OPTIMISATION OF OPTICAL NEUROMORPHIC COMPUTING SYSTEMS

Oliver D. Neill

Submitted in fulfilment of the requirements for the
Degree of *Doctor of Philosophy*

School of Physics and Astronomy
College of Science and Engineering
University of Glasgow

Abstract

As the exponential scaling of computing systems predicted by Moore's law begins to slow, in part due to reaching fundamental physical limitations in the continued miniaturisation of transistors, the development of novel unconventional computing technologies with improved scaling laws is becoming increasingly important. Unconventional computing systems use substrates other than silicon transistors to encode and process information. As an umbrella term, it encompasses fields such as analogue, physical and neuromorphic computing, which focus on computation through continuous variables, complex physical processes, and artificial neurons respectively.

While there exist a wide range of physical processes and dynamics which one could imagine exploiting to process information, the key challenge is in designing scalable systems which can be easily programmed to solve specific tasks. The success of existing computing technologies relies on the strong predictions which can be made about their behaviour. When using alternative physical processes for computing, one must work against noise, instabilities, and unmeasurable internal dynamics, which can limit the determinism and predictability of a system.

In the following work, we present a series of results on optimising physical computing systems with these factors in mind, where we focus primarily on optical substrates due to their natural potential for energy efficiency, speed and parallelism. We approach this in two ways, first considering the design of high-level system architectures suitable for computing, and secondly considering ways of programming and optimising these systems to solve specific tasks. In the case of the former, we introduce a new physical computing architecture which uses quantum resources to improve scaling over an equivalent classical counterpart, and which is realisable with currently available technologies. In the latter, we apply meta-learning and reinforcement learning techniques to develop new optimisation strategies for training physical neural networks *in situ* in a scalable way. Throughout, we analyse the criteria necessary for efficiency and scalability, while also considering ways in which we can ensure the resulting systems are accessible and sustainable.

Beyond these examples, we discuss the broader motivations and requirements for the practical adoption of unconventional and neuromorphic computing systems, and the potential impact they could have on the scaling of future computing technologies, including the efforts towards realising artificial general intelligence and artificial consciousness.

Contents

List of Tables

List of Figures

Acknowledgements

Firstly, thanks go to my supervisor Daniele Faccio, for taking on a new student when least expected, enabling me to work on the projects I found most interesting, and providing guidance and belief in both me and the work. I also give many thanks to those others who have provided mentorship throughout this PhD: to Alex Turpin, who I initially started out with on this PhD journey; and especially to Giulia Marcucci, who provided insights, guidance, and much needed humour throughout the latter stages of this PhD, and without whom this thesis would look very different.

Thanks go to everyone in the Extreme Light group, past and present, for being such a welcoming group of people. The range of experience, skills, and diversity of relentlessly interesting work undertaken in this group has been a constant source of inspiration, and has undoubtedly made me a better scientist.

This thesis represents a tiny fraction of the work done and experiences gained over the last four years. Thanks to those at the UGRacing autonomous driving team, in particular Jim and Joe, who provided many welcome distractions throughout the PhD, and taught me to be a better software engineer. Thanks also to all those on the ZEUS 'Fly Your Thesis' team, for giving me the opportunity to experience microgravity, despite my day-to-day research being far removed from satellite attitude control.

Starting a PhD during a pandemic was far from optimal, and I am grateful to all who have helped me throughout, and made these years as enjoyable as they have been. Cameron Houston, for everything over the many years. Valentin Kapitany, sir, for your persistent optimism*. Philip Binner and Khaled Kassem, for the companionship and encouragement throughout the writing process. Sam Nerenberg, for advice on both science and life, for bringing me into your work and being a great collaborator, and for the plethora of jokes of dubious quality.

Beth, thanks for getting me back out on snow in the mountains. Sorcha—you have made this PhD infinitely easier and more enjoyable. I am grateful for your support, your patience, and your love. And finally, my parents, who have always supported me in all my endeavours, and known when to balance that support with a pragmatic kick to get things done. You fostered my curiosity and love of science from the very beginning, and I hope that the incessant questions and discussions have paid off. I won't be doing another PhD, I promise.

Declaration of originality

Statement

I hereby declare that this thesis is my own original work, except where explicit reference is made to the work of others, and has not been presented in any previous application for a degree at this or any other institution. The material in this thesis is either my own work with or without collaborators, or my own survey of relevant prior work given as context.

Contributions

Chapters 1 and 2 contain introductory material and overviews of the techniques used throughout this thesis, along with a review of the literature, and as such entirely comprise my own work.

The work in Chapter 3 is partially detailed in the paper *Photon Number-Resolving Quantum Reservoir Computing* [Ner+24]. Sam Nerenberg conceived the project, performed initial polarised network simulations, introduced the hybrid input states, and performed the spectral entropy analysis. I developed the Fourier analysis of the network (subsequently matched with prior analyses published in [GLA22; SSM21; VT20]), developed the encoding and decoding schemes, and implemented the end-to-end simulation of the system. In addition to that covered in the paper, I also include here analysis of system performance with respect to procedurally generated fitting tasks and different sampling distributions for the random reservoir.

Chapter 4 develops custom optimisers for training physical systems through the use of reinforcement learning. This project was my original idea, and I implemented and analysed the various systems detailed in this chapter, aspects of which were published in *Optical neural networks trained in situ with reinforcement learning* [NF24]. The work extending and analysing the reinforcement learnt optimiser for deep physical neural networks is in preparation.

In all works, Daniele Faccio contributed through discussions as my supervisor. Giulia Marcucci provided insights and contributed to discussions throughout the latter stages of Chapters 3 and 4.

All figures and tables in this thesis were created by myself unless stated otherwise.

Publications

* indicates equal contributions

- S. Nerenberg*, **O. D. Neill***, G. Marcucci, and D. Faccio, *Photon Number-Resolving Quantum Reservoir Computing*, 2024, DOI: 10.48550/arxiv.2402.06339, arXiv: 2402.06339 [quant-ph].
- **O. D. Neill** and D. Faccio, "Optical neural networks trained in situ with reinforcement learning", in: *Machine Learning in Photonics*, ed. by F. Ferranti, M. K. Hedayati, and A. Fratalocchi, vol. 13017, International Society for Optics and Photonics, SPIE, 2024, p. 130170L, DOI: 10.1117/12.3021870.

Chapter 1

Introduction

There is a scalability problem at the heart of modern computing. The demand for fast, parallel hardware, in part driven by recent successes in the fields of machine learning (ML) and artificial intelligence (AI), has never been higher, while the ability to meet this demand is beginning to slow as we reach fundamental limits in the scaling of silicon computing technologies [Moo06]. Combined with the pressing concerns over the sustainability and energy efficiency of these systems, it becomes clear that in order to continue to scale our computational capacity, a hardware revolution is required [Mar+20b]. Today, there is significant interest and investment being directed towards developing new energy-efficient hardware which can scale with these modern, data-intense computing modalities [23; Wri+23].

While some continue to develop new architectures which retain silicon transistors as the primary mechanism of operation, there is a growing focus on unconventional computing approaches, which deviate from conventional digital electronics for processing information [Ada18]. These systems may sacrifice some of the flexibility and control of conventional computers for properties such as speed, parallelism and energy efficiency [McM23].

Two related forms of unconventional computing which have shown promise are physical and neuromorphic computing. Physical computing refers to the use of physical processes with interesting mathematical properties, which can be adapted to perform computation. A familiar example is the quantum computer, which was originally proposed by Feynman in 1982 as a way to simulate quantum systems directly using quantum processes. However, there are a wide range of classical platforms which can be used for information processing, including systems as conceptually simple as a bucket of water [FS03; MP23].

Neuromorphic computing and neuromorphic systems (NMSs) on the other hand take inspiration from the human brain. This may include mechanisms for learning, information transport, processing and storage, improved energy efficiency and computational density, but in a surrogate physical system which is more interpretable and manufacturable, and ideally which can be integrated with existing computing technologies.

While it may seem natural that mimicking properties of the brain, which is the most efficient computer we have access to, would be a good route to improved computing, the field is still comparatively undeveloped. Currently, proposed NMSs have yet to see significant real world impact—most are only practical at small scales, solving niche or trivial tasks, and are very much still an early research field, rather than a competitive computing paradigm [20].

To move beyond this, the first challenge is to design NMSs which can be scaled to sizes where they can solve useful tasks, beyond toy problems. This has to come from both the design of the system and the method used to program it to solve a task. It is important to recognise that the programming strategy is a vital part of the overall design process.

The second challenge is to make this competitive with silicon computing, in at least some area. The cost or efficiency of a particular system can be measured by a range of metrics depending on application, of which a few include financial cost, energy efficiency, speed, reliability, and utility. For a system to be scalable, it must perform well across all of these criteria, in particular utility, determined by the classes and complexity of the problems which can be solved by it.

In practice, some metrics will be more important than others, and hence there must be trade-offs. In this work, we will focus on optimising primarily for scalability and accessibility. This will lead us to consider new approaches to designing both the architectures, and the methods of programming NMSs. We will make heavy use of ideas from machine learning (ML), which is founded in mathematical optimisation and statistics, and has proven highly successful in recent years at solving large scale, complex tasks. We 'optimise' in two senses—firstly, we consider how to make good decisions when designing the architecture of a NMS and secondly, develop new methods for the literal mathematical optimisation (or programming) of these systems.

The first case considers the design of a computing architecture, where we define the fixed features of the system. These typically cannot be optimised mathematically, as they exist in too large or difficult a search space. Instead, we have to rely on intelligent design, where the decisions made determine the class of tasks that can be solved, the time complexity on those tasks, and the overall energy efficiency of the system. This is analogous to designing the circuitry of a central processing unit (CPU), or the connectivity of an artificial neural network (ANN) in machine learning.

Programming concerns tuning the free parameters provided by the architecture, and can take several forms depending on the type of computing system. The most familiar would be conventional programming, where we choose a sequence of instructions from some set of well-defined primitives, which control the flow of information through the system. We do this by constructing logical algorithms and translating these into a minimal set of universal operations, and while most of the world's software is written in

high-level languages which abstract away the underlying hardware, the result is always the same—a set of instructions which are executed sequentially by the hardware. In machine learning, we can view the architectures as parameterised mathematical models, and the programming as the mathematical optimisation over the free parameters. We will be loose from here on out in our use of programming (in the mathematical sense), programming (in the modern computing sense), training (in the machine learning sense), and optimisation (our blanket term). In all cases, we seek to choose a set of parameters from some available continuous or discrete set (defined by our computing system) which specialise our computing system to solving a task we care about. This is only possible if the task exists within the class of tasks solvable by the system.

In conventional computing, the separation of architecture from programming (i.e. hardware and software) is a deliberate design decision, facilitated by many layers of abstractions. These abstractions allow for end users to focus on the task at hand, and for hardware and software to evolve largely independently, however the overhead induced by these layers of abstraction can be a bottleneck for performance.

In unconventional computing, and especially neuromorphics, the intention is to reduce or remove these layers of abstraction, and to have the hardware and software more closely integrated. This is motivated by the potential for improved performance, energy efficiency, and scalability, however it introduces new challenges in terms of programming and optimisation.

Improvements in these two areas are both key for practical use and accessibility of NMSs, and developing these require very different approaches.

Let's examine these areas in a little more detail, to better understand the considerations which go into selecting platforms, and designing architectures and programming methods for unconventional computing systems.

Architecture. How do we design NMSs which are good at solving some class of task?

We need to design and build systems with interesting dynamics. While the component parts don't necessarily need to be complex, they should be rich enough that complexity can emerge at scale. Silicon transistors in digital systems are, to first approximation, extremely simple devices, however when combined intelligently, they can perform complex computations—indeed generating a universal set of operations for a Turing complete computer.

If universality isn't required or possible in a particular dynamical system, then we need to consider the maximal class of tasks which is solvable by the system, and use this to

evaluate whether it is suitable for a particular application. Typically, it may not be possible to exactly define this, but we can use our understanding of the dynamics to make reasonable assumptions about the types of tasks which a particular system might be applicable to.

Overall, we tend to look for systems which have rich, complex dynamics, such that matching the system to the task becomes less important, and so that development of a particular architecture leads to a wide range of potential applications.

For example, in neuromorphic computing, excitable dynamical systems may be used, as this matches how information is encoded and communicated in the brain, through the use of action potentials, or spikes, which propagate between neurons. Alternatively, quantum systems provide access to entirely new dynamics, which enable new classes of algorithm which give improved performance on certain classes of task. This is made possible by features such as interference, superposition, and large, potentially infinite dimensional state spaces.

The common feature is that we need to design systems which are complex enough that they can perform the task we care about, but simple enough that we can understand and control them. Scalability comes from the connectivity present within a system, including how easily we can add new computational units, energy consumption during operation, and how universal the operations which can be performed are.

Platform. Which sorts of substrates are amenable to performant neuromorphic computing?

A similar question to that of architectures, choosing a substrate selects the types of physical processes available for performing computation. There are many possible options here, with some physical systems being more amenable to computation than others. Electronics, for instance, is well understood, and has been developed over the last century to be extremely reliable and efficient at performing computation. New electronics-based technologies such as memristors and spiking neuromorphic chips are promising, however they tend to face many of the same scaling limitations as silicon computing [Wan+18; Dav+18].

Optical systems are another attractive option. Light has many degrees of freedom in which to encode information, and its non-interacting nature is suited to parallelism through multiplexing and 3D spatial scaling. Speed is another natural benefit, with optical processes and bandwidths orders of magnitude faster than current digital switching transistors. The technology is mature, with much developed by the telecommunications industry, as optics is currently the standard for long distance high-speed data transfer. This means that there is already considerable expertise and technology for interoperating digital electronics and optical systems for information transfer. If we are already

comfortable transmitting data in optical form, then it is natural to consider processing this information in transit, in the optical domain. Efficiency and energy consumption are also key benefits of optical systems, with the potential for very low energy operation, and the ability to operate at room temperature without the need for cryogenic cooling.

Optics is also interesting as it is a strong candidate platform for future quantum computing, with much investment and interest in this area [Mad+22; Mar+24]. Quantum computing may be built on different foundations to neuromorphic computing, but the two share many of the same challenges, and optical systems are excellent candidates for bridging this gap and sharing development resources.

Programming. How do we optimise this system for a specific task within its domain?

In this thesis, we largely focus on the data-driven, machine learning approach to programming, where we seek to do a numeric optimisation over a set of free parameters to solve our task. This relaxes the requirement to model and make mappings from conventional algorithms to unconventional hardware, at the expense of curating large datasets and potentially a reduction in interpretability.

This notion of moving away from deterministic, hand-crafted algorithms built from a set of predetermined simple instructions, towards a more data-driven, optimisation based programming, was coined 'Software 2.0' in 2017 by Karpathy [Kar21]. In this article, he states 'Gradient descent can write code better than you'. At first this may seem controversial—even today, the wide majority of software in the world was written by humans, not learnt by machines. However, the point becomes clearer when we consider the boundary of the class of tasks which are realistically solvable by humans versus that of learnt programs. Until recently, generating realistic looking fake human faces, automatically translating between languages, or generating diagnoses from medical images were all tasks beyond the capabilities of computers. It turned out that wasn't the case: they were just beyond the capabilities of *human programmers*. For most of the history of digital computers, we have been limited to solving tasks which can be solved with tractable algorithms, devisable and implementable by humans through conventional programming. With the developments in ML, we can now access a wider scope of tasks with greater complexity, illustrated in Figure 1.1, where the scale of the problem is beyond human capability to solve directly, at least without significant effort.

While the use of 'complexity' in the original article is left vague, it should not be confused with a program's time or memory complexity. Rather, it should be interpreted more akin to the entropy of the program's source code. A measure such as Kolmogorov (algorithmic) complexity could be employed to concretely compare different programs' complexity, with simpler programs more likely to be devisable by human programmers. For instance, a simple program which uses a for-loop to print the first one million integers may perform more operations at runtime than a simple two-layer dense neural

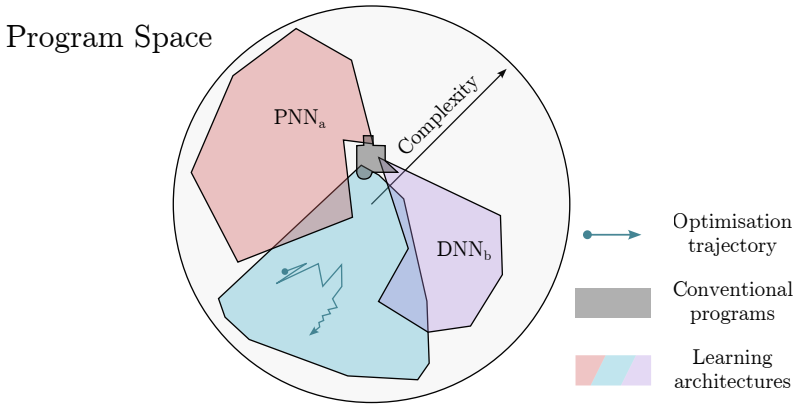


Figure 1.1: **Software 2.0.** The space of possible programs. The space explored by all current algorithms and software is schematically illustrated in grey (note, conventional computers remain Turing complete: this diagram illustrates programs which have been written by humans, not the entire set of possible programs implementable with conventional programming). The spaces of programs realisable for a set of specific learning architectures (i.e. a particular physical neural network (PNN), a particular deep neural network (DNN)) are illustrated in colour. These may overlap with conventional programming and each other, and span a greater area and complexity range in the total space of possible programs, where complexity increases radially. For a given architecture's subspace, iterative optimisation on a task using that architecture generates trajectories within this space. Figure adapted from [Kar21].

network with 10×10 matrices which has been pretrained to solve some task. However, each value in those matrices must be individually specified and computed, meaning the program is not easily compressible. It would be difficult for a programmer to hand-pick these matrices to solve the same task. Under algorithmic complexity, the neural network would be considered more complex than the for-loop.

It is important to point out here that this is not a fundamental limit of the computational tools we have been using—we can rigorously prove through the Church-Turing thesis that any algorithm[1] can be implemented on modern computers [Chu36; Tur37]. Instead, it speaks to the limitations of human ability to effectively search the space of possible programs, at least with our current approaches of developing logically rigorous algorithms.

As a result, this Software 2.0 landscape is unfriendly and alien to humans, which introduces challenges. We sacrifice interpretability and often go against what our intuitions may tell us is the 'right' way to solve a problem, in favour of what performs well. With this comes fewer guarantees and reduced predictability, which may be

[1]We sweep some of the nuance under the rug here—what we mean when we say algorithm is one of the key points of Church-Turing, and requires us to define the concept of a 'calculable function'.

necessary in safety critical applications.

We can broadly divide approaches in physical computing according to this paradigm, where on one hand there are approaches which attempt to implement conventional mathematical operations and instructions in unconventional hardware, allowing a conventional algorithm and programming approach, while on the other hand there are approaches which attempt to use the physical system as a black box (albeit intelligently designed according to the requirements laid out in the earlier discussion on architectures), and train it to solve a task through data-driven methods and mathematical optimisation. In the latter case, we can see again that conventional programming, mathematical programming, mathematical optimisation, and training all take on the same meaning—discovering the parameters for a given architecture which best solve a task.

Beyond the motivation of being able to discover new types of algorithms, which humans may not be able to devise by hand, there is one other key reason for using mathematical optimisation to program physical computing systems. In many cases, traditional programming techniques are simply not feasible, due to incomplete knowledge of the internal dynamics and state of the physical system and an inability to directly map the mathematical operations of a particular algorithm to specific physical processes.

Treating a physical system as a physical neural network (PNN), where we expose some set of tunable parameters, and apply data-driven training methods, allows us to avoid having to encode particular task-specific algorithms or operations by hand. However, there remains one main problem when attempting to apply iterative optimisation schemes common to ML to PNNs—the lack of an analytic, differentiable model for the dependence of the measurable output on the controllable input. This prohibits the use of the gradient-based optimisation algorithms, which form the backbone of modern ML, for training the PNN parameters directly [Mom+24].

Currently, there are two main approaches to getting around this problem—training in simulation and only using the hardware for inference, or gradient-free optimisation techniques such as evolutionary strategies or finite-difference schemes. Simulation-based methods can be effective, allowing established gradient-based optimisation techniques to be used, however misalignment between simulation and physical system tends to limit final inference performance, especially as the system scales and simulations become prohibitively slow or infeasible without high-performance computing, even for fairly simple physical systems [Wri+22; Zho+21]. Gradient-free methods have the advantage that they can be used directly with the physical system, however have their own issues with scaling, with performance dropping dramatically as the dimensionality of the parameter space increases.

Beyond these two options, it is worth mentioning that there are ways we can restrict the types of systems we consider in order to make this programming easier, and one such

approach which has seen success in neuromorphic computing, and which will feature prominently in our work, is the reservoir computing (RC) paradigm. Made popular by its ability to use arbitrary complex physical systems to solve tasks while avoiding difficult training processes, RCs have been demonstrated on a range of complex tasks with good performance, are extremely easy to train, fairly scalable, and amenable to a wide range of physical and dynamical systems. They can also be relatively well understood through the lens of dynamical systems theory.

This thesis aims to serve two main purposes: firstly, to attempt to take a small chip out of the block in designing new approaches for optimising physical computing systems; and secondly, in doing so, to lower the barrier to entry for performing useful computation with physical systems. The greater the access that the scientific community has to implement and train these types of system, the faster the pace of development will be, and the greater the potential impact for improving the efficiency and capabilities of ML and AI systems will be.

We will first introduce some preliminaries, covering the state of physical computing, diving into neuromorphics and complex systems, then optimisation strategies, starting from state of the art in computer science and ML. We then cover technical details required from a physicist's point of view in developing new optimisation techniques, including current approaches to training physical computing systems.

The main contributions in this thesis are then broken down into two chapters. The first project considers whether we can efficiently scale the reservoir computing paradigm through the use of quantum resources. Our focus is on cost-efficient scaling of a simple quantum reservoir computer through the introduction of photon number resolved detection, practical quantum states, and random linear optical scattering. The modelling is deliberately general, allowing implementation in a wide range of optical systems. This is detailed in [Ner+24].

The second project detailed in the thesis, split into two sections, switches focus from architecture to programming, and introduces the idea of using reinforcement learning (RL) to generate custom application specific optimisation algorithms tailored to a particular physical neuromorphic system. The first half introduces the key RL concepts used, and the principles applied to train a simple diffractive optical layer to classify images. We discuss model-based vs model-free RL and how physical priors can be leveraged to improve training, comparing the performance of the new method with existing optimisation techniques. This is detailed in [NF24]. The second half of this chapter details the extension of the RL optimiser to deep neuromorphic systems, which enables training sequential chains of physical systems in a scalable way.

Finally, we analyse the potential future opportunities stemming from these works, placed within the context of the current interest in neuromorphic hardware and the recent advances in ML and AI.

Before we begin, some prerequisites and comments on notation.

We assume the reader has some familiarity with modern ML and deep learning methods, however we will introduce key concepts as and when needed. For further reference, we provide some key texts. [Goo+16] provides an excellent overview of the foundations of deep learning, however is out of date with respect to the most recent advances in the field. [Ant20] is more up to date, taking a hands-on, practical approach. [Sut20] remains the canonical reference on reinforcement learning, which plays a core role in Chapter 4.

On the topic of neuromorphic and physical computing, we review the history of the field in Section 2.1.2, however we also refer the reader to [Abr+24] and [McM23], which provide overviews of physical and optical computing, and [Bru19] which provides an introduction to the specific case of photonic reservoir computing. While the terminology used throughout should be clear, we provide an overview of the main terms relating to neuromorphic computing, and how we interpret them, in Appendix A.

We largely follow the mathematical notation in Goodfellow [Goo+16], and any other notation used is either explained in context or included in the Notation section. Additionally, each results chapter has a dedicated notation section at its end, summarising the main notation specific to that chapter. Similarly, for abbreviations see Abbreviations. Some notation may differ between chapters in order to remain consistent with the existing literature, and in these cases the intention should be clear from context.

Chapter 2

Background

2.1 A brief history of computing

To understand how the computing landscape was shaped into its current form, we need to consider the evolution and constraints present through the development of computing systems. While we may think of computing as being a relatively recent invention, humans have been designing systems to assist with computation and prediction throughout antiquity. The earliest examples tended to focus on predictions of the heavens, from astrolabes which were able to assist in tracking stars and planets, to the earliest known example of a true, analogue computer, the Antikythera mechanism, which was used to predict astronomical positions and eclipses.

In modern history, the development of computation arose from ideas in mathematics and logic, and it was Charles Babbage and Ada Lovelace who pioneered the fields of computing and programming in the early nineteenth century. Babbage's analytical engine was the first general purpose computer[1], and Lovelace's work on the engine's algorithms is considered the first computer program. The development of the Turing machine, and the work of Turing and Church on the theory of computation, laid the groundwork for modern computing, and the development of the first digital computers in the 1940s [Chu36; Tur37]. These early systems were unwieldy, slow and expensive, even despite the shift from mechanical systems to electrical systems using thermionic valves. The use of binary representations of information allowed improved robustness, avoiding some of the complications of analogue computing. These valve systems were the foundation of what we recognise as the modern computer, however it was the development of semiconductor transistors, and the subsequent development of integrated circuits through new lithography manufacturing techniques which truly allowed scalable computing to become a reality.

With the ability to manufacture and interconnect transistors, logic gates could be built, and from these, mathematical operations implemented. A second benefit of

[1] Despite the theory of general purpose computing and computability not being developed for a further hundred years.

the transistor was the ability to store information for long periods of time, where memory cells which could be efficiently written to and read from allowed the close integration of the separate compute and memory units. The organisation of these components in a modern computer is generally attributed to John von Neumann, who, among a plethora of contributions to physics, mathematics and foundational work in computer science, developed the ideas behind the von Neumann architecture of computing [Neu93]. This model comprises an arithmetic unit, which does computation, a control unit which sequentially executes instructions, memory which allows storing of data and the instructions themselves, and input/output devices which allow the computer to interact with the world. Even eighty years after the first description of this architecture, it remains by far the dominant model used in devices today.

For several decades, hardware development focussed on scaling—building faster and larger CPUs, faster and larger memory, better data storage and networking capacity, to allow for distributed computing and communication, and improving energy efficiency, fault tolerance, robustness and speed to allow for the scaling of these systems. As clock speeds started to plateau due to fundamental bandwidth limits in silicon transistors, parallelism became more important, with multicore CPUs being developed, and advanced strategies for speculative execution and application specific circuitry for tasks such as cryptography and floating point arithmetic being developed. As demand for multimedia applications increased, the development of even more application specific circuits for video processing became common, and the dedicated graphics processing unit (GPU) was born. GPUs were designed to perform a single instruction on multiple data in parallel. Restricting each processor to perform the same instruction at the same time made scaling the number of processors much easier than independent processors in conventional CPU, and with a different memory hierarchy and architecture, GPUs were able to efficiently process large arrays of data.

In software, the improvements have focussed on choosing good abstractions which allow for the development of complex algorithms, and improving the efficiency of these algorithms through intelligent design and rigorous logic.

The most recent paradigm shift in this field has been the progress in machine learning and AI, which has largely been facilitated by a shift to using hardware which is particularly efficient at parallel computing—the GPU. While these devices originated as a way of doing the large scale linear algebra and simple, parallel operations needed for graphics processing, this parallelism has turned out to be highly useful for the matrix operations which are at the heart of many machine learning algorithms.

In 2012, AlexNet broke the floodgates on modern ML, by demonstrating that not only could GPUs be used to train large neural network models capable of performing previously difficult image classification tasks, but that the training could be implemented

efficiently through a combination of parameterised convolutional operations (convolutional layers) and the backpropagation algorithm, which made use of the analytic form of the neural network to calculate numeric derivatives, giving a linear approximation to the system compatible with iterative, gradient-based optimisation [KSH12]. The boom in deep learning research, and subsequently AI, has been well documented since.

The resulting decade of advances has seen a noticeable shift in certain applications, from hand-designed algorithms, to data-driven algorithms which are optimised for specific tasks through the use of example data. Even here though, application specific models have been key, with architectures such as the convolutional neural network (CNN), the recurrent neural network (RNN), and the transformer all being designed to be efficient at particular tasks. These work through identifying symmetries of the task we are trying to solve, such as translation invariance in images which motivates CNNs, position invariance in sequences which gives rise to RNNs, or relationships between different segments of an input which are important for context that can be modelled well using transformers. These architectures have been particularly successful in image and speech recognition, generative modelling and natural language processing, with 'AI' now a part of many people's day-to-day lives. While there is still a large gap between these AI systems and human intelligence, the progress has been rapid, and the field is still growing at an exponential rate. Concerted efforts toward artificial general intelligence (AGI) are likely to succeed, even if perhaps over a longer term than the current excitement would suggest.

Today we find ourselves in this new landscape, where the balance between Karpathy's 'Software 1.0' and 'Software 2.0' is still being negotiated. One thing is abundantly clear however, and that is that AI development has already reached significant limits in terms of the hardware and the sustainability of current practices. The GPU industry is one of the largest and most lucrative in the world, and many manufacturers have begun developing more specialised hardware for AI, including neural processors integrated in consumer devices, tensor processing units (TPUs) for training large models, and more recently dedicated neuromorphic chips, such as Intel's Loihi [Dav+18], which deviate from the von Neumann architecture to better mimic aspects of the brain's structure and function.

These devices are dependent on a few large manufacturers, creating monopolies and ethical issues regarding access, and exploitation of resources, including the raw materials needed to build them—rare earth metals and natural resources which fuel conflict and environmental damage. They have typically short lifespans due to wear, and consumers' requirements to remain competitive as new devices are released. Recycling of devices is still not well established, and ultimately they suffer many of the same issues as

conventional silicon chips, resulting in huge data centres being established with power consumption requirements comparable to that of small nation states [Kaa+22].

These properties are all incompatible with the sustainability practices deemed necessary to avoid global environmental catastrophe, and encourage practices where the benefits of these new technologies are limited to those with the resources to invest in their development, and the costs are borne by those who benefit least and who are least well-placed to object.

These challenges will not be solved easily, and in many cases require policy changes and regulation, however, it is important to recognise that a) the genie is out of the bottle, and demand for these tools will not decrease, b) these tools and technologies have potential to introduce positive changes in the world, in otherwise intractable areas, and at scale, and c) there are key inefficiencies in the current approaches which are tractable and where progress can be made.

While we have skipped over many of the details and important contributions to the field, this history has aimed to highlight three continued trends throughout the development of computational systems.

First, the initial trend towards generality and universality, with analysis of the minimal requirements for computation, which once satisfied, subsequently led to the sacrificing of some generality in exchange for application-specific performance. Evidence includes the creation of dedicated circuits for floating point arithmetic and encryption, despite both operations being possible on the minimal Turing machine through software, and recently the trend towards general-purpose GPU (GPGPU) computing and TPUs for AI.

Second, the trend towards increasing complexity and efficiency, which come in discrete steps as breakthroughs are made either in algorithms or, more typically, in hardware. These include the advent of electrical computers, then transistors, then integrated circuits, and more recently, the development of parallel computing, including the use of GPUs for general purpose computing.

Finally, the increase in capability which comes with greater networking and connectivity. This applies at all scales, from improved connectivity and parallelism in individual CPUs, to within specific machines, to large-scale data-centres which allow for large scale distributed computing, and to the global network which allows for the sharing of information and computation across the world.

2.1.1 Optical computing

While the von Neumann architecture clearly won the hardware lottery, there remain a range of other computing paradigms which have been studied and are now relegated to the class of 'unconventional computing' approaches [Ada18]. Here we will give a brief overview of the evolution of optical computing in particular, given the dominant role it will play in the remainder of this thesis.

The Stanford multiplier is one of the earliest examples of an optical computing device, where cylindrical lenses are used to implement matrix-vector multiplication in diffractive optics [Goo96]. Throughout the latter half of the twentieth century, there were many analogue optical computing architectures and schemes proposed [Goo85; Psa+90]. Psaltis et al. demonstrated early implementations of optical neural networks, using an optical system to perform facial recognition in the 1980s and 90s[CP76; Far+85; LQP93]. A recent example showing large-scale matrix-matrix multiplication in optics demonstrates the scaling potential of diffractive optical neuromorphic systems, by exploiting both spatial and frequency degrees of freedom [Lat+24]

While there have been periods of stagnation in the field throughout its history, today there is significant renewed interest in novel optical computing, with commercial enterprises developing and selling access to new forms of optical accelerator [McM23]. One example is the use of a coherent random scattering process as a high-dimensional random projection, where information can be encoded in the spatial modes of a beam of light, processed by the scattering process, and read out via detection. This is equivalent to a high-dimensional random matrix multiplication, which is comparatively expensive to perform *in silico* [Cav+22].

This progress has largely been driven by two separate developments: the availability of low-cost electro-optic devices including light modulators, improved detectors, and integration possibilities through photonic circuits; and, partly with the development of new ML approaches, new algorithms and processes for programming optical devices to solve complex tasks. The latter point should not be undervalued—the ability to frame an optical computer as a neuromorphic computer, where we can use data-driven training schemes from ML to learn solutions to tasks, opens the door to a range of previously unfeasible optical architectures.

For wide scale adoption, commercialisation and integration are important considerations, and again this is an area where optical computing is promising. Photonic circuits, which can be manufactured using similar processes as conventional electronic circuits, are a potential way to integrate optical processors, mitigating many of the irregularities and misalignment issues in free space optics, while also providing excellent compatibility with existing digital electronic computers.

More generally, there have been many examples of application specific tasks being performed using optical systems, such as nonlinear optical waves, spiking neurons implemented in optics, and multimode programmable neural networks [CD10; Raf+20; MPC20; Mar+20b; BP21; Lat+24; Wan+24].

As a final point, optical systems are also interesting due to the crossover with current research trends in quantum computing. Photonics is a natural candidate for implementing qubits, and there are several demonstrations now of quantum machine learning architectures designed on optical platforms [Arr+21; Muj+21]. Implementing neuromorphic systems in optics allows us to make use of benefits of both fields, and this is an area we explore in more detail in Chapter 3.

2.1.2 Neuromorphic computing

Initial approaches to neuromorphic computing were energy-based, with Hopfield networks being one of the earliest examples. These networks, based on the Ising model from statistical physics and exhibiting similarities to spin glass systems [Tar15], represent a form of associative memory. In Hopfield networks, the system's energy is minimised to find a stable state corresponding to a pattern to be recalled. The Ising model is an energy function based on the configuration of coupled nodes, defined as

$$H(\sigma) = -\sum_{ij} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i, \tag{2.1}$$

where $\sigma_i \in \{-1, 1\}$ is the state vector, J_{ij} is the coupling matrix which defines interactions between nodes, and h_i represents an external field biasing the network. Since the system is discrete, there is a finite number of configurations, each associated with its own energy.

Hopfield proposed using such a system for pattern recognition. By optimising the couplings, the network evolves from an initial configuration encoding an input, under an update rule which minimises energy, until it converges to a stable state corresponding to the stored pattern [Hop82].

This concept was later extended into Boltzmann machines and their simpler variant, the restricted Boltzmann machine (RBM) [AHS85]. In these models, a subset of nodes designated as inputs is initialised, and the system evolves according to an update rule based on the Boltzmann probability distribution over energy states. Eventually, the system converges to thermal equilibrium. These networks are trained by adjusting the couplings to make predictions on input data, with the final converged state providing the output. Various implementations of these systems have been developed, including optical versions [Far+85]. Today, energy minimisation remains a central motif throughout neuromorphic computing, conventional machine learning, and theories of learning in biological systems.

In the decades since these initial architectures were devised, we have seen the emergence of multilayer perceptrons, neural networks, architectures such as convolutional neural networks, recurrent neural networks, and transformers. These models have been employed across a wide range of applications, including image recognition, speech recognition, natural language processing, and generative modelling.

In parallel, substantial progress has been made in the neuroscience community to understand biological learning mechanisms. The human brain, the most energy-efficient computing system known, consumes around 20 watts of power, yet performs tasks beyond the reach of today's supercomputers [Len03; SS14]. This is due to the brain being developed under an entirely different set of evolutionary constraints, which selected for energy efficiency and performance in selected tasks most correlated with survival and propagation, while features which we see as desirable in our computing systems, such as interpretability and scalability, provided no benefit and as such never featured.

The brain operates as an analogue, massively parallel, and distributed system in a low-precision, high-noise environment. It trades off efficiency in time for space, with relatively slow speeds but highly efficient utilisation of resources, where distinct areas in the brain perform multiple roles. It is capable of learning and adapting to new tasks and making good use of sparse data, and operates over a wide range of timescales and memory scales. While we can see the brain as an ideal in terms of some metrics (efficiency, adaptability, lifelong learning, multimodality), there are areas where current computing systems outperform, such as speed, precision, reproducibility, and scalability. There are many tasks which humans struggle to perform which are simple for computers, and vice versa. Neuromorphic computing, then, seeks to bridge the gap between these two computing regimes, aiming to build systems which combine the best attributes of both.

Our understanding of the internal dynamics of the brain are now at a point where we can identify specific brain regions responsible for various tasks and observe how these regions interact to perform complex functions. At a microscopic level, we can analyse individual neurons and their collective behaviour. This has led to the development of neuron models, including the original leaky integrate-and-fire model [Abb99], which, despite its simplicity, captures a range of biological neuron behaviours. Such models have in turn inspired the creation of continuous-time neuromorphic systems which replicate neuron behaviour, advancing novel computing architectures [Wu+22; Sub+24; Sta+24].

Today, the fields of neuroscience, computing, and engineering are converging through the development of neuromorphic computing systems, aiming to build brain-inspired hardware that efficiently handles tasks difficult for conventional computers. Beyond just improvements in computing, there is scope for lessons which are learnt in designing,

building, and training neuromorphic systems to inform our understanding of biological learning. Neuromorphic chips, like Intel's Loihi [Dav+18], enable programming of spiking neural networks in software, which can then be executed efficiently on hardware. Such systems are a strong first step to understanding and building efficient biologically inspired artificial learning systems.

In-Memory Computing. In-memory computing is another rapidly developing area that addresses the inefficiencies of the von Neumann architecture associated with the separation of compute and storage. This is important, as in current systems, data transfer can account for up to 40% of total energy consumption, making it a key target for optimisation [Mut+19]. By embedding memory and processing in the same physical location, in-memory computing reduces data movement and the associated energy costs. This is particularly useful for tasks such as matrix multiplication, where large datasets must be processed in parallel. However, challenges remain, particularly in efficiently interconnecting the distributed computing nodes, and such systems are typically programmed using conventional algorithms, which limits their capacity to learn or adapt to new tasks.

Mortal Computation. In a recent paper, Hinton proposed the notion of 'mortal computation' [Hin22], advocating for blurring the boundaries between hardware and software in order to reach the next level of performance. The motivation for this is that the most efficient computing systems, such as the brain, have no separation between hardware and software, or layers of abstraction. They are designed purely for performance. In the brain, 'hardware' and 'software' are inseparable, co-evolving over time in response to environmental stimuli, and one cannot in principle take an 'algorithm' implemented in one brain, execute it on another, and expect to be able to reproduce the same behaviours. By removing some of the layers of abstraction present in conventional computing, systems can achieve higher efficiency, although at the cost of generality and configurability. Even interpretability may be sacrificed, provided the necessary constraints and safety guarantees are met.

2.2 Optimisation and learning

2.2.1 Mathematical optimisation

We have already motivated viewing the programming of a variety of computing systems as an optimisation problem, and in this section we will give an overview of the types of optimisation methods we might want to consider. Optimisation is of course a wide field, and so we will limit ourselves to the most relevant methods for training machine learning models and neuromorphic systems.

Optimisation theory has a long and rich history, and as such, there are many classes of problems and methods of solving them. There are certain properties of a given problem

which turn out to be more or less useful for categorising them, whereby categorising and comparing with known solvable problems, we can often make deductions about the best methods to use to solve it. For instance, there are some optimisation problems with proven analytic solutions, meaning that if we can find similarities between a task of interest and one of these known tasks, we can adapt existing methods to obtain an optimal solution on our new case.

An optimisation task is given by some objective function f and a set of constraints, such as the objective's valid domain $\mathcal{X}$. To optimise this objective is to discover the point or points in the domain which maximise or minimise the objective function, written as

$$\theta^* = \underset{\theta \in \mathcal{X}}{\operatorname{argmin}} f(\theta). \tag{2.2}$$

A common class we would like to consider is that of linear problems, where we describe an optimisation problem as being linear if the objective function is linear in terms of the tunable parameters of the model. This means that the influence on the objective of changing a particular parameter in some way can directly be predicted, which makes linear systems easier to analyse, and as a result, there is a wealth of tools which have been developed for solving them.

Even in the cases of a linear system, the complexity of the problem can make it difficult to solve, for instance, if the optimisation is under-constrained[2]. In these cases, we tend to have to introduce some form of prior knowledge in order to regularise the problem, i.e. we constrain the solution space to give a better global optima. Linear systems can be solved in this way with ridge-regression, which solves a system by inverting a matrix, and is known to be optimal in the case of a linear system with Gaussian noise and a mean-squared error objective function. In the case where the matrix defining the system is not invertible, we need to introduce a small regularisation term, known as Tikhonov regularisation, to ensure the system is well-posed. For a problem with form

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \|y - X\theta\|_2^2 + \lambda \|\theta\|_2^2, \tag{2.3}$$

the solution can be found as

$$\theta^* = (X^T X + \lambda I)^{-1} X^T y. \tag{2.4}$$

Here, the Tikhonov parameter λ can be seen as introducing a prior on the solution, which penalises large parameter values. While other regularisations can be used, this is the most common, has a closed-form solution, and is often well motivated by wanting our parameters to be small, to avoid stability issues such as overfitting.

In nonlinear optimisation, there are many subclasses with different methods for solving,

[2]Equivalently, the terms ill-conditioned and ill-posed are often used to describe such systems.

and in general the field of optimisation becomes a warren of rabbit holes and special cases. The vast majority of these do not have analytic solutions, meaning we need to rely on numerical methods. While these can allow us to solve a much wider range of problems, they may come perhaps with less intuition for, or fewer guarantees about, the solution.

Typically, numerical methods rely on some form of iterative parameter update rule, which allows us to start from a random initial point in the parameter space, and make changes such that the performance improves. These methods aim to reduce the difficulty associated with high-dimensional parameter spaces, where we cannot hope to evaluate the objective function at every point, and instead must rely on local information to guide our search.

One of the most successful methods is gradient-based optimisation, where we use the gradient of the objective function to guide our search. This is also referred to as 'hill-climbing', where we imagine the objective function as a landscape, and we move in the direction of the steepest gradient, hoping to reach the highest peak, or equivalently in gradient-descent, the lowest valley. The main limitation of gradient-based methods is that they require a differentiable objective function, so we can evaluate not only $f(\theta)$, but also $\partial f/\partial\theta$. These methods can be seen as a way of turning our nonlinear problem into a locally-linear, equivalent problem. By calculating the Jacobian of the objective, we linearise the system around the current point, and then make a step in the direction of the optima in this new problem. These steps will only be well-founded within the region where this linearisation holds, meaning that this only works on continuous, relatively well-behaved functions.

We can extend this region of validity by including even more information about the local landscape, for instance through the second derivative (or Hessian), which tells us about the local curvature of the objective, allowing us to correct for the differences between the linear approximation and the true function. It is therefore useful to classify optimisation algorithms according to the order of the derivatives they have access to, with regular gradient descent being a first-order method, and options such as Newton's method being second-order methods.

Second-order information can help in a range of different ways, beyond just better approximation of the local objective function. If we are optimising a probabilistic model, where we want to find, for instance, a set of parameters for a probability distribution such that we maximise the likelihood of the model predicting some observed data[3], then what we find is that using the regular gradient gives suboptimal results, as the parameter updates we use have a fixed step size and depend on the way we parameterise the distribution. To correct for this, we can use the natural gradient, which gives

[3]A problem which is common in machine learning and will feature heavily in Chapter 4.

the best bounded update according to the difference between the prior and posterior distributions as measured by the Kullback-Leibler (KL) divergence [Ama98]. This no longer depends on the parameterisation, and can be more efficient than standard gradient descent, however it does require the calculation of the Fisher information matrix, a second-order measure which can be expensive to calculate, and can be ill-conditioned in high-dimensional spaces.

While gradient methods are excellent options for optimising complex problems, there are cases where it is not feasible to even calculate the first derivative of the objective function—for instance in the case of a black-box system, where we can evaluate the objective function at a given point, but we do not know the analytic form with which to construct a derivative. This leaves us with zeroth-order, or gradient-free, optimisation algorithms. The simplest naïve approach to gradient-free optimisation is simply to test many parameter configurations and choose the best one. We can do this randomly, or in a more structured way, such as using a grid over the parameter domain. These methods are simple to implement, and can be effective in low-dimensional spaces, but quickly become infeasible as the number of parameters grows. This is due to the curse of dimensionality, where the number of samples required to cover the parameter space grows exponentially with the number of parameters.

We can attempt to approximate gradients by sampling, through the use of finite-difference methods, where we evaluate the system at two points close to each other in parameter space, and use the difference in the objective function to estimate the gradient. However, as with random search, this method becomes infeasible as the number of parameters grows, as a single gradient estimate in an n-dimensional space requires at least $n + 1$ evaluations of the system—an origin, plus one evaluation for each dimension.

Evolutionary algorithms aim to improve on these finite difference methods by taking inspiration from biological evolution, i.e. they use trial-and-error across a population, with stochastic updates and some selection criteria. This can be seen as a refinement of the random or grid search methods, where we still have some element of random sampling, however we condition these random changes on past parameters which performed well.

Genetic algorithms (GAs) treat the parameters as a form of genome, and evolve it according to the principles of genetic evolution, such as mutation, crossover, and selection. Evolutionary strategies (ESs) share similarities with GAs, in that they maintain a population and use selection, however they don't tend to use crossover, instead relying on random mutations and adaptive step sizes to explore the parameter space. Where GAs act on discrete representations of a problem, ESs can act in continuous, real-valued spaces, making them suited to optimisation tasks such as training machine learning

models. There are a number of different ESs, which maintain different statistics over their population, and use different methods for selecting the next generation. A popular and effective method is the covariance matrix adaptation evolutionary strategy (CMA-ES), which maintains a covariance matrix over the population, and uses this to adapt the step size and direction of the mutations. We can view this as using the population to generate a privileged direction in parameter space, similar to finite-difference, however using potentially many fewer samples than the dimension of the space. The disadvantage of course is that the covariance matrix is quadratic in the dimension of the space, leading to poor scaling. Methods such as rank-m evolutionary strategy (Rm-ES) attempt to mitigate this by using a low-rank approximation to the covariance matrix, which can be more efficient in high-dimensional spaces [LZ18].

In the case of these iterative type algorithms, it is common for issues to arise due to the shape of the objective function. One issue which often arises is the presence of local optima, points where the objective landscape is stationary, meaning that gradient-based methods, or methods which rely on searching in local neighbourhoods, can become stuck. This has resulted in the classifying of optimisation problems according to properties such as convexity of the objective, which determines the existence of local optima, and smoothness, which allows us to make deductions about the convergence of iterative methods such as hill-climbing. In general, it is strongly in our interests to use whatever freedoms we have in specifying an optimisation problem to ensure that the objective is as 'well-behaved' as possible, with few local optima, and a smooth, continuous landscape.

Finally, it is worth mentioning Bayesian methods, which are a popular class of algorithm within gradient-free optimisation, and which have seen significant use in a range of ML applications, as they can allow us to find the global optima of a system. They work by modelling a distribution over the parameters of the system, and then sampling from this distribution to determine the next point to evaluate. By modelling the system as a distribution, we get uncertainty estimates which allow us to make optimal choices about parameters to test against the optimisation objective. These methods can therefore be highly effective in systems which are expensive to evaluate, however, due to the modelling method, they can become very expensive as the number of free parameters increases, and the methods for approximating the posterior distribution can be slow to converge. As such, we mention them here for completeness but do not consider them further in this work.

Even with the power of modern computing tools and numerical optimisation, there are limits which stem from the complexity of the problem, such as smoothness of the objective function, the dimensionality of the problem, and the presence of local optima. More practically, when choosing an optimisation strategy, we must consider factors

such as the cost of evaluating the objective function, as this may not be trivial in many cases, and can place limits on the time taken to solve a problem to a given level of accuracy. It is also useful to consider that even if we cannot necessarily find the global optima for a particular problem, we may be able to get estimates of our uncertainty, or the likely difference in performance between our solution and the global optima.

In the context which we primarily care about, neuromorphics, physical systems, and non-differentiable black boxes, we will investigate applying both gradient-based and gradient-free methods, and we will also see how optimisation can be made significantly easier in some cases through inclusion of knowledge of the system, such as having a forward model of the system, or identifying symmetries in the parameterisation and objective function.

2.2.2 What does it mean to learn?

Learning, both in humans and machines, can be seen as a form of optimisation. In machines this is abundantly clear from the way we write learning problems mathematically, and from the tools we use to solve them. For instance, the way we train most machine learning methods comprises a dataset of pairs $\mathcal{D} = \{(x, y)\}$, a parameterised model $f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$, and an objective, or loss, function $L : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$ which compares the label y to the predicted label for a given input, $f_\theta(x)$.

The goal of training is to find the parameters θ which minimise the loss function, and hence make the model f_θ as good at predicting the labels y for the inputs x as possible, over the entire dataset. The global optima would be given as

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{(x,y)\in\mathcal{D}} L(y, f_\theta(x)),$$

which looks very similar to our general optimisation given in Equation 2.2. All three elements dictate what is learnt, and how well the learning generalises to new data.

In humans, learning is a more complex process, with different types of learning, but can still be seen as a form of optimisation. The way we learn to recognise faces, for instance, is very different to the way we learn to ride a bike. In the former, there are inbuilt mechanisms which are specialised to this type of pattern recognition, allowing us to very quickly, and subconsciously, learn to recognise a particular person, even from seeing them only briefly. In the latter, we learn through trial and error, and through practice, to balance on the bike, steer, and pedal. In all cases, we are trying to find the best model of the world, given the data we have available to us. This model is used to make predictions about the future, and to make decisions which maximise our utility. One way of characterising human learning is that we aim to *minimise surprise*, in essence training our model of the world to make good predictions about the future, so that we are able to make inferences about how to affect the world around us in

desirable ways, avoid undesirable outcomes, and generally plan.

Two theories of learning in the human brain include predictive coding, and the free energy principle, where both posit that we learn internal models which can make predictions about the world around us from observed data. Predictive coding theorises that the human brain learns by constantly making such predictions and then updating prior beliefs to match the new data [RB99; LMK22]. However, such a scheme also needs to be able to discriminate between good new data, and erroneous information, and hence it is important to maintain a discriminative model which can maintain prior beliefs and stability, avoiding constantly changing the world model in response to noise. This was put forward in Friston's free energy principle [Fri10]. The free energy is equal to $-L(x)$, where

$$L(x) = \ln p(x) - D_{KL}(q(z|x) \,||\, p(z|x)), \tag{2.5}$$

D_{KL} is the KL divergence between the approximate posterior and the true posterior, and $\ln p(x)$ is the log-likelihood of the data under the model.

This equation is also known as the evidence lower bound (ELBO), which plays a prominent role in variational Bayesian methods as a lower-bound on the log-likelihood of observing some data, i.e. $\ln p(x) \geq L(x)$ for some model p. Maximising the ELBO can be seen as encouraging alignment of the model with the true data through the first term in Equation 2.5, while encouraging the model to be stable, and not shift too much in response to new data, via the second term. The ELBO is useful in optimising various distributional models, commonly seen in ML architectures such as variational autoencoders, which learn low-dimensional latent representations of some data distribution, or in the expectation-maximisation (EM) algorithm, which is used to find a maximum likelihood estimate for parameters θ in a model which depends on unobserved, hidden variables z [DLR77; Ber20].

While it is perhaps unsurprising that the fields of neuroscience and artificial learning systems would converge on similar ideas, the fact remains that the ways these two types of systems actually update their models in response to new data are very different.

2.3 Training neuromorphic systems

Optimisation is hard even in well studied or perfectly modelled systems. In systems with only partial information, such as many physical or neuromorphic computing systems, this becomes even harder.

In conventional computing, we can get away without knowing the exact voltages at the gates of the transistors, due to the robust design of the system, and inbuilt fault tolerance. This removes the influence of noise and small perturbations, resulting in a system which we, as end users, can consider to be entirely deterministic. This

determinism, coupled with the high-level abstractions we use for programming, allow us to program the analytic equations for the derivatives of various numeric operations, to save intermediate values produced during a computation, and then combine these to calculate numeric derivatives. This is the basis of the backpropagation algorithm, which is used to train most modern machine learning models.

In neuromorphic systems, we often don't have this luxury, for the precise reason that the abstractions and generality provided in conventional computers are often part of the bottleneck we are trying to overcome by using a neuromorphic computer in the first place. In most systems we want to study, there will be some hidden information, state or dynamics we can't measure or estimate.

Consider the simple case of an information-carrying wave propagating through some bulk scattering medium. While we may be able to write down the equations for encoding information in the carrier, and the method of detection at the output of the system, the scattering process is a mystery to us. If the system is linear, and we can probe the system through controlling the encoded information, we can perform an inverse retrieval to understand the scattering process, but even this only provides a model of this particular linear system. If the medium physically moves, we can't easily make predictions about the new system's behaviour. This assumes that the physical process even *can* be measured and modelled. In many cases there are fundamental limitations due to noise, complexity, or even, as will be seen in Chapter 3, quantum effects. In these cases we have no choice but to accept that the system is non-deterministic, and find optimisation strategies which are robust to this. Aside from the fact that modelling can be hard or even impossible, attempting to model and simulate the physical system often defeats the point of using it in the first place.

2.3.1 Gradient-based optimisation

The first approach to training such systems is to use established gradient-based optimisation, which requires either a model or simulation of the physical system of interest, which can be used as a proxy for training, while the real system is reserved only for inference. This 'training in the dream' ([HS18]) or *in silico* training falls under the category of *digital twins*, which have proven successful in a range of problems that require interaction with the real world, such as control, robotics and autonomous systems [Mat+22; FJL24]. This approach does enable fast learning, and a greater understanding of the dynamics and behaviour of the system during the learning process. On the other hand, it rather defeats the point of using a physical or neuromorphic system in the first place—if we can solve the problem we care about in a simulation then why bother with the physical system at all.

Additionally, training in a simulation assumes that the model we use is well aligned with the real world system, such that we see the same performance on both. This is

almost never the case, even if the real system has high signal-to-noise ratio, stability and determinism, and misalignment can lead to suboptimal performance, especially during gradient-based training. Consider an approximate model $\hat{m}$ which aims to reproduce a real system m, assume that both are smooth and that we can bound the error such that $|m(x) - \hat{m}(x)| \leq \varepsilon \; \forall x \in \mathcal{S}$ where $\mathcal{S}$ is the support of m. In general there is no such bound on the error of the gradient $|m'(x) - \hat{m}'(x)|$. This can be seen by considering m to have some oscillation about a mean $\mu(x)$ i.e. $m(x) = \varepsilon \sin(kx) + \mu(x)$. Then $\hat{m} \coloneqq \mu$ satisfies the error bound, but the gradient error grows arbitrarily large as $k \to \infty$. This can cause stability issues when training, and while typically we would choose to work with 'well-behaved' physical systems where these effects are limited, the compounding of small errors in gradient propagation through larger systems can lead to failure just as in the case of vanishing or exploding gradients in conventional deep learning.

One improvement to *in silico* training can be made by only using the model for the backward pass, and using the physical system for all forward passes. This was introduced as physics aware training (PAT) [Wri+22], where a differentiable forward-model of a neuromorphic system is first generated, and then used in backpropagation with inputs measured from the physical system.

Regardless, whether using sim-to-real transfer or more sophisticated techniques like PAT, a forward model needs to be built. This can be a significant challenge depending on the system at hand, with difficulties ranging from having a good approximate model but being unable to account for small misalignments and noise, to simply being intractable due to the scale of the physical system. Developments in simulations, and especially physics informed neural networks [RPK19], differentiable physics engines [VV22], and neural differential equations [Che+18], all contribute to more effective models, but the fundamental challenge remains.

