repeat many of the low-level details here.

WormSim represents the physical position of the worm using 49 rigid rods. These rods bound 48 repeating physical units, each equipped with dorsal and ventral damped contractile unit representing muscles, and internal contralateral springs representing the elasticity of the worm body. The position (x, y, and angle, θ) and the first derivative of the position are stored for each rod, and are both of dimension and type $\mathbb{R}^{49\times3}$. The muscular voltage is of dimension and type $\mathbb{R}^{48\times2}$, representing the instantaneous level of contraction in each

of the dorsal and ventral units. We refer to the position, velocity, and muscular voltage as the *physical components* of the worm, and are collectively defined as $\mathbf{b}_t \in \mathbb{R}^{390}$. We use the physical model without modification.

The original WormSim model uses a simplified neural model to drive the locomotion. The worm is divided into 12 repeating neural units, each driving four consecutive physical units. Each neural unit contains ventral and dorsal `B` and `D` class neurons (totalling four neurons). To integrate the anatomically correct neural model with the physical model (replacing the approximate neural model), we define a mapping from the neural excitation to the muscular units, and mapping from the proprioceptive feedback into current injected into the relevant neurons. We discuss this mapping now.

To drive the muscles using the anatomically correct connectome we use the biological `DB` and `DD` neurons for dorsal muscle innervation, and `VB` and `VD` for ventral innervation, guided by the findings of Wen et al. [2012] and the original implementation of WormSim. Specifically, the neurons used are `DB{1-7}`, `DD{1-6}`, `VB{1-11}` and `VD{1-13}`. To convert these irregular biological neurons to the 12 regular, repeating neural units used by WormSim, we linearly interpolate the neural signal using the location of the biological neurons (as defined in WormAtlas) and the position of the target repeating unit in WormSim. For instance, if an idealised `VB6` neuron in WormSim is at the midpoint between biological `VB6` and `VB7` neurons, we average the activation in the biological neurons and use this as the activation of the WormSim neuron. This activation is then input directly into the originally specified physical model. We introduce a scaling factor, denoted $w_{\mathrm{m}} \in \mathbb{R}_{\geq 0}$, which scales the biological neural potential into the neural potential used by WormSim. We found the simulations were very sensitive to this parameter, both in terms of the stability of WormSim and the quality of the simulation.

To compute proprioceptive feedback, WormSim uses a weighted average (favouring nearby units) of the dorsal and ventral extension of the physical units. This extension is then converted into a current and is injected back into the `B` class neurons, as proposed by Wen et al. [2012]. The raw proprioceptive feedback is denoted $\mathbf{r}_t \in \mathbb{R}^{12\times 2}$, indicating the sign and magnitude of the current injected into (or drawn from) the neurons. To convert this proprioceptive feedback from the 12 repeating units used by WormSim into biological neurons we again linearly interpolate the proprioceptive signal based on the location of

the neuron (as specified by WormAtlas) and the relative location of each neural unit in the WormSim model. The neurons we allow proprioceptive feedback to flow into are `DB{1-7}` on the dorsal side and `VB{1-11}` on the ventral side. We again introduce a conversion parameter, denoted $w_{\mathrm{s}} \in \mathbb{R}_{\geq 0}$, to scale the stretch receptive current calculated by WormSim into to the current injected into the biological neurons. We find that tuning this parameter is important to create sustained locomotion.

If the value of either w_{s} or w_{m} are too low, the worm does not exhibit sustained locomotion, and the body and neural state quickly returns to its quiescent point. We also found that the body simulator, unlike the neural simulator, is prone to "crashing." Specifically, a crash here refers to the ODE integrator failing to solve the equations of state to the required precision in the allocated computational budget; returning a flag indicating simulator failure instead of the iterated state. Failures can occur if either w_{s} or w_{m} are too high, leading to the muscular potentials growing exponentially and encountering numerical overflow. We illustrate the dependency of failures on parameter values in Figure 4.7. This may also be a result of the *stiffness* of the differential equation in certain regions of parameter space, where as a result of large gradients, very fine time discretisations must be used to accurately solve the system. Although stiff ODE solvers could be deployed, or, more computational budget allocated per iteration, these can increase these computational cost considerably. Crashes also occur if the body state is excessively perturbed with noise, and enters into a physiologically invalid configuration, causing the simulator to exit. Inspection of failed integrations indicated that the majority of failures were as a result invalid physical configurations, and hence we actually leverage this failure to trigger discarding such states by assigning these particles zero probability. The observation that a simulator can simply fail for seemingly trivial alterations to the state or model was the original motivation behind the work presented in Chapter 7.

This collectively defines the evolution of the body pose of the worm and the proprioceptive feedback as a function of the preceding neural voltage and body pose:

$$\mathbf{b}_t \leftarrow f_{\mathrm{body}}(\mathbf{b}_{t-1}, \mathbf{v}_{t-1}, \boldsymbol{\theta}_{\mathrm{body}}), \tag{4.2}$$

$$\mathbf{r}_t \leftarrow f_{\mathrm{prop}}(\mathbf{b}_t, \boldsymbol{\theta}_{\mathrm{body}}), \tag{4.3}$$

where we include all parameters pertaining to locomotion and proprioception in $\boldsymbol{\theta}_{\mathrm{body}}$. By converting the biological potentials into the representation used by WormSim we can drive

the originally specified body model using the anatomically correct neural model, and draw proprioceptive feedback from the WormSim model into the anatomically correct neural model without further modification.

4.1.3 Calcium & Observation Model

The final element we include is a model relating the neural potential to observed calcium fluorescence. We integrate the model for simulating the potential-dependent intracellular calcium ion concentration first presented by Rahmati et al. [2016], and was discussed in Section 2.6. This model simulates the change in intracellular calcium concentration as a function of the neural potential. The observed fluorescence is then a Gaussian perturbed non-linear function of the intracellular calcium concentration.

Combining this model with the neural model was relatively straightforward as the calcium model assumes single-compartment neurons. We therefore simply replace the spiking neural model used in the original work with our non-spiking neural model. The calcium model defines the updated calcium concentration in each of the 302 neurons, denoted $\mathbf{c}_t \in \mathbb{R}^N_{\geq 0}$:

$$\mathbf{c}_t = f_{\text{calcium}}(\mathbf{c}_{t-1}, \mathbf{v}_t, \boldsymbol{\theta}_{\text{calcium}}), \tag{4.4}$$

where $\boldsymbol{\theta}_{\text{calcium}}$ are the parameters of the calcium simulator. The functional form of f_{calcium} is defined in (2.16). The parameter values were drawn from Rahmati et al. [2016].

We then denote the raw fluorescent intensity as $\mathbf{y}_t \in \mathbb{R}^M_+$, where $M \leq N$ denotes the number of observed neurons. Here, we formally define observed neurons as those neurons for which a fluorescence trace could be confidently attributed to a particular neuron by expert annotators. In the dataset released by Kato et al. [2015], the number of neurons identified varies between datasets. We selected one of these datasets as a prototypical dataset, where $M = 49$, and observe only the neurons identified in this dataset as part of our experiments. This corresponds to observing the florescence for a fixed and known subset of neurons. To use the remaining, unlabelled neurons introduces a challenging permutation problem. This problem is further investigated by Linderman et al. [2017]; Mena et al. [2018]. We investigated jointly solving the permutation inference challenge, along with latent state recovery and parameter learning, but instead chose to focus here on performing inference in the state-space model and performing parameter estimation, and defer inferences over the use of unlabelled fluorescence traces to future work. Furthermore, this permutation

problem promises to be eased or even eradicated through the use of multicolour fluorescent proteins [Inoue et al., 2019; Sands et al., 2018].

The fluorescent intensity is a stochastic quantity dependent on the intracellular calcium concentration. We define the distribution over observed fluorescence as:

$$\mathbf{y}_t \sim p(\mathbf{y}_t|\mathbf{c}_t, \boldsymbol{\theta}_{\text{obs}}) = \mathcal{N}\left(\mathbf{A}^{M\times N} \times \boldsymbol{\theta}_{\text{obs},F} \times \frac{\mathbf{c}_t}{\mathbf{c}_t + \boldsymbol{\theta}_{\text{obs},K}} + \boldsymbol{\theta}_{\text{obs},D}, \mathbb{I}^{M\times M} \times \boldsymbol{\sigma}^2\right), \quad (4.5)$$

with variances $\boldsymbol{\sigma}^2$, and where $\mathbf{c}_t$ is the vector of N calcium concentrations. The constants $\boldsymbol{\theta}_{\text{obs},F}$, $\boldsymbol{\theta}_{\text{obs},K}$, $\boldsymbol{\theta}_{\text{obs},D}$ and $\boldsymbol{\sigma}^2$ comprise $\boldsymbol{\theta}_{\text{obs}}$ and can be independently calibrated or estimated on-line from data [Kato et al., 2015; Rahmati et al., 2016]. When we generate synthetic data, we generate noise-free data by setting $\boldsymbol{\sigma}^2 = \mathbf{0}$. The matrix $\mathbf{A}^{M\times N} \in \{0,1\}^{M\times N}$ is a non-square permutation matrix that assigns n-indexed neurons to m-indexed observations. This matrix does not need to be square, but each row must have exactly one non-zero value and each column cannot have more than one non-zero value. For this work we assume that this matrix is specified *a priori* expert annotators.

This model allows us to relate the observed calcium fluorescence traces to the unobserved neural potentials. More specifically, this model defines the likelihood of the latent state given the observed data. This will be used later score different realisations of latent state as part of inference procedures presented later.

4.1.4 Model Summary

The full graphical model of the *C. elegans* simulator is shown in Figure 4.1. To summarise our model, the neuron states, $\mathbf{v}_t \in \mathbb{R}^{302}$ and $\mathbf{c}_t \in \mathbb{R}^{302}_{\geq 0}$, body state $\mathbf{b}_t \in \mathbb{R}^{390}$, proprioceptive feedback $\mathbf{r}_t \in \mathbb{R}^{24}$, and any sensory input $\mathbf{q}_t \in \mathcal{Q}$ (not explicitly used here), define the latent "brain" and "body" state of the worm, collectively denoted at time t as $\mathbf{x}_t \in \mathbb{R}^{1018}$. This latent state is indicated by the dashed box in Figure 4.1. The observed data, $\mathbf{y}_t \in \mathbb{R}^{M}_{\geq 0}$, are the M observed calcium fluorescence traces of length T. Our simulator defines a parametric hidden Markov model, albeit one with complex, high-dimensional non-linear latent state transition dynamics, $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$, and a non-linear observation model, $p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta})$. The emission model presented above provides us with a likelihood that we can evaluate and will make extensive use of when subsequently performing inference. All of the models presented are parameterised, with the total set of parameters denoted $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{neural}}, \boldsymbol{\theta}_{\text{calcium}}, \boldsymbol{\theta}_{\text{body}}, \boldsymbol{\theta}_{\text{obs}}\}$.

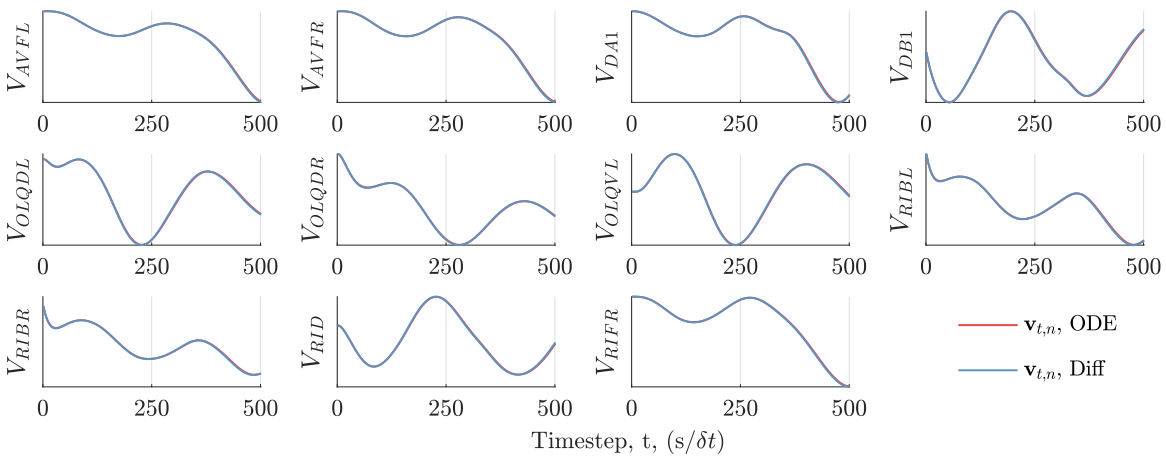


Figure 4.3: Experiment showing the accumulated error when using and ODE solver (`odeint` provided by SciPy) and using finite difference. We consistently observe a negligible difference in the trajectories and hence use finite difference, citing the considerable computational speedup.

4.1.5 Software Implementation

In the previous section we introduced the models we compose to construct our *C. elegans* simulator. We now introduce some computational considerations when structuring and implementing the models such that we can perform inference efficiently.

4.1.5.1 Core Simulator

We implement the Wicks model using vectorised NumPy, and later PyTorch, operations. We also use Euler integration in place of an ODE integrator in the neural model. We found the difference in simulated behaviours to be minimal (see Figure 4.3) but reduced the average time for iterating the simulator from 16.7 milliseconds using the ODE solver to just 0.7 milliseconds by using forward difference. The integration timestep was chosen to be 0.01 seconds, as this yielded numerically stable simulations with negligible difference to the solution obtained using an ODE solver, while also minimising the number of simulation steps required. While Euler integration does introduce integration errors, as we go on to discuss in the next section, we add noise to the state at each timestep to induce a distribution over simulations. The magnitude of the noise we add is far larger than the inaccuracy introduced by the discretisation. The speedup is significant, and hence we can run more particles for the same computational cost, which will likely lead to a more accurate inference overall. We also experimented using sparse representations, but found no computational gains. Adding the calcium and fluorescence models is computationally cheap as they have complexity linear in the number of neurons (as opposed to the squared complexity of the neural model).

WormSim is implemented in C++. We therefore modified WormSim to run in a separate *worker process* in a master-worker configuration. We implement a low-level controller in Python that coordinates the WormSim executable. We use the interprocess communication package ZeroMQ (ZMQ) to communicate between the WormSim instances and the Python controller. On startup, the process executing the Wicks model opens a second low-level controller process, and a WormSim executable is initiated. These processes are passed a unique port which to communicate with each other, along with any parameters or configuration information required for the simulation. The modified WormSim instance then waits for an entire brain-body state to be transmitted via ZMQ. When the neural model requires the body simulator to be called, the entire state is passed to the executable, and the entire iterated state is passed back to the neural model. Hence, each WormSim executable is a static object and can be called by any neural model by simply passing the appropriate state. Upon receipt, the executable performs a prescribed number of integration steps. The iterated state is then returned back to the controller, via ZMQ, and the worker returns to idle mode until a new data packet is received.

4.1.5.2 State Space Estimation

We further develop the implementation to allow multiple particles to be iterated in parallel. This is critical when performing state-space inference using particle filters, as we wish to run many particles in parallel. The factors critical to efficiently parallelising code are to minimise the amount of serial work, minimise the amount of interprocess communication, and reducing synchronisation time. Serial work is work that must be done sequentially on a single node while all other nodes are idle. For allocations of at least ten CPUs or nodes, only one CPU or node is working, effectively capping the efficiency at single-digit percentages. Minimising interprocess communication minimises the time wasted waiting for data to be packaged, transmitted and received (although this partially alleviated by high-bandwidth communication infrastructure). Finally, synchronisation wastage is the time spent idling by some workers while other workers are working, caused by unequal distribution of computation.

To parallelise particle filter sweeps we begin by implementing a second *node* controller that is responsible for controlling the worker processes. This node controller opens two equal-sized pools of *neural* and *body* worker processes. Each neural worker in the first pool runs a single instance of the neural model. Each body worker in the second pool initiates a

single WormSim executable (by initiating a low-level controller), and is paired with a neural worker through a unique and persistent ZMQ socket. The node controller is then responsible for coordinating the state-space inference, including scoring, resampling and distributing the iteration of multiple particles. The node controller initialises particles into shared memory. To iterate the particles the node controller simply transmits the addresses of the particles to be iterated by each neural worker to the worker. Each neural worker then iterates the particle, in-place in shared memory, alternating between calling the neural model and the corresponding WormSim executable. The controller then scores and resamples particles, simply by moving and replicating resampled particles within shared memory.

The tasks performed by the node controller are computationally cheap. This design also ensures the neural model and WormSim pair are never concurrently executing. This means it is easy to select the number of processes in each pool as the number of processors available on the node. Workers also iterate approximately the same number of particles (for reasonably sized pools) minimising synchronisation losses. We find that for reasonably sized particle pools (≥ 3 particles per CPU) we achieve a CPU utilisation of over 90%. By parallelising in this manner we can quickly estimate the distribution over latent states, $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}, \boldsymbol{\theta})$, and estimate the model evidence, $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$, by running particle filter sweeps on single compute nodes.

We investigated iterating particles across multiple compute nodes. To do this, we used the message passing interface OpenMPI [Graham et al., 2005] to distribute iteration of particles to worker nodes across multiple nodes, as opposed to simply distributing particles to worker threads internal to the node. However, we found that this added tremendous complexity to the code and actually resulted in a reduction in the throughput per CPU. We believe this is because of the overheads induced by interprocess communication and synchronisation wastage. Therefore, we therefore did not proceed with this implementation, and instead perform sweeps on a single node.

4.1.5.3 Parameter Estimation

To leverage multiple compute nodes we further develop the implementation to instead allow embarrassingly parallel execution of independent SMC sweeps. This is critical for parameter estimation. We achieve this in our software in two different ways.

The first parallelisation method we implement simply leverages Pythons inbuilt multiprocessing library to parallelise function evaluations. This is ideal for smaller jobs that only use a large single node, such as the autoregressive experiments presented later. Implementing this is easy, as the array of parameters to be evaluated is simply iterated over by the `map` operator. This distributes each evaluation to a worker pool local to that node. This is an efficient and low-overhead method, only requiring a single node, but limits the scalability as a result.

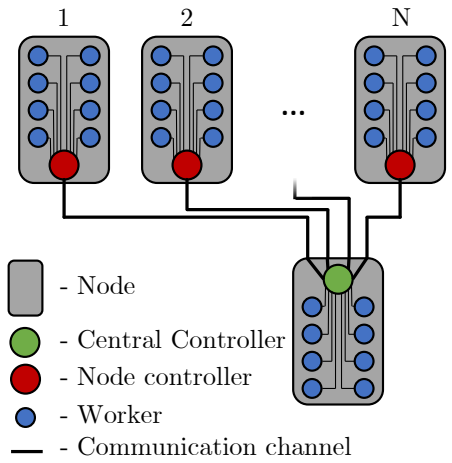


Figure 4.4: Simple schematic of hybrid parallelisation. The controlling node is shown in green, and worker nodes are shown in red. The central controller sits on its own node, with its own workers, and is connected to a single worker per node, referred to as a node controller. Each node controller then communicates with a pool of workers on its own node. Inter-node communication is minimised by parallelising individual SMC sweeps within each of the N nodes, and only receiving parameters from, and sending likelihoods to, the controller. Additional nodes can be added with little computational overhead.

We also implemented a method for efficient distribution of large jobs across multiple nodes, primarily used for evaluating likelihoods in parallel when studying *C. elegans*. We found parallelising individual sweeps across many nodes to be inefficient. We therefore use a *hybrid* parallelism model, shown schematically in Figure 4.4. In this configuration, whole nodes are treated as individual entities. Each node may be equipped with multiple cores and a shared memory unit, but these are only addressable from within that node. There is then a dedicated per-node controller that initiates and coordinates the workers on that node. Each worker then communicates exclusively with the node controller. The node controller handles all communication into and out of the node, communicating with the workers and the *central controller*. The central controller, running on a separate node, coordinates the worker nodes.

Multiple likelihoods can be evaluated in parallel on separate worker nodes, as evaluating the likelihood of $\mathcal{N}_r$ different parameter values separates into $\mathcal{N}_r$ individual calls evaluating a single likelihood. Each evaluation is then parallelised to use all available resources *within* each node, using the efficient parallel single-node state-space estimation described previously. We implement the node controllers and central controller as OpenMPI processes

The workflow proceeds as follows: One OpenMPI process is opened on each available node. A single OpenMPI process is allocated as the central controller, and the remaining processes are allocated as node controllers. Each node controller opens a further pool of workers internal to that node. At the start of execution, the data, $\mathbf{y}$, fixed parameters and any auxiliary configuration parameters (file paths etc) are transmitted by the central controller to each node controller. These settings are stored permanently. When a set of $\mathcal{N}_r$ likelihoods need to be evaluated, the central controller transmits a subset of the $\mathcal{N}_r$ parameter values to each of the node controllers. One node may receive multiple parameters if the number of nodes is less than $\mathcal{N}_r$. Individual likelihood evaluations are then queued on each worker node and are evaluated sequentially. Each likelihood is calculated *within* the node as outlined above. Once the sweep is complete, the node controller returns the likelihood, via OpenMPI, back to the central controller. Accordingly, no particles pass over OpenMPI keeping communication overheads low. The worker then idles until a new set of parameters is sent. Once the central controller has collected all required likelihoods, the appropriate and computationally inexpensive outer step is taken on the controllers node.

We have tested our code on two HPC clusters, Cori, administered by NERSC, and Cedar, administered by ComputeCanada. We distributed across 128 nodes and observed $> 90\%$ CPU utilisation. The expensive-to-compute likelihood is evaluated on each node, and the cheap-to-compute prior is computed on the controller node. Hence, PT swaps can be performed quickly and efficiently without needing to repeat evaluations. A minor drawback of this approach is that whole nodes may sit idle while waiting for other nodes to finish, however, we find this synchronisation inefficiency is minimal once good parameter values have been found.

4.1.6 Summary

Before we proceed, we summarise the components introduced above, and concretely reinforce the relationship between these components and the aims and objectives in the introduction. This model is our assertion, as scientists, about the underlying system structure and interactions between the unobserved states of interest in *C. elegans*. We are able to make predictions about the impact of individual parameters, at the level of individual synapses, on organism-level behaviours by simulating data using this model. We also introduced a number of the computational considerations when building and deploying the simulation and inference pipeline. By careful design we are able to make use of large, distributed HPC

clusters for accelerating computationally demanding inference procedures. We will also leverage this model to infer the unobserved latent states given observed behaviours. This allows us to make and test hypotheses about the unobserved neural function responsible for the observed behaviours. However, for these inferences to be informative, we must refine the model using observed data to ensure the model is as faithful as possible. Inferring both the distribution over unobserved latent states and parameter values conditioned on observed data is a central goal of this work. We therefore tackle these objectives in the next two sections.

4.2 State-space Estimation

We first tackle inferring the latent state of the worm from partial calcium fluorescence traces, given a fixed model. We begin with this for several reasons. Firstly, this is arguably the ideal deployment of the VPC, where a fixed set of parameters have been learned off-line, and a researcher is using these static parameters to impute latent states using a local desktop machine (as opposed to performing the much more computationally demanding joint state-space and parameter inference task). Secondly, the parameter estimation presented in Section 4.3 uses an SMC sweep with fixed parameters for estimating the model evidence. Therefore, if we are unable to reliably estimate the latent states or generate low-variance evidence estimates given the true model, then parameter estimation will inevitably fail. The method was also refined and developed on these fixed model experiments. Finally, the application of SMC is explained in detail here, such that when we go on to discuss parameter estimation we can simply use the SMC pipeline already described, without cluttering the explanation. We first relate each of the model elements introduced above to the expressions and densities originally defined when introducing SMC in Section 3.3.3. We then present experimental details and results from performing SMC on synthetic data.

4.2.1 Sequential Monte Carlo for *C. elegans*

We first define how the model specified above can be used as part of SMC to perform posterior inference. We introduced SMC and particle filtering at length in Section 3.3.3. The target distribution here is the posterior distribution over latent state given all observations and static parameters, $\pi(\mathbf{x}) = p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}, \boldsymbol{\theta})$. This provides the latent neural state imputation element of the VPC. Once this posterior has been solved for, forward simulation of particles distributed according to the last marginal, $p(\mathbf{x}_T|\mathbf{y}_{1:T}, \boldsymbol{\theta})$, provides posterior predictive

inference over state evolution. Finally the evidence approximation computed by SMC allows us to objectively compare models and hypotheses, which will be used later for parameter estimation. We now describe each term in more detail.

4.2.1.1 Transition Kernel

The transition kernel describes the evolution of the latent state, which for a homogeneous HMM is neatly defined as $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$, where $\mathbf{x}_{t-1}$ is the latent state at the previous time step, and $\boldsymbol{\theta}$ are the parameters of the transition kernel. Specifically, $\mathbf{x}_t$ is the 1018-dimensional latent state vector defined by the simulator. However, the transition kernel defined by the model outlined in Section 4.1 is deterministic. To induce a distribution over latent states the neural potentials are perturbed with Gaussian distributed noise. To sample from the transition distribution given the previous latent state vector and parameters, additive noise added to the neural potential, which is then passed through the model.

The variance of the perturbation is scaled according to the expected variance of that state, i.e. states that are expected to vary more have larger perturbations applied. We found this to dramatically improve the quality of the reconstructions across a range of parameters. The standard deviation of the noise is 5mV for the neuron with the largest dynamic range, while the smallest noise term used is 0.0005mV. These variances are determined at the start of each sweep by simulating a corpus of 48 noise-free datasets from the model and calculating the variance of the potential per-neuron. The initial membrane potentials used when generating this corpus of data are sampled from a broad distribution over achievable membrane potentials under the model, where the distribution used is $\mathcal{N}(-20\text{mV}, 0.033\text{mV}^2)$.

4.2.1.2 Likelihood Term

The likelihood term, $p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta})$, defines the (unnormalised) probability that the latent state $\mathbf{x}_t$ yielded the observed value $\mathbf{y}_t$. We use the calcium fluorescence model, defined in (4.5), as the likelihood term and score particles under this likelihood. The variance of the likelihood is scaled according to the expected variance of the observations. This means the particle filter sweep is forced to fit all neuron traces, and not simply those neuron traces that are most variable. The kernel is scaled according to:

$$\sigma_m^2 = 0.02 \times \left(\frac{\hat{\sigma}_m}{\hat{\sigma}_{\max}} + 0.1 \right), \quad \text{where} \quad \hat{\sigma}_{\max} = \max_{m \in 1:M} \hat{\sigma}_m, \tag{4.6}$$

where $\hat{\sigma}_m$ is the standard deviation of the m^{th} observation over time, and 0.02 is the default observation variance. A stabiliser, with value 0.1, is added to prevent overfitting in neurons with negligible activity. This variance term is computed once at the start of inference from the observed data. This scaling dramatically improved the SMC inference result, preventing those neurons whose dynamic range is greater from dominating the resampling process, crushing the significance of the activity in those neurons with a smaller dynamic range.

As a result of the dependency structure implied by the simulator (see Figure 4.1), the likelihood term is only dependent on the intracellular calcium concentration of the observed neurons. Therefore the covariances implied by the transition model are responsible for providing contraction in the posterior over the unobserved neurons.

4.2.1.3 State-prior and Initialisation of Particles

The final component in SMC is the initial distribution over state. We found that the initialisation of particles greatly affects the variance of the reconstruction and evidence approximation. Due to the computational cost of the simulator and high latent dimensionality, simply taking the brute-force approach of using more particles is not feasible. We therefore made a number of assumptions and approximations to improve the quality of the initialisation, while limiting any deviation from the formal mechanics of SMC.

Initially we considered using a large repository of pre-generated neural and body states as a discrete prior over state. However, these states rarely accurately reflected the true state of the worm leading to poor reconstructions. We therefore assume that the initial body pose and velocity are known (two components of $\mathbf{b}_t$). The muscular voltage is then initialised to zero, noting it develops quickly from neural activity and body pose. We stress that the body shape is not used after the initialisation step. We explore this opportunity in Chapter 5.

We also found that directly sampling 302 membrane potentials and calcium concentrations from a prior distribution led to crippling particle degeneracy and poor reconstructions. We therefore use a short procedure to leverage the model itself to refine the initial distribution from which membrane potentials are initialised. The calcium concentration in each observed neuron is initialised by performing importance sampling conditioned on the first observation. A model-dependent prior distribution over neural potential is then estimated by simulating data from the model and fitting a Gaussian distribution to this simulated data for each neuron. A large number of particles are then initialised using this distribution over potential, and

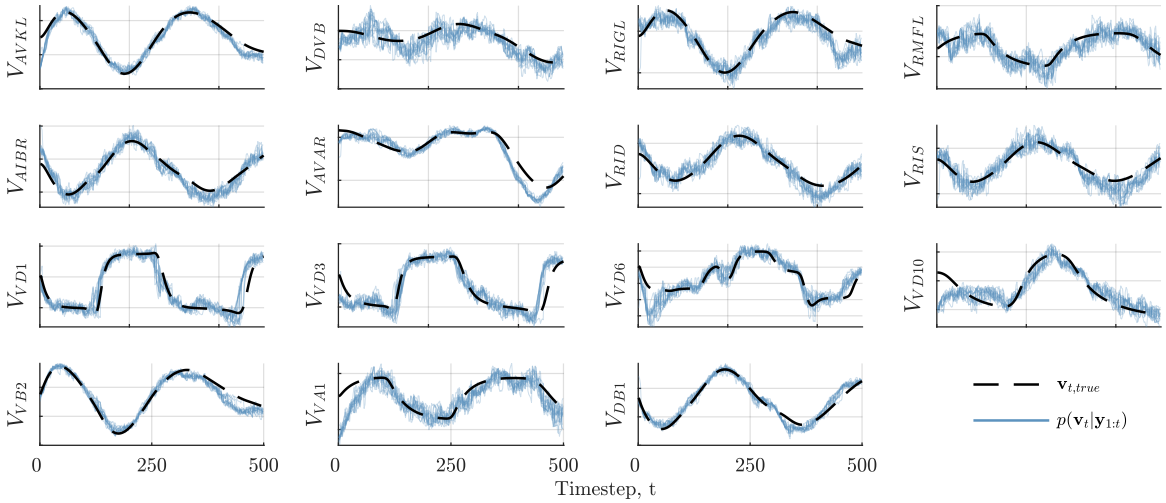


Figure 4.5: Results of the virtual patch clamp experiment introduced in Section 4.2.2. The true voltages for 15 neurons as generated by our simulator are shown as black dashed lines, with the SMC filtering distribution shown in blue. The rows correspond, top to bottom, to unobserved neurons, observed neurons, unobserved motor neurons, and observed motor neurons. We used the "default" parameters of our simulator, which are far too numerous to list even here, although, for consistency with the parameter estimation sections, we use $w_m = 9.0$ and $w_s = 10.3$.

are iterated once through the model. Particles are then resampled using the first observation. The initial potentials of those particles that survive resampling after a single step are then perturbed and are used as a discrete distribution from which the particles used in the sweep are initialised. This refinement was observed to yield initial particles with dramatically lower-variance importance weights, resulting in lower degeneracy and better overall performance. While this is a slight deviation from the formal mechanics of SMC, it largely aims to mitigate deficiencies introduced by using finite particle sets, noting that this procedure effectively pre-emptively removes those particles that would be immediately discarded by the SMC sweep. Importantly, this method incurs little computational cost.

4.2.2 Experiments

Our first experiment demonstrates recovering latent trajectories from synthetic data. A synthetic state trajectory is first generated by sampling from the model. We then apply SMC to condition simulations on the synthetic calcium data and compare the resulting reconstruction to the known ground truth.

For this experiment, the neural and body dynamics are simulated for a total of 500 time steps, corresponding to 5 seconds of data with $\delta t = 0.01$. Calcium fluorescence observations are generated for the 49 neurons identified in one of the datasets gathered by Kato et al. [2015].

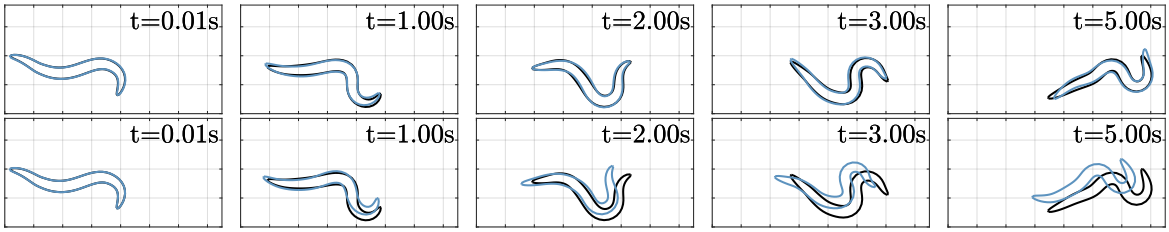


Figure 4.6: Top row: Aligned WormSim filtering reconstructions. Bottom row: Unaligned WormSim reconstructions. The shape of the reconstructed worm is ostensibly correct, but is offset from the true position due to integration error at the start of the trace. Since the extrinsic position of the worm does not influence the neural activity, or the likelihood term, this small displacement cannot be corrected. Therefore, for the purposes of presentation we apply a rigid body transformation to centre the body position of the particle being shown on the true pose, to highlight the estimation of body *shape*. This drift can be corrected however using the extrinsic body pose data, discussed later in Chapter 5. Note that here we are plotting a single particle drawn randomly from the filtering distribution, and hence there is no guarantee that the shapes will *exactly* align.

Observations are generated every 0.05 seconds. Intermittent observations are easily handled by SMC by simply not resampling at timesteps where there is no observation. The number of particles, $\mathcal{N}_p$, used in the SMC sweep was 1000, with 5000 particles used in the initialisation procedure. The wall-clock time was 800 seconds to complete a single SMC sweep when run on a single node equipped with 48 Intel Xeon Platinum 2.10GHz 8160F CPUs.

Results for this are shown in Figure 4.5 and 4.6. The particle distribution for a subset of $\mathbf{v}_t$ is shown in Figure 4.5. The true state is shown in black and the SMC filtering distribution is shown in blue. The blue reconstructions are congruent with the black trace, indicating that the latent behaviour of the complete system is being well-reconstructed despite partial observability and stochasticity in the generative model. Critically, neurons not directly connected to observed neurons (for instance `VD10`) are correctly reconstructed. This indicates that the regularising capacity of the model is sufficient to constrain these variables.

Further confirmation of the power of this method can be seen in the top row of Figure 4.6. This shows the predicted body shape closely matches the true shape, despite only observing the first body pose. There is however a small drift in the position and orientation of the worm during the early timesteps due to poor initialisations, shown in the bottom row of Figure 4.6. This initial drift cannot be corrected as the extrinsic position and rotation of the worm does not influence the neural activity. Therefore, for ease of visual comparison of the estimation of body *shape*, the reconstructions in the top row of Figure 4.6 are rigidly transformed to centre them on the true body. Explicitly conditioning on body shape from video data, as tackled in Chapter 5, would alleviate this issue.

This experiment shows that there is sufficient information in the partial calcium fluorescence observations and that the model has sufficient regularising capacity to reconstruct the latent states. However, this experiment assumed access to the true model. We therefore address recovering the model from observed data.

4.3 Parameter Estimation

In the previous section we introduced inferring the time-varying latent states from limited observations under a fixed model. In this section we tackle jointly learning the global parameter values and inferring the latent states. We consider two different approaches in this section. We first consider performing full posterior inference, sampling from the distribution $p(\mathbf{x}, \boldsymbol{\theta}|\mathbf{y})$, using PMMH, first introduced in Section 3.3.4.7. We then consider a relaxation from full posterior inference, and instead search for the single parameter set with the maximum posterior density, referred to as marginal maximum *a posteriori* (MMAP) parameter estimation, introduced in Section 3.2.

4.3.1 Methods

4.3.1.1 Inference

The first method we investigate is full posterior inference using particle marginal Metropolis-Hastings (PMMH). We introduced PMMH at length in Section 3.3.4.7, and hence do not repeat the low-level details here. PMMH allows us to efficiently draw samples from the posterior distribution $p(\boldsymbol{\theta}, \mathbf{x}|\mathbf{y})$. At its core, PMMH exploits the ability to sample from the true conditional over state conditioned on the observed data and fixed parameter values using an SMC sweep. A Metropolis-Hastings sampler is then constructed by defining a proposal over these static parameter values followed by the true conditional: $q(\boldsymbol{\theta}', \mathbf{x}'|\boldsymbol{\theta}, \mathbf{x}) = p(\mathbf{x}'|\boldsymbol{\theta}', \mathbf{y})q(\boldsymbol{\theta}'|\boldsymbol{\theta})$. A new parameter value is proposed, $\boldsymbol{\theta}'$, and latent states, $\mathbf{x}'$, are then proposed conditioned on this new parameter value from the true conditional distribution. This conditional is sampled from using SMC. Proposed parameters and latent state pairs are then jointly accepted or rejected according to a MH update step using the model evidence approximation computed by the SMC sweep. The key benefit of PMMH is that the user only needs to additionally specify a proposal over global parameters. The SMC sweep automatically provides a proposal over latent states that is perfectly adapted to the proposed parameter values. Designing a proposal over parameters is typically easier than defining

an efficient proposal over the *entire* latent space, as the parameters are often significantly lower-dimensional and more readily interpretable than the latent state.

We also investigate enhancing PMMH using parallel tempering, introduced in Section 3.3.4.5. PT runs multiple tempered MCMC chains in parallel. Each chain is running a PMMH sampler, where the chain is tempered by raising the likelihood to a power, reducing the relative contribution of the likelihood. Hotter chains are observed to move more freely in the smoothed parameter space. Each chain makes MH updates, but can also swap states with another chain. These swaps then allow parameter values with higher probability to trickle down to the cooler chains, and eventually to the coolest chain representing the target distribution. Crucially this swap operation requires no additional computational effort. We discuss the low-level implementation details of these methods in subsequent sections.

4.3.2 Optimisation

A relaxation of this full posterior inference is marginal maximum *a posteriori* (MMAP) parameter estimation. This instead poses the problem as optimisation, searching for the maximum *a posteriori* parameter values after marginalising over the unobserved latent states, $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} p(\boldsymbol{\theta}|\mathbf{y}) = \arg\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. The model evidence, $p(\mathbf{y}|\boldsymbol{\theta})$, is estimated using an SMC sweep, and the prior density over parameters is defined as before. The distribution over latent states is then recovered by using a point estimate on the parameters, $p(\mathbf{x}|\mathbf{y}, \boldsymbol{\theta}^*)$. We discussed in Section 3.4 the tradeoffs of optimisation and full inference approaches. Optimisation is generally a simpler objective than full posterior inference, at the expense of recovering a less expressive solution. Optimisation can also use gradient information to efficiently direct the algorithm to better regions of parameter space. However, it is not generally possible to differentiate through the pseudo-marginal model evidence, $p(\mathbf{y}|\boldsymbol{\theta})$, computed using an SMC sweep without placing significant restrictions on the class of models, or relying on prohibitively computationally demanding methods [Cappé et al., 2007; Le et al., 2018; Naesseth et al., 2018].

Therefore, to maximise the posterior probability using gradient ascent we use variational optimisation (VO) [Staines & Barber, 2012], introduced in Section 3.4.3, to maximise the pseudo-marginal posterior probability of the model parameters. Variational optimisation defines a Monte Carlo estimator for the gradient of a lower bound of the objective function. This bound is constructed by defining parameterised proposal, $q(\boldsymbol{\theta}|\boldsymbol{\phi})$, over the inputs to the

objective function, here $p(\boldsymbol{\theta}, \mathbf{y})$, and computing the derivative of the expectation with respect to $\boldsymbol{\phi} \in \boldsymbol{\Phi}$. This allows us to compute the gradient of a lower bound of the joint density, which is proportional to the posterior density of the model, where the model evidence is computed with the otherwise non-differentiable SMC. To our knowledge, this is the first time that pseudo-marginal methods have been paired with variational optimisation methods. We refer to this procedure as particle marginal variational optimisation (PMVO). Similar to PMMH, PMVO considers the latent state and parameter estimation separately. Therefore, we only need to define a proposal over parameters, and can reuse the SMC procedure introduced previously to estimate the distribution over latent states and compute the model evidence.

The proposal, $q(\boldsymbol{\theta}|\boldsymbol{\phi})$, used is an independent Gaussian proposal distribution over each parameter, such that $\theta_n \sim \mathcal{N}(\phi_n, \sigma_n^2)$, where σ_n^2 a fixed per-parameter variance. As a result, $\Phi = \Theta$. To initialise the proposal parameters, $\boldsymbol{\phi}_0$, we randomly sample $\mathcal{N}_r$ $\boldsymbol{\theta}$ parameters from the prior and select the parameter with the highest joint density, $\boldsymbol{\phi}_0 = \boldsymbol{\theta}_0^{(k^*)},\ k^* = \arg\max_{k=1:\mathcal{N}_r} p(\mathbf{y}|\boldsymbol{\theta}_0^{(k)})p(\boldsymbol{\theta}_0^{(k)}),\ \boldsymbol{\theta}_0^{(k)} \sim p(\boldsymbol{\theta})$, also shown in Figure 4.7. PMVO then proceeds by sampling from the proposal distribution, $\boldsymbol{\theta}_n^{(k)} \sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_{n-1}),\ k = 1:\mathcal{N}_r$, evaluating the likelihood at each of these points using SMC, $p(\mathbf{y}|\boldsymbol{\theta}_n^{(k)})$, multiplying by the prior probability, $p(\boldsymbol{\theta}_n^{(k)})$, and calculating an approximate derivative of the bound with respect to the parameters of the proposal, $\hat{\nabla}_{\boldsymbol{\phi}} U(\boldsymbol{\phi}_n)$, as specified in (3.118). This estimate of the derivative is then used with the stochastic gradient ascent algorithm ADAM [Kingma & Ba, 2014] to update the value of $\boldsymbol{\phi}_n$, denoted as $\boldsymbol{\phi}_n \leftarrow \texttt{ADAM}(\boldsymbol{\phi}_{n-1}, \hat{\nabla}_{\boldsymbol{\phi}}(\boldsymbol{\phi}_{n-1}), L)$, where the learning rate is denoted L. This process repeats, sampling from the proposal, calculating the joint density, estimating the gradient and then taking a gradient step, until some computational budget has been used or a stopping condition is met. Further augments such as introducing tempering of the joint density, annealing the learning rate and reducing the width of the proposal distribution can all be applied to improve performance.

