UNIVERSITY
OF OSLO

Master's thesis

Deep Learning Methods for Quantum Many-body Systems

A study on Neural Quantum States

Daniel Haas Becattini

Computational Sciences: Physics
60 ECTS study points

Department of Physics
Faculty of Mathematics and Natural Sciences

Autumn 2024

Daniel Haas Becattini

Deep Learning Methods for Quantum Many-body Systems

A study on Neural Quantum States

Abstract

The theoretical understanding behind the principles of quantum mechanics is just the first step towards solving quantum problems in real-world scenarios. Recently, a promising new field has emerged at the intersection of quantum variational methods and machine learning. This approach, named Neural Quantum States, takes advantage of neural networks as universal approximators to efficiently parametrise quantum systems.

This thesis explores the intersection of quantum many-body problems and machine learning, focussing on the application of neural network architectures to solve challenging quantum systems. We specifically investigate three methods: a standard variational Monte Carlo (VMC) parametrisation, a restricted Boltzmann machine (RBM), and a Deep Set feed-forward network (DSFFN). We applied these ansätze to two bound fermionic systems: a one-dimensional polarised fermionic system with Gaussian finite-range interaction, up to six particles and a two-dimensional quantum dots system with Coulomb interaction, up to 20 particles. The study employs Slater-Jastrow variants for the ansätze to impose fermionic antisymmetry, correlations, and cusp conditions. We experimented with various machine learning optimisation techniques, including an extensive Bayesian hyperparameter search, several well-known machine learning optimisers, and a stochastic reconfiguration, a quantum analogue of the natural gradient optimiser, known for its advantage in capturing the geometry of the quantum landscape.

Our implementation is based on Python, aiming at building a modern, modular framework with reasonable efficiency provided by using JAX as back-end. We compare the performance of these methods, discussing their strengths and limitations in representing quantum states efficiently. Our computational efficiency evaluation revealed reasonable scalability with JAX as back-end, though not matching C++ implementations as expected.

We successfully obtained ground-state energies, energy components, density profiles, and correlation energies for quantum systems replicating the results of other research studies with occasional better accuracy. Our results for 1D systems consistently surpassed Hartree-Fock and matched small-basis CI calculations. For 2D quantum dots, the addition of correlation factors repeatedly yielded lower energies than Hartree-Fock, approaching DMC calculations and even surpassing them in one instance using Stochastic Reconfiguration. We addressed the challenges of training neural networks in this unsupervised manner, highlighting the importance of reference energy values and correlation factors. Although more expressive models such as DSFFN showed excellent results, they were harder to train, requiring a balance between model complexity and user intuition.

Acknowledgements

This thesis is what brought me to Oslo, but my friends are the reason I stay. I dedicate this work to all my friends and family. To those who started to love me because I came into their life and those who did not stop despite me leaving. None of this would be possible without the support of my family. None of this would be pleasant without the presence of my friends.

I would especially like to thank my supervisor, Morten Hjorth-Jensen, for all the guidance and support. Your enthusiasm and trust in whatever path I wanted to explore kept me motivated from the first day of this project onwards.

A special thank you to Nigar Abbasova, Adam Jakobsen, and Leah Hansen, for being the best second family I could have asked for. To Anna Aasen and Jonny Aarstad Igeh for taking me into the group and making me feel like I belonged somewhere, and to Hkon Kvernmoen, who, on top of that, endured my non-sensical questions about quantum mechanics, convincing me they were in fact not trivial. Finally, to everyone at CCSE: office mates, lunch companies, and table tennis rivals, thank you.

Contents

Chapter 1

Introduction

1.1 The Quantum Many-Body Problem

Quantum mechanics stands as one of the most significant scientific achievements of the twentieth century, impacting both technological developments and our understanding of reality through its precise theoretical formalism. Despite substantial progress since the early 1920s, applying quantum mechanics to practical problems remains challenging, particularly for quantum many-body systems. These systems are central to various fields of physics, including condensed matter physics, quantum chemistry, atomic, molecular, and nuclear physics.

The core concepts of non-relativistic quantum mechanics are well condensed in the Schrödinger equation. However, the theoretical framework is only part of the challenge, and modelling even medium-sized interactive systems is a task of significant computational difficulty. This is because the dimension of a quantum system, or the Hilbert space, scales exponentially, making an exact solution of the Schrödinger equation for many-body systems unattainable. Though some classical systems also lack analytical solutions, this analysis focuses on the scaling of dimensionality, which is significantly more challenging in quantum systems.

With the advent of computers, the focus shifted towards solving the Schrödinger equation computationally and developing approximation methods to tackle larger systems. A brief description of some of the classical methods for approximating solutions follows: the Hartree-Fock (HF) method [90], which provides a reasonably efficient first approximation, neglects higher-order electron correlations by considering only a mean-field picture. Full Configuration Interaction (FCI) [90] offers exact solutions within a given basis set but is computationally infeasible for large systems due to exponential scaling. Truncated Configuration Interaction (CI) methods [86] account for only some set of excitations, balancing accuracy and computational feasibility.

There are additional methods to include, but despite their effectiveness, they all face limitations in either accuracy or scalability. Configuration Interaction methods, for example, are currently restricted to around 20 particles [94]. Both the memory required to store the Hamiltonian matrix and the computational cost of diagonalisation scale in such a manner that exceeds the capabilities of current computational resources.

In a simplistic way, solving the many-body problem means either learning to represent the high-dimensional quantum states in a more efficient way and/or learning to sample observables from them. One prominent method for specifically doing the latter is quantum Monte Carlo, which is a conceptual cornerstone of our work. Under this label, we mention variational Monte Carlo (VMC) and diffusion Monte Carlo (DMC) [5]. The former optimises a parameterised guess for the wavefunction, using the variational principle to minimise the energy and obtain the ground-state. If that state is obtained, other observables can be similarly sampled. Variational Monte Carlo provides a flexible approach to incorporate complicated wavefunctions, and while it has the potential to be unbiased, its accuracy depends on the chosen ansatz. Diffusion Monte Carlo improves upon VMC, offering higher accuracy at the cost of greater computational

complexity and a more heavy bias from a necessary fixed-node approximation. Therefore, it is crucial to carefully select the functional form and the number of parameters. Even if the correct representation is found, the computational cost of optimising the ansatz increases with the number of parameters. The challenge lies in how to effectively represent quantum states using parameterised functions, and machine learning offers one approach to this problem.

1.2 The Many-Body Problem as a Machine Learning Task

The field of machine learning (ML), since its early days, has similarly suffered with dimensionality-exploding problems, in what is referred to as curse of dimensionality. This is because high-dimensional data are incredibly sparse, so the amount of data required for a model to make accurate predictions to unseen data grows exponentially.

Tensor networks (TN), conceptualised in 1971, were used in quantum many-body problems in the 1990s [96] as perhaps the first crossover between machine learning and many-body physics. Although not developed for machine learning purposes, the connections between TN and ML are now better understood. The idea was to use area-law entanglement to build a network-like ansatz with a polynomial number of parameters. TNs further allowed for symmetries on the wavefunction to be enforced, but these methods did not generalise well beyond one-dimensional systems.

In 2017, Carleo and Troyer [12] proposed a new type of parametrised ansatz based on a restricted Boltzmann machine, a stochastic generative neural network specifically aimed at unsupervised learning. Their approach was coined the term neural quantum states (NQS), and its success led to an investigation between tensor networks and the newly defined neural quantum states. We now know that neural quantum states display the same or higher expressive power than practical tensor networks, while being more efficient [85]. This is because RBMs have been shown to obey volume-law scaling, which allows them to represent quantum many-body states irrespective of their level of entanglement [18].

RBMs are universal approximators [54], which led researchers to question if other universal approximators could equally represent quantum states with a small number of parameters. Various neural network architectures, such as convolutional neural networks, feed-forward neural networks, and graph neural networks, among others, have been shown to answer this positively [52]. Certain NQS architectures have shown the capability to surpass classical techniques such as coupled clustering for specific systems [77], while other approaches outperform state-of-the-art DMC calculations [45]. Neural networks have also been used to accurately model excited states [76], and even more recently to approximate the time evolution of quantum systems with great results [53]. This evidence further points to neural networks as a promising path to solve the quantum many-body problem.

This thesis aims to explore this recent intersection between quantum many-body problems and machine learning. To do so, we compare how three methods perform in two bound fermionic systems: a one-dimensional fully polarised fermion system and a two-dimensional quantum dots system. The models tested were a standard VMC parameterisation, an RBM implementation, and a specific variant of a feed-forward network. We further explored the use of Slater-Jastrow variants for the ansätze to impose fermionic antisymmetry and correlations. The implementation was carried out using Python with JAX, inspired by their recent efficient use in quantum many-body problems [93, 46].

1.3 Overview and Thesis Structure

Trying to address the presented many-body problem bears resemblance to some of the methods that we are going to address in this thesis. It is like an optimisation problem, where we try to minimise how wrong we are in our description of nature. Here, we have a space of theories or

approaches, which is clearly infinite and which we test on a reference frame of arbitrary physical systems. This exploration process is non-convex, and moving in one direction brings advantages, accompanied by some disadvantages.

This thesis is not unique in its topics. Previous works have approached the same systems and have used similar methods. However, I tried to make this work personal in its didactic aspect. This means that theory is presented in a way that feels natural to me and connects points that I consider fundamental. I have tried to include here the answer to every theoretical question I asked myself along the way, with the amount of detail and rigour that at the time was sufficient to understand concepts and, eventually, move on. If a certain approach seems too big of a detour, I was probably connecting dots that for me are enlightening. If it seems too trivial, I was probably covering a gap in my knowledge. With this comes an inevitable loss in linearity of narrative, and that I tried to make evident when possible. On that note, the structure of the thesis is presented below.

In the theory part, Chapter 2, provides a review of basic quantum mechanics, followed by a discussion of quantum many-body physics, and a description of the systems examined in this study. Chapter 3 focusses on the computational background, from the theory of Markov chains, its connections with variational Monte Carlo, and diffusion Monte Carlo. Last in the theory section, Chapter 4 starts with a theoretical treatment of statistical learning and optimisation methods. We proceed by detailing the neural networks used and finish by connecting VMC with reinforcement learning and neural quantum states.

With regard to methodology, Chapter 5 provides an in-depth look at our implementations. We explain our codebase and the rationale behind certain decisions, with a minimal theoretical treatment. Further, we show how different components are integrated and how investigations were conducted.

The results are presented in Chapters 6 and 7, along with discussions and a critical evaluation. Lastly, in Chapter 8, we assess whether the study goals were achieved and summarise the conclusions with the possibility of improvements and future research directions.

Part I

Theory

Chapter 2

Quantum Physics Background

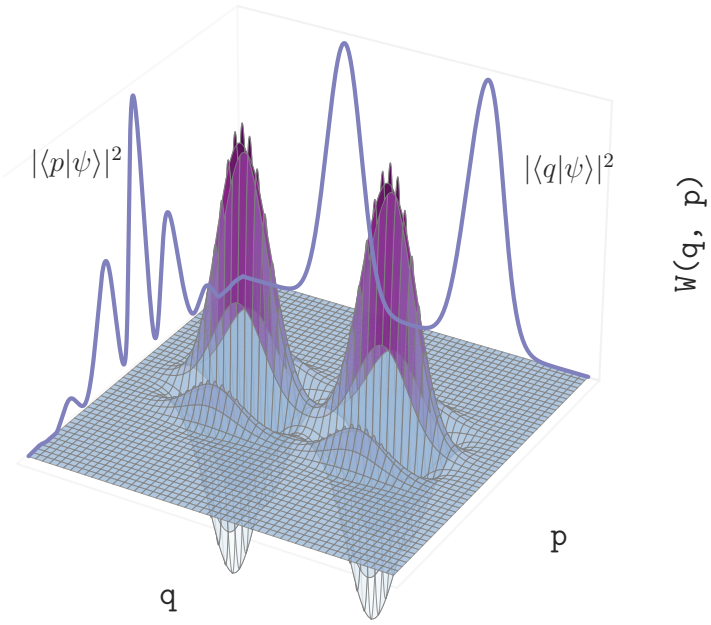


Figure 2.1: Wigner quasi-probability function W as function of position q and momentum p.

2.1 Basic Linear Algebra Background

Dirac Notation and Completeness

We start with a basic review of linear algebra for two main reasons. First, we want to emphasise that from the beginning we are dealing with complex n-dimensional vector spaces, sub-spaces of $\mathbb{C}^n$, so that an element $\boldsymbol{z} \in \mathbb{C}^N$ can be represented as an n-tuple $(z_1, z_2, ..., z_n)$ where z_i is a complex number. Second, we want to migrate to the standard Dirac notation as soon as possible, showing what it means and why we are doing it.

Using the Dirac notation, we represent a standard basis for the vector space $\{e_i\}$ with ket vectors, written as $\{|i\rangle\}$, with $i \in \mathbb{N}$. A generic vector $|a\rangle$ in this space can be decomposed in terms of a complete basis as

$$|a\rangle = \sum_i |i\rangle\, a_i = \sum_i |i\rangle\, \langle i|\, |a\rangle\,.$$

In the second equality, the completeness of the basis is apparent, which can be represented as

$$I = \sum_i |i\rangle \langle i| ,$$

with I being the identity matrix. This completeness relation is crucial in quantum mechanics because it guarantees that the basis spans the entire vector space, allowing the decomposition of a general $|a\rangle$.

Operators and Adjoint Operators

A linear operator $\hat{\mathcal{O}}$ is a mathematical object that maps one vector to another. We say it "acts" on a vector $|a\rangle$, returning another vector, as in $\hat{\mathcal{O}} |a\rangle = |b\rangle$. We can also define the adjoint of this operator as the operator that maps the dual [14] of vector $|a\rangle$ to the dual of vector $|b\rangle$. In Dirac notation, this can be written $\langle a| \hat{\mathcal{O}}^\dagger = \langle b|$. Furthermore, in quantum-mechanical contexts, the dual of the ket vector is called a bra vector, and we denote it $\langle a| = |a\rangle^\dagger$. This way, an inner product is written $\langle a| |b\rangle \coloneqq \langle a|b\rangle$.

For an operator to be called linear, it must obey the following linearity property:

$$\hat{\mathcal{O}}(x |a\rangle + y |b\rangle) = x\hat{\mathcal{O}} |a\rangle + y\hat{\mathcal{O}} |b\rangle ,$$

where x, y are scalars. One fundamental aspect of linear operators is that, given a choice of a basis $\{|i\rangle\}$, we can represent them as matrices. Thus, the linear transformation can be viewed as a matrix-vector multiplication:

$$\hat{\mathcal{O}} |i\rangle = \sum_j |j\rangle O_{ji}.$$

To obtain the matrix elements of the matrix representation of the operator $\hat{\mathcal{O}}$ we can consider the inner product of different basis vectors:

$$O_{ki} = \langle k| \hat{\mathcal{O}} |i\rangle = \sum_j \langle k|j\rangle O_{ji},$$

where $|k\rangle$ and $|j\rangle$ are basis vectors in the vector space on which $\hat{\mathcal{O}}$ acts. The subsequent application of linear operators can similarly be expressed as subsequent matrix multiplications, which means that if $\hat{\mathcal{C}} = \hat{\mathcal{A}}\hat{\mathcal{B}}$, then

$$C_{ij} = \sum_k A_{ik} B_{kj}.$$

From that it is clear that $\hat{\mathcal{A}}\hat{\mathcal{B}} \neq \hat{\mathcal{B}}\hat{\mathcal{A}}$ in general. This distinction can seem obvious, but deserves special attention for its implications in quantum mechanics. Let us define the commutator of operators:

$$[\hat{\mathcal{A}}, \hat{\mathcal{B}}] = \hat{\mathcal{A}}\hat{\mathcal{B}} - \hat{\mathcal{B}}\hat{\mathcal{A}}.$$

When the commutator of two operators is zero, they are said to commute with each other.

Joining the Dots

[a]When two operators commute,

- the sequence of measurements does not affect the result;
- it is possible to find states that are eigenstates for both operators, meaning we can know the outcome for both measurements precisely and simultaneously;
- if an operator commutes with a special operator we call the Hamiltonian ($\hat{H}$), it can be associated with a conserved physical quantity.

[a]Since we are assuming some prior knowledge, we introduce these boxes, in which we connect what is presented with not yet defined concepts. This is a clear compromise in linearity of narrative to make the reading more interesting for those with prior knowledge.

Additional Matrix Properties

Linear algebra is the language of quantum mechanics. As such, there are several properties of matrices that need to be mentioned, even if they seem to appear with no motivation for now.

- **Hermitian Matrices:** If a matrix A is such that $A^\dagger = A$, it is called Hermitian. In quantum mechanics, this is relevant because every observable can be represented by a Hermitian operator.
- **Unitary Matrices:** If a matrix A is such that its inverse is equal to its adjoint ($A^{-1} = A^\dagger$) it is called a unitary matrix. This is relevant because unitary operators preserve the inner product ($\langle Ax, Ay\rangle = \langle x, y\rangle$) and vector norms. Consequently, they preserve probability amplitudes.
- **Orthogonal Matrices:** A unitary matrix with real elements is called orthogonal. They are important because they represent transformations that do not break the orthogonality of vectors - a crucial point in coordinate transformations.

2.1.1 Hilbert Spaces

Quantum and classical mechanical states are different conceptual objects. Unlike a classical state, a general quantum state **needs** to be represented as elements of a vector space known as a Hilbert space. In Newtonian mechanics, the vector spaces used are Euclidean. In contrast, Hilbert vector spaces have some major differences.

First, Euclidean vector spaces are vector spaces over the real numbers, while Hilbert spaces are more generally defined over the complex field. Additionally, Euclidian spaces are finite-dimensional, whereas Hilbert spaces generalise the idea of an inner product to infinite dimension. Finally, Hilbert spaces require us to be more precise with the concept of sequence convergence. In particular, a Hilbert space must be Cauchy-complete. Intuitively, this means that every sequence that "seems like" it should converge (to another element of the space) in fact does.

This third difference is more subtle because we get it for free in Euclidian spaces [80], and thus, it does not often cross our minds - Euclidian spaces inherit Cauchy completeness from the real numbers.

We do not intend to be rigorous in our definition of a Hilbert space, and a complete discussion is found in [80]. We care about the convergence of sequences because we want to use calculus in this vector space, and, therefore, limits have to be well defined. Furthermore, we often want to expand quantum states as an infinite sum of eigenvectors of an observable, again requiring a precise definition of limits.

2.1.2 From Euclidian Vectors to Functions

In this section, we transition from Euclidian vectors to function spaces. Some function spaces can also be vector spaces, so we must define analogous properties for this type of space to be mathematically consistent. Using the previous formalism, we can express the orthonormality of basis vectors as

$$\langle i|j\rangle = \delta_{ij} \coloneqq \begin{cases} 1, & i = j \\ 0 & \text{else,} \end{cases}$$

where δ_{ij} is called the Kronecker delta.

Orthogonality in Function Spaces

Function spaces, however, may have an infinite-dimensional basis, which means that they require an infinite number of basis functions to span the space. Because of that, the inner product becomes an integral. For Euclidian vectors, operations combine point coordinates via some rule, such as addition. However, the image of a function "points to" many points at once, possibly infinitely many. Therefore, the idea of an inner product is naturally extended by the integral over a function multiplied by its dual.

Consider the (enumerable) set of functions $\{\psi_i : I \to \mathbb{C}\}_{i\in\mathbb{N}}$ with I a closed interval. The orthonormality condition now reads

$$\langle \psi_i|\psi_j\rangle = \int_I dx \psi_i^*(x)\psi_j(x) \ = \delta_{ij}.$$

This is not that different from what we had before. Instead of having the product weighted by the vector coefficients, we have it weighted by the function distribution.

Completeness and Function Expansion

In a complete function space, we can expand any general function $a(x)$ on a complete and discrete basis $\{\psi_i : I \to \mathbb{C}\}_{i\in\mathbb{N}}$

$$a(x) = \sum_i \psi_i(x) a_i,$$

where a_i is the i-th component of a in the basis. A general component a_j can be calculated by projection of the function onto the basis

$$a_j = \sum_i \delta_{ji} a_i = \sum_i \langle \psi_j|\psi_i\rangle \, a_i = \sum_i \int dx \, \psi_j^*(x)\psi_i(x)a_i = \int dx \, \psi_j^*(x)a(x).$$

From this we can reconstruct $a(x)$ from its components:

$$a(x) = \sum_j \psi_j(x) \int dy \, \psi_j^*(y)a(y) = \int dy \, \left[\sum_j \psi_j(x)\psi_j^*(y)\right] a(y) = \int dy \, \delta(x-y)a(y).$$

Here, $\delta(x-y)$, which we dare not call a function, is the Dirac delta and can be seen as a continuous analogue of the Kronecker delta. All the similarities to the previous Euclidian vector space formalism might now become obvious, and the discussion about operators is also applicable.

So far we have omitted an important detail. Note that $|\psi_i\rangle$ represents the complete state in the Hilbert space, while the wave function $\psi_i(x)$ is its projection into position space, which means it is the amplitudes of such a state in the eigenstates of the position operator $\hat{X}$. In that sense, $\psi_i(x) \coloneqq \langle x|\psi_i\rangle$.

Operators on Function Spaces

As eluded to before, in quantum mechanics, observables can be represented as linear operators. The action of a linear operator $\hat{\mathcal{O}}$ on a function $a(x)$ yields another function $b(x)$. By the completeness relation, the action of this operator can be defined by how it acts on a complete basis $\{\psi_i\}$. We then write the operator in matrix notation O_{ij}, and it follows

$$\begin{aligned} b(x) &= \hat{\mathcal{O}}a(x) \\ &= \sum_i b_i(x) = \sum_i \sum_j O_{ij} a_j(x). \end{aligned}$$

The matrix elements of the operator, O_{ji}, can be computed by

$$O_{ji} = \int dx\, \psi_j^*(x) \hat{\mathcal{O}} \psi_i(x).$$

Joining the Dots

Note that the matrix representation of the operator is infinite. As will become clear in Sec. 2.2, all this discussion about linear algebra is largely so that we have the appropriate toolbox to solve eigenvalue problems of the type $\hat{\mathcal{O}}\phi(x) = \omega\phi(x)$ under some basis $\{\psi_i\}_{i\in\mathbb{N}}$.

It is also possible to express the (continuous) matrix elements of the abstact operator $\hat{\mathcal{O}}$ in a continuous basis as expected:

$$\langle x| \hat{\mathcal{O}} |y\rangle = O(x, y).$$

2.1.3 Combining Spaces

Given two vector spaces V and W, it is useful to understand which space results from combining them.

One way to combine spaces is through a tensor product. We write the tensor product between the aforementioned spaces as $U = V \otimes W$. For this combination to yield a vector space, we need to be able to map any pair of $(\boldsymbol{v} \in V, \boldsymbol{w} \in W)$ via a linear map to an element $(\boldsymbol{v} \otimes \boldsymbol{w}) \in U$. The element $\boldsymbol{v} \otimes \boldsymbol{w}$ is called the tensor product of $\boldsymbol{v}$ and $\boldsymbol{w}$.

If B_V if a basis of V and B_W if a basis of W, the basis of $V \otimes W$ is the set $\{v \otimes w | v \in B_V, w \in B_W\}$. Note that the dimension of the resulting space is the product of the dimension of the original spaces.

Direct sums usually arise in vector spaces from the opposite problem of decomposing a vector space U into smaller sub-spaces. In this case, it can be decomposed, for example, into the direct sum $V \oplus W$ if every vector $\boldsymbol{u} \in U$ can be expressed as $\boldsymbol{u} = \boldsymbol{v} + \boldsymbol{w}$ with $\boldsymbol{v} \in V$ and $\boldsymbol{w} \in W$, and $V \cap W = \{0\}$.

Joining the Dots

If not for tensor products, we would not be able to represent the entangled states of a composite system in quantum mechanics.

2.2 Basic Quantum Physics

Quantum states are vectors in the Hilbert space, or more accurately, equivalence classes of vectors in the Hilbert space. That is because distinct vectors can represent the same observable if they differ only by a constant nonzero factor λ. Hence, we say they represent the same state $|\Psi\rangle \sim \lambda |\Psi\rangle$. Because λ is simply a scaling constant, we call this equivalence class a "ray".

2.2.1 Why Schrödinger?

In this work, we do not deal with time-dependent systems. Nevertheless, for completeness and a better understanding of the basics of quantum mechanics, it is relevant to give a simple and not rigorous motivation for the time-dependent Schrödinger equation (TDSE). Our isolated system, described by a state $|\Psi\rangle$ in Hilbert space, can generally depend on time and space. As a time evolution should not remove the state from the Hilbert space, we can relate states at different times via some linear time-evolution operator,

$$|\Psi(t)\rangle = \hat{\mathcal{O}}(t) |\Psi(t_0)\rangle . \tag{2.1}$$

If we examine the action of such operator under an infinitesimal transformation, we can take the Taylor expansion to first order, in which case

$$|\Psi(t_0 + dt)\rangle = \hat{\mathcal{O}}(dt) |\Psi(t_0)\rangle \approx (1 + \Lambda dt)) |\Psi(t_0)\rangle .$$

Rearranging and taking limits leads to the definition for the temporal derivative of the wave function

$$\left.\frac{d |\Psi(t)\rangle}{dt}\right|_{t_0} = \Lambda |\Psi(t_0)\rangle .$$

The action of the Λ operator is still unspecified, but by examining some physics properties, we can extract information about it. By Noether's Theorem [68], for example, we know that time translational invariances lead to the conservation of energy. This forces Λ to be proportional to the Hamiltonian operator ($\Lambda = \zeta H$). Furthermore, partial derivatives are anti-Hermitian and we know anti-Hermitian operators only have purely imaginary eigenvalues. This means that ζ must be imaginary, because the Hamiltonian being Hermitian admits only real eigenvalues. We shall not show it, but it is at least reasonable from a dimensional analysis that the proportionality constant resolves to $\zeta = -i/\hbar$, with $\hbar$ the reduced Planck constant, leading to the time-dependent Schrödinger equation:

$$i\hbar \frac{\partial}{\partial t} |\Psi\rangle = \hat{H} |\Psi\rangle .$$

Since $\hbar$ is just a constant, we often change our scales so that $\hbar = 1$. This reasoning for informally obtaining the Schrödinger equation might seem problematic. Usually we start from a differential equation and move towards a solution, but Eq. 2.1 makes it seem like we somehow start from a solution. This is not the case, as $\hat{\mathcal{O}}$ is not known. However, we expect $\hat{\mathcal{O}}$ to be unitary in order to preserve probability amplitudes. Furthermore, such an operator must have an exponential representation because we expect that, for any two points in time, $\hat{\mathcal{O}}(t_1)\hat{\mathcal{O}}(t_2) = \hat{\mathcal{O}}(t_1 + t_2)$. It can be easily verified that

$$|\Psi(t)\rangle = e^{-i\hat{H}t} |\Psi(0)\rangle \tag{2.2}$$

is a solution for the TDSE.

By the spectral theorem on self-adjoint operators, eigenstates of the Hamiltonian indeed form a complete basis for the Hilbert space and hence we can expand the solution $|\Psi(t)\rangle$ in

that basis. In its basis, of course, the Hamiltonian $\hat{H}$ is diagonal. Additionally, because of the systems with which we will be dealing, and the computational scope of this work, we assume a discrete set of eigenvalues E_n on its main diagonal. Indeed, we want to search for a basis in which the eigenvalue equation

$$\hat{H} |\psi_n\rangle = E_n |\psi_n\rangle , \tag{2.3}$$

becomes trivial. This equation, called the time-independent Schrödinger equation (TISE), leads to a very clean description of the time evolution in quantum mechanics. The expansion of $|\Psi(t)\rangle$ in this basis can be writen

$$|\Psi(t)\rangle = \sum_n c_n |\psi_n(t)\rangle = \sum_n c_n e^{-iE_n t} |\psi_n\rangle .$$

2.2.2 Observables and Operators

In quantum mechanics, observables can only be discussed statistically, and they are computed by taking the expectation value of the associated operator acting in a state. Let $\hat{\mathcal{O}}$ be the operator, we represent this average as

$$\langle \hat{O} \rangle = \langle \psi | \hat{O} | \psi \rangle . \tag{2.4}$$

As discussed in Sec. 2.2.1, the wavefunction can have a time dependence, $|\psi(t)\rangle$, giving rise to a time dependence on the statistical value $\langle \hat{O} \rangle_t$. The time dependence can be seen as coming from the state vector or the operator. The former case is called the Schrödinger picture, and the latter, the Heisemberg picture - both equivalent. Indeed, the solution for the TDSE in Eq. 2.2, allows us to write

$$\langle \hat{O} \rangle = \langle \psi_0 | e^{Ht} \hat{O} e^{-iHt} | \psi_0 \rangle .$$

Consequently, we can interpret the state as time independent but evolving in time due to the action of time-dependent operators, in which case we write

$$\mathcal{O}_H(t) = e^{Ht} \hat{O} e^{-iHt}.$$

For pure states, yet to be defined, Eq. 2.4 can be expanded in the operator's basis $|\phi_i\rangle$ as

$$\langle \psi | \hat{O} | \psi \rangle = \sum_i c_i \langle \psi | \phi_i \rangle \langle \phi_i | \psi \rangle = \sum_i c_i | \langle \phi_i | \psi \rangle |^2$$

where c_i are the coefficients of the expansion. $| \langle \phi_i | \psi \rangle |^2$ then represents the probability that $|\psi\rangle$ is measured in state $|\phi_i\rangle$.

2.2.3 Composite Systems

To talk about many-body quantum systems, we need to talk about composite systems: systems composed of more than one quantum object. Interestingly, the description of composite systems varies depending on whether the subsystems are distinguishable or not. If they are, their description is given by the following postulate, which we quote from [67]:

Postulate

"The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through

> n, and system number i is prepared in state $|\psi_i\rangle$, then the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$".

In fact, the number of dimensions in the state space can be understood as the number of distinct configurations a system can adopt, which makes the formalism of the tensor product evident. If systems i and j are each spanned by a respective basis $\{|\psi_i\rangle\}_{i\leq N}$ and $\{|\phi_i\rangle\}_{i\leq M}$, then we can describe the composite system by a state space with basis

$$\{|\psi_i\phi_j\rangle : 1 \leq i \leq N, 1 \leq j \leq M\}.$$

The tensor product concatenates the vector spaces that describe the system. This parallel with degrees of freedom also appears when we consider the Hilbert space with spin and position degrees of freedom, in which case $\mathcal{H}_{total} = \mathcal{H}_{position} \otimes \mathcal{H}_{spin}$.

The fact that the tensor product is not surjective motivates entanglement and also allows for a correct description of composite systems. In Schrödingers' words [83], "Maximal knowledge of a total system does not necessarily include total knowledge of all of its parts, not even when these are fully separated from each other and at the moment are not influencing each other at all.". The tensor product formalism allows the description of purely quantum behaviour.

There are states in the composite Hilbert space $\mathcal{H} = \mathcal{H}_\alpha \otimes \mathcal{H}_\beta \otimes \cdots \otimes \mathcal{H}_\zeta$ that are not accessible by one tensor product of any n states $|\alpha\rangle \otimes |\beta\rangle \otimes \cdots \otimes |\zeta\rangle$. This is clearly the case for entangled states but also for composite states of identical particles, as we shall see in Sec. 2.3. In general, a pure state $|\Psi\rangle$ from the total Hilbert space $\mathcal{H}$ above is written as

$$|\Psi\rangle = \sum_{ij\cdots n\cdots} c_{ij\cdots n\cdots} |\alpha_i\rangle \otimes |\beta_j\rangle \otimes \cdots \otimes |\gamma_n\rangle \otimes \cdots \coloneqq \sum_{ij\cdots n\cdots} c_{ijk\cdots n\cdots} |\alpha_i\beta_j\cdots\gamma_n\cdots\rangle\,, \tag{2.5}$$

where the coefficients in Latin index are free to run from 1 to the dimension of the respective Hilbert space they refer to. For example, i is attached to a state from $\mathcal{H}_\alpha$, so α_i could be any of the elements of its basis. For the state in Eq. 2.5 to be physical, we need to ensure normalisation,

$$\sum_{ij\cdots n\cdots} |c_{ijk\cdots n\cdots}|^2 = 1,$$

but also a specific symmetry, if the composite system is of indistinguishable particles. This will be better discussed in Sec. 2.3 and briefly introduced in the following Sec. 2.2.3.

A Simplified Discussion on Symmetry

In non-relativistic quantum mechanics, there are two fundamental categories of particles: bosons and fermions. Different types of bosons or different types of fermions can differ in terms of physical quantities such as mass and charge. However, after those physical quantities are set, particles of the same category are completely indistinguishable. This is an empirical fact, rather than something deduced from first principles.

Bosons and fermions have their name based on the different statistics they are perceived to follow, namely Bose-Einstein and Fermi-Dirac statistics. That means that the probability density of their wavefunctions behaves differently under particle exchange. Given a two-particle system, the consequence of such indistinguishably can loosely be illustrated in the wavefunction formalism as

$$|\psi(x_1, x_2)|^2 = |\psi(x_2, x_1)|^2 \implies \psi(x_1, x_2) = e^{i\theta}\psi(x_2, x_1).$$

If we attribute to this "switching of labels" an operator, $\hat{P}_{ex}$, it is reasonable to expect that $\hat{P}_{ex}^2 = I$, where I is the identity operator. Consequently, either $\theta = 0$ or $\theta = \pi$. In the first case,

the particles are bosons and $\psi(x_1, x_2) = \psi(x_2, x_1)$. Else, $\psi(x_1, x_2) = -\psi(x_2, x_1)$, and we say the particles are fermions. This last case leads to Pauli's exclusion principle, where two identical fermions cannot occupy the same quantum state. In this simplified case, $x_1 = x_2 = x$ leads to $\psi(x, x) = 0$, and a zero probability density in any case where the particles have the same position. The exclusion principle is, of course, not limited to position representation and can be applied to any type of quantum states.

2.2.4 Pure States, Mixed States and Density Operators

Pure states are those that can be described by one ray in the Hilbert space and can be expressed by a single ket vector. Not all systems can be represented this way, and there is no specific reason to believe that a general system is in a pure state. Systems that do not fit this description are referred to as mixed systems.

Mixed states can appear in two situations: first, a system could be prepared in a way that is not known by the observer, in which case we represent it as a statistical combination of possible preparations. The other instance is when describing an entagled state, such that there is no definite state prior to measurement.

Since mixed states cannot be described with a single ket vector, they are instead described by density matrices, which encapsulates the probabilistic information of the system. In particular, any system, mixed or pure, can be described by density matrices,

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|. \tag{2.6}$$

For a pure state $|\psi\rangle$, we have $\rho = |\psi\rangle\langle\psi|$. In contrast, in Eq. 2.6 the probabilities p_i are the probabilities of the respective pure state that constitutes the mixed state. The big difference for ρ of pure versus mixed states is that for pure states, $\rho^2 = \rho$, or equivalently, $\mathrm{Tr}[\rho^2] = 1$. In general, for mixed states, $\mathrm{Tr}[\rho^2] \leq 1$ and this quantity $\mathrm{Tr}[\rho^2]$ is defined as the purity of the state.

Expectation Values as Traces of Density Operators

Traces of operators are basis-independent. As we know, the trace of a matrix is the sum of its diagonal elements, but it can also be defined on a general basis $\{|i\rangle\}$, which makes it independent of a matrix representation of the operator.

$$\mathrm{Tr}[\mathcal{O}] \coloneqq \sum_i \langle i| \mathcal{O} |i\rangle$$

Another reason why density operators are useful to bring up now is that they allow us to express expectation values in a natural way. The action of an operator in a quantum state is statistical by nature, so expressing them as expectations values is a necessity. If we have a pure state $\rho = |\psi\rangle\langle\psi|$ and an operator for an observable $\hat{\mathcal{O}}$:

$$\begin{aligned}
\mathrm{Tr}[\rho\hat{\mathcal{O}}] &= \sum_i \langle i| \rho\hat{\mathcal{O}} |i\rangle \\
&= \sum_i \langle i| |\psi\rangle \langle\psi| \hat{\mathcal{O}} |i\rangle \\
&= \sum_i \psi_i \langle\psi| \hat{\mathcal{O}} |i\rangle \\
&= \sum_i \langle\psi| \hat{\mathcal{O}} (\psi_i |i\rangle) \\
&= \langle\psi| \hat{\mathcal{O}} |\psi\rangle .
\end{aligned}$$

$\mathrm{Tr}[\rho\hat{\mathcal{O}}]$ is therefore the expectation value of the observable acting on the state. For the general case of a mixed state with probabilities p_j of being in pure states ψ_j we mention for completeness, and without a proof,

$$\mathrm{Tr}[\rho\hat{\mathcal{O}}] = \sum_j p_j \langle\psi_j| \hat{\mathcal{O}} |\psi_j\rangle .$$

2.2.5 Variational Principle

We have shown that the time-independent Schrödinger equation of Eq. 2.3 is an eigenvalue problem which not only can help us solve the TDSE but has significance on its own. Furthermore, all systems considered throughout this work are stationary.

When solving this eigenvalue problem computationally, we work with finite subsets of $\{\psi_i\}_{i\in\mathbb{N}}$, which we call the computational basis. This is an approximation, since the matrix representation of our Hamiltonian operator $\hat{H}$ can be of infinite dimension, in general. Although the approximate nature of the solution might seem frustrating at first, the variational principle offers an understanding of the quality of this approximation after truncating the basis.

For simplicity, we are considering systems with discrete and potentially infinite energy levels E_n, which can be ordered: $E_0 \leq E_1 \leq \cdots \leq E_n \leq \ldots$. From the orthonormality of the basis functions, it follows

$$\langle\psi_i| \hat{H} |\psi_j\rangle = \delta_{ji} E_j,$$

and an arbitrary state guess $|\phi\rangle$ can be decomposed in this energy eigenbasis:

$$|\phi\rangle = \sum_i c_i |\psi_i\rangle = \sum_i |\psi_i\rangle \langle\psi_i|\phi\rangle .$$

It follows that the expectation value for the Hamiltonian can be written

$$\begin{aligned}
\langle\phi| \hat{H} |\phi\rangle &= \sum_j \langle\phi|\psi_j\rangle \langle\psi_j| \hat{H} \sum_i |\psi_i\rangle \langle\psi_i|\phi\rangle \\
&= \sum_{ij} \delta_{ij} \langle\phi|\psi_j\rangle \langle\psi_j| \hat{H} |\psi_i\rangle \langle\psi_i|\phi\rangle = \sum_i E_i |\langle\psi_i|\phi\rangle|^2 \\
&\geq E_0 \sum_i |\langle\psi_i|\phi\rangle|^2.
\end{aligned}$$

Now, if we further normalise the trial state, we can write:

$$\frac{\langle\phi| \hat{H} |\phi\rangle}{\langle\phi|\phi\rangle} \geq E_0.$$

This is the variational principle, which tells us that the energy expectation value is bounded from below by the energy of the ground-state, for any trial state $|\phi\rangle$. The equality then only holds if the trial state is the ground-state. Consequently, we can use the expectation value of the energy as a measure of the quality of a trial wave function. More generally, the variance of any observable is another valid measure of the quality of the trial state, as any eigenstate necessarily leads to a zero variance in the measured observable. To see this, recall that the variance of an operator is given by $\langle\hat{\mathcal{O}}^2\rangle - \langle\hat{\mathcal{O}}\rangle^2$. So if $|\phi\rangle = \lambda |\psi_n\rangle$,

$$\begin{aligned}
\mathrm{Var}(\hat{H}) &= \frac{\langle\phi| \hat{H}^2 |\phi\rangle}{\langle\phi|\phi\rangle} - \left(\frac{\langle\phi| \hat{H} |\phi\rangle}{\langle\phi|\phi\rangle}\right)^2 \\
&= E_n \langle\psi_n| \hat{H} |\psi_n\rangle - (E_n)^2 \\
&= 0.
\end{aligned}$$

Conceptually, the variational principle is simple. Yet, it is central in several quantum many-body methods, such as Hartree-Fock, variational Monte Carlo, and density functional theory.

Joining the Dots

When approaching the energy minimisation problem computationally, we have a parametrised ansatz in terms of a set of parameters $\{\alpha_i\}$. We then assume a functional form of the ansatz and vary the parameters, storing the ansatz that yields the lowest energy as our "best guess" for the ground-state.

2.3 Many-Body Systems

Here and in the following subsections we show the need for a more robust theoretical toolkit when dealing with multiparticle systems. We motivate and briefly explain some of the most standard methods to address the many-body challenge. We, however, call attention to the fact that the many-body methods here contained are not state-of-the-art approaches and serve merely to show natural pathways one should consider when increasing the complexity of systems.

2.3.1 Why Not Schrödinger?

We have loosely motivated why the Schrödinger equation is used in quantum mechanics and how it explains results that cannot be achieved with classical mechanics. Sadly, when dealing with systems of multiple particles, solving the Schrödinger equation becomes impractical.

Consider an N-particle system, each in a state that can be described in Hilbert space $\mathcal{H}_i$. The Hilbert space containing all the particles is then written as the tensor product of all the spaces,

$$\mathcal{H} = \bigotimes_{i}^{N} \mathcal{H}_i,$$

with a complete basis of this whole space being $\{|\alpha\beta\ldots\omega\rangle\}$. However, not only does a general state need to be normalised, it also has to be symmetric or antisymmetric, as discussed in Sec. 2.2.3. For a simplified two-particle case, with single-particle states α and β, we write

$$|\alpha\beta\rangle_{\pm} = \frac{1}{\sqrt{2}}\left[|\alpha\rangle \otimes |\beta\rangle \pm |\beta\rangle \otimes |\alpha\rangle\right], \tag{2.7}$$

with subscript $\pm$ denoting the correct symmetry. Note how physical states then become extremely rare as the dimension of the Hilbert space increases. To simplify wavefunction expressions slightly, hereafter we will use the composite notation $\mathbf{x}_i \coloneqq (\boldsymbol{r}_i, \sigma_i)$, where $\boldsymbol{r}_i$ represent spatial coordinates and σ_i the spin coordinates.

For fermionic systems, the wavefunction that contains the coordinates $\mathbf{x}_i$ of all particles and obeys the anti-symmetrisation and normalisation constraints can be written

$$\Psi_{-}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \sum_{\pi \in S_N} (-1)^{\mathrm{sgn}(\pi)} \psi_{\pi(1)}(\mathbf{x}_1)\psi_{\pi(2)}(\mathbf{x}_2)\ldots\psi_{\pi(N)}(\mathbf{x}_N), \tag{2.8}$$

Title and formalism of this section were taken from the excellent lecture notes on quantum many-body by Thierry Giamarchi [26].

where the summation involves any permutation π of the symmetric group S_N (the set of all permutations of $\{1, 2, \cdots, N\}$). Here, $\mathrm{sgn}(\pi)$ denotes the signature, or the number of inversions of a permutation. Equation 2.8 is often written as

$$\Psi_-(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \hat{S}_- \sqrt{N!} \prod_i^N \psi_i(\mathbf{x}_i), \tag{2.9}$$

where we defined the general symmetrising operator,

$$\hat{S}_\pm \coloneqq \frac{1}{N!} \sum_{\pi \in S_N} (\pm 1)^{\mathrm{sgn}(\pi)} \hat{\pi},$$

and the permutator opperator, which acts as

$$\hat{\pi} \prod \psi_i(\mathbf{x}_i) \coloneqq \prod \psi_i(\mathbf{x}_{\pi(i)}) = \prod \psi_{\pi(i)}(\mathbf{x}_i).$$

For bosonic systems, there is the possibility that any state is occupied by any number of particles. If we have for example, n_i particles in state $|\alpha_i\rangle$, it follows

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \hat{S}_+ \frac{\sqrt{N!}}{\prod_{j=1}^N \sqrt{n_j!}} \prod_i^N \psi_i(\mathbf{x}_i). \tag{2.10}$$

Joining the Dots

Note that $[\hat{S}_\pm, \hat{\mathcal{O}}] = 0$ for any $\hat{\mathcal{O}}$ hermitian since there are no measurements capable of distinguishing the particles.