In some cases there are still advantages of using a simulated model, for instance in ultra-fast applications beyond conventional computers' capabilities, a fast neuromorphic system that we are confident we can accurately model may be the only viable option, and we can accept slow training times for the gain in deployed inference speed.

Beside modelling, it may in some cases be possible to engineer a system such that it not only computes the forward model, but in doing so also produces the required derivatives as outputs. This is highly dependent on the physics of the system being used, with difficulty increasing as we work with systems with fewer abstractions and predictable dynamics. An example of such a system comes from diffractive optics, where a system was designed carefully such that the same hardware used in a forward pass could also be used to propagate errors back to parameters [Guo+21]. Such systems are, however, an exception, and while this may prove to be an important technique for future systems, we would like to be able to explore other, easier options in the interim.

2.3.2 Gradient-free optimisation

Avoiding modelling is important for making the best use of a neuromorphic computing system. It avoids issues with misalignment between model and hardware, and potentially expensive simulations which scale poorly with the complexity of the system. We therefore should aim to optimise directly on the physical hardware, which requires using some form of zeroth-order method. These only require the ability to evaluate the system on different inputs, and so are compatible with physical computing, however the lack of a gradient means that choosing where in the parameter space to sample is not an easy task. Bayesian methods provide a principled approach to sampling the parameter space in order to minimise uncertainty, but range from expensive to unfeasible in high-dimensional spaces. There are a range of iterative approaches using different heuristics to improve sampling efficiency, of which evolutionary strategies are a popular and well established option, but all suffer as the parameter space grows.

Beyond evolutionary strategies and common zeroth-order optimisation techniques, there are a range of methods developed specifically for training machine learning models and neuromorphic systems, which directly target some of the main issues unique to these fields.

The first which we discuss is feedback alignment (FA), which has subsequently been extended to direct feedback alignment (DFA) [Nøk16; Lil+16; NE19]. Loosely speaking, these methods allow us to estimate an update to the parameters in a nonlinear system, by propagating the error on the output backwards through a different model than the forward system. At first sight, this may seem ill-posed—there is no reason to believe that using an unrelated backpropagation model should produce any meaningful result. The key to the method's success lies in the form of the error propagation model, which is a usually chosen to be a fixed random projection. Direct feedback alignment [Nøk16] illustrates learning systems can even adapt to wildly different error signals, by mapping gradients in the output of a deep network back to the parameter space through fixed random projections which, by the Johnson-Lindenstrauss lemma [JL84], roughly preserve orientation. There are many demonstrations that approximate gradients are 'good enough' for training, and while training performance may be degraded, they still outperform zeroth-order methods.

This motivates the idea of general gradient alignment, and whether we can perform optimisation without the exact gradient, provided we can calculate some approximate pseudo-gradient, which is close enough to the true value that the system can learn to accommodate the differences, similarly to DFA. Already we have seen that there are several analytically justified gradient-based updates which can be used to train networks, such as the regular first-derivative, and second derivative methods such as the natural gradient. Indeed, there are several examples where systems have been

successfully trained without exact gradients, and these ideas will be explored further in Chapter 4 [Lil+16; Lau+20; Wri+22; Fil+22].

Aside from pseudo-gradient-based approaches, there are a number of methods which have taken inspiration from biological systems, and which have been shown to be effective in training neural networks. There is no known mechanism in the brain for error signals to propagate backwards through the same pathways as the forward signal, and that the brain is able to learn effectively without this mechanism. This highlights that the brain does not learn via backpropagation as it is used in ML, and prompts us to consider more biologically-plausible rules. These work either by allowing for different pathways for update information to flow through, similar to FA, or use local learning rules which only require information about the local state of the system to update the parameters. These types of rules are harder to map to our original optimisation, as they don't have access to the objective function to gain any estimate for good updates. Instead, they need to rely on local, general objectives which, when optimised for, hopefully correlate with performant behaviour on the true task. An example of this which we have already discussed is the idea of surprise minimisation in the brain, where individual neural circuits aim to update their weights to better match the predictions they made before observing a particular stimulus. While this doesn't directly correspond to optimising for tasks like recognising faces, the emergent behaviour of the system can end up performing well on these types of tasks.

A particular form of local learning is described by Hebb's rule, which is often summarised as "neurons which fire together, wire together" [Heb05]. This describes the tendency for connections between neurons to strengthen when one is active directly before, and contributes to, the firing of the other. This can be seen as a form of unsupervised learning. Another form was put forward by Hinton in 2022, where he described the forward-forward algorithm [Hin22]. This is another unsupervised approach based on contrastive learning, where we train a model to discriminate between different classes of inputs in such a way that a model for the underlying distribution emerges. Instead of using a forward evaluation pass, followed by a backward pass with updates, forward-forward performs two forward passes, one with the normal input, and second with an out-of-distribution, or corrupted input. The local parameters are then updated to better discriminate between the two, in a process which is independent of the updates made to other parameters in the network. This has been shown to perform well on certain types of task, including classification, and spurred a range of new contrastive learning approaches to neuromorphic computing, including application of the algorithm to both regular deep learning models and optical neuromorphic systems [Mal+23; Ogu+23].

The final approach we will discuss here is the use of meta-learning. As with other computing systems, we distinguish between the architecture of a neuromorphic system, and the way it is programmed or trained. There are however some approaches that

blur this line by directly optimising network structures, using techniques such as learnt optimisers or model discovery methods [ZL16; Bel+17; Wic+17; EMH19; Hos+22]. It is important to note that even in these cases, the methods build on decades of experience developing effective models—we don't throw out CNNs and transformers in the hope of discovering something new by throwing neurons at the wall and seeing what sticks. That said, learnt optimisers, and more generally, meta-learning, which aims to learn some aspect of the learning process itself, have shown promise in optimising network architectures and hyperparameters, with the principle being that algorithms discovered through learning hold the potential to be more effective than those designed by hand [And+16; LM17; Bel+17; KS20; Met+22; Che+23a].

Meta-learning can take several forms, such as learning base models which can be trained once on a range of tasks, such that they can easily be fine-tuned with few examples on downstream tasks, or learning algorithms which are capable of tuning model parameters for us, such as the learnt optimisers mentioned above [Hos+22]. Meta-learning will form the foundation of the work in Chapter 4.

2.3.3 Reservoir computing

We have established that programming is a key challenge in physical neuromorphic computing, which arises ultimately from imperfect knowledge of the underlying system. This presents in two ways—an inability to accurately model the internal dynamics of the system, and the difficulties in accessing or measuring the specific internal system state during operation. Without these, we cannot calculate numeric derivatives for use in gradient-based optimisation, and of the remaining zeroth-order optimisation techniques, most do not scale to high dimensional systems.

However, all is not lost—we can neatly side-step these issues by restricting the way we use the physical platform slightly, through the theory of reservoir computing [Cuc+22]. Instead of having a tunable system, we choose to use a fixed, random physical system with interesting dynamics which we think will be suited to solving the types of task we are interested in. In order to solve different tasks, we use a tunable matrix multiplication on the output of the reservoir which gives us our final output. The fact the system is linear in its parameters allows us to train it using well-established linear regression techniques, such as ridge-regression. This consists of a single matrix inversion and a couple of multiplications, performed once. Avoiding iterative training rules and complex algorithms both simplifies the training greatly, and satisfies our low energy consumption criteria. That said, this linearity does mean that if the necessary dynamics are not all generated by the reservoir, then they cannot be learnt by the system. Much of the work in reservoir computing therefore focusses on choosing or designing systems which have a wide range of complex dynamics, such that there is either a surjective mapping to, or synchronisation with, the task's target dynamics. Reservoir computing is a blanket

term, covering several classes of algorithm capable of using fixed complex systems for computation.

Extreme learning machine. Extreme learning machines (ELMs) use a single random layer as a nonlinear projection of the input data,

$$h = \sigma(x),$$

for input data $x \in \mathbb{R}^{d_x}$, output activation $h \in \mathbb{R}^{d_h}$ and nonlinearity $\sigma : \mathbb{R}^{d_x} \to \mathbb{R}^{d_h}$ [HZS06]. The nonlinearity σ is random, high-dimensional, and left untuned. The network is trained on a particular task using linear regression on the output h, giving us a weight matrix W which is used to get the trained output of the system as $y = W \cdot \sigma(x)$. It can be shown that the ELM is a universal interpolator, but not approximator, as in the case of the multilayer perceptron (MLP). This means that it can interpolate functions, but will not necessarily generalise well to new data.

Echo state network. By adding recurrence, we allow fading memory, making the reservoir computer suitable for use on sequential data such as time-series problems. There are many forms for these systems, but a general forward model can be written as

$$h_{t+1} = (1 - \alpha)\, h_t + \alpha\sigma\left(W_h h_t + W_x x_t\right). \tag{2.6}$$

Here, $\alpha \in [0, 1]$ balances between the previous state and the new input, giving some control over the stability of the system. W_h and W_x are the internal state and input weight matrices, respectively, and σ is our nonlinearity as before.

During training, we take labelled data $\{(x_t, y_t)\}$, pass each x_j through the system, and collect them into a design matrix D_{ij}, where each column is the output of the network at time j. We then train a linear model on the output of the reservoir to predict the target y_t, i.e. we learn the weights W_{out} in the equation.

Compared to ELMs, the addition of the echo state property and fading-memory make echo state networks (ESNs) universal function approximators [GO18; GO21]. A second consequence of adding recurrence is that we can model the reservoir as a dynamical system, where Equation 2.6 can be seen as the discrete-time approximation of a continuous-time system,

$$\frac{dh}{dt} = \alpha\left(\sigma\left(W_h h(t) + W_x x(t)\right) - h(t)\right). \tag{2.7}$$

As such, metrics from dynamical systems theory such as Lyapunov exponents and synchronisation can be used to characterise the behaviour of the system.

ESNs have strong connections with concepts such as RNNs and neural differential equations (NDEs), which are prevalent in the traditional machine learning literature. In

an RNN, we repeatedly apply the same neural network to a sequence of inputs, with the output at each iteration being fed back into the network as persistent state. In this way, an RNN can be seen as a differentiable, trainable ESN. NDEs on the other hand are a more general class of model suited to solving problems which can be described using differential equations. Here, a neural network parameterises the form of a differential equation, which can then be solved using standard numerical methods. They can be seen as the continuous extension of recurrent and residual networks.

Liquid state machine. Liquid state machines (LSMs) are a distinct approach developed at the same time, but independently of, the ELM. The principle here is largely the same, where computation takes place in a large randomly initialised recurrent network, with linear learnt parameters used to train on a given task. The key difference is that the LSM are biologically inspired, using spiking neurons, making them suited to continuous-time tasks.

Chapter 3

Photonic quantum reservoir computing

Quantum computing is an example of unconventional physical computing with huge potential for increasing computational performance on specific tasks. However, a large gap has developed between our theoretical understanding of quantum information processing, and our current experimental capabilities, as efforts to build scalable hardware with acceptable noise and error rates continue. Here, we propose a new way of doing application-specific computing using quantum resources which aims to be practical, scalable, and implementable with existing commercial hardware. In particular, we use the infinite dimensional Hilbert space generated by multiphoton interference and photon number resolved detection to create a reservoir computer, a novel paradigm for dynamics-driven machine learning detailed in the previous chapter, with rich dynamics, while considering realistic quantum state preparation, losses, and detection efficiencies. This work is partly detailed in the paper [Ner+24].

3.1 Introduction

Chapter 2 introduced the concept of reservoir computing as an efficient method of solving various problems which is well suited to unconventional computing hardware, however we have yet to discuss an implementation in a physical system. Indeed, one of the main advantages is that it allows us to leverage the properties of complex, physical systems for computing, by providing an easy and efficient method for training. As reservoir computers are linear in their tunable parameters, there is no requirement to be able to interrogate the internal reservoir dynamics, allowing us to use complex dynamical systems without needing to model them or make measurements of internal state.

There are many examples of physical reservoir computers which have been developed in a range of substrates over the last decade or so, showing state-of-the-art performance in certain classes of task—for instance time series prediction, classification, and function

interpolation [Raf+20; Por+21; Gar+23; McM23; Bru19; Tan+19]. A particularly interesting example is quantum reservoir computing, which extends the usual reservoir computing paradigm to use a non-classical reservoir. While quantum computing and reservoir computing have individually seen significant research and successes, each has its own challenges which can limit their practical usage. Conventional reservoir computing efficiently exploits the complex dynamics of classical systems for computation, including noise or instabilities, removing the need for difficult programming or training. However, as with any analogue computing system, it requires a one-to-one mapping or synchronisation with the target dynamics, and so it can be challenging to identify or design systems which have useful dynamics in the first place. Quantum computing, on the other hand, exploits a richer set of dynamics than is available in classical computing, through the use of quantum phenomena such as superposition and entanglement, however current approaches struggle to realise a large-scale, practical system [Wai24; Fed+22]. This is due to either needing a highly stable, noise-robust platform on which to implement hand-crafted quantum algorithms, or needing new data-driven programming methods which are robust to non-ideal, real-world experimental conditions.

Combining the two fields, we can ask whether the dynamics of a quantum system would prove useful in a reservoir computer, and whether reservoir computing's simple training process can allow us to make use of non-ideal quantum systems which otherwise would be unsuitable for computation. We are not the first to ask these questions—recently there have been several proposals for quantum reservoir computing systems, in a range of substrates including photonics [Gar+23; Muj+21; Xio+23; Gho+20; AAM24; GLG23].

In this chapter, we put forward a new approach for building a quantum reservoir computer, specifically an extreme learning machine [HZS06]—a feedforward (and degenerate) variant of reservoir computing with proven performance in regression and interpolation tasks in optical substrates [PMC21; ZLM23; SRF24]. Our method applies photon number resolved detection (PNRD) to access a combinatorially large Hilbert space of quantum measurements. The next section begins with background on the state of quantum computing and quantum machine learning research, motivating the use of reservoir computing, and continues with an overview of the proposed system. Section 3.3 builds up the necessary mathematical tools to model our quantum reservoir computer, followed by Section 3.4, showing results in numerical simulations, and Section 3.5, comparing measurements for both classical and quantum states of light, and characterising their performance in interpolation tasks. The chapter concludes by providing an overview of initial experiments and areas for further investigation.

Throughout the chapter, we highlight how scaling to a combinatorially large Hilbert space allows our reservoir computer to access a high-dimensional space in which we can perform machine learning. We pay particular attention to the practical realisation of this system, analysing the impact of noise, detector performance and sampling on the

performance of the system. This work is partly detailed in the paper [Ner+24].

3.2 Background

3.2.1 Quantum machine learning

There are many parallels with quantum computing and classical physical and neuromorphic computing. In particular, both fields deviate from the conventional von Neumann, *in silico* computing paradigm, instead opting to implement new forms of computing using novel physical dynamics. While the motivation for classical neuromorphic computing mainly hinges on improved efficiency over conventional computing approaches, quantum computing aims to implement entire new classes of algorithms and thus open up new classes of problems, previously intractable with classical computers.

Quantum computing approaches have seen significant investment of late, however still face challenges in scaling and noise robustness, which delay their practical usage and limit access. Quantum computing as a field started out with a focus on implementing quantum gates, analogues of the digital logic gates from which all conventional CPUs are built. These controllable gates modify the state of a qubit, a two level quantum system which provides the quantum analogue of a classical bit of information. By controlling these gates, we can program a device with quantum algorithms which can, in principle, solve problems which are intractable for classical computers.

It is clear that this scheme is heavily inspired by the way we build conventional computers, and yet there have been several bottlenecks in the development of practical systems, such that after decades of research there are still very few large scale, practical quantum computers available. Even those that do exist are limited to the order of hundreds of usable qubits [Ich+24].

The difficulty in building such a computer stems from the range of criteria which an implementation needs to satisfy [DiV00]. The operations provided by the device must be universal, and there must be a way to initialise the system in a known state. For practical usage, the system must also be scalable in the number of quantum resources, as even the simplest quantum computing algorithms require many qubits [Dal+20]. It also needs to guarantee temporal stability and robustness to noise, as noise and decoherence can quickly destroy the quantum information stored in the system. Finally, it must allow us to encode information in the quantum states, and read out results through measurements, in a robust, repeatable way.

One of the main advantages of quantum systems for computing is their ability to use quantum effects to improve the time complexity of solving certain mathematical problems. There exist certain problems, such as the classic example of factoring large numbers, which are intractable for classical computers, but which can be solved on a

quantum computer. In these cases, where quantum computers allow us to solve problems which otherwise would take infeasible lengths of time on any classical system, there is said to be a 'quantum advantage'. Some in the field argue a quantum advantage if a system simply outperforms an equivalent classical counterpart, although this requires some justification of what a fair choice of equivalent classical system may be [Fed+22]. As a result, 'quantum advantage' has become something of a loaded term, with many potential interpretations depending on the area of research and the way in which a quantum system is compared with a classical equivalent. However, it is important to point out that there can still be an advantage or performance improvement in using quantum systems for computing, even in cases where the problem can already be solved classically. This may come in energy efficiency, speed, utilisation of available hardware resources, or even the ability to process data which is already encoded in a quantum state.

3.2.2 Current approaches to quantum machine learning

There have been many approaches to building quantum computers, although currently there are a few substrates used to implement qubits which dominate the research landscape. Qubits based on the Josephson effect rely on superconductivity to generate a two level quantum system. On-chip systems can be manufactured with existing fabrication technologies, allowing for a high level of control, however the need for cryogenic cooling makes them difficult to implement, and large scale coupling of qubits remains hard [Kja+20].

Trapped ions are an alternative means of implementing qubits, where individual ions are trapped in an electromagnetic field and manipulated using lasers, however they are difficult to couple and scale into a feasible quantum computer, and also require cooling in order for quantum effects to dominate over thermal variations [Bru+19].

A third approach is to use photons as qubits, where a quantum state of light is used to encode information [SP19]. Photons can be easily manipulated using existing optical technologies while avoiding the need for cryogenic cooling, however they are difficult to store, making quantum optical memory a challenge. That said, light has many degrees of freedom for encoding information, quantum states can be generated relatively easily, and single-photon detectors are becoming increasingly available. Despite photons not easily interacting with one another, we can couple optical qubits through probabilistic processes such as scattering, interference effects, or nonlinear optical processes. These properties make quantum light an attractive option in certain applications.

Even with a wide range of available platforms, each with their own pros and cons, many of the current gate-based, universal approaches to quantum computing face

difficulties. These may be in realising memory, scaling quantum resources and coupling, or engineering challenges such as cryogenic cooling, noise mitigation, and achieving the robustness required for repeatable experiments. Although these approaches show great promise to scale as a result of years of hard efforts [Mad+22; Arr+21; Bog+20; Luo+23; Kja+20], their current difficulties place the technology out of reach for most laboratories.

Many of the issues stem specifically from the *universality* requirement, as the need to implement a full set of quantum gates and operations places significant constraints on the system. If we can relax this constraint, then we may be able to dodge some of the larger obstacles and use quantum resources for useful computation. There are several approaches which eschew gates and universality in favour of 'simpler' quantum systems, enabling nearer-term, application specific quantum computing.

One such example is Boson sampling, a non-universal quantum computing approach which provides a quantum advantage on a specific task: sampling from the output distribution of number resolved measurements when n indistinguishable particles are scattered into m modes. Boson sampling is #P-hard, intractable with current algorithms and classical computers, meaning that a physical system which solves this problem truly does provide a quantum advantage [CC18; TT96; AA11]. A natural platform for implementing a Boson sampling system is photonics, as linear scatterers are relatively easy to construct, and single photons can be generated to serve as the Bosonic inputs to the network. This has been demonstrated experimentally on large scales, most notably in [Wan+19a; Zho+20].

Another example which has seen significant interest in recent years is quantum machine learning (QML), which introduces principles from machine learning to quantum computing [Bia+17]. QML architectures take many forms, and it is useful to classify them according to three properties as illustrated in Figure 3.1: whether they use quantum or classical inputs, a quantum or classical substrate, and whether they are used for solving quantum or classical problems [Muj+21].

The range of combinations here illustrates the potential ambiguity when we talk about quantum machine learning—it is important to specify which elements of the system use quantum resources. For instance, while some QML approaches aim to provide speed-ups for classical tasks, others are more suited to processing information which is already encoded in quantum states, leading to new methods for extracting and manipulating quantum information [Gho+19; Hua+22].

Within QML, we can build quantum neural networks (QNNs)—parametrised artificial neural networks (ANNs) formed from linked quantum states and their interactions, which may be trained analogously to conventional ANNs. While QML algorithms may be implemented on gate-based quantum computers, there is potential for new,

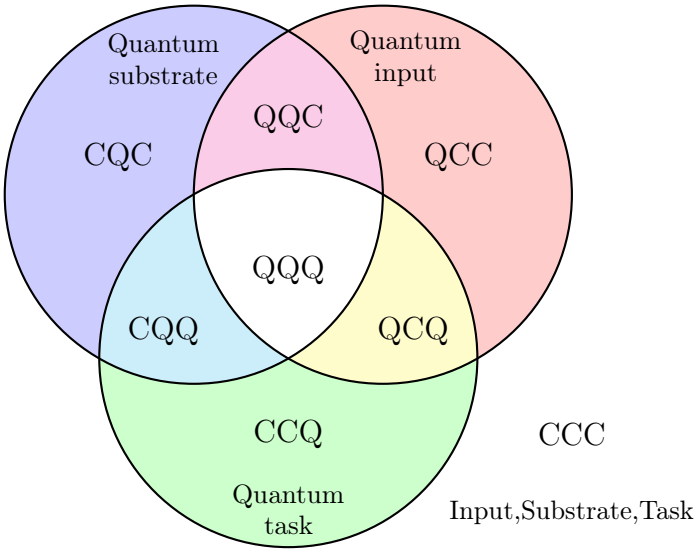


Figure 3.1: **Quantum machine learning taxonomy.** Taxonomy of different machine learning systems based on the quantum or classical (Q/C) nature of the input, substrate and task being performed.

quantum neuromorphic hardware which can be specialised to various tasks on classical or quantum data with minimal experimental overhead, where the same arguments for classical neuromorphic hardware (see Section 2.1.2) hold here.

In this case, reservoir computing becomes particularly relevant—while some quantum systems may be easy to build in the lab, they may also be difficult to model and train using conventional machine learning training schemes. Reservoir computing simplifies the training, and reduces computational and environmental costs compared to more typical ANNs which require iterative training of models which are nonlinear in their parameters.

3.2.3 Proposed QRC overview

Our approach to quantum reservoir computing takes inspiration from Boson sampling, where the required complexity is generated through a quantum state scattering in a random linear optical network. We do not however adhere to the strict requirements of the Boson sampling problem, and instead use the network as a reservoir which, under photon number resolved detection, generates a high-dimensional output space suitable for use in reservoir computing.

Due to the general modelling approach we employ, we can realise the linear optical network with a range of different physical systems, such as bulk scattering media or multimode fibres (MMFs) [Def+16; Raf+20]. Similarly, there are many degrees of freedom open to us for encoding information, such as phase or polarisation. The benefits conferred by the use of quantum resources and photon number resolved (PNR) detection in the following analysis therefore apply in a range of physical reservoir implementations.

We examine a range of quantum and classical states of light such as Fock and coherent states. Our system is compatible with all variants shown in Figure 3.1, where our substrate (e.g. Fock or coherent states, with or without PNR detection), input (e.g. real values encoded in polarisation, or quantum information encoded in quantum input states) and task (e.g. linear quantum operations, or classical ML tasks) can all be either quantum or classical. In our analysis, we primarily consider the CQC regime, using a quantum substrate to solve classical function interpolation tasks.

Our analysis shows that while performance improves with average photon number, we can achieve good performance with states which can be easily generated in a modern photonics laboratory. We pay attention to the implications of noise and detection losses on the performance of the system, incorporating these in simulation, and we introduce and test a set of metrics to quantify the behaviour of different designs in light of these conditions.

3.3 THEORY

In this section we develop the theory needed to construct our quantum reservoir computer. This is composed of three main elements. We cover the types of states we can encode information in and the form of these encodings, the theory of linear optical networks which we use to implement the reservoir which transforms the states, and finally the detection of the resulting output states.

3.3.1 INPUT STATES

In order to describe a range of quantum and classical states, we introduce notation for the three main classes of state from which we will build all other states from: Fock states, distinguishable photon states, and coherent states.

For an m-mode state, we denote indistinguishable photon occupancies with an ordered set $\boldsymbol{n} = (n_1, \ldots, n_m)$, distinguishable photon occupancies with $\boldsymbol{d} = (d_1, \ldots, d_m)$, and coherent states with coherent state amplitudes $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$. When necessary to specify values for these, we will write indistinguishable states using regular integers, distinguishable states using subscripted 1s, where each 1_k represents a single photon which is distinguishable from any photon with different subscript k, and coherent states using italicised decimals. For example:

- Fock state: $\boldsymbol{n} = (2, 1, 0, 0)$ implies a three photon state $|n_1, n_2, n_3, n_4\rangle = |2, 1, 0, 0\rangle$, where all photons are indistinguishable.
- Distinguishable state: $\boldsymbol{d} = (2, 1, 0, 0)$ implies a three photon state $|d_1, d_2, d_3, d_4\rangle = |1_0 1_1, 1_2, 0, 0\rangle$, where all photons are distinguishable.
- Coherent state: $\boldsymbol{\alpha} = (0.5, 1, 0, 0.1)$ implies a coherent state $|\alpha_1, \alpha_2, \alpha_3, \alpha_4\rangle =$

$|0.5, 1.0, 0, 0.1\rangle$, where all constituent photons are indistinguishable.

From this we can write more complex states, such as photon-added coherent ($|\boldsymbol{\alpha} + \mathbf{1}\rangle = |\alpha_1 + 1, \ldots, \alpha_m + 1\rangle$) and subtracted coherent states ($|\boldsymbol{\alpha} - \mathbf{1}\rangle = |\alpha_1 - 1, \ldots, \alpha_m - 1\rangle$). For instance, $|0.5 + 2, 1.0 + 1, 0, 0\rangle$ should be read as two photon-added coherent states with amplitudes 0.5 and 1.0 denoted by the italicised decimals, and 2 and 1 additional photons denoted by the roman integers.

In terms of creation and annihilation operators $a^\dagger$ and a, we can write a Fock state as

$$|n\rangle = \frac{a^\dagger}{\sqrt{n}} |n-1\rangle = \frac{a^{\dagger n}}{\sqrt{n!}} |0\rangle ,$$

while the equivalent multimode Fock state is given as

$$|n_1, n_2, \ldots, n_m\rangle = \prod_{i=1}^{m} \frac{a_i^{\dagger n_i}}{\sqrt{n_i!}} |0, 0, \ldots, 0\rangle , \tag{3.1}$$

where we index the creation operators according to the mode they act on.

The same form applies for distinguishable photon states, with the creation operators distinct according to each photon's distinguishability.

Similarly, we can express multimode coherent states in the Fock basis as the tensor product of sums of Fock states,

$$\begin{aligned} |\alpha_1, \alpha_2, \ldots, \alpha_m\rangle &= \prod_{i=1}^{m} e^{\alpha_i a_i^\dagger - \alpha_i^* a_i} |0, 0, \ldots, 0\rangle \\ &= \bigotimes_{i=1}^{m} \sum_{j=0}^{\infty} e^{-\frac{|\alpha_i|^2}{2}} \frac{{\alpha_i}^j}{\sqrt{j!}} |j\rangle . \end{aligned} \tag{3.2}$$

A coherent state has mean photon number $\langle n \rangle = |\alpha|^2$, and the probability of detecting n photons is given by the Poisson distribution. Coherent states are useful for our purposes as we can consider them in both the quantum and classical pictures, allowing us to compare the performance of our reservoir computing system with both number resolved detection and intensity detection. They are considerably easier to generate in the lab than Fock states, simply using a laser, while the latter requires nonlinear processes or single photon sources.

We can artificially add photons to a coherent state, giving a photon-added coherent state (also referred to in this work as a hybrid state), which is no longer describable

with classical fields, written as

$$|\boldsymbol{\alpha}+\boldsymbol{n}\rangle = |\alpha_1+n_1,\ldots,\alpha_m+n_m\rangle = \bigotimes_{i=1}^{m}\sum_{j=0}^{\infty} e^{-\frac{|\alpha_i|^2}{2}}\frac{{\alpha_i}^j}{\sqrt{j!}}\frac{\sqrt{(j+n_i)!}}{\sqrt{L_{n_i}(-|\alpha_i|^2)n_i!j!}}|j+n_i\rangle \tag{3.3}$$

where $L_n(x)$ is the Laguerre polynomial of degree n[1]. Setting $\alpha_m = 0$ simplifies Equation 3.3 to the standard form of a Fock state, while setting $n_m = 0$ simplifies it to a multimode coherent state.

Finally, N00N states, which will feature in later discussion, are given as a superposition of two maximally bunched Fock states,

$$\psi_{\text{N00N}} = \frac{1}{\sqrt{2}}(|N,0\rangle + |0,N\rangle). \tag{3.4}$$

3.3.2 Linear optical networks

In order to perform computation with these states, we need to encode information in their degrees of freedom, perform some sort of useful transformation on them, and then measure the output. These transformations can be modelled using the theory of linear optical networks (LONs).

LONs are useful for several reasons. Firstly, they are highly versatile—any multimode linear optical system can be described within the theory of LONs. As a result, one can experimentally realise a LON in a wide range of physical systems, including multimode fibres, complex scattering media, linear photonic circuits, and many other optical systems, with some examples illustrated in Figure 3.2. Due to their ubiquity, they have been extensively studied, with both classical and quantum descriptions well understood in the literature. They are passive, multimodal systems, making them ideal candidates for energy-efficient and scalable physical computation [McM23], and they can be designed to realise any finite unitary transformation [Rec+94], making them highly general and allowing a LON-based reservoir computer to be tailored to specific applications. Indeed, they have already been used to demonstrate several forms of quantum computation [TR19; Gar+14; PJ95; Mee+21; Wan+19a; Bro+13; Mad+22; Mar+20a; AA11].

In order to use LONs as the random layer in a reservoir computer, we need to understand how we can model them, which degrees of freedom are available to us, and how we can simulate their effect on the quantum states of light which carry our information.

[1]The normalisation can be found by writing the $\sqrt{j+1}\ldots\sqrt{j+n_i}$ factors produced by the creation operators as a binomial coefficient, and then using the expression $\sum_n x^n/n!\binom{n+m}{m} = e^x L_m(-x)$.

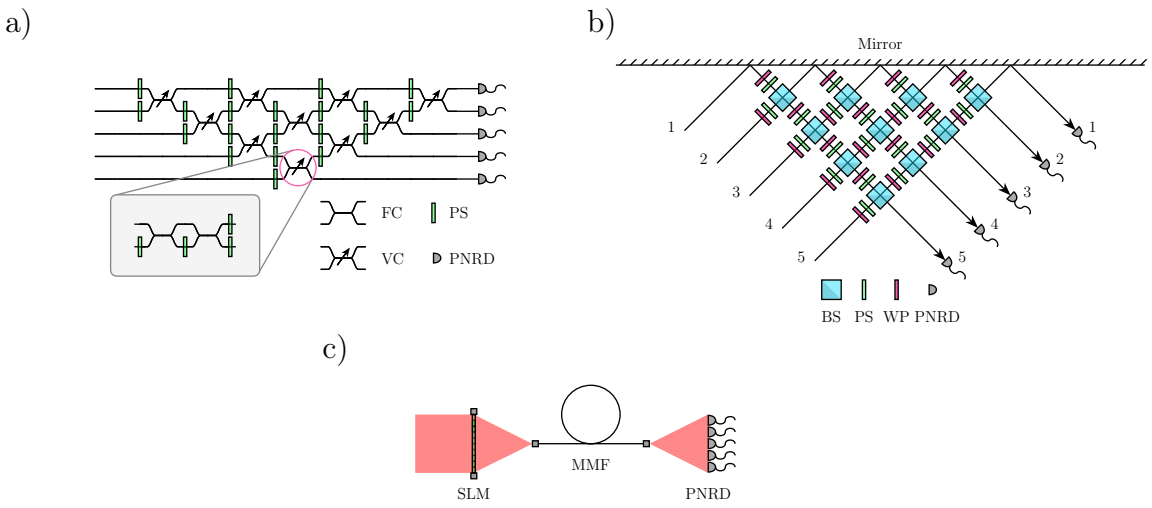


Figure 3.2: **Example LON implementations.** a) Photonic circuit LON with variable couplings (VC) (composed of 50:50 fixed couplings (FC) and phase shifters (PS)). b) LON implemented with a triangular arrangement of beamsplitters (BS), phase shifters (PS) and waveplates (WP). c) LON implemented as a multimode fibre (MMF), where a spatial light modulator (SLM) determines the network input. All examples can be paired with PNRD to measure the output state.

3.3.3 MODE COUPLING

A LON can be modelled simply as a matrix mapping a set of input modes to output modes. A general m-mode scattering matrix takes the form of

$$S = \begin{pmatrix} s_{1,1}e^{i\phi_{1,1}} & s_{1,2}e^{i\phi_{1,2}} & \cdots & s_{1,m}e^{i\phi_{1,m}} \\ s_{2,1}e^{i\phi_{2,1}} & s_{2,2}e^{i\phi_{2,2}} & \cdots & s_{2,m}e^{i\phi_{2,m}} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,1}e^{i\phi_{m,1}} & s_{m,2}e^{i\phi_{m,2}} & \cdots & s_{m,m}e^{i\phi_{m,m}} \end{pmatrix},$$

with $s_{i,j} \in [0,1]$, $\phi_{i,j} \in [0, 2\pi)$. The complex amplitudes introduce interference effects when coupling different modes which, along with energy conservation, constrains the possible values of the coefficients and phases in S. Conservation of energy constrains the absolute values of the eigenvalues of S to be less than or equal to one, with the case where they are all exactly 1 corresponding to a unitary, lossless transformation, and any other case representing a lossy process.

As independent phase shifts in each mode don't affect the overall eigenvalues of the matrix, we can use pre- and post-multiplication with diagonal phase-shift matrices to reduce the number of free parameters in S. The pre-matrix is given as $P = \text{diag}\left(e^{i\theta_i}\right)$ with $\theta_i = \phi_{i,1}$ and the post-matrix as $P' = \text{diag}\left(e^{i\theta'_j}\right)$ with $\theta'_j = \phi_{1,j} - \phi_{1,1}$, such that we can write S as

$$
\begin{aligned}
S &= PS'P' \\
&= \begin{pmatrix} e^{i\theta_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & e^{i\theta_m} \end{pmatrix} \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,m} \\ s_{2,1} & s_{2,2}e^{i\phi'_{2,2}} & \cdots & s_{2,m}e^{i\phi'_{2,m}} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,1} & s_{m,2}e^{i\phi'_{m,2}} & \cdots & s_{m,m}e^{i\phi'_{m,m}} \end{pmatrix} \begin{pmatrix} e^{i\theta'_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & e^{i\theta'_m} \end{pmatrix},
\end{aligned}
$$

where $\phi'_{i,j} = \phi_{i,j} - \theta_i - \theta'_j$. We can therefore limit ourselves to considering matrices of the form S', and drop the prime notation for simplicity.

If we apply S to a set of general, classical input fields $E^{\text{in}} = (A_1 e^{i\delta_1}, \ldots, A_m e^{i\delta_m})$ to get $E^{\text{out}} = SE^{\text{in}}$, energy conservation dictates that $\sum_m |E_m^{\text{out}}|^2 \leq \sum_m |E_m^{\text{in}}|^2$. By setting all A_m to zero except A_j, we can show that $\sum_i |s_{i,j}|^2 \leq 1 \ , \ \forall j$. In the case of a lossless process, these inequalities become strict equalities.

If we have a lossy m-mode network, we can construct an equivalent unitary process by introducing ancillary modes to represent the extra degrees of freedom which couple energy out of the lossy system. This can be seen by considering a loss in a single mode as a variable 2×2 unitary matrix (simply a rotation matrix) coupling the original mode to a virtual loss mode. In doing so we must be careful to avoid reusing virtual loss modes due to interference effects, which could couple losses back into the network. The naïve solution to this is to add a new loss mode for every point loss in the network, however this is prohibitive in simulation due to the computational complexity scaling with mode number. Hernández and Liberal show that we can always accomplish this with only m additional modes, and provide a method for constructing the new $2m$-mode unitary S through a singular value decomposition of the lossy matrix $S' = U \cdot \text{diag}\,(\sigma_i) \cdot V^\dagger$ [HL22],

$$
S = \begin{pmatrix} S' & -A \\ A & S' \end{pmatrix}, \tag{3.5}
$$

where $A = U \cdot \text{diag}\left(\sqrt{1 - |\sigma_i|^2}\right) \cdot V^\dagger$. We append m zeros to our original input state, evaluate the effect of S, and discard the final m values in our output to recover the exact same output as in the lossy case.

With the ability to consider both lossless and lossy processes, the final component we need is a method of constructing scattering matrices in a principled, general way. The key here is found in the work of Reck et al., who showed that we can decompose any m-mode unitary into the product of a series of 2-mode unitaries, which simplifies the problem of constructing a network to one of understanding only the possible 2-mode couplings [Rec+94].

Our general 2×2 can be written as

$$S = \begin{pmatrix} r_1 & t_2 \\ t_1 & r_2 e^{i\phi} \end{pmatrix},$$

where we denote reflectances and transmittances as r_i and t_i, and the phase difference ϕ. Uppu et al. [Upp+16] demonstrate that in the general lossy case we must obey the inequality

$$t_1^2 r_2^2 + t_2^2 r_1^2 + 2 t_1 r_2 t_2 r_1 \cos\phi \leq (1 - t_1^2 - r_1^2)(1 - t_2^2 - r_2^2). \tag{3.6}$$

Given a free choice of $r_1 \in [0, 1]$, $t_1 \in \left[0, \sqrt{1 - r_1^2}\right]$ and $\phi \in [0, 2\pi]$, we can rearrange in order to define the valid support for t_2 and r_2 as

$$\begin{aligned} 1 &\leq \frac{1 - r_1^2 - t_1^2}{r_2^2 + t_2^2 - 2 r_1 r_2 t_1 t_2 \cos(\phi) - r_1^2 r_2^2 - t_1^2 t_2^2} \\ d_2^2 &\leq \frac{1 - d_1^2}{1 - \frac{d_1^2}{d_2^2 d_1^2} |t_1 t_2 + r_1 r_2 e^{i\phi}|^2} \\ &\leq \frac{1 - d_1^2}{1 - d_1^2 |\cos(\gamma_1)\cos(\gamma_2) + \sin(\gamma_1)\sin(\gamma_2) e^{i\phi}|^2} \\ &\leq \frac{1 - d_1^2}{1 - d_1^2 \frac{1}{2}(1 + \cos(2\gamma_1)\cos(2\gamma_2) - \cos(\phi)\sin(2\gamma_1)\sin(2\gamma_2))}, \end{aligned} \tag{3.7}$$

where we have switched coordinate system with $d_i^2 = t_i^2 + r_i^2$, $\gamma_i = \arctan(r_i / t_i)$. We are now free to choose $\gamma_2 \in [0, \pi/2]$, which defines the ratio r_2/t_2 and in turn determines the maximum value of d_2.

Equation 3.7 corresponds to a series of ellipses in the $t_2 r_2$-plane, parameterised by t_1, r_1 and ϕ, which enclose the valid choices of t_2 and r_2. The equation for the ellipse is

$$\frac{(t_2 \cos(\beta) - r_2 \sin(\beta))^2}{a^2} + \frac{(r_2 \cos(\beta) + t_2 \sin(\beta))^2}{b^2} = 1$$

and the parameters can be calculated (see Appendix B.1) as

$$\beta = \frac{1}{2} \arctan\left(\cos(\phi) \frac{2 r_1 t_1}{r_1^2 - t_1^2}\right) \tag{3.8a}$$

$$a^2 = \frac{2(1 - r_1^2 - t_1^2)}{2 - r_1^2 - t_1^2 - |r_1^2 + t_1^2 e^{i2\phi}|}, \tag{3.8b}$$

$$b^2 = \frac{2(1 - r_1^2 - t_1^2)}{2 - r_1^2 - t_1^2 + |r_1^2 + t_1^2 e^{i2\phi}|}. \tag{3.8c}$$

Figure 3.3.b demonstrates these ellipses for a variety of (t_1, r_1, ϕ) values.

In the lossless case $d_1 = d_2 = 1$, which is only satisfied when $\phi = \pi$ and $\gamma_2 = \gamma_1$, making

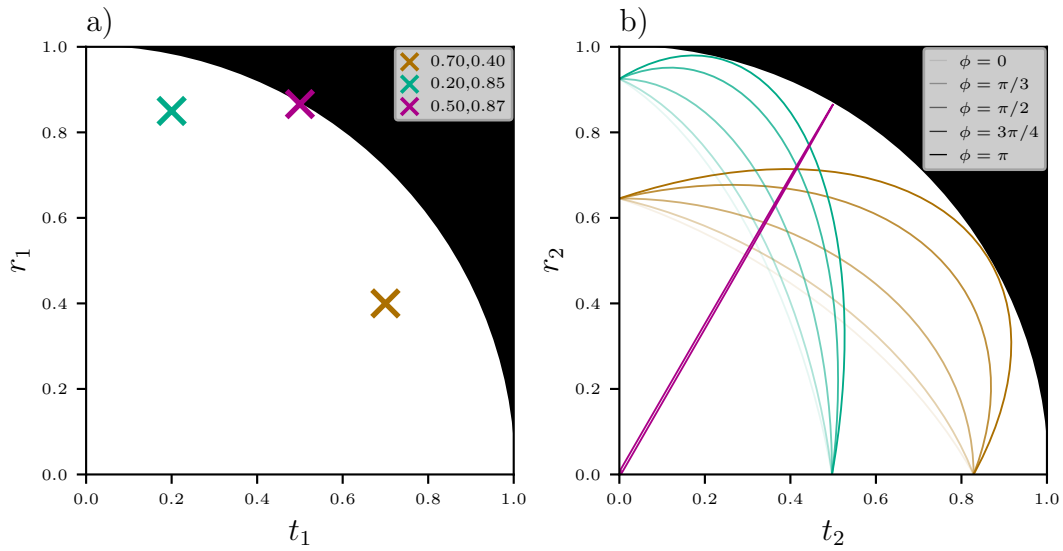


Figure 3.3: **Valid mode coupling support.** Support for the five degrees of freedom of a 2-mode lossy mode coupling, according to energy conservation conditions. a) White region is the set of possible (t_1, r_1) values, with three example pairs shown. b) Corresponding supports for (t_2, r_2) for each pair, considering different values of ϕ, denoted by ellipse transparency. All valid (t_2, r_2) lie in the white region, and are further constrained to lie within the region bounded by the ellipse given by the set (t_1, r_1, ϕ), according to Equations 3.8.

the most general lossless 2×2 coupling

$$S = \begin{pmatrix} r & t \\ t & -r \end{pmatrix}, \tag{3.9}$$

with $r \in [0, 1]$ and $t = \sqrt{1 - r^2}$.

This general coupling can be viewed in a LON as the combination of fixed 50:50 beamsplitters and phase shifts in a Mach-Zehnder interferometer setup,

$$\begin{aligned} S &= P \cdot BS \cdot P \cdot BS \cdot P \\ &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & e^{i(\pi - 2\gamma)} \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} e^{i(\gamma - \pi/2)} & 0 \\ 0 & e^{i\gamma} \end{pmatrix} \\ &= \begin{pmatrix} \sin\gamma & \cos\gamma \\ \cos\gamma & -\sin\gamma \end{pmatrix} \\ &= \begin{pmatrix} r & t \\ t & -r \end{pmatrix}. \end{aligned} \tag{3.10}$$

The phase shift at the start and end of the sequence allow the reintroduction of the phases which were extracted in the earlier pre- and post-multiplication, giving this as the most general form for the lossless 2×2 unitary.

3.3.4 Sampling random mode couplings

We would like to be able to construct random LONs, to reflect some of the random scattering systems we find in the real world which we would like to use for computation. In order to generate a random reservoir we need to be able to couple modes randomly, which is achievable now that we have defined the support for both the lossy and lossless cases of the 2-mode coupling. A natural start is to sample uniformly across the support, as this will give us a broad range of possible networks to explore.

Sampling uniformly in the lossless case is simple—the support is one dimensional, and we only need to choose a point uniformly on the unit circle restricted to the positive quadrant. This can be achieved with $\gamma \sim \mathcal{U}(0, \pi/2)$, $\phi = \pi$ and $r/t = \tan(\gamma)$.

Sampling uniformly across the support in the lossy case is harder, as the support is not a simple geometric shape. In practice, methods such as rejection sampling can be used to generate beamsplitters by uniformly sampling the 5-dimensional hypercube enclosing the support, and rejecting the sample if the validity criterion (Equation 3.6) is violated.

While certain platforms capable of realising this reservoir computing scheme are able to generate any arbitrary unitary transformation (e.g. programmable photonic circuits), the effect of the distribution of the random reservoir on computation performance is especially important when considering non-configurable reservoirs (e.g. multimode fibres).

A prime example can be seen by considering a variable 2-port beamsplitter. In the case of indistinguishable photons incident on both ports, we observe bunching, where the output photons are increasingly likely to be detected in the same output port, as the beamsplitter tends to the 50:50 balanced case (i.e. $r = t$). This is a quantum interference effect known as the Hong-Ou-Mandel (HOM) effect [HOM87].

The bunching can be seen by considering a lossless beamsplitter,

$$S = \frac{1}{\sqrt{2}} \begin{pmatrix} r & \sqrt{1-r^2} \\ \sqrt{1-r^2} & -r \end{pmatrix},$$

which gives the operator relations

$$a_1^\dagger = \frac{r b_1^\dagger + \sqrt{1-r^2} b_2^\dagger}{\sqrt{2}},$$
$$a_2^\dagger = \frac{\sqrt{1-r^2} b_1^\dagger - r b_2^\dagger}{\sqrt{2}}.$$

If we input the state

$$\psi_{\text{in}} = |1,1\rangle_{\text{in}} = a_1^\dagger a_2^\dagger |0,0\rangle$$

then we can calculate the output state as

$$\begin{aligned}\psi_{\text{out}} &\propto \frac{1}{2}(rb_1^\dagger + \sqrt{1-r^2}b_2^\dagger)(\sqrt{1-r^2}b_1^\dagger - rb_2^\dagger)\,|0,0\rangle \\ &= \frac{1}{2}(r\sqrt{1-r^2}{b_1^\dagger}^2 + (1-r^2)b_2^\dagger b_1^\dagger - r^2 b_1^\dagger b_2^\dagger - r\sqrt{1-r^2}{b_2^\dagger}^2)\,|0,0\rangle \\ &= \frac{r\sqrt{1-r^2}}{\sqrt{2}}\left(|2,0\rangle_{\text{out}} - |0,2\rangle_{\text{out}}\right) + \frac{1-2r^2}{2}\,|1,1\rangle_{\text{out}}\,. \end{aligned} \tag{3.11}$$

We see that as $r^2 \to 0.5$, the commutation of different $b_i^\dagger$ causes only bunched states to remain, the basis of the quantum Hong-Ou-Mandel effect.

HOM is useful as it is a fundamentally quantum effect, and so it can be used to verify the quantum nature of a system. It is also a useful tool for characterising the indistinguishability of photons, which is important for many quantum information protocols. Here however, Equation 3.11 implies that the way we sample our random coupling matrices can have a large effect on the bunching of the states in the reservoir, and consequently the types of output states we measure.

3.3.5 Polarising optical components

While the 2-mode lossless beamsplitter shown in Equation 3.10 can be viewed as a variable beamsplitter acting on two spatial modes, we would also like to be able to model common polarising elements, as polarisation is a convenient degree of freedom with which we can encode information.

In the classical picture, we can represent polarisation as a two-mode system, in the horizontal and vertical polarisation basis. Jones vectors (E_H, E_V) let us describe polarisation states in terms of the electric field in each mode, and 2×2 Jones matrices represent the action of birefringent optical elements on these states.

A phase delay in the V mode is given by the matrix

$$P_\eta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\eta} \end{pmatrix},$$

while a rotation of coordinate system is given by rotation matrix

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

With these two matrices we can describe any birefringent element, using the form

$$B(\phi,\theta,\eta) = P_\phi R_\theta P_\eta R_\theta^\dagger P_\phi^\dagger = \begin{pmatrix} \cos^2\theta + e^{i\eta}\sin^2\theta & (1-e^{i\eta})e^{-i\phi}\cos\theta\sin\theta \\ (1-e^{i\eta})e^{i\phi}\cos\theta\sin\theta & \sin^2\theta + e^{i\eta}\cos^2\theta \end{pmatrix} \tag{3.12}$$

where θ is the fast-axis angle with respect to the H-axis, ϕ the circularity, and η the retardation between fast and slow axes.

The rotation matrices in Equation 3.12 are equivalent to the general lossless beamsplitters in Section 3.3.3. This means that not only can we be sure that this is the most general birefringence form, but, provided we have some way of inducing a controllable, continuous-valued phase retardation between two polarisation axes, we can implement any arbitrary polarisation transformation. Methods of implementing continuous controlled birefringence include using the Pockels effect, liquid crystal waveplates, or liquid crystal on silicon (LCoS) spatial light modulators (SLMs).

If we only wish to be able to generate an arbitrary polarisation state starting from a known input state, we can restrict ourselves to using half- and quarter-waveplates, linear retarders (i.e. $\phi = 0$) with ($\eta = \pi$) and ($\eta = \pi/2$) respectively. The case of $\phi \neq 0$ gives a circular or elliptical retarder and will largely be ignored here. The Jones matrices for the half- and quarter-waveplates are given by

$$B_{\lambda/2}(\theta) = B(0, \theta, \pi) = \begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix}$$

and

$$B_{\lambda/4}(\theta) = B(0, \theta, \pi/2) = \frac{e^{-i\pi/4}}{\sqrt{2}} \begin{pmatrix} i + \cos 2\theta & \sin 2\theta \\ \sin 2\theta & i - \cos 2\theta \end{pmatrix}$$

respectively.

Polarising beamsplitters are 4-mode devices, with 2 polarisation modes for each of the 2 spatial modes. Throughout all remaining sections, we order the tensor product of spatial and polarisation modes such that mode i is the $\lfloor i/2 \rfloor^{\text{th}}$ spatial mode,

$$\begin{cases} H, & i \mod 2 = 0 \\ V, & i \mod 2 = 1 \end{cases}, \text{ polarisation mode.}$$

A classical state would then be represented as $(E_{1H}, E_{1V}, \ldots, E_{mH}, E_{mV})$.

While general 4×4 scattering matrices can be derived following the same procedure as in Section 3.3.3, the number of parameters to solve for grows as $2n^2 - 2n + 1$, so for simplicity we restrict ourselves to polarising beamsplitters of the form

$$\begin{pmatrix} r_H & 0 & t_H & 0 \\ 0 & r_V & 0 & t_V \\ t_H & 0 & t_H e^{i\phi_H} & 0 \\ 0 & t_V & 0 & t_V e^{i\phi_V} \end{pmatrix}. \tag{3.13}$$

We assume no coupling between polarisation modes, meaning Equation 3.13 can be

decomposed as a set of two commuting 2×2 general beamsplitters, acting on H and V independently. In the case of non-polarising beamsplitter acting on polarised light, we constrain $r_H = r_V$, $t_H = t_V$, and $\phi_H = \phi_V$.

The final piece of polarisation machinery we need is the Stokes parameters, another convenient way of describing polarisation states, especially due to their relationship with the Poincaré sphere representation, i.e. the position on the Poincaré sphere is given by the final three Stokes parameters normalised by the first parameter. The Stokes parameters and Jones vectors are related by transformations

$$\begin{aligned} S_0 &= |E_H|^2 + |E_V|^2 \\ S_1 &= |E_H|^2 - |E_V|^2 \\ S_2 &= 2\mathrm{Re}(E_H E_V^*) \\ S_3 &= 2\mathrm{Im}(E_H E_V^*) \end{aligned}$$

and

$$\begin{aligned} E_H &= \frac{S_2 + iS_3}{\sqrt{2}\sqrt{S_0 - S_1}} \\ E_V &= \frac{\sqrt{S_0 - S_1}}{\sqrt{2}}. \end{aligned}$$

Note, the Stokes parameters carry no information about global phase, meaning that two equivalent Stokes states may be generated by different Jones vectors. Phase is important in our LON as it dictates the interference, therefore we must always consider the full Jones vector when comparing states.

For systems with many spatial modes, we build up linear optical networks using sequences of components acting on one or two modes at a time, where waveplates couple polarisation modes, beamsplitters couple spatial modes, and phase shifts introduce delays between modes. The full S matrix is built using Givens rotations G_m (Equation 3.14), where for a set of optical elements $\{S^{(i)}\}$ applied sequentially to a state, coupling k_i different spatial and polarisation modes $\{(m_{i,1}, \ldots, m_{i,k_i})\}$, we embed the 2×2 coupling matrix contributed by each element in an $m \times m$ identity matrix, and then multiply all of these embedded matrices together in order, giving

$$S = G_m(S^{(i)}; (m_{i,1}, \ldots, m_{i,k_i})) \cdots G_m(S^{(0)}; (m_{0,1}, \ldots, m_{0,k_0})),$$

where

$$G_m(S_{2\times 2};(p,q)) = \begin{array}{c} \begin{matrix} 1 & & p & & q & & m \end{matrix} \\ \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & S_{11} & \cdots & S_{12} & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & S_{21} & \cdots & S_{22} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} 1 \\ \\ p \\ \\ q \\ \\ m \end{matrix} \end{array} . \tag{3.14}$$

3.3.6 Scattering of quantum states

There are two classes of simulation we can perform with quantum states scattering through a LON: strong simulation and weak simulation. The strong simulation aims to use a classical algorithm to compute the full output state which arises from a given input state being scattered by the network, while the weak simulation only aims to sample from a set of measurement outcomes on this output state. Boson sampling is exactly this weak simulation problem, where the measurement is a number resolved, multiphoton coincidence detection, and can be calculated efficiently using Clifford and Clifford [CC18; CC20]. In our case, we wish to calculate the actual probabilities of these outcomes, and so need to perform the strong simulation[2].

Consider an m-mode LON with scattering matrix S. In the classical case, the effect of this LON on a set of input fields injected into each mode is simply the matrix product of S and the input fields E^{in},

$$E^{\text{out}} = S \cdot E^{\text{in}}. \tag{3.15}$$

In the quantum case, the LON maps states from the input Hilbert space to the output space, acting as a unitary operator $\hat{U}$. Working in the Fock basis, we can write $\hat{U} = \sum_{ij} U_{ij} \left|\boldsymbol{n}_i\right\rangle \left\langle \boldsymbol{n}_j\right|$ for some consistent enumeration of all m-mode Fock state occupancies $\{\boldsymbol{n}_j\}$.

Valid occupancies (and equivalently, measurement outcomes) correspond to weak integer compositions, i.e. all possible solutions of $x_1 + x_2 + \cdots + x_m = n$ where $x_i \geq 0$, $\forall\ i \in \{1,\ldots,m\}$, interpreted as all possible ways of distributing n indistinguishable objects across m bins. There are $M_{n,m}$ of these weak compositions, given by the multiset coefficient[3]

$$M_{n,m} = \left(\!\!\binom{m}{n}\!\!\right) = \frac{(n+m-1)!}{(m-1)!n!}. \tag{3.16}$$

[2]Technically, any real experiment would make estimates through samples, meaning the weak simulation would be sufficient. However, the strong simulation provides a complete deterministic characterisation of a given system, allowing for deeper analysis.

[3]The multiset coefficients, a generalisation of the binomial coefficients, are given as $\left(\!\!\binom{n}{k}\!\!\right) = \binom{n+k-1}{k}$.

One effect of $\hat{U}$ being unitary is that for a particular Fock state input, total photon number is conserved—despite $\hat{U}$ acting on an infinite dimensional space, it is block diagonal, with finitely many non-zero scattering amplitudes which must be calculated for a given input Fock state. This makes it possible to calculate exact output states for a Fock input state.