4.3.2.1 Computation Allocation

For both inference and optimisation approaches, the model evidence, $p(\mathbf{y}|\boldsymbol{\theta})$, is a Monte Carlo estimate of the true model evidence, computed by SMC. Therefore, we must choose how to allocate computation. Running more particles in each SMC sweep yields a lower variance estimate of each likelihood. However, it may be more efficient to run multiple, independent SMC sweeps each with a higher variance evidence approximation, but then average over more

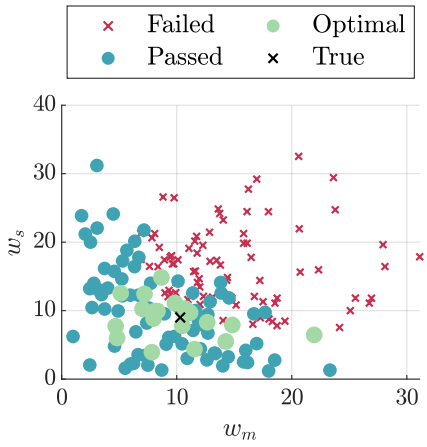


Figure 4.7: Figure 4.7: scatter plot showing failures in initialisation. Parameter values are drawn from the prior distribution. The true parameter value is shown as a black cross. Parameter values for which the body model failed to integrate to the required accuracy in the allocated computational budget, referred to here as the simulator to "crashing," are shown as red crosses. Parameter values that integrate but were not optimal in the batch of initial values are shown as blue dots, and the optimal parameters in each batch are shown as green dots. This shows that large regions of parameter space with mass under the parameter prior actually have zero mass once combined with the model. We return to discuss ameliorating simulator failure in Chapter 7.

estimates to achieve a lower variance likelihood estimate overall. Separate SMC sweeps are computationally independent and therefore can be efficiently distributed. Beyond this, there is a trade-off in parameter learning between making each evaluation as cheap as possible and taking more steps, or, making each evaluation as accurate as possible and taking fewer steps.

The number of particles used in each SMC sweep is denoted $\mathcal{N}_p$, the number of repeat sweeps per parameter value as $\mathcal{N}_s$, and the number of samples used within our VO gradient calculation and the number of temperatures used in the parallel tempering temperature ladder is denoted as $\mathcal{N}_r$, with temperatures denoted $\mathcal{T}_{1:\mathcal{N}_r}$. The number of steps taken, either in inference (MH steps) or optimisation (gradient steps), is denoted $\mathcal{N}_k$. The total computational expenditure of the learning process is therefore $\mathcal{B} = \mathcal{N}_p \times \mathcal{N}_s \times \mathcal{N}_r \times \mathcal{N}_k$. We normalise experiments by this quantity to allow for meaningful comparison of methods.

4.3.3 Autoregressive Model

To investigate the methods discussed above we first perform joint state-space inference and parameter learning in a simplified model family. Specifically, we use a first order

autoregressive (AR) model in-place of the neural simulator:

$$\mathbf{x}_t \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta}) = \boldsymbol{\theta}_{\mathbf{W}} \times \mathbf{x}_{t-1} + \mathcal{N}(0, \sigma_p^2 \times \mathbb{I}^{N\times N}), \tag{4.7}$$

$$\mathbf{y}_t \sim p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}) = \boldsymbol{\theta}_F \frac{\mathbf{x}_t}{\boldsymbol{\theta}_K + \mathbf{x}_t} + \boldsymbol{\theta}_d + \mathcal{N}(0, \sigma_m^2 \times \mathbb{I}^{M\times M}), \tag{4.8}$$

where $\mathbf{x}_t \in \mathbb{R}^N$ is the state vector at time t, and $N = 30$, $\mathbf{y}_t \in \mathbb{R}^M$ is the vector of observations at time t and M traces are observed, $\boldsymbol{\theta}_{\mathbf{W}} \in \mathbb{R}^{N\times N}$ is a sparse matrix dictating the evolution of the latent state, $\sigma_p^2 \in \mathbb{R}_+$ is the variance of the noise in the plant model, and $\sigma_m^2 \in \mathbb{R}_+$ is the variance of the observations. We use a saturating Hill-type function with a unit Hill coefficient as the observation model, similar to the full neural model. We assume that the parameters of the additive noise distributions, σ_p^2 and σ_m^2, are known. The observation function uses parameters of $\{\boldsymbol{\theta}_F,\ \boldsymbol{\theta}_K,\ \boldsymbol{\theta}_d\} = \{1.0,\ 1.0,\ 10.0\}$. The process noise and observation noise kernels are then taken to be independent Gaussian per dimension with diagonal covariance elements of $\sigma_p^2 = \sigma_m^2 = 0.01^2$. The initial distribution over state is defined as $p(\mathbf{x}_0|\boldsymbol{\theta}) = \mathcal{N}(0, 1.0^2 \times \mathbb{I}^{N\times N})$. This is also used at inference time. For each method, we perform a single importance sampling step at $t = 0$ to limit the degeneracy in the first step of the particle filter. We assume that all species are observed, $N = M$. The AR process is then iterated for 200 timesteps. Example trajectories are shown in Figure 4.9a.

We use a sparse, anti-symmetric random transition weight matrix, $\boldsymbol{\theta}_{\mathbf{W}}$, mimicking the structural characteristics of the connectivity matrix in *C. elegans*, while still enabling the generation of stable, oscillatory traces. This matrix is constructed by uniformly sampling elements from the strictly upper triangular region, with a target sparsity, α. The resulting matrix has $D = \lfloor \alpha N(N-1)/2) \rfloor$ parameters. We use a target sparsity of $\alpha = 0.9$. This results in a connectivity matrix with a 44 dimensional parameter space, and hence $\boldsymbol{\theta}_{\mathbf{W}}$ is effectively represented as $\boldsymbol{\theta}_{\mathbf{W}} \in \mathcal{W} = \mathbb{R}^{44}_{\geq 0}$. We assume the matrix of binary connectivities, $C_{n,k} = \mathbb{I}\left[\boldsymbol{\theta}_{\mathbf{W},n,k} \neq 0\right],\ k > n$, is known and that the mapping between the sparse matrix form in (4.7) and this dense vector representation is implicit. Individual connectivity parameters are drawn from a Rayleigh distribution with known mean, $\theta_{W_d} \sim \mathrm{Ray}(\mu),\ d = 1 : D$, where $\mu = 0.0185$. The lower triangular matrix is then set to the negation of the upper triangular portion to enforce anti-symmetry. We place an independent Rayleigh prior over each of the parameters, denoted $p(\boldsymbol{\theta})$. The learning objective in this section is recovering the matrix of transition weights, $\boldsymbol{\theta}_{\mathbf{W}}$, from observed data, $\mathbf{y}_{1:T}$.

For all methods in this section we use $\mathcal{N}_p = 200$. As PMMH operates at only a single location, $\mathcal{N}_r = 1$, we average over more SMC sweeps and so use $\mathcal{N}_s = 5$, and take $\mathcal{N}_k = 8000$ MCMC steps. This yields a budget $\mathcal{B} = 8 \times 10^6$. The proposal distribution is an independent multivariate normal centred on the current parameter value, with diagonal covariance $\mu^2/10 = 0.00185^2\mathbb{I}^N$. We then enhance this approach through the use of parallel chains and likelihood tempering. We tuned the PT hyperparameters (e.g. number of replicates, temperatures) using manual tuning, starting with two chains, and increasing the number of chains and tuning individual parameters until performance saturated. The frequency of swaps and the mixing of chains can be used as diagnostic tools for tuning each replicate. For this experiment we use $\mathcal{N}_r = 5$ and $\mathcal{T}_{1:\mathcal{N}_r} = [4.0, 2.0, 1.33, 1.14, 1.0]$. Due to the increase in evaluations at each step, we set $\mathcal{N}_s = 2$, and reduce the number of steps $\mathcal{N}_k$ to $4,000$, resulting in an expenditure $\mathcal{B} = 8 \times 10^6$. We also use a temperature-dependent proposal distribution, where higher temperature chains use a wider proposal. The hottest chain, $\mathcal{T}_1$ proposes from the prior each time to induce the most mixing. The standard deviation of the proposal for chains $\mathcal{T}_{2:\mathcal{N}_r}$ are $(0.00180, 0.00156, 0.00147, 0.00138)$. Swaps are proposed between adjacent chains at every step, where the adjacent pair is drawn from a uniform distribution. Each SMC sweep is performed on a separate node.

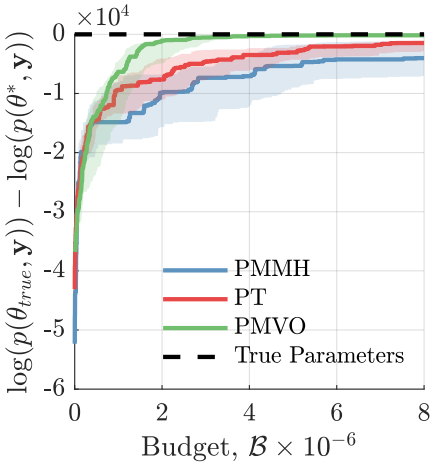
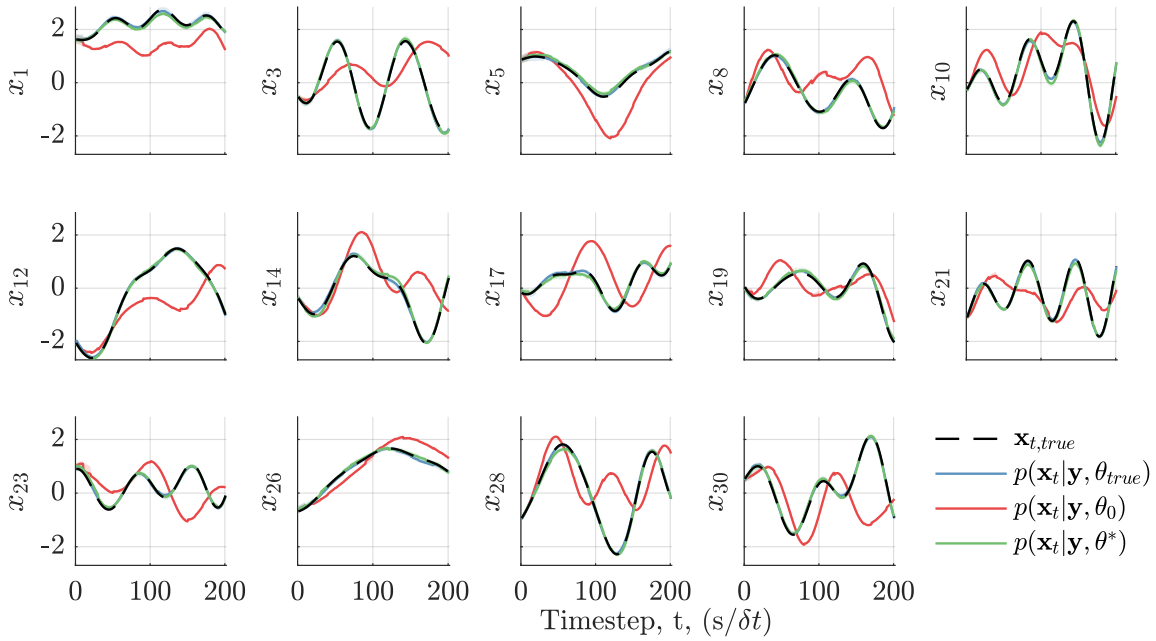
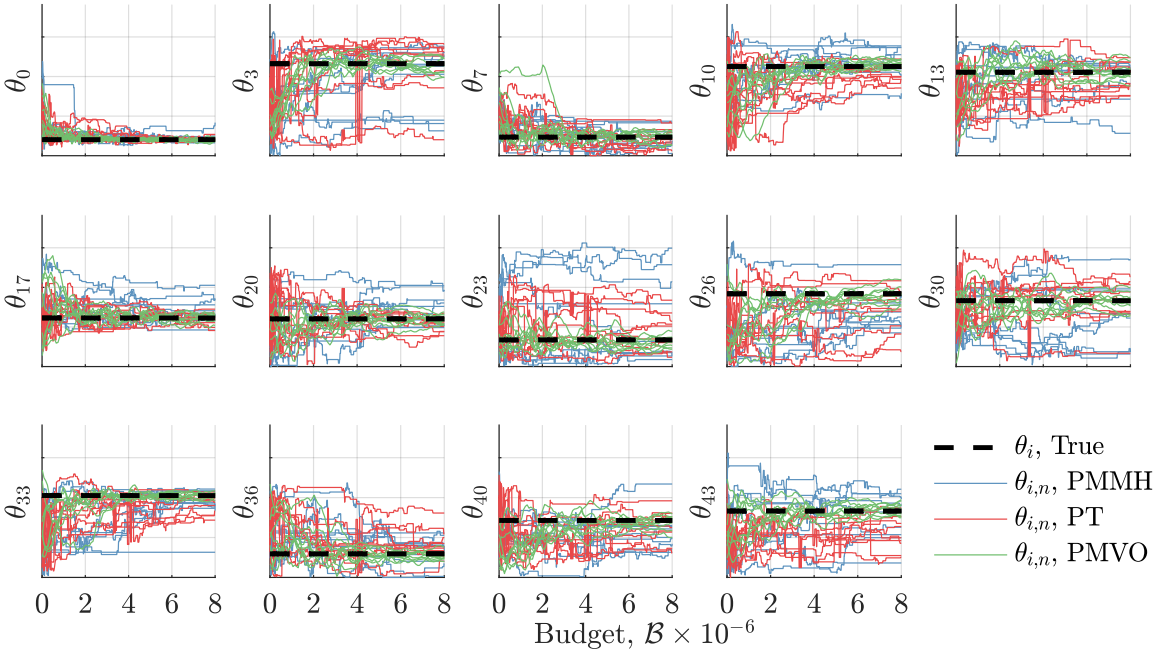


Figure 4.8: Comparison of PMVO, PMMH and PT methods, for parameter estimation in the autoregressive example introduced in Section 4.3.3, across eight experimental repeats. Shown is the difference in the log joint density between the true parameters and the learned parameters for the methods. We normalize all methods by computational budget.

For PMVO we use $\mathcal{N}_r = 20$ and $\mathcal{N}_s = 2$, and, to match the budget, limit ourselves to use $\mathcal{N}_k = 1,000$, such that our expenditure is $\mathcal{B} = 8 \times 10^6$. We use an isotropic Gaussian proposal distribution centred on the current parameter values, with a standard deviation of $\mu/20$. The width of this proposal is reduced by a factor of 5 during the optimisation. Similarly, we begin estimating the gradient on a tempered joint density, with initial temperature, $1/\beta = 100$, which is exponentially annealed to the true density. We also temper the SMC sweep by relaxing the observation kernels used in the SMC sweep by expanding them by a factor of



(a)



(b)

Figure 4.9: Figure 4.9a: The true latent state sequence is shown as a black dashed line, and the filtering distribution from SMC using the true parameters (blue), initial parameters (red) and optimised parameters (green), for 14 of the 30 states. Figure 4.9b Comparison of PMVO, PMMH and PT methods for parameter estimation in the autoregressive example introduced in Section 4.3.3. 8 experimental repeats are shown. We normalise all methods by computational budget. The convergence of 15 of the 44 parameters to the true value, shown in black.

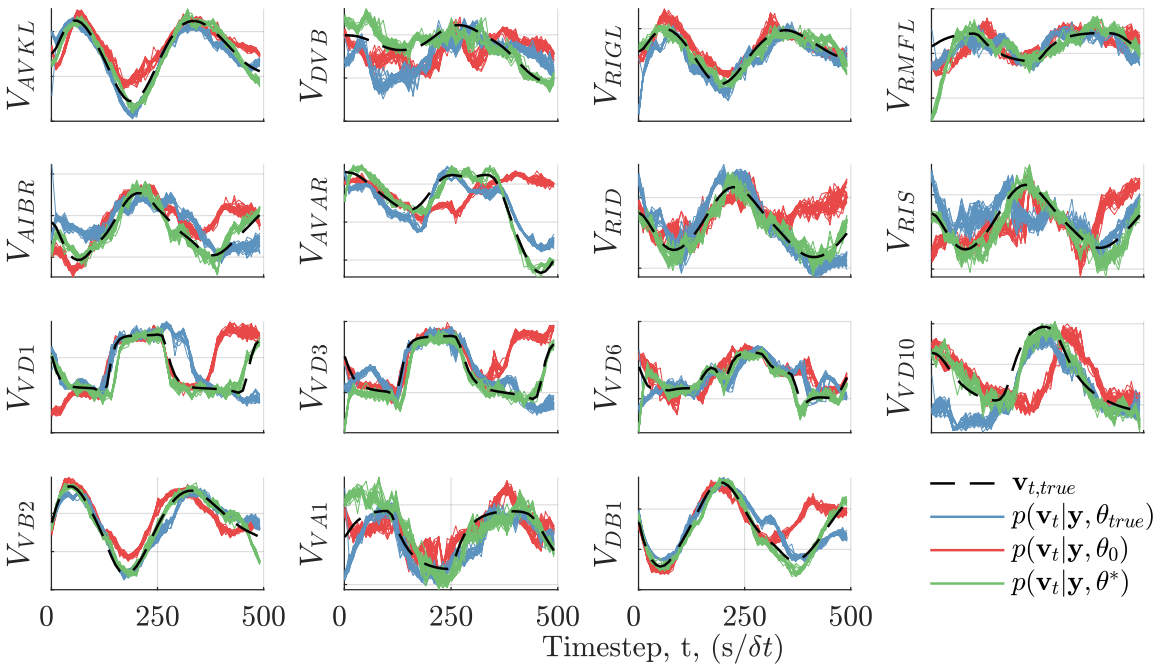
2.5 initially, again, logarithmically annealing this towards the originally specified kernels. We use the ADAM optimiser [Kingma & Ba, 2014] with an initial learning rate of 10^{-3}, which is reduced using optimisation to 10^{-4}.

We show the results of this in Figures 4.8 and 4.9. In Figure 4.9a we show, in black, the latent state of the AR process. The SMC filtering distribution conditioned on the true parameters, randomly initialised parameters, and the learned parameters are shown in blue, red and green respectively. Well-optimised parameters lead to better reconstructions. In Figure 4.9b we show the convergence of the parameter values for each of PMMT, PT, and PMVO (across eight experimental repeats). We see that PMVO recovers the true parameter values more quickly and reliably than PMMH and PT. This improved performance is reflected in Figure 4.8 by the error between the joint density of the true parameters and the MAP parameters reducing more quickly when using PMVO.

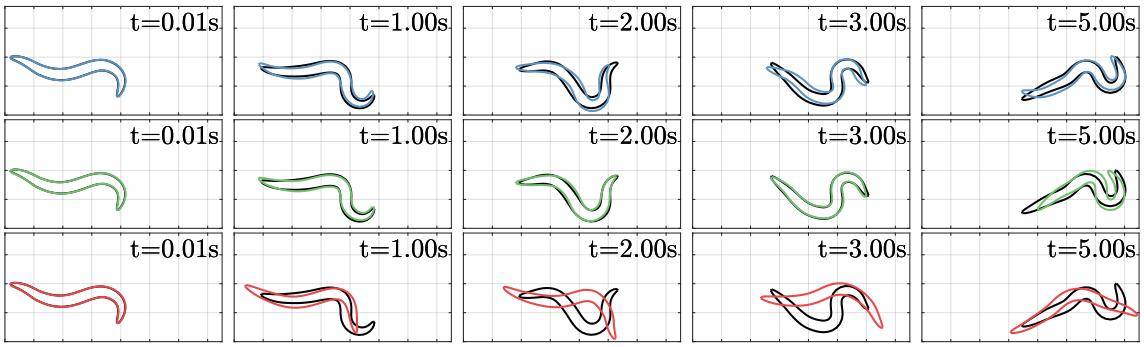
4.3.4 Synthetic *Caenorhabditis elegans* Parameter Estimation

Motivated by the performance of PMVO, we now return to our original objective of recovering parameter values from synthetic *C. elegans* data. Specifically, we estimate the two parameters we introduced by integrating the Wicks model and WormSim, the strength of motor stimulation, w_{m}, and proprioceptive feedback, w_{s}. The values of these parameters cannot be measured and so must be learned from data. We also note these parameters are *highly* influential on the resulting traces (as can be seen particularly clearly in Figure 4.11b). We place a Rayleigh prior over each parameter, with mean parameter equal to the true parameters, $w_{\mathrm{m}} = 9.0$ and $w_{\mathrm{s}} = 10.3$. We note, however, that the likelihood term dominates the prior term, and so the prior, in practice, only guides the initialisation. We observe that many of the initially sampled parameter values cause the simulator to immediately crash, as shown in Figure 4.7. These parameter values are immediately discarded.

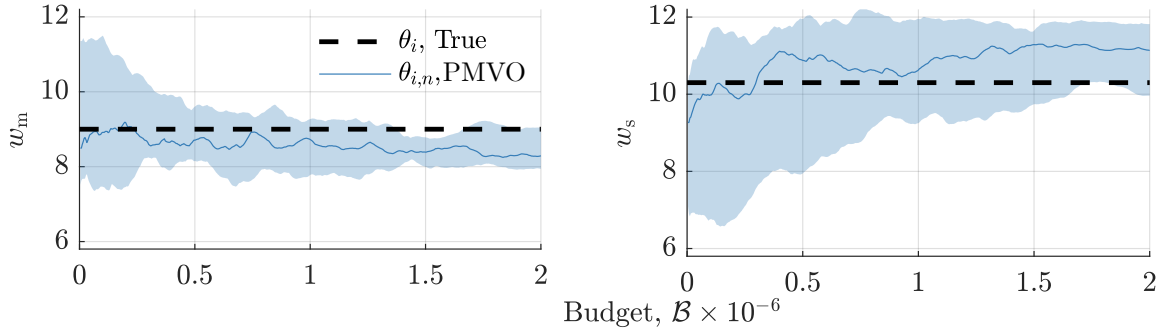
For this optimisation we use $\mathcal{N}_k = 500$, $\mathcal{N}_r = 8$ and $\mathcal{N}_s = 1$. The SMC sweep uses 500 particles in the initial step, sub-sampling to 90 particles after the first observation. The traces used were 500 timesteps in length. All other experimental settings are the same as in Section 4.2. Each individual SMC sweep takes approximately 90 seconds to complete and are each performed on a separate node equipped with 48 Intel Xeon Platinum 2.10GHz 8160F CPUs. The wall-clock time was no more than 17 hours for a single experimental repeat. We use the ADAM optimiser [Kingma & Ba, 2014] with a learning rate of 0.01.



(a)



(b)



(c)

Figure 4.10: Latent state imputations for the *C. elegans* experiments described in Section 4.3.4. Figure 4.10a shows the filtering reconstructions of the neural potentials conditioned on the true generative parameters (blue), initial (unoptimised) parameters (red), and optimized parameters (green). Note that we show a single SMC sweep performed during learning using just 90 particles, and hence these cheap sweeps are noisier than the SMC sweeps shown in Figure 4.5. In Figure 4.10b we plot a single body pose (using the same colour scheme) drawn from the filtering distribution. We see that the unoptimised parameters lead to poor reconstructions of the latent neural potential and body pose. Using the optimised parameters lead to good imputations. Figure 4.10c shows the evolution of parameters during the PMVO optimisation. Shown are median, upper and lower quartile across 20 random restarts.

The results of this experiment are shown in Figure 4.10. In figures 4.10a and 4.10b we show the imputed voltage traces and body poses when using the true parameters (blue), initial parameters (red) and optimised parameters (green). As before, recovery of better parameter values facilitates better imputation and reconstruction of latent states. We note that the sweeps shown here use just 90 particles, and hence each of these individual sweeps are noisier than if we reran the sweeps using the configuration in Figure 4.5. Figure 4.10c shows the convergence paths of the two parameters being addressed for 20 initialisations. This experiment shows that joint latent state inference and parameter estimation in *C. elegans* models using PMVO is viable.

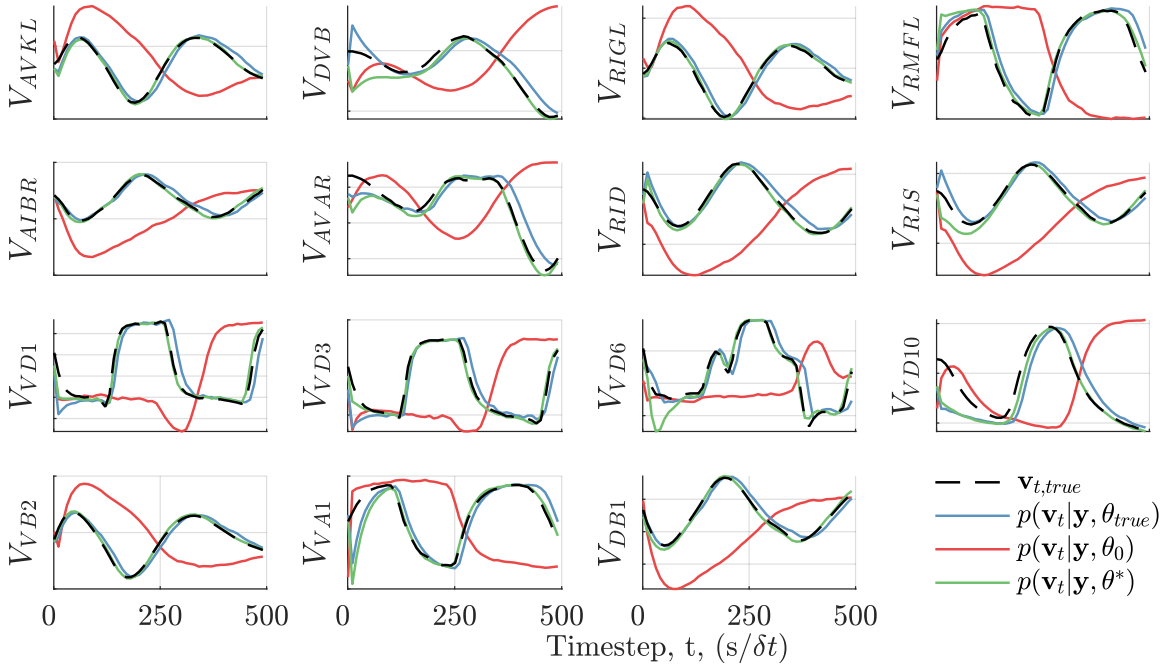
4.3.5 Fully Observed Virtual Patch Clamp Experiment

Finally, we investigate the scenario where we are able to observe *all* of the neurons in the *C. elegans* connectome. Of course, this scenario is infeasible. However, observing and identifying more than 49 neurons is not infeasible as calcium imaging techniques continue to develop, the field of view of calcium imaging expands, and multicolour fluorescent proteins become available. To confirm the benefit of this supposition, we perform SMC and parameter optimisation conditioning on the fluorescence of all 302 neurons. The experimental settings for this are the same as the experiment presented previously, aside from using only 400 particles to calculate the initialisation and just 48 particles in the SMC sweep itself, and reducing the size of the noise kernels used at inference time in the SMC sweep by a factor of 20.

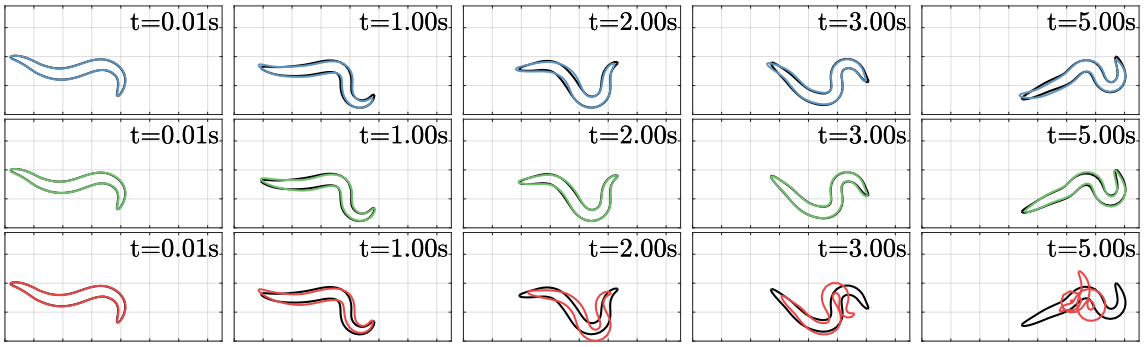
The results of this are shown in Figure 4.11. The SMC sweep performs very well, practically perfectly recovering the unknown parameter values and perfectly reconstructing the latent states and the body pose. This suggests that as the number of observed fluorescent traces is increased, we can reasonably expect latent state and parameter estimation to become more accurate and less computationally onerous.

4.4 Discussion

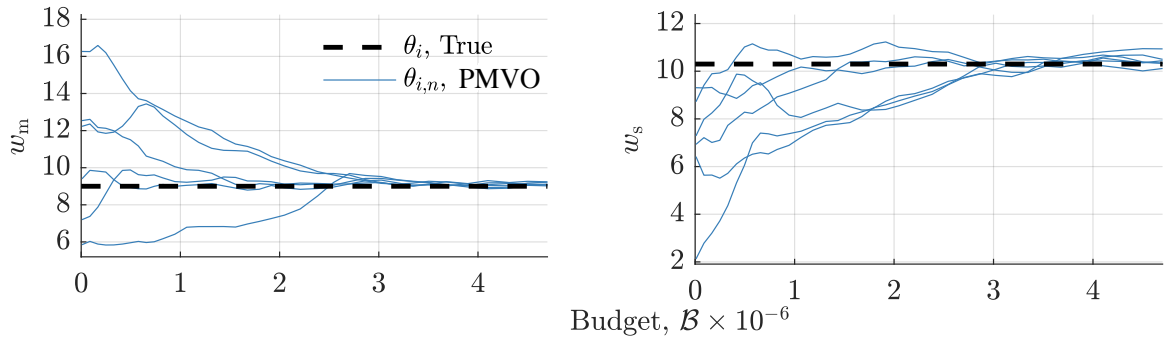
In this chapter we have explored performing Bayesian inference in whole-connectome neural and body *C. elegans* models. We referred to using whole-connectome neural models for latent state imputation, parameter estimation, and subsequent posterior predictive inference and unconditional generation as a *virtual patch clamp*. We developed a brain-body *C. elegans*



(a)



(b)



(c)

Figure 4.11: Latent state imputations for the experiment introduced in Section 4.3.5 where all calcium fluorescence traces are observed. Figure 4.11a shows the filtering reconstructions of the membrane potentials conditioned on the true generative parameters (blue), initial (unoptimised) parameters (red), and optimized parameters (green). We see that the optimised parameters perfectly reconstruct the latent potential traces. Figure 4.11b shows a single particle from the filtering distribution for true, optimised, and initial (unoptimised) parameters. Figure 4.11c shows the convergence of parameters during PMVO optimization.

simulator, prioritising computational tractability, and then used particle-based SMC methods to infer unobserved latent variables from non-invasively gathered, partial observations. We then demonstrated that we can simultaneously estimate unknown global parameter values by maximising the pseudo-marginal model evidence. Once parameters have been estimated, neuroscience experiments can be performed *in silico* by programmatically modifying the simulator. These digital experiments can be framed as performing posterior predictive inference in the learned model, and allow a wider range of hypotheses to be tested cheaply and quickly, compared to *in vivo* experiments.

While previous work has investigated performing imputation of neural spikes, membrane potentials, calcium dynamics, model parameters and connectivities from data [Aitchison et al., 2017; Friedrich et al., 2017; Gerkin et al., 2018; Speiser et al., 2017; Vogelstein et al., 2009, 2010], these studies do not operate under a biologically accurate model of the dynamics of a *whole* connectome, instead investigating individual neurons or small networks of fully observed neurons. As part of our work, we are able to exploit the regularisation provided by connectome-scale dynamics to estimate the state of neurons only distantly connected to observed neurons.

Recent articles discussing open research issues pertaining to computational modelling of *C. elegans* have been published by the *C. elegans* community [Gerkin et al., 2018; Larson et al., 2018; Sarma et al., 2018; Stiefel & Brooks, 2019; Randi & Leifer, 2020]. Figure 2.4, adapted from Sarma et al. [2018], outlines the community planned development pipeline for *C. elegans* simulation. This work addresses the implementation of the box simply labelled "optimization" by Sarma et al. [2018]. Not only is this the first instance of whole-connectome neural simulators being conditioned on data, we also believe this to be one of the largest non-linear state-space models in which joint state and parameter inference has been performed.

However, in this work we only consider synthetic data and a small subset of the total parameters that define the simulator. The data used corresponds to just five seconds of data, whereas the traces published by Kato et al. [2015] are minutes in length. The inferences presented above already required significant computational effort, and hence, expanding these methods to real data is no small undertaking. Longer traces require linearly more computational effort, and hence even performing a single SMC sweep using minutes of data is prohibitively computationally demanding. As the space of parameters considered

grows larger, the parameter inference task becomes exponentially more difficult. Therefore, unfortunately, simply using more particles or running more SMC sweeps is an inadequate method for scaling the tools presented here to real data. However, the methods presented provide a starting point for developing more sophisticated, powerful, and scalable models and inference pipelines. We introduce some extensions identified as during this work that help build towards this end goal.

Foremost, more complete datasets are becoming available. As *in vivo* calcium imaging techniques improve, more neurons can be simultaneously observed, and, crucially, positively identified through the use of multicolour fluorescent stains. This allows the SMC sweep to be conditioned on more data, and reduces the number of neurons for which no fluorescence is observed. We demonstrated that when observing more neurons latent state imputation and parameter estimation becomes markedly easier. However, we also suggest that the simulator can be conditioned on worm body pose data. We found that conditioning on the first body pose was required to make inference tractable. Therefore, developing the inference tools to allow the entire trajectory to be conditioned on an observed pose in addition to fluorescence is pivotal. Worm locomotion can be easily recorded using standard laboratory equipment, and hence, being able to perform inference using just worm body pose is an attractive opportunity. We explore this theme more in Chapter 5.

We also found that initialising the SMC sweep is non-trivial. The simulator is sufficiently computationally demanding and high-dimensional that taking a brute-force approach of initialising orders of magnitude more particles is infeasible. Poor initialisations lead to poor and high variance reconstructions. Even when observing all neurons, as in Figure 4.11a, initialisation remains non-trivial. Therefore, a promising line of investigation is leveraging analytically tractable models, such as linear dynamical systems, that approximate the analytically intractable and computationally demanding, but high-fidelity, full neural simulator. These approximate models are computationally inexpensive to simulate and perform inference in. These models can also leverage low-dimensional embeddings of the worm state to further enhance and expedite inference, where such linear models may be sufficient to accurately capture the behaviour of the worm. This principle is extended by the use of *multifidelity methods* [Fernández-Godino et al., 2016], where low-fidelity models

are used in conjunction with high-fidelity models to enhance inference in the high-fidelity model. We discuss this more in Chapters 5 and 8.

The simulator and inference techniques we propose are also richly extensible. Better models for elements of *C. elegans* behaviours, such as locomotion simulators [Palyanov & Khayrulin, 2015], neural simulators [Gleeson et al., 2018], multi-compartment ion dynamics [Kuramochi & Doi, 2017] and sensory stimuli [Izquierdo & Beer, 2013] exist; although these models incur significantly more computational cost. These models may still require development to explain specific behaviours, for instance, the kinds of habituation that *C. elegans* exhibits [Ardiel & Rankin, 2010]. However, adding these additional models will further improve the fidelity of the simulation, and will allow for more observed phenomena to be succinctly explained under the model. When additional models of these dynamics are developed, by design, they can be straightforwardly integrated into our software toolchain.

This concept highlights a more general principle: the model itself is a parameter. In this chapter we searched over particular scalar valued parameters to maximise model evidence. However, we can also perform inference or optimisation over the structure of the model. Automatically searching over model structures promises to automate certain aspects of neuroscience research, proposing new models of behaviour and interactions, and improving and selecting those models that are promising. We explore this theme more in Chapter 6. We use automated program synthesis themes to jointly recover, from (synthetic) calcium imaging data, the programmatic representation of the synaptic plasticity rule applied to update the connectome weights during simulation, and, the static parameters of the simulator conditioned on the specific rule and the observed data.

We also noted a curious idiosyncrasy of the body simulator. For particular states or parameter values, the simulator crashes. Crashes were observed when the body state was directly perturbed, or, for poorly chosen parameter values. More generally beyond specifically our model however, a crash simply describes when the simulator does not exit properly, and instead raises an exception. These crashes waste computational effort and reduce particle diversity. We explore this theme further in Chapter 7, and use deep generative models to propose perturbations that avoid failures.

We tackle these topics in the following chapters. We then return to evaluate the whole body of work, and discuss further opportunities and future directions in more detail in Chapter 8.

5
Body Inference

In the previous chapter we considered conditioning exclusively on calcium fluorescence traces. Although calcium fluorescence imaging gives an unprecedented view into the neural state of the worm, it is not without drawbacks. Foremost, only a spatially co-located subset of the connectome can be imaged simultaneously. This leaves large, interconnected sub-networks of the connectome unobserved. These unobserved neurons may only be distantly connected to observed neurons, and hence the observed neurons provide little information about the state of these unobserved clusters. More particles must therefore be used to cover the state space, and the posterior distribution over these states remains broad. Furthermore, each observed neuron must have the corresponding neuron label identified manually. This requires considerable effort from expert annotators, and is not *guaranteed* to assign correct neuron labels to each trace. Those traces that cannot be confidently identified are effectively removed from the observed set, further reducing the number of available observations. Finally, calcium fluorescence imaging only observes a noisy transform of a covariate of the neural potential. Hence, using calcium fluorescence introduces additional and non-trivial complexity to the model and inference procedure, and introduces an additional set of parameters that must be measured or estimated.

In this chapter we therefore instead consider conditioning on raw video footage of the worm body pose and locomotion. The body pose of multiple worms can be recorded simultaneously using readily available bench-top equipment [Husson et al., 2005; Yemini et al., 2013]. Specimens require no special preparation and video recording adds little logistical complexity to the experimental procedure. Crucially, the body pose also provides a degree of information on neurons for which fluorescence was not recorded. We therefore propose that body pose is *complementary* to fluorescence data. However, beyond this, the best-case scenario is arguably one where latent neural states and physiological parameters can be recovered from body pose data alone. This would remove the need for calcium

imaging apparatus and specimen preparation, and would therefore make inference in *C. elegans* far more readily accessible.

In this chapter we develop the intuition and tools to make conditioning on body pose possible. We develop a pipeline for pre-processing worm body poses, and show that this processed data can be used to infer latent states and recover global parameter value from synthetic data. We then apply our pipeline to real data, although only achieving limited success. For this study we use the original WormSim model [Boyle et al., 2012], introduced in Section 2.6. The motivation behind this choice is, in part, to evaluate the sufficiency of the originally specified model, with a view to integrating the additional complexity in future iterations of the VPC work. However, this is also motivated by the possibility of separating the inference processes, where a simplified model is used to convert the pose and locomotion into a structured representation before subsequent, and more efficient, application of the VPC. This general idea was introduced in Section 4.4 and is discussed more in Section 5.3. On a high-level, this work is very similar in motivation and execution to the VPC work introduced in the preceding chapter. We therefore refer the reader to the VPC chapter (Chapter 4), neuroscience models (Section 2.6), neuroscience data (Section 2.5), SMC (Section 3.3.3) and PMMH (Section 3.3.4.7) sections for more information on the low-level details.

5.1 Methods

We wish to perform posterior inference, recovering the time-varying neural and physical latent states, and the static, global parameter values, conditioned on video recordings of *C. elegans* locomotion. We denote this posterior distribution $p(\mathbf{x}_{0:T}, \boldsymbol{\theta}|\mathbf{I}_{1:T})$, where $\mathbf{x}_{0:T}$ are the time-varying latent states of the worm, $\boldsymbol{\theta}$ is the unobserved physiological global parameters, and the raw video footage is denoted $\mathbf{I}_{1:T}$. The first topic we discuss is preprocessing the video feed such that it is more immediately usable for inference. We then introduce the specifics of the model used, and conclude by discussing the pipeline for performing the posterior inference.

5.1.1 Data Preparation

The raw video is captured as a series of T individual frames. The dimensions of the video feed depend on the apparatus used. Wide field of view microscopes, capable of recording the whole petri dish (see Figure 2.6a), typically produce images that are thousands of pixels in width and height. Alternatively, high-magnification microscopes produce images with

smaller frame sizes, but with higher image resolutions (see Figure 2.6b). These microscopes often record just a single worm and use a mobile stage to keep the worm in the field of view. While it is hypothetically possible to directly condition on the image data, this would require specification of a likelihood term for the latent state of the simulator given a video frame. Specifying such a term is a research task in its self, as this model must be sufficiently accurate and tuned for the diverse range of experimental conditions and apparatus used. Furthermore, performing inference directly on the video feed may be prohibitively computationally demanding.

Therefore, we first use pre-existing computer vision pipelines to automatically convert the video feed into low-dimensional features. We discussed these *worm trackers* [Husson et al., 2005] in Section 2.5. These trackers are fast, bespoke, and inexpensive computer vision pipelines designed by domain experts to operate under a range of experimental conditions. We propose using these trackers as a feature extractor, performing much of the computer vision "heavy lifting" offline from inference.

Nearly all trackers produce an estimate of the centreline of the worm, referred to as a *skeletonisation*. Most often, the skeleton is represented as a sequence of D x-y control points approximating the centreline of the worm. The value of D is typically dependent on the particular software implementation and configuration used. This value is typically lower for wide field of view microscopes ($D = 11$ in Figure 2.6a) and is higher for high-magnification microscopes ($D = 49$ in Figure 2.6b). We denote application of the tracker to convert the raw video frames, denoted $\mathbf{I}_t \in \mathbb{Z}_+^{W \times H \times 3}$, into a skeleton of the worm, denoted $\mathbf{g}_t \in \mathbb{R}^{2D}$, where typically $2D \lll W \times H \times 3$, as $\mathbf{g}_t \leftarrow H_{\text{skeletonise}}(\mathbf{I}_t)$.