Mathematical determinants naturally embed this fermionic antysymmetry of the operator $\hat{S}_-$ with permutation signature. We therefore write Eq. 2.9 as a determinant of the single-particle wave functions, often called the Slater determinant:

$$\Psi_-(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(\mathbf{x}_1) & \psi_1(\mathbf{x}_2) & \ldots & \psi_1(\mathbf{x}_N) \\ \psi_2(\mathbf{x}_1) & \psi_2(\mathbf{x}_2) & \ldots & \psi_2(\mathbf{x}_N) \\ \vdots & & & \\ \psi_N(\mathbf{x}_1) & \psi_N(\mathbf{x}_2) & \ldots & \psi_N(\mathbf{x}_N) \end{vmatrix} \coloneqq \frac{1}{\sqrt{N!}} \det\{\psi_i(\mathbf{x}_j)\}. \tag{2.11}$$

For bosonic systems, in analogy, we use the permanent, albeit with a different normalisation constant. Indeed, fermions cannot occupy the same quantum state, but bosons can, leading to more possible permutations to be considered.

$$\Psi_+(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \sqrt{\frac{\prod_j n_j!}{N!}} \left\{ \begin{matrix} \psi_1(\mathbf{x}_1) & \psi_1(\mathbf{x}_2) & \ldots & \psi_1(\mathbf{x}_N) \\ \psi_2(\mathbf{x}_1) & \psi_2(\mathbf{x}_2) & \ldots & \psi_2(\mathbf{x}_N) \\ \vdots & & & \\ \psi_N(\mathbf{x}_1) & \psi_N(\mathbf{x}_2) & \ldots & \psi_N(\mathbf{x}_N) \end{matrix} \right\} \coloneqq \sqrt{\frac{\prod_j n_j!}{N!}} \mathrm{perm}\{\psi_i(\mathbf{x}_j)\}. \tag{2.12}$$

The first critical observation is that there can be an intractable number of terms in the wavefunction, as determinants and permanents scale with $N!$. Beyond that, this formalism leads to a convoluted way of defining how operators act on a general wavefunction. To see the

latter problem, note how we need to decompose the action of operators in the individual Hilbert spaces of the particles on which they act. For example, as we expect the total momentum $\hat{P}_{tot}$ of a full system to be the sum of the momenta of individual particles, we write

$$\hat{P}_{tot} = \sum_{i}^{N} \bigotimes_{k}^{N} \hat{A}_k, \quad \text{with } A_k = \begin{cases} \hat{P} & \text{if } i = k \\ I_k & \text{otherwise} \end{cases}, \tag{2.13}$$

where $\hat{P}$ is the standard momentum operator, and I the identity. Both operators and wavefunctions depend explicitly on the number of particles in this formalism, which leads to an extremely system-specific algorithmic way of making calculations.

2.3.2 Fock Space

We then look for an alternative formalism to address the challenges encountered when attempting to solve the Schrödinger equation using the conventional method for many-particle systems. The new formalism which we now present is referred to as second quantisation, or occupation number representation.

The intent of number representation is to turn the obstacle of indistinguishability of particles into an advantage. In fact, knowing how many particles occupy each quantum state of the system suffices to characterise it, and we write

$$|\Psi_{\pm}\rangle = |n_1 n_2 \dots n_i \dots\rangle . \tag{2.14}$$

If we can quantise the quantum states, the numbers n_i, from left to right (even in bra states), represent an ordering of the single-particle states following the ordering of eigenvalues of some operator. This state is indeed a vector in a new type of vector space called Fock space. With the appropriate state in this space, we can reconstruct any given wave function.

A general state in this Fock space is a linear combination of product states formed of single-particle states of dimension up to n, meaning

$$|\Psi\rangle_{\pm} = a\,|0\rangle \oplus \sum_i a_i\,|\psi_i\rangle \oplus \cdots \oplus \sum_{ij} a_{ij}\,|\psi_i\psi_j\rangle \oplus \cdots .$$

The key point is to observe that this expression is the same as Eq. 2.14, while Eq. 2.14 uses a more convenient basis, dubbed the occupancy number basis. Note that this is a basis built on the single-particle states under the assumption that such a basis of non-interacting states even exists. Nonetheless, we can write the Fock space as

$$\mathcal{F}_{\pm}(\mathcal{H}) = \bigoplus_{n=0}^{\infty} \hat{S}_{\pm}\mathcal{H}^{\otimes n},$$

where $\mathcal{H}^{\otimes 0} = \mathbb{C}$ represents the zero-particle state and, in general $\mathcal{H}^{\otimes n} = \bigotimes_j^n \mathcal{H}_j$ is a composite Hilbert space of n single particle systems. As mentioned, a general state in this space can be written in the form of the state of Eq. 2.14.

Because of the way the Fock space is constructed via direct sums of Hilbert spaces, two states with a different number of particles are automatically orthogonal. Additionally, in the case where the total number of particles is the same, we can use the wavefunction expression to show

$$\langle m_1 m_2 \dots m_k | n_1 n_2 \dots n_k \rangle = \prod_{i=1}^{k} \delta_{m_i n_i},$$

which means we can have an orthonormal basis, allowing for operators to be written in such basis as well.

Bosons	**Fermions**
$a_i^\dagger a_j^\dagger - a_j^\dagger a_i^\dagger = [a_i^\dagger, a_j^\dagger] = 0$	$a_i^\dagger a_j^\dagger + a_j^\dagger a_i^\dagger = \{a_i^\dagger, a_j^\dagger\} = 0$
$a_i a_j - a_j a_i = [a_i, a_j] = 0$	$a_i a_j + a_j a_i = \{a_i, a_j\} = 0$
$a_i a_j^\dagger - a_j^\dagger a_i = [a_i, a_j^\dagger] = \delta_{ij}$	$a_i a_j^\dagger + a_j^\dagger a_i = \{a_i, a_j^\dagger\} = \delta_{ij}$

Table 2.1: Algebra for identical particles in number representation. Inspired by Sakurai [81].

In general, allowing particles to be in the same state, if we have N particles with coordinates $\mathbf{x}_i$ and γ states

$$\langle \mathbf{x}_1 \dots \mathbf{x}_N | n_1 \dots n_\gamma \rangle = \frac{\sqrt{N!}}{\sqrt{n_1!} \cdots \sqrt{n_\gamma!}} \hat{S}_\pm \left[\prod_{i=1}^{n_1} \psi_1(\mathbf{x}_i) \prod_{i=n_1+1}^{n_1+n_2} \psi_2(\mathbf{x}_i) \cdots \prod_{i=N-\sum_i^{\gamma-1} n_i}^{N} \psi_\gamma(\mathbf{x}_i) \right]. \tag{2.15}$$

2.3.3 Creation and Annihilation Operators

This framework aims to be general with respect to the number of particles. Consequently, it is logical to introduce an operator that can add or subtract particles from the system. These can be understood as generators for the basis of specific Fock spaces. For example, by applying a creation operator $a_i^\dagger$ to the (only) state of the zero particle Fock space (vacuum state), we generate a basis element of the one particle Fock space that corresponds to a single particle in state α_i. Similarly, there is an annihilation operator, a_i, which is the adjoint of the creator operator.

One caveat is that the algebra for these operators have to be different whether the particle they create or annihilate are bosons or fermions. This means that the way they work under the composition of operators need to be different. For example, for bosons, and using Eq. 2.10

$$a_i^\dagger \left| n_1 \dots n_i \dots n_\gamma \right\rangle_+ = \sqrt{n_i + 1} \left| n_1 \dots n_{i+1} \dots n_\gamma \right\rangle_+ ,$$

while for fermions, the antisymmetry requires that

$$a_i^\dagger \left| n_1 \dots n_i \dots n_\gamma \right\rangle_- = (1 - n_i)(-1)^k \left| n_1 \dots n_{i+1} \dots n_\gamma \right\rangle_- ,$$

where k is the number of permutations required to take n_i to the position of n_1 by exchange of neighbour numbers. Similarly, if we want to destroy the particles in state α_i,

$$\begin{aligned} a_i \left| n_1 \dots n_i \dots n_\gamma \right\rangle_+ &= \sqrt{n_i} \left| n_1 \dots n_{i-1} n_{i-1} \dots n_\gamma \right\rangle_+ \\ a_i \left| n_1 \dots n_i \dots n_\gamma \right\rangle_- &= n_i (-1)^k \left| n_1 \dots n_{i-1} n_{i-1} \dots n_\gamma \right\rangle_- \end{aligned}$$

We shall not demonstrate it here, but from the requirements above one can construct the algebra from the creation and annihilation operators in Tab. 2.1 for fermions and bosons. What matters for our purposes is that we have, in the Fock space: a complete basis of single-particle states, operators respecting the commutation rules, and some definition of a vacuum state $|\varnothing\rangle$.

Conceptually, the vacuum state represents no particles occupying no state, namely $|00\dots0\rangle$ and which allows for the generation of any basis of the Fock space in the following sense:

$$\left| n_1 \dots n_i \dots n_\gamma \right\rangle_\pm = \left(\prod_i^\gamma \frac{(a_i^\dagger)^{n_i}}{\sqrt{n_i!}} \right) |\varnothing\rangle .$$

With this, some common sense things also follow: applying the annihilation operator to the vacuum collapses it to zero. Likewise, annihilating the state α_i from a system not containing

such a state or creating a particle in an occupied state also yields zero. Computing expectation values becomes very convenient, given the commutation rules and tools like Wick's theorem. A trivial example for bosons, using the commutation relations, illustrates this:

$$\begin{aligned}\langle\psi|\psi\rangle &= \langle\varnothing|\, a_i a_i^\dagger \,|\varnothing\rangle \\ &= \langle\varnothing|\, 1 - a_i^\dagger a_i \,|\varnothing\rangle = 1.\end{aligned}$$

2.3.4 Operators in Second Quantisation

Given that in the Fock space we pay special attention to single-particle states, it is important to distinguish between operators that act on one particle at a time, two particles at a time, and so on.

The momentum operator is an example of a one-body operator, while the potential operator, when a two-particle interaction is included, is an example of a two-body operator. To make it general in the number of particles and avoid the definition of Eq. 2.13, we write it in second quantisation formalism. We can write the action of a one-body operator $\hat{\mathcal{O}}$ on a one-particle system by expanding the operator in the single-particle basis $|\alpha\rangle = a_\alpha^\dagger |\varnothing\rangle$:

$$\hat{\mathcal{O}} = \sum_{\alpha,\beta} \langle\alpha|\, O \,|\beta\rangle\, |\alpha\rangle\, \langle\beta| = \sum_{\alpha,\beta} \langle\alpha|\, O \,|\beta\rangle\, a_\alpha^\dagger |\varnothing\rangle\, \langle\varnothing|\, a_\beta. \tag{2.16}$$

We can understand Eq. 2.16 as follows: the operator annihilates a particle in the state β and creates a particle in the state α with a "transition probability" modelled by the matrix elements $\langle\alpha|\, O \,|\beta\rangle$. Note that $|\varnothing\rangle\langle\varnothing|$ simply projects whatever state we have after β annihilation to the vacuum. This is because we only had one particle to begin with. In a general state, we write

$$\hat{\mathcal{O}} = \sum_{\alpha,\beta} \langle\alpha|\, O \,|\beta\rangle\, a_\alpha^\dagger a_\beta.$$

Another one-body operator of particular interest to us is the kinetic energy $\hat{K} = \hat{P}^2/2$. In this case it makes sense to use the momentum basis $\{k_i\}$:

$$\begin{aligned}\hat{K} &= \sum_{k_1,k_2} \langle k_1|\, \frac{P^2}{2} \,|k_2\rangle\, a_{k_1}^\dagger a_{k_2} \\ &= \sum_{k_1,k_2} \delta_{k_1 k_2} \frac{k_2^2}{2} a_{k_1}^\dagger a_{k_2} \\ &= \sum_k \frac{k^2}{2} a_k^\dagger a_k\end{aligned}$$

In fact, $a_k^\dagger a_k$ will count the number of particles that have momentum k in the system, while $k^2/2$ will yield the energy of each of these particles.

If our Hamiltonian is to contain a potential energy contribution from particle interaction, we need to also mention two-body operators. We start from a similar reasoning then the one for one-body operators, but now, following [26], the expression Eq. 2.13 has the form

$$O = \frac{1}{2} \sum_{i \neq j} O_{ij} \bigotimes_{k \neq i,j} \mathbb{I}_k, \tag{2.17}$$

It is clear from the subscripts that O_{ij} acts on the Hilbert space of the two particles i and j at the same time. As we want to express the matrix elements of the two-body operator in the basis of our single particle states, we want to know the elements

$$\langle\alpha\beta|\, O \,|\gamma\delta\rangle = \frac{1}{2} \left[\langle\beta| \otimes \langle\alpha| \pm \langle\alpha| \otimes \langle\beta|\right] O \left[|\gamma\rangle \otimes |\delta\rangle \pm |\delta\rangle \otimes |\gamma\rangle\right].$$

However, we need to pay attention to some points. When writing a many-body state as $|\alpha\beta\rangle$, symmetrisation and normalisation are already assumed, as in Eq. 2.7. Furthermore, the ordering in which the states are written is important, especially given the antisymmetry of the creation and annihilation operators for fermionic systems.

Now, to avoid carrying this tensor product, we use the following notation for what we call ordered kets $|\alpha\beta) = |\alpha\rangle \otimes |\beta\rangle$, without symmetrisation or normalisation constraints. In this case, following the reasoning for the one-body operator, one can show that a general two-body operator is written in second quantisation as

$$\hat{O} = \frac{1}{2} \sum_{\alpha\beta\gamma\delta} (\alpha\beta| \, O \, |\gamma\delta) \, a^\dagger_\alpha a^\dagger_\beta a_\delta a_\gamma.$$

2.4 Hartree-Fock

The Hartree-Fock method (HF) is a foundational approach in quantum many-body theory. Despite not giving the most precise results, it serves as an excellent starting point for understanding more complex methods and provides a benchmark for our results. It is computationally efficient and provides an explicable accuracy with stability conditions. For these reasons, it is commonly used as a first step in the direction of solving a many-body problem.

We discuss the following section with the fermionic antissimetry in mind. In this case, the method consists in approximating the N-body wavefunction by one Slater determinant of unknown single-particle wavefunctions $\psi_i(\mathbf{x}_j)$

$$\Psi \approx \Psi_{HF} = \frac{1}{\sqrt{N!}} \det \{\psi_i(\mathbf{x}_j)\}.$$

These unknowns are determined using an iterative approach guided by the variational principle. First, one minimises the energy functional constrained to the set subspace of wave functions and constrained to a normalisation of the probabilities of the single-particle functions. That will give rise to a set of nonlinear eigenvalue equations called the Hartree-Fock equations,

$$\begin{aligned} \hat{f} \, |\psi_i\rangle &= \epsilon \, |\psi_i\rangle \\ \hat{f} &= \hat{t} + \hat{u}_{ext} + \hat{u}^{HF}. \end{aligned} \tag{2.18}$$

The operator $\hat{f}$, called the Fock operator, is a one-body operator with $\hat{u}^{HF}$ a single-particle potential that will be determined by the iterative method. Furthermore, $\hat{t}$ represents the kinetic energy operator and $\hat{u}_{ext}$ the external potential energy operator. The eigenvalue problem above gives nonlinear equations exactly because this operator depends on the eigenstates $|\psi_i\rangle$ of the other particles, and the equations are solved iteratively.

From the entire space of antisymmetric functions, there is one that satisfies $E_0 = \langle\Psi| \, \hat{H} \, |\Psi\rangle$. Then, instead of searching for the states that satisfy this energy in the whole Hilbert space, with HF we limit the search to the subspace of antisymmetric functions that can be expressed by Slater determinants. The best possible solution achievable on this basis is

$$E_{HF} = \sum_{i=1}^{N} \langle\psi_i|\hat{H}|\psi_i\rangle, \tag{2.19}$$

with $|\psi_i\rangle$ the eigenstates of the HF equations. We will not deduce how to arrive at the Hartree-Fock equations, mainly because our goal is not to implement the method. We instead use it as a theoretical guideline and benchmark.

Conceptually, the potential $\hat{u}^{HF}$ is a mean-field potential, and many-body interactions are simplified to a one-particle interaction with the mean-field. By using such a mean-field picture, we neglect many-body correlations, which, depending on the system at hand, can be very significant.

The HF iterative process begins by approximating the electrons' likely positions without considering two-body interactions. This leads to an estimate of the mean-field, which is used to update the single-particle basis, changing the probability densities. This in effect changes the mean-field itself, in a process that is repeated until convergence. Since this process ignores higher-order correlations, the resulting field may be weaker than if those correlations were considered.

For bosons, the ground-state wavefunction is represented as the product of individual particle wavefunctions, assuming that all particles occupy the lowest energy state simultaneously. This interacting system is known as a Bose-Einstein condensate. Analogous methods to those used for fermionic HF can be applied to bosonic systems, leading to the derivation of the Gross-Pitaevskii equations [79] rather than the Hartree-Fock equations.

2.5 Full Configuration Interaction

Other methods exist that consider linear combinations of Slater determinants, instead of assuming only one determinant as the basis of the problem. These are the so-called post Hartree-Fock methods, and the use of more complicated basis allows for higher-order correlations terms at the expense of computational cost. While the Hartree-Fock method gives too coarse an approximation, full configuration interaction (FCI) is technically an exact method. Let $|\Phi_\alpha\rangle$ be a Slater determinant, FCI consists in expanding the wave function in the basis of all possible Slater determinants in the case of fermionic system, as follows:

$$|\Psi_{\text{FCI}}\rangle = \sum_\alpha c_\alpha |\Phi_\alpha\rangle . \tag{2.20}$$

This expansion can then be put in the Schrödinger equation,

$$\hat{H}\left(\sum_\alpha c_\alpha |\Phi_\alpha\rangle\right) = E\left(\sum_\alpha c_\alpha |\Phi_\alpha\rangle\right),$$

and using the orthonormality of the basis $\langle\Phi_\gamma|\Phi_\alpha\rangle = \delta_{\gamma\alpha}$, it follows

$$\sum_\alpha \langle\Phi_\gamma| H |\Phi_\alpha\rangle c_\alpha = E c_\gamma.$$

This can once more be seen as an eigenvalue problem, where coefficients are elements of a vector, and $\langle\Phi_\gamma| H |\Phi_\alpha\rangle$ the matrix elements of the Hamiltonian in this FCI basis. Solving this boils down to the diagonalisation of such a matrix and calculations of the integrals that yield the matrix elements. Of course, the number of possible states being infinite forces us to reduce our basis to a subset.

The study of truncated CI methods is concerned with different truncation choices and how that interferes with the level of approximation. For example, given a reference determinant $|\Phi_0\rangle$ (usually the HF determinant), we could restrict the determinants of Eq. 2.20 to include only one electron excitation, in which case we have configuration interaction with singles (CIS). For the case where two electron excitation is allowed, we have CID, and if both are allowed, you guessed it: CISD.

Even using a truncated basis can be computationally very expensive. This is because the number of determinants required increases factorially with the number of electrons and orbitals [39]. Therefore, these methods are used only in modest systems with up to no more than 100 electrons.

2.6 Fermionic Systems

After presenting some theoretical background for many-body problems, we address the choice of the systems with which we are working. Throughout this thesis, we deal with fermions in harmonic oscillator traps up to two dimensions. These systems, in which particles are trapped, are

interesting because there is a substantial number of studies to compare results to, and because they have real-world applications. For example, trapped ions serve as qubits for recent quantum computing applications [98], while ultra-cold Fermi gases allow researchers to investigate phenomena such as superfluidity [87].

In its most general form, we can write the Hamiltonian of the systems we will consider as

$$\hat{H} = \sum_{i}^{N} \left(\frac{-\hbar^2}{2m} \nabla_i^2 + V_{ext}(\mathbf{r}_i) \right) + \sum_{i<j}^{N} V_{int}(\mathbf{r}_i, \mathbf{r}_j), \tag{2.21}$$

where m is the particle mass, and $\hbar$ is the reduced Planck's constant. Here, V_{ext} is an external potential dependent on the d-dimensional coordinates of the particles, $\boldsymbol{r}_i$. Similarly, V_{int} defines a two-body interaction between particles, depending on their distance. Lastly, ∇_i^2 is the d-dimensional Laplacian for particle i.

Since all of our systems are confined harmonic oscillators, it is practical to use energy units of $\hbar\omega$ or $\hbar$, and length units of trap units $a_{ho} = \sqrt{\hbar/m\omega}$. We will primarily use energy units of $\hbar$, except in the fully polarised fermionic scenario, where we adopt energy units of $\hbar\omega$ to ease comparison with a specific study on neural quantum states. As a consequence, we show in Appendix A that the Hamiltonian expression can be simplified, and together with the expression for the external interaction, follows:

$$\hat{H} = \sum_{i}^{N} \left(-\frac{1}{2} \nabla_i^2 + \frac{1}{2} \omega^2 r_i^2 \right) + \sum_{i<j}^{N} V_{int}(\mathbf{r}_i, \mathbf{r}_j), \tag{2.22}$$

where $r_i = \|\boldsymbol{r}_i\|$.

We have seen in Sec. 2.2.1 that the general wave function for a fermionic system can be given in terms of the Slater determinant

$$\Psi_-(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \det\{\psi_i(\mathbf{x}_j)\},$$

where we were using the composite notation $\mathbf{x}_i \coloneqq (\boldsymbol{r}_i, \sigma_i)$. Now, given that the expression of the Hamiltonian Eq. 2.22 is spin-separable (in this case, spin-independent), a single-particle wave function can be written as the tensor product of a spatial function and a spin function $\psi_i(\mathbf{x}_j) = \phi_i(\boldsymbol{r}_j) \otimes \chi_i(\sigma_j)$, giving

$$\Psi_-(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \det\{\phi_i(\boldsymbol{r}_j) \otimes \chi_i(\sigma_j)\}. \tag{2.23}$$

This is a very general expression in terms of spin degrees of freedom. The two fermionic systems which we explore will each constrain the spin degrees of freedom in a different way, leading to interesting physical study cases.

One-Dimensional Trapped Spinless Fermions

We start with a one-dimensional trapped fermionic toy model system inspired by [44] and that has been used to test the quality of different optimisers [20]. In such a system, the fermions are said to be fully polarised or spinless. Similarly to quantum dots systems, the Pauli exclusion principle requires an antisymmetric ansatz with respect to particle position, and S-wave interactions are therefore forbidden.

To be able to compare our results with previous works, we further use a Gaussian finite-range two-body interaction given by

$$V_{int}(x_i, x_j) = \frac{V}{\sigma\sqrt{2\pi}} \sum_{i<j}^{N} \exp\left[-\frac{(x_i - x_j)^2}{2\sigma^2} \right].$$

Here, V represents the strength of the interaction and σ its range. In the limit where the range goes to zero, the interaction becomes a contact interaction. The full N-body Hamiltonian in the correct units for the HO can be written as

$$\hat{H} = \sum_{i}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}x_i^2 \right) + \frac{V_0}{\sigma_0\sqrt{2\pi}} \sum_{i<j}^{N} \exp\left[-\frac{(x_i - x_j)^2}{2\sigma_0^2} \right], \tag{2.24}$$

the derivation of which is given in Appendix A.

Before approaching the full interactive problem, it is useful to examine the non-interactive eigenstates of the system, as they can be obtained analytically. Given that the Hamiltonian in question is spin-independent, and that the spin degrees of freedom are fixed by full polarisation, we can proceed by ignoring spin. In this scenario, each fermion occupies a single particle state which is given by:

$$\phi_n(x) = C_n \exp\left(-\frac{x^2}{2} \right) H_n(x).$$

Here, C_n is a normalisation constant,

$$C_n = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}}$$

and $H_n(x)$ is the n-th Hermite polynomial

$$H_n(x) = (-1)^n \exp(x^2) \frac{d^n}{dx^n} \left(\exp(x^2) \right). \tag{2.25}$$

The eigenenergies for a single particle in a one-dimensional quantum harmonic oscillator, in units of $\hbar\omega$, are known to be $\epsilon_n = (n + 1/2)$ for any non-negative integer n. Consequently, the ground-state energy for a system of N fully polarised fermions will be simply the sum of the individual fermionic energies with the correct quantum number. Since anti-parallel spin configurations are not allowed in the fully polarised scenario, each energy level is only occupied by one particle. Consequently, the ground-state energy is given by

$$E_{gs} = \sum_{n=0}^{N-1} \epsilon_n = N^2/2.$$

As the wavefunction antisymmetry is simply positional, and the Hamiltonian spin independent, we can write the many-body wavefunction as the Slater determinant of Hermite polynomials and a Gaussian envelope, reducing Eq. 2.23 to

$$\Psi_-(x_1, x_2, \ldots, x_N) = \frac{1}{\sqrt{N!}} \left[\prod_i^N C_i \exp\left(-\frac{x_i^2}{2} \right) \right] \det\{H_i(x_j)\},$$

with C_i the normalisation constant.

Two-Dimensional Quantum Dots

We now describe an analogous physical system of trapped particles, but instead of fully polarised fermions, we deal with fermions in a closed-shell configuration, sometimes called quantum dots. We further consider a pure two-dimensional isotropic harmonic oscillator potential trap.

Now, instead of a Gaussian finite-range interaction potential, we deal with a repulsive Coulomb interaction. Then, the unperturbed part of the Hamiltonian is analogous (apart for an added dimension), while the interaction term follows

$$V_{int}(\boldsymbol{r}_i, \boldsymbol{r}_j) = \sum_{i<j} \frac{1}{r_{ij}},$$

where we use the convention $r_{ij} = \|\boldsymbol{r}_i - \boldsymbol{r}_j\|$.

To solve the non-interactive problem, however, we have to consider the quantum numbers for the added dimension. For a complete solution of the time-independent Schrödinger equation, we refer to [100]. Given the symmetry of the problem, it can be solved independently for each coordinate, and one generally proceeds by separation of variables. Then the spatial wave function for one fermion in an oscillator potential in two dimensions is

$$\phi_{n_x,n_y}(x,y) = C_{n_x,n_y} H_{n_x}(\sqrt{\omega}x) H_{n_y}(\sqrt{\omega}y) \exp\left(-\omega(x^2+y^2)/2\right). \tag{2.26}$$

Here, C_{n_x,n_y} is the normalisation constant and $H_{n_x}(\sqrt{\omega}x)$ are the Hermite polynomials for each coordinate, following Eq. 2.25, with ω the trap frequency.

Given the closed-shell configuration, we investigate a system only with an even number of particles, where half have spin-up coordinates and the other half have spin-down. This reduces the possible configurations of Eq. 2.23 and, as long as we are consistent, we can set the first half of particles to have spin up and the second half to have spin down,

$$\psi_i(\mathbf{x}_j) = \begin{cases} \phi_i(\boldsymbol{r}_j) \otimes \chi_\uparrow(\uparrow) & \text{if } i \leq N/2, \\ \phi_i(\boldsymbol{r}_j) \otimes \chi_\downarrow(\downarrow) & \text{else,} \end{cases} \tag{2.27}$$

as any $\chi_{\uparrow(\downarrow)}(\downarrow(\uparrow))$ evaluates to zero due to the exclusion principle. This in turn makes the Slater matrix of factors $\{\phi_i(\boldsymbol{r}_j) \otimes \chi_i(\sigma_j)\}$ be block-diagonal, allowing us to write

$$\begin{aligned} \Psi_-(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}} \det\{\phi_i(\boldsymbol{r}_j) \otimes \chi_\uparrow(\uparrow)\} \times \\ \det\{\phi_k(\boldsymbol{r}_l) \otimes \chi_\downarrow(\downarrow)\} \end{aligned}$$

where $1 \leq k,l \leq N/2$ and $N/2 < k,l \leq N$. It can further be shown that the spin functions can be completely factorised out and omitted, leaving us with a wavefunction only in terms of positions:

$$\Psi_-(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N) = \frac{1}{\sqrt{N!}} \det\{\phi_i(\boldsymbol{r}_j)\}^\uparrow \det\{\phi_k(\boldsymbol{r}_l)\}^\downarrow$$

where we added arrows to help remember that $1 \leq k,l \leq N/2$ and $N/2 < k,l \leq N$.

The energy levels are independent for each coordinate and can be written as $\epsilon_{n_x,n_y} = \epsilon_x + \epsilon_y = (n_x + n_y + 1)\omega$. It becomes clear then that the energy levels are degenerate and that the important number, called the principal number that determines the energy value is the sum $n = n_x + n_y$. For N fermions, due to the exclusion principle, even without interaction, we have to carefully analyse the configuration to not allow two fermions to occupy the same quantum state. In this case, if we consider the spin degrees of freedom, it means that two fermions only assume the same spatial quantum numbers if their spin is antiparrallel.

This gives rise to what we call magic numbers in a closed-shell configuration. This number represents how many particles are needed to complete a closed-shell configuration, meaning that the energy levels are filled with all particles having anti-parallel spins configurations in their respective energy level. This number is given by the binomial coefficient

$$N = 2\binom{n+d}{d},$$

where d is the dimension and the pre-factor 2 comes from the two possible spin configurations in the electronic case. For example, for a two-particle system, the ground-state closed shell with principal number 0 can only be obtained by having both single-particle functions assuming $n_x = n_y = 0$ but with anti-parallel spin. From Fig. 2.2 it becomes clear the reason behind the triangular number.

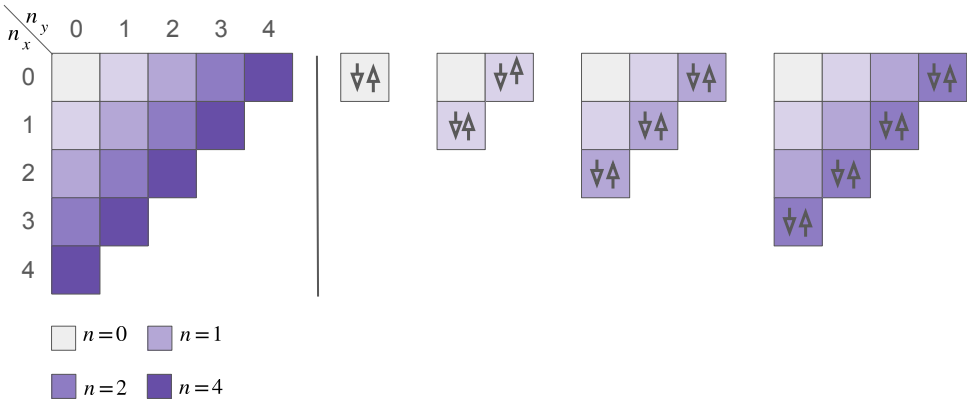


Figure 2.2: Illustration of the possible n_x and n_y configurations in order to yield a specific principal number n which determines the final energy level. Note that to obtain the number of electrons in the closed shell system one needs to fill the shell configurations **up to** the desired principal number.

Kato's Cusp Conditions

The Hamiltonian of a quantum mechanical system, as described by Eq. 2.21, can include an interaction potential that may exhibit singularities. The Coulomb potential, for example, follows an inverse proportionality to the distance between particles, expressed as $1/r_{ij}$, which diverges when the particles are arbitrarily close and $r_{ij} \to 0$.

To ensure that physical quantities such as the energy remain well defined, the singularity must be somehow offset by the kinetic term in the Hamiltonian. The fact that the kinetic energy of the system is merely proportional to the Laplacian of the wavefunction implies that the wavefunction itself must adhere to some mathematical conditions to counterbalance the singularity in the potential.

Kato's theorem, first presented in [43], provides a formal framework for the description of the wave function at those interaction singularity points. More specifically, it states that Ψ must follow

$$\left.\frac{\partial \Psi}{\partial r_{ij}}\right|_{r_{ij}=0} = \frac{m_i m_j}{m_i + m_j} Z_i Z_j \Psi(r_{ij} = 0),$$

where Z represents charge, and m the masses of particles. As a consequence, a wave function following such a condition must have a cusp profile, or sharpness in the particle density at the singularities. The gradient of the wave function changes abruptly at $r_{ij} = 0$ in a way that controls the divergent potential.

As will be shown in Sec. 5.1, this condition will guide our choice of a wave function in the computational implementation. Then we will require the use of additional terms for the ansatz, such as the Jastrow factor and the Padé-Jastrow factor.

2.7 Classical and Quantum Correlations

When discussing Monte Carlo simulations Sec. 3.3, we will talk about correlations, although in a different sense.

Statistically, there are multiple ways to measure how random variables depend on or influence each other. This dependency is what we understand by correlation. Consider two random variables A and B. The fact that those depend on each other can be expressed by the inequality

$$P(A, B) \neq P(A)P(B).$$

There exist correlation coefficients to measure how strongly this inequality holds, such as the Pearson correlation [84]. To make things explicit, there are at least three types of correlation that we would like to distinguish. First, if we allow the particles to interact, for instance, due to Coulomb interaction, observables such as position of one particle influence the probability density function of the others, inducing correlation.

Second, even if identical quantum particles do not interact classically, we know that (anti-)symmetry constraints induce correlation between states. In general, we know that $\psi_{\alpha,\beta}(x_1, x_2) \neq \psi_\alpha(x_1)\psi_\beta(x_2)$, and even if in this example $\psi(x)$ is not a probability function, the argument still holds.

Lastly, entanglement is perhaps the most explicit type of quantum correlation. Entangled particles often have their measurement completely determined by the measurement of the entangled particle. Of course, not all correlated states are entangled. A system could be a classical mixture of factorisable pure states, in which case the joint probability densities for measurements on these two subsystems are accurately described by the classical probability theory of correlated random variables. On the other hand, entangled states can be mixtures of broader, non-separable states.

Chapter 3

Computational Background

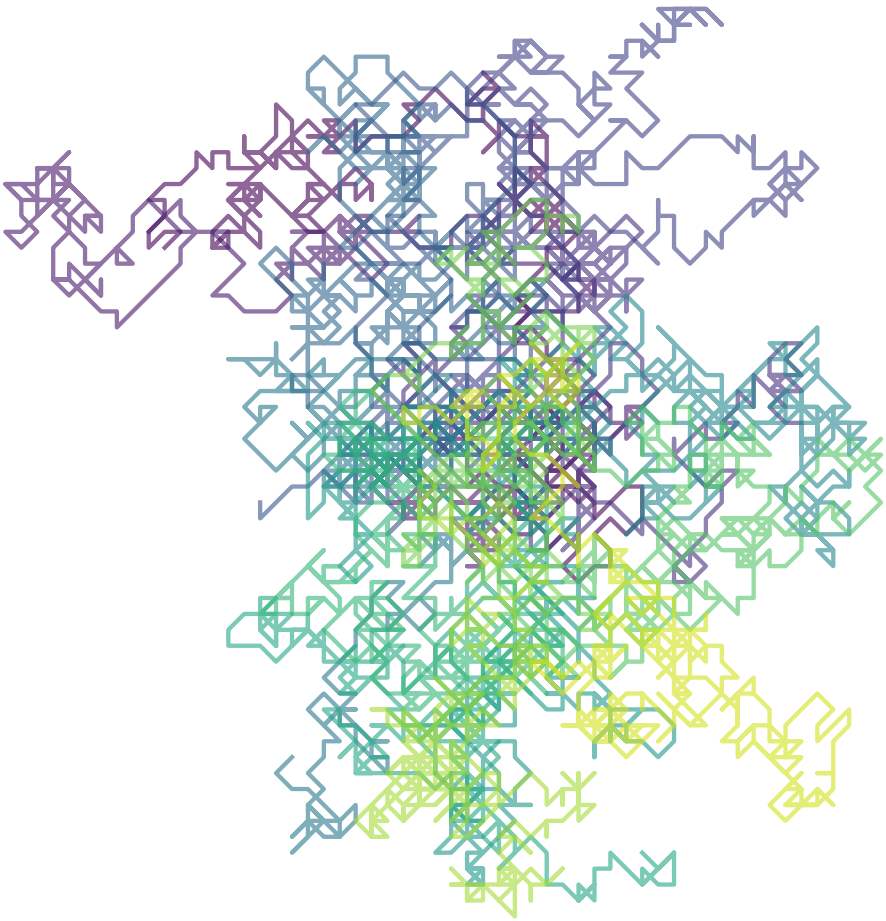


Figure 3.1: Trajectory of 15 random walkers on a lattice, inspired by [69].

3.1 Sampling and Markov Chains

Suppose that we wish to study a phenomenon in nature that is inherently stochastic and for which we lack prior information about its probability distribution. Then, collecting information from one single experiment is not significant. If we want to infer statistics from a set of events, we rely on sampling.

By sampling, we mean selecting a subset of a population from which to collect information.

Doing so with specific statistics techniques allows us to infer information from the whole population without measuring every instance and with a good notion of how wrong the inferred values are. To eliminate unwanted bias in this collection process, it is common to conduct sample selection with some level of randomness.

In this discussion, random or stochastic variables will be denoted as X and represent a function that maps a set of outcomes $\Omega = \{\text{outcome A}, \text{outcome B}\}$ to a set of numbers, for example $\{23, \pi\}$. To quote A. Eagle (2014) [21], one sees that a random variable is "... neither random nor a variable".

In the context of random variables, we will represent P as the probability measure, defined in the sample space, while p is its probability density function. Furthermore, we represent a conditional probability as $P(X = x|Y = y)$, that is, a probability of random variable assuming definite value x given the knowledge that Y has been evaluated y.

Markov Chains

A family of random variables indexed with a sequential notion $\{X_i\}$ is called a stochastic process. While this indexing could be discrete or continuous, in computer simulations we will work with discrete-time Markov chains. Furthermore, we disclaim that this section will compromise mathematical rigour in favour of a short and intuitive presentation of Markov chains. Therefore, some concepts such as irreducibility, ergodicity, and detailed balance will sometimes be used interchangeably.

A Markov chain is a specific type of stochastic process in which the notion of the future outcome is dependent only on the outcome of the current step, and not on the previous ones. More specifically, if the outcome of a random variable of the stochastic process is $X_{i^*} = x$, the outcome of X_{i^*+1} depends only on the value x, and not on any other $\{X_j\}_{j<i^*}$.

The way in which the Markov chain evolves is modelled by the likelihood of a transition between two states, $t(x, y)$, where x and y belong to S, the space of all the outcomes of the stochastic variables. To correctly specify a Markov chain, we need an initial probability distribution π_0 at initial state, and the transition probabilities of the following states, meaning

$$P(X_0 = x) = \pi_0(x),$$
$$P(X_{n+1} = x_{n+1}|X_n = x_n) = t(x_n, x_{n+1}).$$

Note that we did not need to specify the previous states of the chain in the conditional probability, as per the definition of Markov chains. Sampling from this chain can be done by first sampling X_0 according to π_0, and subsequently sampling X_n following $t(X_{n-1}, \cdot)$.

It is common to associate $t(x, y)$ to elements in a transition matrix, representing the probability transition $x \to y$ of one time step. In this case, it makes sense to see $p^m = P(X_{n+m} = y|X_n = x)$ as matrix t being applied in succession. Furthermore, the probability distribution guiding the outcome of the random variable X_n can be written

$$P(X_n = x) \equiv \pi_n(x) = \pi_0 t^n.$$

There are several conditions that must be satisfied for our process to be a relevant Markov chain. For example, fixed a state x, there must be transition altogether (even if to the same state), so

$$\sum_{y \in S} t(x, y) = 1.$$

Another subtle point is that, since we study these chains for long-time evolutions, we want stochastic processes with "nice" asymptotic behaviour. For that to be possible, some requirements must be satisfied. First, we do not care about chains that go to subsets of the state space

and never return. That would cause the isolation of the chain in a way that leads to a divergence in statistical quantities. More specifically, we say we want to study irreducible chains: chains for which any state can be reached in some finite number of steps.

Looking at the other extreme case, we also do not want to study periodic chains, for a similar problem of lack of convergence to a stationary distribution. These conditions are usually condensed into the requirement that the Markov chain satisfies a so-called detailed balance condition, formulated as follows: $\forall x, y \in S \times S$,

$$\pi(x)t(x,y) = \pi(y)t(y,x). \tag{3.1}$$

That means that when the distribution has reached a steady distribution, the proportion of transitions from state x to y is the same as from y to x. Given detailed balance, we can invoke two convergence theorems for finite-state Markov chains. The first one, with respect to the convergence of the distributions, and the other one with respect to the expected values of functions of the random variables. Those are the expected values we seek to sample.

First, if we have $t(x, y)$ the transition matrix of irreducible, aperiodic finite state Markov chains, $\forall x, y \in S \times S$, any initial distribution π_0 will lead to π_n which converges to a stationary π, and we write

$$\lim_{n\to\infty} t^n(x,y) = \pi(y). \tag{3.2}$$

Furthermore, for that stationary π and considering $f(x)$ any function on the state space, for any initial π_0 it follows

$$P\left(\lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} f(X_k) = \sum_x f(x)\pi(x)\right) = 1. \tag{3.3}$$

This means that the expected value of f, which depends on the random variable, when sampled towards infinity, will tend to ever better approximate the true expected value of f. This theorem is crucial for Markov chain Monte Carlo, and it leads us into the topic of Monte Carlo methods.

3.2 Markov Chain Monte Carlo

Monte Carlo Methods

Monte Carlo (MC) methods are a very broad class of computational methods that use sampling to generate numerical results. These methods are used in scenarios where traditional computational approaches are impractical, making it more feasible to tackle the problem by introducing randomness and depending on the law of large numbers for approximate solutions. First seen as a "last resort", MC is now considered a robust approach, present extensively in scientific computing [50].

Monte Carlo methods can be used to solve both probabilistic and deterministic problems. A probabilistic example would be risk assessment simulations, while a deterministic example would be calculations of intractable integrals. We will use Monte Carlo methods in two very intertwined contexts: for the evaluation of high-dimensional integrals and for sampling quantities that follow complicated or unknown probability distributions via Markov chains.

This sampling process for Monte Carlo methods is stochastic and the information obtained about the data is an inferred probability distribution with an associated random error. Despite this randomness in the estimation, the uncertainty range can be arbitrarily reduced, given the sufficient computational time, as eluded to in Eq. 3.2 and Eq. 3.3.

To demonstrate this, let us explore the approximation of an integral using Monte Carlo sampling. Suppose that we want to compute the integral of a function $f(\boldsymbol{x})$, over $\Omega \subset \mathbb{R}^d$, which

has volume V. The naive MC approach is to sample uniformly n points, x_i of Ω to compute the fraction of these samples that is evaluated below $f(\Omega)$. Of course, the probability of this happening must be proportional to the magnitude of $f(x_i)$ and, consequently, to the integral value. Mathematically, this approximation boils down to

$$\int_{\Omega} f(\boldsymbol{x})\, d\boldsymbol{x} \approx V \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_i). \tag{3.4}$$

A Comment on Random Number Generation

The generation of truly random numbers in computer programmes is not feasible. What we do instead is generate apparently uncorrelated numbers with deterministic algorithms called pseudo-random number generators (PRNGs). Given a known initial state determined by a number X_0 which we call a seed, the algorithm can replicate a whole sequence X_n of apparently random numbers. There are a myriad of PRNGs, with significant differences in terms of randomness quality, speed, cycle, and other characteristics.

For a true uniform random variable, the generated numbers should be completely uncorrelated, and, in PRNGs, we try to avoid correlations between generated numbers as much as possible. However, since their number generation is deterministic, some level of correlation is inevitable, even if subtle.

In computer simulations, where random numbers are generated millions of times, a fast algorithm is absolutely essential. For illustration, we mention a well-known and reasonably fast algorithm: the Linear Congruential Generator (LCG). An LCG is defined by the recurrence relation

$$X_{n+1} = (aX_n + c) \mod m,$$

where the choice of m, a, and c determines the specific generator and how long of a sequence of pseudo-random numbers will be before starting to repeat. The length of such a sequence is called the cycle, and modern PRNGs have cycles of around 2^{128} numbers.

Although PRNGs typically generate numbers uniformly between 0 and 1, there are techniques to enable these generators to produce numbers following specific, well-known distributions, such as the Gaussian distribution. Among these methods are the reverse transform method, rejection sampling, and transformation techniques such as the Box-Muller method. Ideally, when one wants to generate random samples following a specific low-dimensional distribution, these methods are excellent choices. Unfortunately, in higher dimensions, these become extremely inefficient, motivating the search for other algorithms. To illustrate this, we briefly introduce the rejection sampling method. Although we do not explicitly use it, it is a perfect bridge between Monte Carlo integration and Markov chain Monte Carlo.

Rejection Sampling and Curse of Dimensionality

The idea of rejection sampling is to use a distribution from which we know how to sample, $g(x)$, to guide the sampling process and mimic the sampling of the distribution we desire, $p(x)$. This will work as long as g is an envelope for p. More specifically, we need $p(x) < Kg(x)$ for some scaling constant $K > 1$ and at any point in the function's domain. The iterative sampling process then follows three steps, which can be better understood with Fig. 3.2.

- Sample a point x from an envelope proposal distribution $g(x)$;
- Sample y form the uniform $U(0, Kg(x))$;
- Accept and keep track only of samples for which $y < p(x)$.

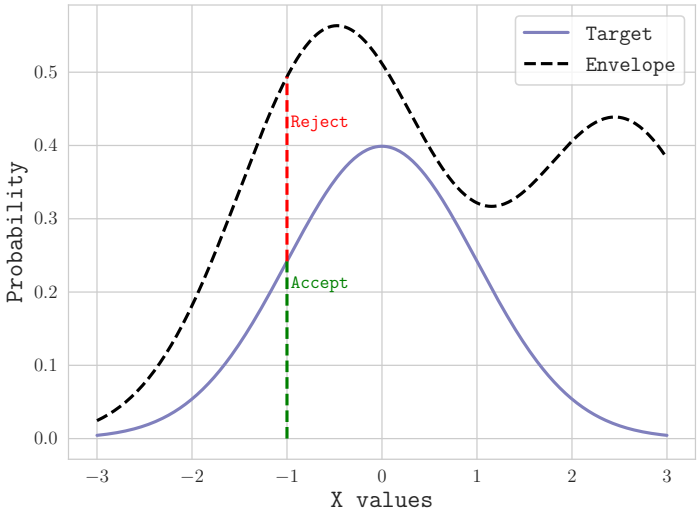


Figure 3.2: Illustration of rejection sampling with an envelope and target distributions. Probabilities are not normalized.