The main convenience of the Fock basis however is the connection it affords between $\hat{U}$ and S, as S describes the effect of the LON on the creation operators: it maps input mode creation operators $\{a_j^\dagger\}$ to output mode creation operators $\{b_j^\dagger\}$ according to

$$b_i^\dagger = \sum_j S_{ij} a_j^\dagger. \tag{3.17}$$

As discussed in Section 3.3.3, any valid S is either unitary, or can be made unitary through the introduction of m ancillary modes according to Hernández and Liberal's singular value decomposition [HL22]. We can therefore invert Equation 3.17 to express the input operators as $a_i^\dagger = \sum_j S_{ij}^\dagger b_j^\dagger$.

In order to calculate a specific scattering amplitude U_{ij} in $\hat{U}$, we could first express the input Fock state as a product of creation operators acting on the vacuum, and then replace the input creation operators with the linear sum of the output creation operators according to S. Then we collect common powers of output creation operators and identify the coefficients U_{ij},

$$\begin{aligned}
|\boldsymbol{n}_j\rangle^{\text{in}} &= \prod_{k=1}^{m} \frac{\left(a_k^\dagger\right)^{n_{jk}}}{\sqrt{n_{jk}!}} |0\rangle \\
&= \prod_{k=1}^{m} \frac{\left(\sum_l S_{kl}^\dagger b_l^\dagger\right)^{n_{jk}}}{\sqrt{n_{jk}!}} |0\rangle \\
&\overset{!}{=} \sum_i U_{ij} \prod_{k=1}^{m} \frac{\left(b_k^\dagger\right)^{n'_{ik}}}{\sqrt{n'_{ik}!}} |0\rangle \\
&= \sum_i U_{ij} \, |\boldsymbol{n}'_i\rangle^{\text{out}},
\end{aligned}$$

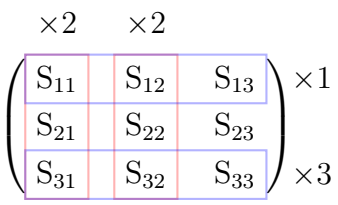
where we differentiate between the input and output bases.

A complete derivation of this is given in [Sch04], where it is shown that these coefficients can be compactly written as the permanent of a sub-matrix $\Lambda_{\boldsymbol{n},\boldsymbol{n}'}$, constructed from the elements of S and the input and output photon occupancies $\boldsymbol{n}$ and $\boldsymbol{n}'$. The columns of these Scheel matrices $\Lambda_{\boldsymbol{n},\boldsymbol{n}'}$ are the n_k repetitions of the k^{th} column of S for all $k \in \{1, \ldots, m\}$, while the rows are the n'_k repetitions of the k^{th} row of S. For example,

given $\boldsymbol{n} = (2, 2, 0)$ and $\boldsymbol{n}' = (1, 0, 3)$, the Scheel matrix is

$$\Lambda_{\boldsymbol{n},\boldsymbol{n}'} = \begin{pmatrix} S_{11} & S_{11} & S_{12} & S_{12} \\ S_{31} & S_{31} & S_{32} & S_{32} \\ S_{31} & S_{31} & S_{32} & S_{32} \\ S_{31} & S_{31} & S_{32} & S_{32}. \end{pmatrix},$$

coming from the sectioning of the scattering matrix as



The permanent of an $N \times N$ matrix A_{ij} is given as

$$\operatorname{perm}(A) = \sum_{\sigma \in S_N} \prod_{i=1}^{N} A_{i\sigma(i)},$$

where S_N is the symmetric group. While structurally this is very similar to the matrix determinant,

$$\det(A) = \sum_{\sigma \in S_N} \operatorname{sgn}(\sigma) \prod_{i=1}^{N} A_{i\sigma(i)},$$

the presence of the permutation signature in the determinant allows for efficient calculation by cancellation of terms, in as little as $O(N^{2.373})$ time using efficient matrix multiplication [Le 14]. In contrast, calculation of the permanent is challenging, with the most common exact method, Ryser's algorithm, calculating the permanent in $O(2^N N^2)$ time

$$\operatorname{perm}(A) = \sum_{S \subseteq \{1,\ldots,N\}} (-1)^{|S|+N} \prod_{i=1}^{N} \sum_{j \in S} A_{ij}.$$

Modifying Ryser slightly[4], we can get complexity of $O(2^N N)$, however this is still intractable for large matrices [Nij14, pp. 220–221]. See Appendix B.2 for details on the complexity of these algorithms.

---

[4] If we process subsets S in Gray code order, then each sum over j goes from summing $|S| - 1$ terms to adding a single new term to the result from the previous subset calculation.

This allows us to write the full form for the scattering amplitude between two Fock states as

$$\langle \boldsymbol{n}' | \hat{U} | \boldsymbol{n} \rangle = \left(\prod_{j=1}^{m} n_j! \right)^{-1/2} \left(\prod_{j=1}^{m} n'_j! \right)^{-1/2} \operatorname{perm}(\Lambda_{\boldsymbol{n},\boldsymbol{n}'}). \tag{3.18}$$

Even with modified Ryser, this is intractable for large n or m. Through the remainder of this work, we use the strong linear optical simulator (SLOS) algorithm, an improvement upon the basic permanent approach with computational complexity $O(nM_{n,m})$, which is linear in the number of output Fock states, allowing us to simulate states of modest $m \lessapprox 18$ and $n \lessapprox 15$ in reasonable time [Heu+23b].

The distinguishable photon states can be calculated similarly, except we prevent coherent interference between the different photons by performing an incoherent sum, i.e. the probability of detecting $\boldsymbol{n}' = (n'_1, \ldots, n'_m)$ photons in the output modes based on an input distinguishable state $|\boldsymbol{d}\rangle$ is proportionate to $\operatorname{perm}(|\Lambda_{\boldsymbol{d},\boldsymbol{n}'}|^2)$, where the absolute value squared is taken element-wise. Finally, general quantum states can be simulated by first writing the density matrix ρ in the Fock basis and then calculating the new state as $\hat{U}\rho\hat{U}^\dagger$.

3.3.7 ENCODING DATA

In order to use linear scattering of a quantum state as a reservoir computer, we need to encode data somehow, such that by measuring the output state in some basis, we realise a high-dimensional nonlinear random projection of this data. We have to define the free parameters which we can vary to encode information, and we can classify these based on whether they modify the input state or the mode coupling matrix.

In the first case, there are several degrees of freedom which parameterise the input state, including polarisation, phase, spatial mode, photon number, wavelength, or time delay. The total photon number and number of independent modes are discrete variables and determine the size of the space of PNR measurement outcomes, so we leave these as hyperparameters, fixed for a particular reservoir implementation. Varying wavelength or time delay introduces a continuous degree of distinguishability. Seeing as we already can model distinguishable photons, we choose to ignore these for now.

The second option, parameterising the mode coupling matrix, can be done in simulation by varying the values of the internal couplings and phase delays. In an experiment, this would be done in a programmable photonic circuit, or a beamsplitter network with rotating waveplates.

When using phase and polarisation (mathematically equivalent as shown in Section 3.3.5), the distinction between parameterising the input state and the mode coupling matrix disappears, as we can start from any state with fixed polarisation, and apply parameterised waveplates to generate an arbitrarily polarised version of the state. These

waveplates can be incorporated as a new matrix $E(x)$ premultiplied on the original, fixed mode coupling matrix S, giving a new parameterised matrix $S(x) = S \cdot E(x)$.

Considering for a moment a single general birefringent waveplate acting on a fixed input state, as we continuously vary the waveplate's parameters, the output polarisation state traces a smooth trajectory through the space of polarised states, which can be visualised on the Poincaré sphere. The elements of the resulting $S(x)$ will be sums of products of the sinusoids present in the waveplate matrices, Equation 3.12.

A complete characterisation and optimisation over possible encodings is beyond the scope of this work, however, we do propose a few natural options. We will define a layer $E(x)$, capable of realising a range of different Poincaré sphere trajectories. We model this layer as a set of controllable waveplates which encode input data x in the polarisation degree of freedom of a fixed input state, before it is injected into the random optical network. Each spatial mode in $E(x)$ consists of a quarter-waveplate followed by a half-waveplate, parameterising the full $E(x)$ matrix with a total of $2m$ waveplate angles. We write these as a function of our input data $(\theta_{Q,m}(x), \theta_{H,m}(x))$. Any encoding scheme using phase or polarisation states will be periodic, so without loss of generality we constrain our input space to the domain $[-1, 1]$. For approximating non-periodic functions, we can restrict the usable input range to $[0, 1]$.

We now define a set of encodings which map our input data to these waveplate angles, parameterised by $(\rho_m, \xi_m, \gamma_m, \nu_m)$ for each spatial mode m, as

$$x' = \mathrm{mod}(1 + x + \rho_m, 2) - 1 \tag{3.19a}$$

$$\theta_{Q,m}(x) = \xi_m(1 + 2x' - 4x'\gamma_m\mathcal{H}(x'))\frac{\pi}{4} \tag{3.19b}$$

$$\theta_{H,m}(x) = \Big(\nu_m x' + \xi_m(\gamma_m\mathcal{H}(x')) + 2\nu_m x'\gamma_m\mathcal{H}(x') - 2\nu_m x'\Big)\frac{\pi}{4}, \tag{3.19c}$$

where $\mathcal{H}$ is the Heaviside step function. This allows us to generate a range of closed trajectories on the Poincaré sphere, as shown in Figure 3.4, periodic on the interval $[-1, 1]$.

$\rho \in [0, 2\pi]$ is a phase shift, $\nu \in \mathbb{Z}$ determines the number of cycles we make around the pole of the Poincaré sphere in one cycle $[-1, 1]$, while $\xi \in \{0, 1\}$ determines whether we leave the equator and include elliptical polarisations. Finally, γ allows us to choose whether the spiral trajectory is interleaved with itself ($\gamma = 1$), or overlaps itself ($\gamma = 0$). In the case where we overlap, there are multiple input x values which map to the same polarisation state. While this on its own would cause problems distinguishing certain inputs, the use of multiple ports to encode the same data allows us to break the degeneracy. The interleaved case avoids overlap, at the expense of discontinuities

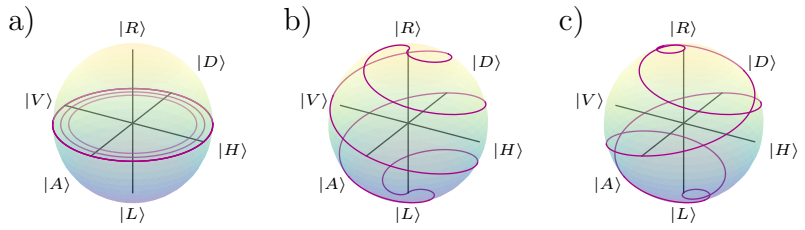


Figure 3.4: **Poincaré encodings.** Poincaré sphere representations of a single port's polarisation trajectory as a function of the input parameter, for various encoding schemes. a) Multilinear[5] ($\xi = 0, \gamma = 0, \nu = 3$). b) Interleaved spiral ($\xi = 1, \gamma = 1, \nu = 3$). c) Overlapping spiral ($\xi = 1, \gamma = 0, \nu = 3$).

in the Jones vector representation of the state. While this introduces interesting new dynamics in the system, we note here that it is not well-behaved under the Fourier analysis approach discussed later in this chapter.

3.3.8 Output state measurement

So far we have seen how to calculate the output state of the LON for a given input state, however to use this output state for computation we need to measure it, or at least some property of it. One of the primary motivations for investigating this system in the first place is the combinatorial scaling of the output Fock space generated by the LON. Since we already calculate the output state in the Fock basis, it makes sense that we detect in this basis.

We can do this with photon number resolved detection (PNRD), a projective measurement where the probability of measuring a particular photon number occupancy $\boldsymbol{n}' = (n_1, \ldots, n_m)$ across m detectors aligned with the LON output modes (our measurement outcomes) is given by the square of the amplitude of the $|\boldsymbol{n}'\rangle$ component in the output state. To make these measurements simpler, we will also only consider polarisation independent detection, tracing out this degree of freedom. This will generate a set of outcomes, with each Fock basis component in the output state contributing probability mass to a specific outcome.

A single PNR measurement of the state is not informative, as it cannot discriminate different input data values. By repeated sampling however, we can estimate the probabilities of these outcomes as a function of the encoded data. These probabilities *will* discriminate inputs and can therefore be used as the reservoir outputs.

As we aim to produce a system which is accessible and implementable with existing technologies, we need to consider the impact of this sampling, along with realistic, non-ideal detection, to understand a real-world implementation of the reservoir computer

[5]The inset curves here are illustrative only. ν simply controls the number of cycles around the equator, all states exist on the surface of the Poincaré sphere.

where we don't have access to deterministic simulated values.

Let's first consider non-ideal detection, and formulate a model which captures noise and losses. Assuming perfect photon sources, noise in a LON can arise in two ways: losses in the network and imperfections in the detectors. While losses in the network will change the output statistics compared to a lossless network, we have already established a method of representing them in Section 3.3.3, so we disregard them in this section. Instead, here we will focus on losses due to non-unit detector quantum efficiency and additional detections due to dark counts. These effects cause misclassification of a Fock basis component as the wrong measurement outcome, and can be modelled probabilistically. They also change the set of available outcomes: for instance, detector losses mean that 0- and 1-count detections are possible for a 2-photon Fock state, which under perfect detection would only ever yield 2-count events.

We consider the case of one PNR detector at each of the LONs output ports, and assume identical independent noise processes for each. All detectors have quantum efficiency η and average dark counts per acquisition window μ, which follow independent and stationary binomial and Poisson processes respectively.

A final consideration is that we may not always wish to estimate the probabilities of every available outcome for a particular state. Instead, we post-select on outcomes of interest. In most of the following discussion, this is done by only considering detection events with a total photon number in a certain range, and discarding any which fall outside this range. This both simplifies the measurement process, and can help limit the impact of detector losses.

3.3.9 Analytic quantum reservoir computer model

We have finally reached the point where we have all the necessary tools at our disposal to write the complete analytic model of the quantum reservoir computer (QRC), taking input data x all the way to sampled estimates of the outcome probabilities. Here we will formally define the mathematics used to simulate the full reservoir computer.

We start by defining our measurement outcomes as the set of all polarisation independent, photon-number resolved detection events,

$$\mathcal{O}^{[a,b]} = \left\{ (n_1, \ldots, n_m) \,\middle|\, \sum_i n_i \in \{a, \ldots, b\} \right\}, \tag{3.20}$$

with cardinality $K = \sum_{n=a}^{b} \binom{n+m-1}{m-1}$. Here, integers a and b act as a form of post-selection, where we only count detections with the total number of counts across all m PNR detectors falling in the range $\{a, \ldots, b\}$, with any detections outside this range discarded.

It is then useful to consider the set of Fock states occupancies which could give rise to a particular outcome in this set, given that we don't resolve polarisation. Given an outcome $\boldsymbol{n}' \in \mathcal{O}^{[a,b]}$, we denote these as the set of $2m$-mode occupancies

$$\mathcal{Q}(\boldsymbol{n}') = \{(n_{1H}, n_{1V}, \ldots, n_{mH}, n_{mV}) \mid n_{iH} + n_{iV} = n'_i,\ \forall\, i \in \{1, \ldots, m\}\}\,. \tag{3.21}$$

Now, given a pure input state $\left|\psi^{\text{in}}\right\rangle$, post-selection range (a, b), and scattering matrix $S(x)$ with corresponding unitary operator $\hat{U}(x)$, we calculate the probability of measuring the i^{th} outcome $\mathcal{O}_i^{[a,b]}$ as

$$F_i(x) = \sum_j \left|\left\langle \mathcal{Q}(\mathcal{O}_i^{[a,b]})_j \middle| \hat{U}(x) \middle| \psi^{\text{in}} \right\rangle\right|^2. \tag{3.22}$$

These probabilities are for the *ideal* set of outcomes, but in reality the detection properties generate a different set of K' non-ideal, noisy outcomes $\mathcal{O}'$ and corresponding probabilities $F'_i(x)$. We can model this by considering a detector with average dark counts per event μ and detection efficiency η, where μ is the dark count rate multiplied by the acquisition time.

Dark counts inflate the number of photons detected and can be modelled as a Poisson distribution

$$\text{Pois}_\mu(n') = \frac{\mu^{n'} e^{-\mu}}{n'!}, \tag{3.23}$$

giving the probability of an additional n' fictitious counts being detected in a given detection event. Detection efficiency losses, on the other hand, decrease the total number of photons detected, and can be modelled as a binomial distribution

$$\text{Binom}_{n,\eta}(n') = \binom{n}{n'} \eta^{n'} (1-\eta)^{n-n'}, \tag{3.24}$$

i.e. the probability of observing n' counts given that n photons were actually present.

To combine these to get the overall probability of observing n' counts given n photons, we convolve the distributions

$$P^{\text{noise}}(n' \mid n) = \sum_p \text{Binom}_{n,\eta}(n'-p)\,\text{Pois}_\mu(p). \tag{3.25}$$

We can then calculate the new non-ideal probabilities as

$$F'_i(x) = \sum_{j=1}^{K} \prod_{k=1}^{m} P^{\text{noise}}\left((\mathcal{O}'_i)_k \mid (\mathcal{O}_j)_k\right) F_j(x), \tag{3.26}$$

where the product runs over the mode components of each measurement outcome.

Note that these $F'_i(x)$ are still entirely deterministic, and don't capture the fact that we

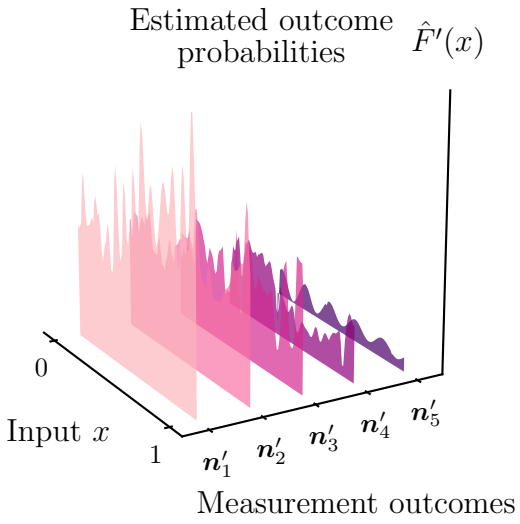


Figure 3.5: **QRC outcome probability estimates.** The estimate of the outcomes' probability mass function, $\hat{F}'_j(x)$, is generated by repeated sampling the device, Equation 3.28.

are sampling from the device. To model this sampling, we first define the probability mass function over the outcomes as

$$p(\boldsymbol{n}') = P(\boldsymbol{n}' = \mathcal{O}'_i) = F'_i(x), \tag{3.27}$$

and then sample N_{samp} times from this distribution to produce a set of observations $\mathcal{P} = \{\boldsymbol{n}'_i\}_{i=1}^{N_{\text{samp}}} \sim p(\boldsymbol{n}')$.

We can then use these samples to estimate the values of $F'_i(x)$ according to

$$\hat{F}'_i(x) = \frac{1}{N_{\text{samp}}} \sum_{j=1}^{N_{\text{samp}}} \mathbb{I}(\mathcal{P}_j = \mathcal{O}'_i), \tag{3.28}$$

for indicator function $\mathbb{I}$[6]. This gives a probability distribution, continuous in our input x and discrete in our measurement outcomes $\mathcal{O}'$, schematically illustrated in Figure 3.5.

This is the final step in simulating the output of the quantum reservoir computer, and we can now use these probabilities as the input to our reservoir training step to solve tasks. To recap, the estimates, and therefore the regression performance, depend on the input data x, the input state $|\psi^{\text{in}}\rangle$, the random sampling and encoding used to generate $S(x)$, the post-selection range (a, b), detector properties μ and η, and the number of samples N_{samp}.

[6]Again, here we use $\hat{F}'_i$ to indicate an estimator for F'_i, not an operator.

3.3.10 QUANTUM RESERVOIR COMPUTER TRAINING

With the estimator for the probability mass function over noisy outcomes $\mathcal{O}'_i(x)$ now defined, we can use it to learn some task with our reservoir computer. A supervised learning task consists of a dataset of r pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{r}$ related by some function $y_i = f(x_i)$ which we want to learn to estimate, i.e. we want to model $\hat{f} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ such that we minimise some objective, typically a mean squared error $\sum_i |\hat{f}(x_i) - y_i|^2$.

Our reservoir computer is trained on this task by first estimating the various outcomes' probabilities on each input in order to construct the $K' \times r$ design matrix

$$D = \left(\hat{F}'_i(x_1), \hat{F}'_i(x_2), \ldots, \hat{F}'_i(x_r)\right) = \begin{pmatrix} F'_1(x_1) & \cdots & F'_1(x_r) \\ \vdots & \ddots & \vdots \\ F'_{K'}(x_1) & \cdots & F'_{K'}(x_r) \end{pmatrix}. \tag{3.29}$$

We then learn weights using ridge regression,

$$W = f(x_j) \cdot D^T \cdot \left(D \cdot D^T + \lambda \boldsymbol{I}_{K'}\right)^{-1}, \tag{3.30}$$

where λ is a Tikhonov parameter. With these weights W, we now can write an estimate for f as

$$\hat{f}(x) = \sum_j W_{ij} \hat{F}'_j(x), \tag{3.31}$$

which is then used for inference. This process is illustrated in Figure 3.6.

3.3.11 FEATURE SPACE: SHAPE, SCALING AND METRICS

Now that we can use the reservoir to approximate functions, it is natural to ask what sort of functions can actually be learnt, and how this depends on the reservoir properties such as modes, photon number, network couplings, input states, and encodings.

The best overall metric of performance is how well the system can solve a particular task, which we measure using the mean squared error (MSE) between the output of the trained reservoir computer and the target function on the validation dataset, however this tells us little about how well the system will perform on a different task. To characterise the system in a task-independent way, there are a range of metrics we can use. An obvious starting point is to look a little closer at how the probabilities of a particular measurement outcome change as we vary our input parameters.

We know that the size of a set of outcomes $\{\mathcal{O}'_i\}$ scales as $K' = \sum_{n=0}^{n_{\max}} \binom{n+m-1}{m-1}$, combinatorial in modes m and maximum number of photons $n_{\max}$. Each outcome $\mathcal{O}'_i$ contributes a single probability $F'_i(x)$ which varies continuously and smoothly with the free parameters which encode our input data. These probabilities can be viewed as a set of (possibly overcomplete, non-orthornormal) basis vectors which generate a

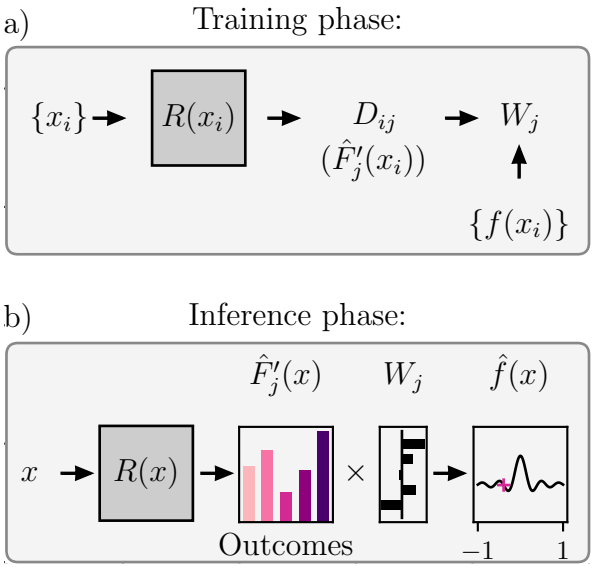


Figure 3.6: **QRC training overview.** a) Training is achieved by passing labelled data $\{x_i\}$ through the device, and storing the vectors $\hat{F}'_j(x)$ for each input in the design matrix D. The elements of this array are then fit to labels $\{f(x_i)\}$ to learn weights, W. b) During inference, the function $f(x)$ is approximated as $\hat{f}(x)$ on new input data by multiplying device output $\hat{F}'_j(x)$ by weights W, Equation 3.31.

space of functionals $\mathcal{L} = \text{span}\{F'_i(x)\}$, which we combine in our linear regression to approximate a target function. The lack of guarantee that these basis functions are orthogonal means that the dimension of the generated functional space $|\mathcal{L}|$ may be less than or equal to K'. This dimension determines the *expressivity* of the reservoir, i.e. the range of functions which can be approximated.

Even if we can generate a large space, we need to consider the degree of similarity between the generating functions. This is measured by cosine similarity, the normalised inner product between two functions. If a set of functions are very similar, then analytically they may span a large space, but in practice a small change to any one of them can change the space, and if they are estimated through sampling, the span of the estimates will be highly sensitive to the sampling used. This will reduce the quality of our reservoir computer's linear regression.

Even without considering sampling, using very similar functions as a basis for linear regression will tend to lead to very large learnt weights, giving an ill-conditioned learnt function $\hat{f}$, highly sensitive to small changes in the data. As an example, the vectors $(0, 1)$ and $(\varepsilon, 1)$ have cosine similarity $1/\sqrt{1+\varepsilon^2}$ and form a basis for $\mathbb{R}^2$ if and only if $\varepsilon \neq 0$. However, the coordinates of a general point within the spanned space, written in this basis, are proportional to $1/\varepsilon$ making them extremely sensitive to small ε. This is a situation we wish to avoid through tailoring our reservoir.

The characteristics of $\mathcal{L}$ and its generating set $\{F_i'(x)\}$ are therefore critical in determining the quality of approximation we can hope to achieve. Using the mathematical understanding of the LON scattering matrix and the Fock states we have built up, we can look a little deeper at the analytic form of the output probabilities over Fock states.

We have established that the scattering amplitude $\langle \boldsymbol{n}' | \hat{U}(x) | \boldsymbol{n} \rangle$ can be calculated as the permanent of the Scheel matrix $\Lambda_{\boldsymbol{n},\boldsymbol{n}'}$, in turn constructed from the mode scattering matrix $S(x)$ according to the photon occupancies of the input and output states. This means that the probability of detecting a particular output Fock state can be written as a sum of products of S-matrix elements, with the particular form depending on the input state used.

As we vary the input data x, the elements of $S(x)$ vary smoothly as the sum of products of sinusoids as determined in Section 3.3.7. The probabilities therefore also vary smoothly as the sum of products of sinusoids. In general, multiplying n sinusoids together gives a signal with bounded frequency comb spectrum, i.e. of the form $\sin^n(x) \approx \sum_{m=0}^{\lfloor (n-1)/2 \rfloor} c_m \cos((n-2m)x) + d_m \sin((n-2m)x)$.

The result is that each of the measurement outcome probabilities can be written as

$$F_i'(x) = \left| \sum_{\omega=-\omega_{\max}}^{\omega_{\max}} c_{i,\omega} e^{i\omega x} \right|^2, \tag{3.32}$$

which is a partial Fourier series of the input data, with some discrete number of total frequencies, N_ω. The number of frequencies, and the values of the frequencies, are determined by the encoding scheme and the number of input photons, while the coefficients c_ω are fixed by the network structure.

This idea is formalised in [GLA22; SSM21; VT20], and allows us to analyse the network's computation capabilities in terms of the frequencies it is capable of generating. Consider attempting to learn a function $f(x)$ using the quantum reservoir—a minimum requirement for perfectly approximating f is that the support of Fourier transform $\mathcal{F}[f(x)](k)$ matches the support of the reservoir's spectrum i.e. the reservoir must be capable of generating all frequencies present in the function.

The clearest path to generating a wider spectrum is by increasing the number of photons in the input state, which increases the size of the Scheel matrices, meaning higher harmonics of the base sinusoids are generated when we take the permanent. This implies that scaling the feature space in our system requires using states with large photon numbers. High photon number Fock states are an obvious option, but generating these can be challenging in practice. This motivates us to consider alternate types of high photon number, quantum states, which might be more practical to produce experimentally, but which still provide good performance in our reservoir computer.

A secondary point to note here is that while having a frequency spectrum which matches our target function is necessary for good approximation, it is not sufficient, as the reservoir must be able to generate the required frequencies across independent outputs. If the reservoir were to output a single signal which contained the correct frequency components, but with different Fourier coefficients, clearly we cannot approximate f as we cannot separate the individual frequency components to be recombined in the linear regression, i.e. the generated functional space is one-dimensional $|\mathcal{L}| = 1$. Having many frequencies present in $F'_i(x)$ doesn't guarantee these will all be present in our estimates $\hat{F}'_i(x)$—if we undersample some measurement outcomes due to them having lower average probabilities, their frequency contributions will be missed.

The number of frequency components is therefore a useful metric to consider when analysing the reservoir output, but doesn't tell the full story when considering the suitability of the reservoir for a particular task.

To better characterise the set of outputs, it would be helpful if we could develop a metric which takes into account the dimension of the space spanned, $\mathcal{L}$, along with the average magnitudes of the functions which generate this space.

Design matrix rank. In a simulation, we have access to the true probabilities $F'_i(x)$, allowing us to calculate the pair-wise inner-products and store them in a matrix

$$G_{ij} = \langle F'_i(x), F'_j(x) \rangle. \tag{3.33}$$

This is analogous to the Gram matrix DD^T which we invert during the reservoir training process Equation 3.30, except that D is composed of the *estimated* probabilities $\hat{F}'_i(x)$, evaluated at finitely many points $\{x_i\}$.

We can calculate the dimension of $\mathcal{L}$ directly as the rank of G. In the case of an experiment, or using the design matrix from a learning task, we can calculate the rank of DD^T, however it will almost always be equal to the upper bound $\min(r, K')$ (for r points and K' measurement outcomes). This is because the estimators $\hat{F}'_i(x)$ are generated from samples taken from distribution $F'_i(x)$. As this is a non-uniform distribution, some basis functions will not be accessible with a practical number of samples, thus reducing the output's effective rank. The outcomes which are undersampled will essentially be noise, and are not useful for learning. Even for outcomes which can be well resolved, the random variation introduced by sampling will artificially inflate the rank measurement. For two outputs with cosine similarity $\mathrm{CSE}(F'_i(x), F'_j(x)) = 1$, the cosine similarity of the corresponding estimators will in general be $\mathrm{CSE}(\hat{F}'_i(x), \hat{F}'_j(x)) < 1$.

The singular value (SV) decomposition is another way of viewing this, as the rank of a

matrix is the number of its non-zero singular values. Comparing the SVs of analytic G, and of DD^T built from samples, the larger SVs in both will be similar, however, any zero-valued SVs in G will be elevated to small, non-zero values in DD^T due to the sampling process artificially distinguishing between otherwise identical functions. We can define a new, conditioned rank R_c as the number of SVs in DD^T which are above some threshold c, which filters out contributions from noise and gives a more accurate measure of the dimension of the system, however this does require us to choose a threshold.

The unconditioned rank is not an ideal metric to use, as when we calculate it on the simulated $F_i'(x)$ it doesn't tell us anything about the functions we can expect to access after sampling, while calculating it on $\hat{F}_i'(x)$ is completely uninformative due to the noise contributions.

Spectral entropy. While the rank may not be especially useful, we can still calculate an *effective rank*, which takes into account the average probability of each outcome to indicate whether we can estimate that outcome's probability through sampling—we can only estimate something if we can measure it.

If the average probabilities vary greatly between different outcomes, then we will need to sample many more times to resolve the dependence on the input data of the low probability outcomes versus the higher probability outcomes. This reduces the efficiency of the system, as we either have a majority of our samples being uninformative, or we limit ourselves to only resolve a small subset of the available measurement outcomes, thereby reducing the functional space available for fitting. To avoid this scaling poorly, we want to choose our reservoir computer's input states, coupling network and encodings to try to guarantee that the average probability of each outcome is as high as possible, or as close to $1/K'$ as possible. This maximises the visibility of each outcome by attempting to make $F_i'(x)$ as uniform over outcomes as possible.

To account for this we define the *spectral entropy*, based on recent studies in complex networks [TAD24],

$$H = -\sum_i \sigma_i \log_2 \sigma_i, \tag{3.34}$$

where σ_i is the singular value spectrum of DD^T, normalised such that $\sum_i \sigma_i = 1$.

One interpretation of H is that we are treating the SVs as a form of probability mass function, and calculating the Shannon entropy. This is maximised when the distribution is uniform, which corresponds to having all SVs of equal magnitude, and therefore all outcomes having equal average probabilities. The spectral entropy provides a way of distilling these singular values into a metric which increases both with the number of independent basis functions, and the uniformity of their distribution.

H can be calculated for both $F_i'(x)$ and $\hat{F}_i'(x)$. In the former case, as spectral entropy increases, we expect the dimension of our functional space to increase, and that we will be able to efficiently estimate $F_i'(x)$ with fewer samples. In the latter case, it indicates the effective dimension of the functional space after sampling, robust to undersampled outcomes which don't contribute meaningful basis vectors to the reservoir training.

3.3.12 Fair comparison of states

A challenge in comparing the performance of different input states is to establish a fair equivalence between states of very different natures. Based on the Fourier analysis of the reservoir, the maximum frequency of the output functions is determined in part by the number of photons, hence it would be unreasonable to compare a single photon Fock state with a coherent state with larger average photon number on this metric.

An obvious solution is to only compare states with equal average photon number, where a Fock state with n photons should be compared to a coherent state with $|\alpha|^2 = n$. However, if we can generate an n photon Fock state, which is hard to do in the lab, then we would like to compare it to the best performing coherent state, which is easy to produce in the lab, under the same set of outcomes, as this is what we would choose to use in an experiment.

This means that for a particular Fock state input and set of outcomes, we should choose a coherent state's α such that the average probabilities over the same outcomes are maximised—the best possible visibility. This gives us the greatest chance of resolving the outputs under sampling, while only considering detection events with the same number of photon counts. The same argument can be made for hybrid states $|\boldsymbol{\alpha} + \boldsymbol{n}\rangle$, taking into account the added photons when calculating $\boldsymbol{\alpha}$.

Note, that in experiment it is not necessarily the case that larger α is better. Even though higher average photon numbers generate a wider range of frequencies and more outcomes, the limits of our PNR detection make detecting and differentiating these states harder, while the growth of the space means that either we sample for longer, use stricter post-selection to limit the number of measurement outcomes, or aggregate certain outcomes in order to build up sufficient statistics, at the expense of averaging some of the available output functions.

3.4 Quantum reservoir computer simulations

We now focus in on the particular system which we will simulate, with a range of different input states, encodings and random networks, and examine the performance first through the lens of the task independent metrics we have established, and then by applying the reservoir computer to solving a range of function approximation tasks.

3.4.1 Encoding schemes

Our encoding layer is as defined in Section 3.3.7. Using the general encoding in Equations 3.19, we can define three specific examples[7]:

Uniform linear. The simplest encoding is one in which all modes are given the same linear polarisation angle $\theta \in [0, 2\pi]$ which traverses two[8] equatorial orbits over the data domain, $\xi = 0, \gamma = 0, \nu = 2$. Illustrated in Figure 3.4.a.

Multislope linear. In the uniform linear scheme, the number of frequencies in the outputs $F_i'(x)$ depends only on the number of photons used. It is natural to try to broaden this spectrum by encoding multiple frequencies directly in each port. One method is to apply a different linear function to the polarisation of each spatial mode, $\xi = 0, \gamma = 0, \nu = 2m$. This causes each spatial mode to do a different number of azimuthal cycles in the polarisation space, illustrated in Figure 3.4.a.

Spiral. By including elliptical polarisations, we can make better use of the topology of the Poincaré sphere. A natural geometry is a spiral which traverses the sphere from pole to pole and returns to form a closed curve, $\xi = 1, \gamma = 0, \nu = 4m$. This encoding undergoes a number of azimuthal revolutions over the domain of the data while also providing low-frequency content from the zenith traversal. Illustrated in Figure 3.4.c.

In all encodings we consider, we choose the phase offsets ρ randomly for each mode, in order to improve phase diversity in the measurement outcome probabilities, and in turn fitting performance. This is important as otherwise there tend to be more degenerate $F_i'(x)$, reducing the reservoir computer's expressivity.

Note, that while most of our analysis will consider scalar x for simplicity, we can generalise to multivariate inputs through encoding schemes which distribute input data elements across the range of network parameters, for example with each waveplate angle controlled by a different component of multivariate x.

3.4.2 Random network

For the random network, we construct a 5-mode, polarised linear optical network in the triangular beamsplitter arrangement of Reck et al. [Rec+94], illustrated in Figure 3.7. Each block in the triangular network couples two spatial modes, each represented as Jones vector, with the full coupling represented as a 4×4 matrix as shown in Section 3.3.5. A block consists of a randomly oriented quarter-waveplate followed by a phase shifter in each spatial mode, followed by a non-polarising variable beamsplitter

[7] We find empirically that the interleaved and overlapping encodings provide similar performance, and so will only show results for encodings with $\gamma = 0$.

[8] This is to ensure that we complete a full cycle in the Jones vector components, ensuring our encoding is periodic over the domain $[-1, 1]$. One equatorial cycle produces the same polarisation state, but with a π global phase shift.

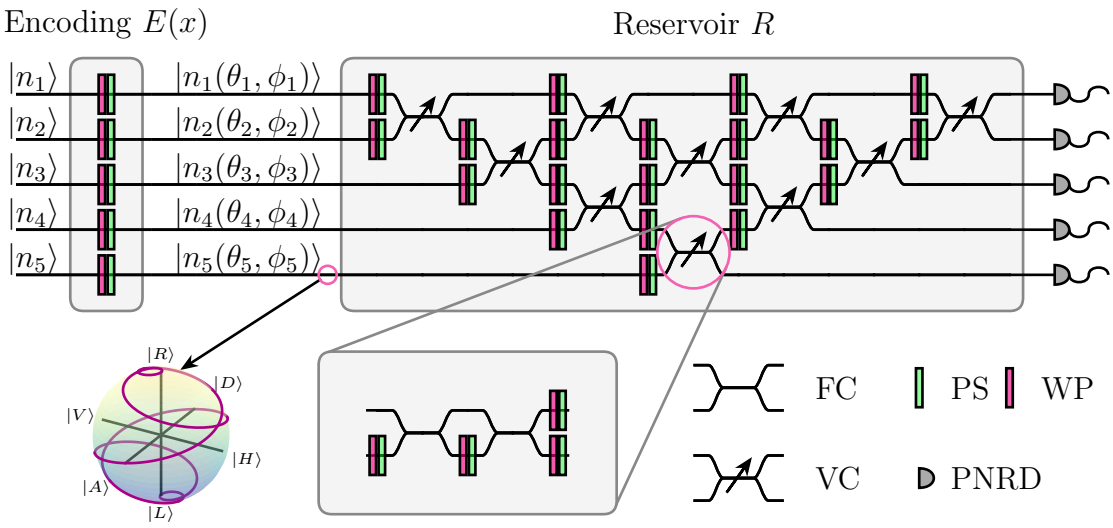


Figure 3.7: **QRC schematic overview.** Polarised quantum states in m different spatial modes are directed into block $E(x)$, which encodes the input data, x, in a multimode polarisation state. The polarisation at each port is parametrised on the Poincaré sphere by x through azimuth and zenith angles θ_i and ϕ_i for $i \in \{1, \ldots, m\}$. This state is fed into a random, fixed linear optical network built from variable couplings (VC) (i.e. beamsplitters), waveplates (WP) and phase shifters (PS). Output states are subject to polarisation-independent measurement by PNR detectors.

which couples the two spatial modes, finally followed by another phase shift in each spatial mode. The scattering matrix for a single block is built from components as

$$C_{ab} = P_a(\theta_{14})P_b(\theta_{24})BS_{ab}(\theta_3)P_a(\theta_{12})Q_a(\theta_{11})P_b(\theta_{22})Q_b(\theta_{21}), \tag{3.35}$$

where Q, P and BS are quarter-waveplate, phase shift, and lossless, non-polarising beamsplitter respectively, and the subscript on each optical element indicates the spatial mode to which it is applied. The parameters for each component are initialised randomly from a uniform distribution $\mathcal{U}(0, 2\pi)$. We then write the triangular network in terms of these random block as

$$R = [C_{12}][C_{23}][C_{12}C_{34}][C_{23}C_{45}][C_{12}C_{34}][C_{23}][C_{12}]. \tag{3.36}$$

The full scattering matrix is then $S = R \cdot E(x)$.

We do not consider lossy networks here for two reasons: on the experimental side, LONs can now be manufactured with tolerances such that losses are negligible for modestly sized networks (for instance integrated photonic circuits), and in simulation, the equivalent $2m$-mode unitary systems quickly become intractable. It would be valuable to investigate these lossy systems further, but we leave this for future work.

3.4.3 States

The number of modes in our states is determined by the LON, hence all the states we consider will be polarised with $m = 5$. We start with a 4-photon Fock state $|1, 1, 1, 1, 0\rangle$, chosen to provide a good balance between complexity of the generated output space, and simulation feasibility. With smaller m and n, the functional space generated by the reservoir is too simple to do meaningful regression, while the simulations start to become challenging as m and n grow beyond this, especially with polarisation doubling the number of effective modes, and hybrid states with comparable 'α's requiring ever more Fock basis elements to be simulated in order to maintain a good approximation to the true quantum state. Using fewer photons also limits the number of input spatial modes which can be excited, in turn limiting the encodings which can be tested.

In order to examine the impact of multiphoton interference on the reservoir's performance, the next state we consider is the distinguishable state $|1_0, 1_1, 1_2, 1_3, 0\rangle$. We can directly compare the output generated by this state with that produced by the equivalent indistinguishable Fock state, where the only difference is the presence of this interference.

Finally, we consider coherent $|\alpha, \alpha, 0, 0, 0\rangle$ and hybrid $|\alpha' + 1, \alpha' + 1, 0, 0, 0\rangle$ states, where the 'α's are chosen as 1.5 and 0.5 respectively, to maximise average total probability of the post-selected outcomes, as discussed in Section 3.3.12.

3.4.4 Simulation

In order to simulate the reservoir computer, we use the implementation of the SLOS algorithm in the Perceval Python library [Heu+23a], which takes a unitary scattering matrix and allows efficient calculation of individual elements of the corresponding Fock space unitary operator $\hat{U}$. The code for generating the input states, the parameterised reservoir unitaries, and for post-processing the outputs was written separately for this project.

In all the following simulations, we choose to set detector parameters $\mu = 0$ and $\eta = 0.9$. To justify neglecting dark counts, we make a conservative estimate of dark noise for a modern single photon detector as 100 cps. Assuming coincidence window of 10 ns, this yields an average of $\mu = 10^{-6}$ dark counts per detector per integration window. We cannot, however, discount the effects of detector quantum efficiency, with the current best number resolving detectors reaching efficiencies of order 90%. This can have a significant impact on the probabilities of the outcomes, especially as average photon number increases.

While the coherent states can be simulated efficiently using classical methods, the hybrid states derived from them are technically represented as infinite sums in the Fock

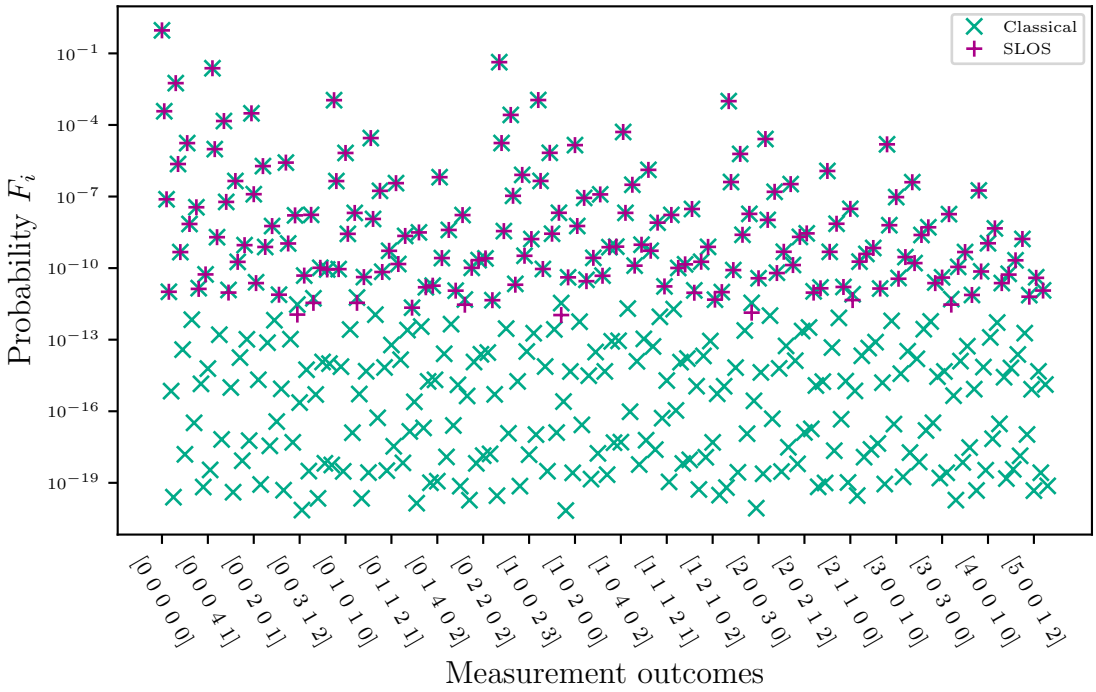


Figure 3.8: **Comparison of coherent state simulations.** We simulate a coherent state $|0.2, 0.2, 0, 0, 0\rangle$ through a 5-port random polarising network in two different ways. In 'SLOS', we first write the state in the Fock basis, truncated at maximum of 5 photons per spatial mode, and then perform the quantum simulation using the SLOS algorithm implemented in [Heu+23a]. In 'Classical', we directly multiply the random S-matrix on the field representation of the state, giving us an H and V coherent state amplitude for each spatial mode. We then write these coherent states in the Fock basis, calculate their tensor product, and sum over polarisations to get probabilities of PNR measurement outcomes which we compare with those calculated in 'SLOS'. We plot these probabilities over outcomes for both methods, and observe excellent agreement, with 'SLOS' providing fewer outputs due to the implementation only calculating amplitudes for outputs with probabilities greater than $\approx 10^{-12}$ for efficiency.

basis, which is not practical for numerical simulation. Therefore, in all simulations, for states with representations $\sum_{n=0}^{\infty} c_n |n\rangle$, we truncate this series as

$$c_n \to \begin{cases} c_n, & \text{if } n \leq 6 \text{ and } |c_n|^2 > \max_i(|c_i|^2)/100, \\ 0, & \text{else.} \end{cases}$$

In all cases, the maximum number of simulated photons in each input mode is 6, giving the most expensive amplitude calculation's parameters as $2m = 10$ modes, $n_{\max} = 12$ photons.

A nice sanity check which we can perform here is to simulate coherent states in our network using both the quantum and classical method, shown in Figure 3.8. In the first case, we write the state in the Fock basis as usual and simulate the output state. In the second, we apply $S(x)$ directly to the field representation $E^{\text{in}} = (\alpha_1, \ldots, \alpha_m)$

to produce $S(x) \cdot E^{\text{in}} = (\alpha_1^{\text{out}}, \ldots, \alpha_m^{\text{out}})$, which we then write in the Fock basis and compare coefficients with our quantum simulation. This allows us to verify that the simulations align, and that in the limit where we only measure average photon number in the quantum simulation, we get agreement with the classical intensity simulation.

As an indication of the cost of running these simulations, calculating the full output state generated by the hybrid state (which is the most expensive to simulate in the quantum picture, due to the large number of Fock basis components and the high photon numbers), across 500 data points $\{x_i\}$, takes approximately 22 minutes on an 28-core Intel i9-10940X CPU, with approximately 4 GB RAM usage per core.

3.5 RESULTS

We start by examining the typical outputs from the reservoir, independent of task. Figure 3.9 shows PNR outputs for our four states and three encodings. Each curve is the probability of measuring a particular PNR outcome as a function of x, with the diversity of the curves varying with the state and encoding. The spectral entropy H is given in the title of each subplot, along with the conditioned rank R_c, where c was chosen to filter out undersampled outcome probabilities, and is fixed[9] at $c = 2 \times 10^{-7}$.

As one might expect, the spiral encoding generates the most complex output, due to the additional harmonics introduced. Through visual inspection, it seems that the Fock state has the greatest diversity in its outputs. The spectral entropies H correlate fairly well with this visual hierarchy, however we see that coherent states perform especially well in this metric. Conditioned rank R_c tracks the visual diversity better, with Fock states performing well, and the both R_c and diversity increasing as we move left to right through encodings.

Worth noting is the scale in the coherent and hybrid state outputs, with many of these probabilities covering a wide range in the log-scale, and dropping below 10^{-4}. We omit the majority of the lower probability outputs here for clarity, however these figures still indicate that, while these states may have relatively large average probabilities over outcomes, the variation is larger. This in turn makes them more likely to be undersampled in particular regions of the input domain, range(x). The Fock and distinguishable states have consistently more measurement outcomes with higher average probability, making them more likely to be well resolved across the entire input domain. This doesn't, however, say anything about the diversity of these outcomes.

[9] This is chosen according to the number of samples, and the singular values of the data matrix. See Section 3.5.2 for details.

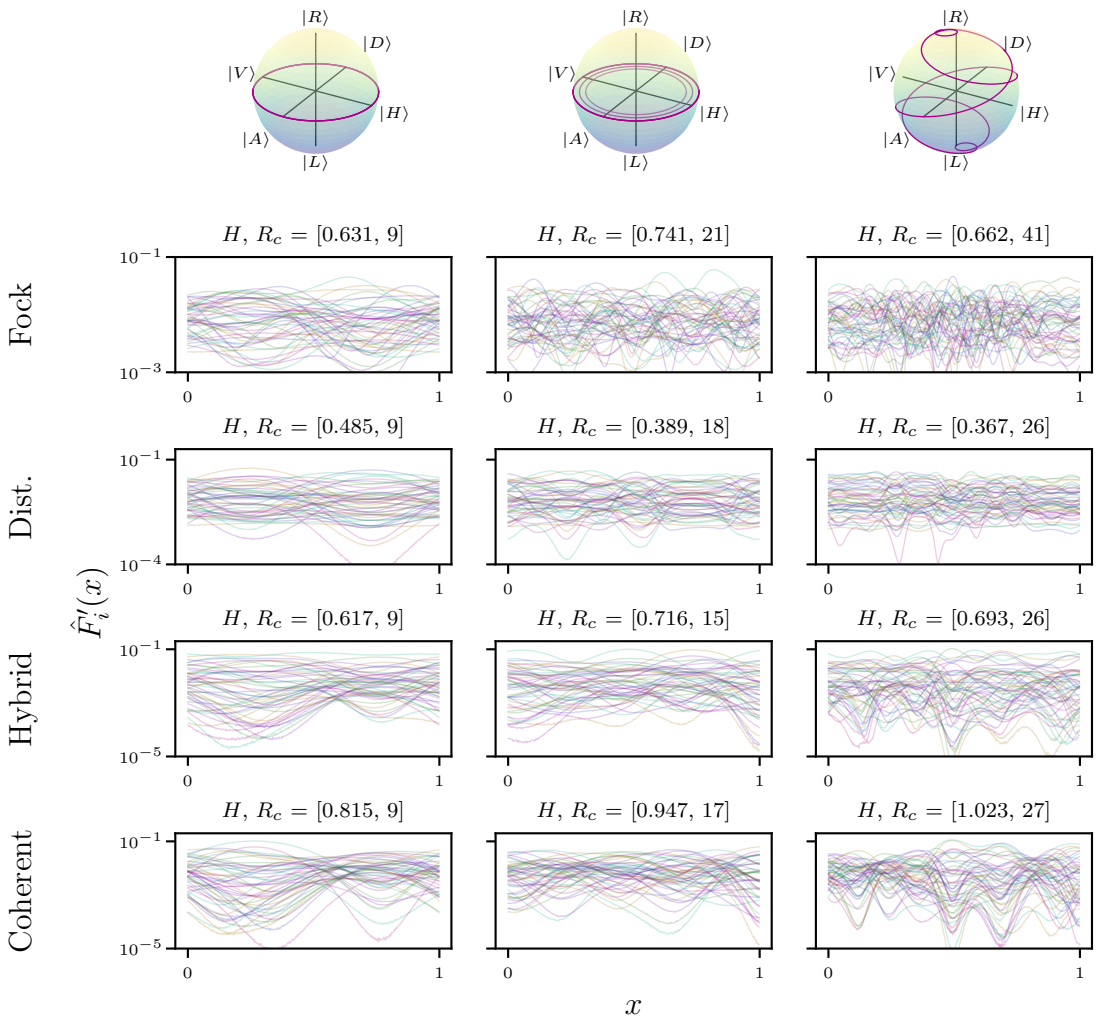


Figure 3.9: **Reservoir output examples.** Each inset displays the probabilities of the fifty most probable measurement outcomes for a given state-encoding pair, as a function of input x. The reservoir is a 5-port polarising device, and the states used are Fock $|1,1,1,1,0\rangle$, distinguishable $|1_0,1_1,1_2,1_3,0\rangle$, hybrid (photon-added coherent) $|0.5+1,0.5+1,0,0,0\rangle$, and coherent $|1.5,1.5,0,0,0\rangle$. Encodings ($m \in \{1,\ldots,5\}$ denotes the specific input port) are single linear ($\xi=0,\gamma=0,\nu_m=2$), multilinear ($\xi=0,\gamma=0,\nu_m=2m$) and overlapping spiral ($\xi=1,\gamma=0,\nu_m=4m$). In all cases, we perform PNR detection with detection parameters $\mu=0$, $\eta=0.9$ and $N_{\text{samp}}=10^7$, and post-select detection events of ≤ 4 photons to produce our set of outcomes. Spectral entropies H and conditioned ranks R_c, ($c=2\times10^{-7}$) are given in the title for each configuration.

3.5.1 Characterising the effect of number resolved measurement

For a fixed optical network, we can highlight the benefit gained through PNRD by comparing a coherent state measured under both PNRD and in intensity, where intensity measurement serves as the simplest classical measurement without further post-processing in the digital domain.

It is worth reiterating the motivation of performing PNRD in the first place—the

scaling of the space of measurement outcomes. In an extreme learning machine such as ours, a large output space is key to being able to approximate a wide range of functions, and this space's dimension will always be bounded by the number of linearly independent measurement outcomes we have. In the intensity case, this is equal to the number of output modes, in this particular case $m = 5$. Making polarisation sensitive measurements at most doubles this. In the PNRD case, the number of measurement outcomes scales combinatorially with the number of modes. This is true even for classical states such as coherent states—even though the intensity of each output mode completely determines the PNRD distribution for that mode, the individual probabilities of PNRD measurement outcomes vary nonlinearly with the input data and generate a larger set of linearly independent output functions than is available with intensity alone. While these probabilities may be calculated or estimated relatively efficiently from intensity measurements of classical states, this incurs an additional post-processing step on a digital computer to either sample from the PNRD distribution, or calculate the full outer product from Equation 3.2. As expected, non-classical states' PNRD distributions remain expensive to simulate, and cannot be recovered from intensity measurement alone.

The consequences of PNRD's scaling can be seen in Figure 3.10, where we plot the analytic outputs of the fifty most probable outcomes for a coherent state with PNRD, and the same coherent state with intensity detection, across our three encodings. Note, that in the intensity simulation, we directly plot intensities and use these in the reservoir training, treating them analogously to the PNR probabilities. For convenience, we normalise the intensities to 1, and write them also as $F_i(x)$, where in this case i runs over the output modes of the network. Despite different encodings causing an increase in the frequency content of the outputs, we can see in Figure 3.10.a that the rank of the classical system is limited to 5, while the quantum system, with the exact same optical network, reaches a rank of 111.

We also show the Fourier spectra generated by the network for each state-encoding pair in Figure 3.10.b. This matches the frequency combs predicted by Equation 3.32, while also demonstrating the improved scaling of N_ω in the PNRD case compared to the intensity measurement[10]. Comparing the non-classical Fock state to the classical coherent state under PNR, we see that while the coherent state generates more frequencies and has a higher rank, the Fock state frequencies have consistently greater magnitudes, making them more robust to sampling.