These control points are noisy and, crucially, are irregularly spaced estimates of the centre of the worm. For wide field of view microscopes with low D values, the noise and irregularity of the control points can be significant (i.e. if the centreline is only resolved with single pixel fidelity). Therefore, we denoise and regularise observations, such that they are amenable for inference for any value of D. To denoise the data we fit a cubic spline through the D control points, denoted $\mathbf{s}_t \leftarrow H_{\text{spline}}(\mathbf{g}_t)$. We do not enforce that the spline passes through each control point, and hence the spline provides a controllable level of denoising. The amount of denoising is a hyperparameter, but is easily selected by visual inspection on a small number of samples. We note that the amount of smoothing required

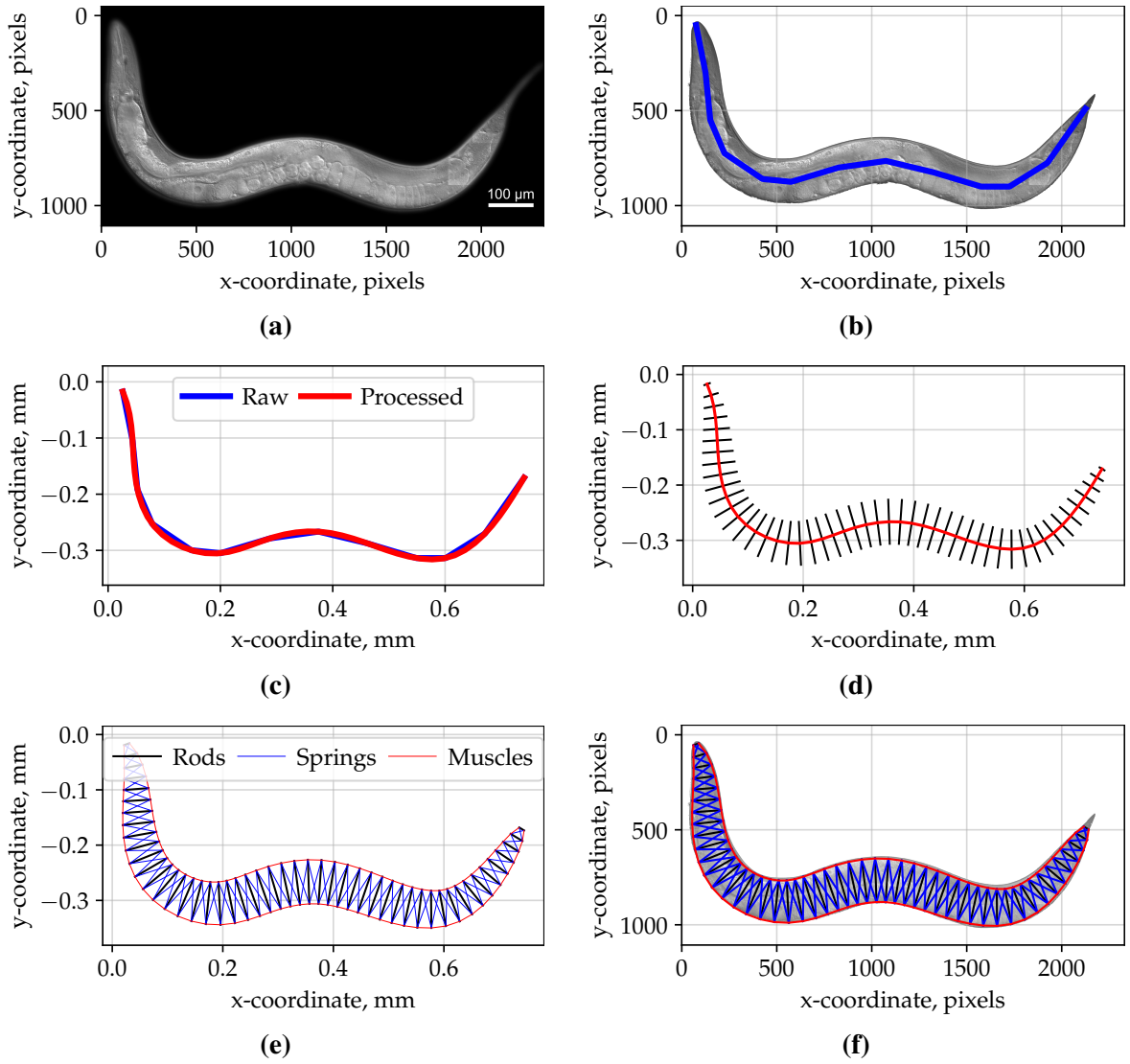


Figure 5.1: The pre-processing pipeline used to prepare raw image observations. Figure 5.1a is an example of a raw image, reproduced with permission from the Chin-Sang laboratory. Figure 5.1b then shows an accurate skeletonisation of the worm, with $D = 11$. Figure 5.1c shows the spline fitted to the skeletonisation. 5.1d then shows the $M = 49$ upsampled control rods fitted through the spline. Figure 5.1e shows the physical components of the WormSim representation. Figure 5.1f then shows this representation overlaid on to the original worm. We note that this uses a high-resolution worm image and hence the original skeletonisation is very accurate. Lower resolution input images yield less accurate skeletonisations and increase the need for the denoising behaviour of the spline.

is generally a function of D, where smaller D values require more smoothing. We then regularise the data by generating M equidistant control points along the spline, denoted $\mathbf{y}_t \leftarrow H_{\text{upsample}}(\mathbf{s}_t)$, where $\mathbf{y}_t \in \mathbb{R}^{2\times M}$. The application of a worm tracker, spline fitting and the control point generation is therefore fully denoted:

$$\mathbf{y}_t \leftarrow H_{\text{upsample}}(H_{\text{smooth}}(H_{\text{skeletonise}}(\mathbf{I}_t))). \tag{5.1}$$

These regularly spaced control points are independent of the D value that was used by the

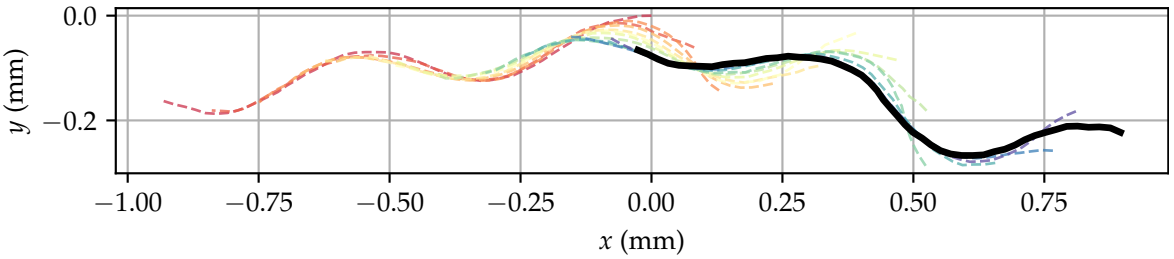


Figure 5.2: Time series of processed body poses, $\mathbf{y}_{1:T}$. Five seconds of data is shown, with early time points shown in red, and later time points shown towards purple. The final pose is shown in black. This is the data we condition on. Data drawn from the dataset made public by Yemini et al. [2013].

worm tracker. Once the preprocessing is complete the raw images can be discarded, and the inference procedure uses only the sequence of low-dimensional, regularised skeletons, $\mathbf{y}_t$. An example of a skeletonised and upsampled worm is shown in Figure 5.1, and a time series of processed body poses is shown in Figure 5.2.

5.1.2 Model

In the previous section we introduced the raw data we wish to condition on, and the set of preprocessing steps we apply to denoise and regularise the data such that it is more amenable to inference. We now discuss the model in which we perform inference. For this work, we consider the original WormSim model [Wicks et al., 1996], described at length in Section 2.6. We refer to the position, velocity and muscular states of the simulator as the *physical* state of the worm, and denote this state $\mathbf{b}_t \in \mathbb{R}^{49\times3+49\times3+48\times2} = \mathbb{R}^{390}$. The muscular activation is driven by a simplified neural network, defined by 24 binary *neural* states, denoted here $\mathbf{v}_t \in \{0,1\}^{24}$. The total state of the worm at time t is denoted $(\mathbf{b}_t, \mathbf{v}_t) = \mathbf{x}_t \in \mathcal{X}$, where this state is representable as a 414-dimensional vector.

WormSim defines a parameterised deterministic function mapping the neural and physical state from one time point to the next time point. To induce a distribution over trajectories we first experimented with perturbing the latent states with Gaussian distributed noise. However this frequently led to the ODE integrator used being unable to solve the differential equation sufficiently accurately, causing the simulator to crash, and raising an exception instead of returning an iterated state. This failure rate can be as high as 80% for even modest perturbations. We investigate avoiding these crashes more in Chapter 7.

Hence, we take a more conservative approach and only perturb the neural state by flipping each neural state with a fixed probability. The probability of a bit-flip is a hyperparameter we chose manually. Low bit-flip probabilities lead to practically deterministic behaviour and a

very narrow distribution over locomotion patterns. Setting the probability too high prevents sustained and realistic locomotion. This bit-flipping, coupled with the deterministic action of the simulator, is the transition kernel of the parametric HMM in which we perform inference, denoted $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the static, global parameter values of the simulator.

The observed data, $\mathbf{y}_{1:T}$, is the sequence of T upsampled body poses discussed in the previous section. These poses are defined by M two-dimensional control points representing the centreline of the worm, such that $\mathbf{y} \in \mathbb{R}^{T\times M\times 2}$. WormSim uses a series of $N = 49$ approximately equidistant control points to represent the centreline of the worm. Hence (and by design) we are able to select $M = 49$, such that the upsampled observations can be used directly as noisy observations of the position coordinates. A point-wise potential function between control points in the observed data and the latent state can then be directly used as a likelihood term. We use an independent multivariate Gaussian likelihood centred on the physical body state coordinates:

$$p(\mathbf{y}_t|\mathbf{x}_t) = \prod_{n=1}^{N} \mathcal{N}(\mathbf{y}_{t,n}; \mathbf{b}_{t,n}, \sigma^2\mathbb{I}^{2\times 2}), \tag{5.2}$$

where σ^2 is a hyperparameter representing the covariance of the observation kernel. In our experiments, we investigated a range of settings for σ^2 and evaluated, by manual inspection on synthetic data, which value resulted in the best reconstructions and latent state imputations.

We also note here that the acquisition rate of recording microscopes is often far below the integration timestep used by the model. For wide field of view microscopes, acquisition rates can be as low as three frames per second. Datasets generated using a mobile stage are often not analysed when the stage is moving, resulting in sequences of frames with no observations. As in the VPC, these missing observations are easily handled by SMC, by simply not resampling on steps where there is no observation available.

5.1.3 WormSim State Fitting

Having specified the transition kernel and likelihood, to fully specify the joint distribution, $p(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$, we only need to further specify the prior over state and parameters. We tackle the former now. In the VPC we first experimented using a discrete prior over worm states, constructed using unconditional generation. However, we found this led to exceptionally poor reconstructions, due to the high-dimensional state space, diversity of achievable states, dependency on parameter values and the low variance of achievable locomotion trajectories.

We therefore assumed that the initial body pose was available, observing that this was practically required to facilitate meaningful state-space inference. This observation was a motivating factor for this work, verifying that this assumption can be made in practice. We therefore develop a second pipeline for data-driven initialisation of the full WormSim state. This pipeline uses the first two observed data points to estimate the initial state, $\mathbf{x}_0$, and are then discarded such that they are not used in the evidence computation.

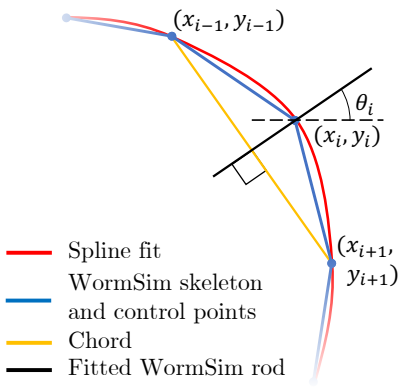


Figure 5.3: Schematic of estimating the angle of the WormSim rods. To estimate the i^{th} angle, we draw a chord between the upsampled $i-1$ and $i+1$ WormSim coordinates, and assume that the rod is perpendicular to this chord. This is empirically verified in Figure 5.4.

The first step is fitting the x-y coordinates of the rods. This is trivial as we have already pre-processed the observed data to be in the same representation used by WormSim. Therefore, we set the x-y coordinate of each rod equal to the corresponding pre-processed control point. After inspecting synthetic data, we observed that the rod is generally perpendicular to the smooth centreline of the worm.[1] We estimate the angle of the i^{th} rod, denoted θ_i, as perpendicular to the chord drawn between the $i-1^{\text{th}}$ and $i+1^{\text{th}}$ rod (akin to Euler central difference). The first and last elements are estimated using just the adjacent point (akin to Euler forward or backward difference). The result of this estimation is denoted by a sequence of tuples: $(x_{0,i}, y_{0,i}, \theta_{0,i})_{i=1:N}$. A schematic of this is shown in Figure 5.3, and an example is shown in Figure 5.1.

We can also estimate the velocity of each of the control points, $\left(\dot{x}_{t,i}, \dot{y}_{t,i}, \dot{\theta}_{t,i}\right)_{i=1:N}$, using an Euler forward difference between the estimated pose for the first and second observation. We also include results of this in Figure 5.1. We investigated using Euler central difference, but ultimately selected forward difference, citing negligible improvement in the fidelity of the estimation while requiring the use of three observations instead of two.

We experimented using this procedure in a particle filter sweep to initialise the physical position of the worm, and randomly initialising the remaining muscular and neural potentials. However, the observed worm simply "swam away" and the reconstructed worms got "left behind." Further inspection of the latent traces suggested that the muscular potentials, and

[1] Note that WormSim does not use a hinged assembly for the body position, rather the rods float under the force of the contractile and spring elements, i.e. the rods are not guaranteed to be perpendicular or equispaced.

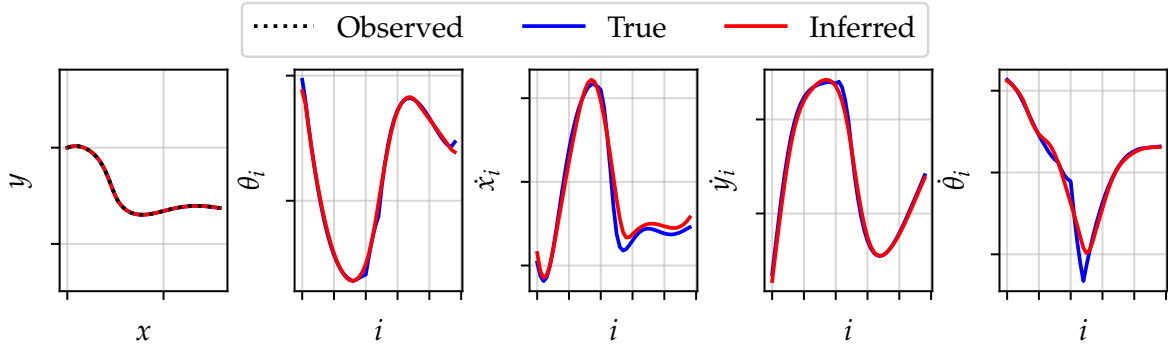


Figure 5.4: Results of estimating further physical states of the worm on synthetic body poses. We predict from the upsampled body pose (leftmost figure) the control rod angle, x-velocity, y-velocity and angle velocity using the procedure outlined in Section 5.1.3. We see that we are able to accurately estimate the unobserved physical components.

particularly the neural potentials, take a non-negligible time to develop from a given body position, and start producing locomotion. Brute-force initialisation using more particles was found to be ineffective and incredibly computationally demanding. We therefore resolved to also estimate the muscular and neural potentials from the initial observation.

The potentials are more difficult to estimate than the physical states, as they are not reliably defined by simple transformations of the physical position, and are highly dependent on the parameters of the simulator. To reliably and rapidly infer these states would require an amortised inference procedure at initialisation. We therefore tested two simple initialisation heuristics: an iterative refinement approach and a regression approach. The iterative refinement approach initialises the body shape and velocity of the worm as above, and initialises the muscular and neuron states to random values. The simulator is then iterated. The iterated muscular and neural states are then overwritten into the first time step, and the process repeated for a finite number of steps. This approach is preferable as it directly uses the model, and hence uses the current parameter values with no additional inputs or artefacts. However, this approach yielded unreliable performance, with some latent states being recovered reliably and other latent states consistently diverging from the true state.

The second approach we tested, and ultimately selected, is to use a small neural network to directly predict the initial latent muscular, $\hat{v}_{m,0,i}$, and neural, $\hat{v}_{n,0,i}$, potentials from the observed body position. This is illustrated in Figure 5.5. The network was trained by minimising the Euclidean distance between true states and estimated states on a large corpus of synthetic data generated using a range of parameter values. We apply Gaussian noise and bit-flipping to the predicted muscular and neural states to produce an initial distribution with

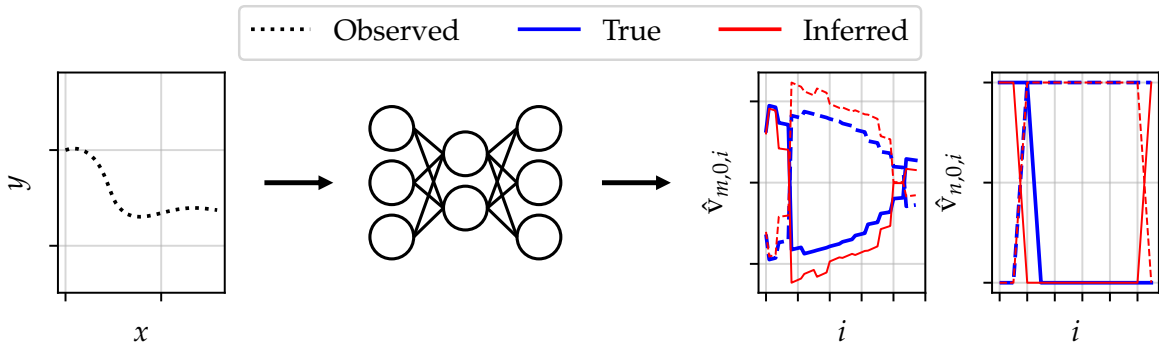


Figure 5.5: Illustration of the MLP described in Section 5.1.3 used to initialise the muscular and neural state. The body pose (leftmost figure) is passed through a two-layer, fully connected MLP to predict the 48 initial dorsal (solid line) and ventral (dashed line) muscular voltages, $\hat{\mathbf{v}}_{m,0}$, and 24 initial dorsal and ventral neural states, $\hat{\mathbf{v}}_{n,0}$. We subsequently add Gaussian noise to the muscular voltage, and randomly bit-flip the neural states to induce a distribution over the states.

some entropy. A drawback to this approach is the dependence on a static, pre-trained artefact to predict states that are highly dependent on the model parameters. However, we found that a simple network produced sufficiently accurate reconstructions for a range of parameter values and was observed to accelerate the development of neural state, preventing the reconstructed worm from being left behind. Establishing a more principled way to initialise these states is a topic we are considering. However, this problem is subsumed into initialising the full neural simulator (as in Chapter 4) once combined, and hence we do not dedicate more time to it here.

5.1.4 Inference

We have now fully defined the model in which we perform inference and the observations on which we condition. The foremost objective is to recover the posterior over time-varying latent states, $\mathbf{x}_{0:T}$, and static global parameter values, $\boldsymbol{\theta}$, given the (processed) observed data, $\mathbf{y}_{1:T}$, denoted $p(\mathbf{x}_{0:T}, \boldsymbol{\theta}|\mathbf{y}_{1:T})$. We use particle marginal Metropolis-Hastings (PMMH), first defined in Section 3.3.4.7, to draw samples from this distribution. For the SMC proposal, proposing latent states given observations and parameters, we use a particle filter sweep sampling from the true conditional distribution, $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}, \boldsymbol{\theta})$. Particles are initialised using the procedure above. Particles are then iterated by randomly bit-flipping neural states and calling WormSim. We score particles under the likelihood defined in (5.2) and use systematic resampling. We use a Gaussian proposal distribution over parameters with a fixed variance, and define a uniform prior over their respective domain.

We note that in Figures 5.6 and 5.8 we plot the reconstructions from a *single* SMC sweep conditioned on the *MAP* parameter values for clarity, as opposed to the distribution compiled

during PMMH. This is partly for ease of interpretation, and also so we can inspect the filtering and smoothing distributions separately. However, it is primarily because this is the most likely use-case of this approach, where a single set of parameters have been found offline, and individual SMC sweeps are then being conducted using static parameters to estimate the latent variables on limited computational hardware.

5.1.5 Implementation

We re-use much of the implementation developed as part of the VPC work, discussed at length in Section 4.1.5. A Python instance is started and acts as a controller. One WormSim instance is started per processor available, and communicates with the controller through a ZeroMQ socket. The controller coordinates all WormSim executables, transferring the state to the executable when a particle needs to be iterated. Iterated particles are returned to the controller, and are resampled using systematic resampling when an observation is encountered. Once a sweep is complete, the controller stores the latent states and the likelihood of parameters. The parameters are then updated on the controller process using a Metropolis-Hastings update. We are able to achieve close to 100% CPU utilisation during sweeps when using moderately sized particle pools (≥ 3 particles per CPU).

5.2 Experiments

We now test the inference process outlined above. We begin by applying the methods to synthetic data, before applying the methods to real data.

5.2.1 Synthetic Data

We begin with synthetic data such that we can evaluate the baseline performance of inference by comparing the inferred distributions to the known ground truth. It was also by experimenting on synthetic data that we identified the need for and developed the more involved initialisation procedure.

To generate the synthetic data we perform unconditional generation by sampling $2T + 2$ timesteps from a fixed model. We discard the first T samples to ensure that the worm is in steady state conditions and is independent of any inconsistencies in the initial state. The initialisation procedure then uses the first two observations, which are then discarded, leaving a trace of T steps. We use $T = 500$ for this experiment, representing five seconds of data. We subsample the data by a factor of four, simulating an acquisition rate of 25Hz, and use

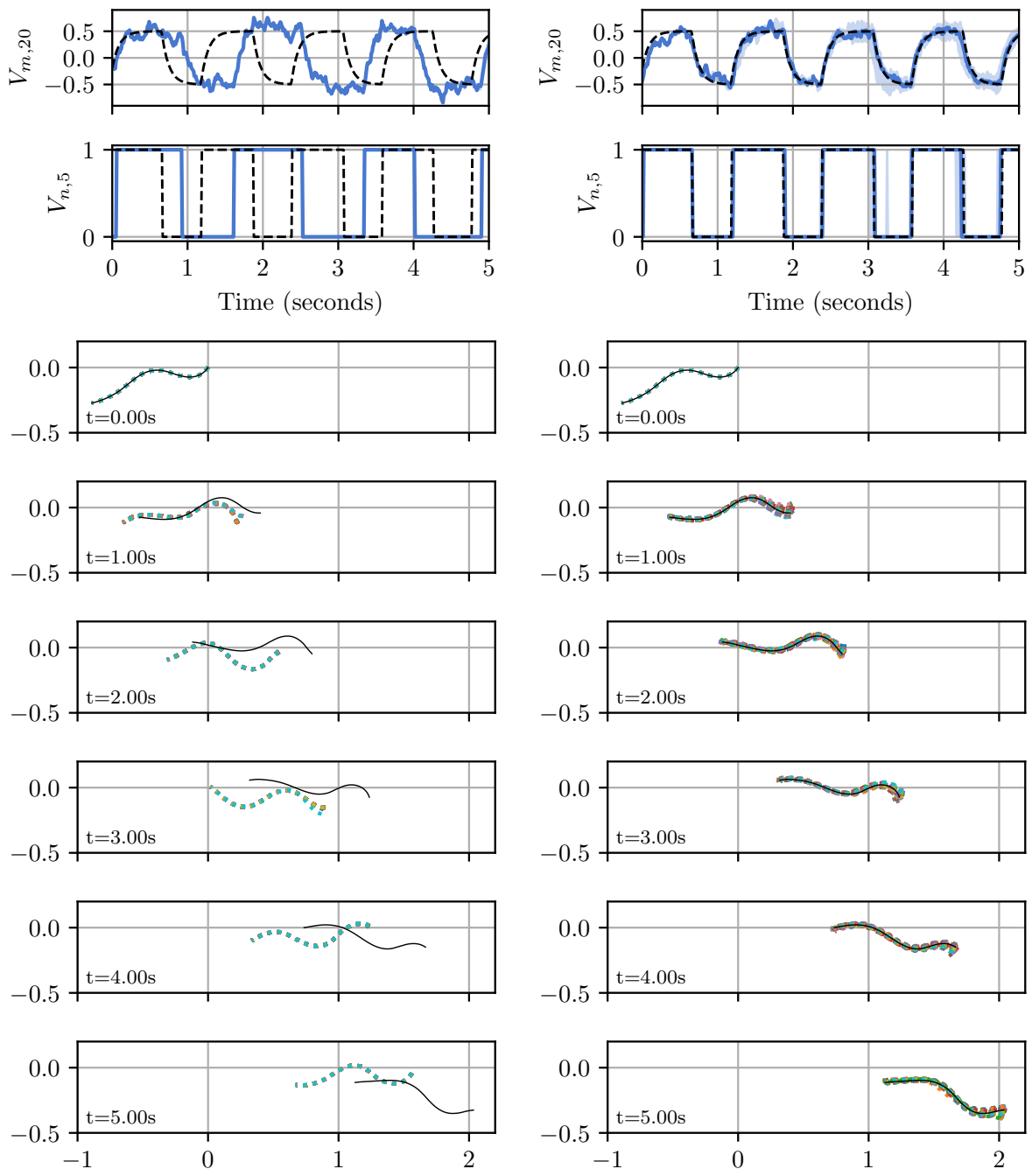
$M = 49$, mimicking the real data we go on to use. We add a small amount of Gaussian noise to the position of each observed point.

We perform joint state-space and parameter inference using PMMH, first introduced in Section 3.3.4.7, inferring the entire latent state sequence, $\mathbf{x}_{0:T}$, conditioned on the upsampled and regularised sequence of observations, as well as the global parameters, $\boldsymbol{\theta}$. We estimate four parameters we found to be highly influential on the simulated data. The *hysteresis*, $\theta_1 \in \mathbb{R}_+$, defines the threshold which must be crossed for a neural unit to change state. Higher hysteresis values generate locomotion with higher curvatures. The *surface viscosity*, $\theta_2 \in [0, 1]$, determines the passive resistive forces on the surface of the worm by interpolating between the viscosity parameters of water and solid agar. The third and fourth parameters dictate the change in neuromuscular innervation along the body of the worm.[2] Physical units closer to the head receive more excitation, with the amount of excitation decaying linearly along the worm. The first unit is innervated with a strength of $\theta_3 \in \mathbb{R}_+$ and the final unit is innervated with a strength of $\theta_4 \in \mathbb{R}_+$. However, the realistic range for all parameters is $[0, 1]$, such that $\boldsymbol{\theta} \in [0, 1]^4$. We consider all other parameters to be fixed and known. We use an independent Gaussian proposal for each of these parameters, with a standard deviation of 0.025, and use 90 particles per SMC sweep.

The results of this experiment are shown in Figures 5.6 and 5.7. Figure 5.6 shows the smoothing distribution over latent voltage traces and filtering distribution over body positions, conditioned on the observed data and the initial parameters, $\boldsymbol{\theta}^{(0)}$, and the MAP parameters, $\boldsymbol{\theta}^{(*)}$. We see that latent states are poorly reconstructed when using the initial parameters, with crippling particle degeneracy (see smoothing reconstruction) and low diversity in the particles (see filtering reconstruction) preventing meaningful reconstructions. In contrast, the reconstruction is nearly perfect when using the learned parameters, with only modest particle degeneracy and good particle diversity in the filtering distribution.

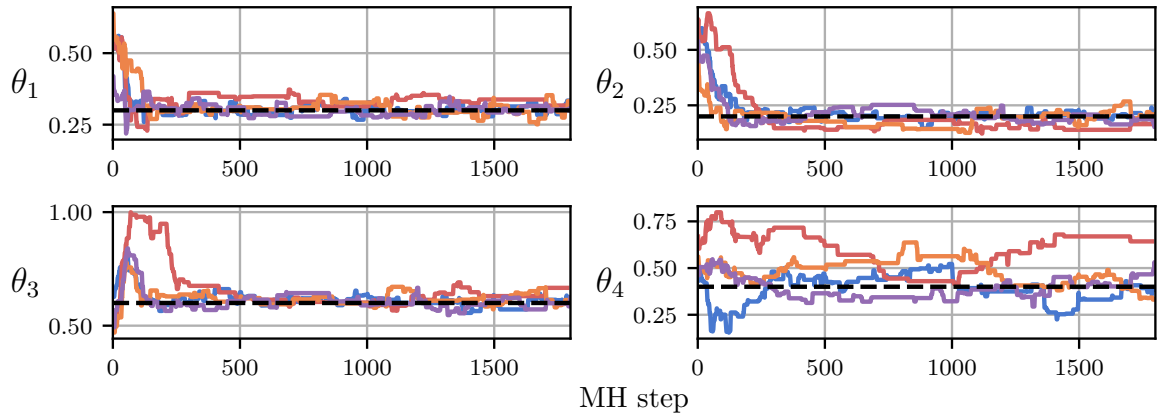
In Figure 5.7 we show four MH chains (in different colours) for the four parameters being inferred. The true parameter values are indicated by a black dashed line. We see that the parameter values quickly approach the true value for θ_1, θ_2 and θ_3, suggesting that the posterior distribution over these parameters is sharply peaked. We find θ_4 explores the space more, suggesting that the posterior distribution over θ_4 is more diffuse. We collapse

[2] This functionality is closely related to w_{m} in Chapter 4.

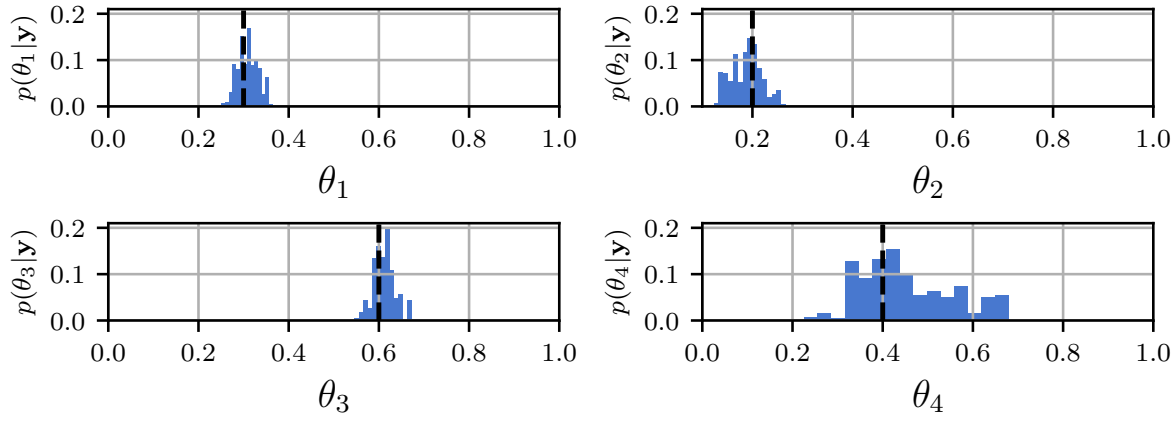


(a) Initial parameters, $\boldsymbol{\theta}^{(0)}$ **(b)** MAP parameters, $\boldsymbol{\theta}^{(*)}$

Figure 5.6: Results of latent state estimation on synthetic data for the experiment introduced in Section 5.2.1. Figure 5.6a shows reconstructions using parameters drawn from the prior, $\boldsymbol{\theta}^{(0)}$, whereas Figure 5.6b shows reconstructions using the MAP parameters, $\boldsymbol{\theta}^{(*)}$. The true state is shown in black. The top two rows show the median and quartiles of the smoothing distribution over muscular and neural voltage. The bottom six rows show the filtering distribution of body pose over time. The reconstructions using the initial parameters are poor, and there is gross particle degeneracy (indicated by low diversity in the filtering distribution and only a single surviving particle in the smoothing distribution). In contrast, the state is accurately estimated using the MAP parameters, there is notably more diversity in the body shapes and there is reduced particle degeneracy (indicated by a broader range of states in the filtering distribution and more surviving particles in the smoothing distribution). The parameter estimation is shown in Figure 5.7.



(a)



(b)

Figure 5.7: Results of parameter estimation for the synthetic experiment introduced in Section 5.2.1. The true parameter value is indicated as a black dashed line. Shown in Figure 5.7a are four separate Markov chains for each of the four parameters being estimated. We can see that the posterior is highly concentrated in three of the four parameters, θ_1, θ_2, and θ_3, corresponding to neural hysteresis, surface viscosity and the head neuromuscular activation. We see that the posterior over the fourth variable, the tail neuromuscular activation, is much less concentrated. This is confirmed in Figure 5.7b where we collapse the chains to a histogram.

the four chains and plot the aggregate histogram for each parameter, discarding the first 250 samples as burn-in. These histograms confirm that significant mass is placed on the true parameter value for all parameters.

This simple synthetic experiment shows that there is sufficient information in the body pose to recover the neural state used by the simulator, and, estimate important physiological parameters from just the body pose.

5.2.2 Real Data

We now apply the methods to real *C. elegans* data released by Yemini et al. [2013]. This repository of data includes the raw video footage of the worms, examples of which are shown

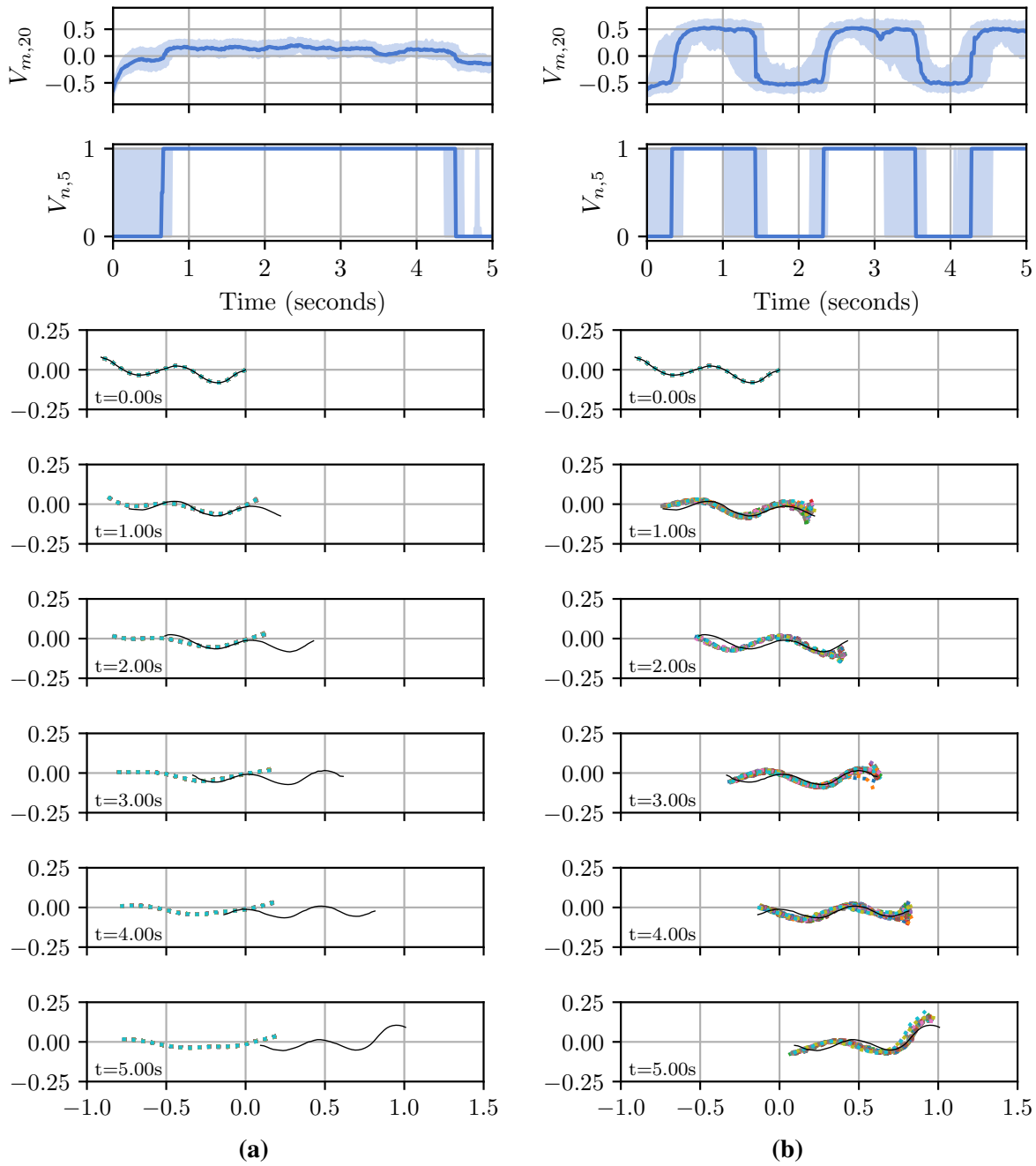


Figure 5.8: Reconstructions of latent states for the experiment introduced in Section 5.2.2 using real *C. elegans* data [Yemini et al., 2013]. In Figure 5.8a we show the filtering distribution over latent muscular voltages, neural voltages, and body position using the initial parameters of the PMMH chain. In Figure 5.8b we show the imputations using the MAP parameters. The reconstructions using the MAP parameters are better than the initial parameters, suggesting that better parameter have been found. However, all reconstructions fail to faithfully capture the evolution of the pose, especially in the tail of the worm. This suggests that the model is insufficient to faithfully capture the complexity of the real data.

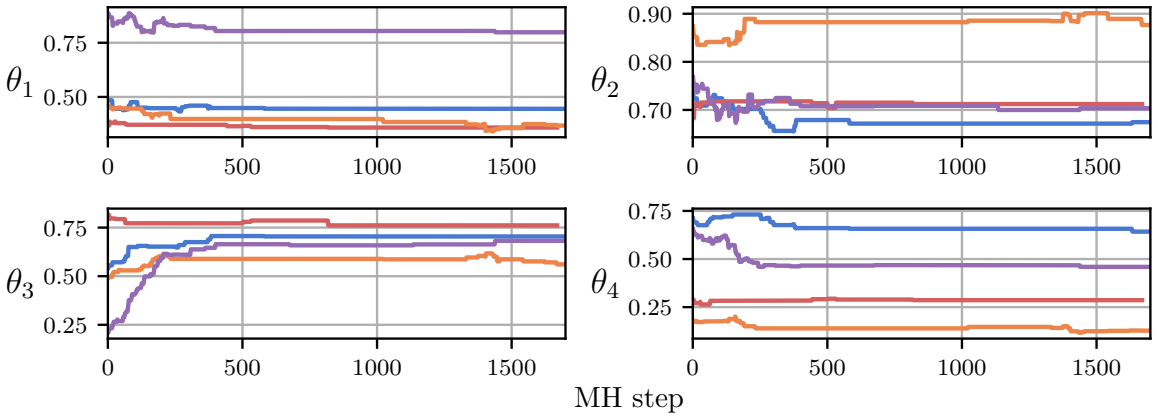


Figure 5.9: Parameter Markov chains for the real data experiment introduced in Section 5.2.2. We see that there is notably less mixing, and that the acceptance rate of the chain is exceptionally low. This lack of mixing may be alleviated through the use of parallel chains.

in Figure 2.6. The repository also includes skeletonisations of the worms, extracted using Worm Tracker 2.0 [Baek et al., 2002; Schafer Laboratory, 2020]. This means we do not need to do the image processing stage ($h_{\text{skeletonise}}$) ourselves and directly use the processed skeletons. We implement a pipeline for automatically gathering and processing data from this repository, and a GUI for quickly inspecting the raw data traces, and clipping and formatting data traces into segments for subsequent analysis.

We perform joint state-space and parameter inference using the same PMMH inference procedure used for synthetic data. The results of this are shown in Figures 5.8 and 5.9. We can see in Figure 5.8 that the reconstructions using the MAP parameters are notably better than using the initial parameters, with the general locomotion envelope of the worm being effectively captured. However, the body shape is poorly reconstructed, especially in the tail of the worm. The latent state imputations in Figure 5.8 are markedly more diffuse than in the synthetic experiments. This suggests that the model is not expressive enough to represent the complexity in the observed data. We see in Figure 5.9 that the MH chains do not mix effectively, and have an exceptionally low acceptance rate.

5.3 Discussion

In this chapter we discussed inferring neural states and parameters from raw video footage of worms. Using raw video for inference has several benefits over calcium fluorescence imaging. Firstly, only basic apparatus and no special specimen preparation is required, and hence video data promises a cheap and easily accessible data modality. Secondly, multiple worms can

be imaged simultaneously, meaning that interactions between specimens, group behaviours, or the variation in the response of individual specimens to complex stimuli can be recorded and analysed. No additional manual labelling or annotation is required and the observations are high-fidelity with little experimental tuning. Finally, we suggest that the body pose information is complementary to calcium fluorescence imaging, and can be leveraged to provide a degree of information on neurons only distantly connected to observed neurons. We therefore advocate for simultaneously recording body pose data for use in inference.

We first developed a pipeline to convert raw data into inference-amenable skeletons, irrespective of the configuration of the experiment. We also generate a coarse estimate of the neural state. We then use this processed data to recover the posterior distribution over the time-varying latent state of the simulator and global simulator parameters. We performed this estimation on synthetic data, and found (somewhat unsurprisingly) we are able to estimate latent states and parameters with high accuracy. We then applied the technique to real data. We were able to recover parameters that allow the bulk motion of the worm to be recovered. However, poor pose reconstructions and mixing of parameter chains suggest that the model is insufficiently expressive to capture the variation in the data. While these results show promise, the general approach requires more work before this becomes a viable analysis method.

Expanding the search to include more model parameters, and introducing tempered parallel chains to force the Markov chains to mix, are immediate opportunities to develop the method. A further opportunity is developing a more sophisticated likelihood metric. It is well known that Euclidean distance can be a poor similarity metric for high-dimensional and structured variables [Suárez et al., 2018; Wang et al., 2005]. Similarly, we believe that learning a model for perturbing the physical state of the worm more effectively than just perturbing neural state would increase particle diversity and help combat particle degeneracy. Developing or learning more effective likelihood functions and perturbation models is a promising avenue of investigation for improving the quality of the raw particle filter sweeps.

However, we suggest that it is actually the simplified neural model that limits the inference performance. Inspecting Figures 4.10b and 4.11b suggests that the physical model is capable of creating a diverse range of body poses and locomotion envelopes. Hence we suggest that the physical component of the WormSim model is sufficient for our broader aim. While the WormSim neural model is straightforward to implement and understand, it is somewhat

restrictive and is not particularly well suited for inference. The bistable neural elements are frequently "reset" by the transition kernel after being flipped, leading to only minor changes in the evolution of the worm state. The real data was also drawn from "clean" datasets, where the worm sustains forward locomotion at a relatively constant speed, with no sharp turns, reversals etc. Even in highly restricted experiments we were unable to drive the neural state to recreate the complexity of even clean real data. While running more particles or expanding the parameter search space may allow *this* dataset to be effectively reconstructed, it is unlikely that the simplified neural model is able to capture this complexity within a viable computational budget. Therefore, developing a neural model that is more flexible and amenable for inference is likely to dramatically improve performance on real data.

The obvious candidate for a more sophisticated neural model is the full neural model of *C. elegans*, as was used in Chapter 4. As many neurons are not related to locomotion, imputing the entire neural state from just body pose data is likely impossible. Therefore, as we suggested previously, the body pose provides *complementary* information to calcium fluorescence data. Integrating the methods presented in this chapter with the VPC methods presented in Chapter 4 is an obvious next step. Inference is then performed conditioning on both calcium fluorescence and body pose data.