With that, the normalised histogram that forms from the accepted samples tends to approximate the target distribution, indicating that the accepted samples indeed follow p. The problem with such a procedure, which is hard to see in low-dimensional examples, is that the rate of convergence for the approximation becomes slow in higher dimensions. The sampling is inefficient. In fact, the volume of rejected samples will increase at a much faster rate than that of the accepted ones, requiring a number of samples that is impractical. This is a consequence of the curse of dimensionality.

Markov Chain Monte Carlo

To finally connect the concept of Monte Carlo integration with Markov chains, we can ask ourselves what would happen if, in Eq. 3.4, we sampled domain points using a general probability distribution p instead of a uniform one. Recall that the expected value of a sample of x following a probability distribution p is denoted

$$\mathbb{E}_{\mathrm{x}\sim p}[f(X)] = \int f(x)p(x)dx,$$

where we omit the domain for simplicity. If we compute the expected value of a function evaluated on domain points sampled following p we solve a related but not exactly the same problem as the pure integral calculation of f. This expectation value can be approximated by the average of the samples,

$$\mathbb{E}_{\mathrm{x}\sim p}[f(\mathrm{x})] \approx \hat{\mathbb{E}}_{\mathrm{x}\sim p}[f(\mathrm{x})] = \frac{1}{n}\sum_{i}^{n} f(x_i).$$

As the number of samples n increases, under the law of large numbers for Markov chains, this approximation will improve arbitrarily. Still, the question of how to generate samples from p with Markov chains remains. As discussed in Sec. 3.1, we require a Markov chain for which the stationary distribution is the one we want to sample from. In fact, there are different possible ways to do that, with maybe the two most common being the Metropolis and Gibbs algorithms.

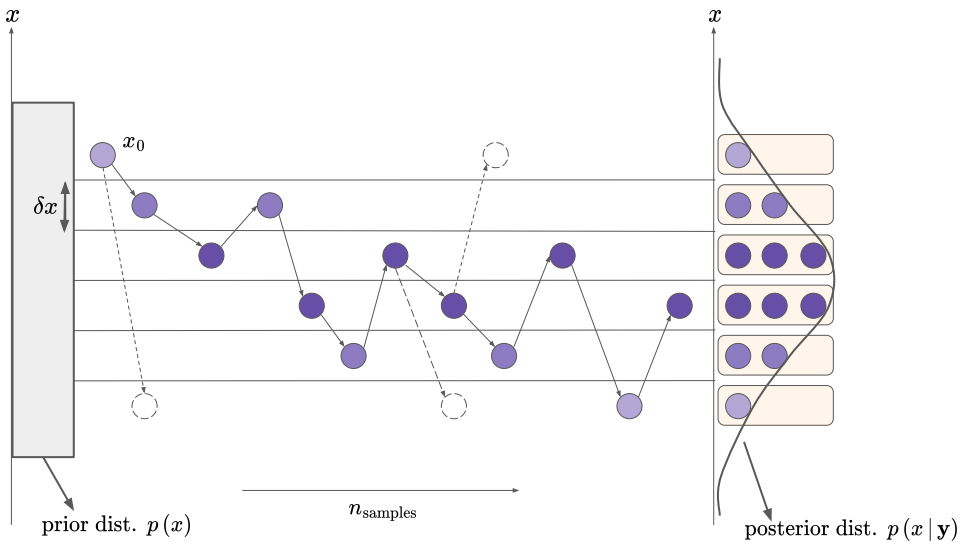


Figure 3.3: Illustration of using a prior distribution sampling to obtain samples from a posterior distribution by accepting and rejecting moves from an initial position x_0. Adapted from Lee et al., 2015 [55]

3.2.1 Metropolis Algorithm

Initially described in [63], the Metropolis algorithm serves as one method to generate a Markov chain to sample from, aiming to achieve a steady distribution that matches the target distribution. So far, we have not addressed how to get the correct transition probabilities $t(x, y)$ required from Eq. 3.2. The idea of Metropolis algorithm is to model a transition in state space via two independent probabilities. First, the transition state y must be available, modelled by a proposal distribution $g(x, y)$. Moreover, there is a probability that y is accepted $a(x, y)$ from the current state x. The fact that these are assumed independent means we can rewrite the detailed balance condition of Eq. 3.1 as

$$\pi(x)a(x, y)g(x, y) = \pi(y)a(y, x)g(y, x),$$

or, rearanging,

$$\frac{a(x, y)}{a(y, x)} = \frac{\pi(y)}{\pi(x)} \frac{g(y, x)}{g(x, y)}. \tag{3.5}$$

Note that g is something that we control and could be, for example, a normal distribution centred around the current state. Furthermore, it plays a similar role to the envelope distribution in rejection sampling. Satisfying Eq. 3.5 above is equivalent to having an acceptance rule of

$$a(x, y) = \min\left(1, \frac{\pi(y)}{\pi(x)} \frac{g(y, x)}{g(x, y)}\right). \tag{3.6}$$

In other words, if we use such an acceptance rule with a proposal distribution that we control, we are still satisfying detailed balance. In this case, the sampled values will converge to the sampled values under the desired stationary probabilities. If we use a symmetric proposal distribution g such as a normal distribution, Eq. 3.6 further simplifies, given that the fraction of proposed probabilities will be one. Under this simplification, the Metropolis–Hastings algorithm is simply called Metropolis, but more often than not the names are used interchangeably.

A final comment on the Metropolis-Hastings algorithm is that, due to the fraction between $\pi(y)$ and $\pi(x)$, it allows us to sample quantities from $\pi(y)$ with any proportional probability distribution $f(x) \propto \pi(x)$.

Sec. 3.3 includes an algorithmic recipe with algorithm box 1, although fitted to our variational Monte Carlo framework. In addition, Fig. 3.3 illustrates the general idea of proposing steps from an initial state x_0 and accepting or rejecting them to generate a posterior distribution.

Joining the Dots

This last comment about allowing for a function $f \propto \pi$ can seem irrelevant but allows us to sample from distributions for which we lack knowledge of partition functions. In our case, it allows us to use an unnormalized ansatz in variational Monte Carlo.

3.3 Variational Monte Carlo

As discussed in Sec. 2.6, we are deeply interested in solving the Hamiltonian eigenvalue problem. More specifically, we are focused on the smallest eigenvalue and its eigenstate. We now see a way to approach this problem via Markov chain Monte Carlo, avoiding explicitly solving the Schrödinger equation.

By the variational principle, discussed in Sec. 2.2.5, any trial wave function yields and expectation value for the energy that is bounded from below by the true ground-state energy of the system. With that in mind, variational Monte Carlo (VMC) is a method that iteratively samples energy values from a parameterised trial wave function $|\Psi_T(\boldsymbol{\theta})\rangle$ and updates its parameters $\boldsymbol{\theta}$ to drive the sampled energies to a minimum. VMC is heavily biased by the functional choice for the trial function, and finding a good initial guess can be extremely difficult. It is therefore an approximate method with potentially large error bars.

Despite these challenges, VMC remains easy to implement in comparison to other more precise methods, such as diffusion Monte Carlo, while also avoiding the infamous sign problem [73]. The idea behind VMC is to note that, by using a trial wave function Ψ_T, one can express the expected value for any observable O on a complete basis $\{|\boldsymbol{\alpha}\rangle\}$ as

$$\frac{\langle\Psi_T|\hat{O}|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle} = \sum_{\boldsymbol{\alpha},\boldsymbol{\beta}} \frac{\langle\Psi_T|\boldsymbol{\alpha}\rangle\langle\boldsymbol{\alpha}|\hat{O}|\boldsymbol{\beta}\rangle\langle\boldsymbol{\beta}|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle} \tag{3.7}$$

$$= \sum_{\boldsymbol{\alpha}} \frac{\langle\Psi_T|\boldsymbol{\alpha}\rangle\langle\boldsymbol{\alpha}|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle} \sum_{\boldsymbol{\beta}} \langle\boldsymbol{\alpha}|\hat{O}|\boldsymbol{\beta}\rangle \frac{\langle\boldsymbol{\beta}|\Psi_T\rangle}{\langle\Psi_T|\boldsymbol{\alpha}\rangle} \tag{3.8}$$

$$= \sum_{\boldsymbol{\alpha}} P(\boldsymbol{\alpha}) \sum_{\boldsymbol{\beta}} O_L(\boldsymbol{\beta}). \tag{3.9}$$

The first term in the summation represents the normalised probability of the state $P(\boldsymbol{\alpha})$, while the second term can be interpreted as a local operator estimator O_L. This allows us to approximate the expectation value on the left-hand side of Eq. 3.9 as the average of sampled values of the observable associated with the local operator. To better illustrate this, let us express the trial wave function as $\Psi_{\boldsymbol{\theta}}(\mathbf{R})$ and deal with the Hamiltonian operator. This representation indicates a wave function parameterised by $\boldsymbol{\theta}$ with $\mathbf{R}$ a collective variable of all positions of a multi-particle system. In this case, the VMC energy follows

$$E(\boldsymbol{\theta}) = \frac{\langle\Psi_{\boldsymbol{\theta}}(\mathbf{R})|\hat{H}|\Psi_{\boldsymbol{\theta}}(\mathbf{R})\rangle}{\langle\Psi_{\boldsymbol{\theta}}(\mathbf{R})|\Psi_{\boldsymbol{\theta}}(\mathbf{R})\rangle} = \int E_L(\mathbf{R}) p(\mathbf{R},\boldsymbol{\theta})\, d\mathbf{R} = \langle E_L\rangle_{\mathbf{R}\sim p(\cdot,\boldsymbol{\theta})}, \tag{3.10}$$

where we have introduced the concept of a local energy E_L,

$$E_L = \frac{\hat{H}\Psi_{\boldsymbol{\theta}}(\mathbf{R})}{\Psi_{\boldsymbol{\theta}}(\mathbf{R})}, \tag{3.11}$$

and the probability density function $p_{\boldsymbol{\theta}}(\mathbf{R})$,

$$p_{\boldsymbol{\theta}}(\mathbf{R}) = \frac{|\Psi_{\boldsymbol{\theta}}(\mathbf{R})|^2}{\int |\Psi_{\boldsymbol{\theta}}(\mathbf{R})|^2 \, d\mathbf{R}}. \tag{3.12}$$

This allows us to use Markov chain Monte Carlo to approximate the high-dimensional integral in Eq. 3.10:

$$\int E_L(\mathbf{R}) p_{\boldsymbol{\theta}}(\mathbf{R}) \, d\mathbf{R} \approx \frac{1}{n} \sum_{\mathbf{R} \in \mathbf{R}_n \sim p_{\boldsymbol{\theta}}(\cdot)} E_L(\mathbf{R}), \tag{3.13}$$

where n denotes the number of samples $\mathbf{R}_n$ in a Markov chain framework. To see why this approximation is necessary, let us disregard spin degrees of freedom and try to compute the integral

$$\langle H \rangle = \frac{\int dR_1 dR_2 \dots dR_N \psi^* H \psi}{\int dR_1 dR_2 \dots dR_N \psi^* \psi}.$$

As detailed in [35], evaluating this integral via Gaussian quadrature with 10 particles and 10 mesh points for each degree of freedom would take around 10^{18} seconds, or ten billion years. This calculation considers 10^{30} floating point operations in three dimensions, assuming that the calculations are performed on an ideal computer, which is totally impractical.

For an illustration on how to proceed with the calculation of the local energy of Eq. 3.11, we can break down the Hamiltonian in terms of kinetic and potential energy. The potential term will depend on the system (external potential trap and particle particle interaction), but the local kinetic term can be written,

$$\hat{K} = -\frac{1}{2} \frac{\nabla^2 \Psi_{\boldsymbol{\theta}}}{\Psi_{\boldsymbol{\theta}}}.$$

For stability reasons, it is common, when dealing with VMC, to work with the wave function in the logarithmic domain [5]. This approach helps especially with convergence stability, as the trial function can assume very small or very large values. The sign of the wavefunction must be kept, of course, if one wishes to retrieve the wavefunction expression and not just the probability distribution. In this case, the kinetic term follows

$$\hat{K} = -\frac{1}{2} \sum_{i=1}^{N} \left[\left(\frac{\partial \ln |\Psi_{\boldsymbol{\theta}}(\mathbf{R})|}{\partial R_i} \right)^2 + \frac{\partial^2 \ln |\Psi_{\boldsymbol{\theta}}(\mathbf{R})|}{\partial R_i^2} \right],$$

with N the number of particles and R_i refering to the vector coordinates of particle i. The derivation of this expression is available in Appendix B.

The minimisation of the expectation value for the local energy can be achieved using a gradient descent optimisation approach, which will be further detailed in Sec. 4.2. In its simplest form, the iterative update of parameters can be represented as

$$\theta_{(t+1)} = \boldsymbol{\theta}_{(t)} - \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}), \tag{3.14}$$

with t the iteration index. A caveat here is that we are taking a derivative of the expectation value, which follows, dropping the T subscript for simplicity:

$$\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \frac{\int d\mathbf{R} \, (\nabla_{\boldsymbol{\theta}} \Psi^*) H \Psi + \Psi^* H (\nabla_{\boldsymbol{\theta}} \Psi)}{\int d\mathbf{R} \, \Psi^* \Psi} - \frac{\left(\int d\mathbf{R} \, \Psi^* H \Psi \right) \nabla_{\boldsymbol{\theta}} \left(\int d\mathbf{R} \, \Psi^* \Psi \right)}{\left(\int d\mathbf{R} \, \Psi^* \Psi \right)^2}.$$

If we let $\mathcal{N} = \int d\mathbf{R}\, \Psi^* \Psi$ for simplicity and considering the wavefunction assumes only real values, it follows

$$\nabla_{\boldsymbol{\theta}} \mathcal{N} = 2 \int d\mathbf{R}\, \Psi \nabla_{\boldsymbol{\theta}} \Psi$$

and

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) &= \int \frac{d\mathbf{R}\, (\nabla_{\boldsymbol{\theta}} \Psi) H \Psi + \Psi H (\nabla_{\boldsymbol{\theta}} \Psi)}{\mathcal{N}} - 2 \langle E_L \rangle \left\langle \Psi^{-1} \nabla_{\boldsymbol{\theta}} \Psi \right\rangle \\ &= 2 \left(\left\langle E_L \cdot \Psi^{-1} \nabla_{\boldsymbol{\theta}} \Psi \right\rangle - \langle E_L \rangle \left\langle \Psi^{-1} \nabla_{\boldsymbol{\theta}} \Psi \right\rangle \right), \end{aligned}$$

which is also equivalent to, and more often seen as

$$\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = 2 \mathbb{E}_{\mathbf{R} \sim |\Psi_{\boldsymbol{\theta}}|^2} \left[(E_L - E(\boldsymbol{\theta})) \nabla_{\boldsymbol{\theta}} \log |\Psi_{\boldsymbol{\theta}}(\mathbf{R})| \right]. \tag{3.15}$$

The real-valued wave function assumption in this derivation is reasonable in sight of our computational implementation: We will be dealing only with stationary states and bounded systems.

3.3.1 Metropolis Algorithm and VMC

With this rewriting, we move the difficulty of the problem from a high-dimensional integral to sampling from a distribution over the position space $\mathbb{R}^{Nd}$, where N is the number of particles and d the number of dimensions of the system. Then, a sequential Monte Carlo Markov chain proposal of steps is denoted $\mathbf{R}^{(i)} \to \mathbf{R}^{(i+1)}$ so that it depends only on the previous state at position $\mathbf{R}^{(i)}$. Whether the new point $\mathbf{R}^{(i+1)}$ should be accepted is evaluated according to an acceptance rule, such as Eq. 3.6 for the Metropolis scheme.

Given that $\mathbf{R}^{(i+1)}$ represents the coordinates of all particles, we propose new positions by moving either all particles at the same time or one at a time. Both Metropolis and the (to be discussed) Langevin Metropolis-Hastings require some proposal rule, which we write as $\boldsymbol{\mu} = \boldsymbol{\mu}(\boldsymbol{\sigma}_1, \boldsymbol{\tau}_1) \in \mathbb{R}^d$, where $\boldsymbol{\sigma}_1$ are system-dependent parameters (such as $\mathbf{R}, \Psi_T(\mathbf{R})$) and $\boldsymbol{\tau}_1$ are method-specific parameters. In addition, we define a metric $\nu = \nu(\boldsymbol{\sigma}_2, \boldsymbol{\tau}_2)$ used in the evaluation for the acceptance or rejection of the proposed points.

In the Metropolis algorithm, ν is compared to a random number drawn from a continuous uniform distribution $r \sim U_c(0, 1)$. Should ν be larger than this number, the step is accepted. Otherwise, the configuration $\mathbf{R}$ remains the same. There is also a need to initialise the particle positions, which greatly affects the convergence of the sampling. We will note this by the function $\boldsymbol{\lambda}$, based on drawing positions following a probability distribution. In case we choose to move one particle at a time, the general movement of $\boldsymbol{R}$ in the state space $\mathcal{S}$ can be seen in Algorithm 1. To better see how this extends in the whole context of variational parameter training, we refer to the Algorithm 3.

Algorithm 1 Procedure to determine n configurations in $\mathcal{S}$. U_c and U_d are continuous and discrete uniform probability distributions respectively. Here, we move one particle at a time.

```
Initialize all particle positions R^(1) ← λ
for i = 1, 2, ..., n do
    Draw a random particle index p ~ U_d[1, N]
    Calculate new particle position R'_p ← R_p^(i) + μ
    Calculate metric w ← ν(R', R^(i))
    Draw uniform number r ~ U_c(0, 1)
    if w ≥ r then
        Accept new position: R_p^(i+1) ← R_p^(i)
    end if
end for
```

In the Metropolis algorithm, the normalisation constant of the probability density function in Eq. 3.12 will cancel when calculating the acceptance rule. More specifically, for the trial wave function of the quantum mechanical system of interest, the ratio of the trial step $\mathbf{R} \to \mathbf{R}'$ follows

$$\nu_M(\mathbf{R}, \mathbf{R}') = \frac{p(\mathbf{R}')}{p(\mathbf{R})} = \frac{|\Psi_T(\mathbf{R}')|^2}{|\Psi_T(\mathbf{R})|^2}, \tag{3.16}$$

which leads to the acceptance rule

$$a(\mathbf{R}, \mathbf{R}') = \min\left(1, \nu_M(\mathbf{R}, \mathbf{R}')\right). \tag{3.17}$$

If a proposed step moves towards a higher probability density, $\nu_M > 1$, and we always accept the step. In addition, to adequately sample from the probability distribution, some steps towards lower density regions must also be accepted. To test whether this should be the case, we also compare ν_M with $r \sim U_c(0, 1)$ (as can be seen in Algorithm 1). If each particle is allowed to move a step length of δ in each dimension for each iteration, using a continuous uniform distribution, the proposal rule can be written as

$$\boldsymbol{\mu}(\delta) = \sum_{i=1}^{D} d_i \delta \hat{e}_{x_i}, \qquad d_i \sim U_c(-1, 1), \tag{3.18}$$

where $\hat{e}_{x_i}$ are single particle unit vectors. If δ is too small, only small changes of $\mathbf{R}$ will be allowed between iterations, and the probability density ratio Eq. 3.16 will often be close to one, in which case $\nu_M > r$ will be true for almost every step and almost every proposal will be accepted. Too high of an acceptance rate results in sample means that slowly converge to the population mean. On the other hand, if δ is too large, the proposed step $\mathbf{R}'$ is likely to land in a low-density region of p, with most steps being rejected, little movement in $\mathcal{S}$ and many repeated samples.

Hence, it is crucial to identify a value of δ that ensures a balanced ratio of accepted to rejected steps. This proportion, commonly known as the acceptance rate A_r, is generally associated with reliable results when it is approximately 0.5 [25].

The steps proposed by the Metropolis algorithm are in some sense naive since the proposal step $\mathbf{R}'$ uses no information on the probability distribution, but only on the probability at individual points by evaluating ν_M. This, once more, can lead to inefficient sampling in high-dimensional spaces and motivates the Langevin Metropolis importance sampling, to be introduced in Sec. 3.4.1.

3.4 Diffusion Monte Carlo

Diffusion Monte Carlo (DMC) is a well-established method for employing Monte Carlo sampling and diffusion theory to computationally determine the ground-state energies in quantum systems. Although we do not use the method directly, we discuss it for two reasons. First, it is recognised for providing potentially exact results, which we use to benchmark our calculations. Moreover, DMC has been shown to require an importance sampling approach to the Metropolis algorithm [42]. This approach, which guides the sampling distribution of the proposal steps, is also used by us in our VMC calculations. Our DMC discussion is conceptual, and a more in-depth treatment can be found in [4, 49]

The DMC method is motivated by looking at the time-dependent Schrödinger equation of Eq. 2.2 under imaginary time evolution. This means that we replace $it \to \tau$, and the solution described on an enumerable eigenbasis $|\phi_i\rangle$, ordered according to the ascending order of eigenvalues can be written

$$\begin{aligned} |\Psi(\tau)\rangle &= e^{-\hat{H}\tau} |\Psi(0)\rangle \\ &= \sum_i e^{-E_i\tau} c_i |\phi_i\rangle = c_0 e^{-E_0\tau} \left[\sum_i \frac{c_i}{c_0} e^{-(E_i - E_0)\tau} |\phi_i\rangle \right]. \end{aligned}$$

This can be understood as an operator $\exp(-\hat{H}\tau)$ that acts so that other states decay exponentially to the ground-state. The closer E_i is to E_0, the slower the decay rate. In practice, this time evolution is often unstable, and it is standard to shift the energy scale towards a trial value E_T, for example the energy of the non-interacting ground-state problem. Then, in the limit of long time evolution, the excited parts of the general state get projected to the ground-state, either by decaying, if $E_i - E_T > 0$ or by amplification, in case $E_L - E_T < 0$:

$$|\Psi(\tau \to \infty)\rangle \to c_0 e^{-E_0\tau} |\phi_0\rangle .$$

Under this lens, DMC models the evolution in imaginary time as a generalised diffusion process, where energy terms act as either sources or sinks. Initially conceptualised by E. Fermi in the 1940s, this scenario is described through random walkers experiencing birth-or-death processes.

The operator responsible for the diffusion evolution, which is now shifted by E_T is the Greens's function,

$$\hat{G}(\tau) = e^{-(\hat{H} - E_T)\tau}.$$

After appropriate basis transformations necessary for us to solve the problem computationally, it is possible to write the Green's function in such a representation that yields

$$\Psi(\mathbf{R}, t + \tau) = \int G(\mathbf{R}, \mathbf{R}', \tau) \Psi(\mathbf{R}', t + \tau) d\mathbf{R}'.$$

Furthermore, under what is called the short-time approximation for the Green's function, one is able to approximate G as a product of a diffusion part, G_d, and a branching part G_b, with G_d attributed to the kinectic energy and G_b to the potential energy.

$$G(\mathbf{R}, \mathbf{R}', \tau) \approx G_d(\mathbf{R}, \mathbf{R}', \tau) G_b(\mathbf{R}, \mathbf{R}', \tau) \tag{3.19}$$

We now provide a brief description of how the rest of a naive DMC algorithm would be carried out. For each iteration, the branching part G_b is used to evaluate whether walkers should be created or destroyed from the sampling procedure. Then, the walkers positions are allowed to move following the diffusion due to G_d. The specifics of the branching process and subsequent data collection are not covered in this discussion. Nevertheless, we explore further the diffusion process as it provides inspiration for our VMC implementation.

The diffusion Green's function must satisfy the diffusion equation:

$$\frac{\partial G_d(\mathbf{R}, \mathbf{R}', \tau)}{\partial_\tau} = -D\nabla^2_{\mathbf{R}} G_d(\mathbf{R}, \mathbf{R}', \tau), \tag{3.20}$$

in which case the solution is

$$G_d(\mathbf{R}', \mathbf{R}, \Delta t) = \frac{\exp\left(-\frac{(\mathbf{R}'-\mathbf{R})^2}{4D\Delta t}\right)}{(4\pi D\Delta t)^{3N/2}}, \tag{3.21}$$

with D a diffusion constant and Δt a free parameter time step. In the quantum mechanical case, $D = \hbar^2/2m$ or, in natural units, simply $D = 1/2$. If no importance sampling algorithm is used, such distribution can be sampled by making walkers move similarly to standard metropolis algorithm with a proposal

$$\mathbf{R}' = \mathbf{R} + \boldsymbol{\eta}\sqrt{2D\Delta t},$$

with $\boldsymbol{\eta}$ such that for each particle, η is a d-dimensional Gaussian. Then, the acceptance criteria follows

$$a(\mathbf{R}, \mathbf{R}') = \min\left(1, \frac{G(\mathbf{R}', \mathbf{R}, \Delta t)}{G(\mathbf{R}, \mathbf{R}', \Delta t)}\nu_M\right),$$

with ν_M the metropolis metric. However, as already mentioned, DMC requires a more guided sampling process, which we introduce now.

3.4.1 Langevin Metropolis Importance Sampling

Often in quantum-mechanical simulations, the probability distribution to be inferred from sampling can be somewhat localised. Then, a significant portion of the stochastic integration steps gets wasted in regions that are not representative of the target distribution. More critically, unguided Monte Carlo sampling leads DMC walkers towards regions of infinite potential and unstable results.

The Langevin Metropolis importance sampling method aims at guiding the acceptance probability to the target distribution of the trial wave function. This is done by introducing a quantum force or drift force, as explained in depth in [13]. Under the action of a drift force, $\mathbf{F}$, the evolution of a probability distribution $P(\mathbf{R}, t)$ can be modelled by the Fokker-Planck equation:

$$\frac{\partial P(\mathbf{R}, t)}{\partial t} = D\nabla\left[(\nabla - \mathbf{F}(\mathbf{R}))P(\mathbf{R}, t)\right]. \tag{3.22}$$

The relation between $\mathbf{F}$ and P can be further investigated under some assumptions. It can be shown [22] that since the force must act in the orthogonal direction of the diffusion, and given that in the stationary state, the left-hand side of Eq. 3.22 must be zero, the quantum force can be expressed as $\mathbf{F} = P^{-1}\nabla P$. In this case, for a particle k at position $\mathbf{R}_k$,

$$\mathbf{F}(\mathbf{R}_k) = \frac{2\nabla\Psi(\mathbf{R}_k)}{\Psi(\mathbf{R}_k)}.$$

Invoking now the Langevin equation [92], which tells us how a stochastic variable can evolve under the time evolution of the distribution, a proposal $\mathbf{R}' = \mathbf{R} + \boldsymbol{\mu}$ is given by

$$\boldsymbol{\mu}(\mathbf{R}, \Delta t) = D\mathbf{F}(\mathbf{R})\Delta t + \xi\sqrt{\Delta t}, \tag{3.23}$$

with ξ following a d-dimensional Gaussian distribution and Δt follows the similar role as the step-length in regular Metropolis. The importance sampling algorithm then expresses the distribution

g in the acceptance rule of Eq. 3.6 as the solution for the Fokker-Planck equation, which is the Green's function

$$G_d(\mathbf{R}', \mathbf{R}, \Delta t) = \frac{\exp\left(-\frac{(\mathbf{R}'-\mathbf{R}-D\Delta t\mathbf{F}(\mathbf{R}))^2}{4D\Delta t}\right)}{(4\pi D\Delta t)^{3N/2}}, \tag{3.24}$$

which corresponds precisely to the diffusion component of the Green's function in the DMC context, G_d, now incorporating a drift term. If we are using a VMC algorithm instead of DMC, there is by definition no branching part of the general Green's function. In that case, we can express the modified acceptance criteria of this method as

$$\begin{aligned} a(\mathbf{R}, \mathbf{R}') &= \min\left(1, \nu_{LMH}(\mathbf{R}, \mathbf{R}')\right), \\ \nu_{LMH} &= \frac{G_d(\mathbf{R}', \mathbf{R}, \Delta t)}{G_d(\mathbf{R}, \mathbf{R}', \Delta t)}\nu_M. \end{aligned} \tag{3.25}$$

In words, the importance sampling guidance works by adjusting the acceptance probability such that the metric of Eq. 3.16 is multiplied by the fraction of two Green's functions that model the transition probability between state $\mathbf{R}' \to \mathbf{R}$ of a probability distribution evolving due to a diffusion and a quantum force. In contrast to Metropolis, this algorithm adds, both in Eq. 3.24 and Eq. 3.23, physical information about the trial wave function, indicating at least in theory a more robust method.

A Comment on DMC

One of the advantages of DMC over VMC is that, in VMC, occasional excited states can contribute to the minimised energy via local minima, whereas in DMC calculations, excited states exponentially decay to the ground-state in the imaginary time evolution.

Despite being extremely precise, DMC also presents a couple of drawbacks. First, there is the constant problem of computational cost. For bosonic systems, DMC scales polynomially with the system size, but for fermionic systems, the scaling is exponential. Furthermore, and perhaps more important, is the fermionic sign problem: DMC can only be performed assuming wave functions that are positive at any evaluated point. Although techniques are used to circumvent this, such as the fixed node approximation [3], they all bias the samples in some way.

Chapter 4

Machine Learning Background

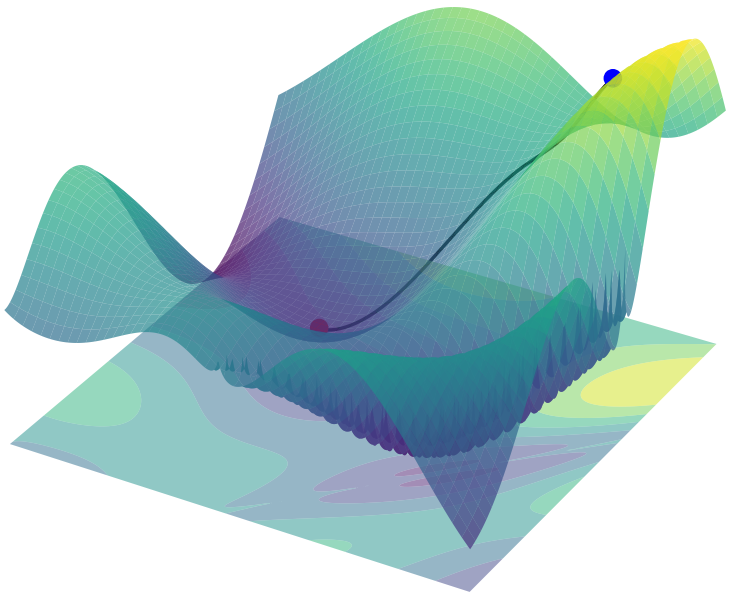


Figure 4.1: A gradient descent trajectory failing to optimise to global minima on the Eggholder function.

The name machine learning, combined with its current overuse, can sometimes be misleading. Machine learning methods need no modern machines as we know them today. Given a pen, paper, and enough time, anyone can do machine learning. More precisely, what these methods do need is a computational machine - be it a human or a modern computer. Of course, the computer here would have an advantage in terms of automation and floating-point operations per second, and that is why we use them.

We will often refer to machine learning as statistical learning, as this name is more informative and helps to reduce some mysticism around the field.

4.1 Statistical Learning

Statistical learning involves utilising statistics to extract knowledge from data. We therefore must clarify the meaning of data and learning in this context. First, there are various types of learning, typically categorised into supervised, unsupervised, and reinforcement learning. In supervised learning, the aim is to create a model that takes labelled examples as input and predicts outputs. The goal of unsupervised learning is to identify patterns or structures within the data, without explicit labels in the inputs. Lastly, reinforcement learning is distinct in that an agent makes decisions and interacts with an environment to obtain rewards based on its actions.

Consider a function $f : X \to Y$, where X is a set of possible inputs with Y the set of all possible outputs. Here, data refers to subsets or sample vectors x_i and y_i from the respective sets, which makes $\{x_i, y_i\} \in X \times Y$. Learning, then, is the process of creating a model $\hat{f}(x_i)$ that model f up to some error ϵ,

$$\hat{f}(x_i) = y_i + \epsilon_i.$$

The error, or noise term, is a deviation of the model prediction from the actual output, and can occur from inherent fluctuations of the dataset or limitations of the model itself. In the case of an idealised model $\hat{f}$, we assume that ϵ follows a normal distribution $\mathcal{N}$, which is motivated by the law of large numbers.

Training a model involves searching for a model which minimises this error as much as possible. We are, however, free to choose how this error is measured, and, as we will discuss, that will greatly influence the minimisation search. For now, let us simply state that such a function, which represents a distance between the model's prediction and ground truth, should map to a real number. We will call it a loss function $\mathcal{L} : \mathcal{H} \to \mathbb{R}$, with $\mathcal{H}$ a set of all possible functions from input to output space[1]. For now, we simply write

$$\mathcal{L}(f, \mathcal{D}) \propto \sum_{(x_i, y_i) \in \mathcal{D}} d(f(x_i), y_i), \tag{4.1}$$

where $\mathcal{D} \subset X \times Y$, and d a measure of the distance of the individual predictions to the truth. For now, it suffices to state that it will be a convex function such as the mean squared error.

4.1.1 Learning as an Optimisation Problem

To quote Bennett [6], "Optimization problems lie at the heart of most machine learning approaches". On that note, the objective of our learning is to find a function that, given a data set $\mathcal{D} = \{(x_i, y_i)\}$ from a hypotesis space $\mathcal{H}$, minimises $\mathcal{L}$. In mathematical terms, the problem becomes

$$\min_{f \in \mathcal{H}} \mathcal{L}(f, \mathcal{D}). \tag{4.2}$$

However, this is only part of the problem. In statistical learning, we want models capable of generalising to unseen data. We want models to learn general features of datasets, rather than hyper-specialising on a specific one. This discussion is central to supervised learning and is deeply related to the concept of bias-variance trade-off, better discussed in [33]. This specialisation process usually comes with an increase in complexity of the model, which leads to a higher variance, albeit smaller bias in the prediction values. To better estimate that, one usually breaks the data set into a training and testing set. Here we deliberately avoid that discussion as this technique is not used in our methods.

[1] Note that this is not just any function, as the type of model adds constraints: it could be linear model, neural network, and so on.

4.2 Gradient-Based Optimisation

There are several ways to approach optimisation problems, such as the one described in Section 4.1.1. The technique of choice depends on the nature of the problem at hand, but perhaps the most famous is gradient-based optimisation. For a large class of problems, many functions we want to minimise - or maximise - are based on physically or statistically reasonable functions. For that reason, they are frequently well behaved in the differential calculus sense. If a specific loss function is sufficiently smooth[2], and has an extrema (minimum or maximum) at some point in the domain, we know $\nabla\mathcal{L} = 0$.

Note how the domain of $\mathcal{L}$ is a very general vector space of functions. To make the differentiation process more intuitive, we parameterise a model $f_{\boldsymbol{\theta}} : X \to Y$. This means that after fixing a functional form, we can explore the hypothesis space by changing the parameters $\boldsymbol{\theta}$, and so the optimisation quest becomes

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}, \mathcal{D}). \tag{4.3}$$

Steepest Descent

The idea behind steepest descent (SD) is to iteratively update $\boldsymbol{\theta}$ in a direction that minimises $\mathcal{L}$ and to do so using the gradient of the loss with respect to the parameters. The motivation is simple: this gradient should be related to the direction in which $\mathcal{L}$ decreases the most for the smallest displacement $\boldsymbol{\delta}_\theta = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t$. The word "related" carries a significant conceptual weight here, especially because we have not defined how to measure such displacement. For now, we shall use the Euclidean norm, in which case we write

$$d\left(\boldsymbol{\theta}_{t+1}\,,\,\boldsymbol{\theta}_t\right) \;=\; \left\|\boldsymbol{\delta}_\theta\right\|_2 .$$

Our wordy motivation for steepest descent can then be written mathematically if we first take a first order approximation of $\mathcal{L}$ around $\boldsymbol{\theta}_t$:

$$\mathcal{L}(\boldsymbol{\theta}) \approx \mathcal{L}(\boldsymbol{\theta}_t) + \boldsymbol{\delta}_\theta^\top \nabla\mathcal{L}(\boldsymbol{\theta}_t). \tag{4.4}$$

Performing an iterative minimisation with the smallest possible step can be posed as a constraint minimisation problem. If we set the distance to an arbitrary but fixed scalar, $d\left(\boldsymbol{\theta}_{t+1}\,,\,\boldsymbol{\theta}_t\right) = \epsilon$, the quest for the optimal parameter can be written

$$\boldsymbol{\theta}_{t+1} \;= \arg\min_{\boldsymbol{\theta}} \Big[\mathcal{L}(\boldsymbol{\theta}_t) + \boldsymbol{\delta}_\theta^\top \nabla\mathcal{L}(\boldsymbol{\theta}_t) \;+\; \epsilon\Big]\,. \tag{4.5}$$

We have constrained the minimization of $\mathcal{L}$ to a function d, which can be solved via Lagrange multipliers (see Appendix C), yielding

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha\nabla\mathcal{L}(\boldsymbol{\theta}_t). \tag{4.6}$$

The parameter α, called the learning rate, is a scaling factor from the Euclidian distance, together with the displacement constraint ϵ. In real-life applications, we often choose such a parameter experimentally. Using too big of a learning rate will mean that the constraint has not been satisfied, and steps are taken outside of the trust region which guarantees the linear approximation of the method.

Steepest descent has some pitfalls. First, all directions in parameter space are treated equally, in terms of scale, by the fixed learning rate. Depending on the parametrisation of our cost function, this might be a terrible assumption. Second, the learning rate is fixed; then, a large learning rate might make it impossible to reach convergence. Note that employing an extremely small learning rate is not practical either, as it would require more iterations to reach convergence and inevitably more computational time. Hereafter we will discuss some of the techniques to try and address these points.

[2] has well deffined derivatives up to some order k over some domain $\mathcal{D}$.

Newton's Method

The most immediate way to address the problem of equal treatment of the parameter scale is to use a higher-order Taylor expansion in Eq. 4.4. This is the essence of Newton's method. In this case, the update rule guided by the minimisation is expressed as

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \left[\mathcal{L}(\boldsymbol{\theta}_t) + \boldsymbol{\delta}_\theta^\top \nabla \mathcal{L}(\boldsymbol{\theta}_t) + \frac{1}{2} \boldsymbol{\delta}_\theta^\top \mathrm{H}(\boldsymbol{\theta}_t) \boldsymbol{\delta}_\theta + \epsilon \right], \tag{4.7}$$

for which the update rule becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \mathrm{H}^{-1}(\boldsymbol{\theta}_t) \nabla \mathcal{L}(\boldsymbol{\theta}_t) \tag{4.8}$$

$$\mathrm{H}(\boldsymbol{\theta}_t) = \nabla^2 \mathcal{L}(\boldsymbol{\theta}_t) \tag{4.9}$$

where H is the so-called Hessian matrix.

Newton's method improves the step towards the direction of the negative gradient of the loss function by adding information of the curvature of the objective function $\mathcal{L}$ via the Hessian matrix. While several of the theoretical arguments behind both SD and Newton's method are tailored to convergence in a convex landscape, they can be used in non-convex problems with some caution and techniques.

Convex functions are characterised by the property that the line segment connecting any two points on the function's graph always remains above the graph itself. A classical example being a parabola, optimisation on those functions is straightforward, as any local minima are also global minima. In contrast, non-convex functions do not have this guarantee, due to the potential presence of several local-minima or saddle points. In the latter class of functions, having bad initialisation points can in the worst scenario lead to divergence in the algorithm or lead to convergence to local minima.

At this point, one might ask why not always use Newton's method. As for most computational problems, the answer is both storage and computational efficiency. The Hessian matrix is quadratic in number of parameters and has to be updated and stored for every iteration of the training process. Calculating the elements of the matrix is also expensive, since it involves the double derivatives of the objective function, and finally, inverting a matrix also has approximately cubic complexity.

Momentum Gradient Descent

The steepest descent method of Subsection 4.2 with update rule given by Eq. 4.6, can be rewritten as

$$\begin{aligned} \boldsymbol{\delta}_t &= \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \boldsymbol{\delta}_t. \end{aligned}$$

A simple improvement in this method is the addition of a moment term $\gamma \in (0, 1)$. This parameter controls the maintenance of some information, or memory, about the convergence behaviour of the parameter update in previous epochs. The analogy arises as the moment of a particle increases, while its potential energy on a downward trajectory is minimised. The intuitive argument for this analogy is that, by having moment, such a particle traversing a landscape is able to overcome potential local minima or move ever so fast when in regions with a steady downhill trajectory. Following this reasoning, the algorithm follows

$$\begin{aligned} \boldsymbol{\delta}_t &= \gamma \boldsymbol{\delta}_{t-1} + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \boldsymbol{\delta}_t. \end{aligned}$$

This is an exponentially weighted average of displacements along the epochs, with the moment term γ controlling how much to remember from the previous iterations. This constant is

therefore a number between 0 and 1, to be fine-tuned experimentally. Note that this memory term invariably contains some information about the curvature of the landscape, while still being part of a first-order method.

Root Mean Squared Propagation (RMSprop)

We have mentioned that a constant learning rate can affect the convergence of the training. We now present two examples among the class of adaptative learning rate methods that attempt to address this problem.

To avoid the computational cost of taking the second derivative of the loss function, RMSProp [29] aims at correlating this information with the moving average on the squared gradient of each parameter, represented here as $\boldsymbol{s}_t$, and modulated by a decay rate β. This method assigns individual effective learning rates for each parameter and also modulates them by a decay rate. This makes the algorithm move fast in the beginning (large learning rate) when we are probably distant from the minima, and then proceed with caution as we hope to approach convergence:

$$\begin{aligned} \boldsymbol{s}_t &= \beta \boldsymbol{s}_{t-1} + (1-\beta)(\nabla_{\boldsymbol{\theta}} \mathcal{L} \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}, \end{aligned}$$

where ϵ serves as a stability factor to avoid division by 0, and is usually a number around 10^{-9}.

Adaptative Moment Estimation (Adam)

Another widely used method that takes into account the second moment of that gradient is Adam or Adaptive Moment Estimation [47]. In contrast to RMSProp, Adam considers both the first order ($\boldsymbol{m}_t$) and the second order moment ($\boldsymbol{s}_t$) of the gradient of each parameter, being a combination of momentum GD and RMSProp. Adam further includes bias corrections to these moments, denoted respectively $\hat{\boldsymbol{m}}_t$ and $\hat{\boldsymbol{s}}_t$. We will not derive the expression of these corrections, but the original authors of the paper argue for their motivation: since the first- and second-moment terms are exponentially weighted averages, their initialisation value of zero inevitably introduces a bias into their accumulation value.

Adam has been shown to be an extremely robust optimisation algorithm, which means that it performs well in a wide class of different problems [78, 102]. For our use and purposes, we simply state the update rules below,

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{s}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} \mathcal{L}, \\ \hat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1-\beta_2^t}, \\ \boldsymbol{s}_t &= \beta_2 \boldsymbol{s}_{t-1} + (1-\beta_2)(\nabla_{\boldsymbol{\theta}} \mathcal{L} \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}), \\ \hat{\boldsymbol{s}}_t &= \frac{\boldsymbol{s}_t}{1-\beta_2^t}, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\hat{\boldsymbol{s}}_t + \epsilon}} \odot \hat{\boldsymbol{m}}_t. \end{aligned}$$

Note that the first and second moments are modulated by different constants β_1, β_2, while ϵ has the same role as in RMSProp.

4.2.1 Stochastic Gradient Descent

Up to this point, the discussion about statistical learning had as a cornerstone the minimisation of the loss of Eq. 4.2, evaluated throughout the dataset $\mathcal{D}$. In reality, most deep learning methods are conducted with an evaluation of the loss and respective gradients on subsets of the

dataset. This technique is adopted not only for efficiency benefits but also for its impact on convergence. To illustrate it, we recall the simple update rule for the steepest descent, in Eq. 4.6. For the calculation of the gradient, the proportionality of Eq. 4.1 can be turned into equality by choosing a mean squared error approach. Recalling also that we are solving a parametrised problem, we write

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|D|} \sum_{(x_i, y_i) \in \mathcal{D}} \nabla_{\boldsymbol{\theta}} || f_{\boldsymbol{\theta}}(x_i) - y_i || \tag{4.10}$$

$$\approx \frac{1}{|\mathcal{B}|} \sum_{(x_i, y_i) \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} || f_{\boldsymbol{\theta}}(x_i) - y_i || \tag{4.11}$$

Here we denote $|\cdot|$ the number of elements in the dataset. Equation 4.10 involves the calculation of a number of derivatives that scale with the size of the dataset, which can become significant. On the other hand, stochastic gradient descent (SGD) makes use of the fact that this evaluation is an expectation, and approximates its result for a smaller subset $\mathcal{B} \subseteq \mathcal{D}$. This subset, called a batch, is often obtained by randomly sampling from the full set, with or without reposition. Then, instead of updating $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}_{t+1}$ only after the computation of the gradient on the entirety of the data set, we immediately update after calculating the gradient of the partial expectation of the sampled losses.