We see that the choice of encoding can have a large impact on the types of functions the reservoir computer can approximate, influencing both the rank of the spanned space (as measured through both effective ranks, H and R_c), and the frequency content of

[10]A reminder, the coherent state is still fully classical. PNRD allows us to measure a different set of properties (the Fock basis representation) compared to intensity measurement, which is more useful for generating a diverse basis for linear regression. See Equation 3.32.

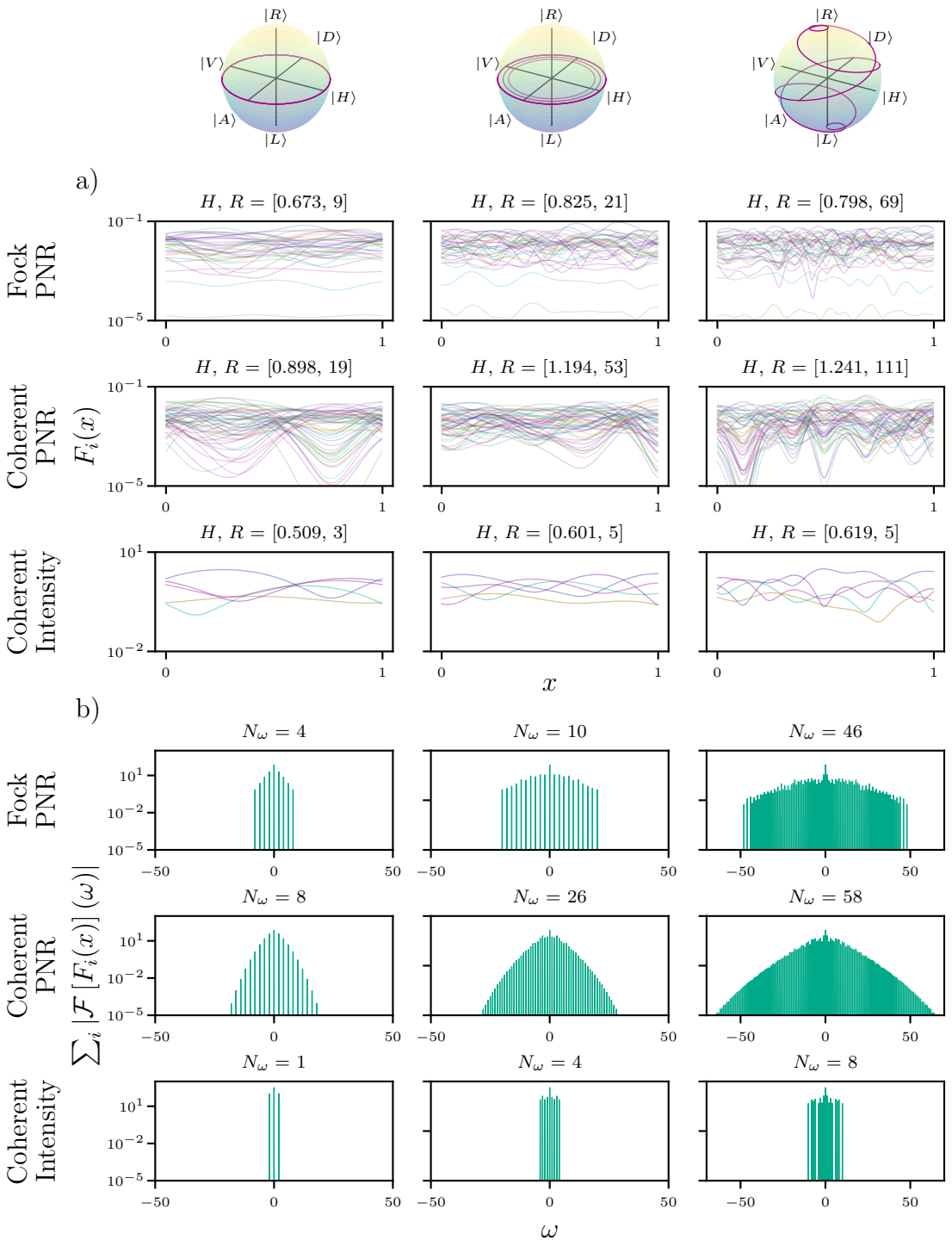


Figure 3.10: **Quantum-classical detection comparison.** Comparison of outputs and metrics for quantum and classical light. We compare reservoir outputs for a Fock state $|\boldsymbol{n}\rangle = |1,1,1,1,0\rangle$ and a coherent state $|\boldsymbol{\alpha}\rangle = |1.5, 1.5, 0, 0, 0\rangle$ under perfect PNR detection, and also the intensity measurement for the same coherent state. Both cases use the same 5-mode, polarised reservoir, and encodings (uniform linear, multilinear $\nu = 2m$, and spiral $\nu = 4m$). a) shows the analytic output functions of the fifty most probable output functions as a function of input data x for each case, while b) shows the corresponding frequency content generated by each state-encoding configuration. R and H refer to the rank (unconditioned, as these outputs aren't estimated from samples) and spectral entropy of the data matrix respectively, while N_ω denotes the number of unique frequencies present in the reservoir's output (excluding the DC component).

the outputs. This highlights that for function approximation tasks, it is important to choose an encoding scheme in a principled way which balances rank, spectral entropy, and frequency content, rather than simply maximising any one of these metrics.

Note, the data displayed in Figure 3.10 (and the spectral entropies) come from the analytic probabilities of measuring a particular outcome, i.e. without detector noise, quantum efficiency, post-selection or sampling. This allows us to look at the true frequency content. The spectral entropy H can be interpreted slightly differently here compared to H from sampled data. Here, H is used as a measure of effective rank, or dimension, of the spanned functional space $\mathcal{L}$, acting as a metric of reservoir complexity. This is also true in the case of sampled data, however, there H also provides an insight into the effect of the sampling. Too low an H indicates the outputs are undersampled, or that too harsh a post-selection filter was applied. Too narrow a post-selection reduces the number of outcomes and in turn the size of $\mathcal{L}$, while too few samples limits the resolution of the basis functions used to perform our linear fit to the task's data.

3.5.2 Metrics of performance for different states

Figure 3.11 shows the normalised singular values of DD^T for a range of encodings and states. Remembering that we ideally want as many of these values to be as large as possible, we see that Fock states consistently have the slowest drop-off across all encodings, while the distinguishable states have the fastest. This is correctly tracked by the spectral entropy H which is higher for Fock than distinguishable. However, we see that the largest spectral entropies consistently come from the coherent state, which is due to its SVs initially staying higher, despite a subsequent steep drop.

A clear feature in all spectra is the inflection at around 10^{-7}, which is due to the noise introduced by the finite number of samples, and whose level scales proportionally to the number of measurement outcomes K', and inversely with N_{samp}. This demonstrates why the standard rank of DD^T is unsuitable as a metric, unless we manually choose a condition number. When we do use a condition number, chosen here by inspection from the inflection point as 2×10^{-7} for all state-encoding pairs, we see that the resulting rank, R_c, is consistently higher for the Fock state than the other states, and lower for the distinguishable state.

On the whole, H and R_c correlate relatively well with one another, although there are some differences in the hierarchies they create between states and encodings. We expect R_c to better measure the true dimension of functional space $\mathcal{L}$ as it entirely ignores the effect of the undersampled measurement outcome probability estimators. H is a more general metric, also robust to the effects of undersampling, but with the added benefit of not requiring us to choose a condition number.

While H does not guarantee performance on a particular task, it does give us information

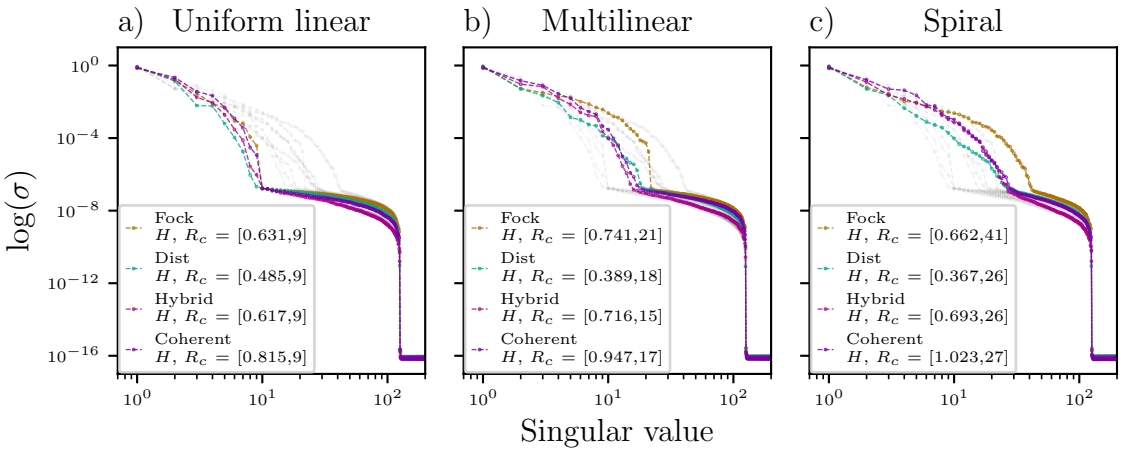


Figure 3.11: **Singular value spectra.** Normalised singular values of Gram matrix DD^T for a combination of states and encodings. All spectra are calculated with detector parameters $\mu = 0$, $\eta = .9$, $N_{\text{samp}} = 10^7$, with measurement outcomes post-selected for detection events of ≤ 4 photons. The states used are Fock: $|1,1,1,1,0\rangle$, distinguishable: $|1_0,1_1,1_2,1_3,0\rangle$, hybrid: $|0.5+1,0.5+1,0,0,0\rangle$ and coherent: $|1.5,1.5,0,0,0\rangle$. Encodings are a) uniform linear, b) multilinear, and c) spiral. Spectral entropies H and rank R_c with condition number 2×10^{-7} are given for each. Shadows of the other encodings are provided in each panel for easier comparison.

about the robustness of the system to sampling-based estimation of reservoir outputs $F_i'(x)$. R_c, on the other hand, gives us a precise measure of the dimension of the functional space spanned by the reservoir, once sampling has been performed. Larger R_c increases the chances that our reservoir will be able to approximate a specific target function.

Beyond these, metrics such as N_ω can be tuned through the choice of encoding, to match the frequency output of the reservoir with that of our target. However, typically in machine learning, we don't know the form of our target function. This makes it impractical to tune the reservoir's encoding for a specific frequency spectrum, meaning we would be better suited trying to maximise the size of the functional space spanned by the reservoir outputs, which is why we emphasise the importance of H and R_c in characterising a reservoir.

A key observation regarding Figure 3.11 is the low entropy of the output produced by the distinguishable photon state compared to the Fock and hybrid states. As the only discrepancy between this state and the Fock state is the presence of multiphoton interference, it indicates that quantum interference does play an important role in our system. Even our coherent state, which can be described classically, leads to higher spectral entropy compared to the distinguishable state when measured with photon number resolving detectors. Multiphoton interference still occurs in this case—we can see this in the way we simulate the Fock representation of the state passing through the

network—however, the action of a linear optical network on a coherent state ensures that the output state remains representable with classical fields. This is verified by matching the outputs of the same coherent state under both quantum and classical simulations.

Finally, we note that in our tests we observe that the results presented here are robust to different samplings of our random network parameters and encoding phase offsets. A more complete statistical analysis would be desirable, however with simulation time constraints we leave this for future work. Similarly, our choice of 5 polarised modes and a maximum of 12 photons was based on practical simulation times, and a larger scaling analysis with higher photon numbers and more modes would be valuable to understand the limits of our system. We expect that the ongoing efforts to demonstrate this system in experiment will be a more efficient way to explore these limits than pushing our compute capabilities for further simulations.

3.5.3 Function approximation tasks

While these metrics allow us to better understand the properties of the reservoir, the ultimate performance metric is how well the reservoir can learn a target function from training data.

To test this, we first define a set of target functions which we want to approximate. We start with a sinc and step function, defined on the range $[0, 1]$. The sinc function was chosen to provide a target which could feasibly be learnt by the network, with a bounded frequency spectrum, but also a nonlinear component which makes it harder than a simple sinusoid. The step function, on the other hand, tests the reservoir's ability to approximate highly nonlinear functions, where a perfect fit would require infinitely many frequency components. We sample these functions 500 times, and then randomly split these samples into two sets of 250 pairs, $\{(x_i, y_i)\}_{\text{train}}$ and $\{(x_i, y_i)\}_{\text{test}}$.

We demonstrate the trained QRC's performance on these two functions in Figure 3.12, where the training and inference was performed as in Section 3.3.10.

The Fock state input yields the lowest error for both functions. The hybrid state's performance is similar to Fock on both tasks, outperforming the coherent state despite its lower spectral entropy. The distinguishable photon state performs poorly on both tasks, while the classical measurement predictably fails to learn any meaningful approximation due to its extremely limited rank.

While not perfectly predictive, the correlation of spectral entropy to MSE across different target functions suggests H does measure robustness to realistic measurement conditions, with sources which generate higher H performing better after sampling.

The difference between the hybrid and coherent state is particularly interesting. Here,

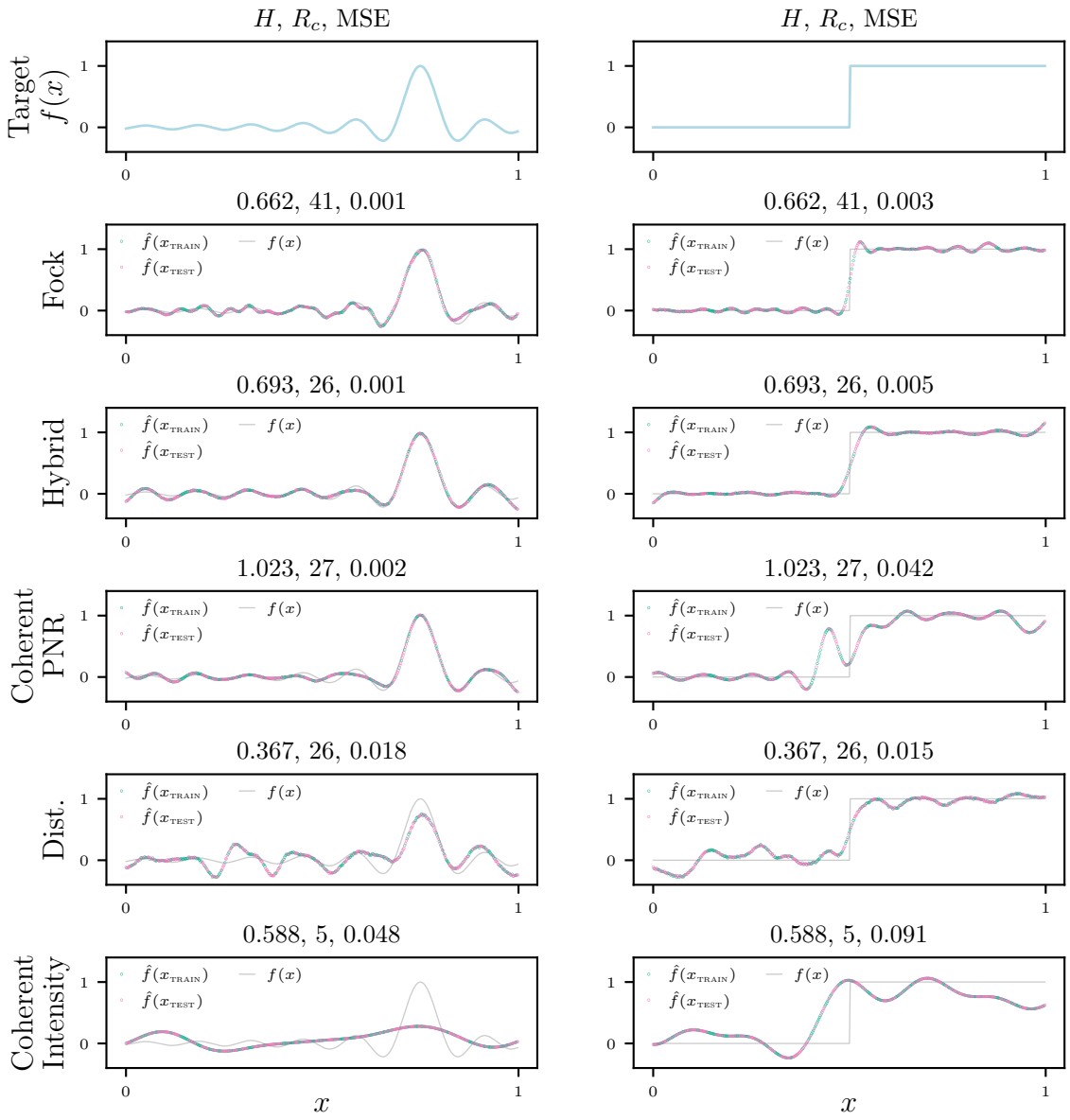


Figure 3.12: **Function approximation.** Compares the approximation $\hat{f}(x)$ to the target function $f(x)$. All trials use spiral encoding as defined in Section 3.4.1, and post-selection on ≤ 4 photon events with $\eta = .9$ and $N_{\text{samp}} = 10^7$. Performance metrics H, R_c ($c = 2 \times 10^{-7}$) and test-set MSE are listed in the title for each task-state combination. The states used are Fock: $|1,1,1,1,0\rangle$, hybrid: $|0.5+1, 0.5+1, 0, 0, 0\rangle$, coherent: $|1.5, 1.5, 0, 0, 0\rangle$, distinguishable: $|1_0, 1_1, 1_2, 1_3, 0\rangle$ and classical coherent: $|1.5, 1.5, 0, 0, 0\rangle$. All states use PNR except coherent intensity, which uses intensity measurement, with the intensity in each output mode used as the estimated measurement outcome $F_i'(x)$. Each trial uses a 50:50 test-train data split, arbitrarily chosen.

the spectral entropy and rank metrics fail to predict the performance difference. The addition of a single photon to each coherent state makes this a non-classical state. Silberberg et al. showed that higher order number states can be generated using photon-added coherent states, where the interference between a single photon and a classical laser source generates N00N states with arbitrary occupation numbers [AAS10; Win+10; Bar+10; LB02; ZVB04]. We hypothesise that we see similar effects in our network, where we generate high photon number N00N states with comparatively greater probability

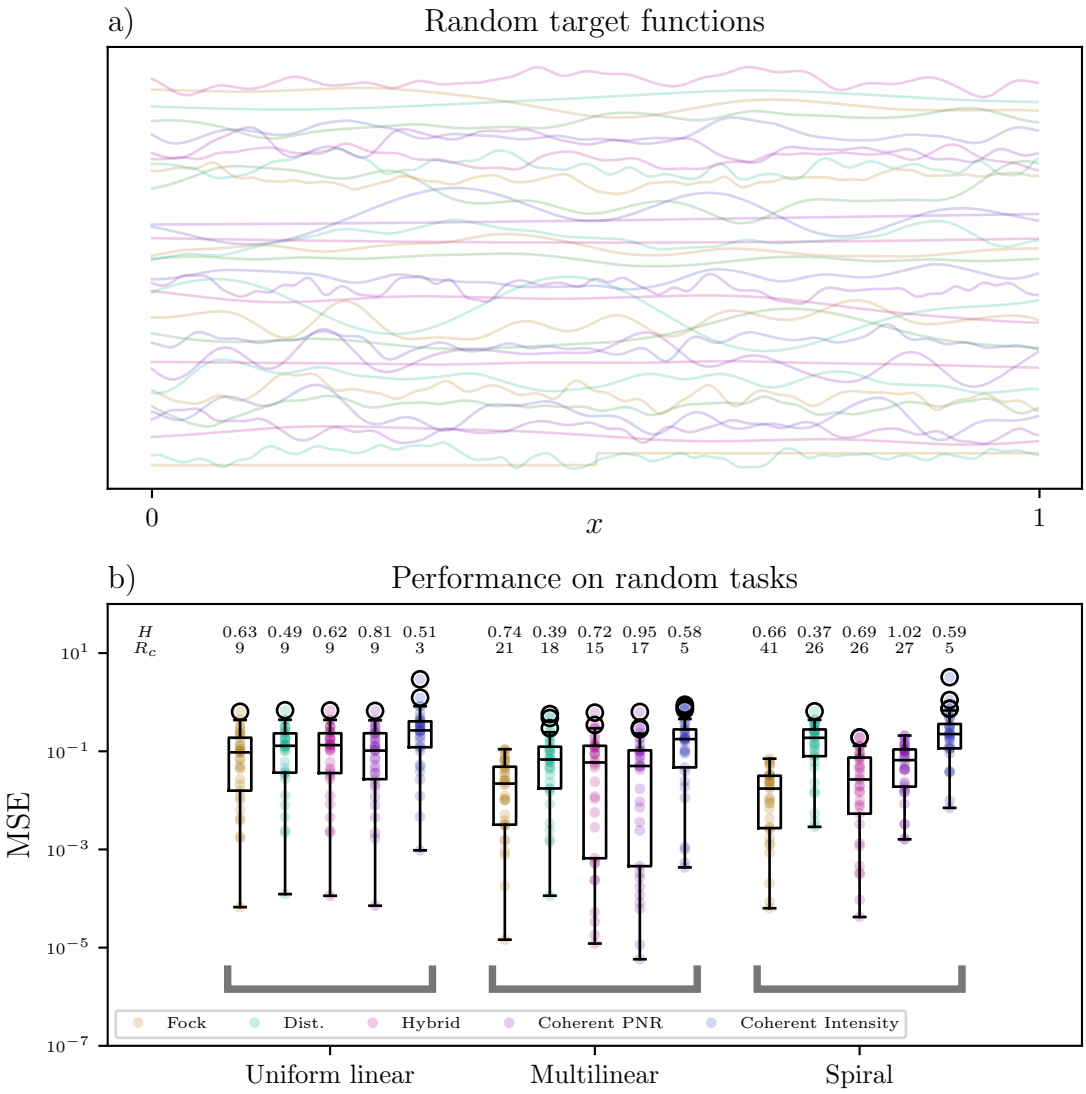


Figure 3.13: **Random function approximation statistics.** a) A set of 32 random functions, generated by randomly fixing a bandwidth, sampling amplitudes and phases uniformly across this bandwidth, and Fourier transforming. Note, they are vertically separated for clarity, but all have similar amplitude and DC offset in practice. b) Mean squared error of the reservoir computer's approximation on the random tasks. We show MSEs for all five states, and three encodings, with the box plot overlays showing standard statistics (median, first quartile Q1, Q1 +1.5 interquartile range IQR).

than in the Fock representation of the coherent state, and that these states scatter in our network similarly to Fock states, contributing to the hybrid state's good performance on the step task.

In order to better understand the performance of the reservoir across a range of tasks, we now generate a set of random target functions and examine the statistics of the approximation errors. Figure 3.13.a shows 32 randomly generated functions which have been passed through different low-pass filters, with passed bandwidth also chosen

randomly for each function. Figure 3.13.b shows the resulting mean squared error of the approximation on test data, for our established set of states and encodings. H and R_c are provided above each state-encoding pair's data. The box plots show the distribution of the MSEs across the 32 functions, with the median, first quartile on either side of the median, and first quartile plus 1.5 times the interquartile range shown.

Here, we again see a trend of improved performance as we move left to right across encodings, which is expected. The classical coherent state still performs worst out of the states tested, with distinguishable photons consistently second. Fock performs well in all cases, with the hybrid and PNR coherent states performing similarly, but with hybrid seeing noticeably better performance in the spiral encoding case.

In terms of analysing the reservoir in a task independent manner, the size of the frequency spectrum generated in the outputs is a clear, easy to interpret metric, and we can justify that matching reservoir and target function spectra is a necessary condition for good approximation. Conditioned rank provides our clearest estimate of the output functional space dimension, and tracks the performance on our tasks well. Spectral entropy, while well-founded in the desire to have as uniform a distribution over measurement outcomes as possible, is less clear in its interpretation. We can view it as an effective rank of the outputs space, but it isn't necessarily a good predictor end task performance. We have also seen that H can vary substantially for the same state with different reservoir samplings, and that this is likely due to differences in the random mode couplings changing the level of quantum interference present in the network, a multimode analogue of the 2×2 case which produces the HOM effect. More work is needed to properly understand the spectral entropy its uses in our system. More generally, developing better metrics for understanding the reservoir's properties and performance is an ongoing process.

A better predictor of end performance is the type of state used, and we show a fairly clear and consistent hierarchy over the states we consider, with the trend being that more classical states perform worse, and that multiphoton quantum interference plays a role in delivering good performance. The performance achieved using hybrid states is particularly encouraging from a practical standpoint—photon-added coherent states with high average photon numbers can easily be generated with current technologies, while high-photon number Fock states remain challenging. Our results thus far suggest that these states could be a practical alternative to Fock states in quantum reservoir computing, allowing us to efficiently access large functional spaces with 'cheap' quantum resources.

Beyond the increase in performance from 'more quantum' resources, we emphasise

that in all cases we see a significant increase in performance from photon number resolved detection: this alone is a quantum process and grants an improvement over the fully classical case due to the scaling of the output Hilbert space, even under the practical considerations of detector noise and sampling. We reiterate that in the case of classical states, this is not a 'quantum advantage' in the sense of outperforming classical systems or using quantum information, but rather a demonstration of the utility of quantum resources in a hybrid classical-quantum system. Classical states such as coherent states may be efficiently simulated on a classical computer, however PNRD provides easy access to a different representation (i.e. the probability of observing particular multiphoton coincidences) which has a greater diversity and faster scaling that intensity measurements, necessary properties for efficient reservoir computing.

3.6 Discussion

3.6.1 Initial experimental implementation

The work presented here is based in theory and validated in simulation, however the ultimate goal is to build a practical implementation of the QRC which can be used to solve interesting tasks. The first step towards that, implementing our system in experiment, is currently underway. We provide a brief overview of the experiment in progress, and discuss some of the challenges faced thus far.

The goal of the experiment is to demonstrate the improved performance on function approximation using PNR detection, therefore, in order to keep things simple, so far we focus on using coherent states as our input states. These are easy to generate, modulate, and have been shown to perform well in our simulations on simple tasks.

For our reservoir, we choose to use a MMF, as they are widely available, by definition can have many spatial modes, which are randomly coupled as light propagates through the fibre, and can have birefringence introduced through stress, making them suitable for use with polarisation encodings.

In order to encode information in the input state, we use a phase-only LCoS SLM, which allows us to generate spatially varying, phase modulated states in a very similar way to the rotating waveplate scheme discussed earlier. The controllable nature of the SLM lets us easily program the encoding via computer, while the high-resolution phase control also lets us optimise coupling of the modulated states into the fibre. While LCoS SLMs are not the fastest phase modulators in the world, the technology is ever improving, and the SLM is not the bottleneck in our reservoir's data throughput—rather it is limited by the integration time of our detectors and the number of samples needed for a given input data-point to suitably estimate the outcome probabilities.

Choosing a detector is a harder process. There are a wide range of options capable of

detecting single photons to select from. These tend to operate through one of two different mechanisms: statistical analysis of multiplexed single photon detections [You+20; Che+23b; Fit+03; Kri+24; Ach+04], and low noise detectors which are able to resolve signals which scale discretely with the number of photons [Los+24; Ham23; Sch+18]. The former is more common, while the latter is more difficult to implement, but is more useful for our purposes as the efficiency of these two methods scale differently [Fit+03].

This difference comes from the fact that even with multiplexed single-photon sensitive detectors with unit quantum efficiency, the performance is capped by the likelihood of two photons being absorbed by the same detector and classed as a single click. In a detector where the signal scales discretely with the number of photons, this is not an issue, meaning that a device with perfect quantum efficiency could perfectly identify the incident state. We therefore settle on using a Hamamatsu qCMOS camera [Ham23], which allows us to resolve individual photons above the noise floor in the analogue voltage output of each pixel.

It is worth noting that our detection scheme doesn't necessarily need to perfectly resolve photon numbers, only that the measurements we make are consistent. If we lose photons due to network losses or low detector quantum efficiency, or if we incorrectly classify a state due to, for instance, performing PNRD with multiplexed single-photon detectors, it doesn't change the operation of the network, it just changes the mapping between the true quantum state and the set of measurement outcomes. This may affect the overall performance on a given task, but it means our detection process is not 'all or nothing'—we can still expect to get improvements over intensity measurement even with non-ideal PNRD for both quantum and classical states[11].

The initial experiment is schematically illustrated in Figure 3.14.a, while Figure 3.14.b shows the physical experiment in the lab. This is currently early in its development, however, initial data does show features similar to those in our simulations. Currently, the read-out rate of the camera is the limiting factor, restricting the number of samples we can take and therefore the signal-to-noise ratio we can achieve in our estimates of the outcome probabilities.

The camera takes one exposure every 2 ms, however, we are limited to approximately 40 frames per second due to inefficiencies in the way data is transferred to the control computer and written to disk. This is a bottleneck which can be overcome, where faster frame rates will allow us to gather more samples, better resolving the output functions generated by the reservoir.

[11] As usual, this assumes no further digital post-processing, i.e. for the same physical system and computational resources.

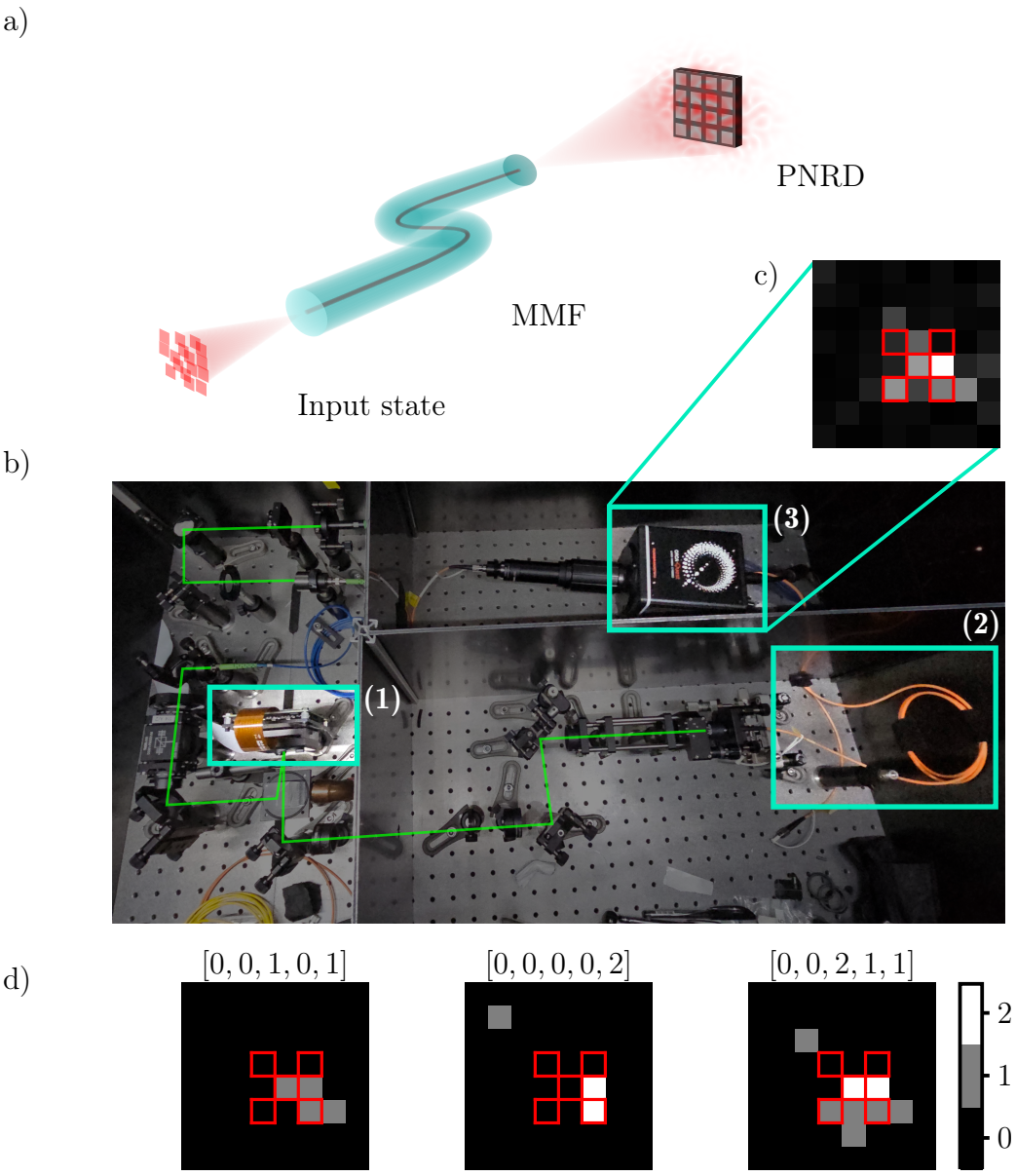


Figure 3.14: **Proposed experiment in MMF.** a) Example reservoir configuration, with spatially varying, phase modulated states injected into a multimode fibre, where path interference occurs via the MMF coupling modes, before detection of the output state with PNRD. b) Initial experiment setup, with a SLM (1) used to generate the input states, MMF acting as the reservoir (2), and a qCMOS camera (3) used for detection. Beam path in green. c) Example time integrated speckle output from the qCMOS in an 8×8 region of interest, with 5 pixels selected to act as detection modes. Intensity in arbitrary units. d) Example number resolved output frames, where each pixel gives the number of detected photons. Title indicates the decoded multiphoton coincidence event from the 5 pixels, reading left to right, top to bottom.

3.6.2 Future prospects

In terms of optimising the reservoir scheme, there are four main areas of investigation—reservoir samplings, encodings, states, and detection.

While the MMF and qCMOS combination is a relatively practical and fast way to realise

and test our reservoir design, it is likely that in the long term, this scheme would best be implemented using integrated devices such as programmable photonic waveguides. These devices can realise mesh coupling geometries similar to that in Figure 3.7, with very low losses, full reconfigurable control over the scattering matrix S, and good stability. Implementations of these systems already exist for optical quantum computing (i.e. [Mar+24]), however, to our knowledge there are currently no systems using number resolved detection. In the short term, one of the key advantages of our system is that many of the selling points of integrated photonics *don't* apply here—specifically, we can implement it in the lab with low-cost optical components, and still achieve many modes, necessary for generating the complexity required for machine learning tasks. Known issues such as losses and temporal instability in MMFs can be mitigated (i.e. through temperature control, vibration isolation, and high-quality components), and in many cases may be a worthwhile trade-off for relative ease and cost of implementation.

We examine several encoding schemes in detail in this work, which, based on the Fourier analysis of the network and with ease of practical implementation in mind, we believe were well-informed and remain good candidates for continued research. However, there are many other possible ways to encode information in phase and polarisation, across many spatial modes, and it is likely that further optimisation can be made in this area. We have already demonstrated that the encoding must be tuned according to the task at hand, and we expect that moving towards higher dimensional tasks, with multivariate inputs, will require more complex encoding schemes. This is a further advantage of using an integrated waveguide system as the reservoir, as these devices typically have many degrees of freedom which can be precisely controlled with digital logic. Efficient integration with existing digital computers is essential for practical use.

In terms of states, we have seen in simulation that high photon number Fock states provide best-in-class performance within our system, but there are many other types of non-classical states which may be more practical to generate using fewer quantum resources. We hope to be able to experimentally investigate more complex states once the coherent state experiment shown above is running effectively, with the photon-added coherent states a natural starting point based on their performance in our simulations. These are more difficult states to generate than coherent states, requiring more complex optical setups, however they are well within the capabilities of modern quantum optics labs. We envisage that this would be done using an spontaneous parametric down-conversion (SPDC) process to generate a photon pair, where one photon is combined with a coherent state of the same wavelength and the other photon is used to herald the presence of the photon-added coherent state [ZVB04]. Beyond these, we expect that improvements in quantum sources will open up even more options for us to investigate. For instance, quantum dots provide a promising alternative to parametric processes for generating Fock states, even though currently they struggle to scale to higher number-states.

When it comes to detection, things are less obvious. It is clear that PNRD is key to the performance of our system, reducing required digital computation even with coherent states, which are otherwise entirely classical. It is also clear that as the number of photons we wish to count increases, the quantum efficiency of our detector becomes increasingly important in order to consistently measure the correct outcome. Multiplexed click detectors do not scale well for these applications, as explained in Section 3.6.1. While detector such as superconducting nanowire single-photon detectors (SNSPDs) provide excellent quantum efficiency, they are expensive and difficult to couple to, making them impractical for our purposes. Currently, we believe that devices such as the qCMOS camera are the best option for this type of experiment, due to their ability to distinguish individual photons above the noise floor. As the quantum efficiencies and frame rates of these detectors are ever improving [Cec+21; Hum+23; SGZ21; Ham23], we anticipate that we will be able to push our reservoir to higher photon numbers, generating larger functional spaces with which to learn more complex tasks.

One final point to keep in mind, is that as we increase the average number of photons in our system, the overall space of number resolved measurement outcomes grows combinatorially. Indeed, this is one of the main advantages of our system. However, it does mean that the number of samples needed to resolve these additional outcomes also grows, which is not practical with detectors with limited frame rates. This will necessitate either capping the overall number of photons we use, or developing new post-selection schemes which preserve the reservoir's complexity, but reduce the number of samples needed to estimate the outcome probabilities.

Post-selection aside, we have to be cautious with arbitrarily increasing the average photon number, as we have the potential to introduce a disruptive degeneracy into our output space if we have non-unit detection efficiencies. This is because higher-number Fock states which are incorrectly measured due to losses will be increasingly likely to collapse into the same, low-number measurement states. This then averages out the independent features present in each of the original low-photon number measurement outcomes, reducing the information across the measured states, even with the exact same detection properties. This remains an issue unless we can achieve extremely high detection efficiency, which likely requires further development of devices like the qCMOS.

3.6.3 Conclusion

The main message we believe we have conveyed in this chapter is that photon number resolving detection is a powerful and increasingly accessible quantum resource which can open up new avenues for performing scalable computation using relatively simple optical systems.

Beyond the use of PNRD, we have also shown evidence of differences in reservoir expressivity and resulting computational performance across different quantum states and encodings. Our analysis of a range of input states shows that we can achieve good performance using classical or 'cheap' quantum states such as photon-added coherent states, avoiding the difficult task of preparing high-photon number Fock states. Our reservoir architecture, based on linear optical networks, may be implemented with a range of simple, widely available optical systems, such as multimode fibres. Using a reservoir computer, specifically an extreme learning machine, allows us to use the complex dynamics generated by the system with simplified training and reduced parameter tuning compared to other, more general, programmable quantum machine learning approaches.

Distilling the network properties further would allow us to make better predictions about the performance of a given network, and potentially allow for more intelligent design, for instance in choosing encodings, or identifying examples of multimode optical systems with scattering properties which are naturally suited to being used in a QRC. We have seen that the frequency content, spectral entropy and conditioned rank of the reservoir outputs provide some insights, and are necessary but not sufficient for good performance. Further analysis will likely require the use of new metrics which better correlate the reservoir's properties with the performance on a given task. There is also much left to investigate when it comes to the connection between the sampling of the random reservoir and the overall system performance. Initial experiments indicate that the way we sample the couplings in our random LON has a large effect on the performance of the system, and this in turn implies that not all multimode scattering systems are created equally when it comes to building a QRC.

In our evaluation of the system, we tested performance on a set of function approximation tasks, however we have also seen good performance in early work on classification tasks and multivariate inputs. We note that while our analysis focusses on classical tasks, using a quantum substrate and input, the QRC may also be well suited to quantum tasks such as state tomography, state generation, or quantum communication—again, all areas for further investigation.

Finally, while advancements in quantum technologies are likely to mitigate some of the current challenges in the field—for instance, generating complex quantum states, improving detector performance, and efficiently training nonlinear quantum machine learning systems—quantum computing remains a formidable challenge, despite decades of development. Nevertheless, we believe that the approach presented here, though not universal, offers a practical and accessible means of approaching certain quantum machine learning tasks using current technologies. Scalability is always a challenge in physical computing systems. Our hope is that the insights gained from this work will be valuable in the continued development of practical quantum computing systems.

3.7 NOTATION

For reference, we summarise some of the main notation specific to this chapter[12].

- m The number of spatial modes in a linear optical network. In the case of a polarised network, each spatial mode is formed of two orthogonal polarisation modes.
- n The total number of photons injected into the network.
- x The input to the reservoir computer, potentially multivariate.
- $S(x)$ The mode scattering matrix of the network, which maps input modes to output modes, where some properties of the network are varied as a function of the reservoir input.
- $U(x)$ The unitary operator over Fock space generated by network topology S.
- $F_i(x)$ The true probability of observing the i^{th} measurement outcome given the jth input is encoded in the reservoir, assuming perfect detectors.
- $\mathcal{O}'$ The set of all possible measurement outcomes, given imperfect detection and some post-selection rule.
- K' The number of outcomes in set $\mathcal{O}'$.
- $F'_i(x)$ The true probability of observing the i^{th} measurement outcome given the jth input is encoded in the reservoir.
- cL The space of functionals spanned by the set of $\{F'_i(x)\}$.
- $\hat{F}'_i(x)$ The estimate of the probability of observing the i^{th} measurement outcome given the jth input is encoded in the reservoir.
- D The design matrix, where D_{ij} is the estimated probability $\hat{F}'_i(x_j)$ of observing the i^{th} measurement outcome given the jth input is encoded in the reservoir.
- σ The singular values of matrix DD^T.
- W The learnt weights which map our reservoir outputs to the target function estimate.

[12]Note, the notation used in this chapter differs slightly from that in the paper [Ner+24].

- R_c Conditioned rank, giving an estimate of the effective dimension of $\mathcal{L}$ after sampling.
- H The spectral entropy of the reservoir, calculated from the normalised singular values of DD^T.
- N_ω The total number of unique, discrete frequency components in the output spectrum of the various $F_i'(x)$.
- $\Lambda_{\boldsymbol{n},\boldsymbol{n}'}$ The Scheel matrix formed from elements of S and used to calculate the scattering amplitude $\langle \boldsymbol{n}' | \hat{U} | \boldsymbol{n} \rangle$.
- $f(x)$ The target function being approximated.
- $\hat{f}(x)$ The learnt estimate of the target function.

Chapter 4

Reinforcement learnt optimisation

We have established that performing gradient-based optimisation directly on a physical system is difficult without a good, differentiable model of the system, and that good models are hard to build. This severely limits the performance of available optimisation strategies for such systems. Here, we propose a new method for learning an approximate model for propagating errors backward through the system, similar to gradient methods. This allows for simpler learnt models, specialised to the hardware, providing robustness to imperfections not captured in an idealised model. By optimising not for accurate gradients but for the ultimate goal, performance on the end task, the method allows improved convergence over vanilla gradient-descent on certain tasks. This work is partly detailed in the paper [NF24].

4.1 Introduction

This chapter shifts focus from the architectural design of neuromorphic systems, to exploring optimisation strategies capable of programming (i.e. training) these systems at the task-specific level. In the examples of neuromorphic systems we've seen so far, the training of the system's free parameters have been intentionally simple—in most cases linear regression or some limited evolutionary optimisation. This is largely due to the fundamental challenge in training physical systems—the differentiability problem.

As discussed in Section 2.3, the non-differentiability of physical systems prohibits the use of gradient-based optimisation algorithms to train the system parameters directly. This is a huge barrier to developing practical physical neural networks (PNNs), as gradient-based optimisation is the workhorse of modern machine learning and has thus far been the key to large-scale learning systems.

Currently, in the neuromorphic and physical computing community there are two main approaches to bypass this roadblock, delineated by whether they attempt to use gradients regardless (despite the non-differentiability), or instead use alternative

information to guide the evolution of our tunable parameters [Bru19; McM23; FDB24; Mom+24].

To access gradient-based training, we have two options. The first is to model and simulate the physical system of interest, use this as a proxy for training, and then evaluate and perform inference on the real system. Such digital twins provide the efficiency of gradient-based optimisation in high-dimensional systems, however the simulations can be costly or even intractable, and misalignment between the simulation and the real system degrades performance. Often, using a simulation undermines the key benefits of using a physical computing system such as speed, computational capacity and energy efficiency. The second approach to using gradients is to carefully design a physical system where we can either measure all the internal states necessary to manually calculate the analytic derivative of the system's output with respect to the parameters, or to design a system where the physical dynamics naturally manifest these derivative themselves. The former is a brute force approach, and can quickly become expensive in terms of measurements. The latter is a more elegant solution, but is difficult to achieve as it relies on clever exploitation of the physical system's dynamics. In both cases, the types of systems which admit these schemes are limited, restricting the generality of the approaches.

On the other hand, if we dispense with gradients entirely we are left with zeroth-order methods, which use alternative heuristics gathered purely from forward passes through our physical system to iteratively update parameters, such as sparse finite differences or evolutionary strategies. These methods are typically sample inefficient, requiring many evaluations of the system to converge, and are not scalable to high-dimensional systems. This is largely because they don't have any model of the system to guide their search, which is especially inefficient in cases where we do actually have an approximate or low-dimensional model of the system.

Overall, model-based methods such as gradient descent tend to underutilise the physical system, suffer from poor alignment with the physical system, and require many simulation evaluations, although they can achieve reasonable performance with relatively little effort. Zeroth-order methods can provide excellent alignment due to using the physical system directly, but are sample inefficient, in part due to underutilisation of priors we have on the system dynamics.

We would like to design a new optimisation strategy with the best of both worlds. The approach we take here will rely on training a separate, conventional machine learning model which is able to approximate the backpropagation process through the otherwise non-differentiable physical system, where we make use of reinforcement learning (RL) to train this *learnt optimiser* model. In order to understand the reasoning for this approach, we will start from first principles and consider the way we might design a

novel optimisation algorithm which effectively balances the benefits of both model-based and zeroth order optimisation techniques.

To begin, we can quickly dismiss any approaches which resemble random search of the parameter space, as these are incompatible with our scalability requirement. Instead, we need an iterative algorithm which is capable of improving the objective over time by conditioning its decisions on prior experience. We therefore need to generate some form of privileged direction in parameter space to move in, given a starting set of parameters. Evolutionary methods do this by maintaining a population of parameter vectors and combining the best performing individuals to create a new population. This is essentially a combination of a sparse finite difference method and multiple parallel optimisations, and therefore still suffers from the curse of dimensionality. Gradient-based methods typically use the first-order linearisation of the system to get this direction, moving from the best previously-found parameter vector in the direction which locally minimises the objective. While we may not have a model for a physical system's derivative, we know that such a linearisation does exist, and in the extreme, sample-inefficient case, this could be calculated by finite differences. We have also seen (Section 2.3.2) that there are a number of approximate gradients which can be used, i.e. reasonable alignment with the true gradient can be sufficient for getting good performance in training.

Physical systems typically have symmetries or analytic approximations which, while not capturing the full dynamics, could be used to inform an optimisation process. Ideally, this would take the form of a low-dimensional model which is simple and cheap to evaluate. This could then be used to provide a preferred direction to move in parameter space, conditioned on information from the evaluations performed on the physical system in order to remain robust to misalignment.

Consider passing some input x through a physical system $f_\theta(x)$ with a particular set of parameters θ, and measuring an output y. We have an error on that output Δy, i.e. a direction we want the output to move in, based on some change to the parameters $\Delta\theta$. All approaches discussed above take the form of some model $m(\Delta\theta \mid \Delta y)$, giving this update to the parameters in terms of the desired change of the output. In gradient descent, m uses the linearisation of the system $\Delta\theta \propto \Delta y \cdot \frac{\partial f_\theta(x)}{\partial \theta}\Big|_x$, which requires the analytic model of f along with the input and parameters in order to evaluate the derivative at the particular point of interest.

A potentially naïve approach might be to ask whether it's possible to parameterise this m as a regular neural network and learn it from the physical system. This is partly inspired by recent successes in the field of meta-learning ([And+16; LM17; Bel+17; KS20; Met+22; Che+23a]), where instead of training a model to solve some task directly, we train it to be able to *independently learn* to solve a range of tasks.

Learning m can be seen as somewhat similar to physics aware training (PAT) [Wri+22],

in that we learn a black-box model relating to the physical system. The difference here is that we are not aiming to learn the full dynamics of the system, only a model which gives us some preferred direction in parameter space which is a good heuristic for conditioning the parameter updates when training a PNN on a set of tasks. Note that m should ideally be independent of the task we are training the PNN on, as it provides a direction in parameter space based on the PNN's properties, not the properties of the task. For instance, in the example above, the task-dependent information which influences the parameter updates is entirely encoded in Δy. Enforcing this condition would allow us to generate a custom optimisation algorithm tailored to the specific PNN architecture, which would be reusable on a range of different tasks.

We also note that, assuming it is possible to learn m, which predicts updates to our tunable parameters, it may be possible to learn a second model which is able to predict 'updates' to the PNN's input, echoing the form of the full backpropagation algorithm $\Delta x \propto \Delta y \cdot \frac{\partial f_\theta(x)}{\partial x}\Big|_x$. This would allow us to embed a PNN in a larger deep network, with tunable parameters before and after the PNN, and then train all the parameters together, using backpropagation for components which we have analytic forms, and our learnt models for the physical components.

In order to train this model using our deep learning toolbox, we need to provide it with inputs and target outputs and ask it to learn the mapping between them. The first challenge is that we don't know the target outputs, as these are the 'pseudo-gradients' we are interested in—if we could calculate these directly, our work would be done. What we do have is a performance metric, such as the loss on a particular task, which we want our learnt model to minimise.

This approach has certain similarities with nonlinear control problems, where we have a plant with some control inputs (our PNN and parameters), and a controller (our model m) with its own parameters which we want to tune to be able to predict the control inputs at each time step.

One technique which has recently seen significant success in solving such control problems is reinforcement learning (RL), specifically deep reinforcement learning (DRL). DRL provides a method for solving complex decision-making problems by training a neural network to predict optimal actions, based on a potentially non-differentiable objective function. This is achieved by interacting with the system, observing the results of actions taken, and updating the policy for choosing actions based on these observations and the change in the objective.

We could therefore imagine using RL to train the parameters of m such that we are left with a model which is tailored to the physical system, and provides us with estimates for parameter updates which can be used to train the system on specific tasks. RL has already been used for training physical neural networks [Bue+18; Tan+22], however to

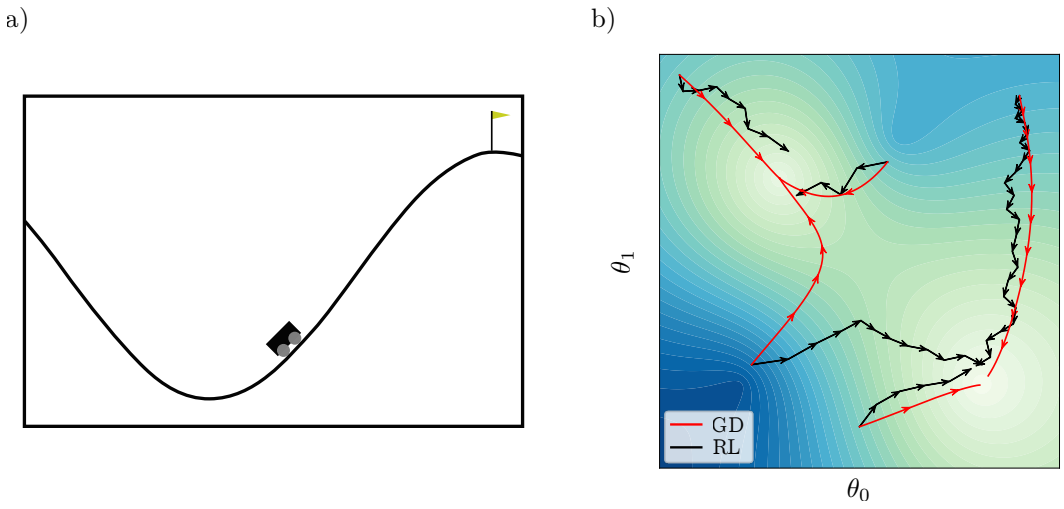


Figure 4.1: **RL learnt gradient-free descent in low-dimensional spaces.** Examples of RL for gradient-free (ascent) descent, in low-dimensional spaces. a) Classic RL toy control problem 'MountainCar', where the agent has to learn to control a car to reach the top of a hill. b) Toy example of learnt RL optimiser versus traditional gradient descent (GD) in 2d parameter space with smooth, random loss landscape, and multiple stationary points, with five different initialisations. In contrast to large ML tasks, the input dataset is not stochastic, making this optimisation easier.

our knowledge this is the first work which uses RL for meta-learning in a PNN.

To further motivate using RL as a way of learning our optimiser m, we can consider an example of a classic benchmark control problem used to test RL algorithms, 'MountainCar', illustrated in Figure 4.1.a. In this picture, we have a vehicle which we control, with the goal of reaching the flag at the top of the hill. The feedback given is extremely sparse, with a constant negative signal at every time step, which only stops when we are at the flag, where the signal becomes zero. In the original problem, the car is underpowered, requiring a complex control policy which is able to build potential energy through oscillations, to reach the top. We can view this hill climbing task as a convenient analogy for learning to reach an optima without gradient signals, one which, while challenging for RL agents, is known to be solvable. In hill-climbing optimisation tasks, we can do better than 'MountainCar', as we have access to a more informative signal—the objective value (i.e. the height of the car below the flag).

This is further shown in Figure 4.1.b, where we demonstrate an RL controller (using the deep deterministic policy gradient (DDPG) [Lil+19] algorithm, detailed in the coming sections), referred to as an agent, which has been trained to solve a simple 2d optimisation problem with a smooth, random loss landscape. The landscape and starting points are randomly initialised, and the agent receives the current loss as its error signal to minimise. This is a dense, informative signal which can be followed to reach a local optima. We compare it with regular gradient descent (GD), and while

both RL and GD do get stuck in local minima, their performance is comparable on this toy task. One point to note is that the learnt RL step size is typically larger than the step size of the GD method[1]. While this is a hyperparameter in GD, in our RL controller, this step size is learnt, potentially allowing the agent to converge to an optima faster in well-behaved loss landscapes. We will discuss these points further in the larger results sections, but this example serves to illustrate the potential of RL for gradient-free optimisation.

In the remainder of this chapter, we formally introduce RL and apply it to learn a custom gradient-free optimiser for training general PNNs *in situ*. We will consider a relatively simple, simulated high-dimensional PNN implemented with diffractive optics as a case study, and train it to solve a range of tasks using this RL-based meta-learning approach. We then extend the scheme to consider training networks composed of multiple, sequentially stacked PNNs, where our learnt optimiser's complexity scales with the complexity of the individual PNNs, not the complexity of the overall network.

We show that our learnt optimiser can improve training time and final accuracy compared to existing gradient-free methods, although in the analyses we have performed so far, we see that it remains more sensitive to initial conditions and is somewhat brittle to task distribution and PNN architecture. We go on to discuss the implications of these results and avenues for future work to further stabilise this approach, making it a viable option for future physical computing.

4.2 Reinforcement learning

Reinforcement learning has existed in one form or another since the earliest days of computing. In 1948, Alan Turing hypothesised a type of computer which could react to pleasure and pain in order to learn, comparing it to how a child learns [Tur04; Web11]. These ideas of experimentation and observing the consequences of our actions are considered to be fundamental parts of how humans learn, and it's natural to consider how this might apply to machine learning systems, and indeed neuromorphic systems.

The key idea in RL is to frame the task to be solved as a form of game, where an 'agent' (a decision-making process) can interact with an 'environment' (a task) and perform trial-and-error learning. For instance, if you want to teach a dog to sit, then backpropagation is unlikely to be particularly useful. You have to make use of the tools at your disposal, which in this case are the dog's ability to observe and correlate actions with rewards. If you give the dog a treat every time it successfully sits on command, eventually this becomes learnt behaviour, and the dog sits on command regardless of

[1] This can be seen in the relative smoothness of the trajectories. The arrow-heads in GD do not correspond one-to-one with steps.

reward provided on completion.

In the language of RL, the dog is the agent, and the environment is the game of listening to commands, choosing an action, and potentially receiving a reward. Every time you give a command, the dog will react in some way. You could imagine that if you just repeat the word 'sit' over and over again, little will be achieved. The dog may wander around, but unless it happens to choose to sit at the exact correct moment, it will never receive a reward and will remain unaware of the potential rewards which come from sitting.