However, we strongly believe that developing low-dimensional, approximate neural models is an important avenue to explore [Archer et al., 2014; Linderman et al., 2019b]. For instance, Kato et al. [2015] perform principal component analysis on calcium fluorescence traces, and find that the principal components are predictive of the locomotion state of the worm (forward, reversal, backward etc). The first step therefore is to develop more tractable, low-dimensional, low-fidelity, potentially even linear, neural models from these high-dimensional and high-fidelity observations with which to drive the locomotion model, or, approximate the full neural model directly. Performing inference in an approximate model is almost certainly more tractable than performing inference in the full neural model, and may actually provide more immediately usable inferences.

An extension of this is that we can further exploit the inverse relationship: the locomotive state is *predictive* of the neural state. Therefore, the results of inference in an approximate, low-fidelity neural model could be used to propose the full neural state of the worm conditioned on the body pose alone, including the state of the neurons not observed by

fluorescence imaging. This more sophisticated proposal can be used to enhance inference in a high-fidelity model, such as the model we used in Chapter 4. Inference in the high-fidelity model can then be performed to achieve the original objective, but also to enhance or refine inferences in the low-fidelity model.

The formalisation of this interplay is *multifidelity* modelling [Fernández-Godino et al., 2016; Peherstorfer et al., 2018; Perdikaris et al., 2017], where low-fidelity models and data (the approximate neural model, body model or locomotion data) are used in conjunction with high-fidelity models and data (the true neural model and calcium fluorescence traces) to improve overall inference performance. We propose that the body model driven by an approximate neural model represents a low-fidelity model that can be analysed to guide inference in the high-fidelity neural model, which is then used to refine inference in the low-fidelity model. These inferences are performed alternately, or, in a more sophisticated fusion-based methodology [Perdikaris et al., 2017]. This allows multiple data modalities to be combined more effectively than simply conditioning a particle filter sweep on more data. It also permits smoothing information (information from future time steps) to be integrated, further increasing efficiency.

The extension of this idea, however, is abstracting the body model entirely from the analysis. In many ways the body pose itself is a nuisance variable. We are not *actually* interested in the temporal dynamics of the non-physiological rods and springs or idealised muscular units. Arguably the body pose is merely a "feature," derived from the neural dynamics and environment that is easily observed and interpreted. Mechanistically, the locomotion model simply provides a sink (muscular activation) and source (proprioceptive feedback) for neural activity. Fully integrating the high-fidelity neural model, or performing multifidelity neuro-phsyical inference, may in fact be a "red herring."

A key observation here is that we were actually able to coarsely, but reliably, estimate the body and muscular state directly from observations. The muscular voltage is driven by known neurons, and proprioceptive feedback flows into known neurons. We therefore propose using a simplified neural model to "preprocess" the body pose, offline from the high-fidelity neural inference. This preprocessing recovers an estimate of the *change* in the activity of the (biological) neurons that interface with the body model required to generate the observed change in muscular activation and body pose. These are therefore effectively

observations of the rate of change of neural potential, and hence can be used to directly condition the neural model, or be leveraged as an additional proposal. This removes the need to perform inference in the body model jointly with the neural model, and allows us to focus on performing inference in just the neural model using the results of this preprocessing. The body model contributes over 90% of the computational cost of iterating the simulator, and hence developing methods that permit relaxing or removing the need to iterate large particle sets at every inference step should be paramount.

A final extension of this idea is exploiting the known inputs and outputs of the body model to actually learn a "neural controller" to generate the arbitrary locomotion patterns. This controller can then be used as an amortised artefact to generate estimates of the neural inputs and outputs for a given locomotion pattern. Crucially, these controllers can be learned using reinforcement learning [Warrington et al., 2020a; Sutton, 1992]. This allows us to access the incredibly powerful and flexible catalogue of RL methods to learn black-box controllers, capable of generating more complex locomotion patterns than it is feasible to create using hand-crafted, low-fidelity neural models. We can then leverage these controllers to act as automated locomotion preprocessors, additional proposal mechanisms, or feature generators, to enhance high-fidelity inference in the neural model.

Although the work presented in this chapter and in Chapter 4 are necessary first steps, the particular approach taken here may be fundamentally limited, especially when scaled to consider more parameters. As a result, more sophisticated inference methods, or even bespoke, model-specific approaches, such as the innovations discussed in this section, are inevitably required to facilitate the analysis discussed in Chapter 1. We return to discuss this further in Chapter 8.

6

Inducting Model Structure

In this chapter we address the second extension identified in the virtual patch clamp work. Until now, we have assumed that the structure of the model in which we perform inference is fixed, and that model selection reduces to simply learning the scalar parameter values of the model. However, we *know* this model is incomplete, since we consider the synaptic weights to be constant throughout the simulation. It is well known that synaptic weights change over time, modifying the low-level neural dynamics. Many of the most sophisticated organism behaviours, such as learning and memory, are as a direct result of this modification [Benfenati, 2007; Domjan, 2015; Hebb, 1949; Markram et al., 2012].

Attempting to quantify this adaptation on different levels is a perennial research topic in both theoretical and experimental neuroscience [Berlucchi & Buchtel, 2009; Gerstner, 2017; Song et al., 2000]. Previous work includes developing low-level models of plasticity derived from rewards or biological constraints [Jordan et al., 2021], developing high-level models of behaviour and adaptation during training [Ashwood et al., 2020; Roy et al., 2018], and even searching for biologically plausible learning rules for use in artificial neural networks [Bengio et al., 1992, 2015]. In this chapter we consider low-level activity-dependent synaptic adaptation rules [Dayan & Abbott, 2001]. Many different mathematical models describing this adaptation have been proposed, including the foundational Hebbian learning model [Hebb, 1949], models that enforce certain inductive principles such as Oja's rule [Oja, 1982], updates that introduce additional state variables such as the Bienenstock-Cooper-Munro rule [Bienenstock et al., 1982], and the wealth of spike-timing dependent rules [Dayan & Abbott, 2001].

Crucially, all activity-dependent weight update rules are representable as mathematical functions conditioned on the current neural state, and hence can be implemented as programmatic functions that operate on the state of a neural simulator. We can therefore construct and integrate different learning rules through the specification of source code. We can then consider the source code that defines the learning rule as a parameter that we can

learn from data, selecting the source code, and hence model, that best explains the observed change in activity. Learning the structure of the model from data is broadly referred to as *model selection* [Linhart & Zucchini, 1986; Wasserman et al., 2000].

These programmatic learning rules are low-level, mechanistic and interpretable generative models of the learning process. This is a fundamentally different approach to simply correlating the change in the emergent behaviour with the stimulus, or, the change in the distribution of activity in response to stimulus [Lim et al., 2015]. Beyond this, we can consider automatically proposing entirely new models of neural adaptation using powerful program synthesis, model synthesis and symbolic regression techniques [Billard & Diday, 2002; David & Kroening, 2017; Gulwani et al., 2017; Drton & Maathuis, 2017; Udrescu & Tegmark, 2020]. This approach allows new models of adaptation to be automatically proposed and tested without input from an expert.

In this chapter we investigate jointly performing model selection, continuous parameter learning and latent state imputation from calcium fluorescence traces. We begin by introducing a flexible learning rule class, and a set of programmatic operations and composition rules, that together define a broad set of learning rules through the specification of source code. We then define a generative procedure for sampling new rules, that can then be automatically inserted into the neural model. We then compute an estimate of the model evidence (albeit under heavy restrictions on the model class) that is differentiable with respect to the continuous parameter values of both the neural model and the learning rule. We then use stochastic gradient ascent to maximise the model evidence for a given rule. We first perform this for each rule in a predefined set of learning rules, before performing this on a set of randomly generated learning rules. We demonstrate on synthetic data that we are able to jointly recover the correct learning rule and continuous model parameters. We conclude by discussing a number of extensions to this work.

The aim of this work is twofold. Firstly, we wish to explore a different family of techniques and formulations to the ones introduced previously. Secondly, and most importantly, we wish to give a demonstration of how ideas drawn from cutting edge machine learning, programming languages and statistics research can combine to achieve one of our original objectives: automating neuroscience research. Although we employ only basic techniques and only present preliminary results on synthetic data, we believe this is

a powerful and informative demonstration of the scope and opportunity provided by this general principle, above and beyond the examples we have already explored. The best-case outcome of this work, similar to that of the VPC, is that data could be gathered and provided to an inference pipeline that automatically generates a set of optimised parameter values and feasible learning rules, expressed as program source code, directly from data. These learning rules can then be inspected by domain experts to qualitatively compare, contrast and understand their function. More generally than considering only learning rules, automated recovery of structured models or programs from data to explain and reproduce observed behaviours would represent a huge step towards automating neuroscience research.

6.1 Problem Formulation

In this section we formally define the estimation task and the nomenclature used. In subsequent sections we place additional restrictions and constraints on the model class we analyse. We seek to recover the synaptic update rule from data by jointly performing model selection with parameter and latent state estimation. On a high level, we consider the learning rule structure as a categorical variable, which in turn defines a continuous learning task. For each categorical learning rule structure, we must estimate the continuous parameter values of the neural simulator and learning rule, conditioned on the specified learning rule, by maximising the model evidence over parameters. The learning rule and parameter pair that yields the highest model evidence is returned as the optimal solution.

6.1.1 Model Definition

We define the learning rule as $h \in \mathcal{H}$, where the space of learning rules, $\mathcal{H}$, contains all permissible learning rules. However, as we eluded to above, we can separate learning rules into a categorical component, defining the structure of the rule, and a continuous component, defining the scalar-valued parameters of the rule. The structure of the rule defines the parameter space, i.e. the number and type of parameters is dependent on the learning rule itself. We therefore define the categorical structure of the learning rule as $L \in \mathcal{L}$, and the continuous parameter values as $\boldsymbol{\phi}_L \in \boldsymbol{\Phi}_L$, such that the set of all categorical forms and continuous parameter values is equal to $\mathcal{H}$. This can be notationally awkward in places, and so often we will denote the overall learning rule as a pair $\left(L, \boldsymbol{\phi}_L\right)$.

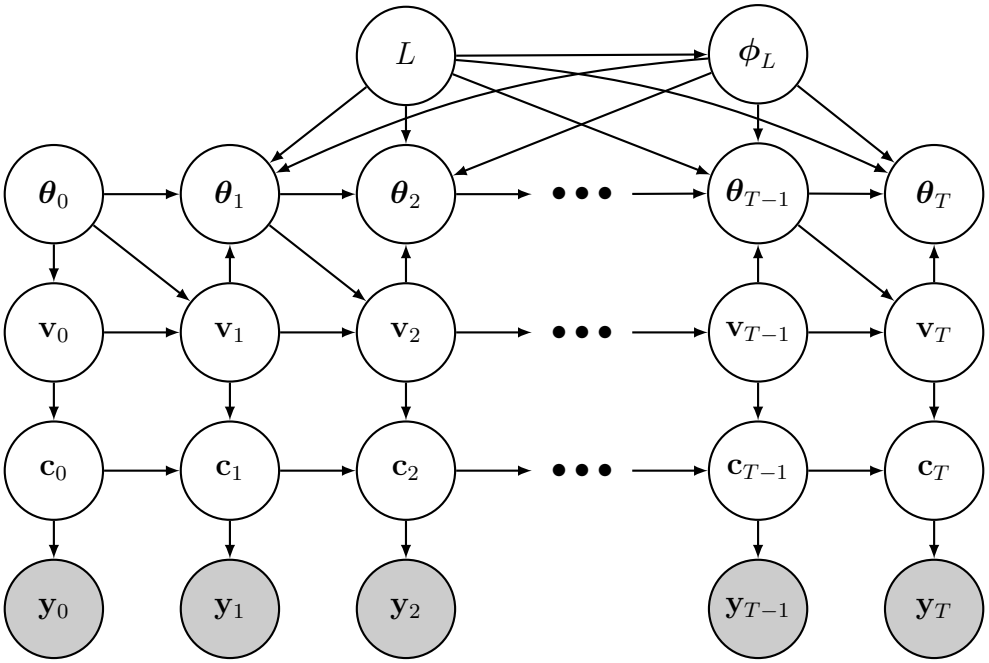


Figure 6.1: Graphical model representing the model in which we perform inference. The membrane potential at the next timestep, $\mathbf{v}_t$, depends on the previous connectome weights, $\boldsymbol{\theta}_{t-1}$, and previous neural potential, $\mathbf{v}_{t-1}$. The updated connectome weights, $\boldsymbol{\theta}_t$, are then dependent on the previous connectome weights, $\boldsymbol{\theta}_{t-1}$, the learning rule, L, the parameters of the learning rule, $\boldsymbol{\phi}_L$, and the current neural potential. The intracellular calcium concentration, $\mathbf{c}_t$, depends on the neural potential and the previous calcium concentration. The observed fluorescence, $\mathbf{y}_t$, then depends on the intracellular calcium. In the experiments we present in Section 6.3 we assume the transitions are deterministic such that we can readily use backpropagation, and hence all nodes depend deterministically on L, $\boldsymbol{\phi}_L$, $\boldsymbol{\theta}_0$, $\mathbf{v}_0$ and $\mathbf{c}_0$. We assume the existence of a zeroth observation, $\mathbf{y}_0$, to initialise the latent states. This observation is not used in the computation of the model evidence.

The model we analyse is then defined as a parametric HMM with three layers of hidden state, shown in Figure 6.1. The global parameters here are the learning rule structure, L, and the static global parameters of the learning rule, $\boldsymbol{\phi}_L$. The first layer is the time-varying neural parameters of the simulator, $\boldsymbol{\theta}_t$. We assumed that these parameters were static in Chapter 4. The second layer of variables is the membrane potential, $\mathbf{v}_t \in \mathbb{R}^N$, and the lowest level is the intracellular calcium concentration, $\mathbf{c}_t \in \mathbb{R}^N_{\geq 0}$, where $N \in \mathbb{Z}_+$ is the number of neurons being modelled. We then observe a fluorescence signal, $\mathbf{y}_t \in \mathbb{R}^M$, conditioned on the calcium concentration, where $M \leq N$ denotes the number of observed neurons, and $\mathbf{A} \in \{0,1\}^{M \times N}$ is a (potentially non-square) permutation matrix mapping from neurons to observations as in (4.5). We assume $\mathbf{A}$ is fixed and known.

We define the potential values and calcium concentration as the *neural state*, denoted $\mathbf{x}_t \in \mathcal{X} = \mathbb{R}^{2 \times N}$. We denote the neural dynamics simulator as $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1}, \boldsymbol{\theta}_{t-1})$, where $\mathbf{x}_{t-1}$ denotes the previous neural state, $\boldsymbol{\theta}_{t-1}$ denotes the instantaneous connectome weights. The

time-evolution of the neural state is modelled using the Wicks neural dynamics model [Wicks et al., 1996] and calcium model [Rahmati et al., 2016], introduced in Section 2.6, with the mathematical forms given by (2.8) and (2.16).

To this model we add synaptic plasticity. We introduced in Section 2.3 that synaptic strength is determined by factors such as the density of ion channels, the number of available vesicles and the probability of vesicle release, all of which can change under plasticity. We denote the instantaneous synaptic strengths as $\boldsymbol{\theta}_t \in \boldsymbol{\Theta} = \mathbb{R}_{\geq 0}^{N \times N \times 2}$, defining two square matrices of chemical and electrical synapse strengths. These matrices are accompanied by a fixed binary connectivity matrix , $\mathbf{C} \in \{0,1\}^{N \times N \times 2}$, indicating which neurons are connected and hence defining the subset of $\boldsymbol{\theta}$ that are variable (the remaining parameters are pinned to zero). The application of the learning rule is denoted $\boldsymbol{\theta}_t \leftarrow g_L(\mathbf{x}_t, \boldsymbol{\theta}_{t-1}, \boldsymbol{\phi}_L)$, where L indicates the specific form of the learning rule. The full time-varying latent state of the simulator is therefore $\boldsymbol{\Theta} \times \mathcal{X}$. We keep these definitions separate for notational ease later. We assume the static neural parameter values (e.g. conductances, membrane resistances) to be fixed and known, and hence we do not notate them here.

We perform the estimation conditioned on calcium fluorescence data as we did in Chapter 4. We denote the observed data $\mathbf{y} = \mathbf{y}_{1:T} \in \mathcal{Y} = \mathbb{R}_{\geq 0}^{T \times M}$, where the trace is of length $T \in \mathbb{Z}_+$. For this work we assume that the fluorescence of all neurons is observed, $M = N$. We denote the emission distribution as $p(\mathbf{y}_t | \mathbf{x}_t)$.

6.1.2 Learning Objective

We initially frame learning as MMAP estimation (Section 3.2), recovering the MAP parameter values while marginalising over the value of the latent variables. The parameters we wish to learn are the initial connectome weights, $\boldsymbol{\theta}_0$, learning rule and learning rule dependent parameters, $\left(L, \boldsymbol{\phi}_L\right)$, and the initial neural state, $\mathbf{x}_0$:

$$L^*, \boldsymbol{\phi}_{L^*}^*, \boldsymbol{\theta}_0^*, \mathbf{x}_0^* = \arg\max_{L \in \mathcal{L}} \left[\arg\max_{\boldsymbol{\phi}_L \in \boldsymbol{\Phi}_L,\, \boldsymbol{\theta}_0 \in \boldsymbol{\Theta},\, \mathbf{x}_0 \in \mathcal{X}} p(L, \boldsymbol{\phi}_L, \boldsymbol{\theta}_0, \mathbf{x}_0 | \mathbf{y}) \right]. \tag{6.1}$$

Foreshadowing somewhat, we also include the initial neural state in the optimisation task. The posterior density can be computed up to a normalisation constant as:

$$p(L, \boldsymbol{\phi}_L, \boldsymbol{\theta}_0, \mathbf{x}_0|\mathbf{y}) \propto \int_{\mathbf{x}\in\mathcal{X}} p(\mathbf{y}|\mathbf{x}_{1:T}) p(\mathbf{x}_{1:T}|\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L) \mathrm{d}\mathbf{x}_{1:T}, \tag{6.2}$$

$$= \int_{\mathbf{x}\in\mathcal{X}} p(\mathbf{y}|\mathbf{x}_{1:T}) p(\mathbf{x}_{1:T}|\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L) \mathrm{d}\mathbf{x}_{1:T} p(L, \boldsymbol{\phi}_L, \boldsymbol{\theta}_0, \mathbf{x}_0), \tag{6.3}$$

$$= \int_{\mathbf{x}\in\mathcal{X}} p(\mathbf{y}|\mathbf{x}_{1:T}) p(\mathbf{x}_{1:T}|\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L) \mathrm{d}\mathbf{x}_{1:T} \times$$
$$p(\mathbf{x}_0|\boldsymbol{\phi}_L, L, \boldsymbol{\theta}_0) p(\boldsymbol{\phi}_L|L) p(L) p(\boldsymbol{\theta}_0). \tag{6.4}$$

We have assumed here that the initial neural parameters are independent of the learning rule and learning rule parameters, and that the distribution over initial states is conditional on all parameters. We consider optimising the learning rule structure, continuous parameter values and initial conditions here. This is as opposed to the more inference-focused approach we took in Chapters 4 and 5. While it is possible to average over categorical model structures in posterior predictive inference, it is a more natural formulation to search for the best learning rule. Furthermore, optimising parameters through a marginalisation requires computation of high-variance REINFORCE gradients [Williams, 1992]. This also allows us to explore alternative learning techniques to what was explored previously.

6.2 Methods

We now introduce the specifics of the approach we use here. We begin by introducing the neural model we use, adapted from the Wicks model introduced in Section 2.6. We then define the class of learning rules we considered, and use this definition to create a generative model of learning rules through the procedural generation of program source code. We conclude by discussing optimisation and model selection.

6.2.1 Static Neural Model

For this experiment we use the idealised tap withdrawal circuit introduced in Section 2.6, originally proposed by Wicks et al. [1996]. This is an assembly of nine idealised neurons responsible for the neural processing behind the tap withdrawal response. We simulate the circuit using the neural, calcium and fluorescence models first introduced in Section 2.6, used in Chapter 4, and denoted here as f. The resulting neural activity for the basic neural circuit is shown in Figure 6.2. We assume the fluorescence of all neurons are observed, and we use an independent Gaussian likelihood over fluorescence.

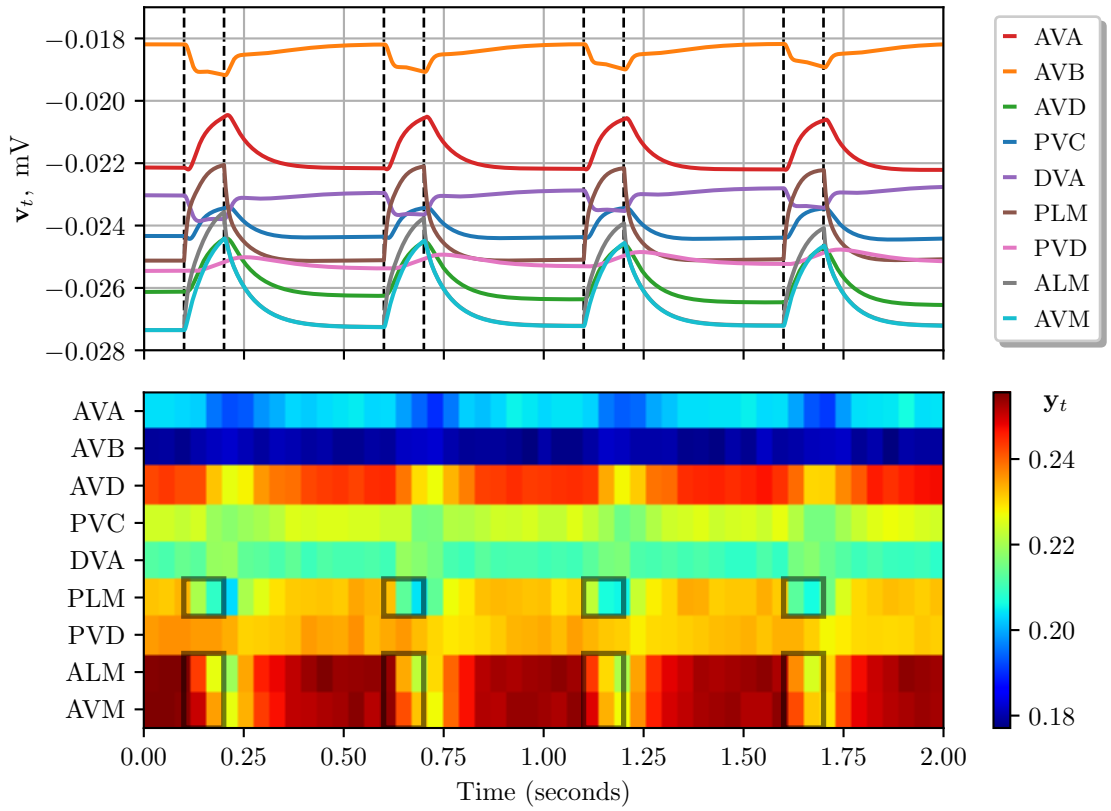


Figure 6.2: Latent potential trajectories and observed fluorescence signals for the model introduced in Section 6.2. The top row shows the latent membrane potentials, $\mathbf{v}_{0:T}$, of the nine idealised neurons of the tap withdrawal circuit, using GHL for both chemical and electrical synapses. We apply a stimulus of 2.5pA to `AVM`, `ALM` and `PLM` for 0.1 seconds [Wicks et al., 1996] (indicated by dashed bands). The changes in neural dynamics are most visible in `DVA` and `AVD`. The simulation length is short and so there is only a minor change in the neural dynamics. The bottom row then shows the raw observed fluorescence signal, $\mathbf{y}$. The stimulated neurons and the stimulus times are indicated by black boxes.

We consider in this chapter the specific case where both neural simulator, f, and learning rule, L, are deterministic functions. As a result the simulated traces are *highly* dependent on the initial time-varying states, $\boldsymbol{\theta}_0$ and $\mathbf{x}_0$. We therefore define the convenient shorthand $\mathbf{x}_{1:T} \leftarrow F(\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L)$. As we assumed deterministic transition functions, the integral in (6.4) simplifies to:

$$p(L, \boldsymbol{\phi}_L, \boldsymbol{\theta}_0, \mathbf{x}_0|\mathbf{y}) \propto p(\mathbf{y}|F(\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L))p(\mathbf{x}_0|\boldsymbol{\phi}_L, L, \boldsymbol{\theta}_0)p(\boldsymbol{\phi}_L|L)p(L)p(\boldsymbol{\theta}_0). \quad (6.5)$$

Crucially, if F is differentiable, this expression is differentiable with respect to $\mathbf{x}_0$, $\boldsymbol{\theta}_0$ and $\boldsymbol{\phi}_L$. We can therefore use gradient descent to learn these parameters for a given learning rule structure L. We implement the model in PyTorch [Paszke et al., 2017] such that this gradient can be computed using automatic differentiation.

6.2.2 Defining Learning Rules

To the static neural model we add functionality for updating the synapse weights dependent on the neural state. We denote this update as $\boldsymbol{\theta}_t \leftarrow g_L(\mathbf{x}_t, \boldsymbol{\theta}_{t-1}, \boldsymbol{\phi}_L)$. This also defines the basic interface to the learning rule as accepting the neural state, $\mathbf{x}_t$, current connectome weights, $\boldsymbol{\theta}_{t-1}$, and any continuously valued parameters of the learning rule, $\boldsymbol{\phi}_L$. We consider the most general case where both electrical and chemical synapses undergo adaptation. Therefore, L decomposes further into two adaptation rules, L^{gap} and L^{chem}. We do not assume these adaptation rules are the same and allow them to have different parameter values. However, we do assume that these rules use the same interface. Therefore, for convenience, we will not be specific and refer simply to L, noting that this actually defines a pair of learning rules with their respective parametrisations and values. We now demonstrate how two common learning rules can be expressed using this interface, and also define the base behaviour.

The Hebbian learning update [Hebb, 1949] is defined in most often in terms of spike rates, $\mathbf{u}_t \in \mathbb{R}^N_{\geq 0}$:

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = \phi_\tau \mathbf{C} \odot (\mathbf{u}_t \otimes \mathbf{u}_t), \tag{6.6}$$

where $\otimes$ denotes a vector outer product, $\odot$ denotes an element-wise vector or matrix product, and $\phi_\tau \in \mathbb{R}_{\geq 0}$ is the *learning rate*. We modify this definition for graded neurons in what we refer to as *graded Hebbian learning* (GHL):

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = \phi_\tau \mathbf{C} \odot (\bar{\mathbf{v}}_t \otimes \bar{\mathbf{v}}_t), \tag{6.7}$$

where we use the deviation of the neural potential from equilibrium, $\bar{\mathbf{v}}_t = \mathbf{v}_t - \mathbf{v}_{\text{eq}}$. We assume access to this normalised potential, $\bar{\mathbf{v}}_t$, as a pre-computed primitive, instead of the absolute potential, $\mathbf{v}_t$, herein. This expression retains much of the behaviour of Hebbian learning. If neurons are simultaneously above or below their respective equilibrium voltages the connection strength between those neurons increases. If the neurons are of opposite polarities, the connection strength decreases. Representing GHL programmatically:

```
def GHL(vbar, theta, phi):
    d_theta = torch.mul(phi.tau, torch.mul(vbar, vbar.unsqueeze(-1))
    return theta + torch.mul(phi.dt, torch.mul(phi.C, d_theta))
```

where we denote the autodifferentiation library PyTorch [Paszke et al., 2017] as `torch`, and `unsqueeze(-1)` is an operator that adds a dimension to the vector, allowing outer products to be computed. The variable parameters for each junction type are $\boldsymbol{\phi}_{\text{GHL}} = \{\phi_\tau\} \in \mathbb{R}_+$.

For convenience we also assume additional constants, such as the connectivity pattern and integration timestep, are programmatically stored in $\boldsymbol{\phi}$.

To extend this to Oja's rule [Oja, 1982] we add a term regularising the weights:

$$\frac{\mathrm{d}\boldsymbol{\theta}_t}{\mathrm{d}t} = \mathbf{C} \odot \left(\phi_\tau \bar{\mathbf{v}}_t \otimes \bar{\mathbf{v}}_t - \phi_\alpha \bar{\mathbf{V}}_t \odot \boldsymbol{\theta}_t \right), \tag{6.8}$$

$$\bar{\mathbf{V}}_t = \begin{bmatrix} (\bar{\mathbf{v}}_t \odot \bar{\mathbf{v}}_t)^T \\ \dots \\ (\bar{\mathbf{v}}_t \odot \bar{\mathbf{v}}_t)^T \end{bmatrix} \tag{6.9}$$

where $\bar{\mathbf{V}}_t$ has N rows. This is expressed more succinctly programmatically:

```
def GOL(vbar, theta, phi):
    d_theta = torch.mul(phi.tau, torch.mul(vbar, vbar.unsqueeze(-1))
    reg     = torch.mul(phi.alpha,
                        (torch.mul(torch.mul(vbar, vbar), theta)).T)
    return theta + torch.mul(phi.dt, torch.mul(phi.C, d_theta - reg))
```

where this *graded Oja's rule* (GOL) retains the same inductive biases of the original Oja's rule. The parameters of (graded) Oja's rule are defined as $\boldsymbol{\phi}_{\mathrm{GOL}} = \{\phi_\tau, \phi_\alpha\} \in \mathbb{R}^2_{\geq 0}$.

We also define the base behaviour corresponding to no learning (NL), where weights are fixed, $\mathrm{d}\boldsymbol{\theta}_t/\mathrm{d}t = 0$. This is easily represented programatically as:

```
def NL(vbar, theta, phi):
    return theta
```

This means that the model class with learning is a superset of the model class without learning. In the case of no learning existing $\boldsymbol{\phi}_{\mathrm{NL}} = \emptyset$.

We have demonstrated that the range of learning rules that can be implemented under this interface, and we will use this interface going forwards. Crucially, as NL, GHL and GOL all use the same function signature, switching which learning rule is used at runtime is straightforward.

6.2.3 Rule Generation

We have defined the interface of the learning rule and demonstrated how different learning rules can be implemented under this interface. We now discuss how we construct the functional form and programmatic definition of the learning rule using ideas from in programming languages, program induction and symbolic regression [Billard & Diday, 2002; Kitzelmann, 2010]. At the core of this process is the *context free grammar* (CFG) [Chomsky, 1956; Saad et al., 2019]. The context free grammar is a formal definition of how one or more operators and operands can be composed to create a second operator (or operand)

that is also valid. Context free indicates that each level in the recursion can be considered independently from the other levels (i.e. there is no mutual exclusivity) and that there is exactly one return value from each operation. More complex operations can then be created by composing multiple definitions together. We can then use this definition to construct a generative model for the form of the update rule. While the full complexities of CFGs are beyond the scope of this thesis, we introduce the core principles here.

A CFG is defined as a four-tuple, $G = (\mathcal{V}, \Sigma, \mathcal{R}, \mathcal{S})$. The set $\mathcal{V}$ is the set of permissible non-terminal characters and Σ is the set of terminal characters. Non-terminal characters must be replaced by terminals from the set Σ during construction. An example of a terminal is a variable. The set $\mathcal{R}$ represents a set of *productions*, which are the rules defining how non-terminal symbols are replaced with terminal signals. Finally, $\mathcal{S}$ is the starting expression. The expression is then defined by starting with $\mathcal{S}$ and recursively replacing non-terminals with terminals or further non-terminals according to the production rules, $\mathcal{R}$, until no non-terminals remain.

This definition is rather theoretical, and is most easily understood by demonstration. We therefore introduce the grammar we use. We use three non-terminals, $<\texttt{param}>$, $<\texttt{prim}>$, and $<\texttt{dim}>$. The first non-terminal indicates that a parameter is to be inserted, where this parameter is automatically added to ϕ_L. The second term indicates that an operation or state will be inserted. The final non-terminal indicates that a zero-indexed integer dimension of a tensor must be substituted. The initial symbol we use is $\mathcal{S} = \texttt{phi.dt} * \texttt{torch.mul}(\texttt{phi.C}, \texttt{torch.mul}(<\texttt{param}>, <\texttt{prim}>))$. This initial symbol stipulates that the gradient of the neural connection weights can only be non-zero where there is a connection, scales the derivative according to the integration timestep, and automatically inserts a scalar learning rate that will scale the gradient. The terminal operators and production rules are then defined as:

```
<prim>  -> <param> | (<prim>) | (<prim>).T | vbar | theta |
           torch.unsqueeze(<prim>, 1) | torch.unsqueeze(<prim>, -1) |
           torch.add(<prim>, <prim>) | -(<prim>) | torch.exp(<prim>) |
           torch.sum(<prim>,<dimension>) | torch.mul(<prim>, <prim>) |
           torch.mul((<prim>).unsqueeze(1), (<prim>).unsqueeze(-1))
<param> -> torch.Variable(torch.tensor([z]), requires_grad=True)
           torch.exp(torch.Variable(torch.tensor([z]), requires_grad=True))
<dim>   -> 0 | 1 | 2 | 3
```

The first row defines that a parameter, bracketed expression, transpose of an expression, normalised voltage or parameter value are all valid expressions. The next line defines unsqueeze operations, allowing for manipulation of the dimensions of variables. The third, fourth and fifth lines define the basic mathematical operations that are allowed. The next two lines state that a parameter *must* be a fresh tracked variable, and is defined as either a real number or a strictly positive number. The final line defines the allowable dimensions. This defines the CFG we use. All of GHL, GOL and NL can be defined using this grammar.

We are also able to exploit this grammar to define a procedure for generating random learning rules. To produce a rule, we randomly select a term from the list above to replace each non-terminal in the current string. When replacing `<prim>` we select `vbar` with 40% probability, instantiate a fresh variable with 10% probability, `theta` with 10% probability, and select an operation (the remaining terms above) with 40% probability, where the specific operation is drawn from a uniform distribution. We also define a maximum recursion depth, where the number of recursive $<$ `prim` $>$ calls is limited to a depth of three. Once this depth is reached, states, variables or parameters must be inserted. This procedure, denoted as `GEN_RULE()`, returns the string encoding of the rule. Any fresh variables are automatically added to the tracked variables, ϕ_L, during the call to `GEN_RULE()`. The rule is then instantiated in the class straightforwardly using a `lambda` member function:

$$\texttt{self.g\_L} = \texttt{eval}(\texttt{'lambda vbar, theta, phi : \{0\}'.format(GEN\_RULE())})$$

This rule is then inscribed into the class as a function such that it can be subsequently called. The variables `vbar`, `theta` and `phi` are assumed to be addressable as local variables in the calling context. We exploit the fact that we can define fresh variables in both the log domain and linear domain, and have a negation operator, and hence we initialise learning rule parameters from $\mathcal{N}(1, 1)$. We see that the exponential variables can change scales to very large and very small parameter values. We note that many generated rules are not logically or algebraically valid learning rules. The learning rule must also return an $N \times N$ matrix. We therefore test the compilation and return type by testing each generated learning with synthetic data. We instantly reject any learning rules that do not define valid mathematical operations or return a value of the wrong dimensions. Using this process we are able to automatically define a range of valid learning rules as functions of the state, where the static,

learnable parameters of these rules are automatically handled by the autodifferentiation package. Examples of rules generated using this procedure are shown in Figure 6.5a.

6.2.4 Parameter Estimation & Rule Scoring

We now introduce the details of how we estimate the model evidence, learn the continuous parameter values, and finally, select the best performing learning rule. We first express the objective in (6.5) in logarithmic space:

$$\log p(L, \boldsymbol{\phi}_L, \boldsymbol{\theta}_0, \mathbf{x}_0|\mathbf{y}) \propto \log p(\mathbf{y}|F(\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L)) + \log p(\mathbf{x}_0|\boldsymbol{\phi}_L, L, \boldsymbol{\theta}_0) + \log p(\boldsymbol{\phi}_L|L) + \log p(L) + \log p(\boldsymbol{\theta}_0). \tag{6.10}$$

The first term corresponds to the likelihood of the parameters given the observed data. This can be easily computed by generating the latent states and evaluating the likelihood at each of these points:

$$\log p(\mathbf{y}|F(\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L)) = \sum_{t=1}^{T} \log p(\mathbf{y}_t|\mathbf{x}_t), \quad \text{where} \quad \mathbf{x}_{1:T} \leftarrow F(\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L). \tag{6.11}$$

The remaining terms then correspond to the prior probabilities of the static variables. We assume a uniform prior over learning rules, L, and the static parameters of the learning rules, $\boldsymbol{\phi}_L$. We then assume a Gaussian distribution over the initial connectome weights.

As the model is deterministic, the simulated traces are heavily dependent on the initialisation of the neural state. We therefore first investigated optimising the initial neural state as a parameter. Although this yielded reasonable estimates of the underlying state, it added additional complexity to the optimisation. We therefore investigated using a simple data-driven pipeline for estimating the initial neural conditions, assuming access to a zeroth observation, $\mathbf{y}_0$. We initialise the calcium concentration by importance sampling under the zeroth observation, where we set $\mathbf{c}_0$ to the concentration with the highest probability. As the calcium dynamics model is a derivative model, we cannot use importance sampling to initialise the neural potential. Therefore, we estimate the initial voltage conditioned on the zeroth observation by fitting a linear function mapping from observations to voltages to a corpus of data simulated using the *current* model. This allows us to create a point estimate of the voltage given an observation under the expected dynamics of the current model, and estimate the maximiser of $p(\mathbf{x}_0|\boldsymbol{\phi}_L, L, \boldsymbol{\theta}_0, \mathbf{y}_0)$. We initialise the neural potential to this value. This zeroth observation then discarded and is not used in the evidence calculation. We find empirically that this approach can estimate the initial latent neural state quickly, accurately

and without adding additional complexity to the optimisation. We therefore use this pipeline herein to estimate $\mathbf{x}_0$ given $\boldsymbol{\phi}_L, L, \boldsymbol{\theta}_0, \mathbf{y}_0$, removing $\mathbf{x}_0$ from the optimisation.

We can therefore compute the gradient of the remaining continuous parameter values as:

$$\nabla_{\boldsymbol{\phi}_L} p(L, \boldsymbol{\phi}_L, \boldsymbol{\theta}_0, \mathbf{x}_0|\mathbf{y}) = \nabla_{\boldsymbol{\phi}_L} \log p(\mathbf{y}|F(\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L)), \tag{6.12}$$

$$\nabla_{\boldsymbol{\theta}_0} p(L, \boldsymbol{\phi}_L, \boldsymbol{\theta}_0, \mathbf{x}_0|\mathbf{y}) = \nabla_{\boldsymbol{\theta}_0} \log p(\mathbf{y}|F(\mathbf{x}_0, \boldsymbol{\theta}_0, L, \boldsymbol{\phi}_L)) + \nabla_{\boldsymbol{\theta}_0} p(\boldsymbol{\theta}_0), \tag{6.13}$$

such that for a fixed learning rule we can learn $\boldsymbol{\theta}_0$ and $\boldsymbol{\phi}_L$ using stochastic gradient descent. For each learning rule defined, or for a fixed number of candidate learning rules sampled from the procedure outlined above, we take a finite number of gradient descent steps per rule, or, until the evidence saturates, and return the model and parameter set with the highest model evidence. Although randomly sampling potential learning rules, for even modestly sized CFGs, is not a scalable approach, it serves as a simple proof-of-principal. Furthermore, we can distribute computing the evidence to worker nodes as it is an embarrassingly parallelisable operation. We discuss more complex procedures in Section 6.4.

6.2.5 Rule Screening

An interesting component of this problem is the equivalence of programmatic forms of learning rules. For instance, the term $(\texttt{theta.T}).\texttt{T}$ is mathematically identical to `theta`, even though its programmatic form is different. Furthermore, learning rules may be supersets of other learning rules. For instance, $\texttt{torch.mul}(\texttt{phi.tau}, \texttt{theta})$ is a superset of $\texttt{torch.add}(\texttt{theta}, \texttt{theta})$, as setting $\texttt{phi.tau} = 2$ yields the same function, but any other value of `phi.tau` yields a different function. Therefore, one may wish to reject rules that have already been tested or are subsets of rules that have already been tested to avoid wasting computational effort. However, re-testing rules is equivalent to a random restart, known to be beneficial in non-convex optimisation, and hence repeating some optimisations may actually be beneficial.

We experimented with two approaches for screening rules. We first perform direct string comparison between two learning rules. If the strings are identical then the rules are clearly identical. If the strings are different, we then tested comparing the functional equivalence of the rules. To do this, we defined a number of learning rule operating points ($\boldsymbol{\theta}_t$ and $\mathbf{v}_t$ values) and applied the new learning rule to each of these points. We then optimise all *previous* learning rules to minimise the difference in the synaptic update. If the previous rule

can perfectly reproduce the update then the rule is either equivalent or is a superset. This approach could reliably find duplicate rules for very simple rules, but performed unreliably for larger rules. We therefore use just string comparison, and defer development of more sophisticated methods to future work.

6.3 Experiments

We now present a number of different experiments on synthetic data. We begin by simply recovering the initial connectome weights and learning rule parameters using gradient descent. We then perform model selection from a finite set of candidate learning rules. We conclude by recovering the learning rule via full induction.

6.3.1 Learning Using Gradient Descent

The first experiment we conduct is ensuring that we are able to learn static parameters using gradient descent. We conduct two experiments, recovering the electrical and chemical connectome weights separately. We apply a current burst to three of the sensory neurons to simulate a tap stimulus, in accordance with Wicks et al. [1996]. We apply a stimulus of 2.5pA to `AVM`, `ALM` and `PLM` for 0.1 seconds, every 0.5 seconds. The results of recovering the electrical connectome weights are shown in Figure 6.3. We see that we are able to accurately recover the gap junction weights from the observed calcium fluorescence quickly, and that the estimated latent state trajectory is almost perfect.

We then repeat the experiment for chemical synapse weights. However, we were unable to effectively recover the initial chemical synapse weights. The simulated traces are close to the real traces and the optimised model evidence is close to the true model evidence. However, there is little concentration in the synaptic weights across experimental repeats. This suggests there are multiple local minima that the optimisation falls into, or, that the model is less sensitive to a number of the chemical synapses and a continuum of solutions are possible. This is an example of where solving for the full posterior distribution, instead of solving for just the pointwise maximiser, would be instructive. We therefore assume the chemical synapse weights are known, and continue investigating our originally outlined goals learning just the gap junction weights. We return to discuss this in Section 6.4.