There is a clear compromise between a reduced computational cost for a smaller batch size and a worse approximation of expectation of the gradients. However, vectorisation techniques can be used to compute the gradient on moderately sized batches, which is not necessarily advantageous in the case of smaller batches.

Stochastic gradient descent is commonly accompanied by a scheduler for the learning rate, which means that one adds a step dependence to α, often an exponential decay. This is because it has been shown that, with an appropriate decay rate, SGD will most likely reach a global minimum if the objective function is convex or quasi-convex [48]. Even if that is not the case, it almost surely leads to a local minimum that is often good enough. All the above-mentioned gradient optimisation algorithms can be employed with the use of batches.

The choice of SGD is also motivated by some improvement in convergence of the minimisation problem. This was first observed empirically, and while the explanation is still disputed, it is generally accepted that the approximation of the gradient incurs in some randomness in the minimisation trajectory that helps the algorithm escape occasional local minima [9, 104].

4.2.2 Natural Gradient

We now present the slightly more niche technique of Natural Gradient Descent (NGD) [1, 2]. With the previous methods in mind, the formalism for NGD can be understood as an innocent preconditioning of the gradient. However, the conceptual arguments surrounding this method are profound.

Natural gradient descent belongs to the class of approximate second-order methods, which aim at more explicitly approximating the Hessian matrix of Newton's method. Interestingly, however, NGD can potentially yield better results than second-order methods [99] because the Hessian of the problem, which could be negative definite, is approximated by a positive semi-definite matrix.

To better understand NGD, it is important to bring the connection between two related perspectives of statistical learning and optimisation: empirical risk minimisation (ERM) and maximum likelihood estimation (MLE). Our approach to the task of training a model, motivated in 4.1.1, was based on ERM. There, learning was seen as the minimisation of prediction errors measured by the expected loss over a given distribution of data. This means that we want to minimise $\mathbb{E}[\mathcal{L}(f_\theta(x), y)]$ with f_θ being our model. The loss function $\mathcal{L}$ can sometimes be called a risk function.

In many other problems and for us specifically in the VMC scenario, it is beneficial to perceive the learning task from the point of view of MLE. In this frame of reference, we want to find parameters for a model f_θ such that the observed data are more probable. We then train the models by a minimisation task over the log-likelihood of the model $\ln(p_\theta(x))$ [33].

In our discussion of the steepest descent, the constraint minimisation problem of Eq. 4.5 was solved considering the notion that the distance between the parameters is the Euclidean distance. In the MLE framework, one approach is to guide the update rule by measuring distances between probability distributions, via the Kullback-Leibler (KL) divergence. If we consider two distributions that differ simply by a change $\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{\delta}_\theta$ in parameter, we can write

$$\begin{aligned} D_{KL}(p_{\boldsymbol{\theta}} \,||\, p_{\boldsymbol{\theta}'}) &= \mathbb{E}_{x\sim p_{\boldsymbol{\theta}}} \left[\ln(p_{\boldsymbol{\theta}}(x)) - \ln(p_{\boldsymbol{\theta}'}(x))\right] \\ &\approx \frac{1}{2}\boldsymbol{\delta}_\theta^\top F(\boldsymbol{\theta})\boldsymbol{\delta}_\theta, \end{aligned}$$

where the approximation comes after second-order expansion of D_{KL} around $\delta\boldsymbol{\theta} = 0$. In this expression, F is the Fisher information matrix (FIM), given by

$$F_{ij}(\boldsymbol{\theta}) = \mathbb{E}_{x\sim p_{\boldsymbol{\theta}}} \left[(\partial_{\theta_i} \ln(p_{\boldsymbol{\theta}}(x)))(\partial_{\theta_j} \ln(p_{\boldsymbol{\theta}}(x)))\right], \tag{4.12}$$

with x representing the observed data points, p is a probability distribution to be studied with respect to the parameterisation $\boldsymbol{\theta}$. In practical implementations, the expectation value of the FIM is approximated by stochastic sampling and is called the empirical Fisher.

With this notion of distance[3], the constraint minimisation problem now becomes

$$\boldsymbol{\theta}_{t+1} = \arg\min_\theta \left[\mathcal{L}(\boldsymbol{\theta}_t) + \nabla\mathcal{L}(\boldsymbol{\theta}_t)^\top \boldsymbol{\delta}_\theta + \frac{1}{2}\boldsymbol{\delta}_\theta^\top F(\boldsymbol{\theta})\boldsymbol{\delta}_\theta\right]. \tag{4.13}$$

Finally, this is allows us to write the parameter update scheme as simply

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha F^{-1}(\boldsymbol{\theta}_t)\nabla\mathcal{L}(\boldsymbol{\theta}_t), \tag{4.14}$$

again, with α the learning rate. This expression makes it clear how NGD can be interpreted naively because its expression resembles Newton's method update rule of Eq. 4.9. In fact, F is an approximation of the Hessian matrix, but Eq. 4.12 yields an optimisation scheme that is invariant in parametrisation. This comes from the parameterisation invariance in the KL divergence and helps mitigate issues such as large disparities in parameter scaling.

Of course, while approximate second-order methods might converge to minima faster than first-order methods, they are computationally more expensive, at least in clock time. For example, NGD adds the overhead of either computing and inverting a possibly large FIM or solving a linear system of equations, even if the full Hessian is not computed. When we discuss the use of the quantum analogue of NGD for VMC calculations, we will see that this point, while still important, is less crucial. There, the computational bottleneck cost is not as much the optimisation scheme, but the energy calculations and generation of proposals via high quality random number generation.

4.2.3 Quantum Natural Gradient

We now try to connect the concepts of variational Monte Carlo and with natural gradient descent from the perspective of stochastic reconfiguration (SR). As shown in [32], the Euclidean metric is not the optimal choice for the energy minimisation task of variational methods. Firstly, VMC does not fall under the typical supervised learning umbrella, therefore, reducing the objective function to $\mathcal{L} = \langle E_L \rangle$ requires specific considerations. This will be better understood in

[3]The KL divergence is technically not a true distance metric because it lacks symmetry. Nevertheless, the symmetry property is satisfied locally.

section Sec. 4.4 where we pose VMC as a reinforcement learning algorithm. The search for the best optimisation strategies for variational minimisation, especially with neural networks, is still an active field of research [20].

One of the first approaches to avoid an optimisation under the Euclidian metric was achieved by driving a constrained variational ansatz to the ground-state via imaginary time evolution as done in [62]. Later, Stokes et al. [88] more rigorously showed the link between the imaginary time evolution view of SR and its understanding as a quantum extension of NGD. Since then, SR and NGD have been extensively used in quantum variational Monte Carlo simulations and variational quantum circuits [74, 77, 44] with higher stability than previous methods.

Just as KL divergence and the FIM are invariant under reparametrisation, so are their quantum counterparts: the Fubini-Study distance [88] and the quantum geometric tensor. These provide an optimisation landscape that is inherent to the geometry of the densities. To provide a concrete example, consider a variational ansatz $\psi_{\boldsymbol{\theta}}$ mapping elements $\boldsymbol{\theta}$ of the parameter space to vectors in the Hilbert space. The distance of interest between elements in the Hilbert space is not the Euclidean distance in parameter space but the Fubini-Study distance. Then, given the state $\psi_{\boldsymbol{\theta}'} = \psi_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}$, it can be written:

$$D(\psi_{\boldsymbol{\theta}}, \psi_{\boldsymbol{\theta}'}) = \arccos \sqrt{\frac{\langle\psi|\psi_{\boldsymbol{\theta}'}\rangle\langle\psi_{\boldsymbol{\theta}'}|\psi\rangle}{\langle\psi|\psi\rangle\langle\psi_{\boldsymbol{\theta}'}|\psi_{\boldsymbol{\theta}'}\rangle}} \approx \frac{1}{2}\delta_{\boldsymbol{\theta}}^{\dagger} G(\boldsymbol{\theta})\delta_{\boldsymbol{\theta}}. \tag{4.15}$$

Similarly to how the KL divergence, when expanded to second order around $\delta\boldsymbol{\theta}$ yields the FIM, the Fubini-Study distance, when expanded to second order, yields the quantum geometric tensor G, also called the Fubini-Study metric:

$$G_{ij}(\boldsymbol{\theta}) = \mathbb{E}_{X\sim P=|\psi|^2}\left[\partial_{\theta_i^*} \ln \psi^*(X, \boldsymbol{\theta}^\dagger)\partial_{\theta_j} \ln \psi(X, \boldsymbol{\theta})\right] \tag{4.16}$$

$$- \mathbb{E}\left[\partial_{\theta_i} \ln \psi^*(X, \boldsymbol{\theta}^\dagger)\right] \mathbb{E}\left[\partial_{\theta_j} \ln \psi(X, \boldsymbol{\theta})\right]. \tag{4.17}$$

As shown in [88], in the case where the system is not dependent on a complex phase factor, the quantum geometric tensor is a multiple of the FIM, $4G_{ij} = F_{ij}$. This is of particular interest for our applications, as we deal only with real parameters of the trial wavefunction because of the stationary states studied and the bounded nature of the system. Also, since the proportionality constant can be included in the learning rate in the update rule, we proceed simply by employing the empirical FIM and using the update rule described by Eq. 4.14, with $p_{\boldsymbol{\theta}} = |\psi|^2$.

4.3 Artificial Neural Networks

The previous exposition on statistical learning was intentionally detached from the theory of artificial neural networks (ANNs). This is to show that the principles behind the methods discussed can be agnostic to the choice of a specific network architecture. Given that neural networks are a central part of the current work, we first introduce the concept of neural networks by justifying their name and talking briefly about their history. Typically, the treatment of the topic of neural networks is obfuscated by multi-layer perceptrons, a type of feed-forward network. In the current work, while we will include them, we also explore additional architectures.

In their most general form, ANNs are computational models represented by connected graphs: they contain nodes (vertices) and connections (edges). These nodes are called artificial neurones because their functionality is inspired, at least loosely, by biological neurones. The concept is based on the transmission of signals throughout the connections of the network, with modulation occurring at the nodes. While in biological systems the signal is electric, in an artificial neural net such a signal is usually a numerical value (analogue ANNs are also possible and have shown interesting use cases [70]).

If we consider a network in which neurones only output binary values, the analogy extends further: a signal of 0 can represent an inactive neurone, while 1 indicates an active one. Just like in logical gates, these simple individual operations form the basis to perform more complex tasks, as more layers are introduced and the operations are composed, allowing for non-linearity in the signal.

The schematics of this graph-like structure has a myriad of format variations, called architectures. In addition, networks can also differ in other aspects, such as activation functions. For a visual example of two of these architectural variations, we refer to Fig. 4.4 and Fig. 4.2.

4.3.1 Boltzmann Machines

Boltzmann machines (BMs) are a subset of a larger group of statistical learning models called generative models. Given two sets of data X and Y, while the more popular discriminative models try to capture the conditional probability $p(Y|X)$, generative models will learn the joint probability $p(X, Y)$ of the train data set, where we consider Y to be the target. Also categorised under the class of energy-based model, a BM ideally detects the latent variables of said data set and specialises in generating data that follow the probability distributions of the training set.

By latent variables, we mean variables that cannot be directly observed in the data but only indirectly inferred. Latent variables are the gems of dimensionality reduction in ML. Good latent variables are those that carry significant information about data without much loss of information. Indeed, this is the whole motivation behind manifold learning [24]. Most high-dimensional data are often artificially high-dimensional and can be projected to lower-dimensional (latent) manifolds without losing much information. For example, the number of degrees of freedom in a general image (pixel values) is incredibly high. However, when a topic for an image is set, this significantly reduces the dimension of the space.

More precisely, BMs are stochastic generative models inspired by the concept of Boltzmann distribution in statistical physics. Unlike in a conventional network, the concept of an output layer is not present in a BM and it is said to be undirected, as is clear from Fig. 4.2. Instead, there is a visible and a hidden layer of nodes assuming, originally, binary values. These nodes are connected, or modulated, by a set of weights and biases.

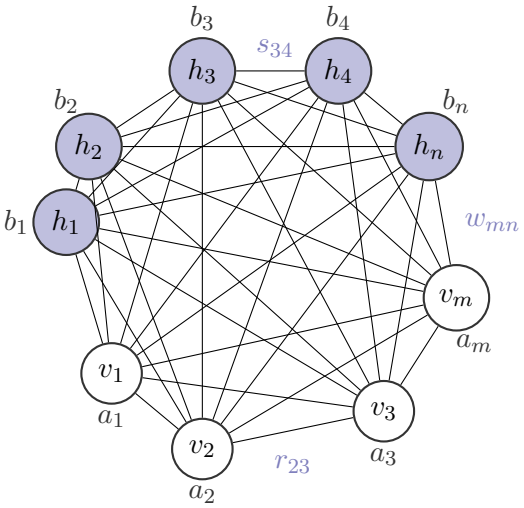


Figure 4.2: Schematics of a BM. The nodes coloured in light purple represent the hidden layer of the network. Letters s and r represent the intra-layer connection, while and w represents outer-layer connections.

Although generative networks can be applied to various tasks, image generation serves as a practical illustration. For example, when provided with a dataset of handwritten digits, the

network captures the probability distribution pattern of the pixel values. Then, after the training phase, these networks can produce samples that resemble those in the training set.

In BMs, the training process starts by sampling the values of the hidden and visible node vectors $\boldsymbol{h}$ and $\boldsymbol{v}$. The inferred probability distribution is then modelled by minimising an energy function $E(\boldsymbol{h}, \boldsymbol{v})$ - hence the name energy-based model. More specifically, the probability is a Boltzmann distribution,

$$P(\boldsymbol{h}, \boldsymbol{v}) = \frac{\exp\left(-E(\boldsymbol{h}, \boldsymbol{v})\right)}{Z}, \tag{4.18}$$

where Z is the partition function to guarantee the probability over the whole parameter space sums to 1. This is in general an intractable function to attain. Fortunately, as discussed in Sec. 3.2.1, if the sampling process is done following a Metropolis algorithm, the fraction between probability distributions ensures that the partition function is in fact not necessary. More explicitly, the energy function is given by

$$E(\boldsymbol{h}, \boldsymbol{v}) = -\boldsymbol{v}^\top R \boldsymbol{v} - \boldsymbol{v}^\top W \boldsymbol{h} - \boldsymbol{h}^\top S \boldsymbol{h} - \boldsymbol{a}^\top \boldsymbol{v} - \boldsymbol{b}^\top \boldsymbol{h},$$

where the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ serve as bias terms and the matrices R, W, and S, are modulating the connections between layers. More specifically, R modulates intra-layer connections in the visible layer, S plays an analogous role for the hidden layer, and W, between hidden and visible units. If we limit the network to only extra-layer connections, we have a restricted Boltzmann machine (RBM).

During training, the weights and biases of the model are updated aiming at maximisation of the likelihood of the model, for example via gradient descent over the negative log-likelihood.

Restricted Boltzmann Machines

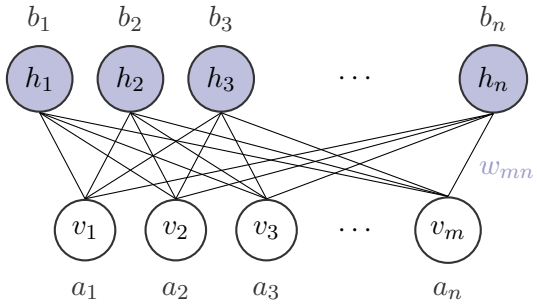


Figure 4.3: Schematics of an RBM. Modified from the code found in [65].

The RBM is the generative model for which we focus part of our work. By limiting the connections of the network, we obtain what can be seen in the schematics of Fig. 4.3. Regardless of this restriction, the probability function can still be expressed by Eq. 4.18, while the energy term will now be, for a binary-binary (E_{bb}) or a Gaussian-binary case (E_{gb}),

$$E_{bb}(\boldsymbol{h}, \boldsymbol{v}) = -\boldsymbol{v}^\top W \boldsymbol{h} - \boldsymbol{a}^\top \boldsymbol{v} - \boldsymbol{b}^\top \boldsymbol{h},$$
$$E_{gb}(\boldsymbol{h}, \boldsymbol{v}) = \frac{\|\boldsymbol{v} - \boldsymbol{a}\|^2}{2\sigma^2} - \boldsymbol{b}^\top \boldsymbol{h} - \frac{\boldsymbol{v}^\top W \boldsymbol{h}}{\sigma^2}.$$

Despite the modified architecture, the principle of training an RBM is analogous to that of BMs. However, training RBMs is considered an easier task than training other undirected models, as noted in [28], because the (unnormalised) conditional $P(\boldsymbol{h}|\boldsymbol{v})$ can be obtained with a closed expression.

As previously touched upon, the values of the hidden and visible nodes were originally conceived as binary sampled variables, but they can also assume continuous values. This choice determines the type of the RBM, and common choices are binary-binary or Gaussian-binary. As the name implies, the former admits only binary values to both hidden and visible units, while the latter takes normally distributed variables in the visible layer.

4.3.2 Feed-Forward Neural Networks

In its general definition, a feed-forward neural network (FFN) is a way to organise a set of compositions of parametrised functions in a sequential way. The way nodes are organised in layers and how they are connected results in different architectures. A visual example of a common FFN setup can be seen in Fig. 4.4.

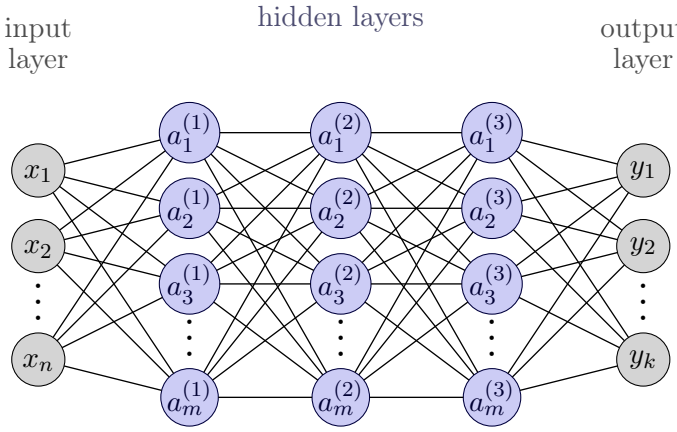


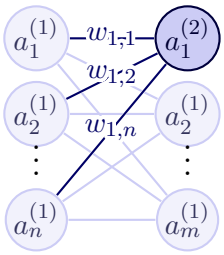
Figure 4.4: A generic multi-layer perceptron with n input nodes, k output nodes and a set of three hidden layers, all with m nodes. The superscript denotes the layer number, and a indicates that the value has been activated by some function σ. Adapted from [64]

.

In contrast to an RBM, this sequence controls the flow of information, or value propagation, in one preferential direction in the network. When an FFN is said to be densely connected, each neurone receives input from all other neurones in a previous layer and, after multiplying it by a set of parameters, adding a bias value, and composing it with some activation function σ, passes the result to neurones in the next layer. This process is called a forward pass of the network and, for an L-layer deep network, is better described in Eq. 4.19:

$$\left.\begin{aligned} &\mathbf{a}^{(0)} = \mathbf{x} \\ &\mathbf{h}^{(i)} = \mathbf{W}^{(i)} * \mathbf{a}^{(i-1)} + \mathbf{b}^{(i)} \\ &\mathbf{a}^{(i)} = \sigma_i(\mathbf{h}^{(i)}) \end{aligned}\right\} \qquad i \in \{1, ..., L-1\},$$
$$\hat{\mathbf{y}} = \sigma_L(W^{(L)} * \mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}). \tag{4.19}$$

Here we denote $\mathbf{W}^{(i)}$ and $\mathbf{b}_i$ as the weight matrix and bias vector for the i-th layer of the network. In addition, in Eq. 4.19, $\mathbf{h}^{(i)}$ represents the output of the i-th hidden layer, while $\mathbf{a}^{(0)} = \mathbf{x}$ is the input layer. Lastly, σ_i represents the activation function of each layer. Fig. 4.5 analyses the forward pass of the first hidden layer separately to provide a better understanding of the matrix operations involved in Eq. 4.19.



$$\begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \\ \vdots \\ a_m^{(2)} \end{pmatrix} = \sigma \left[\begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{pmatrix} \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_n^{(1)} \end{pmatrix} + \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_m^{(1)} \end{pmatrix} \right]$$
$$= \sigma \left(\mathbf{W}^{(1)} \mathbf{a}^{(1)} + \mathbf{b}^{(1)} \right)$$

Figure 4.5: Forward pass between the two first layers of the network. Here, the dependency of the weight matrix elements and the connected nodes is clear, together with the activation of the affine transformation. Adapted from [64].

The activation function σ is what allows the network, which is now our parametrised model $f_{\boldsymbol{\theta}}(\mathbf{x})$ to express non-linear functions and learn from non-linear data, present any interesting dataset. If not for activation functions, the forward pass described in Eq. 4.19 would simply be compositions of linear functions.

Common activation functions include the sigmoid function, which produces outputs between 0 and 1, the hyperbolic tangent, with outputs from -1 to 1. More recent alternatives such as the rectified linear unit (ReLU), aim at mitigating the problem of vanishing gradients. In our implementations, we opted mostly for the more recent Gaussian Error Linear Units (GELU) function [34], which provides enhanced performance in specific deep learning applications, and if given by

$$\mathrm{GELU}(x) = 0.5x \left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}} \right) \right).$$

Knowing which function to use is a complicated task and requires experimentation, as a good choice can greatly influence training efficiency and accuracy of the network.

The choice of the architecture of a network is often guided by the problem to which one aims to tackle, but most networks have some overlapping core structure in the form of the mentioned input, hidden, and output layers and activation functions. The weights and biases are then iteratively updated using a gradient scheme that tries to converge to a global minimum of the cost function $\mathcal{L}$. As mentioned in 4.2, this cost function is a measure of how well the model performs, be it in regression or classification problems.

Training An FNN

The task of supervised learning has some important differences from a usual minimisation problem. We are usually limited in the amount of training data, while in a minimisation problem, we might have access to the entire domain where the model should be optimised. Here we briefly describe the training of an FFN, but a more complete coverage of the topic can be found in [28]. Typically, we calculate and aim to minimise an average loss over a given training set, with an associated probability $\hat{p}$.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}} \left[\mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y) \right],$$

Within the empirical risk minimisation framework, we hope that by minimising this expected value, we will effectively generalise the probability distribution of the true data p. We intentionally do not focus much on this aspect, as in VMC there is no explicit separation between training and test sets. As this expectation value is empirically just an average in the training set, we sometimes use $\nabla\mathcal{L}$ and ∇J interchangeably.

Regardless of the loss of the model, the training process of an FFN will involve the following steps. First, we perform a forward pass to generate predictions. Next, we evaluate these predictions and compute the gradient of the loss function with respect to the parameters. Then, with this gradient information, we update the parameters accordingly, with any of the methods described in the section on gradient-based optimisation Sec. 4.2.

To ensure that the loss information flows backward through the network and updates the parameters correctly, we must consider the function compositions in Eq. 4.19 due to different layers in the network. This involves correct application of the chain rule during the gradient evaluations of $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, which will in fact depend on the activation functions that are used.

To handle flexible and arbitrary networks, gradients are often computed using automatic differentiation, which we explain in more detail in Sec. 5.2.1. This approach can sometimes obscure the backward propagation step. For clarity, we provide a simplified mathematical overview of the backpropagation equations, the demonstrations of which are available in Nielsen (2015) [66]. The gradient of the loss function with respect to the weights and biases of layer i are given by

$$\nabla_{\mathbf{W}^{(i)}}\mathcal{L} = \delta^{(i)}(\mathbf{a}^{(i-1)})^\top \tag{4.20}$$

$$\nabla_{\mathbf{b}^{(i)}}\mathcal{L} = \delta^{(i)}, \tag{4.21}$$

where $\delta^{(i)}$ represents a measure of the loss at layer i, given by

$$\delta^{(i)} = \begin{cases} \nabla_{\hat{\mathbf{y}}}\mathcal{L} \odot \sigma_L'(\mathbf{h}^{(L)}) & \text{if } i = L, \\ (\mathbf{W}^{(i+1)})^\top \delta^{(i+1)} \odot \sigma_i'(\mathbf{h}^{(i)}) & \text{if } i < L. \end{cases}$$

Due to the chain rule evaluation order, $\delta^{(i)}$ depends on $\delta^{(i+1)}$ and we must start from the last layer and progress inward. The expressions further simplify given the knowledge of the activation functions σ. A depiction of the training is given by Algorithm 2.

Algorithm 2 Training a Feedforward Neural Network

Input: Training data $(\mathbf{x}, \mathbf{y})$, initialized weights $\mathbf{W}^{(i)}$, biases $\mathbf{b}^{(i)}$, and learning rate α
for each training iteration **do**
  Forward Pass:
  for each layer $i \in \{1, \ldots, L-1\}$ **do**
    Compute pre-activation: $\mathbf{h}^{(i)} = \mathbf{W}^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}$
    Compute activation: $\mathbf{a}^{(i)} = \sigma_i(\mathbf{h}^{(i)})$
  end for
  Compute output: $\hat{\mathbf{y}} = \sigma_L(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)})$
  Computate loss: $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
  Backward Pass:
  Compute output layer error: $\delta^{(L)}$
  for each layer $i \in \{L-1, \ldots, 1\}$ **do**
    Backpropagate error: $\delta^{(i)}$
  end for
  Gradient Computation:
  for each layer $i \in \{1, \ldots, L\}$ **do**
    Compute gradients $\nabla_{\mathbf{W}^{(i)}}\mathcal{L}$ and $\nabla_{\mathbf{b}^{(i)}}\mathcal{L}$
  end for
  Parameter Update:
  for each layer $i \in \{1, \ldots, L\}$ **do**
    Update weights: $\mathbf{W}^{(i)} \leftarrow \mathbf{W}^{(i)} - \alpha\nabla_{\mathbf{W}^{(i)}}\mathcal{L}$
    Update biases: $\mathbf{b}^{(i)} \leftarrow \mathbf{b}^{(i)} - \alpha\nabla_{\mathbf{b}^{(i)}}\mathcal{L}$
  end for
  Check for convergence or stopping criteria
end for

A Comment on the Capabilities of ANNs

When talking about the learning capabilities of neural networks, we must mention universal approximation theorems. ANNs are compositions of parametrised functions, but how can one ensure that they are capable of approximating an ideal model $\hat{f}$, directing the statistics of the observed data? In fact, there are a set of universality theorems which provide this guarantee under certain conditions.

Cybenko showed in 1989 [15] that multi-layer perceptrons with sigmoid functions can approximate any continuous function. The theorem, proven for at least one hidden layer, does not mention any constraint on the number of epochs or neurones required. This means that such universality could in practice be infeasible. Further developed theorems exist that eliminate the requirement of a sigmoid activation function [56], and others have shown that the universal approximation theorem also applies to convolutional networks, recurrent networks, and more recently graph neural networks [103, 82, 11].

Being able to approximate an arbitrary function is only half the problem. Why would such a model be able to generalise to unseen data? To quote R. Grosse (2021) [30]: "After all, the optimisation landscape is non-convex, highly non-linear, and high-dimensional, so why are we able to train these networks? In many cases, they have far more than enough parameters to memorize the data, so why do they generalise well? [...] the attitude of the neural net community was to train first and ask questions later. Apparently, this worked."

In practice, even if the convergence of a problem is not guaranteed, we often reach "good enough" solutions with NNs. It is not uncommon to use methods for which there is no theoretically solid argument other than documented success. Fortunately, theoretical proofs have so far been presented in the years that follow.

4.4 Reinforcement Learning and VMC

Various forms of reinforcement learning (RL) exist, yet they all address an optimisation problem with the concept of agents. These agents, through a set of actions $\mathcal{A} = \{a_t\}$, are able to influence the environment states $\mathcal{S} = \{s_t\}$. Their goal is to maximise the accumulation of immediate rewards $r_{a_t}(s_t, s')$ for adopting a state s'.

Contrary to supervised learning, in which the stream of input for a model comes from a fixed data set, in reinforcement learning, such a dataset is iteratively dependent on the previous samples. In this sense, we say that the agent interacts and changes the environment, making more explicit the need for balance between exploration and exploitation to train a model.

To try and maximise a cumulative reward, an agent can take random actions to explore configuration space (exploration), but also use information from current and previous decisions to be rewarded (exploitation). Too much exploration leads to multiple sub-optimal choices, while too much exploitation results in local minima from limited prior knowledge.

In VMC, there is an element of exploration when modelling a probability distribution by accepting steps with a guided acceptance rule. Furthermore, VMC uses information from previous samples to minimise the expected value of the local energy, which is cumulative by construction. Not occasionally, reinforcement learning can be modelled by a Markov process, with an associated transition probability $t_a(s, s')$ from s to s' under an action a.

The goal of RL agents is to learn a parametrised policy distribution $\pi_{\boldsymbol{\theta}} : \mathcal{A} \times \mathcal{S} \to [0, 1]$, which, once optimised, will produce a stationary Markov chain distribution that maximises the expected cumulative reward over time. Consider the Markov chain of state and action $\tau_t = (s_t, a_t)$. There are several ways of measuring this expected cumulative reward, one of which being

$$\begin{aligned} \mathcal{J}(\boldsymbol{\theta}) &= \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} [G_t] \\ G_t &= \sum_{k=0}^{\infty} \gamma^k r_{a_{t+k+1}} \end{aligned}$$

with $\gamma \in [0, 1)$ a discount return rate, so that immediate rewards are more important for step t than ones far in the future. This expectation value is usually computed via Monte Carlo averaging, and there are multiple approaches to optimise this reward, such as gradient ascent, with neural networks as agents. Then, it can be shown [89] that

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}} [G_t \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\tau)] .$$

Unfortunately, and this is a problem we also face in VMC, the gradients of expectation values of the estimated loss/reward function have very high variance, motivating the use of approximate second-order methods like NGD, but also techniques like the REINFORCE [97] and a variance reduction technique called the baseline method. When these are used, the gradient is calculated in a more stable way [28] by subtracting a baseline $b(\boldsymbol{\theta})$

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) \propto \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} [(G_t - b(\boldsymbol{\theta})) \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\tau)] ,$$

where b is simply an offset that does not depend on on the action. There are techniques to further find the baseline to best reduce the variance, but it is sometimes simply set as the expected reward $\mathcal{J}(\boldsymbol{\theta})$. Here we see a stark parallel between the gradient of the cumulative reward and the gradient of the expected local energy of the quantum system:

$$\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = 2\mathbb{E}_{\mathbf{R} \sim |\psi_\theta|^2} [(E_L - E(\boldsymbol{\theta})) \nabla_{\boldsymbol{\theta}} \ln \Psi_{\boldsymbol{\theta}}(\mathbf{R})] .$$

At this point, we see it is at least possible to adapt RL methods to improve the VMC scheme. Evidently, some dissimilarities must also be mentioned. As noted in [58], the gradient expressions in VMC involve quantum amplitudes instead of probability, and furthermore, the probability

of the quantum state is often not normalised. However, the analogy between neural network reinforcement learning and VMC remains. Here, the neural networks that suggest quantum states act as agents, with the environments offering feedback in the form of the energy of the state. The parameters are then adjusted to minimise this energy, implying that the reward can correspond to the negative of the local energy.

4.4.1 Neural Quantum States

In 2017, Carleo and Troyer were the first to introduce a method in which larger neural networks are used as a variational ansatz for quantum states [12]. The method, which they called neural quantum states (NQS), was conceptualised using a restricted Boltzmann machine as the network, aiming at solving interacting spins models. Now, NQSs have been applied to a large set of physical problems and using many common types of neural networks, such as feed-forward, convolutional, recurrent, graph neural networks, and more [52]. At the core of these methods is a variational minimisation task under a reinforcement learning framework.

Similar variational ansätze, such as matrix product states [72], have been used previously, but with limited generalisability to different quantum systems. The motivation for employing neural networks is rooted in their representational power, as guaranteed by universal approximation theorems, combined with their capacity for dimensionality reduction. In fact, the field of machine learning has faced challenges analogous to those of quantum mechanics when it comes to the curse of dimensionality.

To describe neural quantum states, we follow [52] and proceed in a similar way to our VMC section, Sec. 3.3. A general state in a basis set of the possible configurations $\{|\boldsymbol{\sigma}\rangle\}$ is written

$$|\psi\rangle = \sum_{\boldsymbol{\sigma}} \psi(\boldsymbol{\sigma}) \, |\boldsymbol{\sigma}\rangle \, ,$$

where $\boldsymbol{\sigma}$ could be spin configurations, as posed originally by Carleo, or positions, in our case. Then, different configurations are sampled and are the inputs to the network, which serve as the coefficients $\psi(\boldsymbol{\sigma})$. The neural quantum states can be expressed as

$$\psi_\theta(\boldsymbol{\sigma}) = \sqrt{p_{\boldsymbol{\theta}}(\boldsymbol{\sigma})} e^{i\phi_{\boldsymbol{\theta}}(\boldsymbol{\sigma})},$$

with amplitude $p_{\boldsymbol{\theta}}(\boldsymbol{\sigma}) = |\psi_{\boldsymbol{\theta}}(\boldsymbol{\sigma})|^2$ and phase $\phi_{\boldsymbol{\theta}}(\boldsymbol{\sigma}) = \mathrm{Im}(\ln \psi_{\boldsymbol{\theta}}(\boldsymbol{\sigma}))$. Different methods exist for handling the phase of the wavefunction. One approach involves using two distinct networks or separate outputs: one for the imaginary component and another for the real component. Alternatively, the network can be trained with complex parameters. This approach increases the size of the parameter space, and since we deal with bounded and time-independent systems, we chose to disregard the imaginary part of the wavefunction altogether.

Then, the expectation values of observables are approximated via Monte Carlo averaging, in complete analogy to the variational Monte Carlo approach:

$$\langle \hat{O} \rangle \approx \sum_{\boldsymbol{\sigma}} P_{\boldsymbol{\theta}}(\boldsymbol{\sigma}) O_{L,\boldsymbol{\theta}}(\boldsymbol{\sigma}),$$

with the normalized configuration probability $P_{\boldsymbol{\theta}}(\boldsymbol{\sigma})$ and the local operator estimator, $O_{L,\boldsymbol{\theta}}(\boldsymbol{\sigma})$ already defined in Sec. 3.3. Again, if one uses sampling algorithms such as Metropolis, the ansätze used in the sampling need not be normalised - which is very useful for deep networks, where calculating the partition function is difficult.

Training an NQS, as most of the interesting machine learning problems, is a non-convex task. Therefore, machine learning techniques and optimisers are beneficial in the parameter update routine. A generic NQS training algorithm is shown in Algorithm 3, where we show that NQSs are trained without the use of external training data. In the displayed case, we use the space configurations of the system, $\boldsymbol{\sigma} = \mathbf{R}$.

The details of our specific RBM, FFN and Deep Set FFN implementations, together with the standard VMC ansatz, can be seen in the Methods part, Sec. 5.1.

Algorithm 3 Neural Network VMC with MCMC Sampling. The parameters $\boldsymbol{\theta}$ are updated based on the local energy $E_L(\mathbf{R}_i)$ calculated for sampled configurations. The Metropolis acceptance ratio A and uniform probability distributions U_c are used to accept or reject proposed configurations. All particles are moved at once.

Initialize the state parameters and position $\psi_{\boldsymbol{\theta}}(\mathbf{R}) \leftarrow \boldsymbol{\theta}_0$, $\mathbf{R}_0$
for each epoch **do**
  for each MCMC step **do**
    $\mathbf{R}' \leftarrow$ Propose new configuration based on $\mathbf{R}$ and $\boldsymbol{\theta}$
    $A \leftarrow$ Compute Metropolis acceptance ratio $A(\mathbf{R} \rightarrow \mathbf{R}') = \min\left(1, \frac{|\Psi_{\boldsymbol{\theta}}(\mathbf{R}')|^2}{|\Psi_{\boldsymbol{\theta}}(\mathbf{R})|^2}\right)$.
    $r \sim U_c(0,1)$ (Draw uniform number)
    if $A \geq r$ (Acceptance criteria) **then**
      $\mathbf{R} \leftarrow \mathbf{R}'$ (Accept new configuration)
    end if
  end for
  $\{\mathbf{R}_i\} \leftarrow$ Sample batch of configurations by MCMC
  $E_L(\mathbf{R}_i) \leftarrow$ Calculate local energy for each $\mathbf{R}_i$
  $\nabla_{\boldsymbol{\theta}}\langle E_L\rangle \leftarrow$ Compute gradient with respect to $\boldsymbol{\theta}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\nabla_{\boldsymbol{\theta}}\langle E_L\rangle$ Update parameters
  Check for convergence or stopping criteria
end for

Part II

Methods

Chapter 5

Methods and Implementations

In this section, we describe our implementations and the specific techniques employed to achieve our results, along with a brief theoretical background when needed.

5.1 Trial Wavefunctions

5.1.1 Standard VMC Ansatz

Our simplest variational ansatz is a parametrised Gaussian and involves no neural network. For a system of N particles and dimension d, with a collective position vector flattened as $\mathbf{R} \in \mathbb{R}^{N \cdot d}$, we allow for $N \cdot d$ variational parameters and write

$$\psi_{\boldsymbol{\theta}}^{(\text{vmc})}(\mathbf{R}) = \exp\left[-(\mathbf{R} \odot \mathbf{R})^{\top}\boldsymbol{\theta}\right], \tag{5.1}$$

where $\odot$ represents element-wise product. When dealing with the log of the wavefunction, we can simply write

$$\ln|\psi_{\boldsymbol{\theta}}^{(\text{vmc})}(\mathbf{R})| = -\sum_{i=1}^{Nd}\theta_i r_i^2.$$

If $\theta_i = 0.5$ for every i, this ansatz accurately represents the ground-state wave function of a non-interacting system of bosons at zero Kelvin. It is, however, incapable of satisfying particle-exchange antisymmetry, no matter the choice of parameters, and also fails to satisfy Kato's cusp condition. The antisymmetry requirement can be satisfied by multiplying the ansatz by a Slater determinant, while the cusp condition is addressed via a Jastrow factor ($\mathcal{J}$) or a Padé-Jastrow factor ($\mathcal{P}$), which will be discussed shortly.

The Slater determinant could be any determinant of single-particle orbitals, for example obtained from the Hartree-Fock calculation or the ground state for the non-interacting problem Φ_0, which is how we proceed. We showed in Sec. 2.6 that, for the fermionic problem, the single particle functions $\{\phi_j(\boldsymbol{r}_i)\}$ are a product of a Gaussian envelope and a product of Hermite polynomials for each coordinate $H_n(\boldsymbol{r}_i) \equiv H_{n_x}(x_i)H_{n_y}(y_i)\cdots$, where n represents the quantum number. Then, for the VMC ansatz we let the parametrised gaussian be the envelope, and write the VMC Slater ansatz,

$$\ln|\Psi_S^{(\text{vmc})}| = \ln|\psi_{\boldsymbol{\theta}}^{(\text{vmc})}(\mathbf{R})| + \ln|\det\{H_j(\boldsymbol{r}_i)\}|. \tag{5.2}$$

As we discuss in Sec. 2.6, the block diagonal form of $\{\phi_j(\boldsymbol{r}_i)\}$ further allows us to split it into a product of smaller spin-up and spin-down determinants. Additionally, we will not carry the Slater subscript alone, as every use of the ansatz for fermions will necessarily be accompanied by a determinant of Hermite polynomials.

Jastrow Factor and Padé-Jastrow Factor

The log ansätze, including the Jastrow factor and Padé-Jastrow factor, follow respectively,

$$\ln|\Psi_{SJ}^{(\mathrm{vmc})}| = \ln|\Psi_{S}^{(\mathrm{vmc})}| + \mathcal{J}_{\boldsymbol{\alpha}}(\mathbf{R}), \tag{5.3}$$

$$\ln|\Psi_{PSJ}^{(\mathrm{vmc})}| = \ln|\Psi_{S}^{(\mathrm{vmc})}| + \mathcal{P}_{\alpha}(\mathbf{R}). \tag{5.4}$$

Both $\mathcal{J}$ and $\mathcal{P}$ introduce inter-particle correlation and depend on the inter-particle distance $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ as

$$\mathcal{J}_{\boldsymbol{\alpha}}(\mathbf{R}) = \sum_{i=1}^{N}\sum_{j>i}^{N} r_{ij}\alpha_{ij},$$

or

$$\mathcal{P}_{\alpha}(\mathbf{R}) = \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{a_{ij}r_{ij}}{1+\alpha r_{ij}}.$$

Note the single variational parameter α in $\mathcal{P}$ instead of a vector. In addition, the factors a_{ij}, which depend on the particles i and j, are chosen to enforce the ansatz to satisfy Kato's cusp conditions, described in Sec. 2.6. For a two and three-dimensional fermionic system, and representing the spin coordinate of particle i as σ_i, these values were determined by Huang et al. [38], and follow

$$a_{ij} = \begin{cases} \frac{1}{d+1} & \text{if } \sigma_i = \sigma_j, \\ \frac{1}{d-1} & \text{else.} \end{cases}$$

It should be mentioned that, while we define the Slater-Jastrow and Padé-Slater-Jastrow factors under the VMC ansatz subsection, it can be added to any of the other ansätze presented below.

5.1.2 Restricted Boltzmann Machine Neural Quantum States

Given the continuous character of the Monte Carlo sampled values, the restricted Boltzmann machine (RBM) of choice should be Gaussian-Binary. The sampled values for the RBM's visible nodes will indeed correspond to the positions of the particles whose probability distribution is related to the wave function that we aim to obtain. Thus, these will be denoted as $\mathbf{R}$ rather than $\boldsymbol{v}$ used in Sec. 4.3.1.

Moreover, the physical quantity we want to sample is related to the marginal probability distribution of the visible units, $p(\mathbf{R})$, while the hidden node's values are related to latent variables that do not represent physical quantities of interest. Consequently, we obtain the marginal distribution of interest by integrating $p(\mathbf{R}, \boldsymbol{h})$ over the hidden nodes' degrees of freedom,

$$\begin{aligned} p(\mathbf{R}) &= \frac{1}{Z}\sum_{\{\boldsymbol{h}\}} e^{-E_{gb}(\mathbf{R},\mathbf{h})} \\ &\propto \sum_{\{\boldsymbol{h}\}} \exp\left\{-\left(\frac{\|\mathbf{R}-\boldsymbol{a}\|^2}{2\sigma^2} - \boldsymbol{b}^\top\boldsymbol{h} - \frac{\mathbf{R}^\top W\boldsymbol{h}}{\sigma^2}\right)\right\}. \end{aligned} \tag{5.5}$$

The proportionality comes from the fact that we do not need the partition function due to the Metropolis sampling. Moreover, we will deal only with hidden units with binary values, so Eq. 5.5 further simplifies, as shown in Appendix D. Finally, if we model the probability density of a wavefunction as the marginal distribution probability of the visible units of the RBM, we can write

$$\left|\psi_{\boldsymbol{\theta}}^{(\mathrm{rbm})}(\mathbf{R})\right|^2 = \exp\left\{-\sum_{i}^{Nd}\frac{(r_i-a_i)^2}{2\sigma^2}\right\}\prod_{j}^{n_h}\left[1+\exp\left\{b_j+\sum_{i}^{Nd}\frac{r_i w_{ij}}{\sigma^2}\right\}\right], \tag{5.6}$$

where n_h is the number of hidden units, and σ is the standard deviation of the Gaussian distribution that the visible units are assumed to follow. In Eq. 5.6, $\boldsymbol{\theta}$ serves as a container for all parameters, and for the use of the log wavefunction,

$$\ln\left|\psi_{\boldsymbol{\theta}}^{(\mathrm{rbm})}(\mathbf{R})\right| = -\frac{1}{2}\sum_{i}^{Nd}\frac{(r_i - a_i)^2}{2\sigma^2} + \frac{1}{2}\sum_{j}^{n_h}\left[1 + \exp\left\{b_j + \sum_{i}^{Nd}\frac{x_i w_{ij}}{\sigma^2}\right\}\right]. \tag{5.7}$$

Once again, on its own this trial function does not obey symmetry, antisymmetry, or cusp conditions. Then, the combination of this ansatz with a Slater determinant, or correlation factors is done in analogy to the VMC, equations 5.2, 5.4 and 5.3,

$$\begin{aligned}
\ln\Psi_{S}^{(\mathrm{rbm})} &= \ln|\psi_{\boldsymbol{\theta}}^{(\mathrm{rbm})}(\mathbf{R})| + \ln|\det\{H_j(\boldsymbol{r}_i)\}|\,, \\
\ln\Psi_{SJ}^{(\mathrm{rbm})} &= \ln\Psi_{S}^{(\mathrm{rbm})} + \mathcal{J}_{\boldsymbol{\alpha}}(\mathbf{R}), \\
\ln\Psi_{PSJ}^{(\mathrm{rbm})} &= \ln\Psi_{S}^{(\mathrm{rbm})} + \mathcal{P}_{\alpha}(\mathbf{R}).
\end{aligned}$$

Lastly, we should mention that to avoid exploding gradients, $\boldsymbol{a}$, $\boldsymbol{b}$, and W were initialised in a scale inversely proportional to the size of the visible layer, such as $\mathcal{N}_{(0,0.1)}/\sqrt{Nd}$. A similar approach was taken for the feed-forward ansatz, which is our next model.