RL is therefore a balance of exploration (trying new behaviours) and exploitation (using what you have learnt to solve tasks). In the following sections, we will formalise the concepts, language and notation used in RL, the various means with which we can achieve this balance, and some key challenges in implementing robust RL, before applying it to our meta-learning system.

4.2.1 To model or not to model...

There are two high-level classes of reinforcement learning algorithms, delineated by the prior information about the environment they have access to.

Model-based RL algorithms have access to a model of the environment, which allows them to predict the outcome of actions before they are taken. This allows the agent to plan ahead and avoid actions which are likely to lead to poor outcomes. There are many similarities here with ideas in control theory such as model predictive control (MPC), which uses a model to predict the optimal control for a system up to some finite-time horizon. The complexity of the model limits the achievable planning horizon, and typically requires some form of linearisation in order to make the optimisation tractable. Model-based RL differs from MPC in that the policy can use the model in different ways to improve the action (control) predictions.

A 'world model' is an extreme model-based approach, where not only is the agent's policy based off its internal model, but the training occurs in the model, referred to in [HS18] as 'training in the dream'. This approach is powerful in cases where the real environment is expensive or unsafe to evaluate, for instance in robotics.

However, all the same challenges with model-based methods for training a PNN apply here. The most common barrier is the lack of a suitable model of the environment. A model can either be built by hand based on the known dynamics of a system, can be empirically measured through techniques such as system identification, or can be learnt directly using techniques such as deep learning. In the case of the latter two, significant interaction with the environment can be required in order to build a suitable proxy for the system.

In any model-based system there will be issues with model bias. Modelling inaccuracies led to differences between the model and the real system, which can be exploited by an agent in potentially unexpected ways. This can lead to an agent which performs well in simulation against the model, but poorly when deployed against the real system of interest.

The alternative is to use model-free algorithms, which are less sample efficient, but tend to be easier to build, and will be our primary focus in the coming sections.

These are most similar to our example of training a dog to sit, in that the understanding of the game and the potential rewards is built up through experimentation and observation alone. The agent learns to predict the value of different actions based on the rewards it receives, and uses this information to guide its future actions. There is no explicit model, however through this experimentation, the agent will learn some empirical rules about the environment which can be used to plan and make decisions. These rules may be simpler than the true environment dynamics, and may even incorrectly attribute cause and effect—all that matters is that they are able to successfully guide decision-making.

4.2.2 Environments

The first task in setting up any RL problem is to define our environment $\mathcal{E}$. For the reader more familiar with supervised learning, this is a wrapper around our task, used to generate data with which to train our agent's control policy neural network, π_ϕ, which serves as the m in the previous section. The main distinction with supervised learning is that here, we build up extra machinery to automatically label this data, based on performance on the task.

The environment is a process which takes actions and returns observables, along with some performance metric which we call the reward r. We will only consider finite, discrete-time environments, where we take some action each time step, and get back an observation of the environment, up to a maximum number of time steps T. Examples of environments include the game of sit as described earlier, or a robotic arm tasked with stacking blocks, where final height is the performance metric.

We can simplify life for ourselves by restricting our environments to be Markov chains. A process is said to have the *Markov property* if the conditional probability distribution of future states depends only upon the present state, not on the sequence of events that preceded it. In other words, the process is memoryless. A *Markov chain* is a stochastic process, either continuous or discrete in time, with the Markov property, defined by a set of states $\mathcal{S} = \{s_i\}$, a transition probability $P(s' \mid s)$, and an initial state distribution $P(s_0)$. This means that the probability of transitioning from one state to another depends solely on the current state, which lets us avoid having to consider the

environment having memory of past events.

Markov chains are a useful tool for modelling a wide range of (potentially stochastic) systems. The assumption of the Markov property allows us to simplify the analysis of a system, and in cases where memory is important, it is often possible to turn a non-Markovian process into a Markovian one at the expense of increasing the state-space to include all relevant information.

When using a Markov chain to model a stochastic system, the type of model depends on the observability of the state and whether the process is autonomous. Fully observable autonomous systems are the simplest and are represented by normal Markov chains. In the case where the system is controlled (i.e. there is an external input $a \in \mathcal{A}$ at each time step for some set of actions $\mathcal{A}$), the transition probability now also depends on the action at each time step. This is known as a Markov decision process (MDP), $(\mathcal{S}, \mathcal{A}, P)$, as the agent can influence the evolution of the state over time through its actions.

If we only partially observe the process, then we have a partially observable Markov decision process (POMDP), $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, Q)$. Here, $\mathcal{S}$ is the set of possible states as before, $\mathcal{A}$ is the set of possible actions, $P(s' \mid s, a)$ is the transition probability, $\mathcal{O}$ is the set of possible observations, and $Q(o \mid s')$ is the observation probability, for $s \in S$ and $a \in \mathcal{A}$.

In all examples we will consider, we operate in discrete time. This means that the environment functions as a mapping $\mathcal{E} : (s_t, a_t) \to (s_{t+1}, o_{t+1}, r_{t+1}, c_{t+1})$ from state-action pairs to a new state, new observation, reward r, and termination condition c. The reward is the signal the environment provides to indicate the value of the current state, and is the only way we have of assessing performance on the task. For our purposes, r may take any real value, with negative rewards corresponding to penalties, aiming to discourage a particular state. The environment is probabilistically initialised to some state s_0, and for simulation purposes is restricted to run a finite maximum number of time steps T. Additionally, the environment can terminate early (indicated by c) in the case of a success or failure condition i.e. the dog sits, the dog died.

4.2.3 Agents

The other half of the RL equation is the agent. This is the decision-making process which implements some policy $\pi_\phi : \mathcal{S} \to \mathcal{A}$ mapping a state to an action, parameterised by ϕ. Our entire goal in RL is to optimise this policy to be as close to the optimal policy π^*, i.e. the policy which will best achieve the intended goal, as defined by the rewards.

The design of the agent can have a large impact on the learning performance. We need to ensure that the actions taken by the agent are diverse so that there is sufficient exploration of the state space to experience as complete a range of possible rewards as

possible. This can be done either through initialisation of the policy of the agent, or by overriding or forcing the actions during learning. In the case of our dog, we don't have direct control over the initial behaviour, but in custom-built agents we can design their initial reactions to various stimuli.

4.2.4 Reward structures

The reward is the only signal that the agent receives to guide its behaviour, and as such is critical to the learning process. Every environment is equipped with a *reward function*, which maps state to a single real value. There are two main approaches to writing reward functions: sparse and dense. In finite-length environments the ideal situation is to provide positive reward only when the task is solved i.e. the robot gets to the goal, or the dog sits. This is the least ambiguous, as it is clear what good behaviour looks like, however it can be extremely difficult to learn, as the agent has to rely on accidentally solving the task during exploration in order to ever receive a positive signal. If you and your dog play the game of 'sit' millions of times, then statistically the dog will receive *some* rewards, but due to the sparsity, the learning process is incredibly inefficient.

The alternative is to provide dense, continuously varying rewards at each time step. This provides more information to the agent and encourages behaviours which led to the desired goal, but it requires being able to define what partial completion looks like, and a prior understanding of which behaviours might lead to the desired goal.

For more complex tasks, the biases of a dense reward system may actually be detrimental to learning, as an agent can develop strange, unexpected behaviour to gain rewards. Indeed, RL algorithms are notorious for exploiting loopholes in reward functions. Consider providing small negative rewards at every time step to encourage a task to be completed quickly. A common failure mode here is the agent learning to terminate the task as fast as possible, for instance by driving the system towards invalid or illegal states. The reward function leads to suicide being more desirable than attempting to complete a difficult task, even with a large reward at the end.

Reward scale is another important consideration: too small, and the agent may not learn fast enough; too large and the agent may learn to farm reward, not exploring the state space any further. In a dense reward function, the aim is to provide a steady reward gradient towards the goal, which can be used to guide the agent towards the optimal policy. A challenge here is that it risks imposing human bias in *how we think the task should be solved* versus the actual optimal solution.

Consistency is important, as highly stochastic environments where the reward changes for the same state-action pair can confuse the agent and lead to slower learning, or even completely impede learning in the worst case. Even if rewards are deterministic, but

the process is partially observable, as far as the agent is concerned this is a stochastic environment, and the benefit of targeting a reward is conditioned on the likelihood of the hidden variable(s) being in a favourable state(s).

Designing a reward function which is both informative and not too complex can be a significant challenge.

4.2.5 Value functions

In reinforcement learning (RL), one constant is that we need some form of value function in order to estimate performance of the method. While rewards indicate immediate quality of a given action in a state, value functions extrapolate to include all future rewards, giving a long-term metric, critical for planning in decision-making processes. This means that they intrinsically depend on the choice of policy used to select future actions. Once defined, a value function can be used to select actions, to estimate desirable states, and for optimising the decision-making process, in the absence of a ground truth model of how actions will influence overall task performance.

The foundation of value estimation in RL is the Bellman equation [Bel10],

$$V(s) = \mathbb{E}_\pi \left[r_t + \gamma V(s_{t+1}) \mid s_t = s \right], \tag{4.1}$$

which allows one to understand the value of a particular state s (or action a) at a point in time, based on the immediate reward r and the (discounted, $\gamma \in [0, 1]$) value of the subsequent state, under a particular policy π. This was originally derived in the context of dynamic programming[2], and provides a proven necessary condition for optimal optimisation of a recursive process [Sut20]. In the context of reinforcement learning, each step in a time-domain task can be viewed as a step in a recursive process, allowing us to use value functions which satisfy the Bellman equation as tools for optimising a control policy.

Note that the type of task can influence the form of the value function. In tasks where there is a natural time limit, it makes sense to use a finite-horizon value function, which only considers the future up to a certain time step. This results in a time-dependent value function, and in turn can lead to time-dependent policies. The alternative is an infinite-horizon value function, which considers all future rewards (i.e. is stationary) and which is more suited to open-ended tasks.

In both cases, it is important to weight future influence with a discount factor γ. This ensures convergence of the potentially infinite sums, while also weighting towards longer or shorter-term outlooks.

[2]Much of the theory behind RL was developed in the context of dynamic programming, control theory, and optimal control.

The form of the value function, often determined by the space of states and actions in the task of interest, has significant implications on how we can develop a control policy. Tasks with spaces which are either discrete, or small enough that they can reasonably be discretised, allow tabular methods to be used. Here, the value function is stored in a learnt lookup table, from which we can look up the value for all possible actions from a given state and choose the best one. For more complex, continuous or stochastic tasks, we tend to use function approximation methods to estimate the value function. To generate a good policy from such a function is a challenging optimisation problem, and sets the scene for most modern reinforcement learning algorithms.

In contrast to typical supervised machine learning approaches, one large challenge in this form of RL is that we need to simultaneously learn the value function and the policy. The coupling between the two make this a non-stationary learning problem, where the balance between stability, exploration of the state-action space, and overall performance is critical for solving the task at hand.

There are several types of value functions, each with a different purpose and use case, and we provide an overview of the common ones here.

State-value function (V-function). Denoted as $V(s)$, this represents the expected return (total future reward) when starting in state s and following a particular policy π, and is given as

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s\right], \tag{4.2}$$

where γ is the discount factor (between 0 and 1), and r_{t+1} is the reward at time $t+1$. It provides a measure of how good it is to be in a specific state under a given policy, allowing policy evaluation and comparison.

As V is always measured according to a policy, it is common to also define the optimal state-value function V^*, i.e. the value under the optimal policy π^*.

Action-value function (Q-function). The action-value function $Q(s, a)$ develops the state-value function to also include the next action. Starting from state s, taking action a, and following a policy π for all future actions, the action-value function is defined as

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a\right]. \tag{4.3}$$

This is key for planning algorithms, as it allows us to understand the consequences of choosing a particular action. Again, it is common to consider the optimal action-value function Q^*, where the optimal policy is followed when generating future actions.

Closely related is the expected state-value function $Q^{\pi}(s)$, which is used to evaluate

the expected return under stochastic policies, and is defined as

$$Q^{\pi}(s) = \sum_{a} \pi(a \mid s) Q^{\pi}(s, a). \tag{4.4}$$

Advantage function. The advantage function $A(s, a)$ measures the difference in value of an action a compared to the average (or baseline) action in state s, according to the policy π, and is given as

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s). \tag{4.5}$$

By decoupling the value from the absolute value of the current state, advantage can reduce variance and improve stability in policy gradient methods, in turn improving learning efficiency. Many implementations of current state-of-the-art policy gradient methods use the advantage function in the policy updates [Sch+15; Mni+16].

4.2.6 On- vs off-policy learning

While we can easily write out a given value function, in practice generating one for a specific task needs to be done through statistical estimation from data gathered by interacting with the environment. When we start to consider the practical consideration of using RL on real tasks and the challenges with learning a policy from data generated by a constantly changing policy, the first distinction which should be made is how the data is gathered, and this falls into one of two camps.

On-policy learning is the simplest approach, where the policy which is being optimised is also used to generate the data which is used to optimise it. This is the most common form of RL and is used in many of the state-of-the-art algorithms. The advantage of on-policy learning is that it is simple to implement, and the policy is guaranteed to improve over time. However, it can be sample inefficient, as the policy is constantly changing, so prior generated data cannot be used. This means that the policy must be good at exploring the state-action space in order to learn about it, which can be inefficient in large or complex state-spaces. Exploration strategy is key for the policy to improve, meaning that the policy must be forced to take actions it would consider suboptimal some percentage of the time in order to potentially find better actions.

Off-policy learning by contrast uses a different policy to generate the data from the one which is being optimised. This allows for more efficient use of data, as the policy can be trained on old data, i.e. from a different training run, an older version of itself, or specifically a policy which is good at exploring the state-action space for data generation. This can lead to more sample-efficient learning, as the policy can be trained on a larger dataset, with better diversity. There is however a trade-off, in that the difference in distribution between the data being used and the policy being trained needs to be accounted for to ensure stability in the learning process. This can be reconciled through

importance sampling.

Finally, it is worth briefly mentioning another distinction in how data is used: online and offline learning. These refer respectively to learning through real time interaction with the environment, versus learning from a previously gathered, fixed dataset of interaction experience. Most examples we consider are online, however we discuss offline learning later due to the benefits it can bring in data utilisation efficiency.

4.2.7 Importance sampling

Importance sampling is a method to estimate properties of one distribution by sampling from another. Consider that we have some random variable $\mathrm{x} \sim f$, and we want to estimate the statistic $\hat{h}_\mathrm{x} = \mathbb{E}_f\left[h(\mathrm{x})\right]$ for some function h. Importance sampling allows us to write this in terms of samples from a different distribution $\mathrm{y} \sim g$ as $\hat{h}'_\mathrm{x} = \mathbb{E}_g\left[h(\mathrm{y})f(\mathrm{y})/g(\mathrm{y})\right]$. This can be seen by rewriting as

$$\mathbb{E}_f\left[h(\mathrm{x})\right] = \int h(x)f(x)dx = \int h(x)\frac{f(x)}{g(x)}g(x)dx = \mathbb{E}_g\left[h(\mathrm{y})f(\mathrm{y})/g(\mathrm{y})\right] \tag{4.6}$$

An important consideration here is that while the estimates may coincide (i.e. $\hat{h}'_\mathrm{x}$ is unbiased), the variances are not equal, as can be seen below:

$$\begin{aligned}
\mathrm{Var}_f\left[h(\mathrm{x})\right] &= \mathbb{E}_f\left[h(\mathrm{x})^2\right] - \mathbb{E}_f\left[h(\mathrm{x})\right]^2 \\
&= \mathbb{E}_g\left[h(\mathrm{x})^2\frac{f(\mathrm{x})}{g(\mathrm{x})}\right] - \mathbb{E}_g\left[h(\mathrm{x})\frac{f(\mathrm{x})}{g(\mathrm{x})}\right]^2 \\
&\neq \mathbb{E}_g\left[\left(h(\mathrm{x})\frac{f(\mathrm{x})}{g(\mathrm{x})}\right)^2\right] - \mathbb{E}_g\left[h(\mathrm{x})\frac{f(\mathrm{x})}{g(\mathrm{x})}\right]^2 \\
&= \mathrm{Var}_g\left[h(\mathrm{x})\frac{f(\mathrm{x})}{g(\mathrm{x})}\right].
\end{aligned}$$

This has implications when considering the confidence in an estimate, as the standard error on some estimate $\hat{\mathrm{x}}$ is $\sigma_{\hat{\mathrm{x}}} = \frac{\sigma_\mathrm{x}}{\sqrt{n}}$ for n samples. If the importance sampled estimator has high variance, then it may be infeasible to get a good estimate of the statistic, even with many samples. There are several variants which address this, with one simple approach to clip and weight the ratio of likelihoods such that the quadratic term in the variance is prevented from growing too large, introduced in the Retrace algorithm [Mun+16]. While this does bias the estimator, it significantly improves the variance and has guaranteed convergence when used to estimate certain classes of value function.

An illustrative example of importance sampling can be seen in Figure 4.2, where we sample from a truncated normal distribution $\mathrm{x} \sim \mathcal{T}_{[0,1]}(2,1)$ and a uniform distribution $\mathrm{y} \sim \mathcal{U}(0,1)$, and estimate the value of $h(x) = x^2$ under both distributions. The statistics for both are shown in Table 4.1.

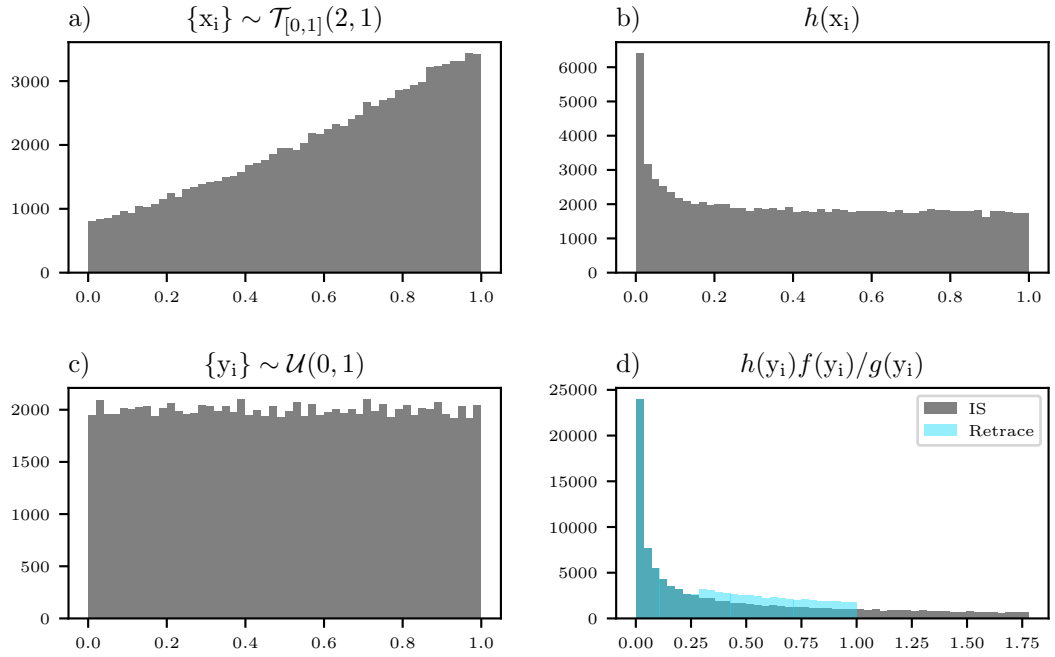


Figure 4.2: **Importance sampling example.** Histograms of a) 10^5 samples $\{x_i\}$ from the truncated normal distribution $\mathcal{T}_{[0,1]}(2,1)$ with p.d.f. $f(x)$, b) $h(x) = x^2$ evaluated on $\{x_i\}$, c) 10^5 samples $\{y_i\}$ from the uniform distribution $\mathcal{U}(0,1)$ with p.d.f. $g(x)$, and d) Importance sampling (IS) $h(x)f(x)/g(x)$ and Retrace $h(x)\min\{1, f(x)/g(x)\}$ evaluated on $\{y_i\}$. Statistics for both b) and d) are shown in Table 4.1.

Off-policy algorithms which use importance sampling derive their improved sample efficiency from being able to reuse old data generated by earlier versions of the policy, or even from entirely different training runs. The is achieved through replay buffers—temporary stores of trajectories recorded earlier in training, which are used with importance sampling to update the policy.

Table 4.1: **Importance sampling statistics.**

	Ground Truth $r = \{x_i\}$ $q(x) = h(x)$	Importance Sampling $r = \{y_i\}$ $q(x) = h(x)f(x)/g(x)$	Retrace $r = \{y_i\}$ $q(x) = h(x)\min\{1, f(x)/g(x)\}$
$\mathbb{E}_r[q(x)]$	0.452954	0.452420	0.323891
$\mathrm{Var}_r[q(x)]$	0.304588	0.500734	0.304616

4.2.8 VALUE ESTIMATION

Now that we have the value functions defined, methods for gathering the data, and methods for estimating relevant statistics on these the data, we need a way to turn this into an algorithm for calculating values and using them to inform the control policy. Specifically, how do we update an initial guess for the value function to better align with the ground truth.

One of the simplest techniques is Monte Carlo (MC) estimation, which samples many trajectories originating from each state under a given policy and averages the final return. While this will converge to the ground truth in the limit, it typically requires many samples to converge and reduce noise, especially in environments with large state-spaces, where some states may be visited infrequently. MC also only considers finite tasks where there is a definite endpoint, and open-ended tasks must be manually truncated to provide a return. Considering only the final return gives the long-term quality of a particular state, however it may also be desirable to consider shorter term rewards.

Temporal difference (TD) learning is a practical alternative for value function estimation and is core in many of the modern RL techniques. It can be seen as a direct solution to the Bellman equation, where the value of a state is calculated as the immediate reward plus the estimated value of the next state. Estimates are updated based on other learnt estimates (i.e. bootstrapping), making it well-suited for ongoing, real-time learning and environments where episodes can be very long or even infinite. As TD typically only considers the immediate next state's estimated value during this bootstrapping process, it is in some sense opposite to Monte Carlo, which only considers the terminal states' values, weighted by the probability of reaching each state. While this may change the resulting value estimates, the ability to learn from partial sequences of data is important for efficiency, as we don't need the full episode trajectory and can update estimates continuously.

In order to actually use TD to update the value function estimates, we calculate the TD error δ, which is the difference between the predicted value and the newly observed value. For a state s when taking action a and moving to state s' with reward r, δ is given by

$$\delta = r + \gamma V(s') - V(s), \tag{4.7}$$

where γ is the discount factor as before.

This provides a signal which can be used to update the value estimate, as minimisation of this error corresponds to aligning the estimate value function with the ground truth. In the simplest cases, where we maintain a table of values, we can directly add this error to the current state's value estimate, which is what's done in the TD(0) algorithm. TD(λ) is a more general algorithm which introduces *eligibility traces*, a method for taking into account future value estimates [Sut20]. The parameter λ controls the balance between TD(0), using only the next state, and MC methods, which use the entire episode.

In the more complex tasks where the tabular approach is infeasible, we use function approximation to represent the value estimates, and in turn develop a control policy. There are a range of algorithms which use the TD error to achieve this, differentiated

by which samples they use, how they update the value function, and how they balance exploration and exploitation. SARSA (State-Action-Reward-State-Action) is an on-policy TD control method where updates are made using the state-action pairs that the agent actually follows. Q-Learning is an off-policy TD control method where the agent updates its Q-values using the maximum possible reward of the next state, regardless of the action it actually takes, i.e. it assumes we take one step and then follow the optimal policy thereafter.

One feature briefly worth mentioning which is used in many RL algorithms is the concept of 'target networks'. These are copies of the policy or value estimation networks which are updated less frequently than the main networks, improving stability. The need for target networks comes from wanting to both continually update our approximation of the value function, while also using it to derive a policy. By using a target network, we allow more experiences to be gathered and applied before deploying the updates to the main networks which decide our actions.

The ability to use TD methods in on-policy and off-policy settings makes it flexible, and TD learning still forms the backbone of the state-of-the-art models. Algorithms which don't use TD include REINFORCE, which is less widely used these days, MC methods, and some model-based methods which may not explicitly use TD.

4.2.9 Policy optimisation

Tabular, value-based methods are excellent where it is possible to actually enumerate states and actions, but most real-world tasks have state-spaces which are too large. In these cases, we need to use function approximation to estimate the value function, and in turn develop a control policy. Assuming the value function can be learnt accurately, deriving a policy consists of maximising this with respect to available actions. This is often non-trivial.

The policy-gradient is one method, which allows us to take the derivative of the policy in order to optimise its parameters based on an error signal on the predicted action, which can in turn be obtained by differentiating the value function. The REINFORCE algorithm, developed in 1987, is one of the earliest policy gradient methods, allowing the use of the derivative of the learnt policy to optimise the policy parameters [Wil92]. Today, policy gradient methods are the most common form of RL used in practice, and have been used to solve a wide range of complex problems, including solving Atari games, playing Go, protein folding, and controlling robots and fusion reactors [Sil+17; Jum+21; Deg+22]. This has been made possible by the development of GPGPU compute and deep learning, providing tools for efficiently parameterising and training complex policies.

This form led to the current set of actor-critic methods which, despite having been

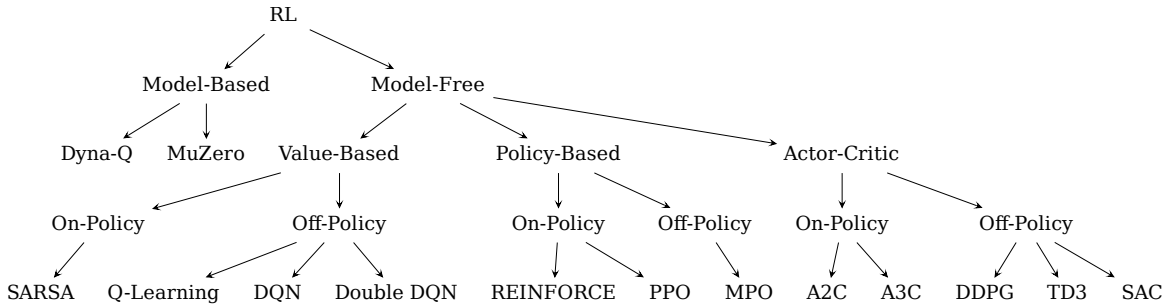


Figure 4.3: **RL taxonomy.** Set of common RL algorithms, classified by their main features. In this work, we use DDPG, MPO and SAC, which are all model-free off-policy methods. Model-based methods have seen use in DeepMind's Alpha- series of models, demonstrating state-of-the-art performance in chess, go and protein folding problems [Jum+21; Ber22], while deep neural network based model-free methods have seen significant use recently in robotics and control tasks, with milestone successes including learning Atari [Mni+13] (Q-learning), and tokamak control [Deg+22] (MPO).

known of for many years [BSA83; PS08], only became practical and successful on larger tasks recently, again on the coattails of deep learning.

For the interested reader, we provide a brief taxonomy of some historically relevant and common modern RL methods in Figure 4.3. Of these, DDPG, maximum *a posteriori* policy optimisation (MPO) and soft actor-critic (SAC) (all off-policy actor-critic methods) will be discussed in the coming sections.

4.3 Optics

While the focus of this chapter is primarily on the optimisation algorithms, we do need some sort of physical system to test them on. We now turn our attention to the particular PNN we will be using and the theory behind it.

In order to demonstrate scalability in our learnt optimiser, any PNN we use should be high-dimensional, nonlinear and non-differentiable. It should be stable over the expected lifetime, low in noise, and high efficiencies in speed, energy consumption and cost are also desirable.

We have already established the benefits which optical systems can provide in the context of neuromorphic computing. The many degrees of freedom of light make it an excellent candidate for encoding information. Both diffractive and integrated optics have been used in currently-available, commercialised, high-dimensional optical accelerators [Cav+22]. However, with high-speed, high-dimensional optical modulators now available, diffractive optics is particularly well suited to fast, parallel information processing in a research setting [Hu+24].

Here we will develop a simple diffractive optical PNN capable of performing operations equivalent to general convolutions and matrix multiplications to test our optimisation

algorithms on, which satisfying the high-dimensional and nonlinear[3] requirements that our PNN should have to be an interesting candidate for RL-based optimisation.

In order to do this, it would be useful to have a working understanding of the theory behind some optical elements that we will be using. This will allow us to simulate the system, and also leads us to identify useful physical properties which we can exploit in the optimisation process.

4.3.1 Optical PNN

Here, we introduce a coherent, diffractive optical PNN, based around the use of LCoS SLMs for encoding digital data in the wavefront of a propagating beam.

SLMs are pixelated displays controlled by a computer which can imprint the data displayed on them in various degrees of freedom of an optical field, i.e. they modulate the field. They typically act on either phase, amplitude or polarisation, they are capable of operating between 100–10 kHz, with newer devices and schemes ever-increasing in speed and resolution [Tza+19; Wan+19b; Par+20]—modern devices can have resolutions of several megapixels. The fastest devices, digital micro-mirror devices (DMDs), use an array of controllable micromirrors which can deflect light in one of two directions. There are imaging schemes which can be used to generate complex phase and amplitude distributions despite the binary nature of the device, however this is at the expense of spatial resolution. LCoS devices apply a voltage across a liquid crystal layer to induce controllable phase retardation in one linear polarisation axis, which in turn allows modulation of the phase or polarisation of light. While typically slower than DMDs due to the limits of the liquid crystal's response time, they are capable of continuous phase modulation with 8 and 16bit dynamic ranges, and are more suited to high-resolution applications. Demand from a range of industries has driven the development of high-speed, high-resolution, relatively economic commercial devices, and new technologies hint at even faster modulation devices in the near future.

As discussed in Section 2.1.1, diffractive optical systems have long been known to provide analogue approaches to mathematical operations such as matrix multiplications and Fourier transforms [Goo96]. With the Fourier transform, we can also naturally implement a range of integral transforms such as convolutions. SLMs allow digital control of the optical fields needed to implement these operations on data of our choosing.

We consider a system where a coherent beam illuminates a phase-only SLM which encodes our input data x. The encoding is a linear mapping from integer pixel values in the range $\{0, \ldots, 255\}$ to phase values in the range $[0, 2\pi]$. The encoded field is then Fourier imaged onto a second phase-only SLM which applies trainable parameters θ.

[3] In the input data.

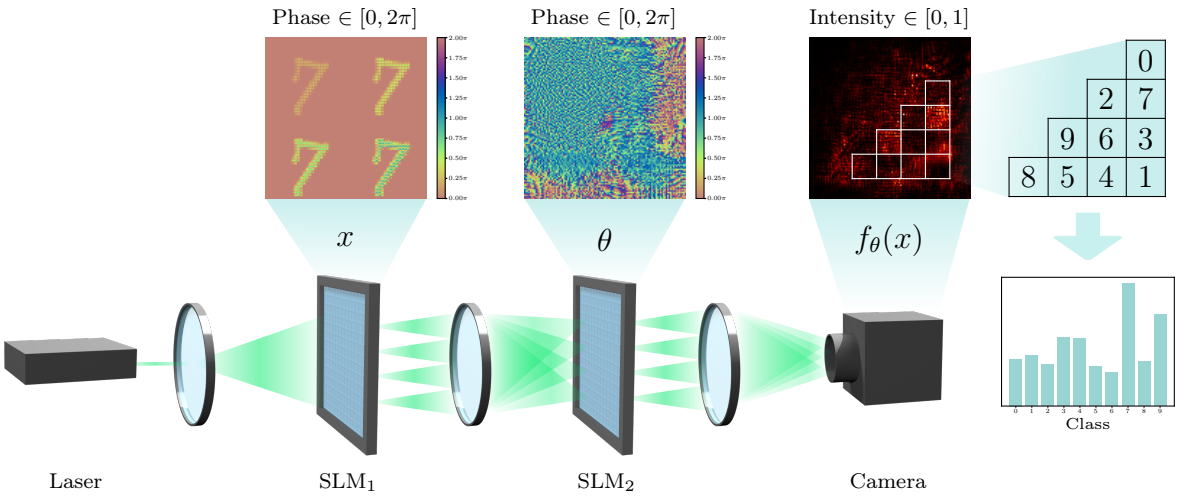


Figure 4.4: **Optical PNN implementation.** Phase-only spatial light modulators (SLMs) encode data x and parameters θ. Output $y = f_\theta(x)$ is far-field intensity imaged via camera. The white grid corresponds to macropixels used in classification tasks, with class labels determined by task-specific permutation. See below for discussion of the tiled input encoding scheme.

This in turn is Fourier imaged onto a camera, with the intensity image used as the output of the PNN. This is illustrated in Figure 4.4.

This system can then be used for a range of tasks such as regression or classification, where the output dimension can be reduced through the use of macropixels. Figure 4.4 shows the PNN in use as an MNIST digit classifier, where the input image is displayed on the first SLM with some encoding[4] mapping data to SLM pixels. The output intensity is binned into macropixels, with each assigned a label and the most intense macropixel taken as the predicted label.

We can write a simplified model of this system as

$$y(u', v') = \left| \mathcal{F}\left[e^{i\theta(k_u, k_v)} \mathcal{F}\left[e^{ix(u,v)} \right] (k_u, k_v) \right] (u', v') \right|^2, \tag{4.8}$$

where (u, v) are the spatial coordinates in the input plane, and $k_\square$ and $\square'$ are the spatial coordinates in the parameter and camera planes respectively.

This simple model is nonlinear, making training parameters θ without gradient-based methods challenging. The primary advantage of the proposed PNN lies in the parallelism and scalability of its input, parameter, and output spaces, as the SLMs and camera can all be high resolution. The ability to use macropixels allows us to reduce the dimension of the system to match that of smaller optimisation tasks.

[4]Here, the tiling improves performance at no extra cost by providing multiple channels to be processed in parallel. This is an empirical observation and depends on the task at hand, and specific form of PNN.

4.3.2 Error propagation

While the hardware implementation of the PNN is non-differentiable, we can still consider this approximate, analytic model of the system, and use this to understand how errors propagate. This deeper understanding of the dynamics of the forward model allows us make more informed decisions when designing the optimisation algorithm.

For clarity of notation, we will first introduce a conjugation operator C, as a map on the space of multivariate functions from the reals to the complex numbers, $\mathcal{G}(\mathbb{R}^n, \mathbb{C}^m)$, defined as

$$C : \mathcal{G}(\mathbb{R}^n, \mathbb{C}^m) \to \mathcal{G}(\mathbb{R}^n, \mathbb{C}^m)$$

such that for any function $f \in \mathcal{G}(\mathbb{R}, \mathbb{C})$, we have

$$(Cf)(x) = f^*(x)$$

for all $x \in \mathbb{R}^n$, $n, m \in \mathbb{Z}$, where $f^*(x)$ denotes the element-wise conjugate of $f(x)$.

Now taking the simplified forward model of the PNN in Equation 4.8, in order to calculate the derivatives of this, we break it down as

$$y(u', v') = |Y(u', v')|^2 = Y(u', v') \odot Y^*(u', v') \tag{4.9a}$$
$$Y(u', v') = \mathcal{F}\left[H(k_u, k_v)\right](u', v') \tag{4.9b}$$
$$H(k_u, k_v) = \Theta(k_u, k_v) \odot G(k_u, k_v) \tag{4.9c}$$
$$G(k_u, k_v) = \mathcal{F}\left[X(u, v)\right](k_u, k_v) \tag{4.9d}$$
$$X(u, v) = e^{ix(u,v)} \tag{4.9e}$$
$$\Theta(k_u, k_v) = e^{i\theta(k_u, k_v)}. \tag{4.9f}$$

We can write the partial or functional derivative of each intermediate step as

$$\frac{\partial y(u', v')}{\partial Y(\tilde{u}', \tilde{v}')} = \delta(\tilde{u}' - u', \tilde{v}' - v')(Y^*(\tilde{u}', \tilde{v}') + Y(\tilde{u}', \tilde{v}')C)$$
$$\frac{\delta Y(u', v')}{\delta H(k_u, k_v)} = \mathcal{F}[\delta(k_u, k_v)](u', v') = e^{-i2\pi(u'k_u + v'k_v)}$$
$$\frac{\partial H(k_u, k_v)}{\partial G(\tilde{k}_u, \tilde{k}_v)} = \Theta(k_u, k_v)\delta(\tilde{k}_u - k_u, \tilde{k}_v - k_v)$$
$$\frac{\partial H(k_u, k_v)}{\partial \Theta(\tilde{k}_u, \tilde{k}_v)} = G(k_u, k_v)\delta(\tilde{k}_u - k_u, \tilde{k}_v - k_v)$$
$$\frac{\delta G(k_u, k_v)}{\delta X(u, v)} = \mathcal{F}[\delta(u, v)](k_u, k_v) = e^{-i2\pi(uk_u + vk_v)}$$
$$\frac{\partial X(u, v)}{\partial x(\tilde{u}, \tilde{v})} = iX(u, v)\delta(\tilde{u} - u, \tilde{v} - v)$$
$$\frac{\partial \Theta(k_u, k_v)}{\partial \theta(\tilde{k}_u, \tilde{k}_v)} = i\Theta(k_u, k_v)\delta(\tilde{k}_u - k_u, \tilde{k}_v - k_v).$$

We can now calculate the full derivatives of the output with respect to the inputs and parameters through the chain rule. Taking care with the integral over functional derivatives, we have

$$\begin{aligned}\frac{\partial y(u',v')}{\partial x(u,v)} &= \frac{\partial y}{\partial Y}\cdot\iint_{-\infty}^{\infty}\frac{\delta Y}{\delta H}\cdot\frac{\partial H}{\partial G}\cdot\frac{\delta G}{\delta X}\cdot\frac{\partial X}{\partial x}\\ &= (Y^*+YC)\iint_{-\infty}^{\infty}\left(e^{-i2\pi(u'k_u+v'k_v)}\right)\left(\Theta(k_u,k_v)\right)\\ &\qquad\times\left(e^{-i2\pi(uk_u+vk_v)}\right)\left(iX(u,v)\right)dk_u\,dk_v\\ &= 2\mathrm{Re}\left[iY^*(u',v')\mathcal{F}_{k_u,k_v}\left[\Theta(k_u,k_v)e^{-i2\pi(uk_u+vk_v)}X(u,v)\right](u',v')\right]\end{aligned}\tag{4.10}$$

and

$$\begin{aligned}\frac{\partial y(u',v')}{\partial\theta(k_u,k_v)} &= \frac{\partial y}{\partial Y}\cdot\iint_{-\infty}^{\infty}\frac{\delta Y}{\delta H}\cdot\frac{\partial H}{\partial\Theta}\cdot\frac{\partial\Theta}{\partial\theta}\\ &= (Y^\dagger+YC)\iint_{-\infty}^{\infty}\left(e^{-i2\pi(u'k_u+v'k_v)}\right)\\ &\qquad\times\left(G(k_u,k_v)\delta(\tilde{k}_u-k_u,\tilde{k}_v-k_v)\right)\left(ie^{i\theta(k_u,k_v)}\right)d\tilde{k}_u d\tilde{k}_v\\ &= 2\mathrm{Re}\left[iY^*(u',v')e^{-i2\pi(u'k_u+v'k_v)}H(k_u,k_v)\right].\end{aligned}\tag{4.11}$$

These derivatives can then be used to backpropagate any error signal on y through to the input x and parameters θ, allowing for training of the network using gradient-based methods.

Examining these derivatives more closely, we see that to calculate the error on θ, we need to know or recalculate H and Y, the fields immediately after the second SLM and at the camera respectively[5]. For the error on x, we also need the complex encodings X and Θ. To calculate these errors for a real optical system would therefore require a setup capable of making full field measurements at a number of points, greatly complicating the system.

For the purposes of numerically simulating this system, we need to discretise the equations. The forward model becomes

$$y_i = \left|\sum_{j,k}F_{ij}e^{i\theta_j}F_{jk}e^{ix_k}\right|^2,\tag{4.12}$$

while the derivatives become

$$\frac{\partial y_i}{\partial\theta_j} = 2\mathrm{Re}\left[iY_i^*F_{ij}H_j\right]\tag{4.13a}$$

[5]While we need to know the fields, and take the real part of the product, this is still independent of global phase as expected, due to the conjugate Y phase cancelling with the H phase.

$$\frac{\partial y_i}{\partial x_j} = 2\mathrm{Re}\left[iY_i^* \sum_l F_{il}\Theta_l F_{lj} X_j\right]. \tag{4.13b}$$

The Fourier transform $\mathcal{F}$ is now represented as the discrete Fourier transform (DFT) matrix F_{ij}, and x and θ are represented as vectors, where we have flattened the 2D input and phase vectors into 1D vectors for clarity.

4.3.3 High fidelity model

This idealised system is a good starting point, but misses much of the complexity in a real optical system.

The input illumination to the system is not going to be a perfect plane wave. SLM phase modulation is nonlinear and discretised according to addressable voltage levels. Accuracy of phase-voltage calibration may introduce differences between the true and simulated holograms. The SLM has a finite pixel pitch and active area, and non-unit fill factor, which lead to diffraction grating effects and a sinc intensity envelope in the far field. Unmodulated light needs to be filtered in order to observe the diffraction pattern induced by the hologram, which typically requires masking out some region of the addressable k-space. Manufacturing tolerances mean the active area of the SLM isn't perfectly optically flat, leading to erroneous wavefront differences compared to the simulation which may or may not be calibrated out. Any Fourier transform is, in practice, performed by lenses with finite aperture, thickness and aberrations. Choice of lenses affects magnification and therefore there may be differences in scale compared to the model, for instance when imaging field G onto the second SLM encoding the parameters θ. Alignment of optical components at each stage is also a challenge to match, with six degrees of freedom at each position. The camera used to measure the output has similar issues to the SLM in terms of finite area, fill factor, and discretisation, while the dynamic range and exposure constrain the information which can be captured in a single measurement. Saturation can lead to bloom artefacts in charge-coupled device (CCD) cameras, and particularly high intensities (i.e. from unmodulated light) will completely obscure the speckle signal of interest.

These are all factors which cause our simulated model to differ from reality, in turn limiting the effectiveness of any model parameters trained in simulation, when deployed on equivalent hardware.

We can build some of these effects into the model, for instance aberrations can be accounted for by extending the model to include parameterised phase masks between elements, using for instance a low-rank Zernike basis. Discretisation can be baked into the model. Using Rayleigh-Sommerfeld propagation and phase masks for lenses instead of Fourier transforms allows flexibility in the imaging of each pixel-device onto the next,

and the 6-DoF positioning of each element can be parameterised[6].

There comes however a point of diminishing returns, and the complexity of the model can quickly become unwieldy, where the extra free parameters introduced cannot easily be fit to the real system. Even with all of these factors, we have neglected drift in the powered devices (SLM, camera, laser), vibration and noise. While drift on longer-timescales can in theory be repeatedly calibrated out, the faster processes are more challenging and would require a non-stationary, stochastic model.

In order to train our optimiser on a representative model of the system we will write a higher-fidelity model taking into account some of these features, and use this to simulate the PNN. While it is correct that much of our work in designing a new optimiser aims to *avoid* complex simulations, for the purposes of development of the algorithms, it will be useful to have a 'cheat' model from which we can get ground truth gradients, and compare our optimiser against. The RL optimiser remains ignorant whether the forward model is simulated or implemented in hardware.

Given $\mathcal{L}(X, Y)$ is the space of continuous functions mapping space X to space Y, and $\mathbb{B} = \{0, \ldots, 2^b - 1\}$ for bit depth b, a more complete model $y : \mathbb{B}^{d_u \times d_v} \times \mathbb{B}^{d_{k_u} \times d_{k_v}} \to \mathbb{B}^{d_{u'} \times d_{v'}}$ of our PNN can be written as

$$y_{i,j} = \underbrace{D\Big[P_2\big[\underbrace{E_{d_{k_u} \times d_{k_v}}[\theta](k_u, k_v)}_{\text{Parameter encoding}} P_1\left[A(u,v)\underbrace{E_{d_u \times d_v}[x](u,v)}_{\text{Data encoding}}\right](k_u, k_v)\big]\,(u', v')\Big]_{i,j}}_{\text{Decoding}} \tag{4.14}$$

with input field $A \in \mathcal{L}(\mathbb{R}^2, \mathbb{C})$, encodings $E_d : \mathbb{B}^d \to \mathcal{L}(\mathbb{R}^2, \mathbb{C})$, propagators $P_{\{1,2\}} : \mathcal{L}(\mathbb{R}^2, \mathbb{C}) \to \mathcal{L}(\mathbb{R}^2, \mathbb{C})$ and decoding $D : \mathcal{L}(\mathbb{R}^2, \mathbb{C}) \to \mathbb{B}^{d_{u'} \times d_{v'}}$.

The encoding can be written as

$$E[x](u,v) = e^{i\phi(u,v)} \text{rect}\left(\frac{u}{d_u \Delta_u}\right) \text{rect}\left(\frac{v}{d_v \Delta_v}\right) \left((1-r) + r \prod_{i,j=0}^{d_u - 1, d_v - 1} \exp\left[i 2\pi \lfloor x_{i,j} / 2^b \rfloor K_e(u,v,i,j)\right]\right) \tag{4.15}$$

where (d_u, d_v) is the number of pixels, (w_u, w_v) is the pixel fill factor, (Δ_u, Δ_v) is the pixel pitch, $\phi(u, v)$ is a phase function to account for a non-flat SLM surface, r is the fraction of light which is modulated, and

[6] Although the freedom here is limited by the discretised simulation domain, which can introduce artefacts (i.e. aliasing) if care is not taken.

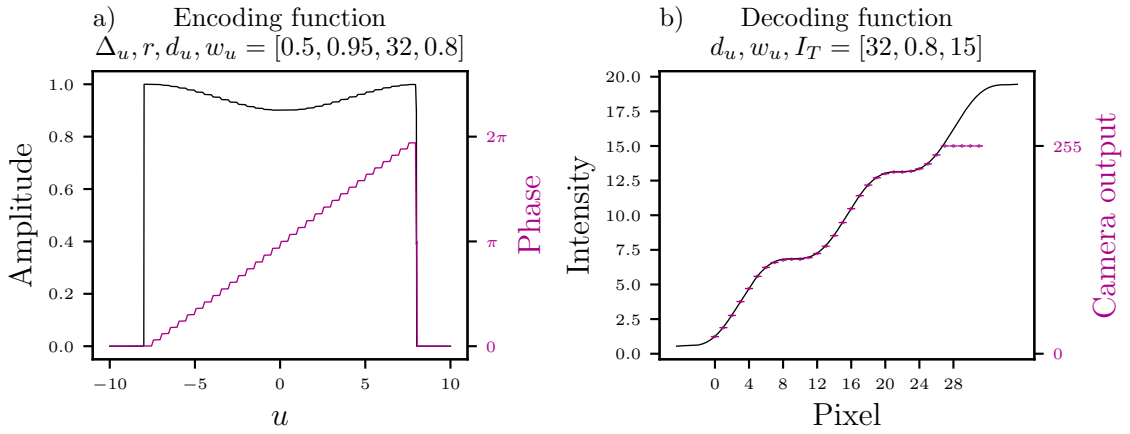


Figure 4.5: **Optical encoding and decoding.** a) Encoding function Equation 4.15 for a 32 pixel input $x_n = 2\pi n,\ n \in \{0, \ldots, 31\}$. $\phi = 0$ for clarity. b) Decoding function Equation 4.18, for arbitrary intensity, demonstrating discretisation and saturation.

$$K_e(u, v, i, j) = \text{trap}\left(\frac{u - \Delta_u\left(i - \frac{d_u - 1}{2}\right)}{\Delta_u w_u}, \frac{1}{w_u} - 1\right) \text{trap}\left(\frac{v - \Delta_v\left(j - \frac{d_v - 1}{2}\right)}{\Delta_v w_v}, \frac{1}{w_v} - 1\right), \quad (4.16)$$

with the trapezoid function given as

$$\text{trap}(x, w) = \begin{cases} 1, & |x| < 0.5 \\ 0, & |x| > w + 0.5 \\ 1 + \frac{1}{w}\left(0.5 - |x|\right), & \text{otherwise.} \end{cases} \quad (4.17)$$

The decoding can be written as

$$D[f(u', v')]_{i,j} = \min\left\{2^b - 1, \left\lfloor \frac{2^b}{I_T} \frac{\iint_{-\infty}^{-\infty} du'dv' |f(u', v')|^2 K_d(u', v', i, j)}{\iint_{-\infty}^{-\infty} du'dv' K_d(u', v', i, j)} \right\rfloor\right\}, \quad (4.18)$$

where we have exposure threshold I_T, rect is the boxcar function, and

$$K_d(u, v, i, j) = \text{rect}\left(\frac{u - \Delta_u\left(i - \frac{d_u - 1}{2}\right)}{\Delta_u w_u}\right) \text{rect}\left(\frac{v - \Delta_v\left(j - \frac{d_v - 1}{2}\right)}{\Delta_v w_v}\right). \quad (4.19)$$

Both encoding and decoding functions are illustrated in Figure 4.5. Note in Figure 4.5.a, the non-unit modulation r causes attenuation which depends on the encoded pixel values. This results in the encoding not being purely phase-affecting as desired.

To describe the propagation, the most general starting point is the full vector wave equation. For our purposes we can make scalar field and steady-state approximations, leading to the Helmholtz equation. From this, the Rayleigh-Sommerfeld equation can be

derived, which, provided we know the field in one specific plane, allows us to calculate the full 3D field as

$$U(x,y,z) = \frac{1}{i\lambda}\iint_{-\infty}^{-\infty} U(x',y',0)\frac{ze^{ikr}}{r^2}\left(1+\frac{i}{kr}\right)dy'dx'. \tag{4.19, RS}$$

The main challenge here is the dependency on $r = \sqrt{z^2+\rho^2}$ with $\rho^2 = (x-x')^2 + (y-y')^2$. We can simplify by making the approximation

$$\begin{aligned} r &= \sqrt{z^2+\rho^2} \\ &= z\sqrt{1+\frac{\rho^2}{z^2}} \\ &\approx z\left(1+\frac{\rho^2}{2z^2}\right) \\ &= z+\frac{\rho^2}{2z}. \end{aligned}$$

We can use this to substitute the r term in the exponential provided that the quadratic and higher binomial terms we drop are less than the period of the exponential, 2π, i.e.

$$\frac{k\rho^4}{8z^3} \ll 2\pi.$$

For the denominator of Equation 4.19, RS, we take the stronger approximation that $r \approx z$ i.e. $z \gg \rho$. This gives the form for the inner term of the integral as

$$\begin{aligned} \frac{ze^{ikr}}{i\lambda r^2}\left(1+\frac{i}{kr}\right) &\approx \frac{ze^{ik(z+\frac{\rho^2}{2z})}}{z^2}\left(\frac{1}{i\lambda}+\frac{i}{i\lambda kz}\right) \\ &\approx e^{ik(z+\frac{\rho^2}{2z})}\left(\frac{1}{i\lambda z}+\frac{1}{2\pi z^2}\right) \\ &\approx \frac{e^{ik(z+\frac{\rho^2}{2z})}}{z}, \end{aligned}$$

where only the $1/i\lambda z$ term is large enough to contribute, giving us the Fresnel diffraction integral,

$$U_{\text{Fresnel}}(x,y,z) = \iint_{-\infty}^{-\infty} U(x',y',0)\frac{e^{ik(z+\frac{\rho^2}{2z})}}{i\lambda z}dy'dx'. \tag{4.20}$$

One can represent this using the Fourier transform, which is useful for numeric calculations,

$$\begin{aligned} U_{\text{Fresnel}}(x,y,z) &= \frac{e^{ikz}}{i\lambda z}e^{i\frac{k}{z}(x^2+y^2)}\iint_{-\infty}^{-\infty} U(x',y',0)e^{i\frac{k}{z}(x'^2+y'^2)}e^{i2\pi(\frac{x}{\lambda z}x'+\frac{y}{\lambda z}y')}dy'dx' \\ &= \frac{e^{ikz}}{i\lambda z}e^{i\frac{k}{z}(x^2+y^2)}\mathcal{F}\left[U(x',y',0)e^{i\frac{k}{z}(x'^2+y'^2)}\right]\left(\frac{x}{\lambda z},\frac{y}{\lambda z}\right). \end{aligned} \tag{4.21}$$

While Equation 4.21 allows us to calculate the propagation efficiently using a single fast Fourier transform (FFT) and motivates the Fourier representation in Equation 4.8, it does restrict the coordinates in the output plane for which we can calculate the field. A more general approach is Rayleigh-Sommerfeld convolution (RSC), which treats Equation 4.19, RS as a convolution with kernel $h_z^{\mathrm{RS}}(x,y) = \frac{ze^{ikr}}{i\lambda r^2}\left(1+\frac{i}{kr}\right)$ where $r=\sqrt{x^2+y^2+z^2}$, leading to the convolutional form

$$\begin{aligned} U_{\mathrm{RSC}}(x,y,z) &= U(x,y,0) * h_z^{\mathrm{RS}}(x,y) \\ &= \mathcal{F}^{-1}\left[\mathcal{F}\left[U(x,y,0)\right](k_x,k_y)\cdot\mathcal{F}\left[h_z^{\mathrm{RS}}(x,y)\right](k_x,k_y)\right](x,y,z). \end{aligned} \tag{4.22}$$

Alternatively, we can view Equation 4.19, RS purely in k-space to derive the angular spectrum method (ASM), where we treat each plane wave component of U separately. Propagating between planes separated by distance z, each plane wave picks up a phase-offset corresponding to the relative optical path length.

Consider we have the plane wave decomposition of monochromatic field U at $z=0$, $U_{z=0}(k_x,k_y) = \mathcal{F}[U(x,y,0)](k_x,k_y)$, then the field at $U(x,y,z)$ can be seen as the integral of each plane component weighted by the phase $\Delta\phi = \vec{k}\cdot\vec{r} = k_x x + k_y y + k_z z$ accumulated as it propagates the distance z,

$$\begin{aligned} U_{\mathrm{ASM}}(x,y,z) &= \iint_{-\infty}^{-\infty} U_{z=0}(k_x,k_y)e^{i\Delta\phi}dk_x dk_y \\ &= \iint_{-\infty}^{-\infty} U_{z=0}(k_x,k_y)e^{iz\sqrt{k^2-k_x^2-k_y^2}}e^{i(k_x x+k_y y)}dk_x dk_y \\ &= \mathcal{F}^{-1}\left[\mathcal{F}\left[U(x,y,0)\right](k_x,k_y)\,e^{iz\sqrt{k^2-k_x^2-k_y^2}}\right](x,y,z). \end{aligned} \tag{4.23}$$

Note that in both ASM and RSC we avoid making approximations, and come straight from the Equation 4.19, RS.

Both Equation 4.22 and Equation 4.23 are equivalent in the form written here—the difference comes when we attempt to calculate them numerically. The discretisation and finite domain in a numeric calculation can introduce aliasing artefacts in our solution, depending on the sampling pitch (d_x, dy), number of samples (N_x, N_y), and propagation distance. We need to consider these values in order to choose the appropriate algorithm. In [ZZJ20], a critical propagation distance

$$z_c = \frac{2N_x {d_x}^2}{\lambda}\sqrt{1-\left(\frac{\lambda}{2d_x}\right)} \tag{4.24}$$

(and similarly for y) is derived based on the sampling theorem. Propagating distances less than this requires the angular spectrum method, while propagating distances greater than this requires the Rayleigh-Sommerfeld convolution. If we choose (d_x, d_y) both less than $\lambda/2$ then RSC is valid for any distance z. Note, that in all cases, when computing

numeric Fourier transforms used by these methods, it is important to zero-pad the input to twice the size in order to satisfy the sampling theorem [Goo96].

All above methods assume propagation in a homogeneous medium. In the case of propagation through thick inhomogeneous media, we need to use split-step methods. In all the coming discussions however, we will assume that any phase objects are thin, i.e. can be represented as multiplicative phase masks in the spatial domain.

