Artificial Neural Networks In Variational Monte Carlo

by

Johan Nereng

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

October 2021

Abstract

In this thesis I explore whether deep neural networks as part of a total trial wave function for a quantum mechanical multi-particle system in a Variational Monte Carlo framework can contribute to better estimates of the ground state energy of the system. I have shown that these trial wave functions outperform the standard approach when applied to a system of two interacting particles in a two- and three-dimensional harmonic oscillator. When applied to correct a pre-trained standard ansatz normally represented by the so-called Padé-Jastrow factor for this system, the neural network approach improved the energy estimates of the standard ansatz by two to three orders of magnitude, depending on oscillator frequency. This is comparable to the performance of Diffusion Monte Carlo. The neural network approach may be a viable alternative to other Quantum Monte Carlo methods as it is easily generalized and requires minimal physical insight into the modelled system. I have also demonstrated that the neural network trial wave function can estimate the ground state energy of a one-dimensional Calogero-Sutherland model for up to fourteen interacting particles with a relative discrepancy $\sim 10^{-3}$ per particle or less.

Acknowledgements

My time at UiO and CCSE has been both trying and rewarding – especially the time spent writing this thesis. I would like to thank my supervisor Morten Hjorth-Jensen for helping and guiding me through many years of courses and now, finally, work on my thesis. Your enthusiasm for teaching and your pedagogical lectures have been a great source of motivation, and were an important reason for why I applied to this master program. Although the last year has been dominated by zoom meetings, I have enjoyed our conversations and learned a lot from you.

I would also like to express my sincere gratitude to Erik Plahte, who has spent countless hours helping me by providing constructive feedback and proofreading several of the chapters. I am not sure I would have been able to complete my thesis without you. The same goes for my close friend Gard Fostad Moe, for his last minute proofreading – you have always been and will always be a close approximation to a brother.

A big thank you is also owed to Even Nordhagen for the many invaluable discussions about both VMC and my results. Your support came at a critical time, and helped me overcome the dreaded imposter syndrome and find the guts to commit to a conclusion.

I also want to acknowledge Bendik Samseth, who was kind enough to help me set up and understand his excellent python library, QFLOW – you are an amazing programmer. The same goes for "BrainStone" from the discord server "Together C & C++", whose guidance enabled me to set up the Compute Engine I used when running my code. And last but not least, Berit and Hildur, for supporting and believing in me.

Johan Nereng

Contents

Chapter 1

Introduction

Since its conception in the beginning of the 20th century, the theory of quantum mechanics has been advanced through the contributions of countless researchers, students, as well as others. There are, however, still challenges to which we have no general solution. These limit the practical use of what we know. Some of these challenges can be studied through approximations and computer simulations, and can often be corroborated with either laboratory experiments or through theoretical study. One such problem is the study of many-body systems, which is relevant in a variety of fields, such as quantum chemistry, nuclear physics, and condensed matter physics [1].

In quantum mechanics, a system is defined by its wave function, which describes its state. When we consider larger systems of many interacting particles, the wave function becomes increasingly complex. Obtaining exact solutions to the many-body Schrödinger equation, which gives the time evolution of the wave function, is generally NP-hard [2]. Even accessing the solution of basic stationary systems is a daunting task for virtually all realistic cases [3]. As such, applications of standard quantum mechanical methods are often either largely inconvenient or even impossible. The ground state of a quantum mechanical system is a stationary state; the one with the lowest energy. This state and its energy is of interest because they tells us how the system behaves at zero temperatures – where quantum effect are usually strongest, and because the behaviour of a system at higher temperatures is often treated as a perturbation to its ground state. Quantum Monte Carlo (QMC) methods offer an alternative for treating the many-body problem when seeking to calculate the ground state energy. QMC is a collection of stochastic computational techniques which are all based on the Monte Carlo method. One such method is Variational Monte Carlo (VMC).

The core idea in VMC is to estimate the ground state energy by numerically integrating samples from the local energy space of an approximation to the system. The variational principle, hence variational Monte Carlo, upon which the method relies [4], maintains that this estimate supplies an upper bound to the ground state energy. The energy samples are obtained by combining a random walker in coordinate space and a *trial wave function*, or clever "guess" on the form of the exact ground state. When this guess, commonly also refereed to as an ansatz, is close to the exact ground state, the energy estimate will tend towards the ground state energy. The trial wave function usually

features variational parameters which can be optimized such that the ansatz becomes a better approximation of the exact ground state. Whether the optimized ansatz is a close match or not depends on its functional form and flexibility. Traditionally, such ansatze advance a specific form based on insights of the physics involved. Such knowledge is however not always available, and the functional form also constitutes a potential hindrance to expressing ansätze which can be generally applied. Incorrect assumptions in this regard can furthermore introduce irreducible systematic errors to the trial wave function.