5.1.3 Feed-Forward Neural Quantum States

Feed-forward neural networks (FFNs)have also been used to model quantum systems, albeit in a slightly different way than RBMs. This difference comes from the fact that FFNs are not probabilistic or generative models in the same way as RBMs. As there is a clear notion of input and output in this network, we could define the neural network with one output as a function $f : \mathbb{R}^{Nd} \to \mathbb{R}$ and write a simplified ansatz as

$$\psi_{\boldsymbol{\theta}}(\mathbf{R}) = \exp\{f_{\boldsymbol{\theta}}(\mathbf{R})\}, \tag{5.8}$$

which means that in the log domain, the ansatz is simply the network. Now, if the neural net only admits real parameters, as we defined, the exponential nature of the ansatz could never satisfy antisymmetry under the exchange of particles. There are several methods to guarantee that antisymmetry is satisfied with feed-forward NQSs, with perhaps the most famous implementation being the FermiNet, by David Pfau et. al [77]. Other successful implementations are also possible and often based on a similar architecture [57, 44]. The implementation of Pfau et. al, is more intricate than the one we bring here, but a standard Slater-Jastrow ansatz can usually be written

$$\psi_{\boldsymbol{\theta}}(\mathbf{R}) = e^{\mathcal{J}_{\boldsymbol{\alpha}}(\mathbf{R})}\sum_{k}\omega_k \det\{\phi_i^{k\uparrow}(\boldsymbol{r}_j^{\uparrow})\}\det\{\phi_i^{k\downarrow}(\boldsymbol{r}_j^{\downarrow})\}, \tag{5.9}$$

where ω_k are simply weights of the sum. Note that there is not necessarily an explicit separation between a Gaussian envelope and a determinant of polynomials, as the generalised single-particle orbitals $\phi_i^k(\boldsymbol{r}_j)$ can be fully parametrised by the network.

In first quantisation, the evaluation of several Slater determinants according to Eq. 5.9 is computationally expensive, but it is also possible to use neural networks in combination with second quantisation formalism [52]. Commonly, the Slater determinants are parametrised by the network which learn transformations that make the single-particle orbitals depend on the configuration, called backflow transformations [37].

We opt to ensure the symmetry of particle exchange using a network architecture based on Deep Sets [101]. Although this architecture can be used to obtain equivariant layers [46, 44], thus ensuring antisymmetry at the architectural level, we do not follow this approach. Instead,

we utilised only its permutation invariant implementation and rely on one Slater determinant for antisymmetry.

A permutation-invariant Deep Set consists of writing a neural network as a composition of two functions and a pooling layer in between. We provide here a specific definition of a Deep Set tailored to our applications, rather than a universal definition. For our case, the Deep Set can be written as $\psi_{\boldsymbol{\theta}}^{(\mathrm{ds})} = \rho(\Sigma(\phi))$ where $\phi : \mathbb{R}^d \to \mathbb{R}^{d_L}$ and $\rho : \mathbb{R}^{d_L} \to \mathbb{R}$. Here, Σ represents any pooling operation over N, such as the average, for each latent dimension. This can be written $\Sigma : \mathbb{R}^{Nd_L} \to \mathbb{R}^{d_L}$. Moreover, L_d represents a latent dimension, and the architecture can be seen in Fig. 5.1.

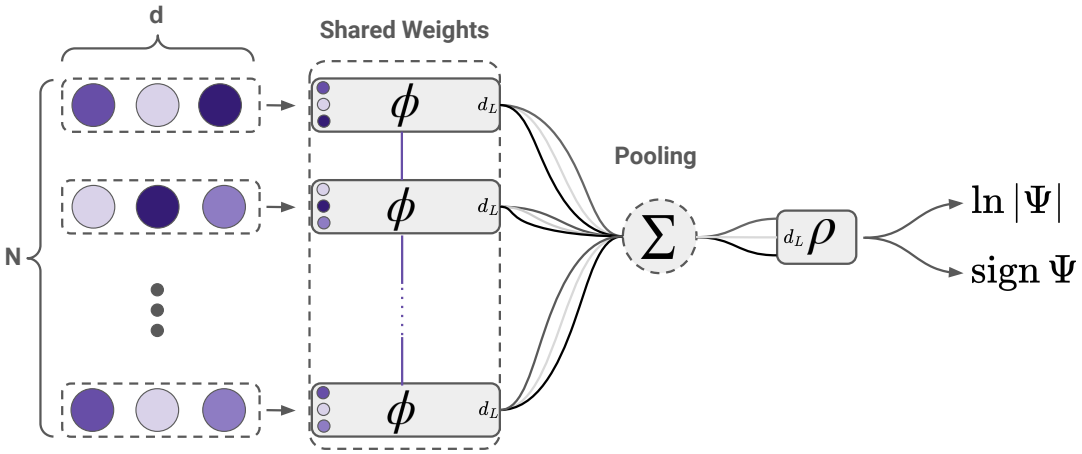


Figure 5.1: Representation of a permutational invariant Deep Set architecture. Figure inspired by [46].

The core idea is that the pooling layer eliminates the specific particle-coordinate association, while the shared weights could theoretically learn inter-particle correlation and encode it to the latent space. This approach scales better than a brute-force symmetrisation in which one sums the outputs for all permuted configurations.

Then, in our code we proceed to use this architecture as the Gaussian envelope, exactly as in the aforementioned methods. Once again, $\psi_{\boldsymbol{\theta}}^{(\mathrm{ds})}(\mathbf{R})$ can be multiplied by a Slater-Jastrow or a Padé-Slater-Jastrow factor, yielding

$$\begin{aligned}
\ln \Psi_S^{(\mathrm{ds})} &= \ln|\psi_{\boldsymbol{\theta}}^{(\mathrm{ds})}(\mathbf{R})| + \ln|\det\{H_j(\boldsymbol{r}_i)\}|\,, \\
\ln \Psi_{SJ}^{(\mathrm{ds})} &= \ln \Psi_S^{(\mathrm{ds})} + \mathcal{J}_{\boldsymbol{\alpha}}(\mathbf{R}), \\
\ln \Psi_{PSJ}^{(\mathrm{ds})} &= \ln \Psi_S^{(\mathrm{ds})} + \mathcal{P}_{\alpha}(\mathbf{R}).
\end{aligned}$$

5.1.4 One and Two-Body Densities

One typical way to analyse many-body correlations is via the one-body density matrix (OBDM),

$$\rho(\boldsymbol{r}_1', \boldsymbol{r}_1) = N \int d\boldsymbol{r}_2 \ldots d\boldsymbol{r}_A \Psi^*(\boldsymbol{r}_1', \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N)\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N),$$

which describes the probability amplitude of finding a particle at position $\boldsymbol{r}_1$ while another one is at $\boldsymbol{r}_1'$. For simplicity of visualisation, we have mainly been generating profiles of the diagonal

of the OBDM, or the one-body density profile:

$$n(\boldsymbol{r}_1) = \rho(\boldsymbol{r}_1, \boldsymbol{r}_1) = N \int d\boldsymbol{r}_2 \dots d\boldsymbol{r}_N \left|\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \dots, \boldsymbol{r}_N)\right|^2 .$$

These were obtained by keeping the post-training Monte Carlo sampled positions and displaying on an extra axis the count of how many instances fall under each bin after a discretisation of the space. This will yield a regular histogram for a one-dimensional system or a three-dimensional histogram for a two-dimensional system. The histogram can be normalised so that the density profile integrates to N. This is easily done with the NumPy `histogram` function. For the three-dimensional histogram, we further smoothed the plot via a Gaussian filter.

Similarly, the two-body density matrix (TBDM), is a good way to identify higher-order correlation, and is given by

$$\rho(\boldsymbol{r}_1', \boldsymbol{r}_2', \boldsymbol{r}_1, \boldsymbol{r}_2) = N(N-1) \int d\boldsymbol{r}_3 \dots d\boldsymbol{r}_A \Psi^*(\boldsymbol{r}_1', \boldsymbol{r}_2', \dots, \boldsymbol{r}_N)\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \dots, \boldsymbol{r}_N).$$

Its diagonal gives the probability of simultaneously finding two particles at these designated positions $\boldsymbol{r}_1, \boldsymbol{r}_2$:

$$n^{(2)}(\boldsymbol{r}_1, \boldsymbol{r}_2) = N(N-1) \int d\boldsymbol{r}_3 \dots d\boldsymbol{r}_N \left|\Psi(\boldsymbol{r}_1, \boldsymbol{r}_2, \dots, \boldsymbol{r}_N)\right|^2 .$$

These quantities can be obtained similarly to the OBD profile, where we now bin unique pairs of particle positions. The caveat is that, if using Cartesian coordinates, we would gain a four-dimensional plot, impossible to visualise. However, we can proceed as [69] and use the radial symmetry of the harmonic trap, reduce the two spatial coordinates to a radial one and disregard the angle. For the sake of visualisation, we then replicate the profile obtained in the four quadrants.

5.2 Computational Differentiation

The methods we employ throughout the thesis have a heavy dependence on the calculations of differentials.

When the derivative of a function is either unknown or we prefer not to calculate it analytically, there are various methods available through computational tools. We will consider computational differentiation to encompass finite difference (FD) methods, symbolic differentiation (SD), and automatic differentiation (AD). When analytical expressions are available, it is advisable to use them rather than rely on computational derivative calculations. This approach will not only be usually quicker but also mitigate the inherent loss of precision that comes with other computational methods.

Each of the above three methods has its advantages and drawbacks. The following is a brief generalisation of some of the important points. One can generally say that symbolic differentiation is inefficient but powerful in that it gives the form of the derivative function to the user. SD is also the best computational method when it comes to sparsity (for example, when a Jacobian matrix is sparse).

Numerical differentiation, together with the related finite difference methods, allows for some control over the precision versus performance trade-off. FD gives a quantification of the precision loss and good generalisability to higher-order differentiation, yet it is dependent on an appropriate and problem-specific choice of step size. The wrong choice leads to large round-off errors in what is called catastrophic cancellation.

For our calculations, except in cases where analytical expressions are used, the method of choice will be AD. AD is very fast and precise and is especially suited to gradient-based optimisations. This is because AD is fast when dealing with partial derivatives with respect to

multiple inputs. AD operates by taking advantage of the computational graph that is created when a computer composes simpler arithmetic operations to compute any function. The presence of this composition-like structure in a computer programme enables a straightforward application of the chain rule of partial derivatives to arrive at the final solution. To do that, of course, the programme needs to have access to the function itself, so this method is not applicable in situations where the function is a black box[1].

5.2.1 Automatic Differentiation

Automatic differentiation exists in two variations: forward mode and reverse mode. Understanding this is relevant, especially to explain why our Hessian implementation is actually very efficient. Consider a function $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}^m$, where we write $\mathcal{L}(\boldsymbol{x_0}) = \boldsymbol{y}$. We will denote its Jacobian evaluated at a point $\boldsymbol{x_0}$ in the domain as $\partial_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x_0}) \in \mathbb{R}^{m\times n}$, or in matrix form:

$$\partial_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_0) = \begin{pmatrix} \partial_{x_1} y_1 & \partial_{x_2} y_1 & \cdots & \partial_{x_n} y_1 \\ \partial_{x_1} y_2 & \partial_{x_2} y_2 & \cdots & \partial_{x_n} y_2 \\ \vdots & \vdots & \ddots & \vdots \\ \partial_{x_1} y_m & \partial_{x_2} y_m, & \cdots & \partial_{x_n} y_m \end{pmatrix}.$$

That function, when calculated in a machine, will be a result of the composition of, for example, k simpler functions $\mathcal{L} = f_k(\cdots f_3(f_2(f_1(\cdot))))$. When applying the chain rule to evaluate the total derivative of $\mathcal{L}$, it is then

$$\partial_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_0) = \left(\left(\left(\left(\partial_{f_{k-1}} f_k \partial_{f_{k-2}} f_{k-1}\right)\cdots\partial_{f_2} f_3\right)\partial_{f_1} f_2 \partial_{\boldsymbol{x}}\right) f_1(\boldsymbol{x_0})\right) \tag{5.10}$$

$$= \left(\partial_{f_{k-1}} f_k \left(\partial_{f_{k-2}} f_{k-1} \cdots \left(\partial_{f_2} f_3 \left(\partial_{f_1} f_2 \partial_{\boldsymbol{x}} f_1(\boldsymbol{x_0})\right)\right)\right)\right) \tag{5.11}$$

The distinction in computational order between Eq. 5.10 and Eq. 5.11 is exactly what differentiates forward mode AD from backward mode AD. When one evaluates the computations from inside-out in terms of the order of the compositions, this is called forward mode.

The Jacobian itself is a linear transformation and can be seen as a function $\partial f : \mathbb{R}^n \to \mathbb{R}^m$. Furthermore, the chain rule of the Jacobians is, in fact, a set of matrix multiplications. Therefore, while being mathematically equivalent, forward and backward modes do not have the same computational complexity in terms of floating-point operations. Which one is more efficient depends on the problem at hand. For example, if $x \in \mathbb{R}^{N_0}, f_1 \in \mathbb{R}^{N_1}, \ldots, f_k \in \mathbb{R}^{N_k}$, backward AD is typically faster if $N_k > N_0$. If $N_k < N_0$, forward AD is favoured [10].

From Sec. 3.3 we see that VMC will rely on several Laplacian computations, which can be obtained via traces of Hessian matrices. When calculating elements of a Hessian, performing a forward mode over a reverse mode AD or the other way around would both be possible paths. In machine learning applications, however, the function we differentiate often has a wide Jacobian, since a loss function commonly maps to a real number, $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$. That is why several AD functions are built in reverse mode. Despite that, a Hessian calculation involves the Jacobian of a Jacobian. Since the inner Jacobian will commonly be wide, differentiating it will be better suit for forward mode. Then, forward-over-reverse tends to be more efficient, and this is how we proceed.

There is of course a case to be made about memory cost. Generally speaking, forward AD is favoured when a neural network is very deep because backward AD requires the storage of more Jacobian matrices.

[1] Not in the same sense as machine learning black-box but in the sense that the computer has no access to the instructions to be performed for the function output calculation.

5.3 Just in Time Compilation

Our current work is done in the Python programming language. This choice is motivated by numerous advantages that Python offers for scientific programming and machine learning, one of which is flexibility, but to the detriment of speed. Although it has had significant improvements in performance in the last years, it still falls short in comparison to purely compiled languages like C++ and Fortran.

Python has elements of compiled and interpreted languages, and the process of running a Python script can be broadly divided into two parts. First, the source code is compiled by CPython into a lower-level language, bytecode. Bytecode is simply a platform-independent representation of the code but is not yet machine code, and therefore Python is not compiled in the traditional sense. Subsequently, there is an interpretation stage, where the bytecode is fed to a virtual machine. This final stage includes processes such as dynamic type checking and memory management components, along with certain features that offer flexibility but can compromise performance.

Fortunately, with specific tools and some change in code structure, it is possible to compile bytecode to machine code of some functions at runtime - a process called just-in-time (JIT) compilation. JIT compiled functions can be run directly by hardware, helping with interpretation overhead and allowing compiler optimisation and accelerated linear algebra (XLA) techniques to take place. This is particularly valuable in scientific programming, where functions are essentially static, often computationally costly, and must be executed countless times in a simulation.

The design choices that allow for the JIT compilation depend on the framework we use to do so, and there are several Python options. We better explain our JAX [10] implementation in Sec. 5.4.6, together with several points that require extra care when implementing JIT compilation.

5.4 Codebase Overview

All the code work for this thesis can be found in the Github repository [16] and, as it can be used as a Python package, a documentation page is available at [17]. The aforementioned pages also contain instructions on how to install the package and run the code. A conceptual representation of the codebase is illustrated in Fig. 5.2. Please note that this diagram does not perfectly map to the repository's modules and is intended to facilitate conceptual understanding. For a detailed view, we refer to the documentation page. We further disclaim that all results were generated on a Mac equipped with an M2 Pro chip and 16 GB of RAM. This machine displayed faster running times than any of the three clusters tested.

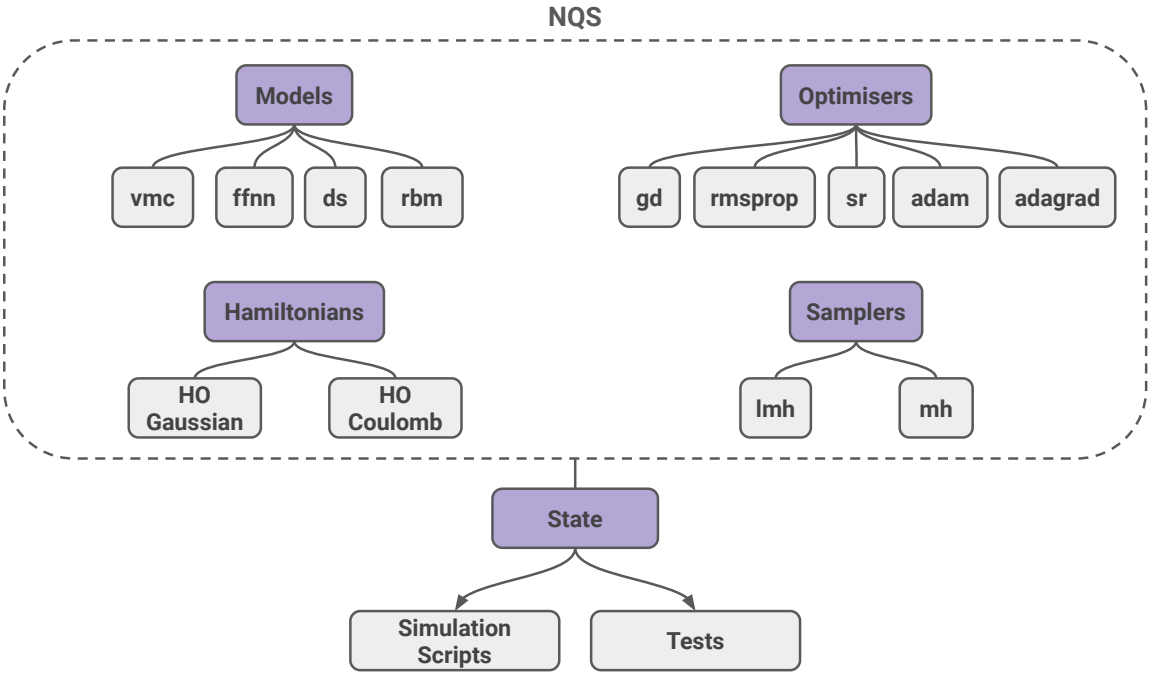


Figure 5.2: Conceptual overview of the NQS codebase, which can be found at [16]. See text for explanations.

5.4.1 Models

In the project, the term 'models' refers to various choices of ansätze. We have implemented essentially four variants, but only show results for 3 of them. The options are a standard parametrised variational Monte Carlo (VMC), a restricted Boltzmann machine (RBM), a deep-set variant of a feed-forward neural network, and a standard feed-forward neural network for which we do not show results. All of these are implemented to represent the natural logarithm of the absolute value of the wavefunction, for which the sign can always be retrieved with NumPy's function `slogdet`. All models can also be accompanied by a Jastrow factor or a Padé-Jastrow factor.

5.4.2 Optimisers

Following the idea of a modular project, all opitimisers can be used with every model or system, and five options are included: gradient descent (`gd`), adaptative moment estimation (`adam`), root mean square propagation (`rmsprop`), adaptative gradient estimation (`adagrad`) and stochastic reconfiguration (`sr`). Their individual hyperparameters can also be freely modified via the same `set_optimzier` method.

It should be mentioned that the implementation of stochastic reconfiguration, when used for multilayer networks such as `dsffn` and `ffnn`, approximates the fisher information metric by a block diagonal matrix, as per [61], which we better explain in Sec. 5.7. We chose to continue calling it `sr` to keep consistency, since the VMC and RBM models technically do not use this approximation.

We here display the `set_optimzier` method, and the associated `optimizer_factory`.

```
def set_optimizer(self, optimizer, eta, **kwargs):
    """
    Set the optimization algorithm for parameter updates in the NQS simulation.

    Parameters:
        optimizer (str): The optimizer to use (e.g., 'adam').
```

```
        eta (float): The learning rate.
        **kwargs: Additional keyword arguments for the optimizer.

    Raises:
        ValueError: If an unsupported optimizer is specified."""
    self._eta = eta
    common_args = {
        "params": self.wf.params,
        "eta": eta,
    }
    self._optimizer = optimizer_factory(optimizer, **common_args, **kwargs)
```

The `optimiser_factory` is responsible for returning the correct class without polluting the main `NQS` class.

```
from nqs.optimizers import Adagrad, Adam, Gd, RmsProp, Sr


def optimizer_factory(opti_type, **kwargs):
    opti_type = opti_type.lower() if isinstance(opti_type, str) else opti_type

    match opti_type:
        case "gd":
            return Gd(**kwargs)
        case "adam":
            return Adam(**kwargs)
        case "rmsprop":
            return RmsProp(**kwargs)
        case "adagrad":
            return Adagrad(**kwargs)
        case "sr":
            return Sr(**kwargs)
        case _: # noqa
            raise NotImplementedError(
                f"No options for {opti_type}, Only the gd, adam, rmsprop, adagrad and sr
                    supported for now."
            )
```

5.4.3 Hamiltonians

The `Hamiltonian` class allows for flexibility in the setup of the physical problem. Any type of particle interaction or external potential can be easily set up. Although it only contains one child class, `HarmonicOscillator`, the type of interaction can be controlled and passed via the `set_hamiltonian` method from the `NQS` class.

The class is accessed when the method `local_energy` is invoked by the selected sampler, and the measurement is conducted on the wavefunction model object, which is our ansatz.

5.4.4 Samplers

The `Sampler` base class has as child classes `Metropolis` and `MetropolisHastings`, where the latter implements the Langevin Metropolis-Hastings importance sampling described in Sec. 3.4.1.

The base class manages common functionalities, such as setting up the random number generator, logging, and handling the scale parameter for sampling resolution, while the child classes perform the method-specific sampling step. The base class also commands the spawning of parallel but individual Markov chain walkers together with the subsequent collection of statistics from the sampling.

It must be mentioned that our Monte Carlo samplings move all particles at the same time instead of one at a time. Although this is more efficient in terms of random number generation

(a large bottleneck in any Monte Carlo calculation), it impedes us from using common tricks to speed up the determinant calculation, such as calculating the determinants via cofactors and updating individual determinant matrix columns.

5.4.5 State

The State module combines all the pieces of the project in a large class, `NQS`. There, some things can be chosen via their appropriate set methods, such as samplers, models, and optimisers. Although de do not want to include all these methods extensively here, their idea can be illustrated via the set optimser and `set_wf`, the latter being responsible for choosing the ansatz.

```
def set_wf(self, wf_type, nparticles, dim, **kwargs):
    """
    Set and initialize the wave function for the NQS simulation.

    Parameters:
        wf_type (str): The type of wave function to use.
        nparticles (int): The number of particles in the system.
        dim (int): The dimensionality of the system.
        **kwargs: Additional keyword arguments for the wave function initialization.

    Raises:
        ValueError: If an invalid wave function type is provided.
    """
    self.N = nparticles
    self.dim = dim
    self._particle = kwargs.get("particle", "none")
    common_args = {
        "nparticles": self.N,
        "dim": self.dim,
        "rng": self.rng(self._seed) if self.rng else np.random.default_rng(),
        "logger": self.logger,
        "logger_level": "INFO",
        "backend": self._backend,
    }
    specific_args = kwargs
    self.wf = wf_factory(wf_type, **common_args, **specific_args)
    self.nqs_type = wf_type
    self._is_initialized_ = True
```

5.4.6 Backend

One of the first things to choose in the NQS class is the back-end, for which the two available options are NumPy [31] and JAX. In the former, the analytical expressions for the gradients and Laplacians of the ansatz are used, which enables a significant speedup in comparison to JAX automatic differentiation (AD). However, having the analytical expressions for the gradients is not always straightforward and can limit the great flexibility of deep networks. In our code, using JAX as a back-end is always possible, while NumPy is only allowed for the standard VMC and RBM ansätze, as their analytical expressions can be easily obtained beforehand.

Fortunately, JAX's just-in-time (JIT) compilation speeds up the code considerably, especially when implemented correctly with vectorised maps (vmaps). While we explain AD and JIT in Sec. 5.2 and Sec. 5.3, vmaps are essentially JAX's way of using vectorisation to gain efficiency and readability.

Unfortunately, extracting JAX's full power requires dealing with some "sharp bits", in the own words of the project developers. Although we acknowledge that our code may not follow the absolute best practices, it is still relevant to address some of these challenging aspects. For a

deeper explanation, we highly recommend the source code and documentation of the framework [10].

First, JAX transformations and compilations only work with functions without side effects. This means that "jitted" functions, for example, should not modify class attributes or methods if those are not passed as input parameters. This can seem straightforward, but is sometimes difficult to implement when the code design is around object orientation. This requirement of functionally pure code is the reason behind some of the unusual choices in our code, and can be difficult to debug.

Moreover, JAX arrays are immutable and inplace updates need to be substituted from `x[idx] = y` to `x.at[idx].set(y)`. In fact, JAX functions do not deal with NumPy arrays. Fortunately, they are automatically converted to JAX arrays with little to no overhead.

The minor overhead is the reason we were lax with our array initialisation. For best practices, it would have been preferable to specify the type of array the code is handling from the start, but more often than not, we opted to start with NumPy arrays and let JAX handle the rest. This point is also relevant because we decided not to use JAX's PRNG even when using it as a back-end.

JAX's RNG is significantly slower than Numpy's defaults, and while there are very good reasons for it, that was not relevant for how we developed our code. We instead leave PRNGs with NumPy, use JAX only for JIT, vmaps and AD, and suffer the overhead from multiple NumPy-to-JAX conversions. In our experience, this was still preferable to using JAX's PRNG.

5.4.7 Parameter Class

We implement the neural nets from scratch, and without pre-built neural network modules. Then, since we want to have a consistent structure for all the possible variational ansatz, we decided to have a specific parameter class.

The `Parameter` class is initialised with a dictionary where the keys are strings representing parameter names, and the values are the parameter data, which can be either NumPy arrays or JAX arrays. This allows for a flexible and extensible structure to hold various types of parameter data.

```
ParameterDataType = Union[np.ndarray, jnp.ndarray]

class Parameter:
    def __init__(self, data: Dict[str, ParameterDataType] = None) -> None:
        self.data = data if data is not None else {}
```

The class provides several methods for setting, getting, and manipulating parameter data. The set method is particularly versatile, allowing for setting parameters using different types of input: Replacing the entire parameter data with another Parameter instance, setting multiple parameters using lists of names and values, updating or adding parameters using a dictionary or setting a single parameter using a name and value pair.

```
def set(
    self,
    names_or_parameter: Union[
        str, List[str], "Parameter", Dict[str, ParameterDataType]
    ],
    values: Union[ParameterDataType, List[ParameterDataType]] = None,
) -> None:
    # Method implementation...
```

Then, the get method retrieves the value of a parameter specified by name, in the optimisation step. To exemplify, a step of a gradient descent with momentum follows:

```
def step(self, params, grad_params_E):
    for key, grad in grad_params_E.items():
```

```
3         self.v[key] = self.gamma * self.v[key] + grad
4         params.set([key], [params.get(key) - self.eta * self.v[key]])
```

One of the key aspects of this class is its integration with JAX's tree utilities, which is essential for enabling JAX's automatic differentiation and other optimisations. The `tree_flatten` and `tree_unflatten` methods are implemented to allow the Parameter class to be used with JAX's `jit`, `grad`, and other transformations. The `tree_flatten` method breaks down the Parameter object into a list of its values (leaves) and a list of keys (auxiliary data). This decomposition is necessary for JAX to understand and manipulate the data structure during computation.

Finally, the `Parameter` class is registered with JAX using the `register_pytree_node` function. This registration tells JAX how to handle instances of the `Parameter` class during its computations.

5.4.8 The Wavefunction Base Class

The `Wavefunction` base class is inherited by any choice of ansatz or model, allowing for modularity. Its design ensures that common functionalities are handled in one place while allowing specific behaviours to be defined in child classes. It, for example, controls, via abstract class methods, the setput of the Slater determinants and the Jastrow factors regardless of the trial function. This is because specific methods are called in the constructor of its child classes.

Very importantly, too, this class enables the setup of the appropriate backend (NumPy or JAX), and the just-in-time compilation for the appropriate functions.

5.4.9 Simulation Scripts

The simulation scripts in this repository manage the entire simulation process. They outline the steps for initialising models, configuring Hamiltonians, choosing and executing samplers, carrying out optimisation, and also gathering the results. These scripts are created to be modular and flexible, enabling easy modification of parameters and settings. This means that almost all options can be arbitrarily combined. For example: any model can take any particle type and use any optimiser, any sampler, and any Hamiltonian. We refer to Appendix E for a minimal display of one of these simulation scripts.

5.5 General Training Strategies

In this section, we present a variety of implementation techniques that were crucial to achieving satisfactory results.

5.5.1 Pretraining and Regularised Potential

Throughout our implementations, we make use of the method of transfer learning. Transfer learning consists in loading the weights of a model trained in a similar problem to accelerate the convergence when using the network in other scenarios. More specifically, this was a crucial step in obtaining reasonable convergence when minimising the energy of interactive systems with multilayer network ansatz. In dimensions larger than 1, we did not manage to get stable sampling without it, giving positions in a clearly unbounded way.

First, we pretrain the network to a supervised regression task, in which we regress the log-probability $\ln(|\psi|^2)$ of the ansatz to the log of a multivariate Gaussian with identity covariance matrix of dimension $(N \times d) \times (N \times d)$, where N is the number of particles, and d the number of dimensions. This is to ensure that the network at least starts with a function with a vanishing profile for large coordinate values.

The second stage involves either pretraining the network to the non-interactive related system, or training in a regularised potential that converges to the true interaction potential towards the end of the training process.

Both these second-stages consist in training the model within the standard VMC - reinforcement learning framework, but the details vary based on the system. If the particle interaction does not exhibit singularities, we pretrain the ansatz to converge to the non-interacting system. Else, we regularise the potential, adopting a method inspired by [46]. While this regularisation and pretraining are different techniques, the regularisation process essentially introduces the interaction slowly and in a specific way that hopefully helps the network learn Kato's cusp condition without the need to impose a fixed Jastrow factor. This regularisation consists of using a potential

$$\frac{1}{r_{ij}} \to \frac{\tanh(r_{ij}/r_0)}{r_{ij}}$$

where r_0 is a hyperparameter. The numerator in this regularisation can be any function of r_{ij} such that $f(r_{ij})$ approaches 1 as r_{ij} approaches 0. More specifically, in our implementations we add a decay rate τ to r_0,

$$\tau = \left(\frac{1}{3r_0}\right)^{2/T}, \tag{5.12}$$

such that, at half of the training process, $\tanh(r_{ij}/r_0) \approx 0.97$. Here, T is the total number of training epochs, and at each iteration, we change $r_0(t) = r_0(0) \cdot \tau^t$, where t is the current epoch. When sampling the observables from the final ansatz after training is done, we simply turn off this regularisation so that the ansatz is kept fixed.

5.5.2 Sampler Tuning

The quality of a Monte Carlo sampling is heavily influenced by the width of the distribution that is sampled from. In our implementations, this width is simply called the scale of the Monte Carlo method. In order to obtain an efficient sampling process, one usually aims at finding a sample width that yields approximately 50% of accepted moves. To achieve that, we created a `tune_scale` method based on the PyMC library [71], but with some minor modifications.

In the PyMC implementation, the method adjusts the proposal scale for a sampler based on the acceptance rate, aiming for an optimal range between 20-50%. Our approach differs by targeting an acceptance rate of 50-70%. Furthermore, our method activates only when the acceptance rate falls outside the 30-70% range, to avoid unnecessary tuning. These target values were determined experimentally, without a rigorous experimentation process.

The tuning process operates via a lookup table that adjusts the proposal scale according to predefined acceptance rate thresholds. If the acceptance rate is extremely low ($<$0.001) or extremely high ($>$0.99), we reinitialize the sampling positions and the weights of the ansatz without stopping the training process. This decision comes from observing that such acceptance rates indicate that the walkers have wondered into regions of impossible convergence, often due to an ill-defined ansatz.

When tuning the sampler, some considerations must be made. One of these is the batch size used to evaluate the acceptance rate. This batch size can be larger than the training batch size for higher precision in the tuning phase or smaller for faster execution. Another key consideration is whether to exit the tuning process if the optimal acceptance rate range is not met.

To address this last point, we devised two methods for the tuning process. The standard method allows setting a maximum number of iterations for the tuning process. If the acceptance rate does not fall within the optimal range within these iterations, the process stops. In contrast, the infinite method continues the tuning indefinitely until the acceptance rate falls within the desired range.

5.5.3 Clipping Gradients and Energy Values

Lastly if should be mentioned that we implemented clipping strategies in two instances to obtain numerical stability. First, following an approach similar to [77], the local energy values during network training are clipped according to the ℓ_1 norm. Specifically, the average local energy $\langle E_L \rangle$ is kept only if it falls within the range $\langle E_L \rangle \pm 5 \times \langle |E_i - \langle E_L \rangle| \rangle$.

Another strategy that was implemented by us but not thoroughly examined was gradient clipping. Gradient clipping consits in rescalling the gradient vectors to have a smaller norm, while still pointing to the same direction in space. This has been shown to help with training convergence in regions where the loss landscape has abrupt changes. Essentlially it consits in defining a threshold ρ and redefining $\nabla\mathcal{L}$ if $\|\nabla\mathcal{L}\| > \rho$ to

$$\nabla\mathcal{L} \leftarrow \frac{\rho\nabla\mathcal{L}}{\|\nabla\mathcal{L}\|}. \tag{5.13}$$

5.5.4 Parallelisation

Good statistics of the quantities obtained via Monte Carlo simulations are heavily dependent on how much data we are able to generate. Assuming that the simulated data represent independent and identically distributed sub-samples of the quantity we are trying to infer, the law of large numbers ensures that having a larger number of points will result in a better reconstruction of the true probability distribution.

One reliable way to obtain more data for Monte Carlo simulations in a similar amount of time is to collect samples from parallel random walkers. This means that simulations are run in parallel and that their individual sampled values can be combined to compose a larger collection of the final data. One important assumption for this parallelisation process is that random walkers are unaware of each other, so the samples do not exhibit undesired correlations, compromising the statistical significance of the computations. This requirement can be satisfied by passing unique random seeds for the different threads of the parallel programme.

In particular, for our Python parallel sampler implementation, we used the Joblib [40] library. The motivation for using Joblib instead of Python's native multiprocessing module is that the latter displayed problematic behaviour when sharing JAX objects among processes. Joblib's `parallel` function achieves embarrassingly simple parallelisation by starting separate Python workers that execute tasks on different CPUs.

5.6 Quantifying Uncertainties

There are multiple ways to analyse the statistical uncertainty of the collected data. Most standard methods rely on resampling techniques, which artificially generate more data by creating varied samples from the limited data set. For example, by shuffling data and resampling it, one can achieve a finer statistical estimation of the measured quantities due to the larger amount of data, hopefully without biasing it too much.

Conventional implementations include bootstrapping and cross-validation, better explained in [33]. VMC calculations deal with a very large amount of data, making these approaches computationally costly. More importantly, perhaps, these methods also do not explicitly take into account the correlation between data points, an important feature when analysing how the random walks evolve in configuration space. Of course, at a stationary limit, sample averages should be time independent, but in practice we do not have this guarantee. Therefore, specifically for VMC calculations, the blocking method is often used.

Quantities measured in a Monte Carlo sampling are the results of a set of sequential experiments, $\{\alpha\}$. The sampled values, such as the mean (μ_α) and variance (σ_α^2) of these experiments,

can therefore be studied as a time series. Here, these quantities are given by

$$\mu_\alpha = \frac{1}{N}\sum_{k=i}^{N} x_{\alpha,k}, \qquad \sigma_\alpha^2 = \frac{1}{N}\sum_{k=1}^{N}(x_{\alpha,k} - \mu_\alpha)^2. \tag{5.14}$$

The entire data set can be partitioned into m experiments blocks, with its m-th block having an average μ_m with a sample mean variance σ_m^2; both can be written as

$$\mu_m = \frac{1}{mN}\sum_{\alpha,k} x_{\alpha,k}, \qquad \sigma_m^2 = \frac{\sigma^2}{N}[1 + 2\sum_{d=1}^{N-1} k_d], \tag{5.15}$$

where σ^2 is the variance of all the data points,

$$\sigma^2 = \frac{1}{mN}\sum_{\alpha k}(x_{\alpha,k} - \mu_m)^2,$$

and k_d is the autocorrelation function [75] between experiments separated by a distance of d in the time series. The blocking algorithm consists of rearranging the total data set into blocks and analysing how σ_m^2 changes with different block sizes. When the number of blocks increases, this variance of the average of the blocks reaches a plateau that gives information about the correlation of the experiments. The aim is to find a distance $d = |k - l|$ between sequential experiments $x_{\alpha,k}$ and $x_{\alpha,l}$ such that experiments in the time series separated by a distance $d' > d$ can be considered uncorrelated. The standard deviation provided by the blocking algorithm will then be the asymptotic value that stabilises for this distance d. This is clear from Eq. 5.15, as $\sigma_m^2 \to \sigma^2/N$ if k_d tends to 0.

In our implementation of the blocking algorithm for the estimation of the standard deviation of the energy, we used the systematised scheme of [41]. This method automates the ordinary visual analysis of the time auto-correlation function devised by [23].

5.6.1 Combining Errors

We have discussed the use of parallelisation techniques in Python to produce more samples in roughly the same time frame via parallel computing and multiple CPUs. Here, we give a more detailed explanation of how we aggregated the errors from each independent experiment.

The empirical expected values and their variances are also random variables, and to appropriately combine mean values and standard errors, we used a meta-analysis study. More specifically, we use inverse variance weighting (IVW) [8] to aggregate the values of random variables so that their average is weighted by the inverse of their variance.

After running n independent walkers, each will give us an empirical expectation value of the energy E_n, with its associated blocking error $\mathrm{SE}(E_n)$. Then, we use IVW to combine these values into a single overall estimate of energy $\bar{E}$ and its associated error $\mathrm{SE}(\bar{E})$ as follows:

$$\bar{E} = \frac{\sum_{i=1}^{n} w_i E_i}{\sum_{i=1}^{n} w_i},$$

with the weights determined by the inverse variance of each estimate $w_i = \mathrm{SE}(E_i)^{-2}$. Then, the standard error of the mean estimate is given by

$$\mathrm{SE}(\bar{E}) = \sqrt{\frac{1}{\sum_{i=1}^{n} w_i}}.$$

5.7 Kronecker-Factored Approximate Curvature

In practical applications, particularly with the emergence of deeper and broader networks in deep learning, calculating the complete Fisher information matrix or its quantum counterpart, the Fubini study metric, becomes impractical. Not only is it computationally demanding to invert the matrix F_{ij} as required by the update rule, but storing the matrix elements also becomes memory intensive. Indeed, in a naive approach, one must recompute and store its values for each training iteration.

If we consider a rectangular neural network structure, in which all L layers have the same height H, and further assuming that there are no bias vectors, the FIM should be a matrix of size $(L \times H^2) \times (L \times H^2)$. Kronecker-factored approximate curvature (KFAC), devised by Martens and Grosse in 2015 [61] is a technique that aims to simplify this structure in two ways. In this brief explanation, we do not provide a derivation of the expressions, and for a detailed proof, we refer to the original publication.

Using the notation of [20], we start by approximating the FIM by a block-diagonal $\breve{F}$,

$$\breve{F}(\boldsymbol{\theta})_{W_{ij}^{(l)} W_{i'j'}^{(l')}} \equiv \delta_{ll'} F(\boldsymbol{\theta})_{W_{ij}^{(l)} W_{i'j'}^{(l')}}, \tag{5.16}$$

where $W_{ij}^{(l)}$ represents a weight matrix of given layer l, and $\delta_{ll'}$ a Kronecker delta. This means that we assume parameters from different layers of the network to be independent in terms of the curvature of the parameter space. This turns the $(L \times H^2) \times (L \times H^2)$ matrix into $L^2 H^2$ blocks of $(H \times H)$ matrices that can be inverted independently.

The second simplification of the KFAC method consists of writing the blocks of the approximated $\breve{F}$ as Kroneker products of statistical averages of the forward activations $\mathbf{a}^{(i)}$ and backward sensitivities $\delta^{(i)}$ of the FFN, discussed in Sec. 4.3.2. Then, it can be shown that

$$\breve{F}(\boldsymbol{\theta})^{(l)} \approx \mathbb{E}_{p_\theta}\left[\mathbf{a}^{(l)} \mathbf{a}^{(l-1)^\top}\right] \otimes \mathbb{E}_{p_\theta}\left[\delta^{(l)} \delta^{(l-1)^\top}\right]. \tag{5.17}$$

However, throughout this work, we do not implement this last simplification.

5.7.1 Trust Regions and Tikhonov Regularisation

When solving second-order optimisation problems practically, the constrained minimisation problems of Eq. 4.5 and Eq. 4.7, discussed in Sec. 4.1.1 are often treated as analogous unconstrained problems with associated trust regions and damping parameters, in what is called the Tikhonov regularisation method. More specifically, consider the second-order approximation model

$$M(\delta) = E(\boldsymbol{\theta}_0) + \nabla E(\boldsymbol{\theta}_0)^\top \delta + \frac{1}{2} \delta^\top G \delta,$$

where $\delta = \boldsymbol{\theta} - \boldsymbol{\theta}_0$, and G is a matrix describing the geometry of the trust region. For example, in Newton's method, it is the Hessian of the objective function, and in the natural gradient method, it is the Fisher information matrix. Following Tikhonov regularisation, the correct update rule is given by the unconstrained minimisation,

$$\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t = \arg\min_{\delta \in R_t} \left(M_t + \frac{\lambda_t}{2} \delta^\top G(\theta_t) \delta \right),$$

where, λ is the damping term and $R_t = \{\delta : \|\delta\| \leq r_t\}$, for some radius r_t, is a trust region for which the second-order approximation is valid. Adding this damping term has the effect of using a regularised version of the FIM, equivalent to $F - \lambda I$. This is important to stabilise the pseudo-inverse calculation of F via singular value decomposition. In our work, we have used a diagonal shift of around $\lambda = 10^{-4}$.

In reality, choosing the correct radius of the trust region at every step is impractical, and learning rate decay schedules have been suggested to ameliorate this difficulty [27]. For our implementations, we proceeded by choosing the learning rate

$$\alpha_t = \min\left(\alpha_0, \sqrt{\frac{\alpha_1}{\delta_\theta^\top G \delta_\theta}}\right) = \min\left(\alpha_0, \sqrt{\frac{\alpha_1}{(F^{-1}\nabla E(\theta))^\top \nabla E(\theta)}}\right),$$

with α_0 and α_1 hyperparameters that require tuning. This is an idea suggested in [58], which requires that the scheduler choice could depend on the curvature of the trust region. This choice ensures that the learning rate is smaller when the curvature of the parameters under the metric defined by G is large. In addition to that, we also added a momentum term of $\gamma = 0.9$ in our implementations. Then, the update rule when implementing specifically our version of SR can be written

$$\begin{aligned} \boldsymbol{v}_t &= \gamma \boldsymbol{v}_{t-1} + \breve{F}^{-1}\nabla E(\theta), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \alpha_t \boldsymbol{v}_t, \end{aligned}$$

where we denote $\breve{F}$ as the block-diagonal approximation of the FIM.

5.8 Hyperparameter Search

Optimising hyperparameters is a crucial part of training machine learning models, as it can greatly influence the quality of the model. Performing extensive brute-force searches over the hyperparameter space is a task that is intractable on its own, as the number of possible configurations scales exponentially. To find the best configuration for a certain model, several techniques can be employed, the simplest being random selection over the parameter space.

Several of our initial investigations to find a good choice of model and parameters begin with a trial-and-error process, as this carries a lower overhead in initial iterative phases of a project. However, for more detailed and subsequent investigations, when the aim is to find the best model, we utilised the Weights & Biases (Wandb) software [7]. This is a well-known tool for experiment tracking and hyperparameter tuning, supporting the automation of hyperparameter searches through Bayesian optimisation and randomised search. Parameter searches using Wandb will hereafter be called sweeps, following the software convention.

We opted to proceed in a slightly different manner depending on the system being investigated. For the fully polarised one-dimensional fermionic system, the runs are executed reasonably quickly, as we deal with a one-dimensional problem and we only gather results for up to 6 fermions. In this scenario, when generating sweeps over the entire set of parameters, we select several hyperparameters such as batch size, neural network architecture, learning rate, whether to use a pretraining stage, the type of optimiser to use, whether to use regularisation, and others.