One point to note is that these methods do not necessarily conserve energy when used in numeric calculations, which can be seen in RSC by considering that

$$\left|\mathcal{F}\left[h_z^{\mathrm{RS}}(x,y)\right](k_x,k_y)\right| \neq 1$$

i.e. this is no longer an all-pass filter. Any numeric calculation has to sample with a finite pitch, truncating the high-frequency components of the spectrum. Even in ASM which truly is an all-pass filter, energy losses are to be expected due to the zero-padding followed by cropping, needed to prevent aliasing. Methods to correct this are omitted here as absolute intensity correctness is not especially important for our purposes—we only require that relative intensity between pixels is correct, as the simulated camera exposure can be set to correct for global reduction in intensity.

Finally, for the purposes of our model, we can write the propagation as

$$P[f(u,v)](u',v') = \mathcal{F}\left[\mathcal{F}\left[f(u,v)\right](k_u,k_v)H(k_u,k_v)\right][u',v'] \tag{4.25}$$

where H is the transfer function of the optical system and, depending on z and the field sampling, will either use the ASM or RSC.

With this high-fidelity model defined, a similar error propagation analysis as performed in Section 4.3.2 could be conducted. In practice, despite the added complexity in the high-fidelity model, the overall structure of the gradient dependencies will remain largely the same. We therefore leave the specifics of the error calculations to a more competent mathematician—auto-differentiation software.

4.3.4 Simulation details

We implement both models in simulation, using a custom python package with the differentiable computing framework JAX [Bra+18] as its numeric backend. This allows easy GPU parallelisation, and auto-differentiation of the models.

The high-fidelity simulation is performed on a grid with a pitch of 620 nm and spatial extent of around 0.7 mm, light wavelength of 532 nm, propagated using the Rayleigh-

Sommerfeld method. As the field is undersampled for the chosen wavelength, we use the RSC method for propagation with all propagation distances greater than the critical distance. For propagation, we make use of the general chirp Z-transform (CZT) to perform Fourier transforms, as this allows us to choose number of samples and the pitch of the output field's coordinate grid [Hu+20]. The CZT is a generalisation of the discrete Fourier transform, framed in a way which is still efficiently calculable via the FFT algorithm. While we have to be careful of aliasing and ensure valid sampling conditions, the additional flexibility of the CZT lets us calculate a version of the discrete Fourier transform with arbitrary grid spacing.

Lenses placed between the SLMs and the readout camera use the ideal thin lens model and have focal lengths chosen to image the full Fourier spectrum of the previous plane onto the active region of the target plane, with some fixed but random misalignment of order 0-30 µm introduced between each plane to emulate experimental conditions.

The SLMs are modelled as a grid of phase modulating pixels, with a pixel pitch of 6.2 µm and fill factor of 0.9. Each SLM is angled off-axis to ensure that the diffraction orders in the Fourier plane originating from the pixelated grid is diffracted to the edges of the region of interest in the Fourier plane. At each SLM, an additional small, fixed, random wavefront distortion is added, modelled using the first 10 Zernike polynomials, simulating misalignment and non-ideal optical components. The camera readout is in intensity with a fixed exposure and time-varying sensor noise.

Figure 4.6 shows some example optical fields generated by the high-fidelity simulation.

4.4 Learnt PNN optimiser

With the foundations for the reinforcement learning model and the optical PNN defined, we can now formulate the system we use to build our learnt optimiser.

For any given PNN, there is a two-step process we need to follow. Firstly, the optimiser model itself is trained, in a relatively expensive process termed meta-training, which learns the policy π_ϕ, using reinforcement learning and a distribution of tasks which we can train the PNN to solve. This is performed once for a particular PNN.

Secondly, the learnt policy π_ϕ is fixed and used to learn optimal PNN parameters θ for each new task of interest during PNN training. This is done online using the same system which performs the final inference on unlabelled data.

Here we will focus on the meta-training stage, where there are several important components which we need to define. Firstly we need to formalise the environment—the form of the MDP which will represent training a PNN, the reward function, and the range of tasks the agent will be trained on during meta-training. Designing as good reward function is critical, as it generates the signal which guides the agent in its

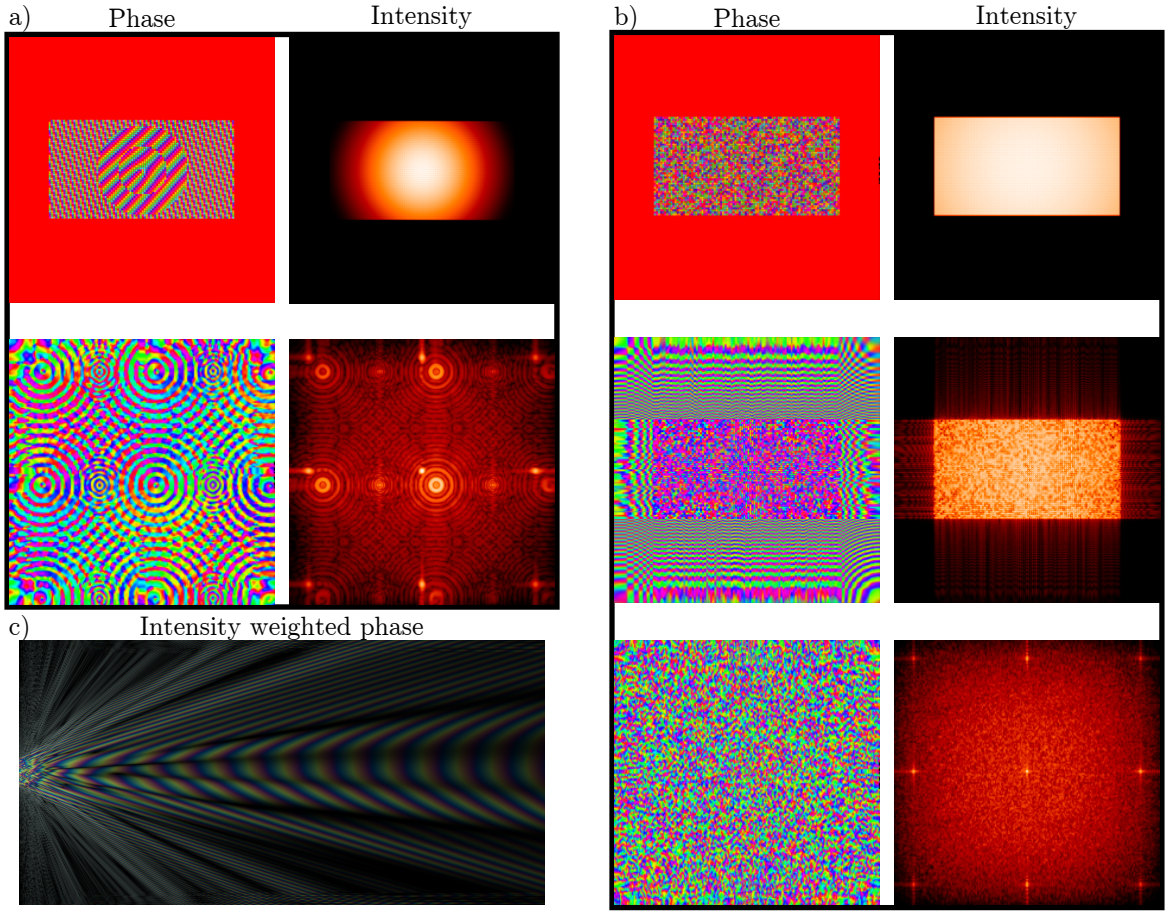


Figure 4.6: **High-fidelity simulation examples.** Various examples of propagated fields using the high-fidelity simulation. a) An SLM with a structured phase hologram, illuminated by a Gaussian beam. The columns show phase and intensity, while the rows show the field in the plane immediately after the SLM, and in the far-field, respectively. b) An SLM with a random hologram, illuminated by a wide Gaussian beam, approximating a plane wave. The rows correspond to planes immediately after the SLM, a short propagation distance after the SLM, and the far-field. c) An intensity-weighted phase image taken as a yz-plane cross-section through the field generated by a random SLM phase mask illuminated by a plane wave, where we observe the various k-space components propagating outwards.

learning, while the set of tasks we use will determine the agent's ability to generalise to new, unseen tasks. Finally, we need to define the agent itself. This will consist of an RL algorithm, and the parameterised neural networks which serve as policy and value estimator.

4.4.1 Environment

When we train a conventional neural network with supervised learning, we are solving a numerical optimisation problem of the form

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=0}^{|\mathcal{D}|} L(f_\theta(x_i), y_i), \tag{4.26}$$

for neural network f with parameters θ, objective (equivalently, cost or loss) function L, and dataset of pairs $\mathcal{D} = \{(x_i, y_i)\}$. The objective L can be seen as a way of comparing the prediction of the network, $f_\theta(x_i)$, with the target y_i.

For most problems which we want to solve with machine learning, we do because the optimisation is not solvable analytically, so we rely on iterative stochastic gradient-based methods such as stochastic gradient descent (SGD), which sample from the dataset and update parameters incrementally, according to some function (which we write here as m) of the local gradient-estimate,

$$\theta \to \theta + m\left(\left.\frac{\partial L}{\partial \theta}\right|_{f_\theta(x),y}\right), \quad (x, y) \sim \mathcal{D}. \tag{4.27}$$

The stochasticity in the optimisation comes from randomly sampling data (x, y) at each iteration. This provides a couple of benefits—firstly, learning can be fast, as we don't need to optimise over the entire dataset at each step, and secondly, we avoid averaging too many different gradients generated by different data samples in each step, which otherwise can prevent learning.

It is usually not possible to guarantee discovery of the global optima (i.e. the value of θ which globally minimises L), but SGD has proven successful in a wide range of applications, and we can condition the loss landscape with regularisers—additional terms in the objective function which constrain the optimisation—to improve the likelihood of finding a good solution.

For our learnt optimiser, we want to replace the function m, which estimates the parameter update $\Delta\theta$, with a learnt policy π_ϕ, but we need to consider what form this should take, and how we can frame our optimisation as a game, or environment, which a reinforcement learning algorithm can interact with.

The first observation we make is that the gradient of the forward system for a particular input and parameter vector,

$$\left.\frac{\partial L}{\partial \theta}\right|_{f_\theta(x),y},$$

depends only on x, y and θ. The update $\Delta\theta$ therefore also only depends on these terms, and the value of the loss[7].

We can consider the tuple $s_t = (x_t, \theta_t, L_t)$ to fully characterise the state of the system at a particular step t in the iterative process, as the update $\Delta\theta_t$ should depend only on these quantities. This allows us to consider a step in our optimisation as a step in a Markov chain, which we can use as the basis of the environment in our RL meta-learning

[7]For convenience, we ignore the many, often necessary, layers of additional features present in modern gradient-based optimisers such as momentum, adaptable learning rates, etc. These will be revisited later.

task.

A couple of points to note here. In gradient-descent, the values we have constructed our state from give the next value of θ deterministically. However, once we have updated θ, we immediately sample a new data point x_{t+1}, meaning that the new overall state $s_{t+1} = (x_{t+1}, \theta_{t+1}, L_{t+1})$ remains stochastic, but only in the x and L components.

Another point to note is that providing the loss value directly in the state definition has some important implications. In GD, the parameter updates for a particular layer depend on the loss implicitly through the error at the output of that layer, Δy, which is derived from L. If we form a state with L directly, then this would allow the RL algorithm to correlate overall performance on a task with the rewards it sees, and should encourage overall performance on the task at hand, rather than constraining predictions to the true gradients. However, it introduces a dependence on the task itself, as the agent would need to develop some understanding of how to map the scalar valued L to an approximate gradient in the parameters, where different tasks with exactly the same L would need potentially very different parameter updates. This makes the update prediction problem highly ill-posed, however we can solve this in two ways. The first is to allow the RL agent to remember past actions and observed losses by expanding the state vector it sees. This has some conceptual similarities to evolutionary strategies such as CMA-ES, which builds estimates of the loss landscape through many samples taken over time in order to learn preferred directions in parameter space. The second approach is to remove the dependence on task, as in SGD, by manually backpropagating the loss to the output of the PNN and using the resulting error on the PNN output, Δy, to create our state, $s = (x, \theta, \Delta y)$. Both approaches have costs and benefits, and we will consider their differences in more detail in the coming sections.

Given these two options for the state, we can define an episode to consist of training the PNN from some randomly initialised θ_0 on a particular task, where a task is defined as a dataset and a loss function $(\mathcal{D}, L)$. At every step t, we sample a pair from the dataset, and use these with the loss function to form the state $s_t = (x_t, \theta_t, L_t)$ (or $s_t = (x_t, \theta_t, \Delta y_t)$). The environment returns this state as the observation, and the agent then provides an action $\Delta\theta$ which we apply to the current parameters. The environment then calculates the new state as the observation for step $t+1$, and the (as yet undefined) reward for step t, and moves to the next step. This process repeats either for a fixed number of steps or until some stopping condition on the loss value is satisfied.

As we would like our RL agent to experience a wide range of different states, so that it may learn a general rule for calculating parameter updates, we want to maximise the diversity in the states. This requires us to train over a range of different tasks, and on

each task, to gather many example states.

Let's imagine a trajectory taken through parameter space under SGD, from a randomly initialised parameter vector, all the way through to a local optima. The types of updates which SGD makes to the parameters will change considerably along this trajectory. For instance, at the start, the step sizes may be larger and the direction we move in may be well-defined by the dataset, however as we get closer to an optima, the gradients will tend to become smaller and noisier, as different influences in the dataset pull the parameters in different directions. It is important that our RL agent experiences all different types of region in the parameter space, so that it is able to generate good parameter updates across the entire length of an episode.

Consider teaching an agent to play chess—with enough experience it may be able to learn to play reasonable openings, however if you then show it an endgame, it will fail to choose good moves, as it hasn't been trained on these sorts of states. The only way for it to learn endgames naturally, is to first get good enough at playing openings and middlegames that it starts to experience endgames. We can cheat the system here a bit, as we may know what common endgames look like and so—exploiting the Markov property— we can show the agent these endgames during its training, without asking it to play the game up to that point.

In our case, we would need to know *a priori* what good parameters look like on a specific task in order to start the agent out at the 'endgame'. Otherwise, we have to rely on the agent getting to these near-completion states naturally. This requires training on many, long episodes, and is a major contributor to the overall cost of meta-training. Truncating episodes early may generate an agent which can optimise a PNN well initially, but whose final performance on a task is capped due to generating bad updates in the later stages of PNN training.

4.4.2 Tasks

We define a task to consists of a dataset of pairs x, y, and a loss function L which evaluates the network predictions $\hat{y}$ against the target 'y's. We also implicitly include an encoding and decoding strategy, which tells us how to map our x to the pixel values of the SLM, and how to interpret the camera pixel intensities as an output label. As discussed, we want to create a set, or distribution, of different tasks which we can sample from during meta-training, to encourage our learnt optimiser to perform well independent of task.

The diversity of this task distribution will dictate the capacity of the optimiser to generalise to new tasks, but also the complexity of training the optimiser in the first place. Too narrow a distribution, with all tasks very similar, and the policy is likely to overfit, learning specific parameters which solve members of this task distribution,

instead of good general-purpose updates. Too broad a distribution, and meta-training will take too long or fail to converge. We need to construct our tasks intelligently such that we can tune this diversity.

During the course of this work, there were many task distributions tested, however the majority of them fell into one of the two categories above—too narrow, leading to overfitting, or too broad, preventing meta-training convergence. Indeed, designing a good task distribution turned out to be harder than expected, and remains one of the main areas for further development in this work.

Considering just image classification tasks, the best performing method of generating a task distribution that we tested turned out to be varying the output decoding for a fixed parent task. In the case of image classification, the decoding is an assignment from intensity macropixels to class labels, which can then be used in the standard categorical cross-entropy loss. We create different tasks by using randomly permuting the map between macropixels and class labels. As our learnt parameters function as a diffraction grating in the PNN, changing the decoding changes the locations on the camera which the PNN must focus light on, in order to classify a given input. Two generated tasks may admit very similar optimal parameter vectors θ in the case where only two macropixels' labels are swapped, or very different optimal parameter vectors for more complex permutations. Once the tasks have been generated, we split them into a training and a test task distribution. Figure 4.7 demonstrates the differences in learnt parameters θ based on different macropixel assignments. We see that the brightest macropixels in the camera output track the permuted labels. While the learnt parameters do vary across the four tasks shows, we also see that there is certain structure common to all solutions. Looking at the similarity of the optimal parameters across different tasks provides a way of quantitatively measuring the diversity of the task distribution.

4.4.3 Reward function

We have discussed in fair detail the importance of getting the reward function right, and the failure modes when we don't. In our case, we want to design a reward which is informative, to ensure that the agent learns quickly, but which doesn't excessively bias the agent, or give rise to unfortunate exploitations.

There are two views on how reward should depend on the current performance on the task (i.e. the loss). The first is that reward should depend only on the *change* in performance, so that the agent isn't penalised for finding itself in a difficult region of the parameter space, only on its ability to move towards better regions. The second is that the reward may also depend on the *absolute* performance, so that the agent is encouraged to explore the parameter space more thoroughly. The scale of the reward is important, and many RL algorithms can be brittle to this. We would like to encourage

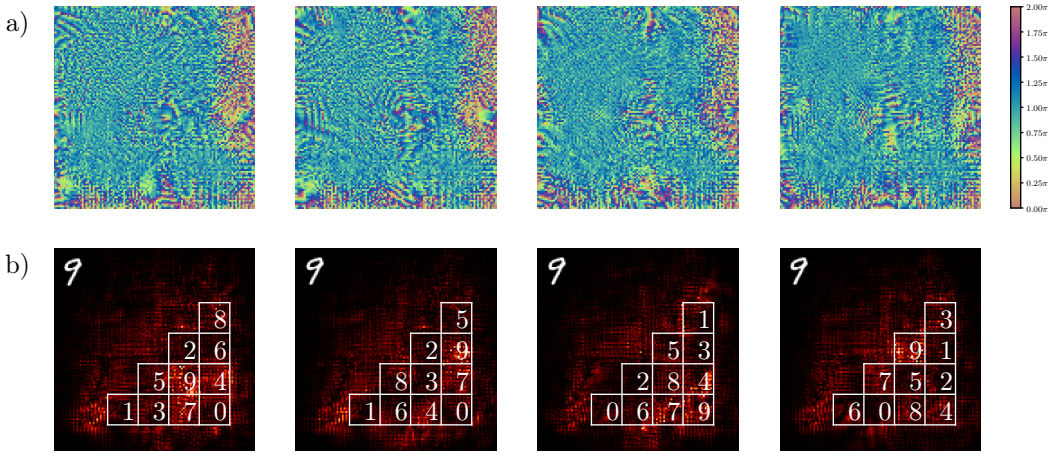


Figure 4.7: **MNIST task distribution by macropixel decoding.** Examples of a) trained parameters θ in range $[0, 2\pi]$ for four different tasks, where each task is MNIST classification with a different assignment from macropixels in b) output intensity to class labels. Inset MNIST image is the input which generates this particular intensity output.

the agent to continue to improve on the loss, across a wide range of loss scales, and across different tasks which may start and finish on very different loss scales.

Given $\ell = L(\hat{y}, y)$ and $\ell' = L(\hat{y}', y)$, where $\hat{y}'$ is the PNN output after the parameter update derived from the previous reward, some of the more obvious reward forms include:

- Absolute loss value, i.e. $r \propto -\ell'$. Encourages decreasing loss, but is overly sensitive to the specific task. Large range of scales makes learning difficult.
- Change in loss value, i.e. $r \propto \ell - \ell'$. Less sensitive to task specifics, but the range of scales is still problematic.
- Ratio in loss value, i.e. $r \propto \ell/\ell' - 1$. Encourages relative improvement, but is sensitive to the initial loss value.
- Logarithmic change in loss value, i.e. $r \propto \log(\ell) - \log(\ell')$
- Alignment with target change, i.e. $r \propto \text{CSE}\left(\frac{\partial \ell}{\partial \hat{y}}, (\hat{y}' - \hat{y})\right)$. Encourages alignment with the true gradient, but risks biasing the system away from the true goal, loss minimisation.

From empirical data gathered over a range of these rewards, we settled on a balanced

approach with

$$R_\alpha(L, \hat{y}', \hat{y}, y) = \alpha(\log(L(\hat{y}, y)) - \log(L(\hat{y}', y))) + (1 - \alpha)\text{CSE}\left(\frac{\partial L(\hat{y}, y)}{\partial \hat{y}}, (\hat{y}' - \hat{y})\right), \tag{4.28}$$

depending only on the changes in loss and the resulting change in the PNN output, with α interpolating between the two extremes. We find that $\alpha = 0.7$ performed well, beating $\alpha = 0$ or $\alpha = 1$ in our tests, however due to the cost of running meta-training this value hasn't been fully optimised.

While the reward function should technically depend only on the values from the current time step (Markov property), more complex schemes, such as basing reward off the moving average of the change in loss over episode steps, have been found to perform fairly well in small tests, but have not been fully explored. Tests where the reward is scaled by the step size have also shown promise, as this penalises the network more for taking large steps in areas where it is unsure, but encourages large steps in regions where the network is confident.

4.4.4 Choosing an RL algorithm

The development of this project was very much an iterative process, with several algorithms and architectures tested. To provide some context for the range of different approaches discussed in the coming results sections, we give a brief chronological overview of the evolution of our learnt optimiser, detailing the reasons behind the decisions made, the experiments performed, and the challenges with each approach.

When choosing an initial algorithm, we were immediately able to dismiss the use of on-policy approaches, as making forward evaluations of the PNN carries a certain cost in both simulation and hardware, making meta-training sample efficiency a priority.

We chose to begin with DDPG [Lil+19]. This is an established actor-critic algorithm which uses off-policy data gathered with a noisy exploration policy, and is relatively simple to implement, with several reference implementations available. This was used to generate the toy example in Figure 4.1.b.

However, through our tests it soon became apparent that deterministic policies were a poor choice in our application, as they rely heavily on having a good exploration strategy. DDPG also suffers from a certain brittleness to hyperparameters, and proved particularly difficult to tune in our use case. This led us to search for an alternate algorithm, which would provide good state-space exploration, while prioritising sample efficiency.

We settled on soft actor-critic (SAC) [Haa+18], which allows the use of stochastic policies, has inbuilt exploration, good robustness to hyperparameters, and is widely

used in the community, meaning there are several stable implementations available.

It's worth mentioning that at this point, we also strongly considered maximum *a posteriori* policy optimisation (MPO) [Abd+18b; Abd+18a], which provides inbuilt exploration through KL regularisation, state-of-the-art sample efficiency, and has been demonstrated on a range of complex control tasks [Deg+22]. Ultimately, it was the availability of stable, robust implementations of SAC and popularity within the community which were the main deciding factors.

Both SAC and MPO support stochastic policies, where instead of predicting optimal actions, we calculate a probability distribution over actions, in turn letting us quantify the model's uncertainty about a particular update. We can write the policy as a deep neural network π_ϕ with trainable parameters ϕ, which maps from our state to a probability distribution over possible parameter updates,

$$\pi(\Delta\theta \,|\, s), \tag{4.29}$$

where state s is one of the two forms given in Section 4.4.1. By sampling from the distribution during meta-training, we can encourage exploration, as two identical states will produce different updates. During inference, we can dispense with exploration and choose the maximum likelihood update, which gives a single update step in our optimisation as

$$\theta \to \theta + \alpha \operatorname*{argmax}_{\Delta\theta} \pi_\phi(\Delta\theta \,|\, s = (x, \theta, \Delta y)), \quad x, y \sim \mathcal{D}. \tag{4.30}$$

We worked with SAC for several months, and initial results training the PNN in simulation were promising, however subsequent efforts to replicate these results proved challenging. Significant efforts were made to analyse SAC, and attempt to improve its stability, however all indicators pointed to the method being extremely sensitive to random initialisations, at least on our problem. While there remain avenues of investigation which could help improve SAC's performance, the sample efficiency limits the rate at which we could iterate and test improvements. As a result, we made the decision to change tack and move to MPO, for its improved sample efficiency and robustness to hyperparameters, where we used the reference implementation in [Hof+20].

In the coming sections, we present the results from SAC, focussing primarily on learning to optimise a single PNN. We subsequently discuss the work with MPO, which mainly concerns the extension of our meta-learning approach to training deep networks.

4.4.5 Network architectures

SAC works by training three deep neural network function approximators, where two of them, a state-value estimate and a state-action value estimate, act as auxiliary

estimators used to train the policy. As with any supervised deep learning, the structure and parameterisation of these networks will determine the types of functions we can learn, and in turn, the upper limit of performance on the learning task. While we expect that the networks will need to learn some sort of representation of the dynamics of the PNN in order to make good estimates, we want to avoid training a model which perfectly reproduces the PNN, as in digital twin methods, due to the expense involved. Even if we wanted to, we have already seen that training a black-box digital twin from data can be a hard task due to the volume and diversity of data required to capture the full dynamics of a system and improve the conditioning of the inverse problem.

We therefore have to balance the overall performance of a network architecture with its size and computational costs. It is desirable for the policy to be fast to evaluate, to improve inference time, and as small as possible while still retaining the capacity to map inputs to pseudo-gradients. Based on the theory of direct feedback alignment (DFA), we know that a random map from the output error to the parameter space will with high likelihood preserve the relative orientations of the backprojected errors, and that networks can accommodate these—technically incorrect—gradients to learn good solutions. Further, assuming that the parameter and output spaces are of similar dimension, we know from Johnson-Lindenstrauss (Section 2.3.2) that the random map can have lower rank and still maintain this orientation preserving property.

Unlike DFA, we use a trainable nonlinear neural network rather than a fixed random matrix, allowing us to train the backprojection based on the observed reward and further improving the alignment with the true gradients. As we would like to be able to work with high-dimensional spaces, we need to build the networks using layers which can scale well. One option is to use convolutional layers, however we find empirically that on this task and PNN they don't perform especially well. Instead, we opt for dense layers, however we represent these using low-rank matrix multiplications to reduce the number of trainable parameters and improve computational efficiency. Each layer is represented by two rectangular matrices, $\mathbb{R}^{k\times d}$ and $\mathbb{R}^{d/2\times k}$, where d is the dimension of the input, $k \ll d$ is the rank of the map, and the output has half the number of elements. This form gives a computational complexity of $\mathrm{O}(3dk - k - d/2)$ for each layer, improving over the full rank matrix complexity $\mathrm{O}(d^2 - d/2)$.

Our SAC algorithm networks are shown in Figure 4.8. The policy is a simple U-net approach, which performs a tunable low-rank matrix multiplication, as described above, at each stage. The value networks are also kept intentionally simple, with both reusing the first half of policy the U-net, denoted H. We process the output of the policy through a final long short-term memory (LSTM) block, which provides fading memory of previous updates. This is a feature common to many state-of-the-art gradient based optimisers, such as Adam [KB14], which implement time-dependent features, such as momentum, by calculating the running average of the calculated gradients. In contrast

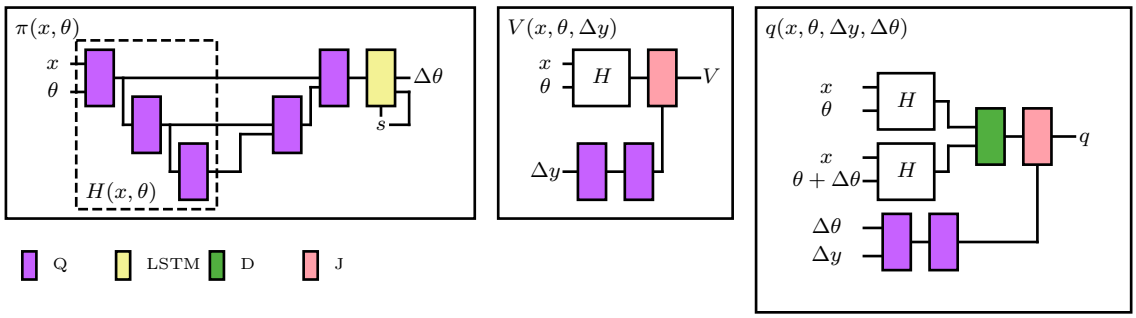


Figure 4.8: **SAC RL network architectures.** Architecture of the policy, state-value, and state-action value networks π, V and q used in the SAC algorithm. The H blocks used in V and q refer to the head of the policy, outlined in the left panel. See text for details of accompanying sub-blocks.

to the usual method of applying these recurrent layers, each forward pass through our network performs only one step of the LSTM[8], and the hidden state is reset at the start of each episode. While this episodic memory does break the Markov assumption, we find empirically that it improves performance of the agent.

To ensure stable policy outputs, we use a logarithmic nonlinearity on the output

$$\text{sign}(x)\log(1+|x|). \tag{4.31}$$

This helps restrict predictions to lie in the region around the origin, such that we avoid making excessively large updates to the PNN parameters in a single step.

The details of the sub-blocks in Figure 4.8 are given as follows:

- $\text{Q}(h_i) = \text{LReLu}(W_l \cdot W_r \cdot \text{concat}(h_1, \ldots, h_i))$

  A low-rank dense matrix operation. Concatenates all inputs, multiplies by a $\mathbb{R}^{k\times d}$ matrix W_r, then a $\mathbb{R}^{d/2\times k}$ matrix W_l. For all experiments we choose $k = 1024$ and d is the dimension of the inputs. The output is passed through a leaky ReLu nonlinearity, with leakage-rate 0.01.

- $\text{LSTM}(h, s)$

  Applies the LSTM operation on the output of the U-net, with LSTM state s[9]. Models the parameter distribution as a multivariate Gaussian, producing a mean and covariance matrix. The covariance matrix is constructed through the Cholesky decomposition. Log nonlinearity Equation 4.31 is used to bias the output towards the origin.

[8]Due to the computational scaling here, we only backpropagate through time a maximum of 3 steps when training the network. This is an area where we would like to improve, but it currently adds significant overhead to the training process.

[9]Different to the environment state.

- $\mathrm{J}(h_1, h_2) = W_2 \cdot \mathrm{LReLu}(W_1 \cdot \mathrm{concat}(h_1, h_2))$

  Two dense layers: $\mathbb{R}^{k \times d}$ matrix W_1, then LReLu nonlinearity, then $\mathbb{R}^{1 \times k}$ matrix W_2, with no nonlinearity on the output.

- $\mathrm{D}(h_1, h_2) = \mathrm{concat}(h_1, h_2, |h_1|^2 - |h_2|^2)$

4.4.6 End-to-end learnt optimiser algorithm

With environment specified, algorithm chosen and networks defined, we can now outline the high-level algorithm for both meta-training and PNN training. We start with meta-training, which is outlined in Algorithm 1.

Note that in stochastic policies, the update is $\Delta\theta \sim \pi_\phi(\Delta\theta \,|\, s)$ and the likelihood $\pi_\phi(s)$ is also stored in the replay buffer, such that the experience can be used later in off policy updates via importance sampling. At evaluation time, the stochastic policy is converted to a deterministic policy via maximum likelihood, i.e. $\pi_\phi(s) = \mathrm{argmax}_{\Delta\theta}\, \pi_\phi(\Delta\theta \,|\, s)$.

With a trained policy network, we can now apply the learnt optimiser to train new tasks on the PNN in a process outlined in Algorithm 2.

Algorithm 1 Learnt optimiser meta-training.

Initialise task distribution $\mathcal{T}$, agent networks π_ϕ, V_{ϕ_V} and q_{ϕ_q}, forward model f, reward function R, experience buffer $\mathcal{B}$.
for episode **do**
  Sample dataset and loss function $(\mathcal{D}, L) \sim \mathcal{T}$.
  Initialise parameters θ_0.
  for $t \leftarrow 0$ to $t_{\max}$ **do**
    Sample $(x, y) \sim \mathcal{D}$.
    Evaluate forward model $\hat{y} = f_{\theta_t}(x)$.
    if end_condition$(t, L(\hat{y}, y))$ **then**
      break
    end if
    Package state $s = \left(\frac{\partial L(\hat{y}, y)}{\partial \hat{y}}, x, \theta_t\right)$.
    Sample parameter update $\Delta\theta \sim \pi_\phi(\Delta\theta \mid s)$.
    $\theta_{t+1} = \theta' = \theta_t + \mathrm{clip}_{[-\alpha, \alpha]}(\Delta\theta)$.
    $\hat{y}' = f_{\theta'}(x)$.
    Calculate reward $r = R(L, \hat{y}', \hat{y}, y)$.
    Package new state $s' = \left(\frac{\partial L(\hat{y}', y)}{\partial \hat{y}'}, x, \theta'\right)$.
    Store experience $(s, \Delta\theta, r, s', \pi_\phi(\Delta\theta \mid s))$ in buffer $\mathcal{B}$.
  end for
  Update ϕ, ϕ_V and ϕ_q according to RL algorithm, sampling from $\mathcal{B}$.
end for

Algorithm 2 Learnt optimiser deployment.

Load learnt policy π_ϕ, task (dataset and loss function) $(\mathcal{D}, L)$.
Initialise parameters θ_0.
for $t \leftarrow 0$ to $t_{\max}$ **do**
  Sample batch $\{(x_j, y_j)\}_{j=1}^{B} \sim \mathcal{D}$.
  Evaluate forward model $\hat{y}_j = f_{\theta_t}(x_j)$.
  if end_condition$(t, L(\hat{y}_j, y_j))$ **then**
    break
  end if
end for

One feature of SGD is that updating network parameters based on the gradient obtained from a single data sample tends to lead to very poor performance. On the other hand, aggregating the gradients from *all* samples in the dataset before updating the parameters also performs poorly. In the former case, the gradient is too noisy, and doesn't correlate well with the true overall task's gradient. In the latter, we see that the gradients tend to average out, and it becomes very easy to get stuck in the parameter space.

The trick to get around this problem is to optimise on mini-batches of data, where we update parameters with the aggregated gradients produced by a small subset of the data. This helps to filter noise from the gradient estimates by averaging over samples, while still giving a good local gradient estimate for a small sample of the data distribution.

The same challenge applies in our learnt optimiser, at two levels. The first is during the meta-training. Here, we wish to force the task independence property in the optimiser, i.e. to be able to generalise to new tasks. This requires us to average over some large distribution of tasks, to learn the more fundamental ability to propagate errors through the PNN, as opposed to the specific challenge of solving one particular task with the PNN. By selecting a different task in each episode, we generate data from a wide range of tasks, exploring a wider region of the state space. Storing this experience in a replay buffer, which is randomly sampled in batches at each update of the agent's networks' parameters, helps to decorrelate the data and improve the stability of the learning process.

The second instance where batching is important is in the individual episodes, during the PNN training process. Here, we use the agent's policy to estimate updates to the PNN parameters on a particular task. As this is exactly the same as the normal SGD picture, we should allow for batching of data to reduce the noise in the update estimates. In practice, we can achieve this by passing a batch of data through our PNN, evaluating the policy on each sample, and then averaging the updates to get the final update. If we use this average update to generate a single reward for the batch, with this reward saved in the experience tuple for each sample, then this effectively makes

the environment behave stochastically, from the agent's point of view. We have several options for how to proceed, choosing at what point to average updates and rewards over the batch.

For each sample, we need to store a tuple of form (s_j, a, r, s') in the replay buffer. For the action a, we can choose either the sample-specific estimate $\Delta\theta_j$, or the averaged $\Delta\theta$. The reward and next state can then be calculated as $R(L, p, q, y_j)$ and $\left(\frac{\partial L(p,y_j)}{\partial p}, x_j, a\right)$ respectively, for $p \in \{\hat{y}'_j, \hat{y}'\}$ and $q \in \{\hat{y}_j, \hat{y}\}$.

The case of $a = \Delta\theta_j$, $p = \hat{y}'_j$ and $q = \hat{y}_j$ recovers the unbatched training of Algorithm 1. Any other case is effectively acting under a different policy, which can be accounted for through importance sampling, provided we can calculate the likelihood of the action under the new 'policy'. For simplicity and to ensure a well-behaved environment, we choose to use $p = \hat{y}'_j$ and $q = \hat{y}_j$, with the batching simply affecting the parameters used in next episode step. This batched meta-training algorithm is detailed in Algorithm 3.

Algorithm 3 Learnt optimiser batched meta-training.

Initialise task distribution $\mathcal{T}$, agent networks π_ϕ, V_{ϕ_V} and q_{ϕ_q}, forward model f, reward function R, experience buffer $\mathcal{B}$.
for episode **do**
  Sample dataset and loss function $(\mathcal{D}, L) \sim \mathcal{T}$.
  Initialise parameters θ_0.
  for $t \leftarrow 0$ to $t_{\max}$ **do**
    Sample batch $\{(x_j, y_j)\}_{j=1}^{B} \sim \mathcal{D}$.
    Evaluate forward model $\hat{y}_j = f_{\theta_t}(x_j)$.
    if end_condition$(t, L(\hat{y}_j, y_j))$ **then**
      break
    end if
    Package states $s_j = \left(\frac{\partial L(\hat{y}_j, y_j)}{\partial \hat{y}_j}, x_j, \theta_t\right)$.
    Sample parameter updates $\Delta\theta_j \sim \pi_\phi(\Delta\theta \mid s_j)$.
    $\theta'_j = \theta_t + \text{clip}_{[-\alpha,\alpha]}(\Delta\theta_j)$.
    $\theta_{t+1} = \theta_t + \text{clip}_{[-\alpha,\alpha]}\left(\sum_j \Delta\theta'_j / B\right)$.
    $\hat{y}'_j = f_{\theta'_j}(x_j)$.
    Calculate rewards $r_j = R(L, \hat{y}'_j, \hat{y}_j, y_j)$.
    Package new states $s'_j = \left(\frac{\partial L(\hat{y}'_j, y_j)}{\partial \hat{y}'_j}, x_j, \theta'_j\right)$.
    Store experiences $(s_j, \Delta\theta_j, r_j, s'_j, \pi_\phi(\Delta\theta \mid s_j))$ for each sample in buffer $\mathcal{B}$.
  end for
  Update ϕ, ϕ_V and ϕ_q according to RL algorithm, sampling from $\mathcal{B}$.
end for

4.4.7 SAC RESULTS

We meta-train a policy using a set of 30 MNIST tasks for a total of 100 episodes, where each task is generated according to the task distribution given in Section 4.4.2. The meta-training was repeated 10 times with different random number generator seeds, which influenced the initialisations for the RL agent's networks' parameters, the

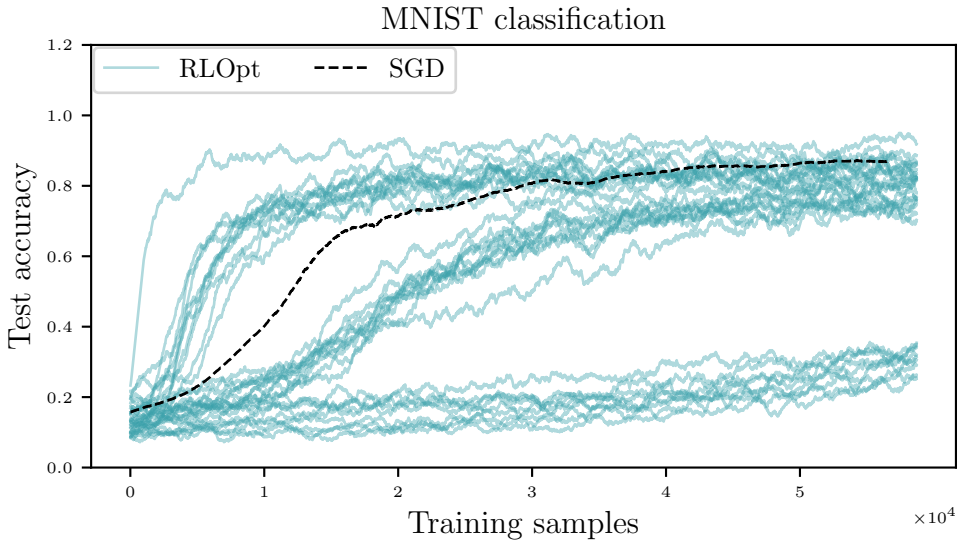


Figure 4.9: **SAC learnt optimiser test accuracy.** Accuracy by number of training samples over 30 test tasks, for gradient descent on the high-fidelity model (SGD), and for the learnt optimiser (RLOpt), using the best performing learnt π_ϕ of the 10 meta-trained models. The tasks are generated according to Section 4.4.2, based on the MNIST classification dataset [LCB10], with 112×112 pixels in the SLMs and camera. For clarity, we omit the full set of 30 traces for SGD, as the distribution is approximately normally distributed over time. Model is SAC, meta-trained in simulation with 1 layer PNN, high-fidelity model.

PNN parameter initialisation, the task and batch sampling in a given episode, and the sampling from the replay buffer used to update the agent's parameters. The reward function used was Equation 4.28 with $\alpha = 0.7$. The forward model which we train the agent to optimise is given by the PNN in Section 4.3.1 and is implemented using the high-fidelity simulation.

Figure 4.9 shows the performance of the system on 30 generated tasks, unseen in meta-training. We also show, in black, the average[10] test accuracy on the same set of test tasks achieved by training the PNN with regular SGD. This is made possible by our use of the high-fidelity simulation, which, while the RL optimiser still only has access to forward passes through the system, technically can be auto-differentiated and optimised, giving us a baseline to compare against. Each curve corresponds to a single procedurally generated task, i.e. a single column in Figure 4.7, where the example optimal parameters in Figure 4.7.a show the final learnt solutions.

There are several features of note in this data. The most prominent is the variation in performance of the learnt optimiser across the different tasks, which appears multimodal. There are three main cases we can identify here. The first consists of traces which fail to learn in the time given, although even for these, there is a general upward trend in accuracy towards the end of training, indicating that we could likely continue

[10]The distribution is approximately normally distributed, so we omit the full traces for clarity.

training for longer to improve performance. There is a second set of tasks which train in reasonable time, underperforming the SGD average however still achieving good accuracy in the 70-90% range. The final set of tasks are those which initially outperform the SGD average, increasing rapidly in accuracy and then plateauing. While some of these are subsequently overtaken by the SGD average, others continue to improve and achieve final accuracy upward of 90%. These cases arise from the fact that we do not explicitly set the step size for the updates generated by the learnt optimiser, and that there are some tasks where the agent is confident in its estimates and quickly takes large steps in the parameter space.

This variability arises from two main sources. The first is the random initialisation of the PNN parameters, as the learnt optimiser is not necessarily invariant to the starting point in parameter space. The parameter update estimates are conditioned on the current parameters, meaning that, depending on the experiences seen during meta-training, the agent's policy for estimating pseudo-gradients may perform better or worse in different regions of the parameter space. The second is that, due to the randomised task generation, there will be some decodings which perform better for classification than others, and therefore different random tasks will have different levels of difficulty. Harder tasks may still have good solutions, but there may be fewer of these well performing local optima in the parameter space, and they may be harder to reach from a typical initialisation. SGD tends to be more robust to this than the learnt optimiser, which we hypothesise comes from the precise numeric gradient giving better chances of finding rarer, high value solutions.

Despite the somewhat limited task distribution, this experiment demonstrates the potential of the learnt optimiser to generalise to a range of related, previously unseen classification tasks. We also note that we are optimising in a relatively high dimensional space, with 112^2 tunable macropixels on the parameter SLM, scales where zeroth order methods would struggle.

All simulated experiments were performed on a workstation (WS1) with a single Nvidia RTX 3090 GPU, an Intel i9-10940X CPU, and 256 GB RAM. Another system (WS2) was used to accelerate some experiments, and consisted of a single Nvidia RTX 2080 GPU and 64 GB RAM. Where the two systems were used together, WS1 acted as experiment controller and ran the ML workload, while the optical simulations were distributed to WS2's GPU, with communication over gigabit Ethernet.

The high fidelity optical simulations are performed using a custom library based on JAX [Bra+18], the SAC implementation is available in SBX [Raf+21], and the custom policy was built using Equinox [KG21].

The data shown in Figure 4.9 consisted of 10 meta-training runs, each with 100 episodes, where each episode ran for a total of 10 epochs. For the MNIST dataset, 10 epochs contributes 600,000 total steps. Each meta-training run had access to sampled data from earlier runs through a persistent replay buffer, meaning later experiments could benefit from the experience of earlier ones. Each meta-training run of 100 episodes, with 600,000 steps per episode, took approximately 20 hours, and the 10 runs were performed over the course of two weeks.

Ideally, and in most RL applications, one would want to train for orders of magnitude more episodes than we did here, and for improved exploration of state space, also increase the time step cut-off in each episode. We are however resource limited, and the computational cost of a single meta-training run is already high. This is currently the biggest limitation in our work, significantly hampering our ability to iterate on the model and improve performance.

As mentioned earlier, the performance of the agent's policy can depend on the region of parameter space it finds itself in, something we would like to discourage. In gradient descent, we can always calculate the gradient regardless of our current position. If the learnt optimiser's accuracy or precision varies with absolute position in parameter space, then we will get inconsistent results depending on our PNN initialisation. The same would apply also to the input data—we don't want our ability to estimate gradients to depend on the specific input data. The simplest method to regularise this would be to simply meta-train with as much diversity in parameter and input vectors as possible. We do our best to achieve this by sampling different tasks, but for high-dimensional spaces it is not tractable to achieve any meaningful, uniform coverage of the full space. The only alternative open to us is to structure our networks and loss functions to encourage this invariance, however this is also difficult, requiring a good understanding of the PNN dynamics. It is important to note that the updates we predict *should* depend on the inputs and parameters, it is the quality of the alignment with the true gradient which we want to ensure is consistent for different inputs and parameters.

4.5 Learnt optimisation for deep networks

While our learnt optimiser is able to perform relatively well on a small physical system, the method currently does not scale well to large systems, as it requires learning a complex mapping to pseudo-gradients for each parameter. A single nonlinear dense layer can in theory act as a universal function approximator given enough nodes[11], however, in practice this is never done—in modern machine learning we always stack layers sequentially, allowing for rich sets of hierarchical representations with fewer nodes.

[11] Indeed, this is the setup in reservoir computing.

Similarly, we can scale our existing PNN in two ways, either by increasing the resolution of our SLMs and camera, or by adding additional parameter SLMs in series, similar to multi-plane light conversion (MPLC) schemes for mode shaping [ZF23]. The former mirrors the scaling of a single layer, and the latter the scaling of a deep network. However, in both cases, our learnt optimiser's complexity and meta-training time scales with the total number of parameters, making neither approach feasible beyond a certain limit. The only realistic way to scale, outside of local learning rules, is with some form of error propagation, which would let us train deep networks independent of the number of layers.

To attempt to build a system which is compatible with this 'pseudo-backpropagation' idea, we move away from all-optical PNNs, towards a new, hybrid electro-optic model, composed of individual PNNs controlled by a digital computer. Despite the considerable interest in all-optical deep PNNs, a hybrid system will have several advantages for our purposes, and balances the strengths of conventional computing, in manipulating data and coordinating hardware, with optical computing, acting as an accelerator for otherwise computation and memory intensive tasks, such as matrix multiplication. We will refer to the individual physical systems as physical neural layers (PNLs), to distinguish that they are now only components of a larger network.

The main advantage of this approach is that we can combine it with our learnt optimiser to get a trainable deep network, with good scaling properties. While we cannot train the individual systems independent of one another, we can understand the benefits of this mode by returning to the original motivation for our learnt optimiser. Thus far, our learnt optimiser calculates parameter updates using a model trained to predict vectors which are aligned with the gradient of the PNN output with respect to the parameters. However, if we can estimate this gradient, then we should also be able to learn approximates to the gradient of the output with respect to the *inputs*. We already calculated this function for our low-fidelity model in Equation 4.13b, and conceptually it makes sense that if we can approximate $\partial f/\partial \theta$ then we could also learn to approximate $\partial f/\partial x$. This would allow us to approximately backpropagate the error on the PNN's output back to its input, which could in turn be backpropagated through any layers preceding our PNN. These layers could, in principle, either be evaluated on a computer, with exact backpropagation as usual, or other PNNs with matching learnt error propagation models, giving us complete freedom to interleave conventional neural network operations, implemented *in silico*, with optical layers.

The system we propose here changes the function of our learnt model slightly, from acting as an optimiser, to a more general error propagation model, and this does introduce some extra considerations. Firstly, the main advantage of the scheme is that the complexity of the learnt model depends only on the complexity of an individual PNL, not on the overall machine learning mode, which is key for scaling. If the same

PNL is reused multiple times in the network, then the same learnt model can be used for each PNL, meaning we still only have to perform meta-training once, for arbitrarily[12] complex deep networks. We will therefore continue to work with the same optical PNL defined in Section 4.3.1.

The main disadvantage of the method is the same as in any DNNs—ensuring the stability of error signals as they are propagated backwards. In conventional deep learning, vanishing and exploding gradients are a common issue, which arise due to the fact that each layer in a network can either amplify or reduce the magnitude of its input. Normalisation in the forward direction can keep the signal stable during evaluation, but, as we have already discussed in Section 2.3.1, guarantees on the stability of a forward model don't translate to guarantees on its gradient, required for training. Solutions such as residual connections and weight initialisation strategies[13] can help to alleviate this problem, but careful architectural design is still required to ensure stability.

The problem of stable error propagation is exacerbated in our learnt optimiser. We want to do meta-training independent of the structure of the deep network we plan to use for inference, meaning we have to train using a single PNL, where our reward function can be based either on loss, or on alignment with the true gradient. In the first case, the agent is free to predict any update it likes, provided the loss decreases, with no incentive to ensure that the error on the input will be well-suited for further backpropagation. The second case, where we further restrict the agent to align with the true gradients, is likely to be better for deep backpropagation, but this restriction is hard to enforce. Even if we get 'reasonable' gradient alignment, such that a single layer can solve a simple task, deep networks with more complex tasks are likely to be more sensitive to the quality of the error signal. We need to either condition the meta-training process through our reward to align with the true gradient, manually stabilise the error signal through, for instance, enforcing small step sizes, restrict the depth of our deep networks, or use a combination of all three.

In order to accommodate this new mode of operation, we will need to make some modifications to the form of the networks we use and the reward function. Based on the stability and sample efficiency issues we faced with SAC, and the discussion in Section 4.4.4, we will also now switch to using the MPO algorithm. The main reason for changing to MPO is the potential improvement in sample efficiency during meta-training, in turn allowing us to iterate development faster, and, given that we already need to modify our policy and value estimator neural networks, we will update our training method to match.

[12] Theoretically. In practice, we will see that there remain challenges in scaling the depth of the network.

[13] These aim to ensure that the eigenvalues of each layer's linearisation, at initialisation time, are close to 1.

4.5.1 Network architecture

The first step to updating our optimiser to work with deep networks is to modify the network architecture. While the approach given in Section 4.4.5 performed relatively well, its design was somewhat arbitrary. We predominantly used dense layers and introduced low-rank maps to make the calculations tractable for the high-dimensional state and action spaces we work with. Given the importance of adhering closer to the true gradient of the system in a learnt error propagation model, which we wish to use for optimisation of deep networks, we expect that we will need networks which are better able to model the true physics of the PNN, while remaining flexible enough to account for differences between our ideal model and the real system, and also remaining efficient to execute. It seems natural that any extra information we can provide the RL agent about the structure of the physical system will improve the learning efficiency—we therefore need to investigate ways of adding additional physical priors in our meta-training process, derived from the form of the PNN in Section 4.3.1.

In conventional neural networks, these sorts of priors are often embedded in the structure of the networks. For instance, structures such as convolutional or recurrent layers allow us to reduce network complexity by reusing weights. The performance of these types of layers stems from some symmetry of the problem being solved, such as translation invariance in images, or shift invariance in time series. Similarly, one of the main motivations for this learnt optimiser was to blur the lines between methods such as zeroth order algorithms, which have no information about the system being optimised, and the model-based, digital twin methods, which, if anything, suffer due to their overreliance on a mismatched model leading to an inability to adapt to real-world conditions. We would like to be able to exploit symmetries present in our physical system, and we can make some educated guesses about the types of transformations which would be useful to embed in the network, based on the theoretical model of the optical system.

For instance, consider the approximate model of both the forward and backward, error propagation passes of our model, given in Equations 4.8, 4.13a, and 4.13b. Here, information is encoded in the phase of spatially varying waves, making the complex numbers a natural domain for studying the system. As a first step, we can therefore choose to use complex-valued neural networks.

While we don't know the exact model of the PNN, we expect that the function our policy should learn would look something like Equation 4.13a, and so we can construct the network using this as a guide. One way we do this is through the use of Fourier layers [Li+21], and element-wise trainable phase-masks, which we embed in the network blocks. While it is technically possible for a dense layer to replicate the Fourier transform by learning a representation of the discrete Fourier transform matrix, in practice it

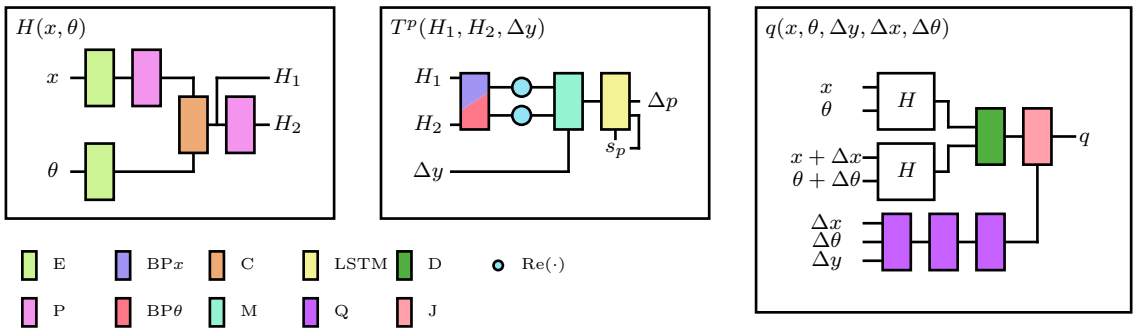


Figure 4.10: **MPO RL network architectures.** Architecture of the policy and state-action value networks π and q used in the MPO algorithm. Our policy is split into a head network, H, and two tail networks, T^θ and T^x. See text for details of accompanying sub-blocks.

is more efficient to manually provide the network with the ability to calculate this, reducing the load on the training process. This gives a way for the model to estimate the free space propagation and associated coordinate transforms. By implementing the Fourier layers using the CZT[14], we expose free parameters associated with the sampling of the discretised integral transform, providing capacity to model optical misalignment in the system.

In both Figure 4.10 and the following definitions, we use p as a placeholder for x or θ, as we reuse elements of the network structures, most notably in the construction of the tail networks $T^{(x)}$ and $T^{(\theta)}$. Capitalised letters and their matching coloured blocks refer to particular sub-neural networks. Two blocks of the same type do not share weights, only their structure. The large blocks, which are never coloured, do reuse weights wherever they are referenced. These include the policy head H, policy tails $T^{(x)}$ and $T^{(\theta)}$, and value estimation network q. For instance, the two E blocks in H will have different weights, but all instances of H will share one common set of weights. This is to ensure that the policy and value networks can share the same learnt representations of the state, reducing the number of parameters which need to be learned.

Looking closely at the structure of H, we see that it mirrors the form of Equation 4.8, although we do not constrain it to be exactly the same. Similarly, the policy tails $T^{(x)}$ and $T^{(\theta)}$ are designed to mirror the form of Equation 4.13, and the value network q is very loosely designed around the structure of our reward function. At each layer in the networks, we allow flexibility in the representations learnt, but provide guidance through the structure of operations used. Low-rank operations remain critical for maintaining efficiency in estimating updates, allowing us to work with high-dimensional state and action spaces.

[14]Via Bluestein's algorithm.

The details of the sub-blocks in Figure 4.10 are given as follows:

- $\mathrm{E}(h) = \mathrm{concat}(h, \exp(i(W_l \cdot W_r + I) \cdot h))$

  Aims to emulate the effects of the SLM encodings. I is the identity matrix, W_r and W_l are as detailed in the Q block below, and the exponentiation is, as in the rest of this chapter, element-wise.

- $\mathrm{P}(h) = \mathcal{Z}_{\psi_1}(\exp(iW) \cdot \mathcal{Z}_{\psi_2}((W_l \cdot W_r + I) \cdot h))$

  Aims to emulate the transfer function of the optical system between devices, i.e. imaging the first SLM's conjugate plane onto the second, and similarly the conjugate plane of the second SLM onto the camera. $\mathcal{Z}$ is the chirp Z-transform (CZT), with tunable parameters ψ_i.