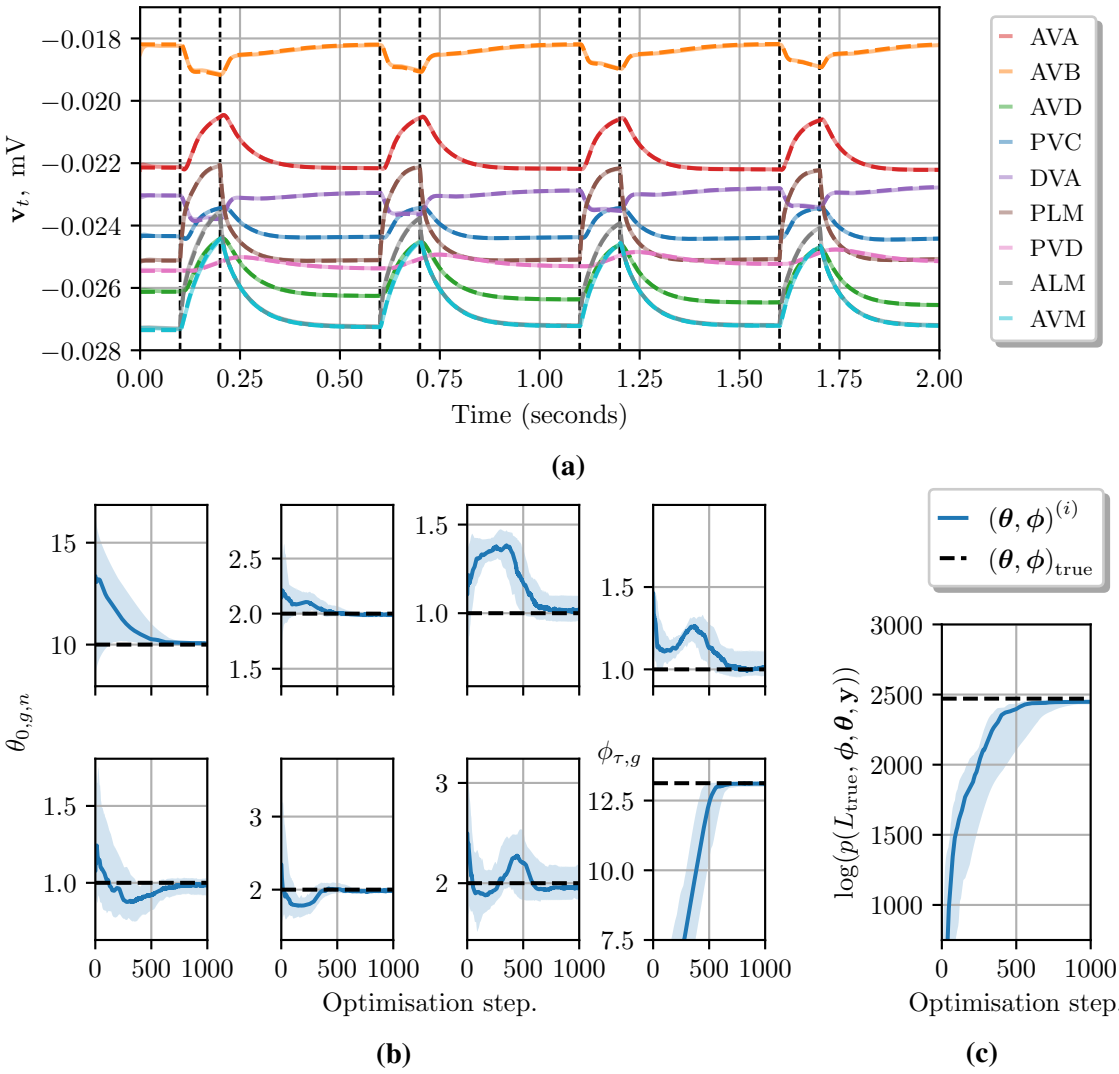


Figure 6.3: Recovery of gap junction weights, $\boldsymbol{\theta} = \mathbf{w}_{\text{g},0}$, using backpropagation, when using the correct learning rule, for the experiment introduced in Section 6.3.1. Figure 6.3a then shows the true neural traces as dashed lines and the reconstruction using the optimised parameters as solid lines. Figures 6.3b and 6.3c confirms that we are able to quickly and accurately estimate the gap junction strengths from calcium fluorescence traces and that the learned parameters faithfully replicate the observed data, indicated by the optimised posterior density matching that of the true parameters. Shown are the median and quartiles across fifty random restarts.

6.3.2 Model Selection

We now introduce also recovering learning rules. For this, we perform a typical model selection experiment, choosing the best-performing candidate from a finite set of possibilities, i.e. $\mathcal{L}$ is set to a finite and pre-specified set. Specifically, we use $L = \left(L^{\text{chem}}, L^{\text{gap}}\right) \in \mathcal{L} = \{(\text{NL}, \text{NL}), (\text{GHL}, \text{NL}), (\text{NL}, \text{GHL}), (\text{GHL}, \text{GHL})\}$, where the first term is the learning rule of the chemical synapses and the second term is the learning rule of the electrical junctions. Each GHL is parameterised by a unique learning rate that must be learned, and hence the parametrisations of the candidate rules are defined as $\boldsymbol{\Phi}_{(\text{NL},\text{NL})} = \emptyset$, $\boldsymbol{\Phi}_{(\text{GHL},\text{NL})} =$

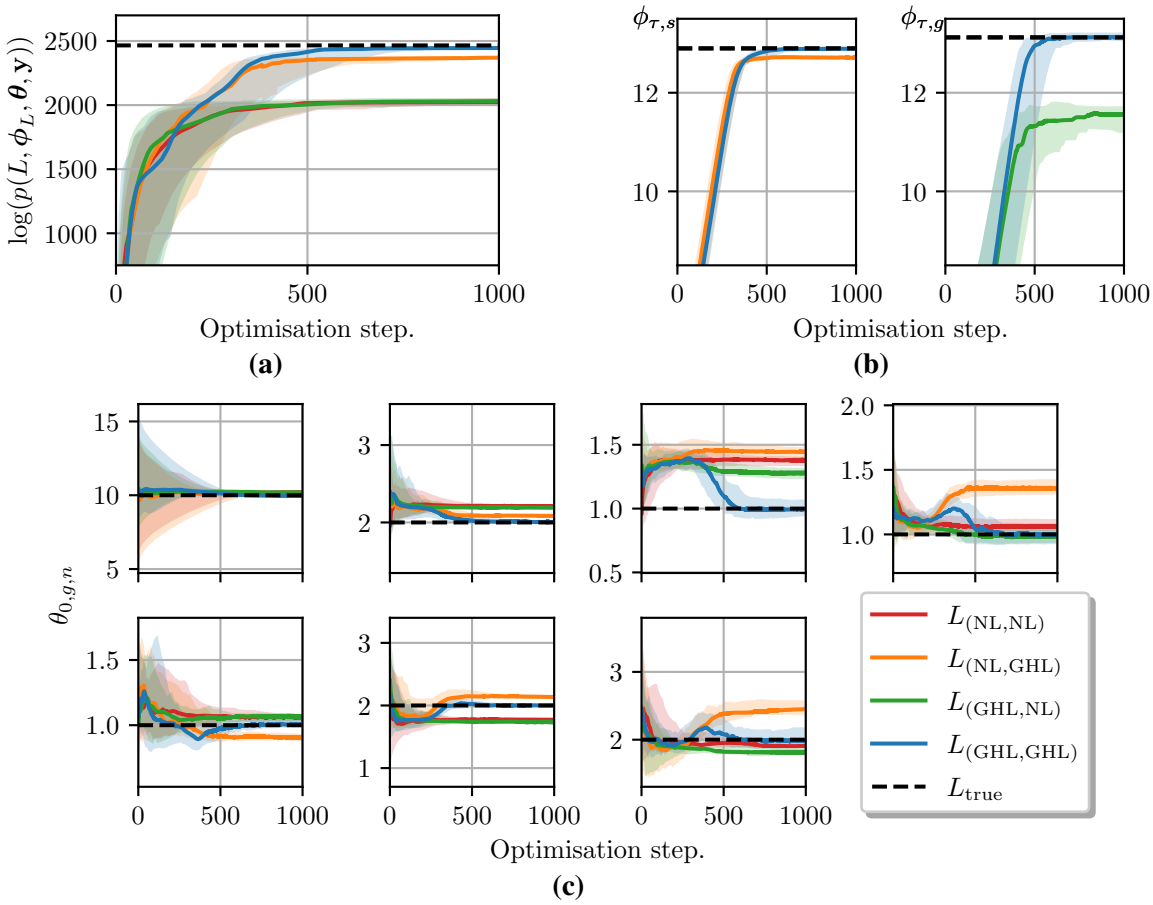


Figure 6.4: Results of the model selection experiment, searching over four learning rules, introduced in Section 6.3.2. The true learning rule is GHL for both chemical and electrical synapses. In Figure 6.4a we plot the log joint density during the optimisation procedure. Only the correct learning rule matches the density of the true model. This is confirmed in Figures 6.4b and 6.4c, where we show the convergence of the model parameters. Only for the correct learning rule do the parameters converge to the correct value. The parameters converge to an incorrect value for an incorrect learning rule. Shown are the median and quartiles across fifty random restarts.

$\mathbb{R}_{\geq 0}$, $\mathbf{\Phi}_{(\mathrm{NL,GHL})} = \mathbb{R}_{\geq 0}$, and $\mathbf{\Phi}_{(\mathrm{GHL,GHL})} = \mathbb{R}^2_{\geq 0}$. This does not require the use of `GEN_RULE` for generating learning rules, but does make use of the grammar for constructing the learning rule candidates. We also learn the initial weight matrices for gap junctions and the continuous parameters of the learning rule. This experiment, although simple, can be considered as an automation of the naive approach taken by human practitioners, where a finite set of rules are specified and the best rule is selected.

The results for this experiment are shown in Figure 6.4. The true learning rule used to generate the synthetic data is GHL for both chemical synapses and electrical gap junctions. In Figure 6.4a we see log model evidence increases during optimisation for all candidate learning rules. The correct learning rule attains the best performance, and is the only rule

that matches the performance of the true model. Figure 6.4b shows recovering the static parameters of the learning rule, ϕ_L, as applicable for the different learning rules (noting that NL does not define any ϕ_L parameters). We again see that the correct ϕ_L values are only recovered for the correct learning rule. In Figure 6.4c we see that the gap junction strengths are recovered accurately for only the correct learning rule. When using an incorrect learning rule, the learned gap junction strengths are incorrect.

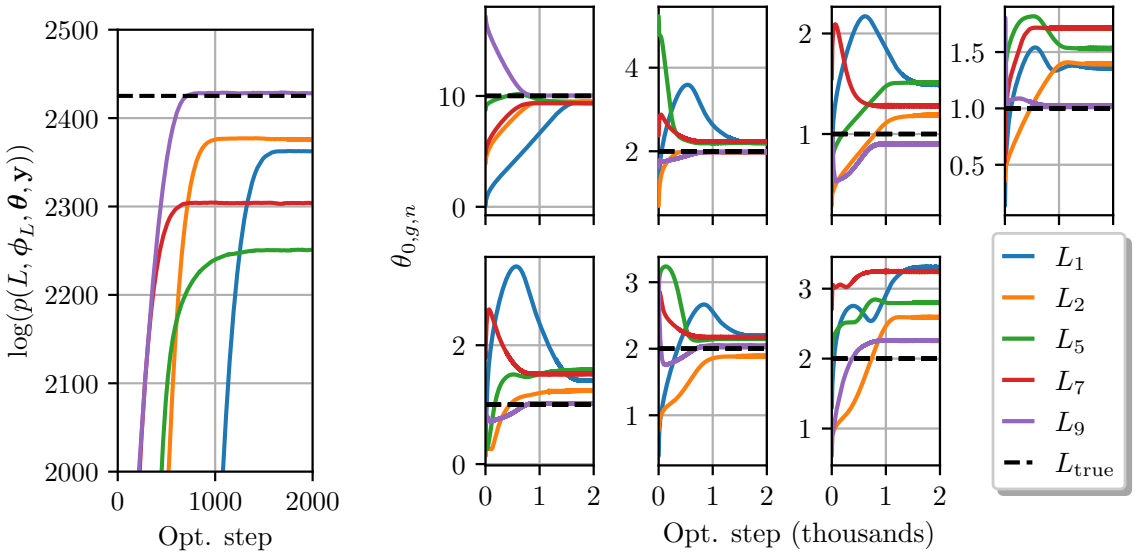
This short experiment demonstrates how we can use the formulation outlined above to test different learning rules, and, estimate the parameters of the simulator using gradient descent. This experiment also shows this formulation can be configured to closely mirror how scientists typically do research. A set of plausible candidate hypothesis classes are defined, and any tunable parameters of each hypothesis are fitted to the dataset of interest. The best hypothesis is then selected as the solution.

6.3.3 Full Induction

We conclude this chapter by introducing full induction of the learning rule. For this, we use the generation procedure outlined in Section 6.2.3 to jointly recover the electrical junction update rule, scalar parameters and intial connectome weights from calcium fluorescence traces. The results of this are shown in Figure 6.5. In Figure 6.5a we show ten examples of learning rules generated using the CFG. We show both the verbatim programmatic form, as sampled under the CFG and generative model, as well as the equivalent mathematical form. We find that some rules are not numerically stable, and lead to either the simulated potentials growing exponentially, or, failure of the autodifferentiation package. This failure is indicated by a value of N in the "Stable" column. We also show the highest joint density obtained by each rule during optimisation. The rule that achieves the highest joint density, L_9, has a mathematical form equivalent to the true learning rule. This is shown in Figure 6.5b. We show the convergence of the log joint during learning for those rules that are stable. We see that a range of performances is attained by the different learning rules. However, as shown in Figure 6.5c, the incorrect parameters are recovered when using the wrong learning rule. We do not show the ϕ_L convergence, as these are not immediately comparable between different learning rules, but note that they converge to the correct effective value only for the correct learning rule.

L_k	Programmatic Form	Full Mathematical Form	Stable	Best Joint
0	$(\texttt{t.add}(\texttt{t.mul}(\texttt{vbar.usq}(1), \texttt{vbar.usq}(-1)), \texttt{theta}))$	$\mathbf{C} \odot \left(e^{\phi_0} \left((\bar{\mathbf{v}}_t \otimes \bar{\mathbf{v}}_t) + \boldsymbol{\theta}_t\right)\right)$	N	N/A
1	$\texttt{t.mul}(\texttt{t.sum}(\texttt{t.usq}(\texttt{phi.g1}, 1), 1).\texttt{usq}(1), \texttt{vbar.usq}(-1)))$	$\mathbf{C} \odot \left(e^{\phi_0} \left(\left(\phi_1 \times \mathbb{1}^{N\times 1}\right) \otimes \bar{\mathbf{v}}_t\right)\right)$	Y	2362
2	$(-\texttt{t.add}(\texttt{t.mul}(\texttt{theta}, \texttt{t.exp}(\texttt{phi.g1})), \texttt{phi.g2}))$	$-\mathbf{C} \odot \left(e^{\phi_0+\phi_1} \boldsymbol{\theta}_t + \phi_2 e^{\phi_0}\right)$	Y	2376
3	$\texttt{t.usq}(\texttt{t.add}(\texttt{phi.g1}, \texttt{vbar}), 1)$	$\mathbf{C} \odot \left(e^{\phi_0} \left(\phi_1 + \bar{\mathbf{v}}_t\right) \otimes \mathbb{1}^{N\times 1}\right)$	N	-41022
4	$\texttt{t.exp}(\texttt{theta})$	$\mathbf{C} \odot e^{\phi_0 + \boldsymbol{\theta}_t}$	N	N/A
5	$(-\texttt{t.mul}(\texttt{theta}, \texttt{theta}))$	$-\mathbf{C} \odot \left(e^{\phi_0} \boldsymbol{\theta}_t \odot \boldsymbol{\theta}_t\right)$	Y	2251
6	$\texttt{t.mul}(\texttt{theta}, \texttt{t.exp}(\texttt{phi.g1})$	$\mathbf{C} \odot \left(e^{\phi_0} \boldsymbol{\theta}_t \left(e^{\phi_1} \otimes \mathbb{1}^{N\times 1}\right)\right)$	N	-67202
7	$(\texttt{t.usq}(\texttt{vbar}, -1))$	$\mathbf{C} \odot \left(e^{\phi_0} \left(\bar{\mathbf{v}}_t \otimes \mathbb{1}^{N\times 1}\right)\right)$	Y	2304
8	$\texttt{t.mul}(\texttt{theta}, \texttt{t.usq}(\texttt{t.exp}(\texttt{phi.g1}), 1))$	$\mathbf{C} \odot \left(e^{\phi_0} \boldsymbol{\theta}_t \otimes \mathbb{1}^{N\times 1}\right)$	N	-115648
9	$\texttt{t.mul}(((\texttt{vbar}).\texttt{T}).\texttt{T.usq}(1), \texttt{vbar.usq}(-1))$	$\mathbf{C} \odot \left(e^{\phi_0} \left(\bar{\mathbf{v}}_t \otimes \bar{\mathbf{v}}_t\right)\right)$	Y	**2428**
True	$\texttt{t.mul}(\texttt{vbar.usq}(1), \texttt{vbar.usq}(-1))$	$\mathbf{C} \odot \left(e^{\phi_0} \left(\bar{\mathbf{v}}_t \otimes \bar{\mathbf{v}}_t\right)\right)$	Y	2425

(a)



(b) **(c)**

Figure 6.5: Results of automated recovery of learning rules using induction as introduced in Section 6.3.3. We recover the update rule for the electrical junctions and the continuous parameters of the learning rule and connectome from synthetic calcium fluorescence data. Table 6.5a shows some of the programmatic learning rules automatically generated using the generative procedure outlined in Section 6.2.3 (we omit the initial symbol $\texttt{phi.dt} * \texttt{torch.mul}(\texttt{phi.C}, \texttt{torch.mul}(< \texttt{param} >, < \texttt{prim} >))$, abbreviate `torch` to `t` and `unsqueese` to `usq`), alongside the equivalent mathematical form of each rule. We generated 100 candidate learning rules, and the correct rule (denoted L_9) was the 19^{th} rule generated. Some of these learning rules are not stable and lead to the simulator or PyTorch failing (normally due to overflow). We also show the optimised joint density after learning the initial gap junction weights (Figure 6.5c) and the static parameters of the learning rule. We again see, both in Table 6.5a and Figure 6.5b, that only the correct learning rule matches the performance of the true learning rule. We also see that a learning rule with a different programmatic form can be equivalent to the true rule. Similarly, the learning rule is often recovered with multiple additional multiplicative and additive parameters (i.e. $\phi_0 \times \phi_1 + \phi_2$). In this scenario, the additive parameters are driven to zero and the product of the multiplicative parameters converges to the true value, even though each of these individual ϕ values are not equal to the true value. This highlights why manual inspection of the rule is important. In Figure 6.5c we show the training curves as we learn the initial gap junction weights for each of the five stable learning rules. We see that the correct gap junction weights are only recovered for the correct learning rule.

The slight difference in performance between the true and learned parameters is attributed to the noise in the observed data, the use of a prior, and noise in the latent state imputation process. If these are factors are removed, the optimal joint density exactly matches and the exact parameters are recovered. Crucially however, the correct learning rule has been recovered automatically, and good estimates of the generative parameters are recovered.

6.4 Discussion

In this chapter we have discussed learning the structure of a model of neural dynamics. Specifically, we considered recovering interpretable synaptic plasticity rules, expressed as code fragments, while also recovering continuously valued parameters from calcium imaging data. We defined a generative model for learning rules that can be integrated into the neural simulator, and used gradient descent to maximise the model evidence over continuous parameters in a purely optimisation-based approach. We perform this optimisation per candidate learning rule and select the best performing learning rule. We concluded by showing that we can generate a random set of learning rules, where the correct learning rule was recovered, although it had a more complex programmatic representation.

However, we were unable to recover chemical synapse weights from data. While the optimised chemical synapse weights generated very similar latent state trajectories to the true data, also reflected by only a small difference in the model evidence between the true model and the learned model, these parameters did not converge to the true underlying parameters, or, a single consistent parameter value across experimental repeats. This suggests that there are multiple local maxima, or, a continuum of solutions. We also use a highly restricted deterministic model such that we can investigate using backpropagation to learn parameters. It is well known however that differentiating through time-series models is especially prone to becoming trapped in local maxima [Cuéllar et al., 2007]. More sophisticated truncated backpropagation approaches [Aicher et al., 2019] may allow us to extend to longer time series, with lower memory and computational overheads, and crucially, avoid local maxima. Using a deterministic model is also highly restrictive, and is unlikely to yield acceptable inference results when used on real data. Therefore, we propose instead that we should instead develop full posterior state-space and parameter inference in a stochastic model, as was used in the VPC, instead of simply maximisation in a deterministic model.

In this chapter we also only considered simplistic program synthesis and symbolic regression techniques. Scaling this symbolic regression to full and unsupervised recovery of complex systems from large sets of data is non-trivial. However, more powerful algorithms for performing this synthesis are continually being developed [Ellis et al., 2019; Udrescu et al., 2020]. Exploiting these more powerful allows the range of learning rules considered under the CFG to be expanded and improve the efficiency with which this space is explored. Although we have not considered it here, this problem can also be formulated as an infinite bandit with non-stationary rewards [Besbes et al., 2014]. Considering the problem under this framework may allow the optimisation of multiple rules simultaneously, where information is shared between arms, and where the least promising learning rules are discontinued without wasting computational resources.

The learning rule class we define is purposefully general, and can be easily extended to include additional dependencies. For instance, calcium-dependent synaptic plasticity [Zucker, 1999] can be introduced by adding the intracellular calcium state, $\mathbf{c}_t$, to the interface definition. The set of possible learning rules that can be generated can also be expanded to be stateful by adding additional variables to $\boldsymbol{\theta}$, with only minor modifications to the CFG and generative procedure. For instance, this would allow time-averaged activities to be used instead of simply instantaneous activity, as in the BCM update rule [Bienenstock et al., 1982].

In this chapter we exclusively discussed recovering learning rules. However, on a broader level, the main theme we explored is quantitatively evaluating different models and model components by comparing the degree to which they explain the observed data. Beyond this, we considered automatically proposing and testing candidate models, with the objective of recovering an interpretable generative model that is capable of explaining an observed phenomena. This extends the continuous parameter learning we discussed in Chapter 4 and 5 to the discrete structure of the model. Although we only present preliminary experiments in this chapter, we believe that this work highlights the extensibility and flexibility afforded by a bottom-up modelling approach when combined with powerful machine learning techniques. Beyond simply estimating parameter values, this approach allows us, to some degree, to automate the development and exploration of entirely new neural models.

7
Brittle Simulators

In Chapter 5 we explored using the body simulator, WormSim, in isolation to impute neural states from observed body pose data. The core of the simulator is a deterministic function of state. Deterministic models are approximations of reality that are easy to interpret and often easier to build than inherently stochastic alternatives. Unfortunately, as nature is capricious, observed data can never be fully explained by deterministic models in practice. Deterministic simulators are often therefore "converted" to "stochastic" simulators by stochastically perturbing the state at each time step, in order to compensate for epistemic uncertainty due to modelling approximations and unmodelled aleatoric uncertainty. In practice, this allows the simulator to better explain the variability observed in real data without requiring excessive observation noise and facilitates better extrapolation from noisy data. Such models are more resilient to misspecification, are capable of providing uncertainty estimates, and provide better inferences in general [Lv et al., 2008; Møller et al., 2011; Pimblott & LaVerne, 1990; Renard et al., 2013; Saarinen et al., 2008].

We first investigated perturbing the physical WormSim states with a small amount of Gaussian noise. Adding state-independent Gaussian noise with heuristically tuned variance to perturb the state is common throughout applied sciences [Warrington et al., 2019; Adhikari & Agrawal, 2013; Allen, 2017; Brockwell & Davis, 2016; Du & Sam, 2006; Fox, 1997; Mbalawata et al., 2013; Reddy & Clinton, 2016; Wood et al., 2020]. However, unlike the perturbations applied to the neural state model in Chapter 4, additive perturbations applied directly to the physical state often caused the simulator to "crash," raising an exception instead of the intended return value. We discussed this more in Section 4.1. These crashes are the result of the ODE integrator being unable to solve the differential equation from the perturbed state to the required accuracy in the allocated computational budget. This failure may be avoidable through the use of more sophisticated or stiff ODE solvers. However, these solvers may incur an untenable computational cost. Failures also occur when the perturbed state enters into an unhandled region of state-space, and hence any simulator is

incapable of solving the equations of state. For instance, independent additive Gaussian noise may cause solid bodies to intersect prior to simulation. This cannot be remedied through using a more powerful solver, and is simply a flaw of the specified proposal by proposing states with zero probability under the model. However, designing a proposal for which these intersections are not encountered may be prohibitively difficult or as computationally demanding as the original simulation to sample from, and hence designing an independent proposal and simply resampling and rejecting until a valid configuration is found may be the only reasonable course of action.

Therefore, failures fall broadly into two categories: failure of the computational implementation (e.g. numerical overflow, solver timeout), and attempting to operate on invalid states (e.g. solid bodies intersecting). However, we do not differentiate between these failure modes, and formally define both types of failure as extending the possible output of the simulator to include $\perp$ (read as "bottom"), denoting simulator failure. Manually establishing the state-perturbation pairs that cause failure is non-trivial. Hence, the simulation artefact can be sensitive to seemingly inconsequential alterations to the state – a property we describe as *brittle*. Failures waste computational resources and reduce the diversity of simulations for a finite sample budget. For instance, when such a simulator is used in sequential Monte Carlo inference, these failures deterministically kill particles, wasting computational resources and reducing sample diversity, increasing the variability of the resulting inferences. Particles that fail are effectively assigned zero probability, and hence are discarded immediately in particle filter sweeps. Furthermore, we observed that particles that failed were significantly more computationally demanding than particles that succeeded. This means that disproportionate computational effort was expended on particles do not contribute to the inference result. As such, we may choose to not perturb the state as aggressively as needed to successfully compensate for the modelling approximations.

As such, we wish to learn a proposal over perturbations such that the simulator exits with high probability, but, crucially, renders the originally specified joint distribution unchanged. The technique we develop reduces the frequency of simulator failure while maintaining the original model, and, is trivially integrated with existing state-space and posterior inference methods. We proceed by framing sampling from brittle simulators as rejection sampling. We then seek to eliminate rejections by estimating the state-dependent density over perturbations

that do not induce failure. We show how to train a conditional autoregressive flow to propose perturbations such that the simulator succeeds with high probability, increasing computational efficiency. We then demonstrate that using this learned proposal yields lower variance results when used in posterior inference with a fixed sample budget, such as pseudo-marginal evidence estimates produced by sequential Monte Carlo sweeps. Much of this material first appeared in Warrington et al. [2020b]. Source code for reproduction of figures and results is available at `https://github.com/plai-group/stdr`.

7.1 Background

We now introduce some specific background material, and re-frame and redefine some material presented previously.

7.1.1 Smoothing Deterministic Models

Deterministic simulators are often stochastically perturbed to increase the diversity of the achievable simulations and to fit data more effectively. The most widespread example of this is perturbing linear dynamical systems with Gaussian noise at each timestep. Linearity then means that the distribution over state at each point in time is also Gaussian distributed with analytically calculable parameters. However, simple linear dynamics may be insufficient for modelling more complex systems. Examples of such systems are: stochastic models of neural dynamics [Warrington et al., 2019; Coutin et al., 2018; Fox, 1997; Goldwyn & Shea-Brown, 2011; Saarinen et al., 2008], econometrics [Lopes & Tsay, 2011], epidemiology [Wood et al., 2020; Allen, 2017] and mobile robotics [Fallon et al., 2012; Thrun et al., 2001]. In these examples, Monte Carlo methods are used, where particles defining the simulator state are perturbed with noise drawn from a distribution, and are then iterated using the simulator to create a discrete distribution over state. Defining a "stochastic" model in this way affords a great deal of freedom to the designer, as specifying and tuning a deterministic function is often easier than specifying the probabilistic dependencies between random variables. This means more complex models can be developed with lower overhead.

7.1.2 Simulator Failure

However, as simulators become more complex, guaranteeing the simulator will not fail for perturbed inputs becomes more difficult, and individual function evaluations become more expensive. Lucas et al. [2013] and Edwards et al. [2011] establish the sensitivity of

earth science models to global parameter values by building a discriminative classifier for parameters that induce failure. Sheikholeslami et al. [2019] take an alternative approach instead treating simulator failure as an imputation problem, fitting a function regressor to predict the outcome of the failed experiment given the neighbouring experiments that successfully terminated. However these methods are limited by the lack of clear probabilistic interpretation in terms of the originally specified joint distribution in time series models, their ability to scale to high dimensions, and their applicability to state-space models.

7.1.3 State-space Inference and Model Selection

Both perturbed deterministic models and inherently stochastic models are ultimately deployed to make inferences about the world. Hence the goal is to be able to recover distributions over unobserved states, predict future states and learn unknown parameters of the model from data. Posterior state-space inference refers to the task of recovering the distribution $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}, \mathcal{M})$, where $\mathbf{x}_{0:T}$ are the latent states, $\mathbf{y}_{1:T}$ are the observed data, and $\mathcal{M}$ denotes the model if multiple different models are available. Inference in Gaussian perturbed linear dynamical systems can be performed using techniques such as Kalman smoothing [Kalman et al., 1960], however, the restrictions on such techniques limit their applicability to complex simulators, and so numerical methods are often used in practice.

A common method for performing posterior inference in complex, simulation based models is sequential Monte Carlo (SMC), or its numerical approximation, particle filtering [Doucet et al., 2001]. We presented a more detailed examination of particle filtering and sequential Monte Carlo in Section 3.3.3, and the vanilla SMC algorithm is shown in Algorithm 3.3. Here, $p(\mathbf{x}_0)$ is the prior over initial state, the dynamics model (which here refers to a brittle simulator) is used as the proposal over state, defined later as $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$, and the likelihood, $p(\mathbf{y}_t|\mathbf{x}_t)$, is used to weight particles under the observed data. SMC produces a discrete approximation of the target distribution by iterating particles through the simulator, and then preferentially continuing those simulations that "explain" the observed data well. While this work does not require a detailed understanding of particle filtering, the core observation required for this work is that the likelihood of failed simulations is defined as zero: $p(\mathbf{y}_t|\mathbf{x}_t = \perp) \coloneqq 0$, and hence failed particles are rejected with certainty. Particles failing exaggerates particle degeneracy, which can dramatically increase the variance of subsequent inference results.

Posterior inference pipelines often also often provide estimates of the model evidence, $p(\mathbf{y}_{1:T}|\boldsymbol{\theta})$. SMC provides such an estimate, referred to as a pseudo-marginal evidence (denoted in Algorithm 3.3 as $Z_{0:N}$). This pseudo-marginal evidence is calculated (in log space) as the sum of the expected value of the unnormalised log importance weights (Algorithm 3.3, Lines 7). This evidence can be combined with the prior probability of each model via Bayes rule to estimate the posterior probability of the model (up to a normalising constant) [MacKay, 2003]. The joint density of different models can then be compared to perform Bayesian model selection, where the model with the highest probability is selected and used to perform inference. This is often referred to as marginal maximum *a posteriori* parameter estimation (or model selection) [Doucet et al., 2002; Kantas et al., 2015]. Performing model selection using this approach requires cheap, low-variance estimates of the model evidence. High failure rates can lead to high-variance evidence estimates, while low additive noise can limit the diversity of achievable simulations. Hence minimising the amount of rejection is critical, while still allowing sufficient perturbations to be used. This is the primary observation that this work builds on top of.

7.2 Methods

We consider deterministic models, expressed as programmatic simulators, describing the time-evolution of a state $\mathbf{x}_t \in \mathcal{X}$, where we denote application of the simulator iterating the state as $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1})$. A stochastic, additive perturbation to state, denoted $\mathbf{z}_t \in \mathcal{X}$, is applied to induce a distribution over iterated states after application of the simulator. The distribution from which this perturbation is sampled is denoted $p(\mathbf{z}_t|\mathbf{x}_{t-1})$, although, in practice, this distribution is often state independent. The iterated state is then calculated as $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1} + \mathbf{z}_t)$.

We consider specifically the case where the simulator can fail for "invalid" inputs, denoted by a return value of $\perp$. Hence the complete definition of f is $f : \mathcal{X} \rightarrow \{\mathcal{X}, \perp\}$. The region of valid inputs is denoted as $\mathcal{X}_A \subset \mathcal{X}$, and the region of invalid inputs as $\mathcal{X}_R \subset \mathcal{X}$, such that $\mathcal{X}_A \sqcup \mathcal{X}_R = \mathcal{X}$. Crucially, the boundary between these regions is unknown. Over the whole support, f defines a many-to-one function, as the whole of $\mathcal{X}_R$ maps to $\perp$. However, the algorithm we derive only requires that f is one-to-one in the accepted region. This is not uncommon in real simulators, and is satisfied by, for example, ODE models. We

Algorithm 7.1 Iterate brittle simulator, $\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$.

procedure ITERATESIMULATOR(f, $\mathbf{x}_{t-1}$)
  $\mathbf{x}_t \leftarrow \perp$
  while $\mathbf{x}_t$ is $\perp$ **do**
    if q_ϕ `is trained` **then**
      $\mathbf{z}_t \sim q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$ // Perturb under q_ϕ.
    else
      $\mathbf{z}_t \sim p(\mathbf{z}_t|\mathbf{x}_{t-1})$ // Perturb under p.
    end if
    $\mathbf{x}_t \leftarrow f(\mathbf{x}_{t-1} + \mathbf{z}_t)$ // Iterate simulator.
  end while
  return $\mathbf{x}_t$
end procedure

define the random variable $A_t \in \{0, 1\}$ to denote whether the state-perturbation pair does not yield simulator failure, and hence is accepted.

In Section 7.2.1 we define the iteration of the perturbed deterministic simulator as a rejection sampler, with a well-defined target distribution. We use this definition and the assumptions on f in Section 7.2.2 to show that we can target the same distribution by learning the state-conditional density of perturbations, conditioned on acceptance. We train an autoregressive flow to fit this density in Section 7.2.3, and describe how this can be used in inference in Section 7.2.5, highlighting the ease with which it can be inserted into a particle filter. We conclude by verifying empirically in Section 7.3 that using this learned proposal distribution in place of the original proposal improves the performance of particle-based state-space inference methods.

7.2.1 Brittle Simulators as Rejection Samplers

We begin by studying the naive approach to sampling from the perturbed system, as is shown in Algorithm 7.1 (assuming the proposal we introduce, q_ϕ, is not trained). This algorithm repeatedly samples from the proposal distribution and evaluates f until the simulator successfully exits. This procedure defines $A_t = \mathbb{I}\left[f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp\right],\ \mathbf{z}_t \sim p(\mathbf{z}_t|\mathbf{x}_{t-1})$, i.e. successfully iterated samples are accepted with certainty. This incurs significant wasted computation as the simulator must be called repeatedly, with failed iterations being discarded. The objective of this work is therefore to derive a more efficient sampling mechanism.

We begin by establishing Algorithm 7.1 as a rejection sampler, targeting the distribution over successfully iterated states. This reasoning is illustrated in Figure 7.1. The behaviour of

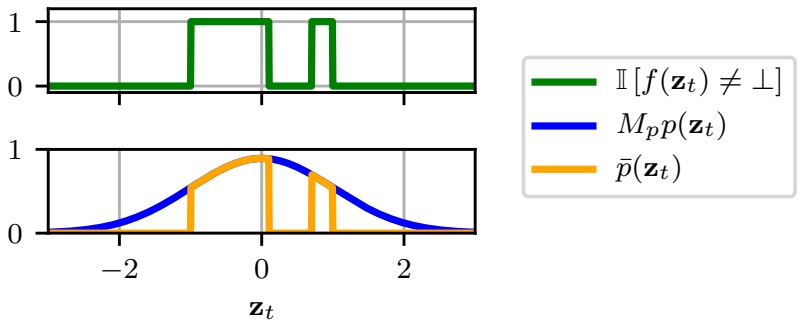


Figure 7.1: Graphical representation of how a brittle deterministic simulator acts as a rejection sampler, targeting $\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})$. For clarity we assume that $\mathbf{x}_t = 0$. The simulator, $f(\mathbf{z}_t)$, returns $\perp$ for unknown input regions, shown by an indicator function value of 0, shown in green. The proposal over perturbations, $\mathbf{z}_t$, is shown in blue, and is a correctly normalised distribution. The target distribution, $\overline{p}(\mathbf{z}_t)$, shown in orange, is implicitly defined as $\overline{p}(\mathbf{z}_t) = \frac{1}{M_p} p(\mathbf{z}_t)\mathbb{I}[f(\mathbf{z}_t) \neq \perp]$, where M_p is the normalizing constant of $\overline{p}$. It is easy to see that M_p is expected fraction of samples that are accepted, and hence corresponds to the acceptance rate of a rejection sampler. Accordingly, the proposal distribution, scaled by M_p, is *exactly* equal to $\overline{p}(\mathbf{z}_t)$ in the accepted region. Algorithm 7.1 therefore implicitly constructs a rejection sampler, where the acceptance criterion reduces to $\mathbb{I}[f(\mathbf{z}_t) \neq \perp]$. This point is subtle but important. Ordinarily, when specifying a rejection sampler, one has to specify an additional scaling constant to ensure that the scaled proposal distribution takes a value greater than or equal to the target density. Selecting a value for this scaling can be difficult as the form of the target function is unknown, and hence values are often chosen conservatively, reducing the efficiency. Furthermore, we cannot evaluate $\overline{p}$, and hence we rely on the fact that $\overline{p} \propto p$ in all regions of $\mathbf{z}_t$ space where samples are accepted. This means we can construct a rejection sampler without needing to specify any additional scaling constants or evaluate the target density.

f and the distribution $p(\mathbf{z}_t|\mathbf{x}_{t-1})$ implicitly define a distribution over successfully iterated states. We denote this distribution as $\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}, A_t = 1)$, and will use this distribution as the target distribution later. We use a bar herein to indicate that the sample was accepted, and hence the distribution places no probability mass on failures. Note there is no bar on $p(\mathbf{z}_t|\mathbf{x}_{t-1})$, indicating that it is defined before the accept/reject behaviours of f, and hence probability mass may be placed on regions that yield failure. The functional form of $\overline{p}$ is unavailable, and the density cannot be evaluated for any input value.

The existence of $\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$ implies the existence of a second distribution: the distribution over *accepted* perturbations, denoted $\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})$. Note that this distribution is also conditioned on acceptance under the chosen simulator, indicated by the presence of a bar. We assume f is one-to-one in the accepted region, and so the change of variables rule can be applied to directly relate this to $\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$. Under our initial algorithm for sampling from a brittle simulator we can therefore write the following identity:

$$\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1}) = \begin{cases} \frac{1}{M_p} p(\mathbf{z}_t|\mathbf{x}_{t-1}), & \text{if } f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

where the normalising constant M_p is the acceptance rate under p. (7.1) indicates accepting with certainty perturbations that exit successfully can be seen as proportionally shifting mass from regions of p where the simulator fails to regions where it does not. We exploit this definition to learn an efficient proposal.

7.2.2 Change of Variable in Brittle Simulator

We now derive how we can learn the proposal distribution, denoted q_ϕ and parametrised by ϕ, to replace p, such that the acceptance rate under q_ϕ (denoted M_{q_ϕ}) tends towards unity, minimising wasted computation. We denote q_ϕ as the proposal we train, which, coupled with the simulator, implicitly defines a proposal over accepted samples, denoted $\overline{q}_\phi$.

Expressing this mathematically, we wish to minimise the distance between the distribution implicitly specified over *accepted* iterated states using the *a priori* specified proposal distribution, $\overline{p}$, and the learned proposal, $\overline{q}_\phi$:

$$\phi^* = \arg\min_\phi \mathbb{E}_{p(\mathbf{x}_{t-1})} \left[\mathrm{KL} \left[\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1}) || \overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1}) \right] \right], \tag{7.2}$$

where we select the Kullback-Leibler (KL) divergence as the metric of distance between distributions. The outer expectation defines this objective as amortised across state space, where we can generate the samples by directly sampling trajectories from the model [Gershman & Goodman, 2014; Le et al., 2016]. We use the forward KL as opposed to the reverse KL, $\mathrm{KL}\left[\overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1})||\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})\right]$, as high-variance REINFORCE estimators must be used to obtain the the differential with respect to ϕ of the reverse KL.

Expanding the KL term yields:

$$\phi^* = \arg\min_\phi \mathbb{E}_{p(\mathbf{x}_{t-1})} \mathbb{E}_{\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})} \left[\log\left(w\right) \right], \tag{7.3}$$

$$w = \frac{\overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})}{\overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1})}. \tag{7.4}$$

Noting that $\overline{q}_\phi$ and $\overline{p}$ are defined only on accepted samples, where f is one-to-one, we can apply a change of variables defined for $\overline{q}_\phi$ as:

$$\overline{q}_\phi(\mathbf{x}_t|\mathbf{x}_{t-1}) = \overline{q}_\phi(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1}) \left| \frac{\mathrm{d}f^{-1}(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t} \right|, \tag{7.5}$$

and likewise for $\overline{p}$. This transforms the distribution over $\mathbf{x}_t$ into a distribution over $\mathbf{z}_t$ and a Jacobian term:

$$w = \frac{\overline{p}(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1}) \left| \frac{\mathrm{d}f^{-1}(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t} \right|}{\overline{q}_\phi(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1}) \left| \frac{\mathrm{d}f^{-1}(\mathbf{x}_t)}{\mathrm{d}\mathbf{x}_t} \right|}. \tag{7.6}$$

taking care to also apply the change of variables in the distribution we are sampling from in (7.3). Noting that the same Jacobian terms appear in the numerator and denominator we are able to cancel these:

$$w = \frac{\overline{p}(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1})}{\overline{q}_\phi(f^{-1}(\mathbf{x}_t)|\mathbf{x}_{t-1})}. \tag{7.7}$$

We can now discard the $\overline{p}$ term as it is independent of ϕ. Noting $f^{-1}(\mathbf{x}_t) = \mathbf{x}_{t-1} + \mathbf{z}_t$ we can write (7.2) as:

$$\phi^* = \arg\max_\phi \mathbb{E}_{p(\mathbf{x}_{t-1})}\mathbb{E}_{\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})}\Big[\log \overline{q}_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})\Big]. \tag{7.8}$$

However, this distribution is defined *after* rejection sampling, and can only be defined as in (7.1):

$$\begin{aligned}\overline{q}_\phi(\mathbf{z}_t|\mathbf{x}_{t-1}) &= q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1}, A_t = 1) \\ &= \begin{cases} \frac{1}{M_{q_\phi}} q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1}) & \text{if } f(\mathbf{x}_{t-1} + \mathbf{z}_t) \neq \perp, \\ 0 & \text{otherwise,} \end{cases}\end{aligned} \tag{7.9}$$

denoting M_{q_ϕ} as the acceptance rate under q_ϕ.