In one-dimensional runs, we conducted three major sweeps, one for each ansatz type: Deep Set feed-forward net (DSFFN), standard variational Monte Carlo (VMC), and restricted Boltzmann machines (RBM). Here, the sweeps include uniform random selection over the number of particles and interaction strength. This affects the energy to which we are converging, but the software enables filtering runs afterward, extracting averages for specific combinations of number of particles and interaction strength. The large number of sweeps ensures a fair representation of the hyperparameters in a uniform search. For instance, we have roughly the same number of runs using stochastic reconfiguration and Adam as optimisers for two, four, and six particles alike.

When dealing with two-dimensional quantum dots, the additional dimension and the increased number of particles make the computations more expensive, so we adopted a different

approach. Here, we explicitly separate the investigation into exploration and exploitation phases. The exploration phase involves performing sweeps over hyperparameters for several short runs (with a small number of epochs and batch sizes for the Monte Carlo averages). To do this efficiently, we used a random Bayesian optimisation search instead of a uniform random search.

Bayesian optimisation is an efficient strategy for hyperparameter tuning, especially when the evaluation of configurations is expensive, or there are simply too many configurations to try [19], which is often the case with deep learning models. Bayesian optimisation works by building a probabilistic model (typically a Gaussian process) of the objective function and using information of previous parameter choices to select the most promising hyperparameters to evaluate next.

After these short sweeps, we select the hyperparameters from which we want to collect statistics. This stage requires a modification of how we separate the sweeps. This is because, in Bayesian optimisation, we must specify the metric to be minimised beforehand. If we include configurations with varying numbers of particles in a single Bayesian sweep and use the expected energy value as the objective function, systems with fewer particles will be favored because they have lower energy. One option to circumvent this could be to guide the Bayesian search to minimise the variance of the energy instead. Nevertheless, even though variance optimisation has an absolute target (0 at the ground state), larger systems are intrinsically harder to train because of multiparticle interactions and correlations. Consequently, simpler systems would be selected more frequently.

For two-dimensional systems, we opted to conduct separate Bayesian sweeps for each number of particles. Furthermore, we only perform sweeps for a harmonic trap frequency of $\omega = 1$, in the hope that the best model will be able to generalise well for different frequencies in subsequent simulations.

Part III

Results and Discussion

Chapter 6

One-dimensional trapped spinless fermions

We now investigate a series of results for the one-dimensional trapped spinless fermions with Gaussian interaction. Here, all fermions are constrained to have spin in the same polarisation. A better description of the physics of the problem is given in 2.6.

6.1 Initial Comparisons

We begin by demonstrating in Fig. 7.1 that all three main trial wavefunctions discussed in Sec. 5.1 can very quickly converge to yield the ground-state of the non-interactive system. This serves as a valuable reference point for the code. These results were obtained without any hyperparameter search or addition of a correlation factor, showing that it is a fairly easy result to obtain. In these experiments, we used the Adam optimiser with learning rate $\alpha = 0.01 * c$ for all models, where c is a scaling factor used both for the Metropolis step and for the learning rates, and equates to $1/\sqrt{N \cdot d}$. All results hereafter will use a Metropolis step of $0.1/\sqrt{N \cdot d}$ as this was shown to be optimal.

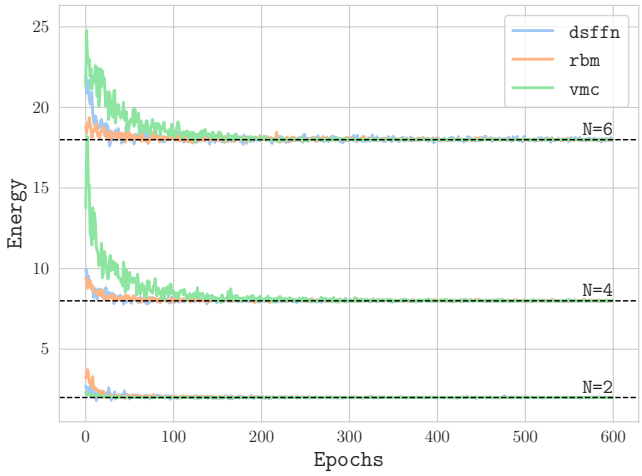


Figure 6.1: Energy convergence curve for the non-interactive one-dimensional fully polarised fermionic system. In black dotted line, the analytical ground-state energies are marked. The models are Deep Set feed-forward network (DSFFN), restricted Boltzmann machine (RBM) and a standard variational Monte Carlo (VMC). Here we used Adam with learning rate of 0.01 and Metropolis sampling.

To check if reaching the ground-state energy results in an accurate wavefunction representation, one can generate a plot of the wavefunction. This is especially possible in the one-dimensional two-particle scenario, where the positions of each particle can be used as axes and

the function value can be represented by a colorbar, as in Fig. 6.2. The values were obtained by uniformly sampling the positions for the particles and evaluating the output of a trained model, together with its sign. Fig. 6.2 further serves to validate that despite working with $\ln|\psi|$ throughout the entire sampling and training process, we can recover the sign of the wavefunction. The profile shown is a classic fermionic representation, with a node region at $X_1 = X_2$ around which the wavefunction values are mirrored, except for a flip in sign. This is a consequence of the Pauli exclusion principle and the fact that we enforce the fermions to be polarised with the same spin.

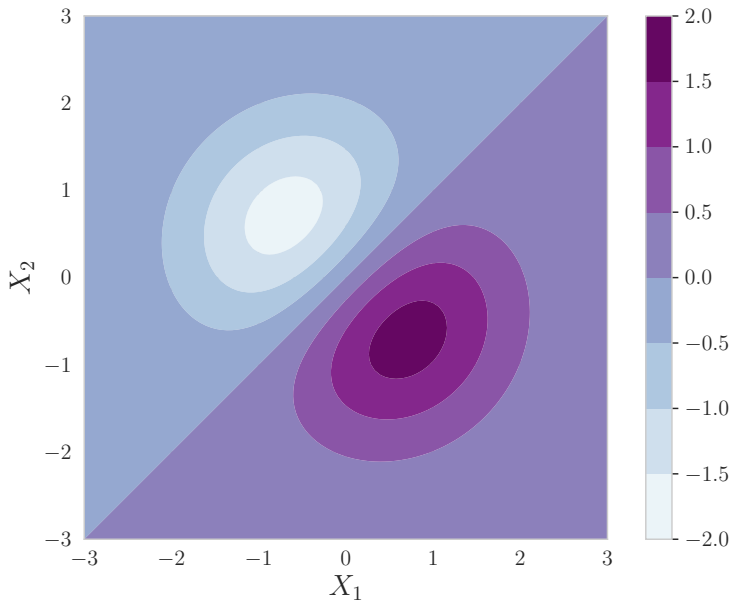


Figure 6.2: Two particle wavefunction for a one-dimensional fermionic ground-state. A clear fermionic nodal structure at $X_1 = X_2$ can be observed.

We can compare the quality of our models and techniques without extensive parameter investigation for a small but interacting system of two particles ($N = 2$). At this stage, the results shown are merely investigations and are not fine-tuned in any way. In Fig. 6.3, we compare our models with four values of interaction strength. We begin with an attractive regime of $V_0 = -20$ and increase until we obtain a repulsive interaction $V_0 = 20$. In all interactive polarised fermionic systems, we limit ourselves to dealing with an interaction range of $\sigma_0 = 0.5$. The energy values displayed are a rolling average over ten epochs for better visuals. If, for example, 600 epochs were used, only 60 points make up the plot.

Furthermore, Fig. 6.3 contains Hartree-Fock and CI energies, obtained with the code available from [44]. It should be mentioned that the basis for the CI calculations was by no means a large one, with only 20 harmonic oscillator modes, and the Hartree-Fock calculations were obtained via N integro-differential equations, where we iterate over the entire spatial grid computing the density and updating the Hamiltonian matrix at each step. Then, approximate eigenvalues are obtained via the Rayleigh-Ritz method.

All results shown in Fig. 6.3 used a block-diagonal approximation of the stochastic reconfiguration (SR) update rule with learning rate of 10^{-3}. For all uses of this SR approximation, we also added a diagonal shift of $\lambda = 10^{-4}$ and a trust region for the update, described in Sec. 5.7. The convergence curves have large deviations because small batch sizes were used. This means that only 500 MC proposals were sampled to collect the averages at each step. Consequently, the energy estimations fluctuate, and the gradient update, which is based on $\nabla\langle E_L\rangle$, also loses stability.

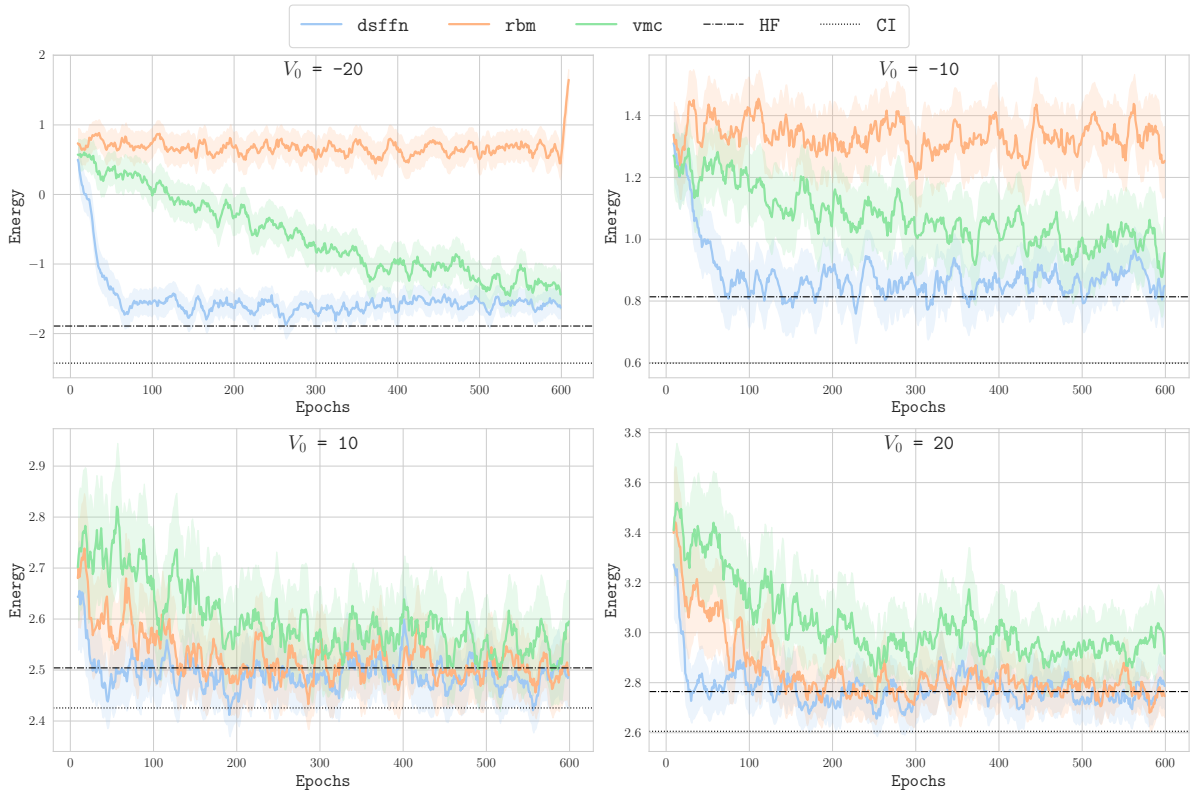


Figure 6.3: Energy convergence curve for the two particle ($N = 2$) interactive problem of one-dimensional fully polarised fermions. The shaded area marks a 95% confidence interval. In black dotted line, one can see Hartree-Fock and CI reference energies. None of the models used were fine-tuned, and the optimiser sheme was the stochastic reconfiguration method with a block-diagonal approximation.

Still with regard to Fig. 6.3, the Deep Set model seems to outperform the other two, especially in the attractive regime. Although this could be attributed to a larger expressive power or more careful enforcement of antisymmetry, special mention must be credited to the pretraining stage. As detailed in Sec. 5.5.1, for feedforward networks, convergence for dimensions greater than one was only achieved by pre-training the ansatz in a supervised manner. Specifically, we fit the log probability of the ansatz to that of a multivariate Gaussian. It is known that in the attractive regime, the one-dimensional spinless fermions exhibit a density profile similar to non-interacting bosons [91], resulting in a density profile that closely resembles a Gaussian. This, together with a larger and expressive power and flexibility, can explain why the DSFFN performs better than the other models in the attractive regime.

6.1.1 Correlation Factor

A last analysis without extensive parameter experimentation can be seen in Fig. 6.4, where we select the deep set FFN ansatz of figure Fig. 6.3 for the same physical systems and compare convergence curves with and without the inclusion of the Jastrow factor. We see that, without the Jastrow factor, the model is capable of oscillating around the energy levels given by the Hartree-Fock calculations. Indeed, this variational ansatz uses a Gaussian envelope only with the determinant of Hermite polynomials, then approximating a single Slater determinant. Only with the introduction of the Jastrow factor can the energy values achieved go significantly lower than the HF approximation.

This observation is interesting. In principle, the universal approximation theorem guarantees that the neural network part of the ansatz can approximate any complex correlation factor which

is not captured by the HF approximation. Although this might indeed be possible, the explicit addition of a correlation factor to the ansatz was what significantly enabled energies lower than HF. However, the statistics in Fig. 6.4 are lacking, and we must examine the values and errors of larger Monte Carlo samples to be able to make this affirmation.

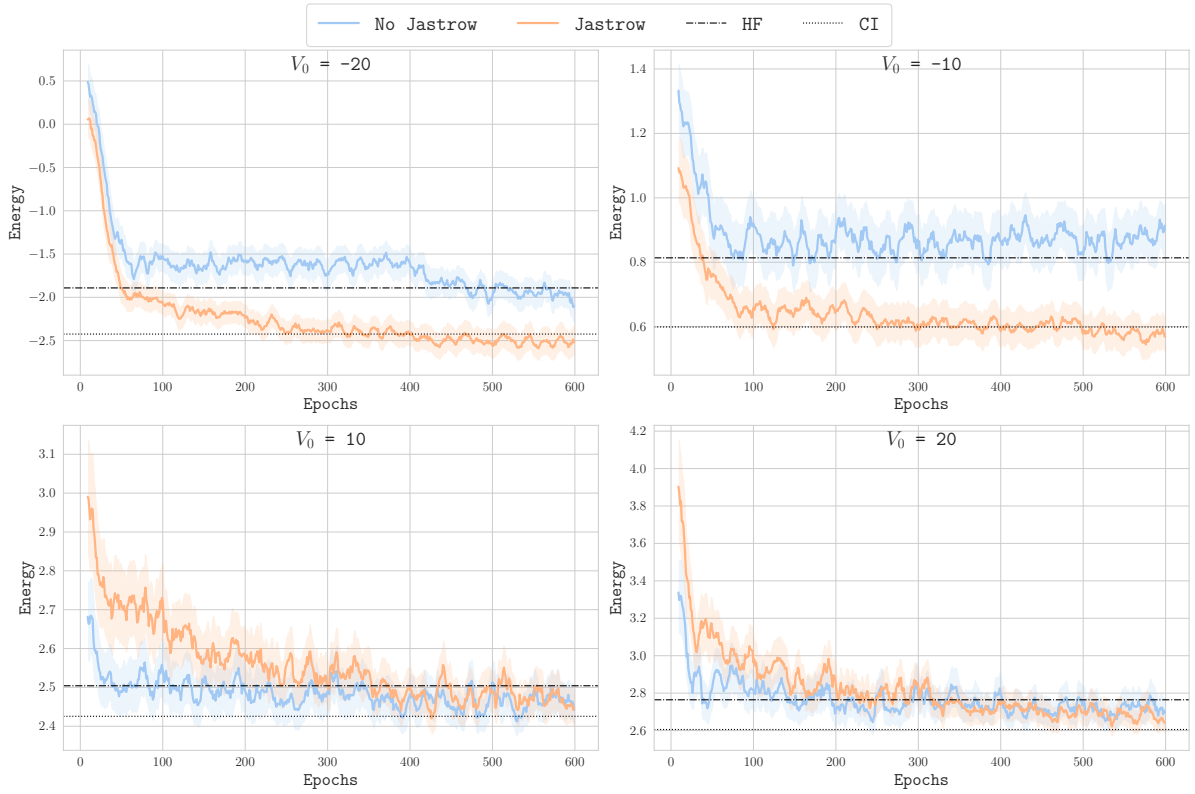


Figure 6.4: Comparison between energy convergence curves for two particle ($N = 2$) for Deep Set feed-forward network with and without Jastrow factor. We an approximate stochastic reconfiguration optimiser, and the plot is displayer with a running average of 10 epochs. The shaded are represents a 95% confidence interval.

6.2 Hyperparameter Search

We then proceed to more thorough investigations via hyperparameter sweep searches. For each of our ansatz, we performed a sweep of approximately 300 configurations. This means that 300 sets of variational training were done, each with a random combination of options such as batch size, learning rate (here called "Eta"), maximum training epochs, network architecture, and more. Figure 6.5 displays the sweep for the Deep Set ansatz, which involved 351 hyperparameter configurations. Analogous searches for VMC and RBM ansätze can be seen in the appendix F.

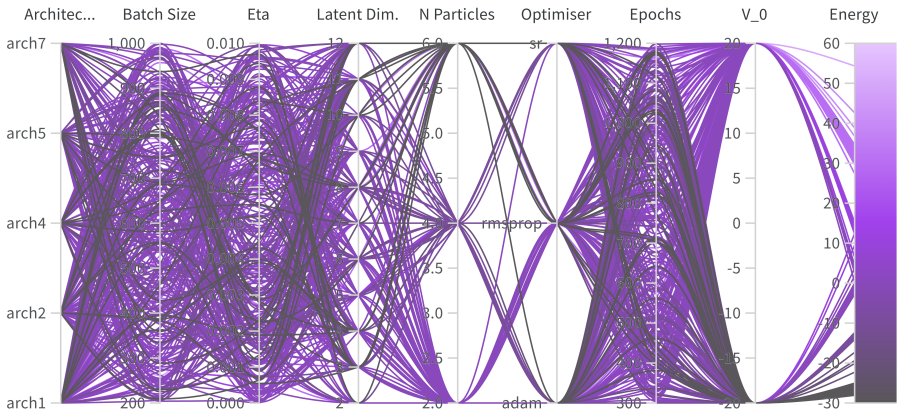


Figure 6.5: Sweep over 351 hyperparameters for the standard DSFNN ansatz, from which the results shall be aggregated from. Table 6.1 displays the different architectures experimented, here simply marked as "arch".

Table 6.1: Deep Set architectures Overview for sweep, for the sweep performed in Fig. 6.5. The values missing in the architecture order were discarded for frequent non-convergence behaviour. The latend dimension L_d is chosen independently.

Architecture Nr.	**Activations**		**Layer Sizes**		
	ϕ	ρ	ϕ	Pooling	ρ
1	GELU $\times$ 4	GELU, linear	7, 5, 3, L_d	Avg	L_d, 3, 1
2	GELU $\times$ 5	GELU $\times$ 3, linear	10, 7, 5, 3, L_d	Avg	L_d, 6, 4, 2, 1
4	GELU $\times$ 5	GELU $\times$ 2, linear	9, 7, 5, 3, L_d	Avg	L_d, 5, 3, 1
5	GELU $\times$ 5	GELU $\times$ 2, linear	14, 9, 7, 5, L_d	Avg	L_d, 5, 3, 1
7	GELU $\times$ 5	GELU $\times$ 2, linear	12, 9, 7, 5, L_d	Avg	L_d, 6, 4, 1

6.2.1 Optimisers

From the sweep of Fig. 6.5, several results can be aggregated and averages can be taken for certain configurations. For instance, Fig. 6.6 isolates the case where the interaction strength is V_0 and, for a different number of particles, extracts the average energy values for the different choices of optimisers. The figure suggests that among the optimisers evaluated, the stochastic reconfiguration variant consistently outperformed the others, significantly contributing to a lower average value across more than 300 different parameter settings and different number of particles. Although we show only attractive interactions of $V_0 = -20$, the same was observed for repulsive interactions. This motivated the choice of the SR optimiser for all the following results. AdaGrad was consistently the worst optimiser tested, at least for the parameters used.

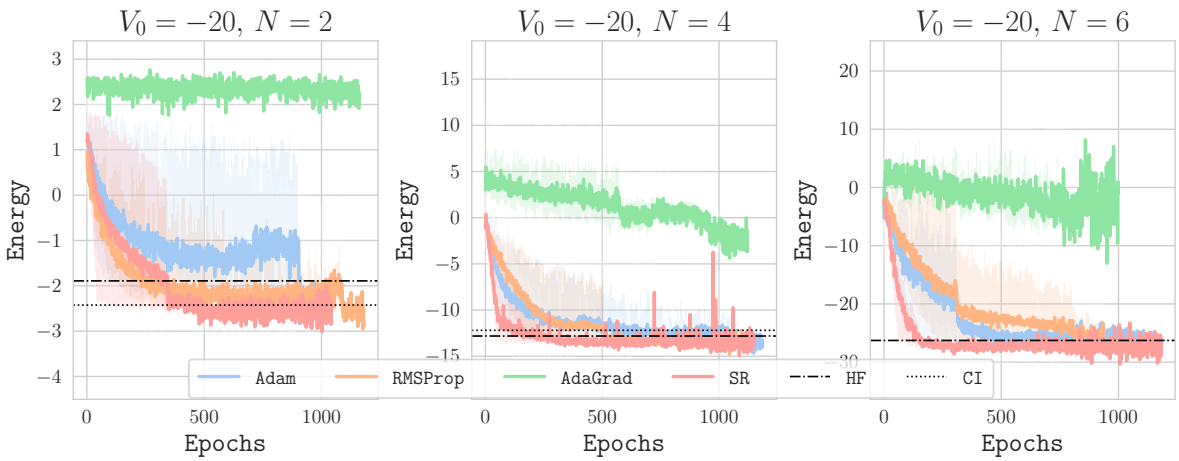


Figure 6.6: Training energy averages and standard deviations for the aggregated values of the hyperparameter search of Fig. 6.6. Results are displayed for two, four and six particles with interaction $V_0 = -20$. Here, the shaded are is not the confidence interval, but the minimum and maximum observed in the parameter search.

6.2.2 Importance Sampling

Another choice that can be investigated from 6.5 is how the expected energy value is affected by the use of importance sampling, discussed in Sec. 3.4.1. Figure 6.7 compares this for different number of particles and the extreme or the interaction strengths, both attractive and repulsive.

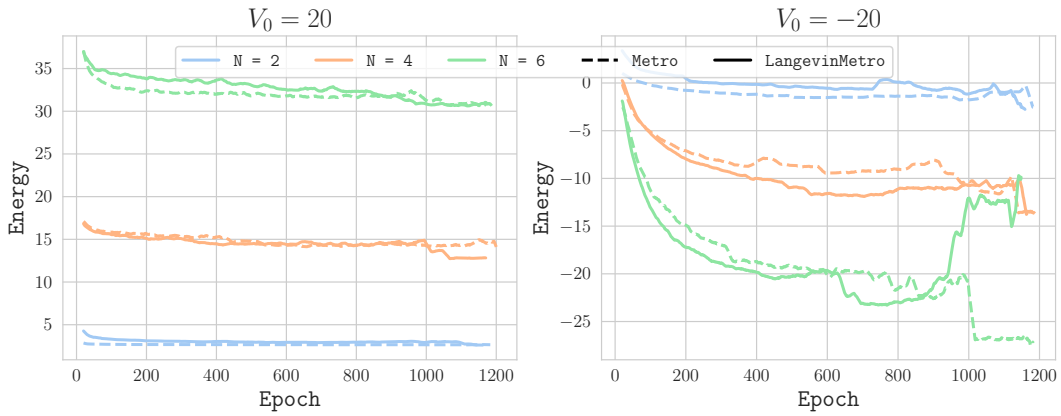


Figure 6.7: Comparison of the averaged values over all 321 runs of the sweep shown in figure Fig. 6.5 aggregated by number of partciles, interaction stren, rolling window of 20. Errorbars are not included. Here, "LagevinMetro" depicts importance sampling.

There is a notable gap regarding whether the choice of importance sampling will, on average, produce better results, as illustrated in Fig. 6.7. This gap seems to depend on the character of the interaction, where the attractive regime favoured importance sampling over the regular Metropolis algorithm. Although it is challenging to speculate on the exact cause, one potential explanation is the width of the distribution we aim to approximate. In the attractive regime, the distribution tends to be more localised, making standard sampling techniques less efficient. Consequently, for broader distributions, various sampling methods might perform equally well. However, when greater efficiency is needed, importance sampling proved to be more effective.

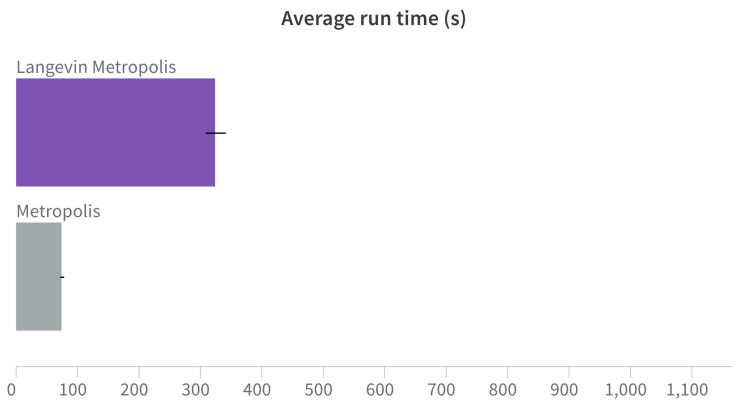


Figure 6.8: Average runtime comparison of the Langevin Metropolis (importance sampling) and the regular Metropolis Makov chain Monte Carlo over 321 randomised runs. The line in black depicts the standard error of the mean.

Despite the importance sampling technique displaying some advantage over a regular Metropolis Sampling, time complexity must be taken into account. On that note, Fig. 6.8 shows the average run-time for all configurations displayed in Fig. 6.5. Not all averages include the same batch size or number of training epochs; however, these seem to be equally distributed between sampler options. Figure 6.5 motivates us not to use importance sampling as a method throughout the remainder of this study. There is approximately a threefold discrepancy between average runtimes, and the difference in minimised energy value did not seem to justify its use. The small difference in energy is further supported by [69]. Although importance sampling has been shown to be necessary for the use of diffusion Monte Carlo [42], its benefits may not be as significant for VMC.

What explains at least part of the discrepancy in computational time is the need to calculate the drift force, $F = 2\nabla \ln |\psi|$, and the ratio of Green's functions, as displayed in Eq. 3.24 and Eq. 3.25. Although this alone should not have an effect as large as observed, one specific point must be mentioned. For implementation reasons, and to make the Langevin-Metropolis method general to all ansätze, the calculation of the drift force required reshape manipulation of the input position vector. We tried to mitigate this, with no success. As mentioned in 5.4.6, JAX arrays are supposed to be static, so we strongly believe that a big part of the difference in computational time is due to JAX not being able to fully pre-compile this calculation.

6.3 Energy Components

Now, instead of displaying the average results in a series of configurations, we selected parameters based on the ranges that gave good results from the hyperparameter search of Sec. 6.2. For the results that follow, some common choices were made. All results displayed hereafter were performed with trial functions that included a Jastrow factor, and the optimiser of choice was always stochastic reconfiguration. These calculations involve longer training processes and more extensive sampling.

More specifically, Fig. 6.9 shows the distribution of the energy components as a function of the interaction strength for all ansätze and different numbers of particles. The different trial functions agreed reasonably well in terms of energy values, and a better comparison can be seen in Table 6.2.

The energy curves in Fig. 6.9 are expected, showing that the energy scale varies with the particle count, while the ratios between the energy components mirror the findings of [44], from which we extract some analysis.

First, the total energy E increases when going from an attractive to a repulsive regime. The potential energy increases with the strength of the interaction, but the apparent plateau, together with a decrease in kinetic energy, indicates fermion localisation. This point will be discussed in the analysis of the density profiles.

When the interaction is extremely attractive, the particles are concentrated in the centre of the trap. In this scenario, the absolute magnitude of the interaction surpasses that of the kinetic energy, and there is almost a cancelation of the trap potential. As also achieved by [44], in some instances we observe $\langle K \rangle \leq \langle V_{int} \rangle$, which is a sign of Wigner crystallisation [95].

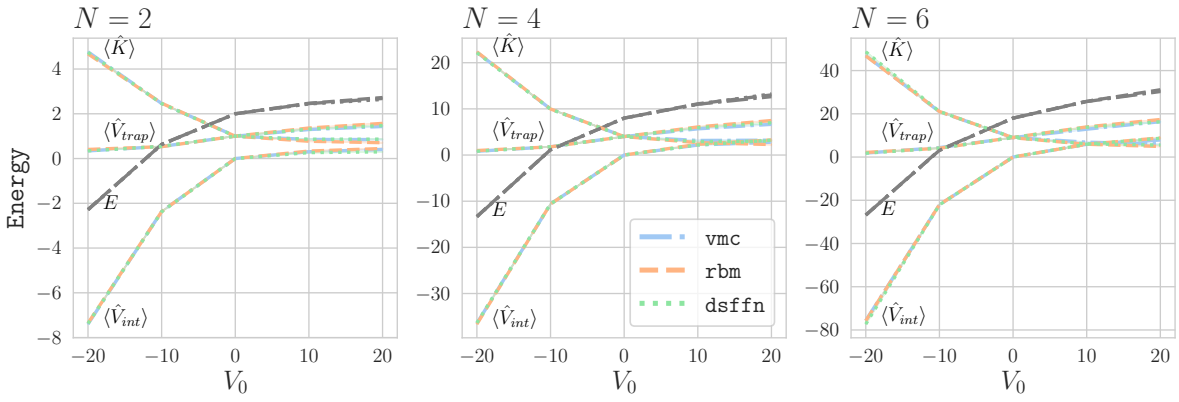


Figure 6.9: Energy components as a function of the interaction strength, V_0, for different number of particles and trial wavefunctions. All wavefunctions were multiplied by a Jastrow factor. The values were obtained with after 10^{24} Monte Carlo samples, SR optimiser and 1500 epochs of training.

In Table 6.2 we also show the ratio between kinetic energy and the sum of potential energies for a four-particle system. As dictated by the virial theorem, in the non-interacting case, $\langle E \rangle = 2\langle \hat{K} \rangle = 2\langle \hat{V}_{trap} \rangle$. This is verified with the fraction of the kinetic and potential energies resulting in 1. Again, we note an acceptable yet not perfect agreement between the different models.

Ansatz	V_0	E	$\langle \hat{K} \rangle$	$\langle \hat{V}_{trap} \rangle$	$\langle \hat{V}_{int} \rangle$	$\frac{\langle \hat{K} \rangle}{\langle \hat{V}_{trap} \rangle + \langle \hat{V}_{int} \rangle}$
DSFFN + SJ	-20	-13.372(3)	22.16(1)	0.8303(5)	-36.364(9)	-0.624
	-10	1.1421(9)	9.941(5)	1.797(1)	-10.595(4)	-1.130
	0	8.00028(6)	4.000(2)	4.000(2)	0(0)	1.000
	10	11.009(1)	2.906(2)	5.921(2)	2.182(2)	0.357
	20	13.244(8)	3.021(8)	6.998(3)	3.224(3)	0.296
RBM + SJ	-20	-13.294(3)	22.36(1)	0.9751(8)	-36.63(1)	-0.627
	-10	1.145(1)	9.983(5)	1.772(1)	-10.610(4)	-1.13
	0	8.00123(8)	3.998(2)	4.004(2)	0(0)	0.999
	10	11.101(1)	2.684(1)	6.089(3)	2.328(2)	0.319
	20	13.025(2)	2.271(1)	7.476(3)	3.278(3)	0.211
VMC + SJ	-20	-13.360(3)	22.261(9)	0.8186(4)	-36.439(9)	-0.625
	-10	1.1452(9)	9.978(5)	1.781(1)	-10.614(4)	-1.130
	0	8.000028(4)	3.998(2)	4.002(2)	0(0)	0.999
	10	11.0009(7)	3.161(2)	5.674(2)	2.166(2)	0.403
	20	12.650(1)	3.222(2)	6.647(2)	2.781(3)	0.321

Table 6.2: A more detailed display of the energy components of Fig. 6.9, for four particles.

6.4 One-Body Densities

In terms of one-body density profile, Fig. 6.10 shows in isolation the two-particle case for all ansätze and different interactions, and Fig. 6.11 shows the other cases, up to six particles. From these figures, it becomes clear that, for both an attractive regime and a larger number of particles, getting the models to agree becomes challenging. However, even when the methods disagree, some qualitative points can be addressed. Even for a different number of particles, once the interaction strength is set, the width of the distribution remains the same. Furthermore, the peak of the distributions becomes taller with an increase in the number of particles as $n(\mathbf{x})$ must integrate to N, but with the system constrained by the trap.

Importantly, as analysed in terms of energy components, the attractive regime demonstrates a peak at the centre of the trap, in a process corresponding to the bosonization of polarised fermions [91]. On the other extreme, the repulsive interaction results in clear peaks or fringes. The number of peaks corresponds to the number of particles in the system and should, in principle, be symmetric around the origin.

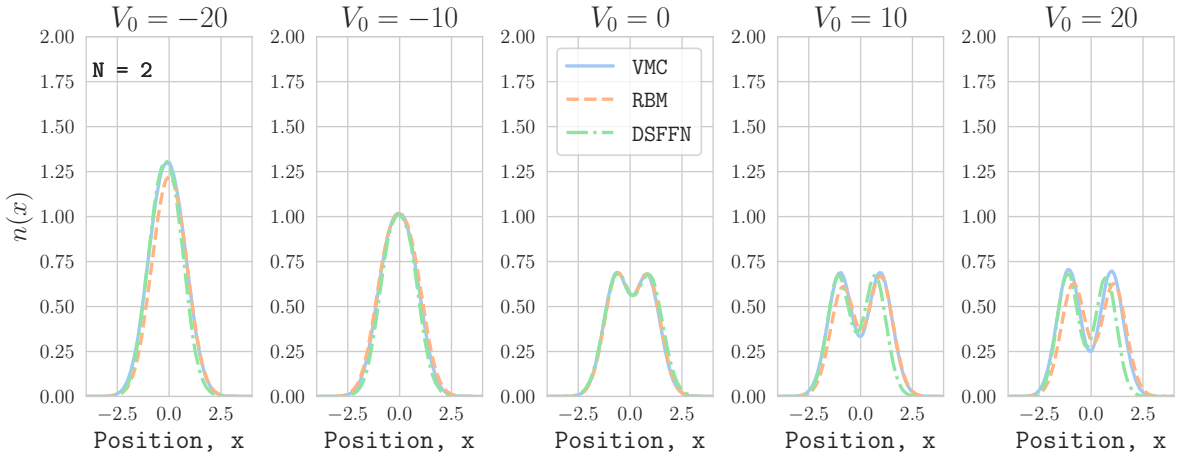


Figure 6.10: Density profiles $n(x)$ for two particles at different interaction strengths and for different trial functions, all accompanied by a Jastrow factor. The profile was based on 2^{24} samples, with average energies displayed in Tab. 6.3

As shown in Fig. 6.11, the peaks become more pronounced with a stronger interaction, although not all models follow the symmetry constraint well. The VMC function appears to perform the best in this regard. Conversely, the RBM shows significant asymmetry with stronger repulsive interactions and fails to capture the fringes for three interaction strengths in the two-particle scenario. We will explore how this relates to the energy value in Sec. 6.5, but we can already discuss this behaviour with respect to the symmetry of the ansätze.

Both mathematically and due to the trap symmetry, the probability density of the antisymmetric wavefunction should be symmetric. Although this seems to be approximately the case in Fig. 6.11 it is certainly not true for all the examples shown. One cause, apart from us not being in the true ground-state, is that the initial ansatz does not perfectly adhere to any symmetry. Due to the potential variations in weights and biases for different particle inputs, the RBM, the VMC Gaussian ansatz, or the Jastrow factor are not ideally symmetric. For instance, even when the Deep Set ensures symmetry and is combined with a determinant that is inherently antisymmetric, the Jastrow factor has the potential to negate this symmetry. We expected this behaviour to be not as dramatic for the Deep Set, but it seems like the VMC was the best in that regard.

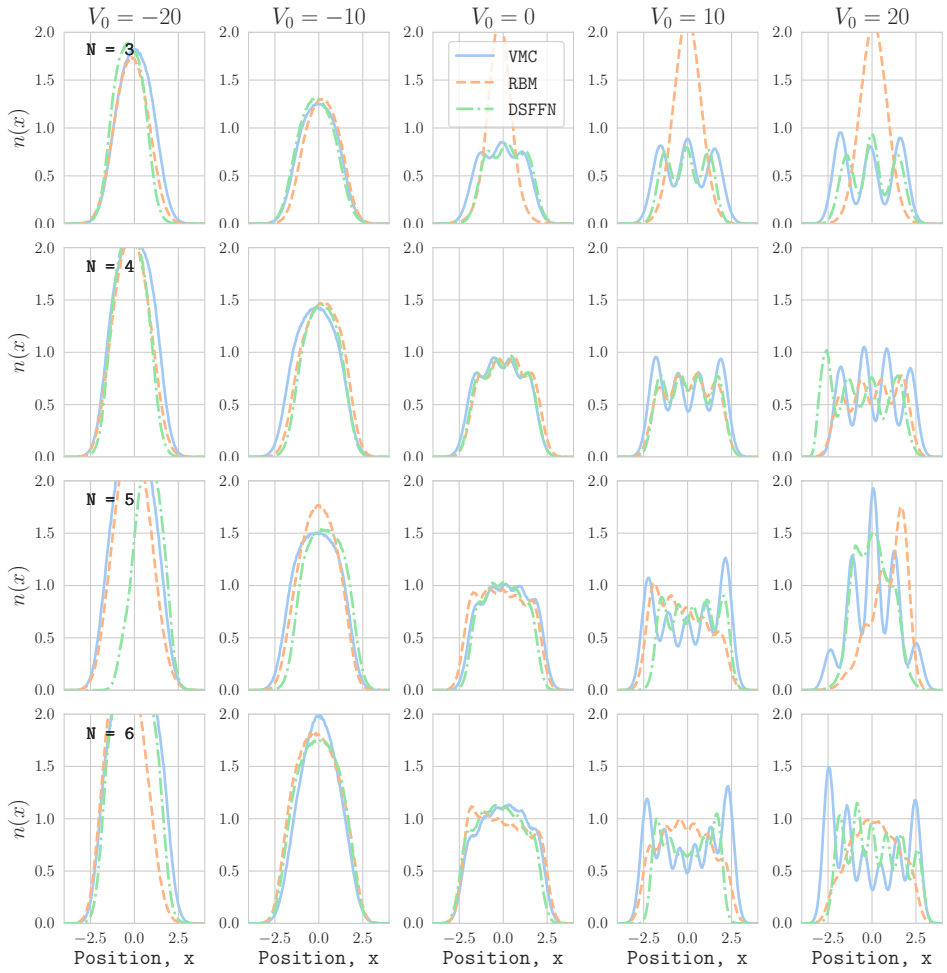


Figure 6.11: Density profiles for different ansätze, after 2^{24} MC samples. All ansatze were multiplied by a Jastrow factor, and the sampled energies can be found in Tab. 6.3

6.5 Overall Energy Comparison

Figure 6.12 shows that regardless of the model, the energy obtained is lower than the Hartree-Fock energy for every interaction V_0. While this figure shows only a two, four and six-particle case, Tab. 6.3 reveals that this is equally true for other cases. Moreover, there is a clear ease in energy minimisation for the attractive regime. Apart from that, to gauge which models yield the lowest average energy, we can look at the average energy value over all particles and interactions.

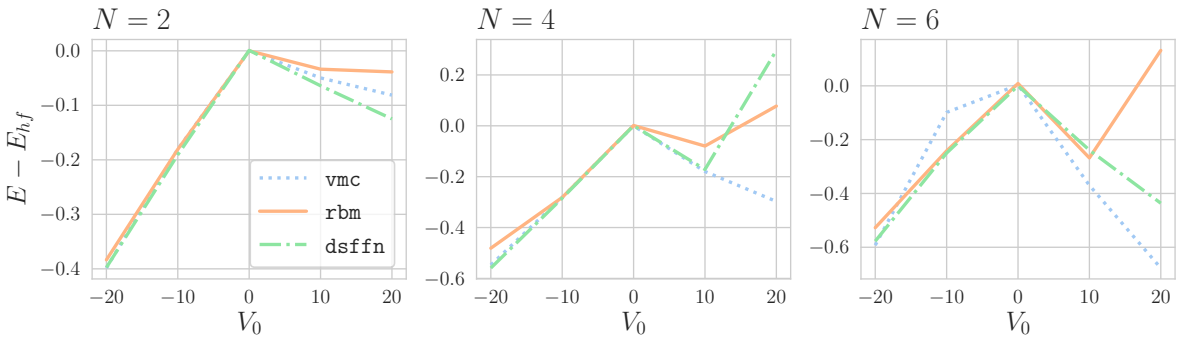


Figure 6.12: Difference between the energy sampled and the HF energy versus interaction strength V_0. Error bars are included but too small to see. For a complete overview of the energies for other number of particles, 6.3. All values are result of 10^{24} samples after 1500 epochs of training with Jastrow factor included.

We further notice that there is a clear correlation between the lowest energy values obtained in Tab. 6.3 and a density profile that is closer to what is expected in 6.11. For instance, the RBM failed to replicate the expected fringes for three particles in the repulsive interaction. Then, even though the density profile was symmetrical, resembling a Gaussian shape, we see that the energy values for the other two models were lower in that regime. Furthermore, both the RBM and DSFFN model displayed some asymmetry in the five-particle attractive regime, and that is reflected in a significantly higher energy value in comparison to the VMC.

After discussing the results for the two-dimensional quantum dots, we will offer a broader assessment of the quality of the proposed ansätze and provide possible explanations for the observed differences. For the moment, it is sufficient to highlight the final average energies presented in Tab. 6.3 indicate that the VMC model performs the best, by a reasonable margin. Behind it is the DSFFN, and last, the RBM. In comparison with the small-basis CI calculations, we observe that in certain cases we can obtain a ground state energy that is lower than theirs. Although this phenomenon is observed in three instances, we focus solely on the validity of the three-particle case with interaction $V_0 = -20$. This selection is due to the small basis CI occasionally showing energy values lower than those from HF calculations. The HF and CI values are sourced from the code provided by [44], with the HF equations being solved using integro-differential equations, and this behaviour is not fully understood.

6.6 Time Scaling Analysis

We now discuss how the presented methods scale with the number of particles. In this scenario, a theoretical analysis is a particular challenge. First, not all methods are or should be trained equally. For example, the Deep Set implementation requires a pre-training stage, as described in 5.5.1, where we regress to a Gaussian function. This can be done only once for every architecture and reused for any other problem where the number of particles is the same. For this reason, pre-training timing is not included in the following analysis.

Additionally, the number of parameters in each model is substantially different and that in should change the required number of training steps for convergence. Nevertheless, to make a fair assessment, we show the wall time for three individual training and sampling steps for each ansatz where we consistently used $2^{20} = 1048576$ samples, with 600 training epochs and a batch size of 300 samples. Although the training and batch sizes used here are smaller than those typically used to achieve optimal results, our focus is to illustrate the scaling rather than the absolute values. These averages as a function of the number of particles can be seen in Fig. 6.13, where we show measurements for both the SR and Adam optimisers.

N	V_0	DSFFN + SJ	RBM + SJ	VMC + SJ	HF [44]	CI [44]
2	-20	-2.289(1)	-2.275(2)	-2.290(1)	-1.891464	-2.4246
	-10	0.6230(5)	0.6329(6)	0.6334(6)	0.813691	0.5991
	0	2.00082(5)	2.000023(7)	2.0000000(7)	2.000000	
	10	2.4401(2)	2.4705(4)	2.4545(3)	2.504193	2.4256
	20	2.6396(3)	2.7258(7)	2.6832(5)	2.764479	2.6057
3	-20	-7.172(2)	-6.51(2)	-7.249(2)	-6.764263	-7.0593
	-10	0.7709(8)	1.014(8)	0.7645(7)	1.016808	0.7300
	0	4.50211(8)	4.589(2)	4.500016(6)	4.500000	
	10	5.9627(6)	6.020(1)	5.9533(4)	6.061344	5.8781
	20	6.760(1)	6.883(2)	6.7106(7)	6.887364	6.5153
4	20	13.244(8)	13.025(2)	12.650(1)	12.947469	12.3408
	10	11.009(1)	11.101(1)	11.0009(7)	11.180564	10.8916
	0	8.00028(6)	8.00123(8)	8.000028(4)	8.000001	
	-10	1.1421(9)	1.145(1)	1.1452(9)	1.426522	1.1113
	-20	-13.372(3)	-13.294(3)	-13.360(3)	-12.812753	-12.1935
5	-20	-19.633(4)	-19.815(5)	-20.065(3)	-19.428348	-17.3300
	-10	1.870(1)	1.912(1)	1.923(1)	2.175499	1.8551
	0	12.50157(8)	12.4975(1)	12.5004177(5)	12.5000	
	10	17.563(1)	17.645(2)	17.560(1)	17.8484	17.4648
	20	21.44(1)	21.033(3)	20.515(3)	20.9826	20.1357
6	-20	-26.855(4)	-26.805(5)	-26.870(4)	-26.2777	-
	-10	3.086(1)	3.094(2)	3.237(2)	3.335750	
	0	18.00015(4)	18.0087(2)	18.0033(1)	18.0000	
	10	25.802(3)	25.773(2)	25.671(2)	26.040932	
	20	30.557(4)	31.125(5)	30.316(5)	30.9932	
Average		4.8237332	4.87986612	4.7355	4.9928	

Table 6.3: Collection of results for different number of particles, interaction regimes and trial functions. All runs were performed with SR and 2^{24} samples, with 1500 epochs of training. All ansätze contained a Jastrow factor. The missing values were not included in the study of reference.