- $\mathrm{BP}x(h_1, h_2) = (\mathrm{Q}(h_1, h_2), h_2^\dagger \odot \exp(iW) \odot (h_1 \cdot \exp(iW_l) \cdot \exp(iW_r) \cdot h_1))$

  Represents the function mapping estimates of complex fields Y and H (see Section 4.3.2), produced by policy head network, to the Jacobian of the PNN output w.r.t. the input. Q is as defined below, and 'W's are real matrices of learnable weights.

- $\mathrm{BP}\theta(h_1, h_2) = (\mathrm{Q}(h_1, h_2), h_2^\dagger \odot \exp(iW) \odot h_1)$

  Similarly, represents the function mapping estimates of Y and H to the Jacobian of the PNN output w.r.t. the parameters.

- $\mathrm{C}(h_1, h_2) = \mathrm{concat}(h_1, h_2, h_1 \odot h_2)$

- $\mathrm{M}(h_1, h_2, \Delta y) = \mathrm{concat}(h_1, h_2, h_1 \cdot h_2 \cdot \Delta y, W_l \cdot W_r \cdot \Delta y)$

  Concatenates the inputs with their matrix product. This reflects the form $h_1 \cdot h_2 \approx \frac{\partial y}{\partial p}$. W_l and W_r are low-rank matrices as in the Q blocks.

- $\mathrm{LSTM}(h, s_p)$

  LSTM block with carried state s_p.

- $\mathrm{Q}(h_i) = \mathrm{LReLu}(W_l \cdot W_r \cdot \mathrm{concat}(h_1, \ldots, h_i))$

  A low-rank dense matrix operation. Concatenates all inputs, multiplies by a $\mathbb{R}^{k \times d}$ matrix W_r, then a $\mathbb{R}^{d/2 \times k}$ matrix W_l. For all experiments we choose $k = 1024$ and d is the dimension of the inputs. The output is passed through a leaky ReLu nonlinearity, with leakage-rate 0.01.

- $\mathrm{J}(h_1, h_2) = W_2 \cdot \mathrm{LReLu}(W_1 \cdot \mathrm{concat}(h_1, h_2))$

  As in block J in Figure 4.8.

- $\mathrm{D}(h_1, h_2) = \text{concat}\left(\mathrm{Re}(h_1), \mathrm{Im}(h_1), \mathrm{Re}(h_2), \mathrm{Im}(h_2), |h_1|^2 - |h_2|^2\right)$

  Assuming the output of the head block learns some representation of the complex field Y, we calculate a loose representation of Δy.

4.5.2 Training the deep learnt optimiser

We have now completed the transition to an MPO agent, with network architectures which we believe should be well-suited to the task of training a deep network, with stronger conditioning on the structure of our PNN. The next question is then how do we train this agent. As much as possible, we would like our learnt error propagation model to be independent of the deep network architecture, so that we can reuse this optimiser on a range of tasks and architectures. This is the key property which makes our optimisation scheme tractable as we scale the total number of network parameters, as in regular backpropagation. We need to be able to assume that whatever input error signal Δx we calculate, it is backpropagated and correctly realised by the changes we make to the parameters of earlier layers in the network. We also need to be able to assume that if we then enact changes Δx and $\Delta\theta$ on the PNN input and parameters, then the change we observe at the output will be consistent with the target error signal Δy, which we used to generate the backpropagated errors. To ensure this, we change our reward function to prioritise gradient alignment over loss minimisation by setting $\alpha = 0.05$.

The largest challenge we face is that we need to train the networks to predict the error signal independent of the form of the input itself. If we meta-train only inputting values of x which are recognisably images, then when we deploy this in a deep network, a PNL in the middle of the network will receive a very different input distribution, where an image has been transformed into some feature representation by the earlier layers. This is essentially the same problem discussed at the end of Section 4.4.7.

The simplest approach is to train on random data, for instance randomly sampling $x \sim \mathcal{U}(-1, 1)^{d_x}$ and $\Delta y \sim \mathcal{U}(-1, 1)^{d_y}$ at each step, and set reward $\alpha = 0$ such that we only optimise for alignment with the target Δy. This would provide the most general training data, as we get task and dataset independence naturally. However, it is incredibly difficult for the agent to learn anything useful from this data, as there is no obvious correlation between samples. In theory with enough training data, this approach could work, but it is not scalable and would require a prohibitive amount of compute.

Instead, our approach to solve this problem is, in each meta-training episode, to sandwich a single PNL in a procedurally generated, conventional deep network. Our full DNN is then $(g^T \circ f_\theta \circ g^H)(x)$, where g^H and g^T are the head and tail of the network respectively,

both implemented as regular differentiable neural networks. These are constructed from a series of dense and convolutional layers, depending on the type of task we are performing.

The head and the tail are separately constructed for each episode by uniformly sampling a set of hyperparameters at the start of the episode:

- Number of layers $\in \{0, 1, 2\}$.
- Type of each layer, either dense or convolutional. The head layers are always convolutional for image tasks. Convolutional layers have a kernel size of 5, a stride of 1, and are zero padded to maintain the input dimension.
- Number of nodes (in $\{d/4, d/2, d, 2d, 4d\}$ where d is the input vector length) or channels (in $\{1, 4, 8, 16, 32\}$) per layer.
- Activation function per layer, between linear, GeLU, and sigmoid.

They are constrained to ensure the dimension of the PNL input and output match the output and input of the head and tail respectively.

The reason behind this seemingly convoluted approach is to generate realistic synthetic data with which to train the agent. In fact, this approach of randomising the network we are training to allow generalisation to different deep architectures is identical to the way we attempt to encourage generalisation across tasks—by randomly generating and sampling from a distribution of synthetic tasks.

From a software point of view, we treat the deep network as any other neural network, and train it using backpropagation. When we perform a forward pass, any time we encounter a physical layer, we dispatch the input to the physical system, measure the output, and then pass that on to the next layer in the network. When we perform a backward pass, we auto-differentiate through any regular layers, and when we encounter a physical layer, we replace the auto-diff code with a call to the learnt optimiser π_ϕ. This makes it clear that the learnt model truly does act as an estimate for the backpropagation operation through the physical layer.

The equivalent of meta-training Algorithm 1 for the deep network optimiser is therefore given in Algorithm 4[15].

[15]For simplicity, we give the unbatched version. The batched version follows the same process as from Algorithm 1 to Algorithm 3.

Algorithm 4 Deep learnt optimiser meta-training.

Initialise task distribution $\mathcal{T}$, agent networks π_ϕ and q_{ϕ_q}, forward model f, reward function R, experience buffer $\mathcal{B}$.
for episode **do**
  Sample dataset and loss function $(\mathcal{D}, L) \sim \mathcal{T}$.
  Initialise deep model head and tail $(g^H_{\theta^H}, g^T_{\theta^T})$.
  Initialise PNL parameters θ_0.
  for $t \leftarrow 0$ to $t_{\max}$ **do**
    Sample $(x^{\text{in}}, y) \sim \mathcal{D}$.
    Evaluate $x = g^H_{\theta^H}(x^{\text{in}})$.
    Evaluate $h = f_\theta(x)$.
    Evaluate $\hat{y} = g^T_{\theta^T}(h)$.
    if end_condition$(t, L(\hat{y}, y))$ **then**
      break
    end if
    Package state $s = \left(\frac{\partial L(\hat{y},y)}{\partial h}, x, \theta_t\right)$.
    Sample updates $(\Delta x, \Delta\theta) \sim \pi_\phi(\Delta x, \Delta\theta \mid s)$.
    Calculate errors on head and tail parameters $\Delta\theta^H$ and $\Delta\theta^T$ with normal backprop, where Δx is the error on the output of g^H.
    Update parameters $\theta, \theta^H, \theta^T$ with Adam optimiser to get $t+1$ values, from $\Delta\theta, \Delta\theta^H, \Delta\theta^T$.
    Evaluate $x' = g^H_{\theta^H_{t+1}}(x^{\text{in}})$.
    Evaluate $h' = f_{\theta_{t+1}}(x')$.
    Evaluate $\hat{y}' = g^T_{\theta^T_{t+1}}(h')$.
    Calculate reward $r = R(L, \hat{y}', \hat{y}, y)$.
    Package new state $s' = \left(\frac{\partial L(\hat{y}',y)}{\partial \hat{y}'}, x', \theta_{t+1}\right)$.
    Store experience $(s, \Delta\theta, r, s')$ in buffer $\mathcal{B}$.
  end for
  Update ϕ and ϕ_q according to RL algorithm, sampling from $\mathcal{B}$.
end for

4.5.3 Results

We train the deep network optimiser as detailed in Section 4.5.2. For evaluation, we solve all test tasks using a 2-layer network where both layers are PNLs, i.e. unlike the meta-training process, we do not use any conventional deep network layers, relying instead only on the PNL transformations for feature extraction.

The PNL used follows the same optics model as in Section 4.3.1, where we take the high-fidelity model as the ground truth system, and the low-fidelity model as a realistic digital twin.

Compared with the results shown in Section 4.4.7, here we also broaden the task distribution. While we still construct synthetic tasks from a seed classification task according to the permutation method in Section 4.4.2, in addition to the MNIST dataset, we also generate tasks for meta-training using the Fashion MNIST dataset [XRV17]. In evaluation, we also introduce the Iris dataset [DG17], which we exclude from the meta-training task distribution. While this is an easy task to solve, the results here demonstrate that the learnt optimiser is able to generalise to some tasks outside the synthetic distribution we use in meta-training. All test tasks, where their datasets were used in meta-training, are evaluated using generated tasks with different permutations, ensuring there is no overlap between training and test tasks.

In order to better evaluate our learnt optimiser against alternative training strategies commonly used in neuromorphic computing, we compare with three other methods:

- SGD on the high-fidelity model, which provides an expected upper bound on final performance, due to the ability to calculate correct gradients.
- SGD on the low-fidelity model, which provides an example of sim-to-real training, with Fourier-optics approximations and no knowledge of the real system's distortion, misalignment and noise.
- Rm-ES [LZ18], which provides a comparison with a model-free, zeroth order optimisation method.

Rm-ES was chosen from the set of available evolutionary strategies for its efficiency in large parameter spaces, and we use the implementation from the evosax library [Lan22], where we set the rank $m = 32$. As opposed to other covariance-based methods such as CMA-ES [HO96; HMK03; ES15], Rm-ES allows us to scale sub-quadratically in parameters by maintaining a low-rank approximation of the covariance matrix, making problems with high-dimensional parameter spaces tractable, while maintaining some second-order information for efficient search. This is yet another example of low-rank approximations helping to balance tractability in large parameter spaces, with the benefits obtained from higher-order information, such as covariance estimates.

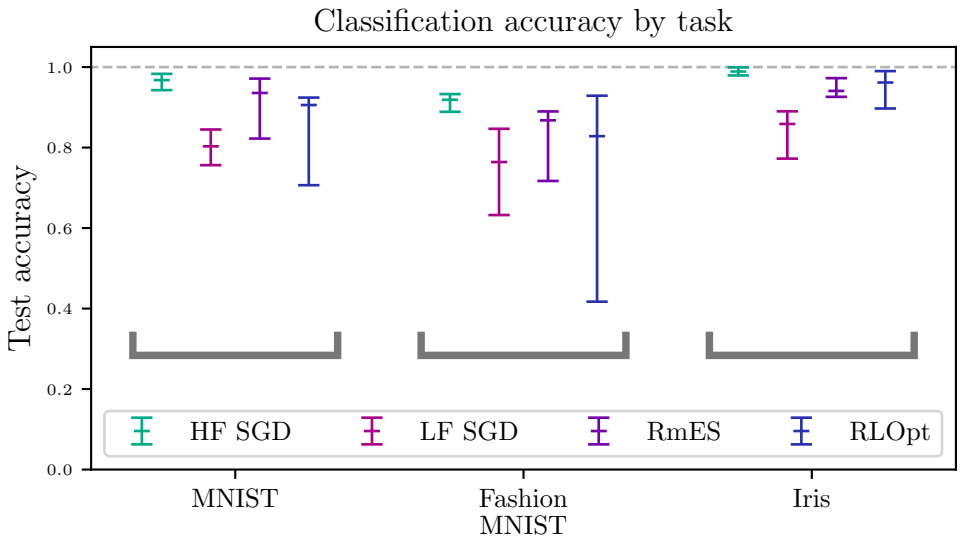


Figure 4.11: **Deep MPO test accuracy.** Classification accuracy of different optimisation strategies on a 2-layer PNN. Median test accuracy over 10 seeds is displayed for each optimiser and task, with error bars showing the min-max variation. Each task trained with SGD on high-fidelity model, SGD on low-fidelity model, rank-m evolutionary strategy (RmES), and RL optimiser (RLOpt). Each trained for minimum 2 epochs, and then until test loss variation $\leq 5\%$ over 3 epochs. Test accuracy for all tasks evaluated on high-fidelity model with unseen data. Evaluated on three datasets: MNIST, Fashion MNIST and Iris [LCB10; XRV17; DG17]. Iris encoded using 2×2 macropixels.

Rm-ES and RLOpt both perform forward passes through the high-fidelity model, and then use their respective methods for updating parameters θ. HF SGD also uses the high-fidelity model for the forward pass, but performs an additional backward pass through this model, where our ability to differentiate in simulation lets us calculate a ground truth SGD evaluation for comparison purposes. LF SGD uses the low-fidelity model for both forward and backward passes, updating parameters using SGD, providing a classic sim-to-real training example. All models' learnt parameters are then evaluated on the high-fidelity model with unseen tasks.

The final test accuracies for the three tasks, four optimisers, all evaluated on 10 seeds (where the initial network parameters are the same for each optimisation method on a given seed and task), are shown in Figure 4.11. The number of samples required to reach the stopping criteria (minimum of 2 epochs, and then until test loss variation is less than 5% over 3 consecutive epochs) are shown in Table 4.2.

The results shown here were obtained using an optimiser meta-trained for 2000 episodes, each with up to 10 epochs (600,000 steps), following Algorithm 4. The entire training process took 11 days, using the same hardware as in Section 4.4.7. The input data space and parameter spaces are the same size, and are limited to a maximum of 112×112 macropixels, giving approximately 25,000 tunable parameters in the 2-layer PNN.

Table 4.2: **Median training samples required for stopping criteria.**

	MNIST	Fashion MNIST	Iris
HF SGD	1.2×10^5	1.2×10^5	3.0×10^2
LF SGD	1.2×10^5	1.8×10^5	3.0×10^2
RmES	1.1×10^7	1.5×10^7	9.0×10^2
RLOpt	3.1×10^6	5.6×10^6	7.7×10^3

We see in Figure 4.11 that, as expected, high-fidelity SGD performs the best, with the lowest spread across seeds. The low-fidelity SGD has the worst median performance, due to not being able to capture the true dynamics of the network which it is evaluated on. The evolutionary strategy tends to outperform the low-fidelity SGD, but we can see in Table 4.2 that it tends to require at least two orders of magnitude more samples to do so. The learnt optimiser performs well, with median accuracies which outperform the low-fidelity SGD, and in the case of Iris, also Rm-ES. We do see however that the learnt optimiser has a larger variation on final performance, with some seeds outperforming Rm-ES and approaching the performance of HF SGD. On the other hand, some seeds struggle to learn well, with performance particularly poor on the Fashion MNIST dataset.

Considering Table 4.2, we see that the SGD-based methods both converge quickly, however the LF model's accuracy plateaus at a lower level than the HF model. The evolutionary strategy requires many more samples to converge, and the learnt optimiser sits somewhere in the middle, outperforming Rm-ES on the two image classification tasks, but not on Iris.

Notably, the number of samples needed by the learnt optimiser is considerably more than the SAC model we saw earlier, on the same MNIST tasks. This is due to the broader task distribution we used in the deep meta-training, along with the reward function which prioritises gradient alignment over loss minimisation. The broader distribution highlights that it is likely the SAC model was at least partially overfit to the MNIST tasks, and had memorised good parameter vector features for this class of tasks. We saw in Figure 4.7 that despite the different label permutations, the optimal parameters still shared some common features, which could be learnt by the model directly, breaking our task invariance criteria.

Overall, these results are encouraging, and demonstrate that the learnt optimiser is able to train a deep PNN composed of individual, non-differentiable physical systems, successfully achieving good performance on image classification tasks, and generalising well to the Iris task, not seen during the meta-training process. While the learnt optimiser is not competitive with SGD on the high-fidelity model, we would not expect

it to be, and on more realistic baselines such as the low-fidelity SGD and evolutionary strategies, it performs well, giving improved accuracy on the former, and improved convergence time on the latter.

The variance and stability of the optimiser remain issues, however these are not issues unique to our application [Hen+19]. We do see that the MPO model with physical priors seems to have improved overall performance and reliability compared to the SAC model, and we are confident that further tuning will continue to improve the performance of the agent—the system has not yet been fully optimised.

As with the SAC model, meta-training time remains one of the largest barriers to progress. While improved hardware, more compute, and moving from simulation to real physical systems could improve the speed we can generate samples and, in turn, training time, these are temporary solutions to the larger impediment to progress—the low sample efficiency on current model-free reinforcement learning methods.

4.6 Discussion

4.6.1 Remaining challenges

We have seen that the learnt optimiser is able to perform well on both single PNNs, where we optimise purely for performance on the end task, and when adapted to act more as an error propagation model, such that we can train deep networks. However, there are clearly still areas to improve on here, and in this section, we give an overview of the most pressing challenges, and the potential ways to address them.

The overall goal of this work was to develop an optimisation scheme which could be used without analytic gradients to train systems in a scalable way. While we have worked with relatively high-dimensional systems, successfully training a deep PNN with order 25,000 parameters, we have faced several issues in the reliability of the system.

The two main bottlenecks thus far have been the length of time needed to perform a single meta-training run, and the stability of the training process. The latter in particular means that for a given training run, there is a reasonable likelihood that nothing meaningful will be learnt. It is worth noting that many of the issues faced during the development of this work are known difficulties with modern RL approaches, including the state-of-the-art algorithms, and a nice summary is provided in [Irp18].

The length of time we need to spend on meta-training is determined by the sample efficiency, essentially how informative each experience gathered at a time step in an episode is in updating our policy and value estimates. This in turn is determined by the exploration strategy, the quality of the reward signal, and the capacity of the neural networks to represent a function which aligns well with the true Jacobian of the linearised PNN. These are the same factors which in part determine the stability of the

training process.

Optimising the task distribution we train with would likely improve the sample efficiency of the meta-training process, as we know that there is a fine balance to be struck between having too broad a distribution, leading to poor performance on all tasks, and too narrow a distribution, leading to poor generalisation. Our approach was based on synthetic tasks, which allowed us to tune this distribution, however for generalisation to more interesting tasks, we need to consider how we might build better distributions more suited to generalisation to real-world tasks. This is a problem common in reinforcement learning, and there have been several approaches to generating synthetic tasks and benchmarks for training agents [RT20; Ope+21; Gis+21]. While these are not directly applicable to our problem, they provide a good starting point for how we might improve out procedurally generated tasks to encourage better generalisation.

In general, we have seen that we need to have some sort of dimensionality reduction in our optimisation process in order to be able to scale to more complex systems. The goal of scaling is not simply to increase the number of tunable parameters, but rather to increase the complexity of the system we are training such that it can solve more complex tasks. We need to increase this computational capacity while maintaining the ability to train the system in a reasonable time frame.

Throughout this work we have made use of low-rank approximations in the structure of our neural networks, making it possible to apply our methods to high-dimensional systems. While this is a good start, and we do expect that a low-dimensional representation is needed, we didn't choose the basis of these representations in any principled way. One way to improve here, is through physical priors, which have also used in our models to design small, efficient networks which should still be able to represent the relatively complex dynamics of the physical systems we are training. We have seen that priors can be introduced in several ways, such as network structure, conditioning the data distributions, or regularisation on the loss terms.

We must still be careful baking in priors however, as this does potentially limit our networks, restricting the representations they can learn to the ones we select for. In effect, we bias the networks towards our idea of how the problem should be solved, which may not be most effective, limiting the learning capacity in potentially unforeseen ways. One of the advantages of using a trainable black-box model here is that it is able to identify shortcuts, symmetries and correlations that it can exploit to solve the problem, and constraining the model with too many priors can prevent it from making good use of these freedoms.

Alternatively, we could imagine better incorporating existing knowledge and models of our physical systems by designing a bootstrapping process, where we first train parameters using a low-fidelity model which is cheap to evaluate, before using a learnt

optimiser specialised to the target hardware to fine-tune the resulting parameters. For instance, in [Zho+21] a deep optical neural network's parameters are first trained in simulation, transferred to the target hardware, and then iteratively updated *in situ* on hardware to account for the sim-to-real misalignment, with other similar examples including works [Lin+18; Lee+20]. While this method is expensive, due to the number of additional parameter updates introduced by the fine-tuning process, it is still cheaper than zeroth order methods, highlighting the potential improvements which can be gained through a bootstrapping process with a—potentially lower fidelity—digital twin.

One of the reasons this approach—locating a region in parameter space which works well in the low fidelity model and using it as a starting point in the real system—works, is that there is a hierarchical representation of the system. What we mean by this is that we can refine the quality of a model without drastically changing the layout of the objective function in the parameter space, meaning that parameters which perform well in a low-fidelity model will also likely perform well in the higher-fidelity system. This is a property of the physical system and the way we model the different levels of fidelity.

For instance, consider once more our optical PNN described in Section 4.3.1. We can interpret the operation of the PNN as a convolution on the input data, where our parameters are the frequency domain representation of the convolution kernel. If we start off by only tuning the central region of the parameter SLM, this acts as a band-limited convolution kernel on the input data. For certain tasks, kernels with low-frequency components may be sufficient to get good performance. We would expect that if we solve this optimisation, and then reintroduce the higher frequency components, that the overall form of the new, optimal kernel, will be similar to the band-limited one. We can therefore improve training performance by iteratively increasing the number of tunable parameters, only when the performance on previous resolution has stabilised.

We do a coarse optimisation over a smaller parameter space, then reintroduce some additional parameters, where the progress made thus far in the optimisation is maintained as the parameter space is expanded. In analogy, the low-fidelity model helps us identify the mountain range in parameter-objective space with the highest peaks, refining the model slightly lets us identify a specific mountain, and at the highest fidelity we focus in on an exact peak. While the particular PNN we have considered may not be the optimal example for such iterative refinement, we could imagine other systems which are better suited to this kind of parameter annealing.

Even with improvements to the task distribution and conditioning of the agent, it is likely that model-free RL will always have poor sample efficiency for our use case. Introducing physical priors has helped here, and in doing so we are in effect making our model-free algorithm more model-based. One of the main advantages of RL is its generality—we can apply these methods to any physical system, regardless of whether

we have a model. In practice however, we have seen that the generality has a cost, relying on the meta-training process to solve our problems for us. With this in mind, we could decide to switch entirely to a model-based approach, similar to MPC. MPC is well established in control applications, and model-based RL has seen many uses in robotics and games [Jum+21; Ber22]. Other model-based methods such as MuZero [Sch+20] are able to learn a model containing only the important features for decision-making, and then use this for planning, reducing the complexity of the problem by only considering the vital components for solving the task at hand.

We have established that reinforcement learning has strong connections to control theory, and we anticipate that bridging these domains and further utilising ideas from control theory will provide more efficient methods of training complex physical systems for use as application specific computational accelerators. This would allow us to use our prior knowledge of the physical system's dynamics to improve the training performance, while remaining robust to dynamics not captured by our models. If we can develop a model which allows us to plan, then we can improve the likelihood that the actions we take will lead to positive rewards, and ensure that the experiences we generate during exploration are informative. While we want to avoid fully relying on a model which doesn't match the real system, we could in principal use a low-fidelity model to get gradient estimates, and use RL to map these gradients to gradients of the real system. More work will be required to better understand how we might start to adapt these schemes from classical control, to the rather unwieldy framework of meta-learning.

4.6.2 Towards experiments in hardware

While much of the work so far has been in simulation, the ultimate goal is to deploy the learnt optimiser on a real optical system. However, so far, the instabilities in learning and sensitivity to initial conditions have made it challenging to apply this work on the hardware PNN in the lab. This difficulty is not necessarily due to the hardware system's complexity exceeding that of our simulations, but rather the rate at which we can perform forward passes through the system.

In simulations, we can achieve high sampling rates, but only on relatively low-dimensional systems due to memory limitations, which bound the number of SLM pixels and field discretisations we can use. In the hardware PNN, we can achieve very high-dimensional transformations, with megapixel SLMs readily available, but their frame rates are approximately four times slower than the high-fidelity simulation. In meta-training, we can choose to reduce the size of the input and parameter spaces, but maintaining a high sample rate is critical, due to the low sample efficiency of the model-free RL algorithms we are using. This means that while the overall data throughput of both methods is comparable, total throughput is not the limiting factor. In other words, it is the sheer number of forward model evaluations which matters most for learning a

good policy, rather than the size of the input and parameter spaces.

Our meta-training process in simulation is currently measured in days to weeks, and, despite the improvements made through tuning the algorithms, networks, task distribution, reward function and associated hyperparameters, we still occasionally encounter meta-training runs which fail to converge on a good policy. We could apply our methods in hardware, accepting that a meta-training run will take on order of weeks to converge, however this assumes that we can guarantee that a meta-training run will be successful. Given the sensitivity to initial conditions and the instability of the training process we have seen in our simulations, this is not currently the case.

While we believe we have made significant progress towards improving robustness to random initialisation and meta-training sample efficiency, further work is required to make the meta-training process consistently repeatable. Once this is achieved, the cost of meta-training on the PNN hardware can be warranted, enabling us to focus on scaling the optimiser to PNNs with larger parameter spaces. At that point, the higher dimensional transformations achievable in hardware offset the impact of low frame rates compared to simulated systems.

The alternative is to revisit our choice of PNN, and consider moving to faster systems, such as DMDs, or away from free space optics entirely. These are reasonable considerations, and throughout this work we have intentionally tried to keep the learnt optimiser as general as possible, to allow for easy transfer to different physical systems. The only modifications which would be required would be to reevaluate the physical priors we embed in the model to match the new system's characteristics. That said, we have seen in Section 4.4.7 that, at least on the PNN considered here, the method is able to learn reasonable policies even with networks which aren't heavily conditioned on the physical system's structure. We therefore expect that this approach would be applicable on a range of physical and neuromorphic systems, provided that there is some tailoring of the architecture and optimisation process to one another.

4.6.3 Conclusion

Reinforcement learning has been shown to be a viable approach to training non-differentiable systems such as physical neural networks, holding potential for broader applications in a wider range of physical computing and neuromorphic systems.

We have shown that it is possible to use reinforcement learning to learn a model which maps from errors on the output of a non-differentiable physical system, to updates to the system's parameters which ultimately act to minimise some objective function, allowing us to train the system to perform a range of tasks. While RL has been used in the past to train physical computing systems on particular tasks, to our knowledge this is the first time it has been used to train an *optimiser* specialised to such systems.

This learnt optimiser allows us to learn rules which are robust to factors such as noise, distortion, and misalignment, which are typically not considered in digital-twins and sim-to-real transfer optimisation schemes. By training an optimiser, we can trade off a relatively expensive initial training cost, for efficient inference—corresponding to training the PNN on a task—over a range of tasks.

We extended this system to predict errors not just on the system's parameters, but also the inputs, demonstrating that it is possible to train a black-box model using RL to approximate the full backpropagation process through an otherwise non-differentiable system, such that we get good alignment with the true error gradients. With such a model, we can then train deep networks composed of a mixture of conventional silicon-based parameterised layers, and unconventional physical computing systems, using proven iterative optimisers such as SGD or Adam.

The elegance of this approach is that it can be smoothly integrated with existing gradient-based pipelines for training neural networks, where we simply dispatch to the physical system and the learnt optimiser whenever we need to do a forward or backward pass through the physical layer. Close integration with existing digital computing technologies will be important for any future physical computing system, and this method provides one path towards achieving this.

While there remain many areas where improvements could be made to this system, the results presented here demonstrate the potential of this approach, and we have given a comprehensive overview of what we believe to be the most important areas for future work to focus on, to improve the performance and scalability of this method.

It seems clear that any scalable optimisation scheme for physical systems will need to be co-designed with the physical system in order to reduce the effective size of the parameter space, either through low-rank approximations, physical priors, or hierarchical optimisation algorithms with better scaling laws.

We believe that learnt optimisers such as ours are particularly applicable in situations where the physical system is hard to model effectively, is static over long periods of time, and where a series of different but related tasks are to be performed on the same system, making the initial meta-training costs acceptable for downstream efficiency gains.

4.7 NOTATION

For reference, we summarise some of the main notation specific to this chapter.

- $f_\theta(x)$: The physical system's mapping from parameters θ and input x to output.
- $\hat{f}_\theta(x)$: Approximate model of physical system.
- $\mathcal{T}$: Distribution of tasks
- $(\mathcal{D}, L)$: Particular task with dataset $\mathcal{D}$ and loss function L. A dataset consists of labelled pairs $\mathcal{D} = \{(x_i, y_i)\}$, while $L(\hat{y}, y) : \mathbb{R}^{d_{\hat{y}}} \times \mathbb{R}^{d_y} \to \mathbb{R}$. Note, for simplicity, the output of f doesn't necessarily need to be the same dimension as the target y; in this case the loss function is assumed to map $\hat{y}$ to a space where the two are comparable.
- $R(L, \hat{y}', \hat{y}, y)$: General reward function, which may depend on the loss function L, the PNN output after a parameter update $\hat{y}'$, the previous PNN output $\hat{y}$, and the target y.
- $\hat{\square}$: An estimate or prediction of $\square$, i.e. $\hat{y} = f_\theta(x)$ is a learnt estimate of label y.
- $\square'$: The value of $\square$ after a parameter update, i.e. $\theta' = \theta + \Delta\theta$, $\hat{y}' = f_{\theta+\Delta\theta}(x)$.
- ℓ: The loss value at a particular set of x, y, θ, i.e. $\ell = L(f_\theta(x), y)$.
- $\pi_\phi(s)$: A policy network, parameterised by ϕ, which depends on the current state s and predicts a parameter update. For stochastic policies $\pi_\phi(\Delta\theta \mid s)$, this is a conditional distribution over updates, and the inference policy is obtained through maximum likelihood $\pi_\phi(s) = \text{argmax}_{\Delta\theta}\, \pi_\phi(\Delta\theta \mid s)$.
- $q_{\phi_q}(s, a)$: A state-action value network parameterised by ϕ_q, trained to estimate some joint value function of a state-action pair (s, a).
- $V_{\phi_V}(s)$: A state value network parameterised by ϕ_V, trained to estimate some value function of state s.

Chapter 5

Conclusion

While there is much that machine learning and AI can offer today, and we expect these tools to become ever more useful, there remains a large gap between the capabilities of biological systems such as the human brain, and these artificial systems.

Stemming from both fundamental human curiosity, and the desire to solve complex problems which face our society, there is a huge drive to further develop these AI systems beyond their current 'stochastic parrot' ([Ben+21]) forms, in order to achieve artificial general intelligence (AGI), and potentially artificial consciousness.

Personally, I believe that these goals will inevitably be accomplished, and that there is nothing inherently privileged about the human brain which, with enough time, human endeavour could not replicate in an artificial system. However, it also seems clear that there will need to be several fundamental changes in the way that we approach computing before this is achievable. This can be seen even from a pure scaling point of view, considering that currently, the largest AI models (GPT-o1) have several orders of magnitude fewer connections than synapses in the human brain.

These changes need to be made first in the physical substrates which we use to process information, allowing us to scale beyond Moore's law by developing systems with better thermal, power, and space efficiency compared to current silicon von Neumann architectures, while maintaining some level of generality. As a direct result, this will require new methods for programming these systems which scale in step with them, allowing us to exploit the complexity and richness of these platforms to solve specific tasks.

Consider a potential 'ultimate' computer designed according to fundamental physical limits, such as the smallest scales on which we can design systems before noise and quantum effects break deterministic operation, the largest three-dimensional energy and connectivity densities which could be realised, and the fastest speeds at which we can transfer information. In order to operate at these limits, such a system would likely need to be analogue in at least some areas, and designed with minimal abstractions in order to maximise the efficiency of the computation. At the very least, the abstractions

which would allow us to program such a machine would look very different to those we operate with today.

While such a device may never exist, taking a shorter term, more realistic outlook, there is definitely a resurgence of analogue and unconventional computing research in academia and commercial enterprises alike, as the development of algorithmic and data-driven methods outpaces the scaling of the platforms they run on [Cav+22; Mar+24; ext24; Dav+18; Sha+21].

By exploiting new physics and dynamics, physical and neuromorphic computing systems have the potential to unlock new scaling laws, and therefore play a large role in the future of high-performance computing [Fil+22]. However, these efficiency and scaling advantages often come with other costs, such as poor programmability, interpretability, and limited generality. While we can in some cases build abstractions which allow us to better understand and program these systems, we need to take care not to undermine the performance advantages in doing so.

In this thesis, we characterised the developments of these unconventional computing systems according to two main areas: the physical substrate, and the programming model. These are delineated by which aspects of the system we consider to be 'hardware' and 'software', or which parts are fixed or tunable for specific tasks. While it is true that the data-driven optimisation approaches to programming seen in machine learning blur these boundaries, allowing us to optimise aspects of the substrates, we will always have some hyperparameters or design decisions which need to be made in an intelligent way, and which determine the types of problems we can solve with a particular architecture.

Based on this classification, we have examined two main examples in depth. Firstly, by designing and analysing a new unconventional computing platform, our quantum reservoir computer detailed in Chapter 3, we have shown that exploiting the physics of photon number resolved measurement allows us to unlock a highly scalable and rich space of dynamics which can be used for computing. Our focus on the use of simple quantum resources, practical methods of realising unitary transformations through linear optical networks, and realistic detection properties means that this system is not only theoretically interesting, but also currently accessible with low-cost resources, achievable in many optical laboratory settings.

Secondly, by identifying the key limitations in programming unconventional and neuromorphic computing platforms, we developed in Chapter 4 an entirely new approach to training neuromorphic computing systems in a data-driven way, using a learnt optimiser. This approach is based on the desire to balance model-free zeroth order optimisations that are compatible with non-differentiable physical systems, while introducing physical priors and models to improve the efficiency and scaling of the training process. The optimiser we put forward has particular benefits in that it follows a scaling law similar

to that of backpropagation, allowing us to train deep networks composed of individual physical systems.

While there are many avenues left for further development in both works, there are certain common conclusions that can be drawn. A consistent feature that we relied on throughout the thesis was the use of low-dimensional approximations to complex systems. These took the form of low-rank matrix approximations, frequency decompositions, low-fidelity models that capture broad stroke dynamics, and physical priors used to condition neural network architectures. In all cases, they served the same purpose—providing tractable ways to analyse or model the complex dynamics of our systems. The coming hardware revolution will require the breaking of many layers of abstraction, but it seems clear that any practical unconventional computing system will need to maintain some principled method of reducing the complexity of the system into tractable parts. Interpretability and provable deterministic behaviour will be necessary for any safety critical applications, and these areas remain challenging even for conventional machine learning, let alone neuromorphic computing.

It is also likely that meta-learning techniques, including those based on RL methods, will begin to feature more heavily in the neuromorphic computing community, and they have already begun to see use in QML applications. Reservoir computing is one avenue towards realising QML, however there are a wide range of other approaches that are not so easy to train, and these will need robust approaches to parameter tuning in high-dimensional spaces [Don+08; Ver+19; Wil+21; He+22; YPK23].

There is much speculation that the large language models and generative image models that have filtered into our daily lives, and which currently represent the largest trained AI models, are heavily overparameterised for what they are capable of. This can be seen in the success of fine-tuning techniques such as low-rank adaptation (LoRA) [Hu+21], where a tiny fraction of the parameters in a pre-trained model can be updated to give very different results. While overparameterisation may be necessary to some extent for robust, stable predictions, it does hint at a problem in the way we optimise these systems. The scaling of these models has been possible due to the relative efficiency of backpropagation and GPU compute, but this is hitting limits now. We expect that for these models to scale further, they will need to also adopt more efficient architectures and optimisation techniques, which exploit the structure of the models and the tasks they solve in a more principled way.

While much of the focus in unconventional computing is on computational efficiency, we can also view the current investment and efforts in the field as an opportunity for improving some of the sustainability and accessibility issues present in current, conventional methods. From a practical point of view, it is important that new computing systems are energy efficient, however there is an additional moral and ethical

imperative to ensure they are sustainable to operate, that their benefits are shared widely, and that they are not used to further entrench existing divisions. Already, we see a trend of groups formerly committed to open-source machine learning and AI research going closed-source and proprietary, as commercial interests take over, and this has huge implications for the future uses of these technologies, and who will benefit most from them. This gradual limiting of transparency has been driven ultimately by the lack of large scale accessible compute, limiting access to the forefront of the field to those who are able to build data centres to train ever larger models, reducing competition and diversity in the field. In the longer term, unconventional computing has the potential to democratise access to large scale compute, just as accessible GPUs have done over the last decade, by allowing for more efficient and scalable systems which can be built and maintained by smaller groups or individuals. We need to ensure that whichever unconventional computing systems win the performance lottery also embody these principles of sustainability and accessibility.

To conclude, it is likely that unconventional computing architectures will play an increasingly large role in future computing paradigms, including AI. This thesis aims to take a small step towards realising improved, optimised neuromorphic computing systems, and to provide a framework for understanding the critical criteria such systems must meet. There are many applications where we expect neuromorphics to be useful—not just in the traditional AI and machine learning domains, or even in the search for AGI, but also in scientific computing, and specific applications which require, for instance, extreme processing speeds. That said, biological learning systems like the brain remain the most efficient that we have, and while they may not be the paragon of possible computing architectures, it is their ability to learn quickly and efficiently which is the true mark of intelligence. Until we can replicate this in artificial systems, artificial general intelligence and artificial consciousness will remain in the realm of science fiction.

Appendix A

Terminology

In the field of novel information processing and computing technologies, a wide range of terminology is used. In an effort to provide clarity and context for our work, here we summarise our uses of some key terms throughout this thesis.

Modern computing relies on one particular high level architecture—the von Neumann, silicon transistor based computer. Algorithms are implemented in this model using a set of abstract instructions which manipulate state stored in memory. We refer to this as conventional computing, and from the Church-Turing thesis, it is provably able to compute any *computable function*, making it extremely general.

Application specific systems, by contrast, sacrifice some aspects of general-purpose computing, for an advantage in particular tasks. Early examples include specialised cryptography hardware built into CPUs, and GPUs which are optimised for highly parallel graphics processing, and have seen significant use in machine learning due to this parallelism. The TPU is an extension of this, discarding graphics processing features and optimising purely for tensor operations.

Unconventional computing refers to systems which deviate from the conventional computing paradigm. These include methods that can be simulated on a traditional computer, but operate in some fundamentally different way. This includes fields as diverse as quantum, optical, memristor based, chemical based, and DNA based computing, and we also consider machine learning within this category [Ada18].

Physical or natural computing refers to systems that use physical processes to encode and process information[1], and can be viewed as a subset of unconventional computing. The prime example here is the human brain, which computes using the connectivity, and electrical and chemical properties of neurons.

Analogue computing involves systems which encode and process information with continuous variables—for practical reasons many physical computing systems are analogue

[1]We distinguish physical computing from conventional digital silicon transistor methods, despite these clearly having a physical substrate.

in their representation of information.

Neuromorphic computing draws inspiration from the brain. While some define it as strictly adhering to biological principles (i.e. spiking neurons, local learning), there is a trend (which we adopt here) to consider any system using artificial neurons as being neuromorphic.

Physical neural networks are a subset of both physical and neuromorphic computing, where we implement systems which look like traditional artificial neural networks in physical systems, i.e. they have data inputs, artificial neurons with tunable parameters, and use optimisation-based programming over these parameters, based on an objective function on the dataset and the network's output.

At a high level, this thesis aims to contribute some improvements to the field of physical neuromorphic computing. We break this down into two themes: the architectural design of a computing system, and the method of programming it.

Architectural design can be seen as an optimisation over the system's fixed features, which determine properties such as the class of tasks that can be solved, the limiting time complexity on those tasks, and the energy efficiency of the system. Conventional computers are the most familiar, and over the years, several layers of abstraction have emerged, with the highest level being the programming languages which are used to implement algorithms. The modern artificial neural networks used in machine learning, represent an alternative architecture. Despite being implemented on conventional computers, they are logically different, being built from parameterised elements (i.e. dense, convolutional, attention layers) combined in specific ways, while hyperparameters dictate the way they are programmed, a process typically referred to as 'training'. In physical computing systems, the architecture consists of the substrate, the information carrying medium, coupling, time dynamics, and other physical features which influence system behaviour.

Programming a computing system involves encoding a task into the architecture and solving it by adjusting the degrees of freedom of the system. The result can be viewed as the 'algorithm' running on top of the architecture. In a conventional computer, the hardware is abstracted away, leaving a set of instructions with which humans design and build algorithms by hand, choosing how to represent inputs and manipulate state held in memory (Software 1.0). However, programming may also take the form of a mathematical optimisation over the system's parameters, a process of algorithmic discovery, (Software 2.0). This is the case in ML, where a dataset and objective function defines an optimisation problem, which can be solved to produce an algorithm i.e. a fully trained neural network. In this case, computer programming and mathematical optimisation, or programming, coincide.

In this thesis, all the unconventional computing systems we consider are analogue, physical neural networks and are programmed by optimising over the system's parameters. We therefore use 'programming', 'training', 'learning', and 'optimising' interchangeably to refer to this process.

Appendix B

Derivations

B.1 General beamsplitter ellipse derivation

The derivation of the ellipse equations from the beamsplitter inequality (Equation 3.6) can be seen below. We can rewrite the inequality as

$$r_2^2 + t_2^2 - r_1^2 r_2^2 - t_1^2 t_2^2 - 2r_1 r_2 t_1 t_2 \cos(\phi) \leq 1 - r_1^2 - t_1^2. \tag{B.1}$$

Rotating coordinate system with $r_2 \to y\cos(\beta) + x\sin(\beta)$ and $t_2 \to x\cos(\beta) - y\sin(\beta)$, expanding and gathering terms, we get

$$\begin{aligned}\frac{1}{2}\left(2 - r_1^2 - t_1^2 + (r_1^2 - t_1^2)\cos(2\beta) + 2r_1 t_1 \cos(\phi)\sin(2\beta)\right) x^2 \\ + \frac{1}{2}\left(2 - r_1^2 - t_1^2 - (r_1^2 - t_1^2)\cos(2\beta) - 2r_1 t_1 \cos(\phi)\sin(2\beta)\right) y^2 \\ + \left((t_1^2 - r_1^2)\sin(2\beta) + 2r_1 t_1 \cos(\phi)\cos(2\beta)\right) xy \leq 1 - r_1^2 - t_1^2.\end{aligned} \tag{B.2}$$

By gathering coefficients of x and y we can identify the cross-term, which for an ellipse aligned with the axes should be zero. This allows us to solve for β,

$$\begin{aligned} 0 &= (t_1^2 - r_1^2)\sin(2\beta) + 2r_1 t_1 \cos(\phi)\cos(2\beta) \\ \sin(2\beta) &= \frac{2r_1 t_1}{r_1^2 - t_1^2}\cos(\phi)\cos(2\beta) \\ \tan(2\beta) &= \frac{2r_1 t_1}{r_1^2 - t_1^2}\cos(\phi) \\ \beta &= \frac{1}{2}\arctan\left(\frac{2r_1 t_1}{r_1^2 - t_1^2}\cos(\phi)\right). \end{aligned}$$

We can then rewrite Equation B.2 as

$$g_+ x^2 + g_- y^2 \leq 1 - r_1^2 - t_1^2$$

with $g_\pm = \frac{1}{2}\left(2 - r_1^2 - t_1^2 \pm h\right)$ and $h = (r_1^2 - t_1^2)\cos(2\beta) + 2r_1 t_1 \cos(\phi)\sin(2\beta)$. The

boundary of this region matches the equation for an ellipse with semi-major and -minor axes, $a^2 = \frac{1-r_1^2-t_1^2}{g_+}$ and $b^2 = \frac{1-r_1^2-t_1^2}{g_-}$. Solving for these values using the above solution for $\tan(2\beta)$ and the identities

$$\cos(2\beta) = \cos(\arctan(\tan(2\beta))) = (1+\tan^2(2\beta))^{-\frac{1}{2}}$$
$$\sin(2\beta) = \sin(\arctan(\tan(2\beta))) = \tan(2\beta)(1+\tan^2(2\beta))^{-\frac{1}{2}}$$
$$a^2 + b^2 + 2ab\cos\theta = |a + be^{i\theta}|^2,$$

we get

$$\begin{aligned}
h &= (r_1^2 - t_1^2)\cos(2\beta) + 2r_1t_1\cos(\phi)\sin(2\beta) \\
&= (r_1^2 - t_1^2)\frac{r_1^2 - t_1^2}{|r_1^2 + t_1^2 e^{i2\phi}|} + 2r_1t_1\cos(\phi)\frac{2r_1t_1\cos(\phi)}{|r_1^2 + t_1^2 e^{i2\phi}|} \\
&= \frac{(r_1^2 - t_1^2)^2 + 4r_1^2t_1^2\cos^2(\phi)}{|r_1^2 + t_1^2 e^{i2\phi}|} \\
&= \frac{r_1^4 + t_1^4 + 2r_1^2t_1^2\cos(2\phi)}{|r_1^2 + t_1^2 e^{i2\phi}|} \\
&= |r_1^2 + t_1^2 e^{i2\phi}| \\
\Rightarrow g_\pm &= \frac{1}{2}\left(2 - r_1^2 - t_1^2 \pm |r_1^2 + t_1^2 e^{i2\phi}|\right).
\end{aligned}$$

This then gives the ellipse parameters as

$$\begin{aligned}
\beta &= \frac{1}{2}\arctan\left(\cos(\phi)\frac{2r_1t_1}{r_1^2 - t_1^2}\right), \\
a^2 &= \frac{2(1 - r_1^2 - t_1^2)}{2 - r_1^2 - t_1^2 - |r_1^2 + t_1^2 e^{i2\phi}|}, \\
b^2 &= \frac{2(1 - r_1^2 - t_1^2)}{2 - r_1^2 - t_1^2 + |r_1^2 + t_1^2 e^{i2\phi}|}.
\end{aligned}$$

B.2 Computational complexity of permanent calculations

Ryser's algorithm calculates the permanent of an $N \times N$ matrix A as

$$\text{perm}(A) = \sum_{S \subseteq \{1,\ldots,N\}} (-1)^{|S|+N} \prod_{i=1}^{N} \sum_{j \in S} A_{ij}.$$

A single subset comprises $N(|S|-1)$ additions a, followed by $N-1$ multiplications b. Total operations are then

$$\begin{aligned}
\#\text{Ops} &= \sum_{S \subseteq \{1,\ldots,N\}} (N(|S|-1)a + (N-1)b) \\
&= \sum_{k=0}^{N} \binom{N}{k} (N(k-1)a + (N-1)b) \\
&= Na \sum_{k=0}^{N} \binom{N}{k} k + (-Na + (N-1)b) \sum_{k=0}^{N} \binom{N}{k} \\
&= (N^2 2^{N-1} - N)a + (N-1)2^N b \\
&\Rightarrow O(N^2 2^N).
\end{aligned}$$

If instead we process each subset in Gray code order (i.e. each subset differs from the last only by inclusion or exclusion of a single index) and store the results from the inner summation, then we only perform N additions for each subset,

$$\begin{aligned}
\#\text{Ops} &= \sum_{S \subseteq \{1,\ldots,N\}} (Na + (N-1)b) \\
&= 2^N (Na + (N-1)b) \\
&\Rightarrow O(N 2^N).
\end{aligned}$$

Bibliography

* indicates equal contributions

[20] "Computing on the brain", in: *Nat. Electron.* 3.7 (July 2020), pp. 347–347, ISSN: 2520-1131, DOI: 10.1038/s41928-020-0457-1.

[23] "AI hardware has an energy problem", in: *Nat. Electron.* 6.7 (July 1, 2023), pp. 463–463, ISSN: 2520-1131, DOI: 10.1038/s41928-023-01014-x.

[AA11] S. Aaronson and A. Arkhipov, "The computational complexity of linear optics", in: *Proceedings of the forty-third annual ACM symposium on Theory of computing*, Nov. 2011, pp. 333–342, DOI: 10.48550/arXiv.1011.3245, arXiv: 1011.3245.

[AAM24] A. H. Abbas, H. Abdel-Ghani, and I. S. Maksymov, "Classical and Quantum Physical Reservoir Computing for Onboard Artificial Intelligence Systems: A Perspective", in: (June 2024), DOI: 10.20944/preprints202406.1128.v1.

[AAS10] I. Afek, O. Ambar, and Y. Silberberg, "High-NOON States by Mixing Quantum and Classical Light", in: *Science* 328.5980 (2010), pp. 879–881, DOI: 10.1126/science.1188172.

[Abb99] L. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)", in: *Brain Res. Bull.* 50.5–6 (Nov. 1999), pp. 303–304, ISSN: 0361-9230, DOI: 10.1016/s0361-9230(99)00161-6.

[Abd+18a] A. Abdolmaleki, J. T. Springenberg, J. Degrave, S. Bohez, Y. Tassa, D. Belov, N. Heess, and M. Riedmiller, *Relative Entropy Regularized Policy Iteration*, Dec. 2018, DOI: 10.48550/arxiv.1812.02256, arXiv: 1812.02256 [cs.LG].

[Abd+18b] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, *Maximum a posteriori policy optimisation*, June 2018, DOI: 10.48550/ARXIV.1806.06920, arXiv: 1806.06920 [cs.LG].

[Abr+24] S. Abreu, I. Boikov, M. Goldmann, T. Jonuzi, A. Lupo, S. Masaad, L. Nguyen, E. Picco, G. Pourcel, A. Skalli, L. Talandier, B. Vettelschoss, E. A. Vlieg, A. Argyris, P. Bienstman, D. Brunner, J. Dambre, L. Daudet, J. D. Domenech, I. Fischer, F. Horst, S. Massar, C. R. Mirasso, B. J. Offrein, A. Rossi, M. C. Soriano, S. Sygletos, and S. K. Turitsyn, "A photonics perspective on computing with physical substrates", in: *Rev. Phys.* 12 (Dec. 1, 2024), p. 100093, ISSN: 2405-4283, DOI: 10.1016/j.revip.2024.100093.

[Ach+04] D. Achilles, C. Silberhorn, C. Sliwa, K. Banaszek, I. A. Walmsley, M. J. Fitch, B. C. Jacobs, T. B. Pittman, and J. D. Franson, "Photon-number-resolving detection using time-multiplexing", in: *J. Mod. Opt.* 51-9 (10 2004), pp. 1499–1515, ISSN: 1362-3044, DOI: 10.1080/09500340408235288.

[Ada18] A. Adamatzky, ed., *Unconventional Computing*, SpringerLink, New York, NY: Springer US, 2018, ISBN: 9781493968831.

[AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines*", in: *Cogn. Sci.* 9.1 (Jan. 1985), pp. 147–169, ISSN: 1551-6709, DOI: 10.1207/s15516709cog0901_7.

[Ama98] S.-i. Amari, "Natural Gradient Works Efficiently in Learning", in: *Neural Comput.* 10.2 (Feb. 1998), pp. 251–276, ISSN: 1530-888X, DOI: 10.1162/089976698300017746.

[And+16] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to Learn by Gradient Descent by Gradient Descent", in: *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 3988–3996.

[Ant20] L. P. G. Antiga, *Deep learning with PyTorch*, ed. by E. Stevens and T. Viehmann, Shelter Island, NY: Manning, 2020, ISBN: 9781638354079.

[Arr+21] J. M. Arrazola, V. Bergholm, K. Brádler, T. R. Bromley, M. J. Collins, I. Dhand, A. Fumagalli, T. Gerrits, A. Goussev, L. G. Helt, J. Hundal, T. Isacsson, R. B. Israel, J. Izaac, S. Jahangiri, R. Janik, N. Killoran, S. P. Kumar, J. Lavoie, A. E. Lita, D. H. Mahler, M. Menotti, B. Morrison, S. W. Nam, L. Neuhaus, H. Y. Qi, N. Quesada, A. Repingon, K. K. Sabapathy, M. Schuld, D. Su, J. Swinarton, A. Száva, K. Tan, P. Tan, V. D. Vaidya, Z. Vernon, Z. Zabaneh, and Y. Zhang, "Quantum circuits with many photons on a programmable nanophotonic chip", in: *Nature* 591 (7848 Mar. 2021), pp. 54–60, ISSN: 1476-4687, DOI: 10.1038/s41586-021-03202-1.

[Bar+10] M. Barbieri, N. Spagnolo, M. G. Genoni, F. Ferreyrol, R. Blandino, M. G. Paris, P. Grangier, and R. Tualle-Brouri, "Non-Gaussianity of quantum states: An experimental test on single-photon-added coherent states", in: *Phys. Rev. A - At. Mol. Opt. Phys.* 82 (6 Dec. 2010), ISSN: 1050-2947, DOI: 10.1103/PhysRevA.82.063833.

[Bel+17] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural Optimizer Search with Reinforcement Learning", in: *Proceedings of the 34th International Conference on Machine Learning*, PMLR, July 17, 2017, pp. 459–468, URL: https://proceedings.mlr.press/v70/bello17a.html.

[Bel10] R. Bellman, *Dynamic programming*, Princeton landmarks in mathematics, Princeton: Princeton University Press, 2010, ISBN: 1400835380.

[Ben+21] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?", in: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, ACM, Mar. 2021, DOI: 10.1145/3442188.3445922.

[Ber20] M. N. Bernstein, *Expectation-maximization: theory and intuition*, May 13, 2020, URL: https://mbernste.github.io/posts/em/.

[Ber22] D. P. Bertsekas, *Lessons from AlphaZero for Optimal, Model Predictive, and AdaptiveControl*, Athena Scientific, 2022, ISBN: 978-1-886529-17-5.

[Bia+17] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning", in: *Nature* 549 (7671 Sept. 2017), pp. 195–202, ISSN: 1476-4687, DOI: 10.1038/nature23474.

[Bog+20] W. Bogaerts, D. Pérez, J. Capmany, D. A. Miller, J. Poon, D. Englund, F. Morichetti, and A. Melloni, *Programmable photonic circuits*, Oct. 2020, DOI: 10.1038/s41586-020-2764-0.

[BP21] D. Brunner and D. Psaltis, "Competitive photonic neural networks", in: *Nat. Photonics* 15.5 (May 2021), pp. 323–324, ISSN: 1749-4893, DOI: 10.1038/s41566-021-00803-0.

[Bra+18] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *JAX: composable transformations of Python+NumPy programs*, 2018, URL: http://github.com/google/jax.

[Bro+13] M. A. Broome, A. Fedrizzi, S. Rahimi-Keshari, J. Dove, S. Aaronson, T. C. Ralph, and A. G. White, "Photonic boson sampling in a tunable circuit", in: *Science* 339 (6121 Feb. 2013), pp. 794–798, ISSN: 1095-9203, DOI: 10.1126/science.1231440.

[Bru+19] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges", in: *Appl. Phys. Rev.* 6.2 (May 2019), ISSN: 1931-9401, DOI: 10.1063/1.5088164.

[Bru19] D. Brunner, *Photonic Reservoir Computing*, Berlin Boston: Gruyter, Walter de GmbH, July 2019, ISBN: 3110583496, URL: https://www.ebook.de/de/product/38427878/photonic_reservoir_computing.html.

[BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems", in: *IEEE Trans. on Syst. Man, Cybern.* SMC-13.5 (Sept. 1983), pp. 834–846, ISSN: 2168-2909, DOI: 10.1109/tsmc.1983.6313077.

[Bue+18] J. Bueno, S. Maktoobi, L. Froehly, I. Fischer, M. Jacquot, L. Larger, and D. Brunner, "Reinforcement learning in a large-scale photonic recurrent neural network", in: *Optica* 5.6 (June 20, 2018), pp. 756–760, ISSN: 2334-2536, DOI: 10.1364/OPTICA.5.000756.