However, there is an infinite family of q_ϕ proposals that yield $\overline{p} = \overline{q}_\phi$, each a maximiser of (7.8) but with different rejection rates. Noting however that there is only a single q_ϕ that has a rejection rate of zero *and* renders $\overline{q}_\phi = \overline{p}$, and that this distribution also renders $q_\phi = \overline{q}_\phi$, we can instead optimise $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$:

$$\phi^* = \arg\max_\phi \mathbb{E}_{p(\mathbf{x}_{t-1})}\mathbb{E}_{\overline{p}(\mathbf{z}_t|\mathbf{x}_{t-1})}\left[\log q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})\right], \tag{7.10}$$

with no consideration of rejection behaviour under q_ϕ.

The objective stated in (7.10) implicitly rewards q_ϕ distributions that place minimal mass on rejections by placing as much mass on accepted samples as possible. This expression is differentiable with respect to ϕ and so we can maximise this quantity through gradient ascent with minibatches drawn from $p(\mathbf{x}_{t-1})$. This expression shows that we can learn the distribution over accepted $\mathbf{x}_t$ values by learning the distribution over the accepted $\mathbf{z}_t$, *without* needing to calculate the Jacobian or inverse of f. Doing so minimises wasted computation, targets the same overall joint distribution, and retains interpretability by utilising the simulator.

One might alternatively try to achieve low rejection rates by adding a regularisation term to (7.8) penalising high M_{q_ϕ}. However, differentiation of M_{q_ϕ} is intractable, meaning direct optimisation of (7.8) is intractable.

Algorithm 7.2 Train flow approximator, q_ϕ

procedure TRAINFLOW($p(\mathbf{x})$, $\overline{p}(\mathbf{z}|\mathbf{x})$, K, N, q, ϕ_0, η)
  for $n = 0 : N-1$ **do** // Gradient steps to take.
    for $k = 1 : K$ **do** // Sample fresh minibatch.
      $\mathbf{x}^{(k)} \sim p(\mathbf{x})$ // Sample from state prior.
      $\mathbf{z}^{(k)} \sim \overline{p}(\mathbf{z}^{(k)}|\mathbf{x}^{(k)})$ // Sample perturbation.
    end for
    $E_n \leftarrow \sum_{k=1}^{K} \log q_{\phi_n}\left(\mathbf{z}^{(k)}|\mathbf{x}^{(k)}\right)$ // Compute divergence term.
    $\mathbf{G}_n \leftarrow \nabla_{\phi_n} E_n$ // Apply backprop.
    $\phi_{n+1} \leftarrow \phi_n + \eta\left(\mathbf{G}_n\right)$ // Apply update.
  end for
  return ϕ_N // Return learned parameters.
end procedure

The critical point to take from this section is that we can define a variational distribution over accepted perturbations that can be trained using only successful iterations. This variational distribution, if sufficiently expressive, will exactly match the implicitly specified distribution over accepted perturbations, but can be sampled from without needing to repeatedly call the simulator. This means that, in the case of a perfectly trained proposal, the joint distribution is unchanged and no simulator calls will fail. We now discuss the practicalities of training and using such a proposal.

7.2.3 Training q_ϕ

To train $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$ we first define a method for sampling state-perturbation pairs. We initialise the simulator to a state sampled from a distribution over initial value, and then iterate the perturbed simulator forward for some finite period. All state-perturbation pairs sampled during the trajectory are treated as a training example, and, in total, represent a discrete approximation to the prior over state for all time, and accepted state-conditional perturbation, i.e. $\mathbf{x} \sim p(\mathbf{x})$ and $\mathbf{z} \sim \overline{p}(\mathbf{z}|\mathbf{x})$, and hence we do not explicitly index variables by t.

We use these samples to train the conditional density estimator, $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$, by maximising the conditional density of the sampled perturbation under the true distribution, as in (7.10), as this minimises the desired KL divergence originally stated in (7.2). Our conditional density estimator is fully differentiable and can be trained using stochastic gradient descent, as shown in Algorithm 7.2. The details of the chosen architecture is explained in Section 7.2.4. The result of this procedure is an artefact that approximates the density over valid perturbations conditioned on state, $\overline{p}(\mathbf{z}|\mathbf{x}) \approx \overline{q}_\phi(\mathbf{z}|\mathbf{x})$.

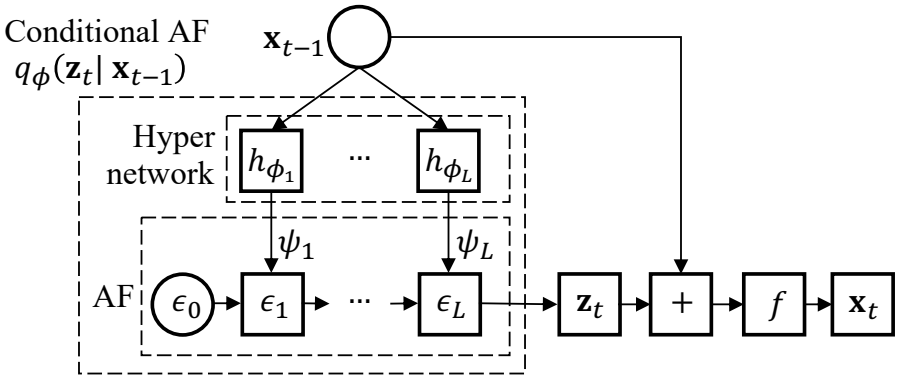


Figure 7.2: Diagram visualizing how q_ϕ is structured and used. The previous state is input to the hypernetwork, a series of L single layer neural networks, denoted h_{ϕ_l}. Each network outputs parameters, denoted ϕ_l, for each of the L layers in the flow conditioned on the state. The flow samples a perturbation as $\mathbf{z}_t \sim q_\phi\left(\mathbf{z}_t|\mathbf{x}_{t-1}\right)$, with the internal states of the flow denoted by ϵ_l. This perturbation is summed with the previous state and passed through the simulator, f, outputting the iterated state, $\mathbf{x}_t$.

7.2.4 Implementation

We parameterise the density q_ϕ using a conditional autoregressive flow (CAF) [Larochelle & Murray, 2011], introduced in detail in Section 3.5. Flows define a parameterised density estimator that can be trained using stochastic gradient descent. Flow variants have been used in image generation [Kingma & Dhariwal, 2018], as priors for variational autoencoders [Kingma et al., 2016], and in likelihood-free inference [Lueckmann et al., 2019; Papamakarios et al., 2019]. Specifically, we structure q_ϕ using a masked autoregressive flow [Papamakarios et al., 2017], with five single-layer MADE blocks [Germain et al., 2015], and batch normalisation at the input to each intermediate MADE block. The dimensionality of the flow is the number of states perturbed in the original model. We implement conditioning through the use of a hypernetwork [Ha et al., 2016], which outputs the parameters of the flow layers given $\mathbf{x}_{t-1}$ as input, as shown in Figure 7.2. The hypernetworks are single-layer neural networks defined per flow layer. We note that the learnable parameters of the flow are therefore the parameters of the hypernetwork, and not the parameters of the flow itself. Together, the flow and hypernetwork define $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$, and can be jointly trained using stochastic gradient descent. The networks are implemented in PyTorch [Paszke et al., 2017] and are optimised using ADAM [Kingma & Ba, 2014].

7.2.5 Using q_ϕ

Once q_ϕ has been trained, it can be deployed to enhance posterior inference, simply by replacing samples from $p(\mathbf{z}_t|\mathbf{x}_{t-1})$ with $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$. We highlight particularly the ease with which it can be combined with vanilla SMC, as introduced in Section 3.3, to enhance posterior state-space inference. The state is iterated by sampling from the proposal on Line 4 of Algorithm 3.3, where the proposal used is the iteration of the brittle simulator, $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \overline{p}(\mathbf{x}_t|\mathbf{x}_{t-1})$, and where the procedure for sampling from $\overline{p}$ is defined in Algorithm 7.1. Instead of sampling from $p(\mathbf{z}_t|\mathbf{x}_{t-1})$ in Algorithm 7.1, the sample is drawn from $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$, and as such the sample is more likely to be accepted. This modification requires only changing a single function call made inside the implementation of Algorithm 7.1. The SMC sweep will now propose perturbations more efficiently from the learned state-conditional distribution over perturbations.

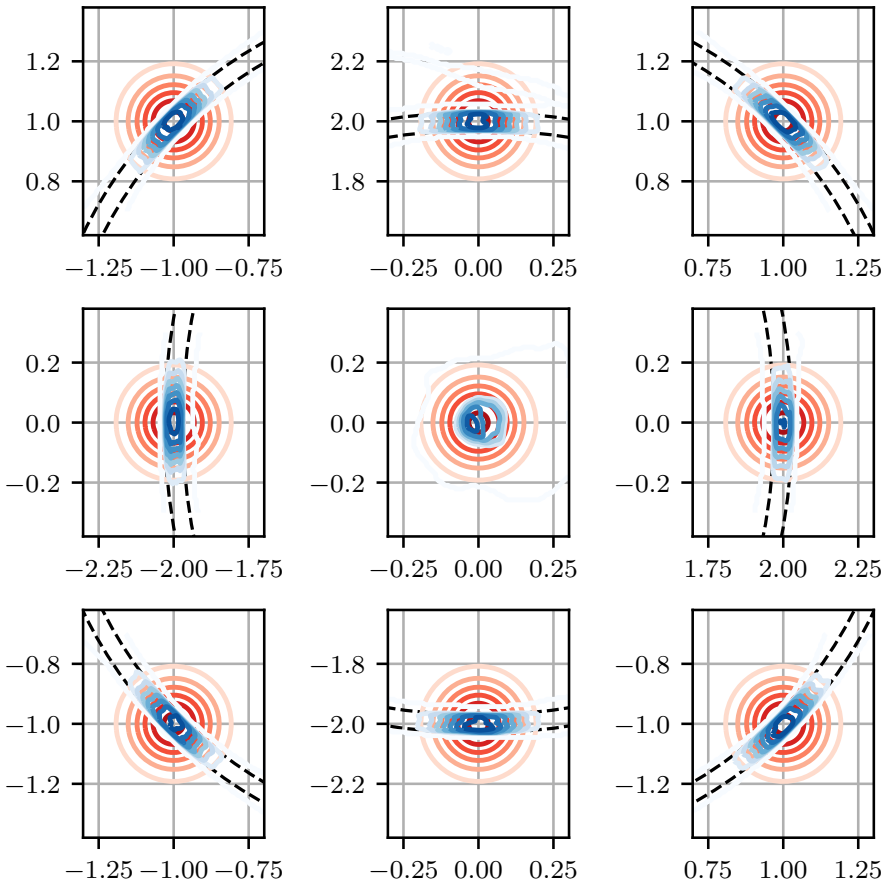
7.3 Experiments

We now experimentally validate the method presented above.

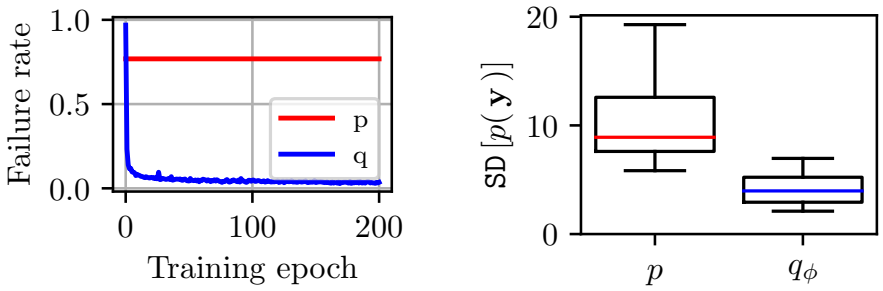
7.3.1 Annulus

We first demonstrate our approach on a toy problem. The true generative model is a constant speed circular orbit around the origin in the x-y plane, such that the simulator state is defined as $\mathbf{x}_t = \{x_t, y_t, \dot{x}_t, \dot{y}_t\} \in \mathbb{R}^4$. To analyse this data we use a misspecified model that deterministically simulates linear forward motion. To overcome the model mismatch and fit the observed data, we add Gaussian noise to position and velocity. The true radius is unknown and so we must amortise over possible radii.

We impose a failure constraint limiting the change in the distance of the point from the origin to a fixed threshold. This means that at each point there is an acceptable "annulus," such that the radius of the point only changes by a small amount. This condition was selected to mirror our observation that states in brittle simulators have large allowable perturbations in particular directions, but only very small permissible perturbations in other directions. The state in this example can be perturbed to either move the current position around the acceptable annulus, or, accelerate the point along the direction tangential to the annulus. Any significant radial movement or acceleration will cause failure.



(a)



(b) **(c)**

Figure 7.3: Results for the annulus problem introduced in Section 7.3.1. Figure 7.3a shows the state-space of this example. The position component of state is centred in each frame and the particle is at rest (zero velocity). We the show the learned velocity perturbation, which for unit-time integration corresponds directly to position permutation. The state-conditional acceptable region of perturbations is inside the black dashed band. The learned state-dependent proposal distribution over velocity (shown in blue) is well-approximating the original proposal (shown in red) *inside* the acceptable region, with minimal mass in the invalid region, all but eliminating rejection. This is confirmed in Figure 7.3b, where we show using the originally specified p yields approximately 75% rejection, while q_ϕ effectively eliminates rejection. Figure 7.3c shows the statistically significant reduction in the variance of the pseudo-marginal evidence approximation achieved by using q_ϕ ($p < 0.0001$). Each SMC sweep uses 100 particles, and we compute the variance across 100 independent SMC sweeps. We then construct the plot by comparing across randomly generated datasets 100 datasets.

For this experiment we use $T = 100$, $\delta t = 1$, a unit Gaussian prior over initial position, and initialise velocity from a zero-mean, 0.1 variance Gaussian. In the true generative model we update the velocity at each time step to be perpendicular to the radius, such that the state traces out a circle. We then observe the x-y coordinates perturbed with zero-mean, 0.1 variance Gaussian noise. The model used at inference does not correct the velocity to be tangential at each time point, and instead adds zero-mean, 0.1 variance Gaussian noise to the position and velocity.

The results of this experiment are shown in Figure 7.3. The interior of the black dashed lines in Figure 7.3a indicates the permissible $\dot{x}$-$\dot{y}$ perturbation, for the given position and zero velocity, where we have centred each distribution on the current position for ease of visual inspection. Red contours indicate the original density $p(\mathbf{z}_t|\mathbf{x}_{t-1})$, and blue contours indicate the learned density $q_\phi(\mathbf{z}_t|\mathbf{x}_{t-1})$. The fraction of the probability mass outside the black dashed region is the expected rejection rate. Figure 7.3b shows the rejection rate drops from approximately 75% under the original model to approximately 4% using a trained q_ϕ.

We then use the learned q_ϕ as the perturbation proposal in an SMC sweep, where we condition on noisy observations of the x-y coordinates. As we focus on the sample efficiency of the sweep, we fix the number of calls to the simulator in Algorithm 7.1 to a single call, instead of proposing and rejecting until acceptance. Failed particles are then not resampled (with certainty) during the resampling. This means that each iteration of the SMC makes a fixed number of calls to the simulator, and hence we can compare algorithms under a fixed sample budget. We compute evidences using an SMC sweep on a trace of length 100, using 100 particles. We then compute the variance of each SMC sweep by performing 100 independent SMC sweeps. We then repeat this for 100 randomly generated datasets to generate the distribution over variance estimates. Figure 7.3c shows that we recover lower variance evidence approximations for a fixed sample budget by using q_ϕ instead of p. A paired t-test evaluating the difference in variance returns a p-value of less than 0.0001, indicating a strong statistical difference between the performance under p and q_ϕ, confirming that using q_ϕ increases the fidelity of inference for a fixed sample budget.

7.3.2 Bouncing Balls

Our second example uses a simulator of balls bouncing elastically, as shown in Figure 7.4a. We model the position and velocity of each ball, such that the dimensionality of the

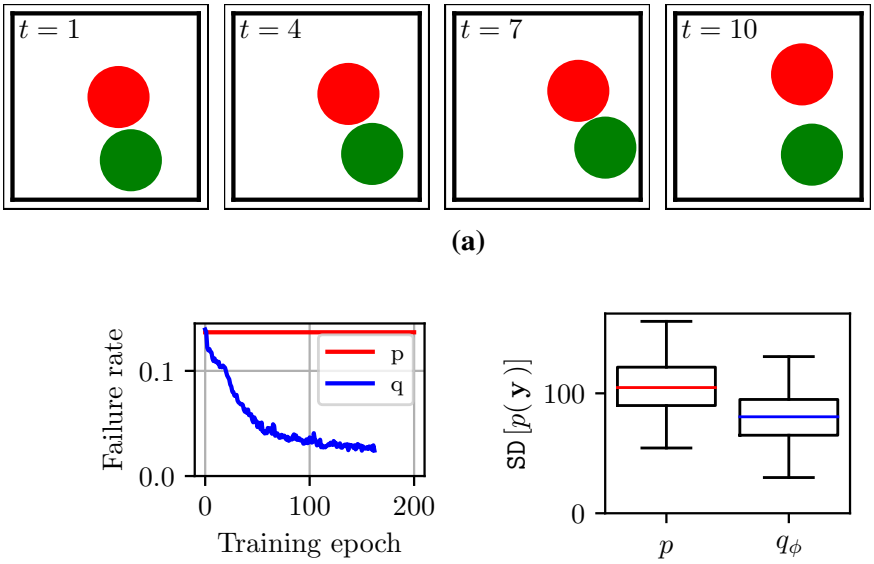
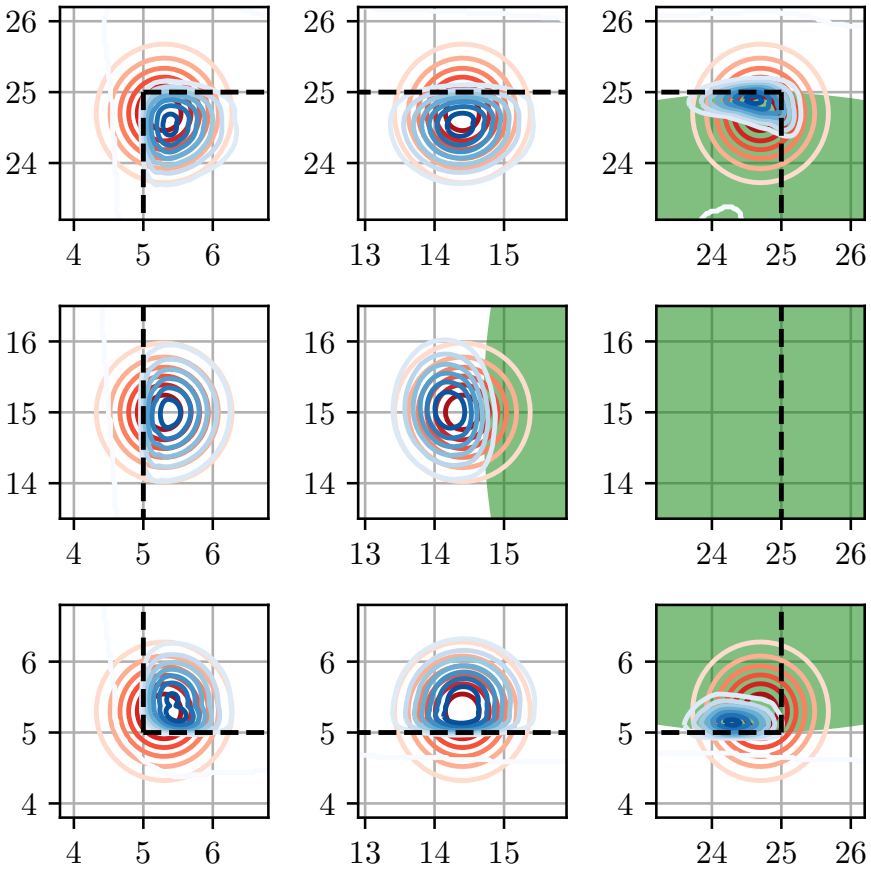


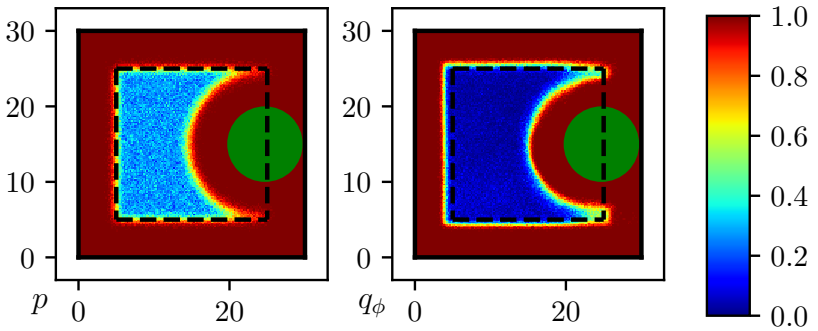
Figure 7.4: Results of the bouncing balls experiment introduced in Section 7.3.2. For this experiment, we use two balls of radius five and unit mass in an enclosure of size 30. As a result, the state-space of the simulator is eight-dimensional, representing the x-y position and velocity of each ball. Figure 7.4a shows an example trajectory of the system. 7.4b shows the reduction from approximately 14% rejection under p to 3% rejection under q_ϕ. Figure 7.4c shows the reduction in the standard deviation of the evidence approximation. The reduction is lower in this example as the system is reasonably "stable" with a low rejection rate for much of state space. The reduction is still statistically significant, scoring < 0.0001 on a paired t-test.

state vector, $\mathbf{x}_t$, is four times the number of balls (i.e. $\mathbf{x}_t \in \mathbb{R}^8$ here). We add a small amount of Gaussian noise at each iteration to the position and velocity of each ball. This perturbation induces the possibility that two balls overlap, or, a ball intersects with the wall, representing an invalid physical configuration and results in simulator failure. We note that here, we are conditioning on the state of *all* balls simultaneously, and proposing the perturbation to the state *jointly*.

We simulate the elastic collisions between the balls using the standard Newtonian equations of motion. We use two balls with radius 5 and equal mass, in a square enclosure with size 30. We perturb the position and velocity with Gaussian distributed noise with zero mean and standard deviation 0.5 and 0.1 respectively. The prior over the initial position is uniform in the allowed region, and a zero mean Gaussian over velocity in the x and y directions, with standard deviation $1.0/\sqrt{2}$. When performing state-space inference we observe only the centre of mass of the ball perturbed by Gaussian noise with a standard deviation of 2.0. As before, we use 100 particles in each SMC sweep on a trace of length



(a)



(b)

Figure 7.5: Figures investigating the spatial characteristics of the bouncing balls example. Figure 7.5a shows, in red, the proposal distribution over the perturbation to the position of the first ball specified in the model, and the learned proposal in blue. The edge of the permissible region of the enclosure is shown as a black dashed line. The second ball is fixed at $[25, 15]$, and the region that is therefore invalid is shaded in green. The flow has learned to deflect away from the disallowed regions. 7.5b shows the rejection rate as a function of the position of the first ball, with the second ball in the position shown. The trained proposal (right) has all but eliminated rejection in the permissible space compared to the a-priori specified proposal (left). The rejection rate under p is high in the interior as the second ball may also leave the enclosure, whereas q_ϕ has practically eliminated rejection by *jointly* proposing perturbations.

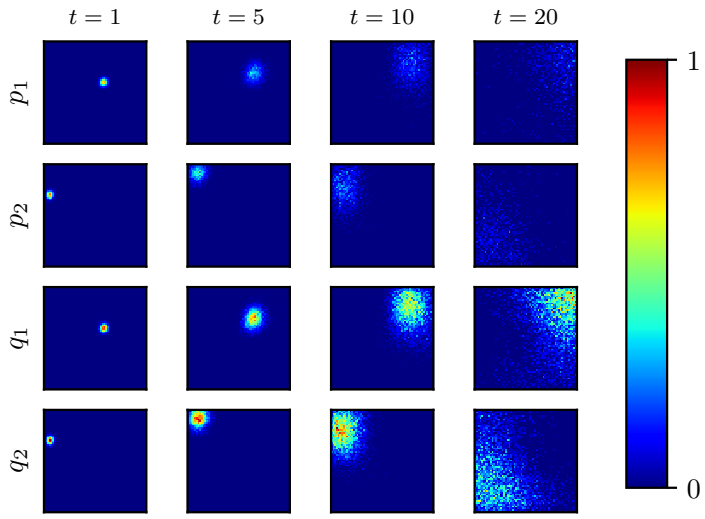


Figure 7.6: Figure showing the trajectories under p and q_ϕ. Columns are normalised heatmaps plotted at times 1, 5, 10, and 20. The top two rows are the positions of the two balls using p, and the bottom two rows are the positions of the two balls using q_ϕ. For a single starting point, it can be seen that there are many more samples surviving to $t = 20$ under q_ϕ, indicated by more, brighter pixels.

100. We then perform 20 SMC sweeps to compute the variance of the evidence, and repeat this for 100 random datasets to compute the distribution over variances.

The results of this experiment are shown in Figures 7.4, 7.5 and 7.6. We see in Figure 7.4b that rejection has been all-but eliminated through the use of q_ϕ. We also see a reduction in the variance of the evidence approximation computed by a particle filter when using q_ϕ instead of p. Figure 7.5a shows the distribution over position perturbation of a single ball, conditioned on the other ball being stationary in the shown position. Blue contours show the estimated distribution over accepted perturbations learned by autoregressive flow. Figure 7.5b shows the rejection rate under p and q_ϕ as a function of the position of the first ball, with the second ball fixed in the position shown, showing that rejection has been all but eliminated. We include in Figure 7.6 a heatmap plot indicating the distribution over the position of each ball during unconditional generation. The top two rows use p as the perturbation distribution, whereas the bottom row uses q_ϕ, where p_1 and q_1 indicate the first ball, and p_2 and q_2 indicate the second ball. We initialise 1000 particles and iterate the simulator. We plot the heatmap representing the distribution over the position of the balls at each time. If a particle fails, it is not resampled, and hence is discarded, reducing the total number of particles. We normalise the columns by the number of surviving particles. We can see that using q_ϕ results in many more particles surviving to $t = 20$, whereas comparatively few survive when using p.

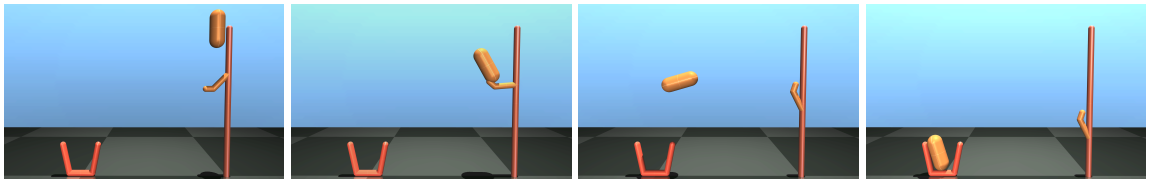
7.3.3 MuJoCo

We now apply our method to the popular robotics simulator MuJoCo [Todorov et al., 2012]. We study the built-in example *tosser*, where a capsule is tossed by an actuator into a bucket, shown in Figure 7.7a. Tosser displays chaotic behaviour, as minor changes in the position of the capsule result in large changes the trajectory. Hence, maintaining a diverse set of particles in these critical states is paramount for inference.

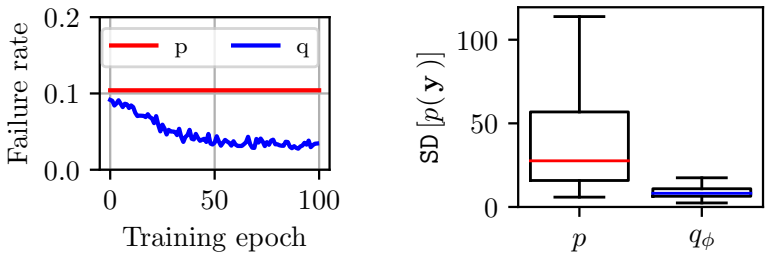
MuJoCo allows some overlap between the objects to simulate contact dynamics. This is an example of model misspecification borne out of the requirements of reasonably writing a simulator. We therefore place a hard limit on the amount objects are allowed to overlap. This is an example of a user-specified constraint that requires the simulator to be run to evaluate. We introduce the limit on overlap leveraging MuJoCos in-built collision detection, rejecting overlaps above 0.005. Typical overlaps in the standard execution of MuJoCo are below this limit.

During inference we observe only the x-y position of the capsule with Gaussian distributed noise, with standard deviation 0.1. We perturb the x-y position and velocity of the capsule with Gaussian distributed noise, with standard deviation 0.005 and 0.1 respectively. We perturb the angle and angular velocity of the capsule with Gaussian distributed noise, with standard deviation 0.05. These values were chosen to be in line with typical simulated values in the tosser example. We place a prior over the initial position and velocity with standard deviation 0.01 for position and 0.1 for velocities, and mean equal to their true position. In this, the state input to the normalising flow is the position and angle, and derivatives, of the capsule, as well as the state of the actuator. The actuator state is not perturbed under the model. We also input time into the normalising flow as the control dynamics are not constant with time.

Figure 7.7 shows the results of this experiment. The capsule is mostly in free space resulting in an average rejection rate under p of 10%. Figure 7.7b shows that the autoregressive flow learns a proposal with a lower rejection rate, reaching 3% rejection. However these rejections are concentrated in the critical regions of state-space, where chaotic behaviour occurs, and so this reduction yields a large reduction in the variance of the evidence approximation, as shown in Figure 7.7c. To evaluate this we perform 20 SMC sweeps per trace, using 100 particles per sweep, and repeat for 50 random traces.

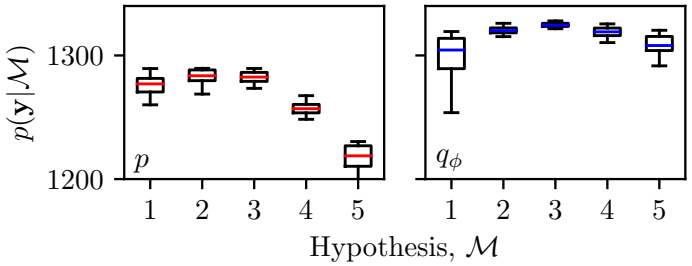


(a)



(b)

(c)

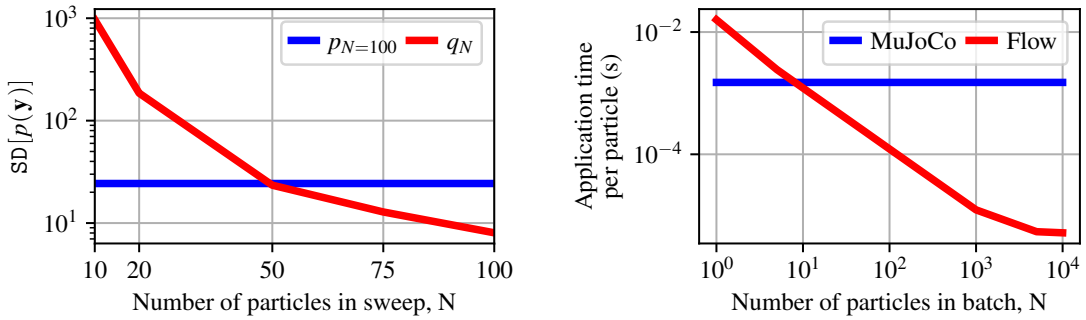


(d)

Figure 7.7: Results of the *tosser* experiment introduced in Section 7.3.3. Figure 7.7a shows the evolution of state over time. Figure 7.7b shows that the CAF learned markedly reduces the number of rejections below the baseline rejection rate, under p, of approximately 10%. Figure 7.7c shows the results of performing SMC using the *a priori* specified proposal and our learned autoregressive flow. The autoregressive flow creates an evidence estimator with a much lower variance, with a p-value of less than 0.0001 in a paired t-test, indicating a strong statistical difference in performance. Figure 7.7d shows the results of performing hypothesis testing, where hypothesis 3 is correct, under a uniform prior over hypothesis. The incorrect hypothesis is selected using p, while using q_ϕ the correct hypothesis is selected, with a statistically significant confidence.

We find that there is a strongly statistically significant difference in the variance of the evidence estimates ($p < 0.0001$).

We then evaluate our method on hypothesis testing using pseudo-marginal evidence estimates. The results for this are shown in Figure 7.7d. We test 5 different hypothesis of the mass of the capsule. Using p results in higher variance evidence approximations than when q_ϕ is used for most masses. Additionally, under p the wrong model is selected (2 instead of 3), although with low significance ($p = 0.125$), while using q_ϕ selects the correct hypothesis with $p = 0.0127$. For this experiment we note that q_ϕ was trained on a single value of mass, and that this "training mass" was different to the "testing mass." We believe this contributes

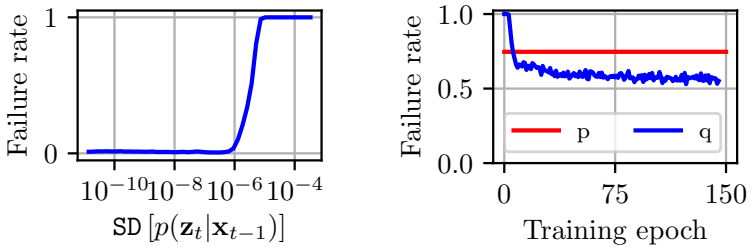


(a) **(b)**

Figure 7.8: Additional investigation of the computational burden of the tosser example. 7.8a shows that we need approximately half the number of particles when using q_ϕ to achieve the same variance in the pseudo-marginal evidence approximation when 100 particles are used in conjunction with p. Figure 7.8b shows the relative computational cost of iteration of the simulator and sampling from the flow. We can see that the computational cost of the flow is less than the cost of the simulator for roughly ten particles or more, as the flow can be sampled in parallel on a GPU, whereas the simulator must be iterated sequentially. For larger batches, the cost of the flow is negligible compared to the cost of the simulator. These results would be further exaggerated for more expensive simulators, such as the WormSim simulator.

to the increased variance in hypothesis 1, which is very light compared to the training mass. Training a q_ϕ with a further level of amortisation over different mass values would further increase the fidelity of the model selection. This is intimately linked with the larger project of jointly learning the model, and so we defer investigation to future works.

We also compare the runtime savings using the trained flow in place of the originally specified distribution for a fixed variance of the pseudo-marginal evidence approximation. The results of this are shown in Figure 7.8a. We compute the variance of the pseudo-marginal evidence approximation using p for an SMC sweep using 100 particles, shown as a constant line in blue. We then compute the variance of the SMC sweep using 10, 20, 50, 75, and 100 particles when using q_ϕ. We see that the variance under q_ϕ is approximately the same when using 50 particles, corresponding to a halving of the computational effort expended on the simulator. Of course, we then must compare the computational cost of iteration of the simulator to the cost of sampling from the flow, the results of which are shown in Figure 7.8b. We see that for 50 particles the cost of sampling from the flow is approximately an order of magnitude lower than iteration of the simulator, and as such represents a small computational overhead. As the number of particles grows, this cost drops further due to GPU efficiencies, whereas the cost of the simulator remains constant. This shows that the flow can be used at deployment time with minimal computational overheads, especially for larger particle



(a) **(b)**

Figure 7.9: Results from the WormSim example introduced in Section 7.3.4. Figure 7.9a shows the rate at which the simulator fails increases sharply as a function of the standard deviation of the applied perturbation. Figure 7.9b shows the reduction in rejections during training.

batches. The computational cost of the flow is small compared to the computational savings derived from fewer rejections. This effect is exaggerated with more expensive simulators.

7.3.4 Neuroscience Simulator

We conclude by applying our algorithm to the previously introduced *C. elegans* locomotion model, WormSim [Boyle et al., 2012]. For this experiment, we use the original implementation of WormSim, which uses a 414 dimensional state representation. We modify the compiled C++ WormSim code only such that it can be interacted with via a Python interface. We apply zero mean Gaussian noise to only the x-y coordinates of the control points of the worm, defining a 98 dimensional flow, where the standard deviation of this noise is 0.000005. The flow is conditioned on the full 414 dimensional state representation of the worm. We normalise the position of the worm such that its head is at the origin before application of the flow. The expected rate of failure increases sharply as a function of the scale of the perturbation applied, as shown in Figure 7.9a, as the integrator used in WormSim is unable to integrate highly perturbed states.

The rejection rate during training is shown in Figure 7.9b. We are able to learn an autoregressive flow with lower rejection rates, reaching approximately 53% rejection, when p has approximately 75% rejection. Although the rejection rate is higher than ultimately desired, we include this example as a demonstration of how rejections occur in simulators through integrator failure. We believe larger flows with regularised parameters can reduce the rejection rate further.

7.4 Discussion

In this work we have tackled reducing simulator failures caused by naively perturbing the input state. We achieve this by showing that stochastically perturbed simulators define a rejection sampler with a well defined target distribution and learning a conditional autoregressive flow to estimate the state-dependent proposal distribution conditioned on acceptance. We then show that using this learned proposal reduces the variance of inference results, with applications for Bayesian model selection. We believe this work has readily transferable practical contributions to not just the machine learning community, but the wider scientific community where such naively modified simulation platforms are widely deployed.

As part of the experiments we present we identify an extension: introducing an additional level of amortization over static simulation parameters. This extension builds towards our larger research vision of building toolchains for efficient inference and learning in brittle simulators and simulators that use rejection sampling. Further development will facilitate efficient gradient-based model learning in these brittle simulators.

8
Conclusion

In this thesis we have explored analysing the neural function of the nematode roundworm *C. elegans* using machine learning. *C. elegans* has been studied for decades, however, the vast majority of research is reliant on experimental techniques to expose neural mechanisms for *in vivo* measurement. The logistical difficulty of obtaining high-fidelity observations from many neurons means that often only small assemblies of neurons are analysed, as opposed to the vast networks of neurons known to be required to perform meaningful neural computation. These factors limit the range of hypotheses that can be quantitatively tested. These limitations also create a fundamental disconnect between the organism-level behaviours we can observe, and the scope of the underlying neural activity we can quantify. However, the very premise of Bayesian inference is to infer unobserved variables from observable covariates. The tools provided by Bayesian inference, and more generally machine learning, promise the ability to instead *infer* the unobserved variables of interest from easily measured covariates, using a structured and interpretable model.

Machine learning, particularly computer vision, has already revolutionised much of connectomics, automatically reconstructing large volumes of tissue in hours, as opposed to the days, months, or even years that manual annotation would have taken previously. While these *structural* connectomes yield invaluable information on the organisation of the entire brain at single-neuron fidelity, they yield comparatively little information about the *functional* organisation of the brain. Simply knowing that two neurons are connected does not necessarily shed light on *why* they are connected, or, how the information that flows between the neurons contributes to intelligent behaviour. To understand that, we must enhance the structural connectome with functional information; we must instead recover the functional connectome.

Possession of an accurate functional connectome, expressed as a parameterised simulator of the low-level neural interactions, would allow *in silico* experimentation. Hypotheses can be tested at single-neuron or single-synapse fidelity simply by iterating the simulator and inspecting the data generated by the program. These virtual experiments are perfectly

observable, reproducible and can be reconfigured quickly by simply changing configuration files. This expands the range of hypotheses that can be tested, makes experiments more reproducible by third parties, and removes the logistical overhead of conducting experiments *in vivo*. Furthermore, combining a functional connectome with state-space inference methodologies facilitates estimation of unobservable physiological neural states and parameters of interest from readily and non-invasively obtained data. Crucially, we intend for these methods to be computationally tractable and reconfigurable such that neuroscience experimentalists can use these tools on standard desktop hardware, without needing to also be a well-versed computer scientist.

We therefore advocate in this thesis that recovering and using functional connectomes for connectome-scale simulation and inference is critical to furthering not only research in *C. elegans*, but also connectomics and neuroscience research more widely. We posit that much of neuroscience research can be posed as posterior inference, parameter estimation or Bayesian model selection. As such, we dedicate much of thesis to discussing and developing the machine learning tools and methodologies required for augmenting structural connectomes with functional information inferred from data, using these functional connectomes to achieve neuroscience objectives, and for automating mechanical aspects of neuroscience research.

We first described using a physiological model to infer brain-wide latent states and global parameters from readily obtainable data, and subsequent posterior predictive inference, as a "virtual patch clamp." We introduced a parametrised brain-body model of *C. elegans*, and then performed particle-based inference to estimate the distribution over physiologically relevant time-varying latent states and static global parameters from calcium fluorescence data. We perform this inference on synthetic data, designed to be representative of the calcium fluorescence data that is being currently produced by laboratories. Although the method and results presented are in their infancy, we believe they are still the most ambitious and complete attempt to impute unobserved variables of interest, to refine models of *C. elegans*, and to provide a framework for performing *in silico* neuroscience experiments at connectome scale to date. When these methods are scaled to large bodies of real *C. elegans* data, the results could be profound for neuroscience, and would represent a true paradigm shift in how neuroscience research is conducted.

As part of this work we identified several extensions. We first investigated conditioning on raw *C. elegans* body pose data. Conditioning on body pose time series is particularly appealing as there are already large repositories of body pose data, the apparatus required for measuring the pose is inexpensive and widely available, specimens require no special preparation, and there is no need for manual annotation or solving a permutation problem as with calcium fluorescence imaging. We develop a pipeline for processing raw video footage to be immediately amenable for inference. We then successfully impute latent states and global parameters from synthetic data. However, when applied to real data, the results are less compelling. We believe that the locomotion model is sufficiently expressive, but the driving neural circuitry must be improved to allow more complex locomotion patterns to be created. We believe that this approach and data modality is invaluable for scaling to full-connectome analysis of real specimens. We also highlight several extensions from this work that may further improve performance or provide alternative approaches. We discuss these in more detail later.

We then discussed considering the neural model itself as a parameter that can be learned. Specifically, we considered recovering the synaptic weight update rule as an interpretable code fragment directly from data. This rule is pivotal in allowing biological organisms to learn. Studying synaptic plasticity and developing new and more expressive learning rules has long been a staple of neuroscience research. We consider this problem as a mixed continuous-discrete symbolic regression task. We develop an implementation that allows a learning rule to be expressed as a code fragment, generated randomly at run-time, and inserted into the model. This allows a diverse range of candidate learning rules to be generated and tested. We then use gradient descent to learn the static parameters of the learning rule, as well as the static connectome parameters. We find that using the incorrect learning rule leads to poor parameter estimates, suggesting that blindly assuming the model is correct may lead to spurious results. We believe this is the first time automated program induction or symbolic regression principals have been used in conjunction with models of neural interactions to automatically generate and test interpretable neural models and hypotheses. While inducing the learning rule may ultimately be overly-ambitious, introducing this flexibility into inference and model learning, and actually proposing *new* models with no user interaction, is a critical step towards automating neuroscience research.

Finally, we observed that the body model is prone to failure when randomly perturbed. As simulators become more complex, and machine learning methods are proliferated and used by practitioners in other fields, failures such as this are likely to become more common. Developing methods to not only accelerate inference, but allow inference to be performed at all, are likely to become increasingly beneficial as methodologies. We develop a method to reduce these failures by targeting the distribution of accepted particles post-iteration, and show that it is possible to pass this distribution back to the perturbation applied prior to iteration. We learn this distribution directly using a conditional autoregressive flow. We then show that we can efficiently deploy the flow to dramatically improve the performance of inference in the model. This work is a novel application embedding deep generative models directly into posterior inference pipelines.