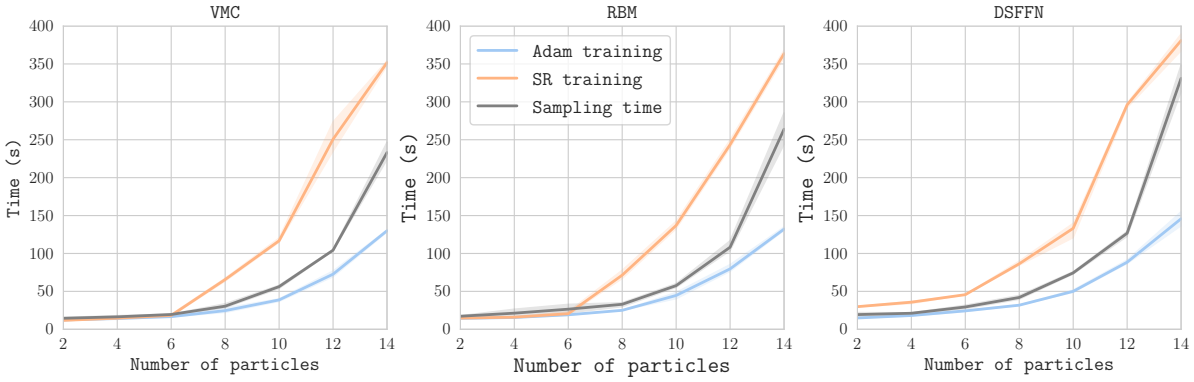


Figure 6.13: Wall time scaling in seconds as a function of number of particles, up to 14 electrons. We display separately the time for sampling and training, where the final wall time is their sum.

We should also mention that the sampling time does not depend on the optimiser, which is why only one common curve is displayed in Fig. 6.13. As expected, the SR method scales worse than Adam, from the costly requirement of inverting the approximate FIM at every epoch, together with the computation of the matrix elements.

To examine scaling estimations, we present polynomial and exponential fits of the wall time scaling in Tab 6.4. Furthermore, for each type of fit, we analyse the effects of constraining the proportionality factor to a constant value, $a = 0.5$. The analysis of the unconstrained coefficient reveals that the choice of optimiser, whether for the exponential or polynomial fit, does not significantly affect the order of scaling but rather impacts a constant factor.

Theoretically, the SR in its brute-force implementation does not scale in the same order as Adam, and it is hard to say if the difference of around 0.5 for the constrained polynomial fit is expected. While Adam's time complexity is $\mathcal{O}(N_\theta)$ in time and space, with N_θ the number of parameters, SR involves a matrix inversion time complexity ($\mathcal{O}(N_\theta^3)$) and takes $\mathcal{O}(N_\theta^2)$ in space. Our block diagonal approximation of the FIM indeed reduces the time and space complexities, but there are still (smaller) matrix inversions in place, as discussed in Sec. 5.7. It might be the case that some precompilation from JAX enables optimisations to take place.

If we accept the polynomial constant of $a = 0.5$, despite a worse R^2 score, all the methods follow a polynomial degree scaling between 2 and 3, with DSFFN being the worst and VMC the fastest as expected. A better assessment would surely require us to go beyond the 14 particles used.

Contrary to the DSFFN, the RBM and VMC do not require backpropagation under several compositions of non-linear activation functions. Also, the number of parameters are different. The VMC scales as $N \cdot d$ with particles N and dimensions d, while the RBM scales as $N \cdot d \cdot (1 + H) + H$, where H is the number of hidden nodes. For the choice of feed-forward network, the analysis is more complicated as the number of nodes changes for different layers. For timing purposes, we used architecture one from Tab. 6.1, which yielded good results with around $7d + L_d^2 + 8L_d + 68$ parameters, where L_d is the size of the latent dimension, often between four and ten.

Table 6.4: Polynomial and exponential fits of time scaling vs. Number of Particles. The leftmost fits do not constrain any coefficients, while the leftmost fits constrains $a = 0.5$.

Ansatz	**Opt.**	**Poly.** (aN^b)			**Exp.** (ae^{Nb})			**Poly.** ($0.5N^b$)		**Exp.** ($0.5e^{Nb}$)	
		a	**b**	$\mathbf{R^2}$	**a**	**b**	$\mathbf{R^2}$	**b**	$\mathbf{R^2}$	**b**	$\mathbf{R^2}$
VMC	Adam	0.03	3.15	0.96	2.89	0.27	0.99	2.05	0.91	0.40	0.92
	SR	0.13	3.01	0.99	8.09	0.27	0.98	2.48	0.97	0.48	0.84
RBM	Adam	0.07	2.85	0.96	3.93	0.25	0.99	2.07	0.93	0.40	0.88
	SR	0.18	2.90	1.00	9.14	0.27	0.99	2.49	0.99	0.48	0.83
DSFFN	Adam	0.17	2.53	0.93	5.29	0.24	0.98	2.10	0.92	0.41	0.79
	SR	0.43	2.59	0.97	14.37	0.24	0.97	2.52	0.97	0.48	0.73

Chapter 7

Two-dimensional Quantum Dots

We now present a series of results gathered for the two-dimensional trapped fermionic system in closed-shell configuration, also known as quantum dots. A better description of the physics of this problem is given in Sec. 2.6.

7.1 Initial Comparisons

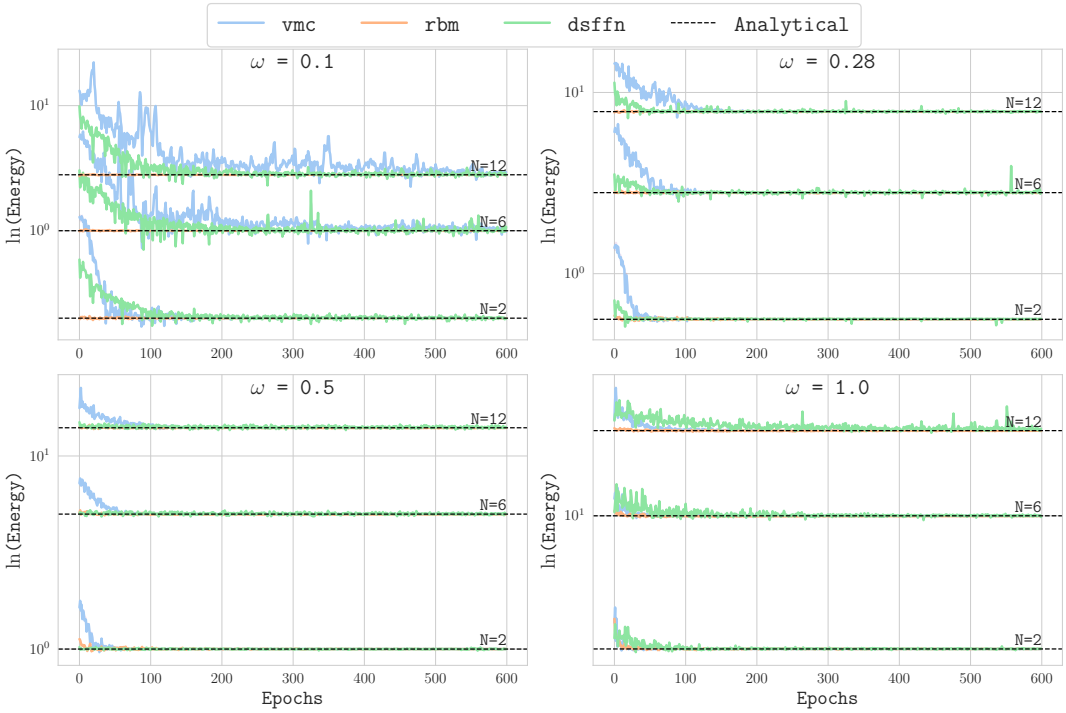


Figure 7.1: Logarithmic energy convergence curve for the non-interactive problem of two-dimensional quantum dots. In black dotted line, the analytical ground state energy is marked. The three displayed models are deep set FFN ("dsffn"), restricted Boltzmann machine ("rbm") and a standard variational Monte Carlo ("vmc"). Except for the Deep Set, Adam optimizer was used. The RBM line is hard to see as the RBM is initialised very close to the analytical solution.

We again start by showing in Fig. 7.1 that any of the models can obtain the ground state for different frequencies. We show four values of frequency, which will be used throughout the rest of the study, and we use different models and numbers of particles. Similarly to the case of the non-interacting, one-dimensional spinless fermions, we see that this task is reasonably simple,

as no fine-tuning or Jastrow factors were used. We display the energy values on a logarithmic scale for the ease of comparison between energies of systems with different numbers of particles.

7.2 Hyperparameter Search

In a manner similar to the one-dimensional scenario, we performed a set of hyperparameter optimisations for each of the four particle numbers tested, as detailed in 5.8. To avoid repetition, the individual results of these configuration sweeps are not shown, but they are comparable to what is presented in 6.5. Unlike the one-dimensional case, we fixed the batch size at 500 proposals and the training cycles at 400 for these short experiments. The parameters tested included the presence or absence of a pre-training step, the use of the regularised gradual Coulomb interaction scheme, different correlation factors (none, Padé-Jastrow, or Jastrow), optimisers, learning rates, and different latent dimension sizes, when the DSFFN is used. It should also be noted that a frequency of $\omega = 1$ was used for all sweeps at this stage.

7.2.1 Optimisers

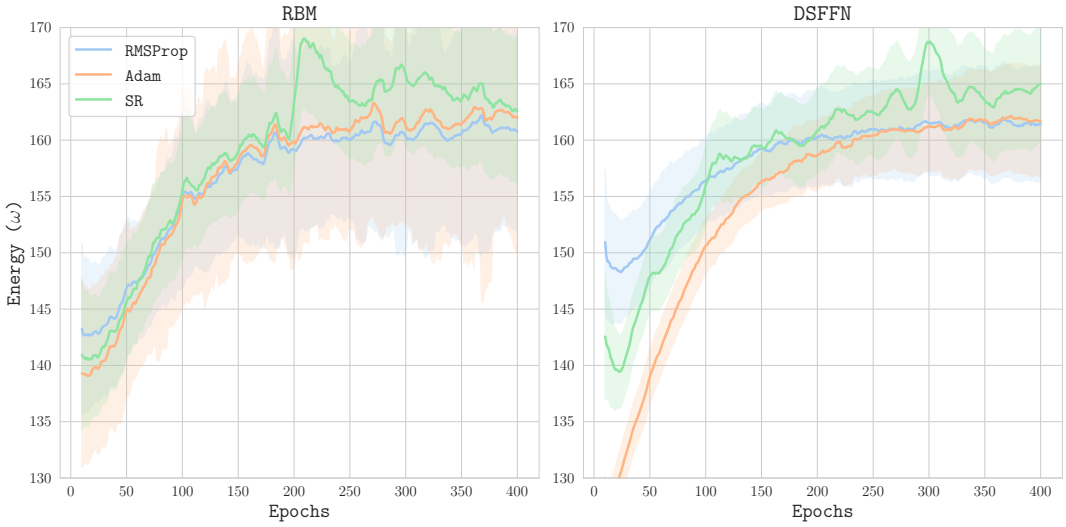


Figure 7.2: Aggregated values over a Bayesian sweep for 20 particles and frequency of 1.0. A similar trend is seen for both the RBM and DSFFN. The shaded region is not the error bars, but the minimum an maximum values obtained for over the sweep.

In terms of optimiser choice, we show in Fig. 7.2 the averages of 99 randomly selected parameters for a 20-particle case for the DSSFN ansatz. An analogous behaviour was observed for the two other trial functions. We note that the use of the Bayesian optimisation scheme, explained in 5.8, made the parameter search unbalanced and prevented the choice of stochastic reconfiguration. On that note, for the 20 particle case, SR was chosen 14 times, ADAM 25 and RMSProp was the most frequent, with 60 runs. This was seen for sweeps involving other number of particles as well, and will be further discussed.

The reason for the infrequent SR choice is that it showed very unpredictable behaviour with respect to the choice of other parameters. When SR managed to converge, it produced good outcomes; however, its variability and often lack of convergence between different configurations were substantial. Then, despite some good individual results, on average RMSProp or Adam were favoured. For this reason, except when showing some selected results, RMSProp was the optimiser of choice for the results to follow.

7.2.2 Correlation Factor

Still investigating the averaged results over random parameters, two analysis can be made with respect to the choice of correlation factor. The following analysis for Fig. 7.3 is done for a 12-particle case for the DSFFN, but the findings were the same for other choices of ansätze and number of particles. Firstly, as indicated in Fig. 7.3, the absence of any correlation factor resulted in the highest energy values, with the Jastrow factor ("j" in the figure) producing the lowest energy and the Padé-Jastrow factor ("pj") having the lowest energy.

The same can be said with respect to the standard deviation of the energy, further supporting that the Padé-Jastrow factor brings us closer to the ground-state. This indicates that, without correlation factors, even using an ansatz based on parametrised neural networks does not steer us too far from the Hartree-Fock picture, where one single Slater determinant does not enable us to capture significant correlations.

The lower energy values achieved for the Padé-Jastrow factor is also a positive theory confirmation, as it is supposed to satisfy Kato's cusp condition in a way that is not necessarily the case for the Jastrow factor [38], while also being symmetric under particle exchange.

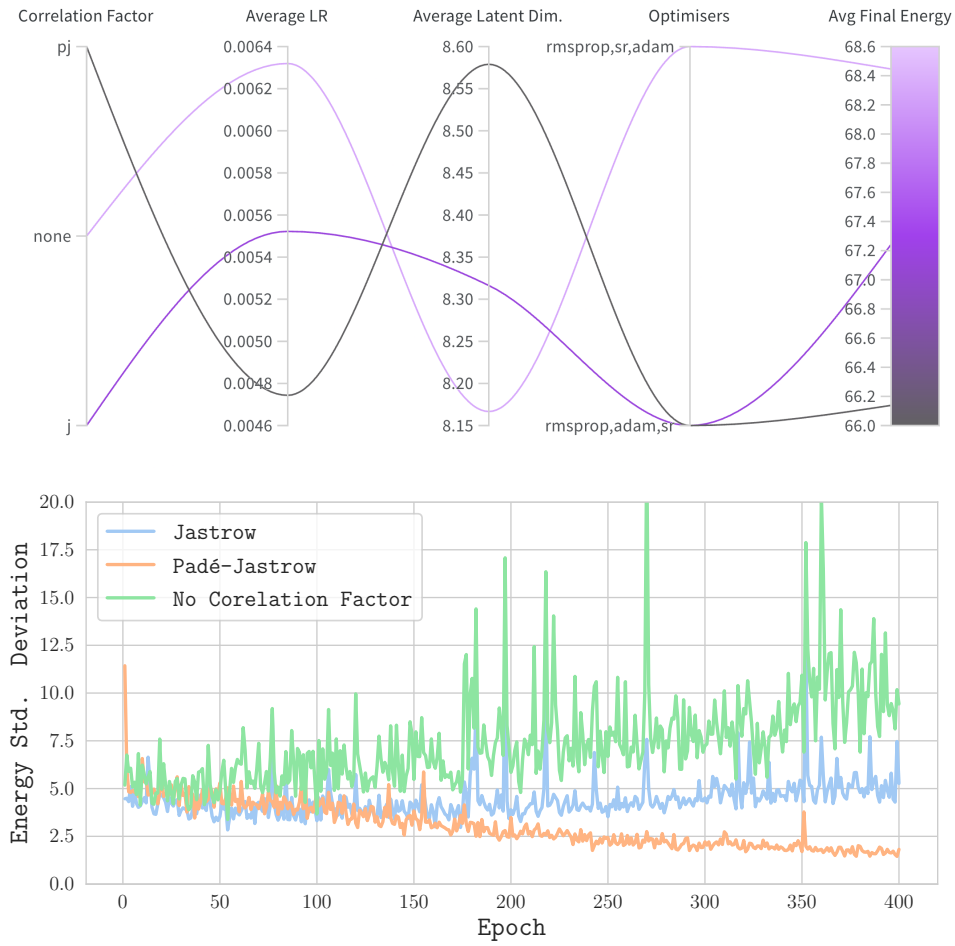


Figure 7.3: Aggregated values over a Bayesian sweep and several parameters (more than what is shown in the parallel line plot) for a 12-particle system and DSFFN trial function. 165 different parameter configuration were performed.

Lastly, an analysis can be performed with respect to the choice of correlation factor. Figure 7.4 shows the effect of the correlation factor with the use of a regularised gradual Coulomb

interaction potential. Despite these results only being shown for the VMC ansatz and two particles, what is evident is that consistently better results were achieved with both the use of a Padé-Jastrow factor together with the regularised gradual Coulomb interaction. This point is particularly clear when looking at the standard deviations. Although the Padé-Jastrow curve, without the regularised potential, may eventually achieve a lower minimum than the Jastrow curve after the 400 epochs displayed, this occurs much faster with the inclusion of the gradual Coulomb interaction. Again, it is evident that without a correlation factor, the results were sub-optimal in terms of both energy reduction and standard deviation.

Although we do not demonstrate it here, the combination of gradual Coulomb and the Padé-Jastrow factor was also consistently favoured when using the Deep Set network. While nonlinear activation functions theoretically allow the network to capture particle correlations, and this process is further simplified with the regularised gradual Coulomb, the results were better with the combination of these methods than with them in isolation.

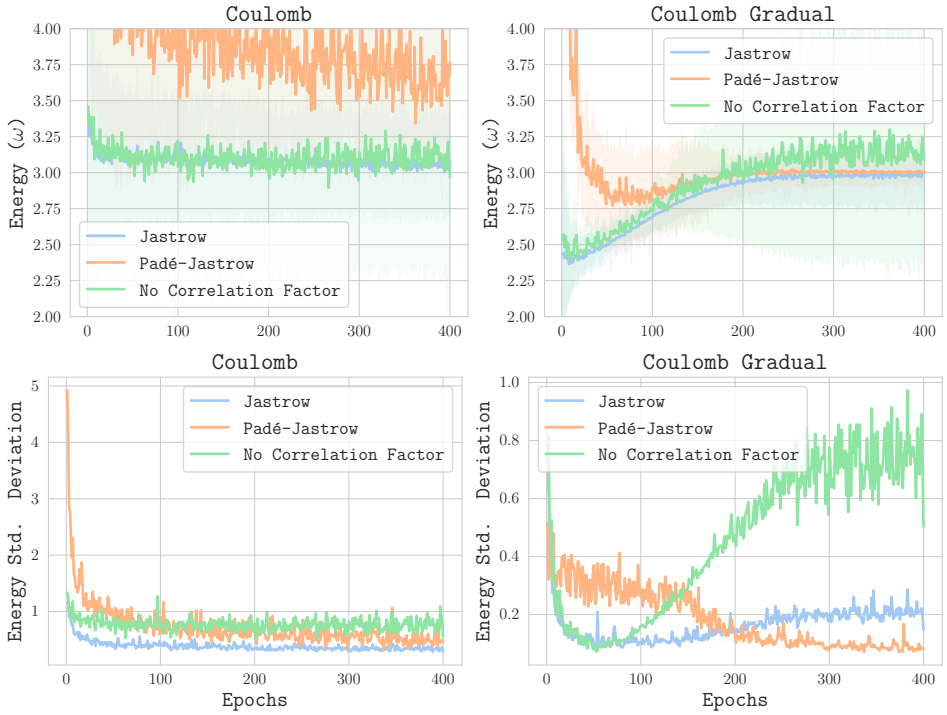


Figure 7.4: Aggregated values over a Bayesian sweep and several parameters (see text) for the VMC ansatz and two particle case. Here, a frequency of $\omega = 1$ is selected. The shaded region is not the confidence interval, but the maximum and minimum values obtained from all parameter configurations.

7.3 One and Two-Body Densities

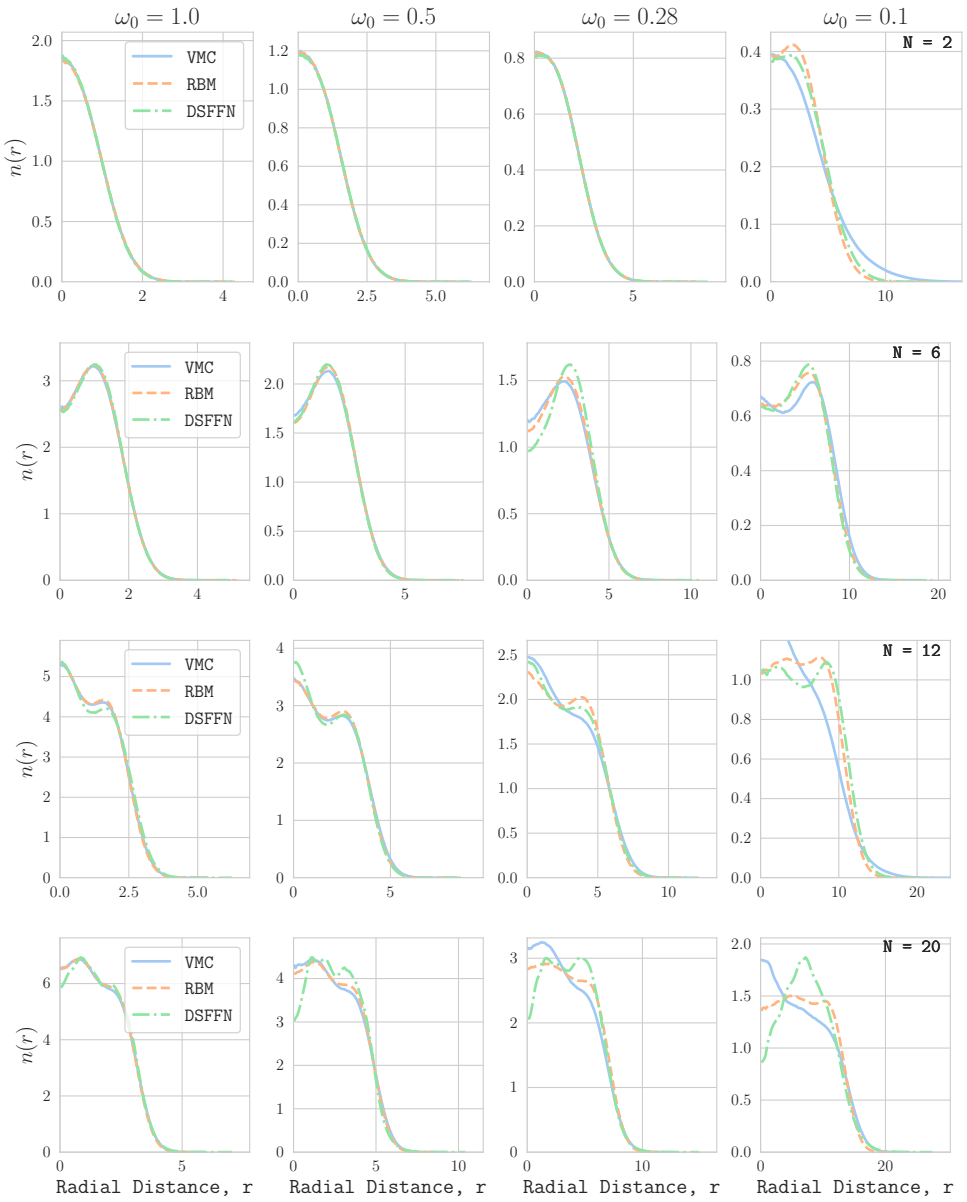


Figure 7.5: Radial density profiles for different ansätze, after 2^{24} MC samples. All ansätze were multiplied by a Padé-Jastrow factor, and the sampled energies can be found in Tab. 6.3

In Fig. 7.5, we display one-body radial densities for up to 20 particles and for different frequencies. A comparison can be made in terms of the different trial functions, but all the models shown contained a Padé-Jastrow factor. It is clear that for higher frequency traps (leftmost in the plots), the radial scale significantly changes. One might overlook the fact that the plots are rescaled for consistent visibility. As expected, for higher frequencies, the particles are more localised and confined closer to the origin. In terms of particle number, again as expected, the profile heights are more pronounced with a larger number of particles, as $n(r)$ integrates to N. One sees that a higher density of fermions also induces more peaks in the profile. This is because there is less effective space, yet fermions still have to satisfy the exclusion principle, where fermions must occupy distinct energy levels. Then, the occupation of these levels lead to

a shell structure where the fermions are forced to occupy higher energy states which leads to the fringes in the density profile.

For lower-frequency traps, we also know that the quantum energy levels become closer together. This closer spacing directly impacts the character of the fringes in the density profile. Due to the closer spacing of the energy levels, more levels must be populated because of the exclusion principle. This results in a greater number of fringes appearing in the density profile. Nevertheless, these fringes are typically less distinct and more diffuse compared to those in higher frequency traps. Despite the difficulties in capturing correlations for the low-frequency regime, all models in Fig. 7.5 seem to replicate this in some way.

For different ansätze, there is generally good agreement for higher frequencies. This is not the case for the inferior right part Fig. 7.5. However, knowing which of the ansatz better represents what is expected is hard only with the density profiles, and this analysis will have to wait until we investigate the energy values.

Although unfortunate, this discrepancy between the profiles for low-frequency and higher number of particles is expected. Lower frequency traps, which are spatially wider, are particularly hard to model when the Coulomb interaction is present. This is due to the long-rage interaction of the Coulomb potential, which follows $1/r$ with r the distance between the particles. Then, even when separated by large distances, the particles still significantly affect each other. This increases the complexity of the correlation, which becomes less localised. This would not be the case, for example, if instead we had a Yukawa potential, which takes the form $\exp(-kr)/r$, with k being a constant. This potential decreases more rapidly as the distance between particles increases.

The exact same analysis that was performed for the radial density profile can also be done for the two-dimensional one-body densities of Fig. 7.6. We therefore do not repeat it here, but only display the 12 and 20 particle cases for a frequency of $\omega = 1.0$. The two and six particle cases can be seen in the Appendix G.

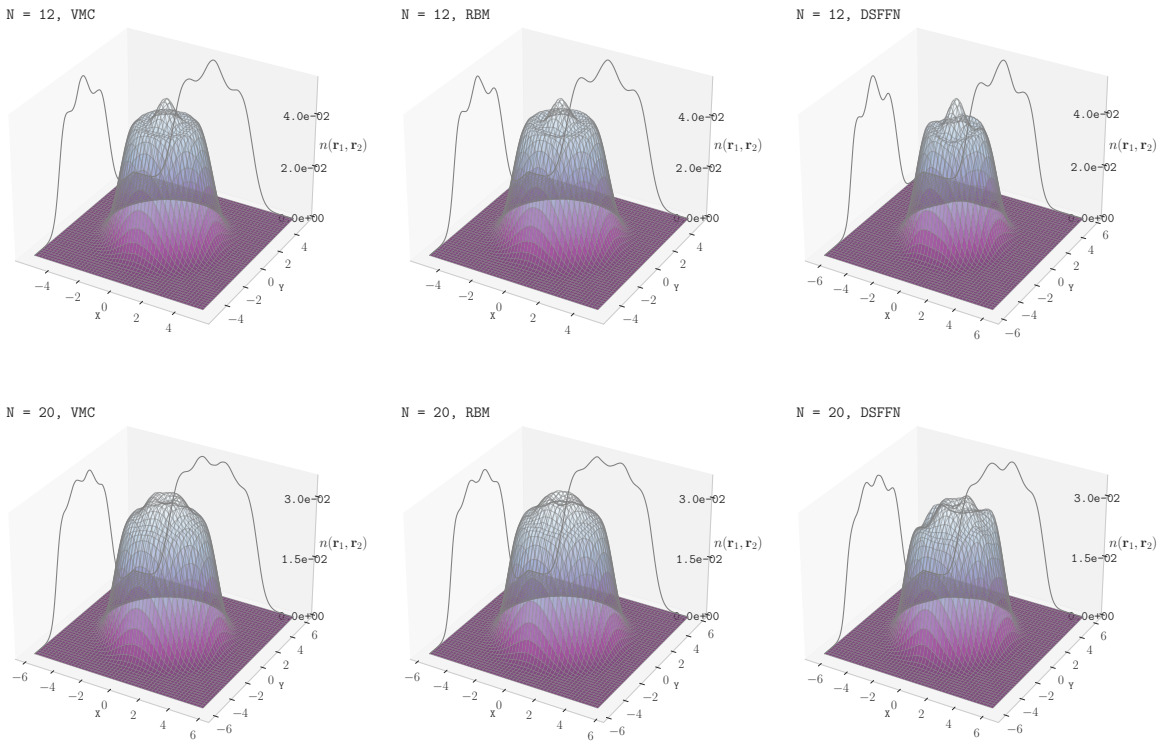


Figure 7.6: Two-dimensional one body density profiles for 12 and 20 particles, for all the ansätze used, all of which contained a Jastrow factor. The profiles were made from 2^{24} samples, and the final average energies for these can be seen in Tab. 7.2

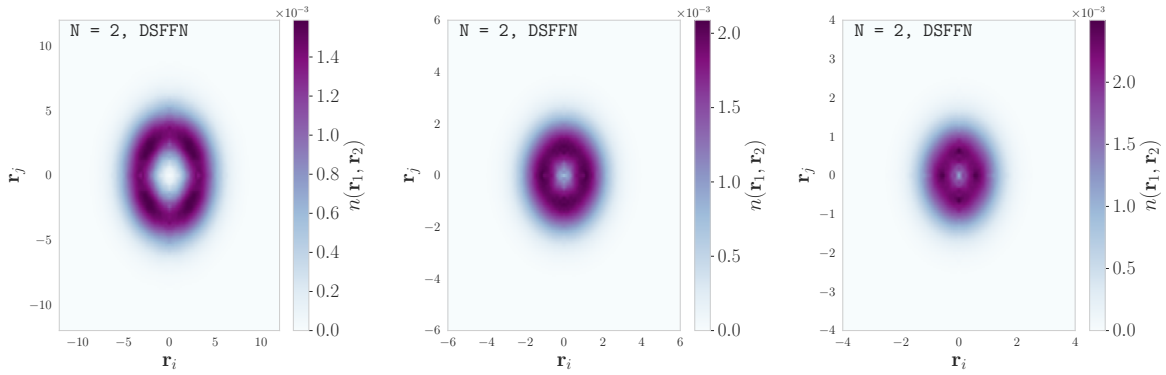


Figure 7.7: Radial two-body density profiles for the DSFFN ansatz and two particles. From left to right we have frequencies of $\omega = 0.1$, $\omega = 0.5$ and $\omega = 1.0$. The results shown were obtained from 2^{24} samples and 3000 training epochs with RMSProp. The densities of the first quadrant were mirrored on the three others to give a symmetric representation.

Figure 7.7 shows that the lower frequency traps, which are wider spatially, also reflect this wider profile in the two-body densities. In fact, pairs of particles are less likely to be seen in the middle of the trap with a lower frequency value. While this figure shows, for the case of two particles, how the density of two bodies changes with frequency, Fig. 7.8 shows how the density of two bodies depends both on the number of particles and the choice of ansatz, but now for $\omega = 0.5$.

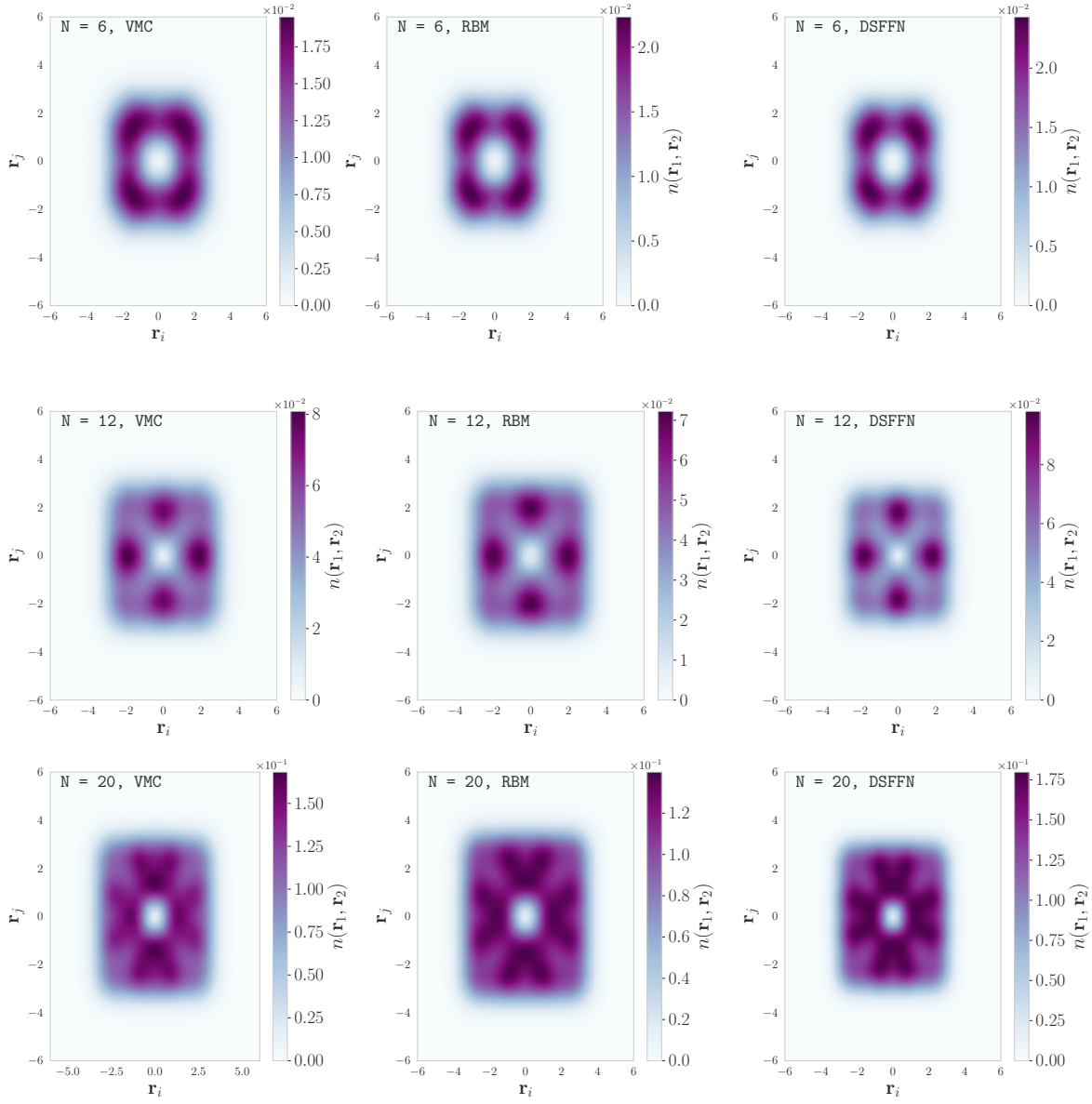


Figure 7.8: Radial two-body density profiles for $\omega = 0.5$, for all ansätze and different numbers of particles. From top to bottom we have 6, 12 and 20 particles and from left to right we have VMC, RBM and DSFFN. The results shown were obtained from 2^{24} samples, trained with 3000 epochs with RMSProp. The densities of the first quadrant were mirrored on the three others to give a symmetric representation.

It seems that, from Fig. 7.8, all models are equally capable of capturing approximately the same correlations. The density profiles exhibit a clear symmetry around the origin. Of course, this symmetry was artificially induced by us, in the process of converting the two spatial coordinates into a radial one and replicating the obtained results in the other quadrants. However, this is not an unreasonable procedure, as we see from the one-body density profiles of Fig. 7.5 that the position distribution is radially symmetric.

We also notice that pairs of particles prefer to avoid the trap's central region due to their repulsion. As the particle count increases, there is a trend towards the localisation of particle pairs, avoiding the diagonals, which are equidistant from the origin. It appears as though the particles are avoiding the same spatial sphere, which aligns with physical intuition. This pattern is evident with 12 particles, but we observe changes as more particles are added, where they

start to avoid not only diagonals but also the principal axes.

7.4 Overall Energy Comparison

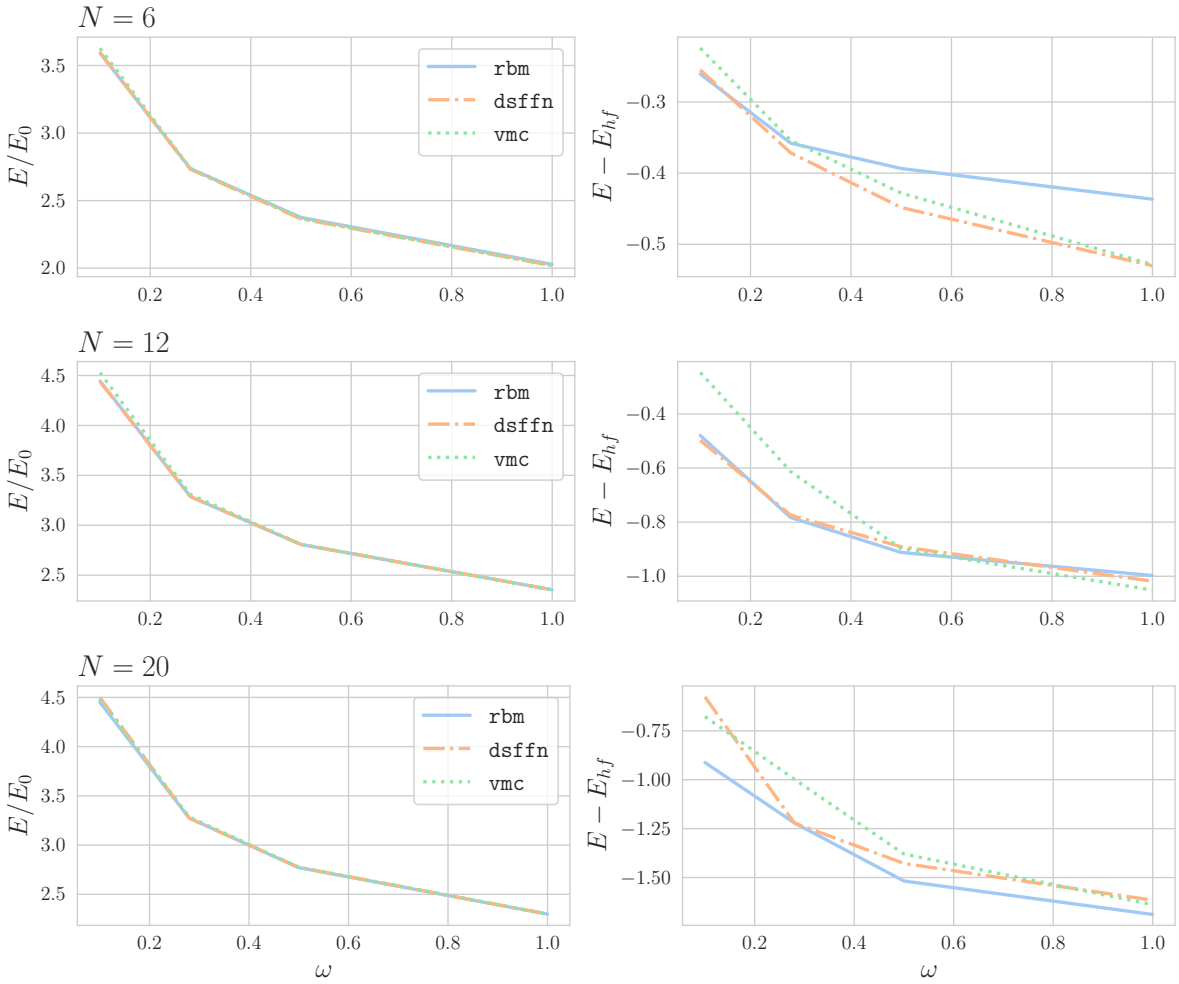


Figure 7.9: On the left, the ratio between the measured energy and the non-interactive energy for the same system. On the right, the measure of the correlation energy, $E - E_{hf}$, where E_{hf} is the Hartree-Fock energy value taken from [60]. The two-particle case is too similar to the others and is shown in the Apendix G

The right side of Fig. 7.9 is a good reassurance that all our models used achieve energies lower than the HF energy for all ranges of frequencies. These results were gathered after training for 3000 epochs and for 2^{24} MC samplings. This is particularly due to the addition of the Jastrow factor in all models at this point. While all models show comparable correlation energies, the RBM model demonstrates exceptionally good performance for the 20-particle case but performs poorly for the six-particle case. The reasons for this discrepancy remain unclear, but it is likely due to sub-optimal hyperparameter choices. Typically, we would anticipate the opposite behaviour.

The fact that Fig. 7.9 shows a correlation energy closer to 0 for lower frequency traps further supports our previous discussion about the difficulty of capturing intricate correlations due to the long-range Coulomb interaction. The larger the frequency value, the easier it is for our model to capture correlations that HF is not capable of.

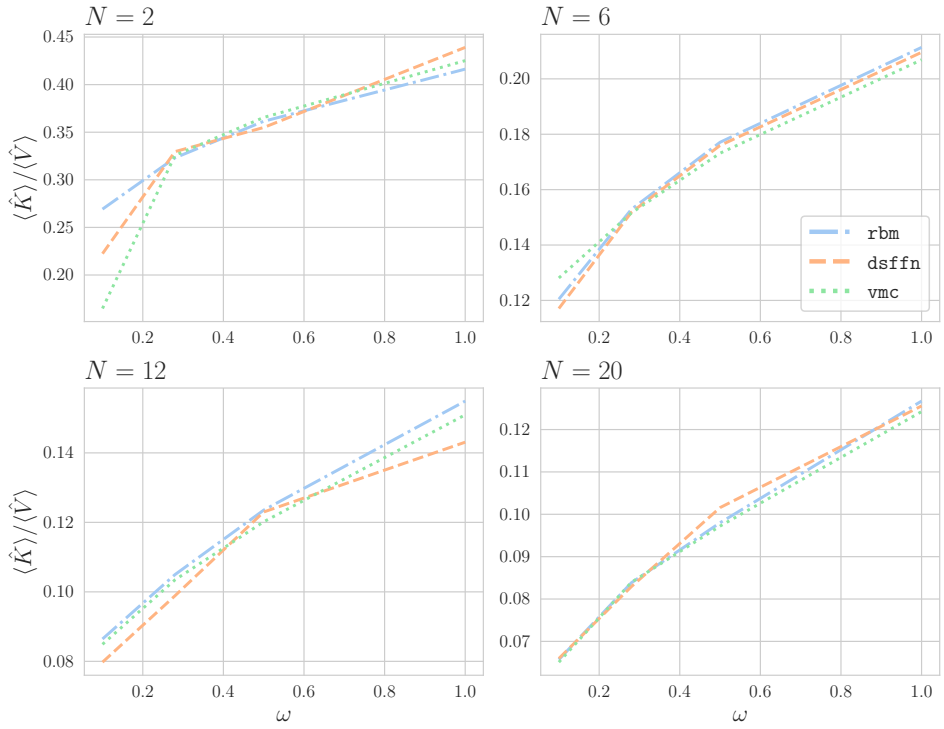


Figure 7.10: Fraction between kinetic energy and total potential energy as function of the frequency. The total potential energy corresponds to the sum of of interaction energy and the trap energy.

The left-hand side of Fig. 7.9 shows that as a fraction of the non-interactive energy, all models agree well. The increase in fraction E/E_0 with a decrease in frequency aligns with what is found in other studies. As discussed in [51], in fact, this ratio diverges, due to its scaling as $(\omega^{-1/3})$ in the limit of $\omega \to 0$.

Finally, the proportion between energy components can be investigated in Fig. 7.10 and in more detail for a six-particle case, in Tab. 7.1. Naturally, because we are in an interactive regime, the virial theorem is not expected to hold. While there is a distinct trend, a more detailed analysis would require us to investigate beyond the frequency of 1.0 and below 0.1. This is because, with the range investigated, we are unable to replicate the expected plateau in the ratio of $\langle K \rangle$ and $\langle V \rangle$ observed in [69]. The trend we observe at least indicates that with an increase in frequency comes a domination of the kinetic energy term. At low frequencies, the opposite is observed, with potential energy becoming the dominant component of the energy profile, as anticipated. This ratio decreases for larger quantum dots, where the interaction energy increases in proportion as a result of the higher particle count.