[Cav+22] A. Cavaillès, P. Boucher, L. Daudet, I. Carron, S. Gigan, and K. Müller, "A high-fidelity and large-scale reconfigurable photonic processor for NISQ applications", in: *Opt. Express Vol. 30, Issue 17, pp. 30058-30065 (2022)* 30.17 (May 3, 2022), p. 30058, ISSN: 1094-4087, DOI: 10.1364/oe.462071, arXiv: 2205.01704 [quant-ph].

[CC18] P. Clifford and R. Clifford, "The Classical Complexity of Boson Sampling", in: Proceedings, Society for Industrial and Applied Mathematics, Jan. 2018, pp. 146–155, DOI: 10.1137/1.9781611975031.10, arXiv: 1706.01260 [cs.DS].

[CC20] P. Clifford and R. Clifford, *Faster classical Boson Sampling*, May 2020, DOI: 10.48550/arXiv.2005.04214, arXiv: 2005.04214.

[CD10] H. Caulfield and S. Dolev, "Why future supercomputing requires optics", in: *Nat. Photonics* 4 (May 2010), DOI: 10.1038/nphoton.2010.94.

[Cec+21] F. Ceccarelli, G. Acconcia, A. Gulinatti, M. Ghioni, I. Rech, and R. Osellame, "Recent Advances and Future Perspectives of Single-Photon Avalanche Diodes for Quantum Photonics Applications", in: *Adv. Quantum Technol.* 4 (2 Feb. 2021), ISSN: 2511-9044, DOI: 10.1002/qute.202000102.

[Che+18] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations", in: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 6572–6583.

[Che+23a] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le, "Symbolic Discovery of Optimization Algorithms", in: *Advances in Neural Information Processing Systems*, vol. 36, arXiv, 2023, pp. 49205–49233, DOI: 10.48550/ARXIV.2302.06675, arXiv: 2302.06675 [cs.LG].

[Che+23b] R. Cheng, Y. Zhou, S. Wang, M. Shen, T. Taher, and H. X. Tang, "A 100-pixel photon-number-resolving detector unveiling photon statistics", in: *Nat. Photonics* 17 (1 Jan. 2023), pp. 112–119, ISSN: 1749-4893, DOI: 10.1038/s41566-022-01119-3.

[Chu36] A. Church, "An Unsolvable Problem of Elementary Number Theory", in: *Am. J. Math.* 58.2 (Apr. 1936), p. 345, ISSN: 0002-9327, DOI: 10.2307/2371045.

[CP76] D. Casasent and D. Psaltis, "Position, rotation, and scale invariant optical correlation", in: *Appl. Opt.* 15.7 (July 1976), p. 1795, ISSN: 1539-4522, DOI: 10.1364/ao.15.001795.

[Cuc+22] M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann, "Hands-on reservoir computing: a tutorial for practical implementation", in: *Neuromorphic Comput. Eng.* 2.3 (Aug. 2022), p. 032002, DOI: 10.1088/2634-4386/ac7db7.

[Dal+20] A. M. Dalzell, A. W. Harrow, D. E. Koh, and R. L. La Placa, "How many qubits are needed for quantum computational supremacy?", in: *Quantum* 4 (May 2020), p. 264, ISSN: 2521-327X, DOI: 10.22331/q-2020-05-11-264.

[Dav+18] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning", in: *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99, ISSN: 1937-4143, DOI: 10.1109/mm.2018.112130359.

[Def+16] H. Defienne, M. Barbieri, I. A. Walmsley, B. J. Smith, and S. Gigan, "Two-photon quantum walk in a multimode fiber", in: *Sci. Adv.* 2 (1 Jan. 2016), ISSN: 2375-2548, DOI: 10.1126/sciadv.1501054.

[Deg+22] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J.-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller, "Magnetic control of tokamak plasmas

through deep reinforcement learning", in: *Nature* 602.7897 (Feb. 2022), pp. 414–419, ISSN: 1476-4687, DOI: 10.1038/s41586-021-04301-9.

[DG17] D. Dua and C. Graff, *UCI Machine Learning Repository*, 2017, URL: http://archive.ics.uci.edu/ml.

[DiV00] D. P. DiVincenzo, "The Physical Implementation of Quantum Computation", in: *Fortschritte der Physik* 48.9–11 (Sept. 2000), pp. 771–783, ISSN: 1521-3978, DOI: 10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data Via the EM Algorithm", in: *J. Royal Stat. Soc. Ser. B: Stat. Methodol.* 39.1 (Sept. 1977), pp. 1–22, ISSN: 1467-9868, DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[Don+08] D. Dong, C. Chen, H. Li, and T.-J. Tarn, "Quantum Reinforcement Learning", in: *IEEE Trans. on Syst. Man, Cybern. Part B (Cybernetics)* 38.5 (Oct. 2008), pp. 1207–1220, ISSN: 1083-4419, DOI: 10.1109/tsmcb.2008.925743.

[EMH19] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey", in: *J. Mach. Learn. Res.* 20.55 (2019), pp. 1–21, ISSN: 1533-7928, URL: http://jmlr.org/papers/v20/18-598.html.

[ES15] A. E. Eiben and J. Smith, "From evolutionary computation to the evolution of things", in: *Nature* 521.7553 (May 2015), pp. 476–482, ISSN: 1476-4687, DOI: 10.1038/nature14544.

[ext24] extropic, *Ushering in the Thermodynamic Future - Litepaper*, Mar. 11, 2024, URL: https://www.extropic.ai/future.

[Far+85] N. H. Farhat, D. Psaltis, A. Prata, and E. Paek, "Optical implementation of the Hopfield model", in: *Appl. Opt.* 24.10 (May 1985), p. 1469, ISSN: 1539-4522, DOI: 10.1364/ao.24.001469.

[FDB24] N. Farmakidis, B. Dong, and H. Bhaskaran, "Integrated photonic neuromorphic computing: opportunities and challenges", in: *Nat. Rev. Electr. Eng.* 1.6 (June 2024), pp. 358–373, ISSN: 2948-1201, DOI: 10.1038/s44287-024-00050-9.

[Fed+22] A. K. Fedorov, N. Gisin, S. M. Beloussov, and A. I. Lvovsky, "Quantum computing at the quantum advantage threshold: a down-to-business review", in: (Mar. 31, 2022), DOI: 10.48550/ARXIV.2203.17181, arXiv: 2203.17181 [quant-ph].

[Fil+22] M. J. Filipovich, A. Cappelli, D. Hesslow, and J. Launay, "Scaling Laws Beyond Backpropagation", in: (Oct. 26, 2022), DOI: 10.48550/ARXIV.2210.14593, arXiv: 2210.14593 [cs.LG].

[Fit+03] M. J. Fitch, B. C. Jacobs, T. B. Pittman, and J. D. Franson, *Photon number resolution using time-multiplexed single-photon detectors*, 2003, DOI: 10.1103/PhysRevA.68.043814.

[FJL24] J. Feng, V. Jirsa, and W. Lu, "Human brain computing and brain-inspired intelligence", in: *National Sci. Rev.* 11.5 (Apr. 2024), ISSN: 2053-714X, DOI: 10.1093/nsr/nwae144.

[Fri10] K. Friston, "The free-energy principle: a unified brain theory?", in: *Nat. Rev. Neurosci.* 11.2 (Jan. 2010), pp. 127–138, ISSN: 1471-0048, DOI: 10.1038/nrn2787.

[FS03] C. Fernando and S. Sojakka, "Pattern Recognition in a Bucket", in: *Advances in Artificial Life*, Springer Berlin Heidelberg, 2003, pp. 588–597, ISBN: 9783540394327, DOI: 10.1007/978-3-540-39432-7_63.

[Gar+14] B. T. Gard, K. R. Motes, J. P. Olson, P. P. Rohde, and J. P. Dowling, "An introduction to boson-sampling", in: (June 2014), DOI: 10.1142/9789814678704_0008.

[Gar+23] J. García-Beni, G. L. Giorgi, M. C. Soriano, and R. Zambrini, "Scalable photonic platform for real-time quantum reservoir computing", in: *Phys. Rev. Appl.* 20 (1 July 2023), p. 014051, DOI: 10.1103/PhysRevApplied.20.014051.

[Gho+19] S. Ghosh, A. Opala, M. Matuszewski, T. Paterek, and T. C. H. Liew, "Quantum reservoir processing", in: *npj Quantum Inf.* 5.1 (1 Dec. 2019), ISSN: 2056-6387, DOI: 10.1038/s41534-019-0149-8.

[Gho+20] S. Ghosh, A. Opala, M. Matuszewski, T. Paterek, and T. C. H. Liew, "Reconstructing quantum states with quantum reservoir networks", in: *IEEE Trans. on Neural Networks Learn. Syst.* (2020), pp. 1–8, ISSN: 2162-237X, 2162-2388, DOI: 10.1109/TNNLS.2020.3009716, arXiv: 2008.06378.

[Gis+21] L. Gisslén, A. Eakins, C. Gordillo, J. Bergdahl, and K. Tollmar, "Adversarial Reinforcement Learning for Procedural Content Generation", in: (Mar. 8, 2021), DOI: 10.48550/ARXIV.2103.04847, arXiv: 2103.04847 [cs.LG].

[GLA22] B. Y. Gan, D. Leykam, and D. G. Angelakis, "Fock state-enhanced expressivity of quantum machine learning models", in: *EPJ Quantum Technol.* 9 (1 Dec. 2022), ISSN: 2196-0763, DOI: 10.1140/epjqt/s40507-022-00135-0.

[GLG23] N. Götting, F. Lohof, and C. Gies, "Exploring quantumness in quantum reservoir computing", in: *Phys. Rev. A* 108.5 (Nov. 27, 2023), p. 052427, ISSN: 2469-9926, DOI: 10.1103/PhysRevA.108.052427.

[GO18] L. Grigoryeva and J.-P. Ortega, "Echo state networks are universal", in: *Neural Networks* 108 (Dec. 2018), pp. 495–508, ISSN: 0893-6080, DOI: 10.1016/j.neunet.2018.08.025.

[GO21] L. Gonon and J.-P. Ortega, "Fading memory echo state networks are universal", in: *Neural Networks* 138 (June 2021), pp. 10–13, ISSN: 0893-6080, DOI: 10.1016/j.neunet.2021.01.025.

[Goo+16] I. Goodfellow, J. Bengio, A. Courville, and F. Bach, *Deep Learning*, MIT Press Ltd, Nov. 2016, ISBN: 978-0262035613, URL: https://www.ebook.de/de/product/26337726/ian_goodfellow_joshua_bengio_aaron_courville_francis_bach_deep_learning.html.

[Goo85] J. W. Goodman, "Fan-in and Fan-out with Optical Interconnections", in: *Opt. Acta: Int. J. Opt.* 32.12 (Dec. 1985), pp. 1489–1496, ISSN: 0030-3909, DOI: 10.1080/713821684.

[Goo96] J. Goodman, *Introduction to Fourier optics*, New York: McGraw-Hill, 1996, ISBN: 0070242542.

[Guo+21] X. Guo, T. D. Barrett, Z. M. Wang, and A. I. Lvovsky, "Backpropagation through nonlinear units for the all-optical training of neural networks", in: *Photonics Res.* 9.3 (Mar. 2021), B71, DOI: 10.1364/PRJ.411104.

[Haa+18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", in: *arXiv preprint arXiv:1801.01290* (2018), DOI: 10.48550/arxiv.1801.01290, arXiv: 1801.01290 [cs.LG].

[Ham23] Hamamatsu Photonics, *qCMOS ® : Quantitative CMOS technology enabled by Photon Number Resolving*, 2023, URL: https://www.hamamatsu.com/content/dam/hamamatsu-photonics/sites/documents/99_SALES_LIBRARY/sys/SCAS0149E_qCMOS_whitepaper.pdf.

[He+22] Z. He, C. Chen, L. Li, S. Zheng, and H. Situ, "Quantum Architecture Search with Meta-Learning", in: *Adv. Quantum Technol.* 5.8 (June 2022), ISSN: 2511-9044, DOI: 10.1002/qute.202100134.

[Heb05] D. O. Hebb, *The Organization of Behavior*, Hoboken: Taylor and Francis, 2005, ISBN: 9780805843002.

[Hen+19] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, *Deep Reinforcement Learning that Matters*, arXiv:1709.06560, Jan. 29, 2019, DOI: 10.48550/arXiv.1709.06560, arXiv: 1709.06560[cs, stat].

[Heu+23a] N. Heurtel, A. Fyrillas, G. d. Gliniasty, R. Le Bihan, S. Malherbe, M. Pailhas, E. Bertasi, B. Bourdoncle, P.-E. Emeriau, R. Mezher, L. Music, N. Belabas, B. Valiron, P. Senellart, S. Mansfield, and J. Senellart, "Perceval: A Software Platform for Discrete Variable Photonic Quantum Computing", in: *Quantum* 7 (Feb. 2023), p. 931, ISSN: 2521-327X, DOI: 10.22331/q-2023-02-21-931.

[Heu+23b] N. Heurtel, S. Mansfield, J. Senellart, and B. Valiron, "Strong simulation of linear optical processes", in: *Comput. Phys. Commun.* 291 (June 2023), p. 108848, ISSN: 0010-4655, DOI: 10.1016/j.cpc.2023.108848.

[Hin22] G. Hinton, "The Forward-Forward Algorithm: Some Preliminary Investigations", in: (Dec. 2022), DOI: 10.48550/ARXIV.2212.13345, arXiv: 2212.13345 [cs.LG].

[HL22] O. Hernández and I. Liberal, "Generalized approach to quantum interference in lossy N-port devices via a singular value decomposition", in: *Opt. Express* 30.17 (Aug. 2022), pp. 31267–31286, DOI: 10.1364/OE.456495.

[HMK03] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)", in: *Evol. Comput.* 11.1 (Mar. 2003), pp. 1–18, ISSN: 1530-9304, DOI: 10.1162/106365603321828970.

[HO96] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation", in: *Proceedings of IEEE International Conference on Evolutionary Computation*, ICEC-96, IEEE, 1996, DOI: 10.1109/icec.1996.542381.

[Hof+20] M. W. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, N. Momchev, D. Sinopalnikov, P. Stańczyk, S. Ramos, A. Raichuk, D. Vincent, L. Hussenot, R. Dadashi, G. Dulac-Arnold, M. Orsini, A. Jacq, J. Ferret, N. Vieillard, S. K. S. Ghasemipour, S. Girgin, O. Pietquin, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Friesen, R. Haroun, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, S. Srinivasan, A. Cowie, Z. Wang, B. Piot, and N. de Freitas, "Acme: A Research Framework for Distributed Reinforcement Learning", in: *arXiv preprint arXiv:2006.00979* (2020), URL: https://arxiv.org/abs/2006.00979.

[HOM87] C. K. Hong, Z. Y. Ou, and L. Mandel, "Measurement of subpicosecond time intervals between two photons by interference", in: *Phys. Rev. Lett.* 59 (18 Nov. 1987), pp. 2044–2046, DOI: 10.1103/PhysRevLett.59.2044.

[Hop82] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.", in: *Proc. National Acad. Sci.* 79.8 (Apr. 1982), pp. 2554–2558, ISSN: 1091-6490, DOI: 10.1073/pnas.79.8.2554.

[Hos+22] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-Learning in Neural Networks: A Survey", in: *IEEE Trans. on Pattern Anal. Mach. Intell.* 44.9 (Sept. 2022), pp. 5149–5169, ISSN: 1939-3539, DOI: 10.1109/TPAMI.2021.3079209.

[HS18] D. Ha and J. Schmidhuber, "Recurrent World Models Facilitate Policy Evolution", in: *Advances in Neural Information Processing Systems 31*, Curran Associates, Inc., 2018, pp. 2451–2463, arXiv: 1803.10122.

[Hu+20] Y. Hu, Z. Wang, X. Wang, S. Ji, C. Zhang, J. Li, W. Zhu, D. Wu, and J. Chu, "Efficient full-path optical calculation of scalar and vector diffraction using the Bluestein method", in: *Light. Sci. & Appl.* 9.1 (July 2020), DOI: 10.1038/s41377-020-00362-z.

[Hu+21] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models", in: (June 17, 2021), DOI: 10.48550/ARXIV.2106.09685, arXiv: 2106.09685 [cs.CL].

[Hu+24] J. Hu, D. Mengu, D. C. Tzarouchis, B. Edwards, N. Engheta, and A. Ozcan, "Diffractive optical computing in free space", in: *Nat. Commun.* 15.1 (Feb. 20, 2024), p. 1525, ISSN: 2041-1723, DOI: 10.1038/s41467-024-45982-w.

[Hua+22] H.-Y. Huang, M. Broughton, J. Cotler, S. Chen, J. Li, M. Mohseni, H. Neven, R. Babbush, R. Kueng, J. Preskill, and J. R. McClean, "Quantum advantage in learning from experiments", in: *Science* 376.6598 (2022), pp. 1182–1186, DOI: 10.1126/science.abn7293.

[Hum+23] R. Hummatov, A. E. Lita, T. Farrahi, N. Otrooshi, S. Fayer, M. J. Collins, M. Durkin, D. Bennett, J. Ullom, R. P. Mirin, and S. W. Nam, "Fast transition-edge sensors suitable for photonic quantum computing", in: *J. Appl. Phys.* 133 (23 June 2023), ISSN: 1089-7550, DOI: 10.1063/5.0149478.

[HZS06] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications", in: *Neurocomputing* 70.1–3 (Dec. 2006), pp. 489–501, ISSN: 0925-2312, DOI: 10.1016/j.neucom.2005.12.126.

[Ich+24] T. Ichikawa, H. Hakoshima, K. Inui, K. Ito, R. Matsuda, K. Mitarai, K. Miyamoto, W. Mizukami, K. Mizuta, T. Mori, Y. Nakano, A. Nakayama, K. N. Okada, T. Sugimoto, S. Takahira, N. Takemori, S. Tsukano, H. Ueda, R. Watanabe, Y. Yoshida, and K. Fujii, "Current

numbers of qubits and their uses", in: *Nat. Rev. Phys.* 6.6 (May 2024), pp. 345–347, ISSN: 2522-5820, DOI: 10.1038/s42254-024-00725-0.

[Irp18] A. Irpan, *Deep Reinforcement Learning Doesn't Work Yet*, 2018, URL: https://www.alexirpan.com/2018/02/14/rl-hard.html.

[JL84] W. B. Johnson and J. Lindenstrauss, *Extensions of Lipschitz mappings into a Hilbert space*, 1984, DOI: 10.1090/conm/026/737400.

[Jum+21] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with AlphaFold", in: *Nature* 596.7873 (July 2021), pp. 583–589, ISSN: 1476-4687, DOI: 10.1038/s41586-021-03819-2.

[Kaa+22] L. H. Kaack, P. L. Donti, E. Strubell, G. Kamiya, F. Creutzig, and D. Rolnick, "Aligning artificial intelligence with climate change mitigation", in: *Nat. Clim. Chang.* 12.6 (June 2022), pp. 518–527, ISSN: 1758-6798, DOI: 10.1038/s41558-022-01377-7.

[Kar21] A. Karpathy, *Software 2.0*, Mar. 13, 2021, URL: https://karpathy.medium.com/software-2-0-a64152b37c35.

[KB14] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", in: (Dec. 22, 2014), DOI: 10.48550/ARXIV.1412.6980, arXiv: 1412.6980 [cs.LG].

[KG21] P. Kidger and C. Garcia, "Equinox: neural networks in JAX via callable PyTrees and filtered transformations", in: *Differ. Program. workshop at Neural Inf. Process. Syst. 2021* (2021), arXiv: 2111.00254.

[Kja+20] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting Qubits: Current State of Play", in: *Annu. Rev. Condens. Matter Phys.* 11.1 (2020), pp. 369–395, DOI: 10.1146/annurev-conmatphys-031119-050605.

[Kri+24] S. Krishnaswamy, F. Schule, L. Ares, V. Dyachuk, M. Stefszky, B. Brecht, C. Silberhorn, and J. Sperling, "Retrieval of photon statistics from click detection", in: (Mar. 2024), DOI: 10.48550/arXiv.2403.11673, arXiv: 2403.11673.

[KS20] L. Kirsch and J. Schmidhuber, "Meta Learning Backpropagation And Improving It", in: *35th Conf. on Neural Inf. Process. Syst. (NeurIPS 2021)* (Dec. 2020), DOI: 10.48550/ARXIV.2012.14905, arXiv: 2012.14905 [cs.LG].

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", in: *Advances in Neural Information Processing Systems*, ed. by F. Pereira, C. Burges, L. Bottou, and K. Weinberger, vol. 25, Curran Associates, Inc., 2012, URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Lan22] R. T. Lange, "evosax: JAX-based Evolution Strategies", in: (2022), arXiv: 2212.04180.

[Lat+24] M. H. Latifpour, B. J. Park, Y. Yamamoto, and M.-G. Suh, "Hyperspectral in-memory computing with optical frequency combs and programmable optical memories", in: *Optica* 11.7 (July 2024), pp. 932–939, DOI: 10.1364/OPTICA.522378.

[Lau+20] J. Launay, I. Poli, F. Boniface, and F. Krzakala, "Direct Feedback Alignment Scales to Modern Deep Learning Tasks and Architectures", in: (June 23, 2020), DOI: 10.48550/arxiv.2006.12878, arXiv: 2006.12878 [stat.ML].

[LB02] A. I. Lvovsky and S. A. Babichev, "Synthesis and tomographic characterization of the displaced Fock state of light", in: *Phys. Rev. A - At. Mol. Opt. Phys.* 66 (1 2002), p. 4, ISSN: 1094-1622, DOI: 10.1103/PhysRevA.66.011801.

[LCB10] Y. LeCun, C. Cortes, and C. Burges, *MNIST handwritten digit database*, 2010, URL: http://yann.lecun.com/exdb/mnist.

[Le 14] F. Le Gall, "Powers of tensors and fast matrix multiplication", in: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, ACM, July 2014, DOI: 10.1145/2608628.2608664.

[Lee+20] J. Lee, J. Jeong, J. Cho, D. Yoo, B. Lee, and B. Lee, "Deep neural network for multi-depth hologram generation and its training strategy", in: *Opt. Express* 28.18 (Aug. 2020), p. 27137, DOI: 10.1364/oe.402317.

[Len03] P. Lennie, "The Cost of Cortical Computation", in: *Curr. Biol.* 13.6 (Mar. 2003), pp. 493–497, ISSN: 0960-9822, DOI: 10.1016/s0960-9822(03)00135-0.

[Li+21] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier Neural Operator for Parametric Partial Differential Equations", in: *arXiv preprint arXiv:2010.08895* (2021), DOI: 10.48550/arxiv.2010.08895, arXiv: 2010.08895 [cs.LG].

[Lil+16] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning", in: *Nat. Commun.* 7.1 (Nov. 2016), ISSN: 2041-1723, DOI: 10.1038/ncomms13276.

[Lil+19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, tech. rep., July 2019, DOI: 10.48550/arXiv.1509.02971, arXiv: 1509.02971.

[Lin+18] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi, and A. Ozcan, "All-optical machine learning using diffractive deep neural networks", in: *Science* 361.6406 (July 2018), pp. 1004–1008, DOI: 10.1126/science.aat8084.

[LM17] K. Li and J. Malik, "Learning to Optimize Neural Nets", in: (Nov. 2017), DOI: 10.48550/arXiv.1703.00441, arXiv: 1703.00441 [cs.LG].

[LMK22] A. Luczak, B. L. McNaughton, and Y. Kubo, "Neurons learn by predicting future activity", in: *Nat. Mach. Intell.* 4.1 (Jan. 2022), pp. 62–72, ISSN: 2522-5839, DOI: 10.1038/s42256-021-00430-y.

[Los+24] J. W. N. Los, M. Sidorova, B. Lopez-Rodriguez, P. Qualm, J. Chang, S. Steinhauer, V. Zwiller, and I. E. Zadeh, "High-performance photon number resolving detectors for 850-950 nm wavelength range", in: *APL Photonics* 9.6 (June 2024), p. 066101, ISSN: 2378-0967, DOI: 10.1063/5.0204340, arXiv: 2401.07265.

[LQP93] H.-Y. S. Li, Y. Qiao, and D. Psaltis, "Optical network for real-time face recognition", in: *Appl. Opt.* 32.26 (Sept. 1993), p. 5026, ISSN: 1539-4522, DOI: 10.1364/ao.32.005026.

[Luo+23] W. Luo, L. Cao, Y. Shi, L. Wan, H. Zhang, S. Li, G. Chen, Y. Li, S. Li, Y. Wang, S. Sun, M. F. Karim, H. Cai, L. C. Kwek, and A. Q. Liu, *Recent progress in quantum photonic chips for quantum communication and internet*, Dec. 2023, DOI: 10.1038/s41377-023-01173-8.

[LZ18] Z. Li and Q. Zhang, "A Simple Yet Efficient Evolution Strategy for Large-Scale Black-Box Optimization", in: *IEEE Trans. on Evol. Comput.* 22.5 (2018), pp. 637–646, DOI: 10.1109/TEVC.2017.2765682.

[Mad+22] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins, A. E. Lita, T. Gerrits, S. W. Nam, V. D. Vaidya, M. Menotti, I. Dhand, Z. Vernon, N. Quesada, and J. Lavoie, "Quantum computational advantage with a programmable photonic processor", in: *Nature* 606 (7912 June 2022), pp. 75–81, ISSN: 1476-4687, DOI: 10.1038/s41586-022-04725-x.

[Mal+23] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-Tuning Language Models with Just Forward Passes", in: *Advances in Neural Information Processing Systems*, ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, vol. 36, Curran Associates, Inc., 2023, pp. 53038–53075, URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/a627810151be4d13f907ac898ff7e948-Paper-Conference.pdf.

[Mar+20a] G. Marcucci, D. Pierangeli, P. W. H. Pinkse, M. Malik, and C. Conti, "Programming multi-level quantum gates in disordered computing reservoirs via machine learning", in: *Opt. Express* 28.9 (Apr. 2020), pp. 14018–14027, DOI: 10.1364/OE.389432.

[Mar+20b] D. Marković, A. Mizrahi, D. Querlioz, and J. Grollier, "Physics for neuromorphic computing", in: *Nat. Rev. Phys.* 2.9 (Sept. 2020), pp. 499–510, ISSN: 2522-5820, DOI: 10.1038/s42254-020-0208-2.

[Mar+24] N. Maring, A. Fyrillas, M. Pont, E. Ivanov, P. Stepanov, N. Margaria, W. Hease, A. Pishchagin, A. Lemaître, I. Sagnes, T. H. Au, S. Boissier, E. Bertasi, A. Baert, M. Valdivia, M. Billard, O. Acar, A. Brieussel, R. Mezher, S. C. Wein, A. Salavrakos, P. Sinnott, D. A. Fioretto, P.-E. Emeriau, N. Belabas, S. Mansfield, P. Senellart, J. Senellart, and N. Somaschi, "A versatile single-photon-based quantum computing platform", in: *Nat. Photonics* 18.6 (Mar. 2024), pp. 603–609, ISSN: 1749-4893, DOI: 10.1038/s41566-024-01403-4.

[Mat+22] Y. Matsuo, Y. LeCun, M. Sahani, D. Precup, D. Silver, M. Sugiyama, E. Uchibe, and J. Morimoto, "Deep learning, reinforcement learning, and world models", in: *Neural Networks* 152 (Aug. 2022), pp. 267–275, ISSN: 0893-6080, DOI: 10.1016/j.neunet.2022.03.037.

[McM23] P. L. McMahon, "The physics of optical computing", in: *Nat. Rev. Phys.* 5.12 (Dec. 1, 2023), pp. 717–734, ISSN: 2522-5820, DOI: 10.1038/s42254-023-00645-5.

[Mee+21] R. van der Meer, S. Huber, P. W. H. Pinkse, R. García-Patrón, and J. J. Renema, "Boson Sampling in Low-depth Optical Systems", in: (Oct. 2021), DOI: 10.48550/arXiv.2110.05099, arXiv: 2110.05099.

[Met+22] L. Metz, J. Harrison, C. D. Freeman, A. Merchant, L. Beyer, J. Bradbury, N. Agrawal, B. Poole, I. Mordatch, A. Roberts, and J. Sohl-Dickstein, "VeLO: Training Versatile Learned Optimizers by Scaling Up", in: (Nov. 17, 2022), DOI: 10.48550/ARXIV.2211.09760, arXiv: 2211.09760 [cs.LG].

[Mni+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning", in: (Dec. 19, 2013), DOI: 10.48550/ARXIV.1312.5602, arXiv: 1312.5602 [cs.LG].

[Mni+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning", in: *ICML 2016* (Feb. 4, 2016), DOI: 10.48550/ARXIV.1602.01783, arXiv: 1602.01783 [cs.LG].

[Mom+24] A. Momeni, B. Rahmani, B. Scellier, L. G. Wright, P. L. McMahon, C. C. Wanjura, Y. Li, A. Skalli, N. G. Berloff, T. Onodera, I. Oguz, F. Morichetti, P. del Hougne, M. L. Gallo, A. Sebastian, A. Mirhoseini, C. Zhang, D. Marković, D. Brunner, C. Moser, S. Gigan, F. Marquardt, A. Ozcan, J. Grollier, A. J. Liu, D. Psaltis, A. Alù, and R. Fleury, *Training of Physical Neural Networks*, arXiv:2406.03372, June 2024, DOI: 10.48550/arXiv.2406.03372, arXiv: 2406.03372.

[Moo06] G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.", in: *IEEE Solid-State Circuits Soc. Newsl.* 11.3 (2006), pp. 33–35, DOI: 10.1109/N-SSC.2006.4785860.

[MP23] I. S. Maksymov and A. Pototsky, "Reservoir computing based on solitary-like waves dynamics of liquid film flows: A proof of concept", in: *Europhys. Lett.* 142.4 (May 2023), p. 43001, ISSN: 1286-4854, DOI: 10.1209/0295-5075/acd471.

[MPC20] G. Marcucci, D. Pierangeli, and C. Conti, "Theory of neuromorphic computing by waves: machine learning by rogue waves, dispersive shocks, and solitons", in: *Phys. Rev. Lett.* 125.9 (Aug. 2020), p. 093901, ISSN: 0031-9007, 1079-7114, DOI: 10.1103/PhysRevLett.125.093901, arXiv: 1912.07044.

[Muj+21] P. Mujal, R. Martínez-Peña, J. Nokkala, J. García-Beni, G. L. Giorgi, M. C. Soriano, and R. Zambrini, "Opportunities in Quantum Reservoir Computing and Extreme Learning Machines", in: *Adv. Quantum Technol.* 4.8 (2021), p. 2100027, DOI: 10.1002/qute.202100027.

[Mun+16] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and Efficient Off-Policy Reinforcement Learning", in: *Advances in Neural Information Processing Systems*, ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, vol. 29, Curran Associates, Inc., June 2016, DOI: 10.48550/arxiv.1606.02647, arXiv: 1606.02647 [cs.LG].

[Mut+19] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation", in: *Microprocess. Microsystems* 67 (June 2019), pp. 28–41, ISSN: 0141-9331, DOI: 10.1016/j.micpro.2019.01.009.

[NE19] A. Nøkland and L. H. Eidnes, "Training Neural Networks with Local Error Signals", in: *Int. Conf. on Mach. Learn.* (Jan. 2019), DOI: 10.48550/arxiv.1901.06656, arXiv: 1901.06656v2 [stat.ML].

[Ner+24] S. Nerenberg*, **O. D. Neill***, G. Marcucci, and D. Faccio, *Photon Number-Resolving Quantum Reservoir Computing*, 2024, DOI: 10.48550/arxiv.2402.06339, arXiv: 2402.06339 [quant-ph].

[Neu93] J. von Neumann, "First draft of a report on the EDVAC", in: *IEEE Ann. Hist. Comput.* 15.4 (1993), pp. 27–75, ISSN: 1058-6180, DOI: 10.1109/85.238389.

[NF24] **O. D. Neill** and D. Faccio, "Optical neural networks trained in situ with reinforcement learning", in: *Machine Learning in Photonics*, ed. by F. Ferranti, M. K. Hedayati, and A. Fratalocchi, vol. 13017, International Society for Optics and Photonics, SPIE, 2024, p. 130170L, DOI: 10.1117/12.3021870.

[Nij14] A. Nijenhuis, *Combinatorial Algorithms*, ed. by H. S. Wilf and W. Rheinboldt, Computer science and applied mathematics, Burlington: Elsevier Science, 2014, ISBN: 9780125192606.

[Nøk16] A. Nøkland, "Direct Feedback Alignment Provides Learning in Deep Neural Networks", in: *Neural Information Processing Systems*, Sept. 2016, DOI: 10.48550/arxiv.1609.01596, arXiv: 1609.01596 [stat.ML].

[Ogu+23] I. Oguz, J. Ke, Q. Weng, F. Yang, M. Yildirim, N. U. Dinc, J.-L. Hsieh, C. Moser, and D. Psaltis, "Forward–forward training of an optical neural network", in: *Opt. Lett.* 48.20 (Oct. 2023), pp. 5249–5252, ISSN: 1539-4794, DOI: 10.1364/OL.496884.

[Ope+21] Open Ended Learning Team, A. Stooke, A. Mahajan, C. Barros, C. Deck, J. Bauer, J. Sygnowski, M. Trebacz, M. Jaderberg, M. Mathieu, N. McAleese, N. Bradley-Schmieg, N. Wong, N. Porcel, R. Raileanu, S. Hughes-Fitt, V. Dalibard, and W. M. Czarnecki, "Open-Ended Learning Leads to Generally Capable Agents", in: (July 27, 2021), DOI: 10.48550/ARXIV.2107.12808, arXiv: 2107.12808 [cs.LG].

[Par+20] J. Park, B. G. Jeong, S. I. Kim, D. Lee, J. Kim, C. Shin, C. B. Lee, T. Otsuka, J. Kyoung, S. Kim, K.-Y. Yang, Y.-Y. Park, J. Lee, I. Hwang, J. Jang, S. H. Song, M. L. Brongersma, K. Ha, S.-W. Hwang, H. Choo, and B. L. Choi, "All-solid-state spatial light modulator with independent phase and amplitude control for

three-dimensional LiDAR applications", in: *Nat. Nanotechnol.* 16.1 (Oct. 2020), pp. 69–76, ISSN: 1748-3395, DOI: 10.1038/s41565-020-00787-y.

[PJ95] S. S. Paivi Torma and I. Jex, "Hamiltonian theory of symmetric optical network transforms", in: *Phys. Rev. A* 52 (6 1995), DOI: 10.1103/physreva.52.4853.

[PMC21] D. Pierangeli, G. Marcucci, and C. Conti, "Photonic extreme learning machine by free-space optical propagation", in: *Photonics Res.* 9.8 (July 2021), p. 1446, ISSN: 2327-9125, DOI: 10.1364/prj.423531.

[Por+21] X. Porte, A. Skalli, N. Haghighi, S. Reitzenstein, J. A. Lott, and D. Brunner, "A complete, parallel and autonomous photonic neural network in a semiconductor multimode laser", in: *J. Physics: Photonics* 3.2 (Apr. 2021), p. 024017, ISSN: 2515-7647, DOI: 10.1088/2515-7647/abf6bd, arXiv: 2012.11153 [physics].

[PS08] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients", in: *Neural Networks* 21.4 (May 2008), pp. 682–697, ISSN: 0893-6080, DOI: 10.1016/j.neunet.2008.02.003.

[Psa+90] D. Psaltis, D. Brady, X.-G. Gu, and S. Lin, "Holography in artificial neural networks", in: *Nature* 343.6256 (Jan. 1990), pp. 325–330, ISSN: 1476-4687, DOI: 10.1038/343325a0.

[Raf+20] M. Rafayelyan, J. Dong, Y. Tan, F. Krzakala, and S. Gigan, "Large-Scale Optical Reservoir Computing for Spatiotemporal Chaotic Systems Prediction", in: *Phys. Rev. X* 10 (4 Nov. 2020), ISSN: 2160-3308, DOI: 10.1103/PhysRevX.10.041037, arXiv: 2001.09131 [physics.optics].

[Raf+21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations", in: *J. Mach. Learn. Res.* 22.268 (2021), pp. 1–8, URL: http://jmlr.org/papers/v22/20-1364.html.

[RB99] R. P. N. Rao and D. H. Ballard, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects", in: *Nat. Neurosci.* 2.1 (Jan. 1999), pp. 79–87, ISSN: 1546-1726, DOI: 10.1038/4580.

[Rec+94] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, *Experimental realization of any discrete unitary operator*, July 1994, DOI: 10.1103/PhysRevLett.73.58.

[RPK19] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations", in: *J. Comput. Phys.* 378 (Feb. 2019), pp. 686–707, ISSN: 0021-9991, DOI: 10.1016/j.jcp.2018.10.045.

[RT20] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation", in: *Nat. Mach. Intell.* 2.8 (Aug. 2020), pp. 428–436, ISSN: 2522-5839, DOI: 10.1038/s42256-020-0208-z.

[Sch+15] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation", in: (June 8, 2015), DOI: 10.48550/ARXIV.1506.02438, arXiv: 1506.02438 [cs.LG].

[Sch+18] M. Schmidt, M. von Helversen, M. López, F. Gericke, E. Schlottmann, T. Heindel, S. Kück, S. Reitzenstein, and J. Beyer, "Photon-Number-Resolving Transition-Edge Sensors for the Metrology of Quantum Light Sources", in: *J. Low Temp. Phys.* 193 (5-6 Dec. 2018), pp. 1243–1250, ISSN: 1573-7357, DOI: 10.1007/s10909-018-1932-1.

[Sch+20] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, chess and shogi by planning with a learned model", in: *Nature* 588.7839 (Dec. 2020), pp. 604–609, ISSN: 1476-4687, DOI: 10.1038/s41586-020-03051-4.

[Sch04] S. Scheel, *Permanents in linear optical networks*, 2004, DOI: 10.48550/arXiv.quant-ph/0406127, arXiv: quant-ph/0406127 [quant-ph].

[SGZ21] S. Steinhauer, S. Gyger, and V. Zwiller, *Progress on large-scale superconducting nanowire single-photon detectors*, Mar. 2021, DOI: 10.1063/5.0044057.

[Sha+21] B. J. Shastri, A. N. Tait, T. Ferreira de Lima, W. H. P. Pernice, H. Bhaskaran, C. D. Wright, and P. R. Prucnal, "Photonics for artificial intelligence and neuromorphic computing", in: *Nat. Photonics* 15.2 (Feb. 2021), pp. 102–114, ISSN: 1749-4893, DOI: 10.1038/s41566-020-00754-y.

[Sil+17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge", in: *Nature* 550.7676 (Oct. 2017), pp. 354–359, ISSN: 1476-4687, DOI: 10.1038/nature24270.

[SP19] S. Slussarenko and G. J. Pryde, "Photonic quantum information processing: A concise review", in: *Appl. Phys. Rev.* 6.4 (Oct. 2019), ISSN: 1931-9401, DOI: 10.1063/1.5115814.

[SRF24] N. A. Silva, V. Rocha, and T. D. Ferreira, "Optical Extreme Learning Machines with Atomic Vapors", in: *Atoms* 12.2 (Feb. 2024), p. 10, ISSN: 2218-2004, DOI: 10.3390/atoms12020010.

[SS14] B. Sengupta and M. B. Stemmler, "Power Consumption During Neuronal Computation", in: *Proc. IEEE* 102.5 (May 2014), pp. 738–750, ISSN: 1558-2256, DOI: 10.1109/jproc.2014.2307755.

[SSM21] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum-machine-learning models", in: *Phys. Rev. A* 103 (3 Mar. 2021), ISSN: 2469-9934, DOI: 10.1103/PhysRevA.103.032430.

[Sta+24] A. Stanojevic, S. Woźniak, G. Bellec, G. Cherubini, A. Pantazi, and W. Gerstner, "High-performance deep spiking neural networks with 0.3 spikes per neuron", in: *Nat. Commun.* 15.1 (Aug. 9, 2024), p. 6793, ISSN: 2041-1723, DOI: 10.1038/s41467-024-51110-5.

[Sub+24] A. Subramoney, G. Bellec, F. Scherr, R. Legenstein, and W. Maass, "Fast learning without synaptic plasticity in spiking neural networks", in: *Sci. Reports* 14.1 (Apr. 12, 2024), p. 8557, ISSN: 2045-2322, DOI: 10.1038/s41598-024-55769-0.

[Sut20] R. S. Sutton, *Reinforcement learning*, ed. by A. Barto, Adaptive computation and machine learning, Cambridge, Massachusetts: The MIT Press, 2020, ISBN: 9780262039246, URL: http://incompleteideas.net/book/the-book.html.

[TAD24] V. Thibeault, A. Allard, and P. Desrosiers, "The low-rank hypothesis of complex systems", in: *Nat. Phys.* (Jan. 2024), ISSN: 1745-2473, DOI: 10.1038/s41567-023-02303-0.

[Tan+19] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review", in: *Neural Networks* 115 (July 2019), pp. 100–123, ISSN: 0893-6080, DOI: 10.1016/j.neunet.2019.03.005, arXiv: 1808.04962.

[Tan+22] Y. Tang, P. T. Zamani, R. Chen, J. Ma, M. Qi, C. Yu, and W. Gao, "Device-system Co-design of Photonic Neuromorphic Processor using Reinforcement Learning", in: (Mar. 9, 2022), DOI: 10.48550/ARXIV.2203.06061, arXiv: 2203.06061 [cs.ET].

[Tar15] A. Taroni, "90 years of the Ising model", in: *Nat. Phys.* 11.12 (Dec. 2015), pp. 997–997, ISSN: 1745-2481, DOI: 10.1038/nphys3595.

[TR19] S. H. Tan and P. P. Rohde, *The resurgence of the linear optics quantum interferometer – recent advances and applications*, Nov. 2019, DOI: 10.1016/j.revip.2019.100030.

[TT96] L. Troyansky and N. Tishby, "On the quantum evaluation of the determinant and the permanent of a matrix", in: *Proc. Phys. Comput.* 96 (1996).

[Tur04] A. Turing, "Intelligent Machinery (1948)", in: *The Essential Turing*, Oxford University PressOxford, Sept. 2004, pp. 395–432, ISBN: 9780191916526, DOI: 10.1093/oso/9780198250791.003.0016.

[Tur37] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem", in: *Proc. Lond. Math. Soc.* s2-42.1 (1937), pp. 230–265, ISSN: 0024-6115, DOI: 10.1112/plms/s2-42.1.230.

[Tza+19] O. Tzang, E. Niv, S. Singh, S. Labouesse, G. Myatt, and R. Piestun, "Wavefront shaping in complex media with a 350 kHz modulator via a 1D-to-2D transform", in: *Nat. Photonics* 13.11 (Aug. 2019), pp. 788–793, ISSN: 1749-4893, DOI: 10.1038/s41566-019-0503-6.

[Upp+16] R. Uppu, T. A. W. Wolterink, T. B. H. Tentrup, and P. W. H. Pinkse, "Quantum optics of lossy asymmetric beam splitters", in: *Opt. Express* 24.15 (July 2016), pp. 16440–16449, DOI: 10.1364/OE.24.016440.

[Ver+19] G. Verdon, M. Broughton, J. R. McClean, K. J. Sung, R. Babbush, Z. Jiang, H. Neven, and M. Mohseni, "Learning to learn with quantum neural networks via classical neural networks", in: (July 11, 2019), DOI: 10.48550/ARXIV.1907.05415, arXiv: 1907.05415 [quant-ph].

[VT20] F. J. G. Vidal and D. O. Theis, "Input Redundancy for Parameterized Quantum Circuits", in: *Front. Phys.* 8 (Aug. 2020), ISSN: 2296-424X, DOI: 10.3389/fphy.2020.00297.

[VV22] L. Valantinas and T. Vettenburg, *A physics-defined recurrent neural network to compute coherent light wave scattering on the millimetre scale*, arXiv:2208.01118, Dec. 2022, DOI: 10.48550/arXiv.2208.01118, arXiv: 2208.01118.

[Wai24] X. Waintal, "The Quantum House Of Cards", in: *Proc. National Acad. Sci.* 121.1 (Jan. 2, 2024), e2313269120, ISSN: 0027-8424, 1091-6490, DOI: 10.1073/pnas.2313269120, arXiv: 2312.17570[quant-ph].

[Wan+18] Z. Wang, S. Joshi, S. Savel'ev, W. Song, R. Midya, Y. Li, M. Rao, P. Yan, S. Asapu, Y. Zhuo, H. Jiang, P. Lin, C. Li, J. H. Yoon, N. K. Upadhyay, J. Zhang, M. Hu, J. P. Strachan, M. Barnell, Q. Wu, H. Wu, R. S. Williams, Q. Xia, and J. J. Yang, "Fully memristive neural networks for pattern classification with unsupervised learning", in: *Nat. Electron.* 1.2 (Feb. 2018), pp. 137–145, ISSN: 2520-1131, DOI: 10.1038/s41928-018-0023-2.

[Wan+19a] H. Wang, J. Qin, X. Ding, M. C. Chen, S. Chen, X. You, Y. M. He, X. Jiang, L. You, Z. Wang, C. Schneider, J. J. Renema, S. Höfling, C. Y. Lu, and J. W. Pan, "Boson Sampling with 20 Input Photons and a 60-Mode Interferometer in a 1014 -Dimensional Hilbert

Space", in: *Phys. Rev. Lett.* 123 (25 Dec. 2019), ISSN: 1079-7114, DOI: 10.1103/PhysRevLett.123.250503.

[Wan+19b] X. Wang, J. A. J. Fells, W. C. Yip, T. Ali, J.-d. Lin, C. Welch, G. H. Mehl, M. J. Booth, T. D. Wilkinson, S. M. Morris, and S. J. Elston, "Fast and low loss flexoelectro-optic liquid crystal phase modulator with a chiral nematic reflector", in: *Sci. Reports* 9.1 (May 2019), ISSN: 2045-2322, DOI: 10.1038/s41598-019-42831-5.

[Wan+24] H. Wang, J. Hu, A. Morandi, A. Nardi, F. Xia, X. Li, R. Savo, Q. Liu, R. Grange, and S. Gigan, "Large-scale photonic computing with nonlinear disordered media", in: *Nat. Comput. Sci.* 4.6 (June 2024), pp. 429–439, ISSN: 2662-8457, DOI: 10.1038/s43588-024-00644-1.

[Web11] C. S. Webster, "Alan Turing's unorganized machines and artificial neural networks: his remarkable early work and future possibilities", in: *Evol. Intell.* 5.1 (July 2011), pp. 35–43, ISSN: 1864-5917, DOI: 10.1007/s12065-011-0060-5.

[Wic+17] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned Optimizers that Scale and Generalize", in: (Mar. 2017), DOI: 10.48550/ARXIV.1703.04813, arXiv: 1703.04813 [cs.LG].

[Wil+21] M. Wilson, R. Stromswold, F. Wudarski, S. Hadfield, N. M. Tubman, and E. G. Rieffel, "Optimizing quantum heuristics with meta-learning", in: *Quantum Mach. Intell.* 3.1 (Apr. 2021), ISSN: 2524-4914, DOI: 10.1007/s42484-020-00022-w.

[Wil92] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning", in: *Mach. Learn.* 8.3 (May 1, 1992), pp. 229–256, ISSN: 1573-0565, DOI: 10.1007/BF00992696.

[Win+10] A. Windhager, M. Suda, C. Pacher, M. Peev, and A. Poppe, "Quantum Interference between a Single-Photon Fock State and a Coherent State", in: (Sept. 2010), DOI: 10.1016/j.optcom.2010.12.019.

[Wri+22] L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. T. Schachter, Z. Hu, and P. L. McMahon, "Deep physical neural networks trained with backpropagation", in: *Nature* 601.7894 (Jan. 2022), pp. 549–555, DOI: 10.1038/s41586-021-04223-6.

[Wri+23] D. Wright, C. Igel, G. Samuel, and R. Selvan, "Efficiency is Not Enough: A Critical Perspective of Environmentally Sustainable AI", in: *arXiv preprint arXiv:2309.02065* (Sept. 2023), DOI: 10.48550/arXiv.2309.02065, arXiv: 2309.02065 [cs.LG].

[Wu+22] Y. Wu, R. Zhao, J. Zhu, F. Chen, M. Xu, G. Li, S. Song, L. Deng, G. Wang, H. Zheng, S. Ma, J. Pei, Y. Zhang, M. Zhao, and L. Shi, "Brain-inspired global-local learning incorporated with neuromorphic computing", in: *Nat. Commun.* 13.1 (Jan. 10, 2022), p. 65, ISSN: 2041-1723, DOI: 10.1038/s41467-021-27653-2.

[Xio+23] W. Xiong, G. Facelli, M. Sahebi, O. Agnel, T. Chotibut, S. Thanasilp, and Z. Holmes, *On fundamental aspects of quantum extreme learning machines*, arXiv:2312.15124, Dec. 23, 2023, DOI: 10.48550/arXiv.2312.15124, arXiv: 2312.15124[quant-ph,stat] [quant-ph].

[XRV17] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms", in: *CoRR* abs/1708.07747 (2017), arXiv: 1708.07747.

[You+20] C. You, M. A. Quiroz-Juárez, A. Lambert, N. Bhusal, C. Dong, A. Perez-Leija, A. Javaid, R. D. J. León-Montiel, and O. S. Magaña-Loaiza, "Identification of light sources using machine learning", in: *Appl. Phys. Rev.* 7 (2 June 2020), ISSN: 1931-9401, DOI: 10.1063/1.5133846.

[YPK23] W. J. Yun, J. Park, and J. Kim, "Quantum Multi-Agent Meta Reinforcement Learning", in: *Proc. AAAI Conf. on Artif. Intell.* 37.9 (June 2023), pp. 11087–11095, ISSN: 2159-5399, DOI: 10.1609/aaai.v37i9.26313.

[ZF23] Y. Zhang and N. K. Fontaine, "Multi-Plane Light Conversion: A Practical Tutorial", in: (Apr. 22, 2023), DOI: 10.48550/ARXIV.2304.11323, arXiv: 2304.11323 [physics.optics].

[Zho+20] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, P. Hu, X.-Y. Yang, W.-J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N.-L. Liu, C.-Y. Lu, and J.-W. Pan, "Quantum computational advantage using photons", in: *Science* 370.6523 (Dec. 2020), pp. 1460–1463, ISSN: 0036-8075, 1095-9203, DOI: 10.1126/science.abe8770.

[Zho+21] T. Zhou, X. Lin, J. Wu, Y. Chen, H. Xie, Y. Li, J. Fan, H. Wu, L. Fang, and Q. Dai, "Large-scale neuromorphic optoelectronic computing with a reconfigurable diffractive processing unit", in: *Nat. Photonics* 15.5 (May 2021), pp. 367–373, ISSN: 1749-4893, DOI: 10.1038/s41566-021-00796-w.

[ZL16] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning", in: (Nov. 2016), DOI: 10.48550/ARXIV.1611.01578, arXiv: 1611.01578 [cs.LG].

[ZLM23] M. Zajnulina, A. Lupo, and S. Massar, "Weak Kerr Nonlinearity Boosts the Performance of Frequency-Multiplexed Photonic Extreme Learning Machines: A Multifaceted Approach", in: (Dec. 19, 2023), DOI: 10.48550/ARXIV.2312.12296, arXiv: 2312.12296 [physics.optics].

[ZVB04] A. Zavatta, S. Viciani, and M. Bellini, "Quantum-to-Classical Transition with Single-Photon-Added Coherent States of Light", in: *Science* 306.5696 (Oct. 2004), pp. 660–662, ISSN: 1095-9203, DOI: 10.1126/science.1103190.

[ZZJ20] W. Zhang, H. Zhang, and G. Jin, "Adaptive-sampling angular spectrum method with full utilization of space-bandwidth product", in: *Opt. Lett.* 45.16 (Aug. 2020), p. 4416, ISSN: 1539-4794, DOI: 10.1364/ol.393111.

Notation

(a, b)	Open interval over the reals, i.e. $\{x \in \mathbb{R} \mid a < x < b\}$.
$[a, b]$	Closed interval over the reals, i.e. $\{x \in \mathbb{R} \mid a \leq x \leq b\}$.
$\mathcal{T}_{[a,b]}(\mu, \sigma^2)$	Truncated normal distribution i.e. normal with mean μ, variance σ^2, truncated to the interval $[a, b]$.
$\delta(x)$	Dirac delta distribution, (loosely) defined such that $\int_{-\infty}^{\infty} f(x)\delta(x)\,\mathrm{d}x = f(0)$.
δ_i	One-hot vector of length N, $[\delta_{i1}, \delta_{i2}, \ldots, \delta_{iN}]$, i.e. only non-zero entry is at index i.
δ_{ij}	Kronecker delta $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$.
$\mathbb{E}_{\mathrm{x}\sim p}[f(x)]$	Expectation of $f(x)$ under sampling from distribution p.
$\mathcal{F}[f(x)](k)$	Fourier transform of $f(x)$.
$\langle \mathrm{x} \rangle$	Expected value of x.
$\mathbf{1}_A(x)$	Indicator function, given as $\mathbf{1}_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$.
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2.
$\mathcal{U}(a, b)$	Uniform distribution over range $[a, b]$.
$\mathrm{CSE}(a, b)$	Cosine error, i.e. $\frac{\langle a,b \rangle}{\lvert a \rvert \lvert b \rvert}$ for a suitably defined inner product $\langle \cdot, \cdot \rangle$.
$\odot$	The Hadamard product (i.e. element-wise multiplication of two matrices, vectors or tensors).
$\mathbb{R}^{d_1 \times \cdots \times d_n}$	Space of real tensors with shape $(d_1, \ldots, d_n)$.
$\sim$	Distributed as, i.e. $\mathrm{x} \sim p$ indicates that the random variable x is distributed according to probability distribution p.

Abbreviations

AGI	Artificial general intelligence
AI	Artificial intelligence
ANN	Artificial neural network
ASM	Angular spectrum method
CCD	Charge-coupled device
CMA-ES	Covariance matrix adaptation evolutionary strategy
CNN	Convolutional neural network
CPU	Central processing unit
CZT	Chirp Z-transform
DDPG	Deep deterministic policy gradient
DFA	Direct feedback alignment
DFT	Discrete Fourier transform
DMD	Digital micro-mirror device
DNN	Deep neural network
DRL	Deep reinforcement learning
ELBO	Evidence lower bound
ELM	Extreme learning machine
EM	Expectation-maximisation
ESN	Echo state network
ESs	Evolutionary strategies
FA	Feedback alignment
FFT	Fast Fourier transform
GA	Genetic algorithm
GD	Gradient descent
GPGPU	General-purpose GPU
GPU	Graphics processing unit
HOM	Hong-Ou-Mandel
KL	Kullback-Leibler
LCoS	Liquid crystal on silicon
LON	Linear optical network
LoRA	Low-rank adaptation
LSM	Liquid state machine
LSTM	Long short-term memory

MC	Monte Carlo
MDP	Markov decision process
ML	Machine learning
MLP	Multilayer perceptron
MMF	Multimode fibre
MPC	Model predictive control
MPLC	Multi-plane light conversion
MPO	Maximum *a posteriori* policy optimisation
MSE	Mean squared error
NDE	Neural differential equation
NMS	Neuromorphic system
PAT	Physics aware training
PNL	Physical neural layer
PNN	Physical neural network
PNR	Photon number resolved
PNRD	Photon number resolved detection
POMDP	Partially observable Markov decision process
QML	Quantum machine learning
QNN	Quantum neural network
QRC	Quantum reservoir computer
RC	Reservoir computer
RL	Reinforcement learning
Rm-ES	Rank-m evolutionary strategy
RNN	Recurrent neural network
RSC	Rayleigh-Sommerfeld convolution
SAC	Soft actor-critic
SGD	Stochastic gradient descent
SLM	Spatial light modulator
SLOS	Strong linear optical simulator
SNSPD	Superconducting nanowire single-photon detector
SPDC	Spontaneous parametric down-conversion
SV	Singular value
TD	Temporal difference
TPU	Tensor processing unit