The core aims of this thesis were three-fold. Firstly, we set out to explore, in detail, whole-connectome simulation and inference in *C. elegans* models. We believe that this is an under-explored topic that has game-changing potential for not only how neuroscience is performed, but for shaping our understanding of how cognition emerges from the atomic components of the brain. We were ambitious from the outset, and although we made significant progress, this is ultimately a topic that is substantially more challenging than we first envisaged, and likely spans more than a single PhD. Advancing this topic requires both cutting edge machine learning and neuroscience insight. Hence researchers must be comfortable in both domains to enable efficient collaboration. Therefore, our second objective was to present a thorough and unified introduction to enough machine learning and neuroscience background material such that a novice researcher looking to take up this area has access to a complete overview of the field and methods. Finally, we propose several topics and ideas that are crucial for advancing this line of research and are within reach. We also reflect, on a more meta-level, the insights that can be taken forward from this thesis. These proposals and insights are arguably the most valuable contribution of this thesis. We discuss the immediate and practical extensions in Section 8.1, and discuss the deeper insights in Section 8.2.

8.1 Next steps

The majority of the work in this thesis has been experimentally validated using synthetic data. This is ultimately because scaling inference to real data traces remains challenging. The data traces we use in Chapter 4 are just five seconds in length. Hence, scaling this

inference to process traces that are minutes in length is no small undertaking. We also wish to scale these methods to learn more than a handful of parameters: ideally we would like to recover *the entire connectome*. While computers are getting faster, we wish to create more efficient inference methods such that these methods can be run without super-computer scale computational hardware. We therefore discuss opportunities for enhancing these methods such that application to real data becomes tractable.

Calcium imaging technologies are continually being to developed and improved [Prevedel, 2020]. Microscopes and processing pipelines with higher-resolutions, wider fields of view and increased acquisition rates will immediately increase the availability and quality of data [Cramer et al., 2019; Stringer & Pachitariu, 2019; Yoshida et al., 2018]. However, the most promising innovation however is the development of multicolour calcium indicators [Inoue et al., 2019; Sands et al., 2018; Yemini et al., 2021], allowing neurons to fluoresce with different wavelengths. Selective multicolour calcium stains promise to simplify segmentation and labelling in fluorescence images, and, importantly, increase the number of *usable* traces in each dataset. We have shown (in simulation) that in the ideal case where all fluorescence traces can be observed, parameters and latent states can be recovered with high precision. Therefore, it stands to reason the inference problem will be eased as calcium imaging techniques and hardware improve. A further promising development is *voltage fluorescence imaging* [Knöpfel & Song, 2019; Peterka et al., 2011]. Similar to calcium fluorescence imaging, voltage imaging uses selectively fluorescent proteins that emit visible-light photons as a function of intracellular potential. If we could estimate the potential directly, then we could remove the non-trivial complexity introduced through the modelling of calcium. While voltage imaging is in its infancy compared to calcium fluorescence imaging, its development and proliferation promises to provide a more readily usable observation modality than calcium fluorescence imaging.

However, it is unlikely that microscopy techniques will ever be able to perfectly observe all neurons, and hence we cannot simply rely on engineering to solve the problem. Therefore, accessing more data, neuroscientific understanding, and powerful inference methods are pivotal. We suggest that *multi-fidelity* methods are key in the development. Multi-fidelity methods seek to reduce the overall computational cost and improve inference by leveraging additional and inexpensive low-fidelity models to complement high-fidelity

models. These models operate on a more coarse-grained level, and can more effectively leverage additional macro-scale data. This can then be used to "guide" inference in the high-fidelity model. Foremost we propose developing coarse-grained neural models to quickly provide approximate inference results, conditioned on minutes or hours of data, that can be used to guide inference in the high-fidelity neural model. These coarse models may use low-dimensional linear models [Fieseler et al., 2020; Linderman et al., 2019a] or exploit principal components of the data [Kato et al., 2015]. Developing low-fidelity neural models that are interoperable with high-fidelity neural models is an exciting avenue of investigation.

Multi-fidelity principals may also prove key to how we leverage body pose data, beyond simply using the pose as an additional likelihood term as was discussed in Chapter 5. This may be as simple as using the body pose to create a coarse-grained neural state proposal, while more advanced methods selectively call the high-fidelity neural model or the considerably more expensive locomotion simulator on demand. However, we propose that the best-case outcome is actually instead performing inference in the raw body poses offline, and then projecting a low-dimensional representation of the body pose into the high-dimensional neural model. This low-dimensional, but highly informative *pre-processed* pose "observation," can be used in-place of the raw body pose, or, to guide the neural simulation. This suggestion builds on the observation that we can effectively fit the body pose representation used by the simulator to observed data, and hence we can perform inference in the body model offline to estimate the distribution over the muscular activity required to generate the observed locomotion, and then estimate the distribution over neural activity required to drive this muscular activation. This distribution can then be used as either a data-driven proposal, or, an additional, more informative likelihood term to enhance the state-space inference results.

In contrast to developing low-fidelity models, we must also continue to develop high-fidelity models of additional elements of *C. elegans* physiology and behaviour. Examples of these elements include chemosensation [Bargmann, 2006] and peptome interactions [McDiarmid et al., 2015]. These are crucial mechanisms in *C. elegans*, and underpin many complex behaviours. For instance, leaving chemosensation unmodelled means that the model is unable to explain certain behaviours, and hence excessive uncertainty must be added to compensate. Adding such models allows for more accurate simulation of *C. elegans*, and allows a more diverse range of hypotheses and mechanisms to be studied and tested given

data. Many of the general ideas and concepts presented in this thesis may actually present a tool for more effective iterative refinement of neurophysical models, beyond that of simply hand designing and tuning models, by using machine learning.

A promising avenue of investigation that we have not discussed in detail is using reinforcement learning (RL) [Warrington et al., 2020a; Sutton, 1992]. We propose that RL can be used to learn neural controllers that generate specific locomotion or behavioural patterns. These controllers are conditioned on physiological states, and take actions in the physiological model, i.e. providing stimulating current to a motor neuron. The controls generated by the controller can then be scored under the true model, used as pseudo-observations, or, approximate solutions to initialise or expedite inference. Indeed, one could even consider using reinforcement learning themes to learn the connectome directly. The neural model we present in Chapter 4 is nothing more than a parameterised model, which could be considered as a neural "policy." Hence, if a reward function can be defined, it is hypothetically possible to construct a policy gradient over the parameters of the neural model, such that desired behaviours are generated. This may provide a more flexible and generalisable method for learning physiological parameters beyond performing marginal MAP estimation on fixed calcium traces.

The most daunting extension to consider is what this research entails for functional connectomics beyond *C. elegans*. We studied *C. elegans* in this work because of its small and fixed connectome. The next smallest and widely studied connectome is that of the common fruit fly, *Drosophila melanogaster*. The *Drosophila* connectome is estimated to have between $100,000$ and $200,000$ neurons, compared to just 302 in *C. elegans*. No complete reconstruction of this connectome exists, although concerted research efforts have reconstructed large swathes of the connectome. However, a major problem is that the connectome of *Drosophila* is not fixed, and hence the analysis tools we present here are less immediately applicable to *Drosophila* than they are to *C. elegans*. It is unlikely that we will ever be able to cheaply and rapidly construct complete *Drosophila* connectomes. Therefore, an outstanding question is how to apply the results of analysis of the functional connectome of *C. elegans* to *Drosophila*. Somewhat surprisingly, the most directly applicable models may therefore actually be the low-fidelity models operating on approximate connectomes estimated from data [Mishchencko et al., 2011]. These low-fidelity inferences can then

used to perform inference over the high-fidelity structure of the connectome. Performing inference in *Drosophila* is orders of magnitudes more complex than *C. elegans*, and hence this transfer represents a truly monumental challenge.

We conclude our dedicated discussion of *C. elegans* by posing a number of best-case outcomes for this line of research. These goals constitute something of a "moonshot," and are the end goals that researchers in this field should be aiming for, and continuously reframing research efforts to build towards. The first goal is the development of software on which manually configured exploratory simulations can be performed nearly instantaneously. This would facilitate rapid screening of experimental configurations, hypotheses, and model exploration to aide hypothesis generation. The second goal is then performing latent state imputation (using a fixed model) on moderate hardware in near-real time. Such imputation would allow experimentalists to analyse and enhance data in a time commensurate with which it was gathered. The third goal is that reasonable parameter estimates can be generated within hours on powerful – but not extreme – computational hardware. This means that experimentalists could schedule analysis of several datasets overnight. These results, combined with further rapid *in silico* experiments, could then be used to design and screen the next iteration of *in vivo* experiments within the space of hours, shortening the experimental design and execution cycle to a single day. The final goal is flexible and accurate conditional generation targeting all facets of the model given a wide range of design criteria. This would allow theoreticians and experimentalists alike to quickly and accurately generate synthetic data, parameters, adaptations to the structure of the model, or even whole new models, satisfying certain design criteria. These criteria could be as varied as sufficiently reconstructing observed time series data, to the expression rate of a particular behavioural motif, to enforcing particular regularisations on the network parameters. This data could be used to form and support hypotheses without the need for challenging (or impossible) *in vivo* experimentation. Accurate *in silico* experiments may replace many biological experiments and allow those without access to organic specimens or sophisticated measuring equipment to study *C. elegans*. If predictions made by the model were accurate over a broad range of criteria, this would truly constitute *in silico* neuroscience.

8.2 Final Thoughts

We conclude by discussing some of the overarching themes and highest-level insights we take from this entire body of research. We came into this research with a clear goal, which can be written, somewhat playfully, as:

$$\text{parameters}^*, \text{latents}^* = \underset{\text{parameters, latents}}{\arg\max}\ p(\text{parameters, latents} \mid \text{neuroscience, data}). \quad (8.1)$$

That is to say, we sought to develop tools to estimate the global parameter values and latent states for a particular specimen, given the entire accumulated body of neuroscience research and a dataset. While this was our specific goal, this general line of research has implications beyond just computational modelling of *C. elegans*.

The first overarching theme of this work is shifting the focus from designing complex experimental procedures to directly measure variables of interest *in vivo*, to designing expressive, interpretable models and then *inferring* quantities of interest from what can be easily measured in practice. Although we are not suggesting that this will entirely replace the development of more advanced experimental techniques, this will unload from experimentalists some of the burden of designing and conducting more complex experiments to measure particular variables, by instead providing distributions over these difficult to observe variables conditioned on what can be easily gathered. These methods promise to automate analysis of experimental results and expedite designing the most informative *in vivo* experiment, conditioned on the results from the previous experiment. This shift to using automated machine learning approaches also allows massive computational firepower to be leveraged to complement expert analysis. This promises to release researchers to spend less time performing menial experimental and data analysis tasks, and spend more time tackling the hardest research questions.

An extension of this is also shifting analysis methods from discriminative methods to generative methods with interpretable parameters. Much of scientific research is remarkably similar to the basic elementary school method: The independent variable is changed and the change in the dependent variable is measured. A line of best fit is then drawn through the data or a significance level computed, and a conclusion is reached. These methods are fundamentally limited in their expressiveness and interpretability. Generative modelling allows models to be constructed that are far more complex than simply the independent

and dependent variables, and are imbued with large and highly parameterised latent spaces. Crucially, generative modelling and the Bayesian methodology recovers the distribution over a random variable, as opposed to simply a point estimate of a value with no quantification of uncertainty. Possession of distributions and uncertainties often facilitates more sophisticated analysis. Furthermore, we are even able to recover distributions over the models themselves. Indeed, the work of theoreticians may become markedly less theoretical, as we become able to quantitatively compare and contrast generative models with different latent structures and assumptions on observed data. This generative stance resonates with many researchers' basic scientific impulses: to develop interpretable *explanations* of observed phenomena; as opposed to merely correlating observed quantities.

More broadly, this also highlights a real and immediate opportunity for machine learning to contribute, in a dramatic way, to the practice and scope of neuroscience research. As the connectomes, models and datasets that researchers create and analyse grows, the scope and scale of hypotheses that can be constructed and tested also grows, leading to new opportunities for neuroscientific insight to be garnered, powered by machine learning. Furthermore, inference techniques and methods capable of handling uncertainty may be *required* to analyse partial or approximate connectomes, particularly for those organisms whose connectomes are too vast to fully reconstruct and are not as tightly stereotyped as *C. elegans*. New insights gathered from these larger connectomes may not only unlock neuroscientific understanding at an unprecedented scale and detail, but may also provide new intuitions for designing more effective machine learning methodologies. Beyond this, this challenging and diverse problem domain offers an opportunity for new machine learning methodologies to be developed. Large state spaces, stretching into millions of neurons with complex dependency structures and high computational costs, coupled with the need for interpretability throughout, means that there is an enormous range of tools and techniques that need to be developed to make this modelling tractable.

Fundamental to this work is modelling extremely complex behaviours by modelling micro-scale neural circuits. The atomic interactions between neurons can then be composed to simulate entire assemblies of neurons. This allows the function and properties of individual neurons to be quantitatively related to assembly-level and organism-level behaviours and functions, just as Cajal did in the optic nerve. Critically, this "bottom-up" approach facilitates

analysis at all intermediate scales. Once the individual neurons and the interactions between neurons can be faithfully simulated, the function and interaction of multiple sub-systems at different organisational scales, such as individual synapses, individual neurons, pairs of neurons, assemblies of neurons, cortical columns, or entire brains, can be quantitatively analysed. This work is therefore part of a drive to shift focus from analysing individual neurons and approximate, high-level models, to instead analysing connectome-scale, single-neuron precision, bottom-up models of neural activity.

Beyond this, analysing single specimens is arguably somewhat uninteresting. What we are truly interested in is the variability across specimens. This may be as simplistic as the variation of the activity in individual neurons in response to identical stimuli, but may be as expansive as deducing the distribution over changes to neural assemblies or connectomes in response to learning. While it may be possible to measure the neural state, estimate parameters, or reconstruct connectomes for a single specimen at a single point in time, we need to develop methods for estimating the change in these quantities over time or between specimens. Simultaneously analysing multiple specimens is crucial for analysing inter-specimen interactions and behaviours. Accordingly, imputing the neural activity of potentially hundreds of interacting worms in the same assay is *required* for analysing such behaviours. The data required to perform this analysis is far beyond what will ever likely be afforded by calcium imaging methods, and hence using techniques such as were presented in Chapter 5 are pivotal. Therefore, a key driver behind this work is developing methods and practices, either *in silico* or *in vivo*, that scale to allow multiple specimens to be observed and processed simultaneously without disturbing the organism. This can be seen as part of a wider scaling-up in how ambitious we are in gathering and analysing data, increasing from analysing individual specimens to multiple, potentially hundreds, of specimens in parallel.

We also look to shift focus from studying the *structural* connectome to studying the *functional* connectome. While analysing the morphology of individual neurons or extracting connectomes from large regions of tissue allows quantitative statements about the regularity of connections to be made, it conveys less information about the function and purpose of these connections. It is instead the functional connectome that conveys much of the information required to analyse behaviour. Possession of a high-quality functional connectome simulation will help us answer the “why” and “how,” as opposed to simply the “what” as is afforded

by the structural connectome, and the "where" as is afforded by, for instance, EEG and fMRI. Furthermore, the functional connectome also encompasses the action of neural mechanisms not expressed by the connectivity graph alone, and hence these mechanisms can be investigated and evaluated jointly with connectome-based mechanisms. Although we did not discuss these mechanisms in detail in this thesis, studying the functional connectome in the way that we have presented may provide a tractable method for studying these less easily quantified mechanisms. Therefore full recovery of the functional connectome promises deeper and more complete insights into neural organisation beyond the structural layout, allowing us to study the very formation of behaviour, cognition, and even consciousness itself.

We conclude this thesis with our most blue-sky observation, pertaining to nothing less than fully artificially intelligent lifeforms. Unsurprisingly, consideration of artificially intelligent worms is hardly widespread in philosophical literature or science fiction canon. However, simulated specimens may eventually exhibit learning, hazard avoidance, pain response, mating, and eventually, even death. While it is dubious whether a real worm can be considered as *intelligent*; it is much less dubious, and is maybe even logically correct, to conclude that a simulated organism that is indistinguishable from a real specimen by any conceivable statistical test is *as intelligent as the real specimen*: a "worm Turing test." Arguably therefore, digital *Drosophila*, or even digital mice, are intelligent enough that passing such a Turing test transforms challenging philosophical and theoretical questions long the preserve of deep thinkers across the ages, such as the ethics of artificial intelligence and the origins of consciousness, into practical questions that require answers now.

Bibliography

Aarts, E. and Korst, J. Simulated annealing and Boltzmann machines. 1988.

Abbott, L. F. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999.

Abbott, L. and Kepler, T. B. Model neurons: from Hodgkin-Huxley to Hopfield. In *Statistical mechanics of neural networks*, pp. 5–18. Springer, 1990.

Adhikari, R. and Agrawal, R. K. An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*, 2013.

Adrian, E. D. The impulses produced by sensory nerve endings: Part i. *The Journal of physiology*, 61 (1):49, 1926.

Aicher, C., Foti, N. J., and Fox, E. B. Adaptively truncating backpropagation through time to control gradient bias, 2019.

Aitchison, L., Russell, L., Packer, A. M., Yan, J., Castonguay, P., Hausser, M., and Turaga, S. C. Model-based Bayesian inference of neural activity and connectivity from all-optical interrogation of a neural circuit. In *Advances in Neural Information Processing Systems*, pp. 3486–3495, 2017.

Akerboom, J., Carreras Calderón, N., Tian, L., Wabnig, S., Prigge, M., Tolö, J., Gordus, A., Orger, M., Severi, K., Macklin, J., Patel, R., Pulver, S., Wardill, T., Fischer, E., Schüler, C., Chen, T.-W., Sarkisyan, K., Marvin, J., Bargmann, C., Kim, D., Kügler, S., Lagnado, L., Hegemann, P., Gottschalk, A., Schreiter, E., and Looger, L. Genetically encoded calcium indicators for multi-color neural activity imaging and combination with optogenetics. *Frontiers in Molecular Neuroscience*, 6:2, 2013. ISSN 1662-5099. doi: 10.3389/fnmol.2013.00002.

Ali, F. and Kwan, A. C. Interpreting in vivo calcium signals from neuronal cell bodies, axons, and dendrites: a review. *Neurophotonics*, 7(1):011402, 2019.

Allen, L. J. A primer on stochastic epidemic models: Formulation, numerical simulation, and analysis. *Infectious Disease Modelling*, 2(2):128–142, 2017.

Altekar, G., Dwarkadas, S., Huelsenbeck, J. P., and Ronquist, F. Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference. *Bioinformatics*, 20(3):407–415, 01 2004. ISSN 1367-4803. doi: 10.1093/bioinformatics/btg427.

Altman, N. S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

Altun, Z. and Hall, D. Nervous system, general description. In *WormAtlas*. 2011. doi: 10.3908/wormatlas.1.18.

Amano, S., Kitamura, K., and Hosono, R. Hierarchy of habituation induced by mechanical stimuli in *Caenorhabditis elegans*. *Zoological science*, 16(3):423–429, 1999.

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

Andrieu, C., Doucet, A., and Holenstein, R. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

Angelopoulos, N. and Cussens, J. Bayesian learning of Bayesian networks with informative priors. *Annals of Mathematics and Artificial Intelligence*, 54(1-3):53–98, 2008.

Ankeny, R. A. The natural history of *Caenorhabditis elegans* research. *Nature Reviews Genetics*, 2 (6):474–479, 2001.

Arber, S. Motor circuits in action: specification, connectivity, and function. *Neuron*, 74(6):975–989, 2012.

Archer, E. W., Koster, U., Pillow, J. W., and Macke, J. H. Low-dimensional models of neural population activity in sensory cortical circuits. *Advances in neural information processing systems*, 27:343–351, 2014.

Ardiel, E. L. and Rankin, C. H. An elegant mind: learning and memory in *Caenorhabditis elegans*.

Learning & memory, 17(4):191–201, 2010.

Ardiel, E. L., Alex, J. Y., Giles, A. C., and Rankin, C. H. Habituation as an adaptive shift in response strategy mediated by neuropeptides. *npj Science of Learning*, 2(1):9, 2017.

Ashwood, Z., Roy, N. A., Bak, J. H., and Pillow, J. W. Inferring learning rules from animal decision-making. *Advances in Neural Information Processing Systems*, 33, 2020.

Avery, L. and You, Y. J. *C. elegans* feeding. In *WormBook: The Online Review of* C. elegans *Biology [Internet]*. WormBook, 2012. doi: doi/10.1895/wormbook.1.150.1.

Baek, J.-H., Cosman, P., Feng, Z., Silver, J., and Schafer, W. R. Using machine vision to analyze and classify *Caenorhabditis elegans* behavioral phenotypes quantitatively. *Journal of neuroscience methods*, 118(1):9–21, 2002.

Barber, D. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

Bargmann, C. I. Chemosensation in *C. elegans*. 2006.

Battiti, R. First-and second-order methods for learning: between steepest descent and Newton method. *Neural computation*, 4(2):141–166, 1992.

Bayarri, M. J. and Berger, J. O. The interplay of bayesian and frequentist analysis. *Statistical Science*, pp. 58–80, 2004.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

Beichl, I. and Sullivan, F. The metropolis algorithm. *Computing in Science & Engineering*, 2(1): 65–69, 2000.

Bellucci, A., Mercuri, N. B., Venneri, A., Faustini, G., Longhena, F., Pizzi, M., Missale, C., and Spano, P. Review: Parkinson's disease: from synaptic loss to connectome dysfunction. *Neuropathol. Appl. Neurobiol.*, 42(1):77–94, February 2016.

Benaïm, M., Bouguet, F., Cloez, B., et al. Ergodicity of inhomogeneous markov chains through asymptotic pseudotrajectories. *Annals of Applied Probability*, 27(5):3004–3049, 2017.

Benfenati, F. Synaptic plasticity and the neurobiology of learning and memory. *Acta Biomed*, 78 (Suppl 1):58–66, 2007.

Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, volume 2. Univ. of Texas, 1992.

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

Berlucchi, G. and Buchtel, H. A. Neuronal plasticity: historical roots and evolution of meaning. *Experimental Brain Research*, 192:307–319, 2009.

Berman, G. J., Bialek, W., and Shaevitz, J. W. Predictability and hierarchy in *Drosophila* behavior. *Proceedings of the National Academy of Sciences*, 113(42):11943–11948, 2016.

Berri, S., Boyle, J. H., Tassieri, M., Hope, I. A., and Cohen, N. Forward locomotion of the nematode *C. elegans* is achieved through modulation of a single gait. *HFSP journal*, 3(3):186–193, 2009.

Bertsekas, D. P. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

Besbes, O., Gur, Y., and Zeevi, A. Stochastic multi-armed-bandit problem with non-stationary rewards. *Advances in neural information processing systems*, 27:199–207, 2014.

Bhatt, D. H., Zhang, S., and Gan, W.-B. Dendritic spine dynamics. *Annual Review of Physiology*, 71 (1):261–282, 2009. doi: 10.1146/annurev.physiol.010908.163140. PMID: 19575680.

Bienenstock, E., Cooper, L. N., and Munro, P. Theory of the development of the neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2:32–48, 1982.

Biermann, A. W. Automatic programming: A tutorial on formal methodologies. *Journal of Symbolic Computation*, 1(2):119–142, 1985.

Billard, L. and Diday, E. Symbolic regression analysis. In *Classification, Clustering, and Data Analysis*, pp. 281–288. Springer, 2002.

Bishop, C. M. *Pattern recognition and machine learning*. springer, 2006.

Blei, D. M. Build, compute, critique, repeat: Data analysis with latent variable models. *Annual Review of Statistics and Its Application*, 1:203–232, 2014.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Bogner, A., Jouneau, P.-H., Thollet, G., Basset, D., and Gauthier, C. A history of scanning electron microscopy developments: Towards "wet-stem" imaging. *Micron*, 38(4):390–401, 2007. ISSN 0968-4328. doi: http://dx.doi.org/10.1016/j.micron.2006.06.008. Microscopy of Nanostructures.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Bounoutas, A. and Chalfie, M. Touch sensitivity in *Caenorhabditis elegans*. *Pflügers Archiv-European Journal of Physiology*, 454(5):691–702, 2007.

Box, G. E. Robustness in the strategy of scientific model building. In *Robustness in statistics*, pp. 201–236. Elsevier, 1979.

Boyd, S. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Boyle, J. H., Berri, S., and Cohen, N. Gait modulation in *C. elegans*: an integrated neuromechanical model. *Frontiers in computational neuroscience*, 6:10, 2012.

Bozorgmehr, T., Ardiel, E. L., McEwan, A. H., and Rankin, C. H. Mechanisms of plasticity in a caenorhabditis elegans mechanosensory circuit. *Frontiers in physiology*, 4:88, 2013.

Branco, T., Marra, V., and Staras, K. Examining size-strength relationships at hippocampal synapses using an ultrastructural measurement of synaptic release probability. *J. Struct. Biol.*, 172(2): 203–210, November 2010.

Brand, J. v. d., Peng, B., Song, Z., and Weinstein, O. Training (overparametrized) neural networks in near-linear time. *arXiv preprint arXiv:2006.11648*, 2020.

Brekelmans, R., Masrani, V., Bui, T., Wood, F., Galstyan, A., Steeg, G. V., and Nielsen, F. Annealed importance sampling with q-paths. *arXiv preprint arXiv:2012.07823*, 2020.

Brenner, S. and Wood, W. B. The nematode *Caenorhabditis elegans*. *Cold Spring Harbor Laboratory*, 1:988, 1988.

Brenner, S. Nobel lecture: nature's gift to science. *Bioscience reports*, 23(5-6):225–237, 2003.

Bretscher, A. J., Kodama-Namba, E., Busch, K. E., Murphy, R. J., Soltesz, Z., Laurent, P., and de Bono, M. Temperature, oxygen, and salt-sensing neurons in *C. elegans* are carbon dioxide sensors that control avoidance behavior. *Neuron*, 69(6):1099–1113, 2011.

Brette, R. Philosophy of the spike: Rate-based vs. spike-based theories of the brain. *Frontiers in Systems Neuroscience*, 9:151, 2015. ISSN 1662-5137. doi: 10.3389/fnsys.2015.00151.

Brini, M., Calì, T., Ottolini, D., and Carafoli, E. Neuronal calcium signaling: function and dysfunction. *Cellular and molecular life sciences*, 71(15):2787–2814, 2014.

Brockwell, P. J. and Davis, R. A. *Introduction to time series and forecasting*. springer, 2016.

Brooks, S. Markov chain monte carlo method and its application. *Journal of the royal statistical society: series D (the Statistician)*, 47(1):69–100, 1998.

Bruel-Jungerman, E., Davis, S., and Laroche, S. Brain plasticity mechanisms and memory: a party of four. *Neuroscientist*, 13(5):492–505, October 2007.

Cajal, S. *Comparative study of the sensory areas of the human cortex*. 1899.

Cappé, O., Godsill, S. J., and Moulines, E. An overview of existing methods and recent advances in sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899–924, 2007.

Cauchy, A. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

Chen, X., Feng, X., and Guang, S. Targeted genome engineering in *Caenorhabditis elegans*. *Cell & bioscience*, 6(1):60, 2016.

Chiang, A.-S., Lin, C.-Y., Chuang, C.-C., Chang, H.-M., Hsieh, C.-H., Yeh, C.-W., Shih, C.-T., Wu, J.-J., Wang, G.-T., Chen, Y.-C., Wu, C.-C., Chen, G.-Y., Ching, Y.-T., Lee, P.-C., Lin, C.-Y., Lin, H.-H., Wu, C.-C., Hsu, H.-W., Huang, Y.-A., Chen, J.-Y., Chiang, H.-J., Lu, C.-F., Ni, R.-F., Yeh, C.-Y., and Hwang, J.-K. Three-dimensional reconstruction of brain-wide wiring networks in drosophila at single-cell resolution. *Current Biology*, 21(1):1–11, 2011. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2010.11.056.

Chib, S. and Greenberg, E. Understanding the Metropolis-Hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

Chomsky, N. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

Chopin, N. and Papaspiliopoulos, O. *An introduction to sequential Monte Carlo*. Springer, 2020.

Chung, S. H., Sun, L., and Gabel, C. V. In vivo neuronal calcium imaging in *C. elegans*. *J Vis Exp*, (74), April 2013.

Cimino, G. Reticular theory versus neuron theory in the work of camillo golgi., 1999.

Coenen, A., Fine, E., and Zayachkivska, O. Adolf beck: A forgotten pioneer in electroencephalography. *Journal of the History of the Neurosciences*, 23(3):276–286, 2014. doi: 10.1080/0964704X.2013.867600. PMID: 24735457.

Cohen, N. and Denham, J. E. Whole animal modeling: piecing together nematode locomotion. *Current Opinion in Systems Biology*, 2018.

Cohen, N. and Sanders, T. Nematode locomotion: dissecting the neuronal–environmental loop. *Current opinion in neurobiology*, 25:99–106, 2014.

Coutin, L., Guglielmi, J.-M., and Marie, N. On a fractional stochastic Hodgkin–Huxley model. *International Journal of Biomathematics*, 11(05):1850061, 2018.

Cox, R. T. Probability, frequency and reasonable expectation. *American journal of physics*, 14(1):1–13, 1946.

Cramer, J. V., Gesierich, B., Roth, S., Dichgans, M., Düring, M., and Liesz, A. In vivo widefield calcium imaging of the mouse cortex for analysis of network connectivity in health and brain disease. *Neuroimage*, 199:570–584, 2019.

Cuéllar, M. P., Delgado, M., and Pegalajar, M. An application of non-linear programming to train recurrent neural networks in time series prediction problems. In *Enterprise Information Systems VII*, pp. 95–102. Springer, 2007.

Dana, H., Sun, Y., Mohar, B., Hulse, B. K., Kerlin, A. M., Hasseman, J. P., Tsegaye, G., Tsang, A., Wong, A., Patel, R., et al. High-performance calcium sensors for imaging activity in neuronal populations and microcompartments. *Nature methods*, 16(7):649–657, 2019.

Dangel, F., Kunstner, F., and Hennig, P. Backpack: Packing more into backprop. *arXiv preprint arXiv:1912.10985*, 2019.

David, C. and Kroening, D. Program synthesis: challenges and opportunities. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2104):20150403, 2017.

Dayan, P. and Abbott, L. F. *Theoretical Neuroscience: Computational and mathematical modeling of neural systems*. The MIT Press, Cambridge, Massachusetts, 2001.

DeFelipe, J. Sesquicentenary of the birthday of santiago ramón y cajal, the father of modern neuroscience. *TRENDS in Neurosciences*, 25(9):481–484, 2002.

Del Moral, P. Feynman-Kac formulae. In *Feynman-Kac Formulae*, pp. 47–93. Springer, 2004.

Del Moral, P., Doucet, A., Jasra, A., et al. On adaptive resampling strategies for sequential Monte Carlo methods. *Bernoulli*, 18(1):252–278, 2012.

Denk, W. and Horstmann, H. Serial block-face scanning electron microscopy to reconstruct three-

dimensional tissue nanostructure. *PLoS Biol*, 2(11):1900–1909, 2004.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dhir, N., Perov, Y., Wijers, M., Wood, F., Markham, A., Trethowan, P., du Preez, B., Loveridge, A., and Macdonald, D. Tracking african lions with nonparametric hierarchical models using probabilistic programming. In *Proceedings of the International Society of Bayesian Analysis (ISBA) 2016 World Meeting*, 2016.

Dick, J., Kuo, F. Y., and Sloan, I. H. High-dimensional integration: the quasi-monte carlo way. *Acta Numerica*, 22:133, 2013.

Dickson, B. J. Wired for sex: the neurobiology of *Drosophila* mating decisions. *Science*, 322(5903): 904–909, 2008.

Ding, S. S., Schumacher, L. J., Javer, A. E., Endres, R. G., and Brown, A. E. Shared behavioral mechanisms underlie *C. elegans* aggregation and swarming. *eLife*, 8:e43318, 2019.

Domjan, M. *The Principles of Learning and Behaviour*. Cengage Learning, Stamford, Connecticut, 2015.

Doty, R. W. Consciousness from neurons. *Acta neurobiologiae experimentalis*, 35(5-6):791–804, 1975.

Doucet, A. and Johansen, A. A tutorial on particle filtering and smoothing: fifteen years later, 2010.

Doucet, A., De Freitas, N., and Gordon, N. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo methods in practice*, pp. 3–14. Springer, 2001.

Doucet, A., Godsill, S. J., and Robert, C. P. Marginal maximum a posteriori estimation using Markov chain Monte Carlo. *Statistics and Computing*, 12(1):77–84, 2002.

Drton, M. and Maathuis, M. H. Structure learning in graphical modeling. *Annual Review of Statistics and Its Application*, 4(1):365–393, 2017. doi: 10.1146/annurev-statistics-060116-053803.

Du, N. H. and Sam, V. H. Dynamics of a stochastic Lotka–Volterra model perturbed by white noise. *Journal of mathematical analysis and applications*, 324(1):82–97, 2006.

Dzamba, D., Harantova, L., Butenko, O., and Anderova, M. Glial cells–the key elements of alzheimer s disease. *Current Alzheimer Research*, 13(8):894–911, 2016.

Earl, D. J. and Deem, M. W. Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, 7(23):3910–3916, 2005.

Edwards, J. S. and Huntford, R. Fridtiof Nansen: from the neuron to the north polar sea. *Endeavour*, 22(2):76–80, 1998.

Edwards, N. R., Cameron, D., and Rougier, J. Precalibrating an intermediate complexity climate model. *Climate dynamics*, 37(7-8):1469–1482, 2011.

Einevoll, G. T., Destexhe, A., Diesmann, M., Grün, S., Jirsa, V., de Kamps, M., Migliore, M., Ness, T. V., Plesser, H. E., and Schürmann, F. The scientific case for brain simulations. *Neuron*, 102(4): 735 – 744, 2019. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2019.03.027.

Eliasmith, C. *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.

Eliasmith, C. and Trujillo, O. The use and abuse of large-scale brain models. *Current opinion in neurobiology*, 25:1–6, 2014.

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

Ellis, K., Nye, M., Pu, Y., Sosa, F., Tenenbaum, J., and Solar-Lezama, A. Write, execute, assess: Program synthesis with a repl, 2019.

Fallon, M. F., Johannsson, H., and Leonard, J. J. Efficient scene simulation for robust Monte Carlo localization using an rgb-d camera. In *2012 IEEE international conference on robotics and automation*, pp. 1663–1670. IEEE, 2012.

Fang-Yen, C., Wyart, M., Xie, J., Kawai, R., Kodger, T., Chen, S., Wen, Q., and Samuel, A. D. T.

Biomechanical analysis of gait adaptation in the nematode *Caenorhabditis elegans*. *Proceedings of the National Academy of Sciences*, 107(47):20323–20328, 2010. ISSN 0027-8424. doi: 10.1073/pnas.1003016107.

Fernández-Godino, M. G., Park, C., Kim, N.-H., and Haftka, R. T. Review of multi-fidelity models. *arXiv preprint arXiv:1609.07196*, 2016.

Feurer, M. and Hutter, F. Towards further automation in automl. In *ICML AutoML workshop*, pp. 13, 2018.

Fieseler, C., Zimmer, M., and Kutz, J. N. Unsupervised learning of control signals and their encodings in *Caenorhabditis elegans* whole-brain recordings. *J R Soc Interface*, 17(173):20200459, Dec 2020.

Finger, S. *Origins of Neuroscience: A History of Explorations Into Brain Function*. Oxford University Press paperback. Oxford University Press, 2001. ISBN 9780195146943.

Flanders, H. Differentiation under the integral sign. *The American Mathematical Monthly*, 80(6): 615–627, 1973.

Fouad, A. D., Teng, S., Mark, J. R., Liu, A., Alvarez-Illera, P., Ji, H., Du, A., Bhirgoo, P. D., Cornblath, E., Guan, S. A., et al. Distributed rhythm generators underlie *Caenorhabditis elegans* forward locomotion. *Elife*, 7:e29913, 2018.

Fox, R. F. Stochastic versions of the Hodgkin-Huxley equations. *Biophysical journal*, 72(5):2068–2074, 1997.

Frézal, L. and Félix, M.-A. The natural history of model organisms: *C. elegans* outside the petri dish. *Elife*, 4:e05849, 2015.

Friedrich, J., Zhou, P., and Paninski, L. Fast online deconvolution of calcium imaging data. *PLOS Computational Biology*, 13(3):1–26, 03 2017. doi: 10.1371/journal.pcbi.1005423.

Gao, S., Guan, S. A., Fouad, A. D., Meng, J., Huang, Y.-C., Li, Y., Alcaire, S., Hung, W., Kawano, T., Lu, Y., et al. Excitatory motor neurons are local central pattern generators in an anatomically compressed motor circuit for reverse locomotion. *bioRxiv*, pp. 135418, 2017.

Garson, J. The birth of information in the brain: Edgar Adrian and the vacuum tube. *Science in Context*, 28(1):31–52, 2015.

Gauthier, J. L., Koay, S. A., Nieh, E. H., Tank, D. W., Pillow, J. W., and Charles, A. S. Detecting and correcting false transients in calcium imaging. *bioRxiv*, pp. 473470, 2018.

Geduldig, D. and Junge, D. Sodium and calcium components of action potentials in Aplysia giant neurone. *The Journal of physiology*, 199(2):347–365, 1968.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. *Bayesian data analysis*. CRC press, 2013.

Gelman, A. et al. Objections to bayesian statistics. *Bayesian Analysis*, 3(3):445–449, 2008.

Gerkin, R. C., Jarvis, R. J., and Crook, S. M. Towards systematic, data-driven validation of a collaborative, multi-scale model of *Caenorhabditis elegans*. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1758):20170381, 2018.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889, 2015.

Gershman, S. and Goodman, N. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

Gerstner, W. *Biological Learning: Synaptic Plasticity, Hebb Rule and Spike Timing Dependent Plasticity*, pp. 140–143. Springer US, Boston, MA, 2017. ISBN 978-1-4899-7687-1. doi: 10.1007/978-1-4899-7687-1_80.

Gewaltig, M.-O. and Diesmann, M. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007.

Ghahramani, Z. Automating machine learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9852:XX, 2016.

Gilks, W. R. and Berzuini, C. Following a moving target—Monte Carlo inference for dynamic

Bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63 (1):127–146, 2001.

Giovannucci, A., Friedrich, J., Gunn, P., Kalfon, J., Brown, B. L., Koay, S. A., Taxidis, J., Najafi, F., Gauthier, J. L., Zhou, P., et al. CaImAn an open source tool for scalable calcium imaging data analysis. *Elife*, 8:e38173, 2019.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

Glasgow, S. D., McPhedrain, R., Madranges, J. F., Kennedy, T. E., and Ruthazer, E. S. Approaches and limitations in the investigation of synaptic transmission and plasticity. *Frontiers in synaptic neuroscience*, 11:20, 2019.

Gleeson, P., Lung, D., Grosu, R., Hasani, R., and Larson, S. D. c302: a multiscale framework for modelling the nervous system of *Caenorhabditis elegans*. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1758):20170379, 2018.

Glover, G. H. Overview of functional magnetic resonance imaging. *Neurosurgery Clinics*, 22(2): 133–139, 2011.

Goldwyn, J. H. and Shea-Brown, E. The what and where of adding channel noise to the Hodgkin-Huxley equations. *PLOS Computational Biology*, 7(11):1–9, 11 2011.

Golgi, C. Sulla struttura della sostanza grigia del cervello. *Gazzetta Medica Italiana. Lombardia*, 33: 244–246, 1873.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Goodman, M. B., Hall, D. H., Avery, L., and Lockery, S. R. Active currents regulate sensitivity and dynamic range in *C. elegans* neurons. *Neuron*, 20(4):763–772, 1998.

Gordus, A., Pokala, N., Levy, S., Flavell, S. W., and Bargmann, C. I. Feedback from network states generates variability in a probabilistic olfactory circuit. *Cell*, 161(2):215–227, 2015.

Graham, R. L., Woodall, T. S., and Squyres, J. M. Open mpi: A flexible high performance mpi. In *International Conference on Parallel Processing and Applied Mathematics*, pp. 228–239. Springer, 2005.

Gray Roncal, W., Kleissas, D. M., Vogelstein, J. T., Manavalan, P., Lillaney, K., Pekala, M., Burns, R., Vogelstein, R. J., Priebe, C. E., Chevillet, M. A., and Hager, G. D. An Automated Images-to-Graphs Framework for High Resolution Connectomics. *ArXiv e-prints*, November 2014.

Grienberger, C. and Konnerth, A. Imaging calcium in neurons. *Neuron*, 73(5):862–885, 2012.

Gruninger, T. R., Gualberto, D. G., LeBoeuf, B., and Garcia, L. R. Integration of male mating and feeding behaviors in *Caenorhabditis elegans*. *Journal of Neuroscience*, 26(1):169–179, 2006.

Gu, S. S., Ghahramani, Z., and Turner, R. E. Neural adaptive sequential Monte Carlo. *Advances in neural information processing systems*, 28:2629–2637, 2015.

Gulwani, S., Polozov, O., and Singh, R. Program synthesis. In *Foundations and Trends in Programming Languages*, volume 4, pp. 1–119. now Publishing Inc., 2017.

Ha, D., Dai, A., and Le, Q. V. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Haario, H., Saksman, E., and Tamminen, J. Componentwise adaptation for high dimensional MCMC. *Computational Statistics*, 20(2):265–273, 2005.

Haas, J. S. A new measure for the strength of electrical synapses. *Frontiers in cellular neuroscience*, 9:378, 2015.

Halmos, P. R. *Measure theory*, volume 18. Springer, 2013.

Hart, A. C. Behavior. In *WormBook: The Online Review of* C. elegans *Biology [Internet]*. WormBook, 2006. doi: doi/10.1895/wormbook.1.87.1.

Hasani, R. M., Beneder, V., Fuchs, M., Lung, D., and Grosu, R. SIM-CE: An advanced simulink platform for studying the brain of *Caenorhabditis elegans*. *arXiv preprint arXiv:1703.06270*, 2017.

Hastings, W. K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

Hebb, D. O. *The organization of behavior: A neuropsychological theory*. Psychology Press, 1949.

Helmstaedter, M., Briggman, K. L., and Denk, W. 3d structural imaging of the brain with photons and electrons. *Current Opinion in Neurobiology*, 18(6):633–641, 2008.

Heng, J., Bishop, A. N., Deligiannidis, G., Doucet, A., et al. Controlled sequential Monte Carlo. *Annals of Statistics*, 48(5):2904–2929, 2020.