Ansatz	ω	E	$\langle\hat{K}\rangle$	$\langle\hat{V}_{trap}\rangle$	$\langle\hat{V}_{int}\rangle$
dsffn	0.5	11.823(1)	1.769(3)	4.261(5)	5.793(3)
	1.0	20.189(1)	3.498(5)	7.651(7)	9.040(4)
	0.28	7.6481(9)	1.008(2)	2.756(4)	3.884(3)
	0.1	3.5975(7)	0.377(1)	1.266(2)	1.954(2)
rbm	0.5	11.8778(9)	1.787(2)	4.308(4)	5.782(3)
	1.0	20.2826(8)	3.539(4)	7.658(5)	9.086(3)
	0.28	7.6617(7)	1.016(2)	2.702(3)	3.943(2)
	0.1	3.5917(5)	0.386(1)	1.292(2)	1.913(1)
vmc	0.5	11.8432(8)	1.748(2)	4.370(3)	5.725(2)
	1.0	20.1908(8)	3.461(4)	7.727(5)	9.003(3)
	0.28	7.6657(7)	1.009(2)	2.729(2)	3.928(2)
	0.1	3.6281(6)	0.412(1)	1.397(2)	1.819(1)

Table 7.1: Energy components from the six particle case for the energy values displayed in 7.2.

Lastly, we display in Tab. 7.2 a collection of the energy values for all models at different frequencies and different numbers of particles. These values were the corresponding values associated to the body densities analysed and all plots other than the ones in the hyperparameter parameter investigation of 7.2. It should be noted that the displayed values result from experiments conducted with a mix of common and unique parameters. Specifically, although all results were achieved after training for 3000 epochs, using RMSProp (except for the SR column), and a batch size of 1000 proposals per epoch, some experiments varied in learning rates and network architectures. This variation was unavoidable, as the scale of the system dictates the architectural choices in several instances, and for larger quantum dots, smaller learning rates were required.

N	ω	**DSFFN + PJ** SR	**DSFFN + PJ** RMSProp	**RBM + PJ**	**VMC + PJ**	**HF** [60]	**DMC** [36]
2	0.1	**0.44146(4)**	0.4463(2)	0.44126(3)	0.545(1)	0.525635	0.44079(1)
	0.28	**1.02218(2)**	1.02248(4)	1.02172(1)	1.02257(3)	1.14171	1.02164(1)
	0.5	**1.66047(6)**	1.66097(7)	1.65982(1)	1.66062129(6)	1.79974	1.65977(1)
	1.0	3.00070(5)	3.0023(1)	**3.00009(1)**	3.00108(5)	3.16190	3.00000(1)
6	0.1		3.5975(7)	3.5917(5)	3.6281(6)	3.85238	3.55385(5)
	0.28	**7.6361(4)**	7.6481(9)	7.6617(7)	7.6657(7)	8.01957	7.60019(6)
	0.5	**11.800(4)**	11.823(1)	11.8778(9)	11.8432(8)	12.271300	11.78484(6)
	1.0	**20.156(7)**	20.189(1)	20.2826(8)	20.1908(8)	20.719200	20.15932(8)
12	0.1		**12.427(1)**	12.445(2)	12.678(3)	12.9247	12.26984(8)
	0.28		25.776(1)	**25.767(2)**	25.937(2)	26.5500	25.63577(9)
	0.5		39.325(1)	**39.304(2)**	39.318(2)	40.2161	39.1596(1)
	1.0		65.893(1)	65.914(2)	**65.860(2)**	66.9113	65.7001(1)
20	0.1		30.614(4)	**30.278(3)**	30.513(4)	31.1902	29.9779(1)
	0.28		**62.317(2)**	62.318(5)	62.541(4)	63.5390	61.9268(1)
	0.5		94.305(2)	**94.215(4)**	94.353(4)	95.7328	93.8752(1)
	1.0		156.388(3)	**156.315(4)**	156.365(4)	158.004	155.8822(1)
Average			33.5272	33.5058	33.6916	34.1600	33.3523

Table 7.2: Collection of the results for different frequencies and ansätze choices, in comparison to Hartree-Fock and DMC energies. The text in bold font marks the best obtained value for that row. All values were obtained with RMSProp except when shown otherwise. For training, 3000 epochs were used with batch size of 1000. For sampling, 2^{24} proposal steps are taken. The missing values for SR mean that we were unable to achieve convergence.

All ansätze in Fig. 7.2 showed energy averages lower than Hartree-Fock, but higher than DMC. If we for now disregard the SR column we see that the preferred model was the RBM together with the Padé-Jastrow factor. The DSFFN network was very similar, differing by just 0.02 on average, while the VMC had the highest energies.

To address the use of the Stochastic Reconfiguration method, we revisit the topic briefly mentioned in the Bayesian hyperparameter search of Sec. 7.2. When this method converged, it demonstrated great results. For instance, at a frequency of $\omega = 1.0$ with six particles, we achieved results superior to those obtained from the DMC calculations. Unfortunately, achieving such a convergence proved to be a challenge. The required parameter choices were unpredictable and lacked intuitive guidance.

One potential explanation is that the neural networks employed were simply too complex models. Although this might be a contributing factor, our experiments with smaller networks did not yield any energy convergence patterns with any of the optimisers tested. Only with more expressive and larger networks did we achieve good results beyond one-dimensional systems.

There might be an unfortunate gap in the complexities of the models tested. A 20-particle VMC ansatz used 40 variational parameters and behaved predictably based on parameter choices. For example, a smaller learning rate generally led to more stable convergence, similar to using a larger batch size. A similar argument applies to the RBM, where changing the number of hidden nodes, sometimes favoured training. An RBM we used for the 20-particle case with 6 hidden nodes displayed 200 parameters, and the Deep Set networks that we used had a similar number. However, we speculate that the non-linear activation layers significantly increased the model's complexity. While this is in general desirable, as we want to be able to capture complex pattern in data, it seems to have made the training process more difficult than

expected.

7.5 Time Scaling Analysis

Both Fig. 7.11 and Table 7.3 analyse the wall time scaling for the three models, only for the RMSProp optimiser. The choice of this optimiser was based on its use in presenting nearly all results for this two-dimensional system. Then, while some low-energy values were obtained with SR, we reserve its time scaling discussion for the one-dimensional system, Sec. 6.6. Here we analyse the averages over three independent runs for each measurement, with 2^{19} sampling steps, up to 22 particles, and for 500 training epochs.

Interestingly, Fig. 7.11 and Tab. 7.3 seem to indicate very similar polynomial-order scaling for the different trial functions. This affirmation is based on the very close values of b in the polynomial or exponential fits with a constrained constant factor. In Sec. 6.6 we do a more detailed analysis of how the methods scale with the number of parameters. This analysis suggests that the DSFFN should scale worse than what was measured, and we present some hypotheses as to why this is not the case.

Firstly, there might be efficient accelerated linear algebra optimisation at play due to the use of just-in-time compilations. More importantly, the choice of architecture size for the DSFFN might have been too small. Here we measure the wall times for the smallest architecture used (architecture one in Tab. 6.1). This was not to be deceptive, but instead a random decision, as we needed to employ varying architectures based on the particle count and frequency to achieve optimal results.

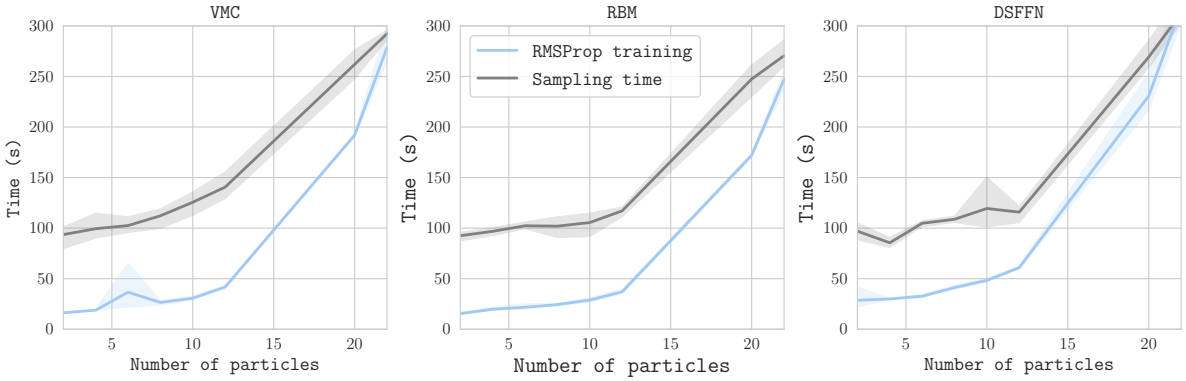


Figure 7.11: Wall time scaling in seconds as a function of number of particles, up to 22 electrons. We display separately the time for sampling and training, where the final wall time is their sum.

For the sake of comparison, it is worth mentioning that we can evaluate our scaling against another study that implemented the same system in C++ [69]. In their study, they did not use automatic differentiation as we do. For their RBM, under the same constraint of $a = 0.5$, they achieved a polynomial degree of approximately $b = 1.5$. This indicates that our scaling is poorer, which is expected given that we are using Python. Nevertheless, our scaling is not significantly worse, and our energy results are comparable.

Table 7.3: Polynomial and exponential fits of time scaling vs. Number of Particles. The leftmost fits do not constrain any coefficients, while the leftmost fits constrains $a = 0.5$.

Ansatz	**Opt.**	**Poly.** (aN^b)			**Exp.** (ae^{Nb})			**Poly.** $(0.5N^b)$		**Exp.** $(0.5e^{Nb})$	
		a	**b**	$\mathbf{R^2}$	**a**	**b**	$\mathbf{R^2}$	**b**	$\mathbf{R^2}$	**b**	$\mathbf{R^2}$
VMC	RMSProp	0.05	2.76	0.96	6.60	0.17	0.98	2.02	0.94	0.29	0.93
RBM	RMSProp	0.04	2.78	0.98	5.70	0.17	0.99	1.98	0.96	0.28	0.94
DSFFN	RMSProp	0.24	2.32	0.97	11.68	0.15	0.99	2.07	0.96	0.30	0.89

Part IV

Conclusion

Chapter 8

Conclusion

Concluding Points

In conclusion, this thesis has explored the application of various machine learning techniques to solve quantum many-body problems, specifically focussing on one-dimensional trapped spinless fermions and two-dimensional quantum dots. By making use of neural networks such as Deep Set feed-forward networks (DSFFN) and restricted Boltzmann machines (RBM), we have demonstrated the potential of these networks to be used as variational Monte Carlo (VMC) functions, following ideas of reinforcement learning. From our findings, this approach enabled us to approximate ground-state energies and wavefunctions with good accuracy.

In particular, the effectiveness of neural network ansätze was evident as the DSFFN in combination to stochastic reconfiguration yielded in an isolated case, energy values below DMC calculations, and overall very good results. If an average analysis of the energies is made and if we disregard the stochastic reconfiguration method due to its difficult convergence behaviour, the RBM implementation was slightly favoured in the two-dimensional fermionic trap with Coulomb interaction. In contrast, for the one-dimensional case, where SR convergence was easy and used for all ansätze, the VMC without neural networks was optimal.

In general, the inclusion of correlation factors, such as the Jastrow and Padé-Jastrow factors, significantly improved the energy estimates, bringing them closer to the reference values of other research. This improvement from the addition of correlation factors was present in both one-dimensional and two-dimensional systems, and the Padé-Jastrow factor, in particular, was shown to be the best correlation factor for the Coulomb interaction potential.

We additionally were able to study correlations from one- and two-body density profiles, observing typical fermionic behaviour. In the spinless fermion system, we were able to observe crystallisation and bosonisation under the repulsive and interaction attractive regimes, respectively. This was also observed for the two-dimensional system as a function of the frequency of the trap. Similarly, we were able to study the distribution of energy components for both cases qualitatively, further showing that neural networks can be used to extract and understand the physics of simulated systems. For large quantum dots systems and lower frequency values, it was significantly hard for all ansätze to obtain equally good results. Nonetheless, the values obtained were still acceptable and below the Hartree-Fock energy.

The optimisation of hyperparameters through a Bayesian approach proved to be useful, yet misleading, in some cases. While it guided us to generally good choices, it prevented us from properly experimenting with the optimiser that eventually yielded the best results. For the two-dimensional quantum dot system, systematically lower energies were found with the SR reconfiguration method. However, this method was avoided by Bayesian optimisation search due to a rare convergence. Different optimisers, learning rates, and network architectures were explored, with RMSProp and Adam being overall the most robust for the tested configurations. In fact, SR yielded excellent results, but Adam and RMSProp were consistently good, rarely displaying divergence.

The time scaling analysis indicated that the neural network models could handle an increasing number of particles with reasonable computational resources. Although Python showed slightly poorer scaling compared to C++ implementations, overall performance was not a constraint in our case.

In summary, we can confidently say that the work here developed demonstrated the promising capabilities of machine learning techniques in tackling quantum many-body problems. Our investigations contributed to an understanding of the newly conceptualised field of neural quantum states, while also providing meaningful results, comparable to other studies.

Paths For Future Work

This thesis concludes with several possible paths for future research. Given the reasonable quality of the results obtained, a natural progression would be to extend our methodologies to larger quantum systems. This expansion would allow us to investigate the limits of our methods in more complex scenarios, exploring different interaction potentials and, at the same time, higher-dimensional systems.

An orthogonal but equally important direction is to improve the accuracy and reliability of our quantum state optimisation. As already mentioned, our results obtained with the stochastic reconfiguration method demonstrated very positive results, but with unreliable behaviour. Then perhaps the most immediate path to try and improve this method is to investigate recent advancements in optimisation techniques, such as the Decision Geometry approach proposed by [20]. Implementing and evaluating these novel optimisation schemes could enhance the convergence and stability of our models.

Our current incorporation of physical constraints to the ansatz is also somewhat primitive. A natural path to improve this would be to extend our Deep Set implementations to utilise equivariant layers, constructing more general Slater determinants with the networks parametrising the single-particle states while embedding backflow correlations [59]. Similarly, exploring the integration of a Pfaffian ansatz, as demonstrated in the work of [46], could be a fruitful direction.

Lastly, to enable more extensive parameter experimentation and improve ground-state minimisation for larger systems, it would be necessary to conduct a thorough profiling of our JAX implementation to identify and address performance bottlenecks.

Part V

Appendices

A Eliminating Dimensions

As described in Sec. 2.6, we will prefer to deal with energy units of either $\hbar\omega$ or $\hbar$ and lenght units of $a_{ho} = \sqrt{\hbar/m\omega}$. We now show how this allows us to write the simplified Hamiltonian expressions. Starting from the noninteractive part of the Hamiltonian,

$$\hat{H}_0 = \sum_i^N \left(\frac{-\hbar^2}{2m}\nabla_i^2 + \frac{1}{2}m\omega^2 r_i^2 \right),$$

we make a change of coordinates $r \to r/a_{ho}$ so that the Cartesian coordinates become $\hat{x}_i = x_i/a_{ho}$ and the components of the gradient vector undergo the change of coordinates

$$\frac{\partial}{\partial \hat{x}_i} = \frac{\partial}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial x_i} = \frac{1}{a_{ho}}\frac{\partial}{\partial \hat{x}_i},$$

which naturally propagate to the Laplacian as

$$\hat{\nabla}_i^2 = \frac{1}{a_{ho}^2}\nabla_i^2.$$

Now, in the new coordinate system, but going back to the original notation not to pollute the demonstration, the Hamiltonian turns into

$$\hat{H}_0 = \sum_i^N -\frac{\hbar}{2m}\frac{1}{a_{ho}^2}\nabla_i^2 + \frac{m}{2}\left(\omega r_i/a_{ho}\right)^2.$$

Using the definition of a_{ho}, some terms simplify giving

$$\hat{H}_0 = \sum_i^N -\frac{\hbar\omega}{2}\nabla_i^2 + \frac{\hbar\omega}{2}r_i^2.$$

Finally, by using energy units of $\hbar\omega$,

$$\frac{\hat{H}_0}{\hbar\omega} = \sum_i^N -\frac{1}{2}\nabla_i^2 + \frac{1}{2}r_i^2.$$

Now for the interactive part of the Hamiltonian, we have two scenarios. For the Coulomb interaction, can use an arbitrary unit of charge such that

$$\sum_{i<j} V_{int}(\mathbf{r}_i, \mathbf{r}_j) = \frac{1}{4\pi\epsilon_0}\sum_{i<j}\frac{q_i q_j}{r_{ij}} \to \sum_{i<j}\frac{1}{r_{ij}},$$

where ϵ_0 is the permittivity of free space, and q_i the charge of particle i.

For the finite range interaction, we redefine the interaction range $\sigma_0 = \sigma/a_{ho}$ and the interaction strength $V_o = V/(a_{ho}\hbar\omega)$

$$\sum_{i<j} V_{int}(\mathbf{r}_i, \mathbf{r}_j) = \frac{V}{\sigma\sqrt{2\pi}}\sum_{i<j}\exp\left[-\frac{r_{ij}^2}{2\sigma^2}\right] \to \frac{V_0(a_{ho}\hbar\omega)}{\sigma_0 a_{ho}\sqrt{2\pi}}\sum_{i<j}\exp\left[-\frac{a_{ho}^2 r_{ij}^2}{2a_{ho}^2\sigma_0^2}\right],$$

Which finally, in energy units of $\hbar\omega$ becomes

$$\sum_{i<j} V_{int}(\mathbf{r}_i, \mathbf{r}_j) = \frac{V_0}{\sigma_0\sqrt{2\pi}}\sum_{i<j}\exp\left[-\frac{r_{ij}^2}{2\sigma_0^2}\right].$$

B VMC Derivations

Let $f(\boldsymbol{r}) = \ln \Psi(\boldsymbol{r})$ and $E_L = K_L + V_L$. Note then that

$$\nabla f(\boldsymbol{r}) = \frac{\nabla \Psi(\boldsymbol{r})}{\Psi(\boldsymbol{r})} \tag{1}$$

$$\nabla^2 f(\boldsymbol{r}) = \frac{\nabla^2 \Psi(\boldsymbol{r})}{\Psi(\boldsymbol{r})} - \left(\frac{\nabla \Psi(\boldsymbol{r})}{\Psi(\boldsymbol{r})}\right)^2, \tag{2}$$

which leads to

$$K_L(\boldsymbol{r}) = -\frac{1}{2}\frac{\nabla^2 \Psi(\boldsymbol{r})}{\Psi(\boldsymbol{r})} \tag{3}$$

$$= -\frac{1}{2}\left(\nabla^2 f(\boldsymbol{r}) + (\nabla f(\boldsymbol{r}))^2\right). \tag{4}$$

C Steepest Descent Derivations

We posed the steepest descent problem as the constrained optimisation of minimising

$$\mathcal{L}(\boldsymbol{\theta}_t) + (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top \nabla \mathcal{L}(\boldsymbol{\theta}_t) + \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2,$$

subject to $\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2 = \epsilon$.

For that we introduce a Lagrange multiplier λ to incorporate the constraint into the objective function, leading to the Lagrangian:

$$\mathcal{G}(\boldsymbol{\theta}, \lambda) = \mathcal{L}(\boldsymbol{\theta}_t) + (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top \nabla \mathcal{L}(\boldsymbol{\theta}_t) + \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 + \lambda\left(\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2 - \epsilon\right).$$

Taking derivatives with respect to $\boldsymbol{\theta}$ and λ, and seting them to zero, we have

$$\nabla_{\boldsymbol{\theta}} \mathcal{G}(\boldsymbol{\theta}, \lambda) = \nabla \mathcal{L}(\boldsymbol{\theta}_t) + (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \lambda \frac{\boldsymbol{\theta} - \boldsymbol{\theta}_t}{\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|} = 0,$$

$$\frac{\partial \mathcal{G}}{\partial \lambda} = \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\| - \epsilon = 0.$$

The gradient with respect to λ simply enforces the constrain, $\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\| = \epsilon$, but from the gradient with respect to $\boldsymbol{\theta}$ we find

$$(\boldsymbol{\theta} - \boldsymbol{\theta}_t)(1 + \frac{\lambda}{\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|}) = -\nabla \mathcal{L}.$$

Here we could further investigate the λ factor, but the important thing is that this yields

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}_t),$$

for some α that depends on the constraint ϵ.

D Gaussian-Binary RBM Expressions

The marginal probability distribution for the Gaussian-binary RBM's visible nodes is given by

$$p(\mathbf{R}) = \exp\left\{-\sum_i^{n_r} \frac{(r_i - a_i)^2}{2\sigma^2}\right\} \sum_{\{\boldsymbol{h}\}} \exp\left\{\sum_j^{n_h} b_j h_j + \sum_{i,j}^{n_r, n_h} \frac{r_i w_{ij} h_j}{\sigma^2}\right\},$$

where we can rewrite the summation over all possible vectors $\boldsymbol{h}$ as

$$\sum_{\{h\}} \exp\left(\sum_j \left(b_j + \sum_i \frac{r_i w_{ij}}{\sigma^2}\right) h_j\right).$$

Now, if we denote for simplicity:

$$\theta_j = b_j + \sum_i \frac{r_i w_{ij}}{\sigma^2},$$

the expression becomes

$$\begin{aligned}\sum_{\{h\}} \exp\left(\sum_j \theta_j h_j\right) &= \prod_j \left(\sum_{h_j=0}^{1} \exp(\theta_j h_j)\right) \\ &= \prod_j \left(1 + \exp(\theta_j)\right).\end{aligned}$$

Substituting back θ_j gives us

$$p(\mathbf{R}) = \exp\left\{-\sum_i^{n_r} \frac{(r_i - a_i)^2}{2\sigma^2}\right\} \prod_j \left[1 + \exp\left\{b_j + \sum_i \frac{r_i w_{ij}}{\sigma^2}\right\}\right].$$

E Minimal Running NQS Script

We here provide a minimal example for a simulation script for running an NQS simulation. The configurations, of course, have to be set up correctly in a yaml file which has to be passed as argument.

```python
import argparse, os, jax, yaml
import numpy as np
from nqs.state import nqs

def initialize_system(config):
    system = nqs.NQS(
        backend=config["backend"],
        logger_level="INFO",
        seed=config["seed"],
    )

    common_kwargs = {
        "correlation": config["correlation"],
        "particle": config["particle"],
        "nhidden": config.get("nhidden"),
        "sigma2": 1.0 / np.sqrt(config["omega"]) if config["nqs_type"] != "dsffn" else None
    }
    config["common_kwargs"] = common_kwargs
    system.set_wf(config["nqs_type"], config["nparticles"], config["dim"], **common_kwargs)
    return system

def run_experiment(config):
    system = initialize_system(config)
    system.set_sampler(
        mcmc_alg=config["mcmc_alg"],
        scale=1 / np.sqrt(config["nparticles"] * config["dim"]),
    )
    system.set_hamiltonian(
        type_="ho",
```

```
        int_type=config["interaction_type"],
        omega=config["omega"],
        r0_reg=10,
        training_cycles=config["training_cycles"],
    )
    system.set_optimizer(
        optimizer=config["optimizer"],
        eta=config["eta"] / np.sqrt(config["nparticles"] * config["dim"]),
    )

    if config["nqs_type"] == "dsffn":
        system.pretrain(
            model="Gaussian",
            max_iter=1000,
            batch_size=2000,
            logger_level="INFO",
            args=config["common_kwargs"],
        )

    history = system.train(
        max_iter=config["training_cycles"],
        batch_size=config["batch_size"],
        early_stop=False,
        history=True,
        tune=False,
        grad_clip=0,
        seed=config["seed"],
    )

    df_samples = system.sample(
        config["nsamples"],
        config["nchains"],
        config["seed"],
        save_positions=config["save_positions"],
    )

def load_config(config_path):
    with open(config_path, "r") as file:
        return yaml.safe_load(file)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Run NQS Experiment")
    parser.add_argument("--config", type=str, required=True, help="Path to the configuration
        file")
    args = parser.parse_args()

    config_path = os.path.join(os.path.dirname(__file__), args.config)
    run_experiment(load_config(config_path))
```

Listing 1: Simplified example of execution code for a NQS experiment

F Additional Results 1D Case

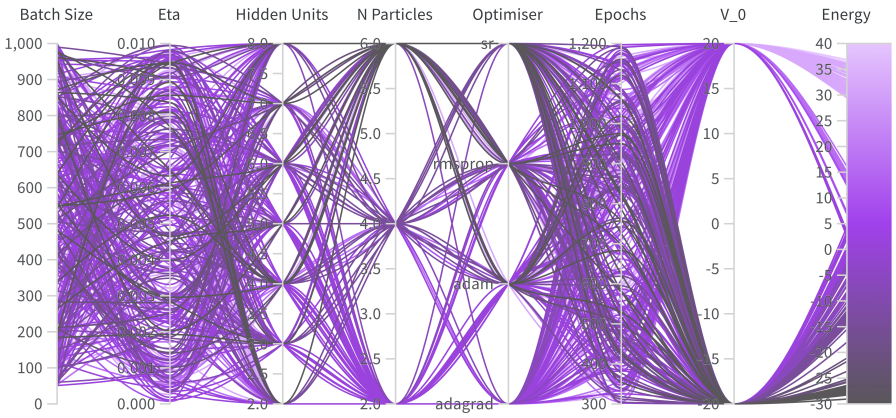


Figure 1: Sweep over 324 hyperparameters for the standard RBM ansatz, from which the results shall be aggregated for the one-dimensional system.

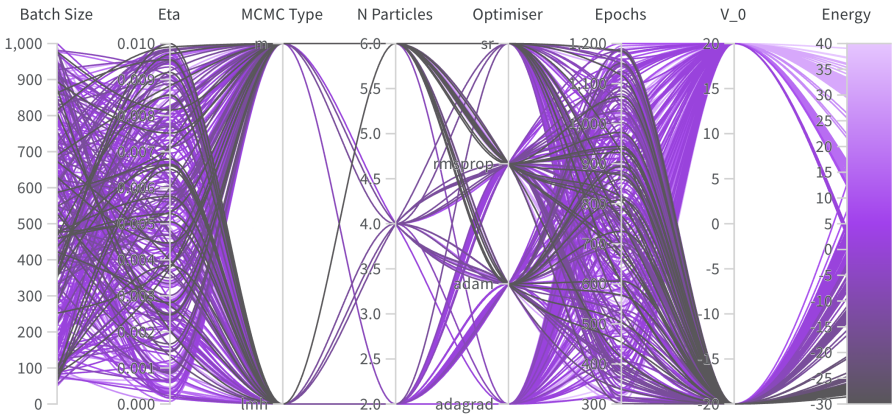


Figure 2: Sweep over 321 hyperparameters for the standard VMC ansatz, from which the results shall be aggregated for the one-dimensional system.

G Additional Results 2D Case

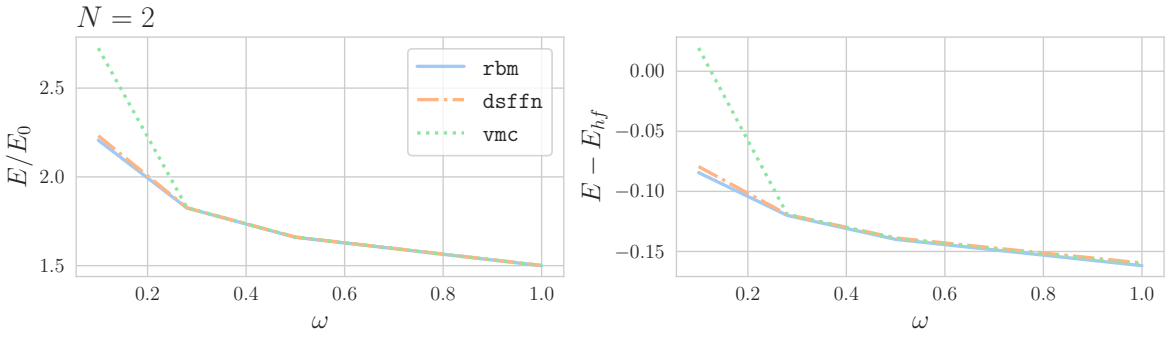


Figure 3: On the left, the energy over non-interactive energy values for two-particle two dimensional quantum dot as a function of frequency of the trap. On the right, the correlation energy with E_{hf} the Hartree-Fock energy.

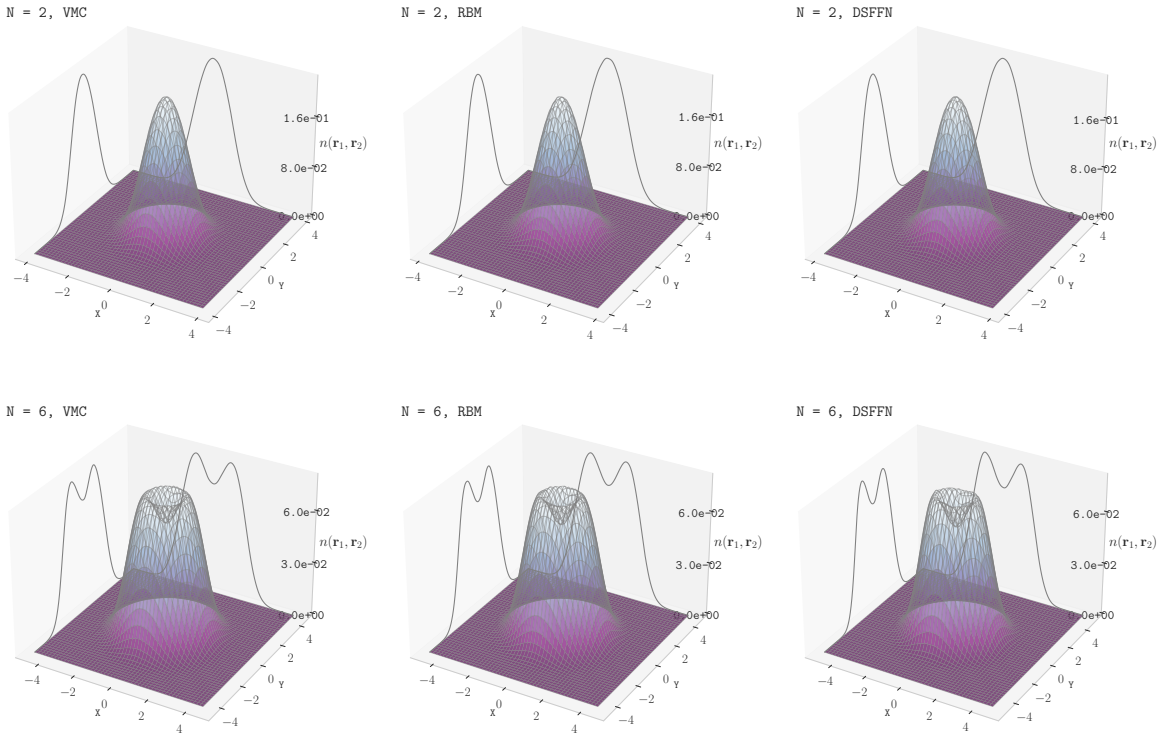


Figure 4: One-body density for the two and six particle case in a trap frequency of $\omega = 1.0$

Bibliography

[1] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251, 1998.

[2] Shun-Ichi Amari and Scott C Douglas. Why natural gradient? 2:1213, 1998.

[3] James B Anderson. A random-walk simulation of the schrödinger equation: H+ 3. *The Journal of Chemical Physics*, 63:1499, 1975.

[4] Vesa Apaja. Many-body physics, 2018. Lecture notes from University of Jyväskylä, Finland. Accessed on May 1, 2024.

[5] Federico Becca and Sandro Sorella. *Quantum Monte Carlo approaches for correlated systems*. Cambridge University Press, 2017.

[6] Kristin P Bennett and Emilio Parrado-Hernández. The interplay of optimization and machine learning research. *The Journal of Machine Learning Research*, 7:1265, 2006.

[7] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[8] Michael Borenstein, Larry V Hedges, Julian PT Higgins, and Hannah R Rothstein. *Introduction to meta-analysis*. John Wiley & Sons, 2021.

[9] Léon Bottou et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

[10] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

[11] Rickard Brüel Gabrielsson. Universal function approximation on graphs. *Advances in neural information processing systems*, 33:19762, 2020.

[12] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355:602, 2017.

[13] Siu A Chin. Quadratic diffusion monte carlo algorithms for solving atomic many-body problems. *Physical Review A*, 42:6991, 1990.

[14] Charles W Curtis. *Linear algebra: an introductory approach*. Springer Science & Business Media, 2012.

[15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2:303, 1989.

[16] Daniel Haas. Codebase for thesis. `https://github.com/Daniel-Haas-B/NeuralQuantumState`, 2024.

[17] Daniel Haas. Documentation page for thesis. `https://neural-quantum-state.readthedocs.io/en/latest/index.html`, 2024.

[18] Dong-Ling Deng, Xiaopeng Li, and S Das Sarma. Quantum entanglement in neural network states. *Physical Review X*, 7(2):021021, 2017.

[19] Ian Dewancker, Michael McCourt, and Scott Clark. Bayesian optimization for machine learning : A practical guidebook, 2016.

[20] M. Drissi, J. W. T Keeble, J. Rozalén Sarmiento, and A. Rios. Second-order optimization strategies for neural network quantum states. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 382:20240057, 2024.

[21] A Eagle. Chance versus randomness. 2014.

[22] Alexander Fleischer. Quantum monte carlo methods for studying quantum dots. Master's thesis, University of Oslo, 2018.

[23] H. Flyvbjerg and H. G. Petersen. Error estimates on averages of correlated data. *The Journal of Chemical Physics*, 91:461, 1989. Publisher: American Institute of Physics.

[24] Cong Fu, Yonghui Zhang, Deng Cai, and Xiang Ren. Atsne: Efficient and robust visualization on gpu through hierarchical optimization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, page 176, 2019.

[25] A. Gelman, W. R. Gilks, and G. O. Roberts. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The Annals of Applied Probability*, 7:110, 1997.

[26] Thierry Giamarchi, Anibal Iucci, and Christophe Berthod. Quantum monte carlo, 2008. Lecture notes from University of Geneva. Accessed on May 1, 2024.

[27] Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac. Neural-network quantum states, string-bond states, and chiral topological states. *Physical Review X*, 8(1):011006, 2018.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[29] Alex Graves. Generating sequences with recurrent neural networks, 2014.

[30] Roger Grosse. Topics in machine learning: Neural net training dynamics. `https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2021/`, 2021. Accessed: 2024-05-09.

[31] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357, 2020.

[32] Aram W Harrow and John C Napp. Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. *Physical Review Letters*, 126:140502, 2021.

[33] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[34] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

[35] Morten Hjorth-Jensen. Advanced topics in computational physics, 2021. Accessed: 2024-05-12.

[36] Jørgen Høgberget. Quantum monte-carlo studies of generalized many-body systems. Master's thesis, 2013.

[37] M Holzmann, DM Ceperley, C Pierleoni, and K Esler. Backflow correlations for the electron gas and metallic hydrogen. *Physical Review E*, 68(4):046707, 2003.

[38] Chien-Jung Huang, Claudia Filippi, and CJ Umrigar. Spin contamination in quantum monte carlo wave functions. *The Journal of chemical physics*, 108:8838, 1998.

[39] Frank Jensen. *Introduction to computational chemistry*. John wiley & sons, 2017.

[40] Joblib Development Team. Joblib: running python functions as pipeline jobs. `https://joblib.readthedocs.io/`, 2020.

[41] Marius Jonsson. Standard error estimation by an automated blocking method. *Physical Review E*, 98:043304, 2018.

[42] Malvin H Kalos, Dominique Levesque, and Loup Verlet. Helium at zero temperature with hard-sphere and other forces. *Physical Review A*, 9:2178, 1974.

[43] Tosio Kato. On the eigenfunctions of many-particle systems in quantum mechanics. *Communications on Pure and Applied Mathematics*, 10:151, 1957.

[44] J. W. T. Keeble, M. Drissi, A. Rojo-Francàs, B. Juliá-Díaz, and A. Rios. Machine learning one-dimensional spinless trapped fermionic systems with neural-network quantum states. *Phyics Review A*, 108:063320, Dec 2023.

[45] Jane Kim, Gabriel Pescia, Bryce Fore, Jannes Nys, Giuseppe Carleo, Stefano Gandolfi, Morten Hjorth-Jensen, and Alessandro Lovato. Neural-network quantum states for ultracold fermi gases, 2023.

[46] Jane Mee Kim. *Solving the Quantum Many-Body Problem with Neural-Network Quantum States*. PhD thesis, Michigan State University, 2023.

[47] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, San Diega, CA, USA, 2015.

[48] Krzysztof C Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90:1, 2001.

[49] Ioan Kosztin, Byron Faber, and Klaus Schulten. Introduction to the diffusion monte carlo method. *American Journal of Physics*, 64:633, 1996.

[50] Dirk P Kroese, Tim Brereton, Thomas Taimre, and Zdravko I Botev. Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6:386, 2014.

[51] Ronen Kroeze, Jom Luiten, and Servaas Kokkelmans. Trapped electrons in the quantum degenerate regime, 2016.

[52] Hannah Lange, Anka Van de Walle, Atiye Abedinnia, and Annabelle Bohrdt. From architectures to applications: A review of neural quantum states, 2024.

[53] Scott Lawrence, Arlee Shelby, and Yukari Yamauchi. Quantum states from normalizing flows, 2024.

[54] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.

[55] Jaewook Lee, Woosuk Sung, and Joo-Ho Choi. Metamodel for efficient estimation of capacity-fade uncertainty in li-ion batteries for electric vehicles. *Energies*, 8:5538, 2015.

[56] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861, 1993.

[57] Jeffmin Lin, Gil Goldshlager, and Lin Lin. Explicitly antisymmetrized neural network layers for variational monte carlo simulation. *Journal of Computational Physics*, 474:111765, 2023.

[58] Sheng-Hsuan Lin. Solving quantum many-body problem with feed-forward neural networks. Master's thesis, Ludwig Maximilian University of Munich, 2018.

[59] Di Luo and Bryan K. Clark. Backflow transformations via neural networks for quantum many-body wave functions. *Physical Review Letters*, 122(22), June 2019.

[60] Alfred Alocias Mariadason. Quantum many-body simulations of double dot system. Master's thesis, University of Oslo, 2018.

[61] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, page 2408. PMLR, 2015.

[62] Sam McArdle, Tyson Jones, Suguru Endo, Ying Li, Simon C Benjamin, and Xiao Yuan. Variational ansatz-based quantum simulation of imaginary time evolution. *npj Quantum Information*, 5:75, 2019.

[63] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21:1087, June 1953.

[64] Izaak Neutelings. Neural networks in tikz. `https://tikz.net/neural_networks/#full_code`, 2021. Accessed: 2024-05-30.

[65] Zihou Ng. Draw restricted boltzmann machines using tikz. `https://gist.github.com/stwind/999544e9002e0aa477653fddf95d4dc5`, 2020. Accessed: 2024-05-27.

[66] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[67] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[68] Emmy Noether. Invariante variationsprobleme. In *Gesammelte Abhandlungen-Collected Papers*, page 231. Springer, 1983.

[69] Even Marius Nordhagen. Studies of quantum dots using machine learning. Master's thesis, University of Oslo, 2019.

[70] Murat Onen, Nicolas Emond, Baoming Wang, Difei Zhang, Frances M Ross, Ju Li, Bilge Yildiz, and Jesús A Del Alamo. Nanosecond protonic programmable resistors for analog deep learning. *Science*, 377:539, 2022.

[71] Abril-Pla Oriol, Andreani Virgile, Carroll Colin, Dong Larry, Fonnesbeck Christopher J., Kochurov Maxim, Kumar Ravin, Lao Jupeng, Luhmann Christian C., Martin Osvaldo A., Osthege Michael, Vieira Ricardo, Wiecki Thomas, and Zinkov Robert. Pymc: A modern and comprehensive probabilistic programming framework in python. *PeerJ Computer Science*, 9:e1516, 2023.

[72] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of physics*, 349:117, 2014.

[73] Gaopei Pan and Zi Yang Meng. *The sign problem in quantum Monte Carlo simulations*, page 879. Elsevier, 2024.

[74] Chae-Yeun Park and Michael J Kastoryano. Geometry of learning neural quantum states. *Physical Review Research*, 2:023232, 2020.

[75] Kun Il Park and M Park. *Fundamentals of probability and stochastic processes with applications to communications*. Springer, 2018.

[76] David Pfau, Simon Axelrod, Halvard Sutterud, Ingrid von Glehn, and James S. Spencer. Natural quantum monte carlo computation of excited states, 2024.

[77] David Pfau, James S Spencer, Alexander GDG Matthews, and W Matthew C Foulkes. Ab initio solution of the many-electron schrödinger equation with deep neural networks. *Physical Review Research*, 2:033429, 2020.

[78] Mohamed Reyad, Amany M Sarhan, and Mohammad Arafa. A modified adam algorithm for deep neural network optimization. *Neural Computing and Applications*, 35:17095, 2023.

[79] Jesus Rogel-Salazar. The gross–pitaevskii equation and bose–einstein condensates. *European Journal of Physics*, 34(2):247, 2013.

[80] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976.

[81] Jun John Sakurai and Jim Napolitano. *Modern quantum mechanics*. Cambridge University Press, 2020.

[82] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I 16*, page 632. Springer, 2006.

[83] Erwin Schrödinger. Die gegenwärtige situation in der quantenmechanik. *Naturwissenschaften*, 23:844, 1935.

[84] Philip Sedgwick. Pearson's correlation coefficient. *British Medical Journal Publishing Group*, 345, 2012.

[85] Or Sharir, Amnon Shashua, and Giuseppe Carleo. Neural tensor contractions and the expressive power of deep neural quantum states. *Physical Review B*, 106(20):205136, 2022.

[86] Isaiah Shavitt and Rodney J. Bartlett. *Many-Body Methods in Chemistry and Physics: MBPT and Coupled-Cluster Theory*. Cambridge Molecular Science. Cambridge University Press, 2009.

[87] Lennart Sobirey, Niclas Luick, Markus Bohlen, Hauke Biss, Henning Moritz, and Thomas Lompe. Observation of superfluidity in a strongly correlated two-dimensional fermi gas. *Science*, 372(6544):844, May 2021.

[88] James Stokes, Josh Izaac, Nathan Killoran, and Giuseppe Carleo. Quantum natural gradient. *Quantum*, 4:269, 2020.

[89] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[90] Attila Szabo and Neil S Ostlund. *Modern quantum chemistry: introduction to advanced electronic structure theory*. Courier Corporation, 2012.

[91] Manuel Valiente. Bose-fermi dualities for arbitrary one-dimensional quantum systems in the universal low-energy regime. *Physical Review A*, 102(5):053304, 2020.

[92] Nicolaas Godfried Van Kampen. *Stochastic processes in physics and chemistry*, volume 1. Elsevier, 1992.

[93] Filippo Vicentini, Damian Hofmann, Attila Szabó, Dian Wu, Christopher Roth, Clemens Giuliani, Gabriel Pescia, Jannes Nys, Vladimir Vargas-Calderón, Nikita Astrakhantsev, et al. Netket 3: Machine learning toolbox for many-body quantum systems. *SciPost Physics Codebases*, page 007, 2022.

[94] Konstantinos D Vogiatzis, Dongxia Ma, Jeppe Olsen, Laura Gagliardi, and Wibe A De Jong. Pushing configuration-interaction to the limit: Towards massively parallel mcscf calculations. *The Journal of Chemical Physics*, 147(18), 2017.

[95] DinhDuy Vu and S. Das Sarma. One-dimensional few-electron effective wigner crystal in quantum and classical regimes. *Physical Review B*, 101(12), March 2020.

[96] Steven R White. Density matrix formulation for quantum renormalization groups. *Physical Review Letters*, 69(19):2863, 1992.

[97] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229, 1992.

[98] Matthias Wittemer, Jan-Philipp Schröder, Frederick Hakelberg, Philip Kiefer, Christian Fey, Ralf Schuetzhold, Ulrich Warring, and Tobias Schaetz. Trapped-ion toolkit for studies of quantum harmonic oscillators under extreme conditions. *Philosophical Transactions of the Royal Society A*, 378(2177):20190230, 2020.

[99] Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy Lillicrap. Logan: Latent optimisation for generative adversarial networks, 2020.

[100] Guanghan Xia. Quantum harmonic oscillators in one and two dimensions. *Highlights in Science, Engineering and Technology*, 64:213, 2023.

[101] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[102] Yushun Zhang, Congliang Chen, Naichen Shi, Ruoyu Sun, and Zhi-Quan Luo. Adam can converge without any modification on update rules. *Advances in neural information processing systems*, 35:28386, 2022.

[103] Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48:787, 2020.

[104] Mo Zhou, Tianyi Liu, Yan Li, Dachao Lin, Enlu Zhou, and Tuo Zhao. Toward understanding the importance of noise in training neural networks. In *International Conference on Machine Learning*, page 7594. PMLR, 2019.