Herlihy, M. and Shavit, N. *The Art of Multiprocessor Programming, Revised Reprint*. Elsevier Science, 2012. ISBN 9780123977953.

Hertz, J. A. *Introduction to the theory of neural computation*. CRC Press, 2018.

Hildebrand, F. B. *Introduction to numerical analysis*. Courier Corporation, 1987.

Hill, A. V. The heat of shortening and the dynamic constants of muscle. *Proc. R. Soc. Lond. B*, 126 (843):136–195, 1938.

Hines, M. and Carnevale, N. The NEURON simulation environment. *NEURON*, 9(6), 2006.

Ho, T. K. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pp. 278–282. IEEE, 1995.

Hodgkin, A. L. and Huxley, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

Hope, I. A. C. elegans*: a practical approach*, volume 213. OUP Oxford, 1999.

Hubel, D. and Wiesel, T. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

Husson, S. J., Costa, W. S., Schmitt, C., and Gottschalk, A. Keeping track of worm trackers. In *WormBook: The Online Review of* C. elegans *Biology [Internet]*. WormBook, 2005.

Huttenlocher, P. R. *Neural plasticity*. Harvard University Press, 2009.

Hutter, F., Kotthoff, L., and Vanschoren, J. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

Hyman, S. E., Malenka, R. C., and Nestler, E. J. Neural mechanisms of addiction: the role of reward-related learning and memory. *Annu. Rev. Neurosci.*, 29:565–598, 2006.

Inoue, M., Takeuchi, A., Manita, S., ichiro Horigane, S., Sakamoto, M., Kawakami, R., Yamaguchi, K., Otomo, K., Yokoyama, H., Kim, R., Yokoyama, T., Takemoto-Kimura, S., Abe, M., Okamura, M., Kondo, Y., Quirin, S., Ramakrishnan, C., Imamura, T., Sakimura, K., Nemoto, T., Kano, M., Fujii, H., Deisseroth, K., Kitamura, K., and Bito, H. Rational engineering of XCaMPs, a multicolor GECI suite for in vivo imaging of complex brain circuit dynamics. *Cell*, 177(5):1346 – 1360.e24, 2019. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2019.04.007.

Izquierdo, E. J. and Beer, R. D. Connecting a connectome to behavior: an ensemble of neuroanatomical models of *C. elegans* klinotaxis. *PLoS computational biology*, 9(2):e1002890, 2013.

Jain, P. and Kar, P. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3-4):142–363, 2017. ISSN 1935-8237. doi: 10.1561/2200000058.

Jain, V., Seung, H. S., and Turaga, S. C. Machines that learn to segment images: a crucial technology for connectomics. *Curr. Opin. Neurobiol.*, 20(5):653–666, October 2010.

Janz, D., Paige, B., Rainforth, T., van de Meent, J.-W., and Wood, F. Probabilistic structure discovery in time series data. *arXiv preprint arXiv:1611.06863*, 2016.

Javer, A., Currie, M., Lee, C. W., Hokanson, J., Li, K., Martineau, C. N., Yemini, E., Grundy, L. J., Li, C., Ch'ng, Q., et al. An open-source platform for analyzing and sharing worm-behavior data. *Nature methods*, 15(9):645, 2018.

Jaynes, E. T. *Probability theory: the logic of science*. 1995.

Jebara, T. *Machine learning: discriminative and generative*, volume 755. Springer Science & Business Media, 2012.

Jordan, J., Schmidt, M., Senn, W., and Petrovici, M. A. Evolving to learn: discovering interpretable plasticity rules for spiking networks, 2021.

Jordan, M. I. Are you a bayesian or a frequentist? 2009.

Kalman, R. E. et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

Kantas, N., Doucet, A., Singh, S. S., and Maciejowski, J. M. An overview of sequential Monte Carlo methods for parameter estimation in general state-space models. *IFAC Proceedings Volumes*, 42 (10):774–785, 2009.

Kantas, N., Doucet, A., Singh, S. S., Maciejowski, J., Chopin, N., et al. On particle methods for parameter estimation in state-space models. *Statistical science*, 30(3):328–351, 2015.

Karagiannis, G. and Andrieu, C. Annealed importance sampling reversible jump mcmc algorithms. *Journal of Computational and Graphical Statistics*, 22(3):623–648, 2013.

Kasthuri, N., Hayworth, K. J., and Berger, D. R. e. a. Saturated Reconstruction of a Volume of Neocortex. *Cell*, 162(3):648–661, July 2015.

Kato, S., Kaplan, H. S., Schrödel, T., Skora, S., Lindsay, T. H., Yemini, E., Lockery, S., and Zimmer, M. Global brain dynamics embed the motor command sequence of *Caenorhabditis elegans*. *Cell*, 163(3):656–669, 2015.

Keshavan, A. and Poline, J.-B. From the wet lab to the web lab: a paradigm shift in brain imaging research. *Frontiers in Neuroinformatics*, 13:3, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10215–10224, 2018.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pp. 4743–4751, 2016.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *science*, 220 (4598):671–680, 1983.

Kita, J. M. and Wightman, R. M. Microelectrodes for studying neurobiology. *Current opinion in chemical biology*, 12(5):491–496, 2008.

Kitagawa, G. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of computational and graphical statistics*, 5(1):1–25, 1996.

Kitzelmann, E. Inductive programming: A survey of program synthesis techniques. In Schmid, U., Kitzelmann, E., and Plasmeijer, R. (eds.), *Approaches and Applications of Inductive Programming*, pp. 50–73, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-11931-6.

Knöpfel, T. and Song, C. Optical voltage imaging in neurons: moving from technology development to practical tool. *Nature Reviews Neuroscience*, pp. 1–9, 2019.

Knott, G., Marchman, H., Wall, D., and Lich, B. Serial section scanning electron microscopy of adult brain tissue using focused ion beam milling. *The Journal of Neuroscience*, 28(12):2959–2964, 2008.

Kobyzev, I., Prince, S., and Brubaker, M. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Koller, D. and Friedman, N. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Kolmogorov, A. N. and Bharucha-Reid, A. T. *Foundations of the theory of probability: Second*

English Edition. Courier Dover Publications, 2018.

Koubeissi, M. Z., Bartolomei, F., Beltagy, A., and Picard, F. Electrical stimulation of a small brain area reversibly disrupts consciousness. *Epilepsy & Behavior*, 37:32 – 35, 2014. ISSN 1525-5050. doi: https://doi.org/10.1016/j.yebeh.2014.05.027.

Kristan, W. B. and Katz, P. Form and function in systems neuroscience. *Current biology*, 16(19): R828–R831, 2006.

Krizhevsky, A., Sutskever, I., and Hinton, G. ImageNet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *NIPS 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

Kroese, D. P., Brereton, T., Taimre, T., and Botev, Z. I. Why the Monte Carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):386–392, 2014.

Kulkarni, V. Y. and Sinha, P. K. Pruning of random forest classifiers: A survey and future directions. In *2012 International Conference on Data Science & Engineering (ICDSE)*, pp. 64–68. IEEE, 2012.

Kunert, J., Shlizerman, E., and Kutz, J. N. Low-dimensional functionality of complex network dynamics: Neurosensory integration in the *Caenorhabditis elegans* connectome. *Physical Review E*, 89(5):052805, 2014.

Kunert, J. M., Proctor, J. L., Brunton, S. L., and Kutz, J. N. Spatiotemporal feedback and network structure drive and encode *Caenorhabditis elegans* locomotion. *PLoS computational biology*, 13(1): e1005303, 2017.

Kuramochi, M. and Doi, M. A computational model based on multi-regional calcium imaging represents the spatio-temporal dynamics in a *Caenorhabditis elegans* sensory neuron. *PLOS ONE*, 12(1):1–19, 01 2017. doi: 10.1371/journal.pone.0168415.

Kwong, K. K., Belliveau, J. W., Chesler, D. A., Goldberg, I. E., Weisskoff, R. M., Poncelet, B. P., Kennedy, D. N., Hoppel, B. E., Cohen, M. S., and Turner, R. Dynamic magnetic resonance imaging of human brain activity during primary sensory stimulation. *Proceedings of the National Academy of Sciences*, 89(12):5675–5679, 1992.

Łącki, M. K. and Miasojedow, B. State-dependent swap strategies and automatic reduction of number of temperatures in adaptive parallel tempering algorithm. *Statistics and Computing*, 26(5):951–964, 2016.

Larochelle, H. and Murray, I. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 29–37, 2011.

Larson, S. D., Gleeson, P., and Brown, A. E. Connectome to behaviour: modelling *Caenorhabditis elegans* at cellular resolution, 2018.

Le, T. A., Baydin, A. G., and Wood, F. Inference compilation and universal probabilistic programming. *arXiv preprint arXiv:1610.09900*, 2016.

Le, T. A., Baydin, A. G., and Wood, F. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1338–1348, Fort Lauderdale, FL, USA, 2017. PMLR.

Le, T. A., Igl, M., Rainforth, T., Jin, T., and Wood, F. Auto-encoding sequential Monte Carlo, 2018.

Lebois, F., Sauvage, P., Py, C., Cardoso, O., Ladoux, B., Hersen, P., and Di Meglio, J.-M. Locomotion control of *Caenorhabditis elegans* through confinement. *Biophysical journal*, 102(12):2791–2798, 2012.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 0028-0836. Insight.

Leighton, S. B. Sem images of block faces, cut by a miniature microtome within the sem-a technical note. *Scanning electron microscopy*, (Pt 2):73–76, 1980.

Lemaréchal, C. Cauchy and the gradient method. *Doc Math Extra*, 251:254, 2012.

Lichtman, J. and Denk, W. The big and the small: challenges of imaging the brain's circuits. *Science*,

334(6056):618–623, 2011.

Lim, S., McKee, J. L., Woloszyn, L., Amit, Y., Freedman, D. J., Sheinberg, D. L., and Brunel, N. Inferring learning rules from distributions of firing rates in cortical neurons. *Nature neuroscience*, 18(12):1804–1810, 2015.

Lin, M. Z. and Schnitzer, M. J. Genetically encoded indicators of neuronal activity. *Nature neuroscience*, 19(9):1142, 2016.

Linderman, S., Nichols, A., Blei, D., Zimmer, M., and Paninski, L. Hierarchical recurrent state space models reveal discrete and continuous dynamics of neural activity in *C. elegans*. *bioRxiv preprint bioRxiv:621540*, 2019a.

Linderman, S. W., Mena, G. E., Cooper, H., Paninski, L., and Cunningham, J. P. Reparameterizing the Birkhoff polytope for variational permutation inference. *arXiv preprint arXiv:1710.09508*, 2017.

Linderman, S. W., Nichols, A. L., Blei, D. M., Zimmer, M., and Paninski, L. Hierarchical recurrent state space models reveal discrete and continuous dynamics of neural activity in *C. elegans*. *bioRxiv*, pp. 621540, 2019b.

Lindsley, D. B. Psychological phenomena and the electroencephalogram. *Electroencephalography and clinical neurophysiology*, 4(4):443–456, 1952.

Linhart, H. and Zucchini, W. *Model selection*. John Wiley & Sons, 1986.

Liu, P., Chen, B., and Wang, Z.-W. Gap junctions synchronize action potentials and Ca2+ transients in *Caenorhabditis elegans* body wall muscle. *Journal of Biological Chemistry*, 286(51):44285–44293, 2011.

Lockery, S. R. and Goodman, M. B. The quest for action potentials in *C. elegans* neurons hits a plateau. *Nature neuroscience*, 12(4):377, 2009.

Lockery, S. and Sejnowski, T. Distributed processing of sensory information in the leech. iii. a dynamical neural network model of the local bending reflex. *Journal of Neuroscience*, 12(10): 3877–3895, 1992.

Loewenstein, Y., Yanover, U., and Rumpel, S. Predicting the dynamics of network connectivity in the neocortex. *Journal of Neuroscience*, 35(36):12535–12544, 2015. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.2917-14.2015.

Lokhorst, G.-J. C. and Kaitaro, T. T. The originality of descartes theory about the pineal gland. *Journal of the History of the Neurosciences*, 10(1):6–18, 2001.

Lopes, H. F. and Tsay, R. S. Particle filters and Bayesian inference in financial econometrics. *Journal of Forecasting*, 30(1):168–209, 2011. doi: 10.1002/for.1195.

Lucas, D., Klein, R., Tannahill, J., Ivanova, D., Brandon, S., Domyancic, D., and Zhang, Y. Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4):1157–1171, 2013.

Lueckmann, J.-M., Bassetto, G., Karaletsos, T., and Macke, J. H. Likelihood-free inference with emulator networks. In *Symposium on Advances in Approximate Bayesian Inference*, pp. 32–53, 2019.

Lv, Q., Schneider, M. K., and Pitchford, J. W. Individualism in plant populations: Using stochastic differential equations to model individual neighbourhood-dependent plant growth. *Theoretical Population Biology*, 74(1):74–83, 2008. ISSN 0040-5809. doi: https://doi.org/10.1016/j.tpb.2008.05.003.

MacKay, D. J. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Maclaurin, D. and Adams, R. P. Firefly Monte Carlo: Exact MCMC with subsets of data. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

Maley, C. J. Continuous neural spikes and information theory. *Review of Philosophy and Psychology*, pp. 1–21, 2018.

Manor, Y., Nadim, F., Abbott, L. F., and Marder, E. Temporal dynamics of graded synaptic

transmission in the lobster stomatogastric ganglion. *Journal of Neuroscience*, 17(14):5610–5621, 1997. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.17-14-05610.1997.

Marblestone, A. Simple *C. elegans*. https://https://github.com/adammarblestone/simple-C-elegans, 2016.

Markram, H. The blue brain project. *Nat Rev Neurosci*, 7(2):153–160, February 2006. ISSN 1471-003X. doi: 10.1038/nrn1848.

Markram, H., Gerstner, W., and Sjöström, P. J. Spike-timing-dependent plasticity: a comprehensive overview. *Frontiers in synaptic neuroscience*, 4:2, 2012.

Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., Ailamaki, A., Alonso-Nanclares, L., Antille, N., Arsever, S., et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

Mateos-Aparicio, P. and Rodríguez-Moreno, A. The impact of studying brain plasticity. *Frontiers in Cellular Neuroscience*, 13:66, 2019. ISSN 1662-5102. doi: 10.3389/fncel.2019.00066.

Mayford, M., Siegelbaum, S., and Kandel, E. Synapses and memory storage. *Cold Spring Harbor perspectives in biology*, 4(6):a005751, 2012.

Mbalawata, I. S., Särkkä, S., and Haario, H. Parameter estimation in stochastic differential equations with Markov chain Monte Carlo and non-linear Kalman filtering. *Computational Statistics*, 28(3): 1195–1223, Jun 2013. ISSN 1613-9658. doi: 10.1007/s00180-012-0352-y.

McDiarmid, T. A., Ardiel, E. L., and Rankin, C. H. The role of neuropeptides in learning and memory in *Caenorhabditis elegans*. *Current Opinion in Behavioral Sciences*, 2:15–20, 2015.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learning latent permutations with Gumbel-Sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

Metaxakis, A., Petratou, D., and Tavernarakis, N. Multimodal sensory processing in *Caenorhabditis elegans*. *Open biology*, 8(6):180049, 2018.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Meyer, D., Bonhoeffer, T., and Scheuss, V. Balance and stability of synaptic structures during synaptic plasticity. *Neuron*, 82(2):430–443, 2014.

Miasojedow, B., Moulines, E., and Vihola, M. An adaptive parallel tempering algorithm. *Journal of Computational and Graphical Statistics*, 22(3):649–664, 2013.

Mikula, S., Binding, J., and Denk, W. Staining and embedding the whole mouse brain for electron microscopy. *Nature methods*, 9(12):1198–1201, 2012.

Mishchencko, Y., Vogelstein, J. T., and Paninski, L. A Bayesian approach for inferring neuronal connectivity from calcium fluorescent imaging data. *The Annals of Applied Statistics*, 5:1229–1261, 2011.

Mockus, J. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

Molleman, A. *Patch clamping: an introductory guide to patch clamp electrophysiology*. John Wiley & Sons, 2003.

Møller, J. K., Madsen, H., and Carstensen, J. Parameter estimation in a simple stochastic differential equation for phytoplankton modelling. *Ecological modelling*, 222(11):1793–1799, 2011.

Murray, I. and Ghahramani, Z. Bayesian learning in undirected graphical models: approximate mcmc algorithms. *arXiv preprint arXiv:1207.4134*, 2012.

Naesseth, C. A., Linderman, S. W., Ranganath, R., and Blei, D. M. Variational sequential Monte Carlo, 2018.

Nagy, T., Tóth, J., and Ladics, T. Automatic model generation, 2019.

Nasser, I. M. and Abu-Naser, S. S. Lung cancer detection using artificial neural network. *International Journal of Engineering and Information Systems (IJEAIS)*, 3(3):17–23, 2019.

Neal, R. M. *Probabilistic inference using Markov chain Monte Carlo methods*. 1993.

Neal, R. M. Annealed importance sampling. *Statistics and computing*, 11(2):125–139, 2001.

Neher, E. and Sakmann, B. The patch clamp technique. *Scientific American*, 266(3):44–51, 1992.

Nelder, J. A. and Mead, R. A simplex method for function minimization. *The computer journal*, 7(4): 308–313, 1965.

Newman, G. R. and Hobot, J. A. Resins for combined light and electron microscopy: a half century of development. *The Histochemical Journal*, 31(8):495–505, 1999.

Newton, I. *Philosophiae naturalis principia mathematica*, volume 3. Apud Guil. & Joh. Innys, Regiæ Societatis typographos, 1726.

Ng, A. Y. and Jordan, M. I. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in neural information processing systems*, pp. 841–848, 2002.

Nguyen, J. P., Shipley, F. B., Linder, A. N., Plummer, G. S., Liu, M., Setru, S. U., Shaevitz, J. W., and Leifer, A. M. Whole-brain calcium imaging with cellular resolution in freely behaving *Caenorhabditis elegans*. *Proceedings of the National Academy of Sciences*, 113(8):E1074–E1081, 2016.

Nigon, V. M. and Félix, M.-A. History of research on *C. elegans* and other free-living nematodes as model organisms. In *WormBook: The Online Review of* C. elegans *Biology [Internet]*. WormBook, 2018.

Obien, M. E. J., Deligkaris, K., Bullmann, T., Bakkum, D. J., and Frey, U. Revealing neuronal function through microelectrode array recordings. *Frontiers in neuroscience*, 8:423, 2015.

Oh, G. and Valois, J.-S. HCNAF: Hyper-conditioned neural autoregressive flow and its application for probabilistic occupancy map forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14550–14559, 2020.

Oja, E. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

Olivares, E. O., Izquierdo, E. J., and Beer, R. D. Potential role of a ventral nerve cord central pattern generator in forward and backward locomotion in *Caenorhabditis elegans*. *Network Neuroscience*, 2(3):323–343, 2018.

Ondrúśka, P., Gurău, C., Marchegiani, L., Tong, C. H., and Posner, I. Scheduled perception for energy-efficient path following. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4799–4806, May 2015. doi: 10.1109/ICRA.2015.7139866.

OpenWorm. c302. `https://github.com/openworm/c302`, 2018. Accessed: 2018-05-15.

Ore, O. Pascal and the invention of probability theory. *The American Mathematical Monthly*, 67(5): 409–419, 1960. ISSN 00029890, 19300972.

Orloff, J. and Bloom, J. Comparison of frequentist and bayesian inference, 2014.

Owen, A. B. *Monte Carlo theory, methods and examples*. 2013.

Palyanov, A. Y. and Khayrulin, S. S. Sibernetic: A software complex based on the pci sph algorithm aimed at simulation problems in biomechanics. *Russian Journal of Genetics: Applied Research*, 5 (6):635–641, November 2015. doi: 10.1134/S2079059715060052.

Palyanov, A., Khayrulin, S., and Larson, S. D. Application of smoothed particle hydrodynamics to modeling mechanisms of biological tissue. *Advances in Engineering Software*, 98:1–11, 2016. ISSN 18735339. doi: 10.1016/j.advengsoft.2016.03.002.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

Papamakarios, G., Sterratt, D., and Murray, I. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 837–848, 2019.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Pearce, J. M. Marie-jean-pierre flourens (1794–1867) and cortical localization. *European neurology*, 61(5):311–314, 2009.

Peherstorfer, B., Willcox, K., and Gunzburger, M. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. *Siam Review*, 60(3):550–591, 2018.

Perdikaris, P., Raissi, M., Damianou, A., Lawrence, N. D., and Karniadakis, G. E. Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2198):20160751, 2017.

Peterka, D. S., Takahashi, H., and Yuste, R. Imaging voltage in neurons. *Neuron*, 69(1):9–21, 2011.

Pfeffer, A. *Practical probabilistic programming*. Manning Publications Co., 2016.

Piccolino, M. and Bresadola, M. *Shocking frogs: Galvani, Volta, and the electric origins of neuroscience*. OUP Us, 2013.

Picconi, B., Piccoli, G., and Calabresi, P. Synaptic dysfunction in parkinson's disease. *Synaptic Plasticity*, pp. 553–572, 2012.

Pimblott, S. M. and LaVerne, J. A. Comparison of stochastic and deterministic methods for modeling spur kinetics. *Radiation Research*, 122(1):12–23, 1990. ISSN 00337587, 19385404.

Popper, K. *The logic of scientific discovery*. Routledge, 1959.

Prevedel, R. Large-scale fluorescence imaging in neuroscience. *Imaging from Cells to Animals In Vivo*, pp. 337, 2020.

Quiroga, R. Q., Reddy, L., Kreiman, G., Koch, C., and Fried, I. Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107, 2005.

Rahmati, V., Kirmse, K., Marković, D., Holthoff, K., and Kiebel, S. J. Inferring neuronal dynamics from calcium imaging data using biophysical models and Bayesian inference. *PLoS computational biology*, 12(2):e1004736, 2016.

Rainforth, T., van de Meent, J., Osborne, M., and Wood, F. Bayesian optimization for probabilistic programs. *1st NIPS Workshop on Black Box Learning and Inference*, pp. 280–288, 2015.

Rainforth, T. W. G. *Automating inference, learning, and design using probabilistic programming*. PhD thesis, University of Oxford, 2017.

Rainforth, T. and Wood, F. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*, 2015.

Rainforth, T., Cornish, R., Yang, H., Warrington, A., and Wood, F. On nesting Monte Carlo estimators. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4267–4276, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Ramot, D., Johnson, B. E., Berry Jr, T. L., Carnell, L., and Goodman, M. B. The parallel worm tracker: a platform for measuring average speed and drug-induced paralysis in nematodes. *PloS one*, 3(5):e2208, 2008.

Randi, F. and Leifer, A. M. Measuring and modeling whole-brain neural dynamics in caenorhabditis elegans. *Current Opinion in Neurobiology*, 65:167–175, 2020.

Rasmussen, C. E. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pp. 63–71. Springer, 2003.

Reddy, K. and Clinton, V. Simulating stock prices using geometric Brownian motion: Evidence from australian companies. *Australasian Accounting, Business and Finance Journal*, 10(3):23–47, 2016.

Renard, P., Alcolea, A., and Gingsbourger, D. Stochastic versus deterministic approaches. In *Environmental Modelling: Finding Simplicity in Complexity, Second Edition (eds J. Wainwright and M. Mulligan)*, pp. 133–149. Wiley Online Library, 2013.

Reynolds, D. A. Gaussian mixture models. *Encyclopedia of biometrics*, 741, 2009.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows, 2015.

Richmond, J. Synaptic function. In *WormBook: The Online Review of* C. elegans *Biology [Internet]*. WormBook, 2007.

Riddle, D. L., Blumenthal, T., and Meyer, B. J. Section iii, defecation motor program. In C. elegans *II, second edition*. Cold Spring Harbor (NY), 1997a.

Riddle, D. L., Blumenthal, T., Meyer, B. J., and Priess, J. R. C. elegans ii. 1997b.

Rish, I. et al. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pp. 41–46, 2001.

Ritchie, D., Mildenhall, B., Goodman, N. D., and Hanrahan, P. Controlling procedural modeling programs with stochastically-ordered sequential Monte Carlo. *ACM Transactions on Graphics (TOG)*, 34(4):105, 2015.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Robert, C. and Casella, G. A short history of markov chain monte carlo: Subjective recollections from incomplete data. *Statistical Science*, pp. 102–115, 2011.

Robert, C. P. and Casella, G. The Metropolis—Hastings algorithm. In *Monte Carlo Statistical Methods*, pp. 231–283. Springer, 1999.

Rong, G., Kim, E. H., Qiang, Y., Di, W., Zhong, Y., Zhao, X., Fang, H., and Clark, H. A. Imaging sodium flux during action potentials in neurons with fluorescent nanosensors and transparent microelectrodes. *ACS sensors*, 3(12):2499–2505, 2018.

Ronneberger, O., Fischer, P., and Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, pp. 234–241. Springer International Publishing, Cham, 2015. ISBN 978-3-319-24574-4.

Rose, J. K. and Rankin, C. H. Analyses of habituation in *Caenorhabditis elegans*. *Learning & Memory*, 8(2):63–69, 2001.

Roy, N. A., Bak, J. H., Akrami, A., Brody, C., and Pillow, J. W. Efficient inference for time-varying behavior during learning. In *Advances in neural information processing systems*, pp. 5695–5705, 2018.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Ruffio, E., Saury, D., Petit, D., and Girault, M. Tutorial 2: Zero-order optimization algorithms. 2011.

Russell, J. T. Imaging calcium signals in vivo: a powerful tool in physiology and pharmacology. *British journal of pharmacology*, 163(8):1605–1625, 2011.

Saad, F. A., Cusumano-Towner, M. F., Schaechtle, U., Rinard, M. C., and Mansinghka, V. K. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–32, 2019.

Saarinen, A., Linne, M.-L., and Yli-Harja, O. Modeling single neuron behavior using stochastic differential equations. *Neurocomputing*, 69(10):1091–1096, 2006. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2005.12.052. Computational Neuroscience: Trends in Research 2006.

Saarinen, A., Linne, M.-L., and Yli-Harja, O. Stochastic differential equation model for cerebellar granule cell excitability. *PLOS Computational Biology*, 4(2):1–11, 02 2008. doi: 10.1371/journal.pcbi.1000004.

Samaniego, F. J. *A comparison of the Bayesian and frequentist approaches to estimation*. Springer Science & Business Media, 2010.

Sambridge, M. A Parallel Tempering algorithm for probabilistic sampling and multimodal optimization. *Geophysical Journal International*, 196(1):357–374, 10 2013. ISSN 0956-540X. doi: 10.1093/gji/ggt342.

Sands, B., Burnaevskiy, N., Yun, S. R., Crane, M. M., Kaeberlein, M., and Mendenhall, A. A toolkit for dna assembly, genome engineering and multicolor imaging for *C. elegans*. *Translational Medicine of Aging*, 2:1–10, 2018. ISSN 2468-5011. doi: https://doi.org/10.1016/j.tma.2018.01.001.

Sarma, G. P., Lee, C. W., Portegys, T., Ghayoomie, V., Jacobs, T., Alicea, B., Cantarelli, M., Currie, M., Gerkin, R. C., Gingell, S., et al. Openworm: overview and recent advances in integrative

biological simulation of *Caenorhabditis elegans*. *Philosophical Transactions of the Royal Society B*, 373(1758):20170382, 2018.

Schaechtle, U., Saad, F., Radul, A., and Mansinghka, V. Time series structure discovery via probabilistic program synthesis, 2017.

Schafer, W. R. Mechanosensory molecules and circuits in *C. elegans*. *Pflügers Archiv-European Journal of Physiology*, 467(1):39–48, 2015.

Schafer Laboratory. Worm tracker 2.0, 2020.

Schalek, R., Kasthuri, N., Hayworth, K., Berger, D., Tapia, J., Morgan, J., Turaga, S., Fagerholm, E., Seung, H. S., and Lichtman, J. Development of high-throughput, high-resolution 3d reconstruction of large-volume biological tissue using automated tape collection ultramicrotomy and scanning electron microscope. *Microscopy and Microanalysis*, 17:966–967, July 2011. ISSN 1435-8115. doi: 10.1017/S1431927611005708.

Schmidhuber, J. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Scholz, M., Linder, A. N., Randi, F., Sharma, A. K., Yu, X., Shaevitz, J. W., and Leifer, A. M. Predicting natural behavior from whole-brain neural dynamics. *bioRxiv*, pp. 445643, 2018.

Schuster, I., Strathmann, H., Paige, B., and Sejdinovic, D. Kernel sequential Monte Carlo. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 390–409. Springer, 2017.

Seung, H. S. Neuroscience: towards functional connectomics. *Nature*, 471(7337):170–172, March 2011. ISSN 0028-0836. doi: 10.1038/471170a.

Seung, H. S. *Connectome: How the Brain's Wiring Makes Us who We are*. A Mariner Book. Houghton Mifflin Harcourt, 2012. ISBN 9780547508184.

Sheikholeslami, R., Razavi, S., and Haghnegahdar, A. What do we do with model simulation crashes? recommendations for global sensitivity analysis of earth and environmental systems models. *Geoscientific Model Development Discussions*, 2019:1–32, 2019. doi: 10.5194/gmd-2019-17.

Sheng, M., Sabatini, B., and Sudhof, T. Synapses and Alzheimer's disease. *Cold Spring Harb Perspect Biol*, 4(5), May 2012.

Shepherd, G. M. *Neurobiology*. Oxford University Press, 1988.

Shepherd, G. M. *Foundations of the neuron doctrine*. Oxford University Press, 2015.

Shindou, T., Ochi-Shindou, M., Murayama, T., Saita, E.-i., Momohara, Y., Wickens, J. R., and Maruyama, I. N. Active propagation of dendritic electrical signals in *C. elegans*. *Scientific reports*, 9(1):1–12, 2019.

Simpson, D. Phrenology and the neurosciences: contributions of fj gall and jg spurzheim. *ANZ journal of surgery*, 75(6):475–482, 2005.

Sköld, M. and Roberts, G. O. Density estimation for the Metropolis–Hastings algorithm. *Scandinavian journal of statistics*, 30(4):699–718, 2003.

Sofroniew, N. J., Flickinger, D., King, J., and Svoboda, K. A large field of view two-photon mesoscope with subcellular resolution for in vivo imaging. *Elife*, 5:e14472, 2016.

Song, S., Miller, K. D., and Abbott, L. F. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926, 2000.

Speiser, A., Yan, J., Archer, E. W., Buesing, L., Turaga, S. C., and Macke, J. H. Fast amortized inference of neural activity from calcium imaging data with variational autoencoders. In *Advances in Neural Information Processing Systems*, pp. 4024–4034, 2017.

Spira, M. E. and Hai, A. Multi-electrode array technologies for neuroscience and cardiology. *Nature nanotechnology*, 8(2):83, 2013.

Staines, J. and Barber, D. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

Stephan, C., Cammann, H., Semjonow, A., Diamandis, E. P., Wymenga, L. F., Lein, M., Sinha, P., Loening, S. A., and Jung, K. Multicenter evaluation of an artificial neural network to increase the prostate cancer detection rate and reduce unnecessary biopsies. *Clinical Chemistry*, 48(8):

1279–1287, 2002.

Sterken, M. G., Snoek, L. B., Kammenga, J. E., and Andersen, E. C. The laboratory domestication of *Caenorhabditis elegans*. *Trends in Genetics*, 31(5):224–231, 2015.

Stiefel, K. M. and Brooks, D. S. Why is there no successful whole brain simulation (yet)? *Biological Theory*, March 2019. ISSN 1555-5550. doi: 10.1007/s13752-019-00319-5.

Stiernagle, T. Maintenance of *C. elegans*. 1999.

Stosiek, C., Garaschuk, O., Holthoff, K., and Konnerth, A. In vivo two-photon calcium imaging of neuronal networks. *Proceedings of the National Academy of Sciences*, 100(12):7319–7324, 2003.

Stringer, C. and Pachitariu, M. Computational processing of neural recordings from calcium imaging data. *Current opinion in neurobiology*, 55:22–31, 2019.

Suárez, J., García, S., and Herrera, F. A tutorial on distance metric learning: Mathematical foundations, algorithms and software. *CoRR*, abs/1812.05944, 2018.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. 2018.

Sutton, R. *Reinforcement Learning*. The Springer International Series in Engineering and Computer Science. Springer US, 1992. ISBN 9780792392347.

Swendsen, R. H. and Wang, J.-S. Replica Monte Carlo simulation of spin-glasses. *Physical review letters*, 57(21):2607, 1986.

Swierczek, N. A., Giles, A. C., Rankin, C. H., and Kerr, R. A. High-throughput behavioral analysis in *C. elegans*. *Nature methods*, 8(7):592, 2011.

Syed, S., Bouchard-Côté, A., Deligiannidis, G., and Doucet, A. Non-reversible parallel tempering: a scalable highly parallel mcmc scheme, 2020.

Szigeti, B., Gleeson, P., Vella, M., Khayrulin, S., Palyanov, A., Hokanson, J., Currie, M., Cantarelli, M., Idili, G., and Larson, S. OpenWorm: an open-science approach to modeling *Caenorhabditis elegans*. *Frontiers in Computational Neuroscience*, 8(November):1–7, 2014. ISSN 1662-5188. doi: 10.3389/fncom.2014.00137.

Talbott, W. Bayesian epistemology. 2001.

Tao, T. *An introduction to measure theory*, volume 126. 2011.

Tarantola, A. Popper, bayes and the inverse problem. *Nature physics*, 2(8):492–494, 2006.

Tee, J. and Taylor, D. P. Is information in the brain represented in continuous or discrete form? *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 6(3):199–209, 2020.

Terada, S.-I., Kobayashi, K., Ohkura, M., Nakai, J., and Matsuzaki, M. Super-wide-field two-photon imaging with a micro-optical device moving in post-objective space. *Nature communications*, 9(1): 1–14, 2018.

Thomas, J. H. Genetic analysis of defecation in *Caenorhabditis elegans*. *Genetics*, 124(4):855–872, 1990.

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, 128(1-2):99–141, 2001.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Tolpin, D., van de Meent, J.-W., Yang, H., and Wood, F. Design and implementation of probabilistic programming language anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*, pp. 6. ACM, 2016.

Udrescu, S.-M. and Tegmark, M. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *arXiv preprint arXiv:2006.10782*, 2020.

van de Meent, J.-W., Paige, B., and Wood, F. Tempering by subsampling. *arXiv preprint arXiv:1401.7145*, 2014.

van de Meent, J.-W., Paige, B., Yang, H., and Wood, F. An introduction to probabilistic programming, 2018.

Van Merriënboer, B., Breuleux, O., Bergeron, A., and Lamblin, P. Automatic differentiation in ml: Where we are and where we should be going. In *Advances in neural information processing systems*, pp. 8757–8767, 2018.

Varshney, L. R., Chen, B. L., Paniagua, E., Hall, D. H., and Chklovskii, D. B. Structural properties of the *Caenorhabditis elegans* neuronal network. *PLOS Computational Biology*, 7(2):1–21, 02 2011. doi: 10.1371/journal.pcbi.1001066.

Vidal-Gadea, A., Topper, S., Young, L., Crisp, A., Kressin, L., Elbel, E., Maples, T., Brauner, M., Erbguth, K., Axelrod, A., et al. *Caenorhabditis elegans* selects distinct crawling and swimming gaits via dopamine and serotonin. *Proceedings of the National Academy of Sciences*, 108(42): 17504–17509, 2011.

Vita-More, N. and Barranco, D. Persistence of long-term memory in vitrified and revived *Caenorhabditis elegans*. *Rejuvenation research*, 18(5):458–463, 2015.

Vogelstein, J. T., Watson, B. O., Packer, A. M., Yuste, R., Jedynak, B., and Paninski, L. Spike inference from calcium imaging using sequential Monte Carlo methods. *Biophysical journal*, 97(2): 636–655, 2009.

Vogelstein, J. T., Packer, A. M., Machado, T. A., Sippy, T., Babadi, B., Yuste, R., and Paninski, L. Fast nonnegative deconvolution for spike train inference from population calcium imaging. *Journal of neurophysiology*, 104(6):3691–3704, 2010.

Wagenmakers, E.-J., Lee, M., Lodewyckx, T., and Iverson, G. J. *Bayesian Versus Frequentist Inference*, pp. 181–207. Springer New York, New York, NY, 2008. ISBN 978-0-387-09612-4. doi: 10.1007/978-0-387-09612-4_9.

Wang, L., Zhang, Y., and Feng, J. On the Euclidean distance of images. *IEEE transactions on pattern analysis and machine intelligence*, 27(8):1334–1339, 2005.

Warrington, A. and Dhir, N. Generalising cost-optimal particle filtering. *ICRA 2018 Workshop on Informative Path Planning and Adaptive Sampling, arXiv:1805.00890*, 2018.

Warrington, A. and Wood, F. Updating the VESICLE-CNN synapse detector. *NeurIPS 2017 workshop: BigNeuro 2017: Analyzing brain data from nano to macroscale, arXiv preprint arXiv:1710.11397*, 2017.

Warrington, A., Spencer, A., and Wood, F. The virtual patch clamp: Imputing c. elegans membrane potentials from calcium imaging. *arXiv preprint arXiv:1907.11075*, 2019. Also appeared in abstract form at *COSYNE 2021*.

Warrington, A., Lavington, J. W., Scibior, A., Schmidt, M., and Wood, F. Robust asymmetric learning in POMDPs. *arXiv preprint arXiv:2012.15566*, 2020a.

Warrington, A., Naderiparizi, S., and Wood, F. Coping with simulators that don't always return. In Chiappa, S. and Calandra, R. (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 1748–1758, Online, 26–28 Aug 2020b. PMLR.

Wasserman, L. et al. Bayesian model selection and model averaging. *Journal of mathematical psychology*, 44(1):92–107, 2000.

Wen, Q., Po, M. D., Hulme, E., Chen, S., Liu, X., Kwok, S. W., Gershow, M., Leifer, A. M., Butler, V., Fang-Yen, C., et al. Proprioceptive coupling within motor neurons drives *C. elegans* forward locomotion. *Neuron*, 76(4):750–761, 2012.

White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. The structure of the nervous system of the nematode *Caenorhabditis elegans*. *Philos. Trans. R. Soc. Lond., B, Biol. Sci.*, 314(1165):1–340, November 1986.

Whiteley, N., Lee, A., et al. Twisted particle filters. *The Annals of Statistics*, 42(1):115–141, 2014.

Wicks, S. R. and Rankin, C. H. Integration of mechanosensory stimuli in caenorhabditis elegans.

Journal of Neuroscience, 15(3):2434–2444, 1995.

Wicks, S. R., Roehrig, C. J., and Rankin, C. H. A dynamic network simulation of the nematode tap withdrawal circuit: predictions concerning synaptic function using behavioral criteria. *Journal of Neuroscience*, 16(12):4017–4031, 1996.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wittenburg, N. and Baumeister, R. Thermal avoidance in *Caenorhabditis elegans*: An approach to the study of nociception. *Proceedings of the National Academy of Sciences*, 96(18):10477–10482, 1999. ISSN 0027-8424. doi: 10.1073/pnas.96.18.10477.

Wood, F., Meent, J. W., and Mansinghka, V. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pp. 1024–1032. PMLR, 2014.

Wood, F., Warrington, A., Naderiparizi, S., Weilbach, C., Masrani, V., Harvey, W., Scibior, A., Beronov, B., and Nasseri, A. Planning as inference in epidemiological models. *arXiv preprint arXiv:2003.13221*, 2020.

WormAtlas. http://www.wormatlas.org. Accessed: 2018-05-15.

Xie, J., Xu, L., and Chen, E. Image denoising and inpainting with deep neural networks. *Advances in neural information processing systems*, 25:341–349, 2012.

Xie, Z., Clary, P., Dao, J., Morais, P., Hurst, J., and Panne, M. Learning locomotion skills for Cassie: Iterative design and sim-to-real. In *Conference on Robot Learning*, pp. 317–329, 2020.

Xu, C. S., Hayworth, K. J., Lu, Z., Grob, P., Hassan, A. M., Garcia-Cerdan, J. G., Niyogi, K. K., Nogales, E., Weinberg, R. J., and Hess, H. F. Enhanced FIB-SEM systems for large-volume 3D imaging. *Elife*, 6:e25916, 2017.

Yan, G., Vértes, P. E., Towlson, E. K., Chew, Y. L., Walker, D. S., Schafer, W. R., and Barabási, A.-L. Network control principles predict neuron function in the *Caenorhabditis elegans* connectome. *Nature*, 550(7677):519, 2017.

Yasuda, R., Nimchinsky, E. A., Scheuss, V., Pologruto, T. A., Oertner, T. G., Sabatini, B. L., and Svoboda, K. Imaging calcium concentration dynamics in small neuronal compartments. *Sci. STKE*, 2004(219):pl5–pl5, 2004.

Yemini, E., Jucikas, T., Grundy, L., Brown, A., and Schafer, W. A database of *C. elegans* behavioral phenotypes. *Nature Methods*, 10(9):877–879, 2013. doi: 10.1038/nmeth.2560.A.

Yemini, E., Lin, A., Nejatbakhsh, A., Varol, E., Sun, R., Mena, G. E., Samuel, A. D., Paninski, L., Venkatachalam, V., and Hobert, O. Neuropal: A multicolor atlas for whole-brain neuronal identification in c. elegans. *Cell*, 184(1):272–288, 2021.

Yoshida, E., Terada, S.-I., Tanaka, Y. H., Kobayashi, K., Ohkura, M., Nakai, J., and Matsuzaki, M. In vivo wide-field calcium imaging of mouse thalamocortical synapses with an 8K ultra-high-definition camera. *Scientific reports*, 8(1):1–15, 2018.

Yoshimizu, T., Shidara, H., Ashida, K., Hotta, K., and Oka, K. Effect of interactions among individuals on the chemotaxis behaviours of *Caenorhabditis elegans*. *Journal of Experimental Biology*, 221 (11):jeb182790, 2018.

Zarowny, L., Aggarwal, A., Rutten, V., Kolb, I., Patel, R., Huang, H.-Y., Chang, Y.-F., Phan, T., Kanyo, R., Ahrens, M., Allison, W. T., Podgorski, K., and Campbell, R. E. A bright and high-performance genetically encoded Ca2+ indicator based on mNeonGreen fluorescent protein. *bioRxiv*, 2020. doi: 10.1101/2020.01.16.909291.

Zhao, Y., Inayat, S., Dikin, D., Singer, J., Ruoff, R., and Troy, J. B. Patch clamp technique: review of the current state of the art and potential contributions from nanoengineering. *Proceedings of the Institution of Mechanical Engineers, Part N: Journal of Nanoengineering and Nanosystems*, 222(1): 1–11, 2008.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer*

vision, pp. 2223–2232, 2017.

Zucker, R. S. Calcium-and activity-dependent synaptic plasticity. *Current opinion in neurobiology*, 9 (3):305–313, 1999.