This thesis examined the application of artificial feed forward neural networks [5] (NNs) in VMC. These neural networks are machine learning methods [6], and consist of layers of interconnected nodes, where each node represents a non-linear transformation of the output of the nodes in the previous layer. Such networks can theoretically approximate a function to arbitrary precision[7] given a sufficient number of nodes. As such, NNs can potentially remedy the issues associated with assumptions about the form of the trial wave function. In this thesis, I combined deep NNs, meaning of many layers, with the exact single particle contribution of a wave function in a total ansatz. By doing so, the network "learns" the particle correlation of the wave function gradually by means of gradient based optimization[8]. The main goal of this thesis was to study if and how neural networks can be applied to both few and many-body systems of interacting particles. It also aims to examine whether NNs are effective at correcting a pre-trained standard ansatz, and how network optimization is affected by certain factors. These factors were choices such as network width and depth, meaning number of nodes in each layer and the number of layers, and sample size per iteration of gradient descent. In order to obtain relevant results, I carried out simulations using the self-contained python library QFLOW [9], which is designed to accommodate neural networks in VMC.

Using neural networks to model particle correlations was shown to outperform the standard VMC ansatz, which is normally represented by the so-called Padé-Jastrow factor[10], when applied to a system of two interacting particles in a two and three dimensional harmonic oscillator. I have also demonstrated that the neural network trial wave function can estimate the ground state energy of a one-dimensional Calogero-Sutherland [11] model for up to fourteen interacting particles with a relative discrepancy $\sim 10^{-3}$ per particle or less. The networks were furthermore used to "correct" a pre-trained standard VMC ansatz for two interacting particles in a harmonic oscillator with various oscillator frequencies. This was observed to improve the energy estimate by two to three orders of magnitude, which is comparable to the performance of Diffusion Monte Carlo [12] (DMC). As DMC is commonly considered to be a highly accurate QMC method, the NN approach may be a viable alternative. Especially because NNs require minimal preparation and knowledge of the physics of the system. This thesis was limited to the two previously mentioned systems, and further study of more complex systems is required to confirm the conclusions. Several papers on neural networks in VMC, some using more sophisticated NNs and examine many-body fermion systems, have recently been published[2, 4, 11, 13], and describe promising results.

1.0.1 Structure of thesis

This thesis is divided into four parts; two dedicated to theory, one to describing the methods, and a final part on implementation, results and conclusions.

In the first part I present a brief overview of some key concepts in basic quantum mechanics, such as what a wave function is and how we define a Hamiltonian. This is done in order to provide a theoretical foundation for the discussion of results, and in order to define the core concepts involved in Variational Monte Carlo (VMC). I furthermore discuss the Born-Oppenheimer approximation and introduce atomic units, which facilitate the formulation of a simple and general expression for the Hamiltonian of a many-body system. Lastly, I briefly present the quantum mechanical systems used when studying and benchmarking the neural networks (NNs).

Next, I provide a similar theoretical basis for the machine learning (ML) methods of the thesis. This includes discussing machine learning in general, as well as specific techniques. Either to motivate later discussion on techniques used in this thesis, or because they are necessary ingredients in my approach, such as gradient based optimization – with which the networks are trained. I then move on to neural networks, and specifically feed forward neural networks. This includes details on of the feed forward process, which generates the network output, and back propagation. The latter is the mathematical procedure for adjusting the weights and biases, which constitute the variational parameters when an NN is used in VMC.

In the third part, I discuss my methods. First, in the chapter on Variational Monte Carlo (VMC), I go through how VMC is applied in this thesis. I then introduce the Monte Carlo method and variational principle. These, together with the sampling algorithms, constitute the algorithmic VMC approach to ground state energy estimation. Next, I dedicate a chapter to discussing the practicalities of defining and applying trial wave functions (TWFs). This begins by discussing optimization of TWFs by defining the cost function through energy minimization. Moving on, I specify the TWF related quantities required to execute a VMC simulation. The next subject is the NN ansatz – here I define and discuss important aspects of the primary ansatz of the thesis. In order to provide an estimate of the uncertainty of the ground state energy estimate, I use the blocking method, introduced in the last section of this chapter.

The fourth, and final part, covers implementation and results. I present, and briefly discuss, my primary tool for executing Variational Monte Carlo (VMC) simulations, the self-contained QFLOW[9] library. This library is specifically designed for NNs in VMC, and has a Python interface with a C++ backend to handle the more computationally intensive calculations. A crucial step between preparation and execution is verifying the integrity of methods in the computer code implementation. This was done by testing that reasonable ground state energy estimates could be obtained, and that the NNs optimized as expected.

I have dedicated the final two chapters to presenting and discussing results relevant to the main goals of this thesis. First, I compare how different architectures, meaning the number of layers and nodes in an NN, optimize when used in an ansatz. I then present and discuss how the single particle contribution in the NN ansatz affects the

final energy estimate, and briefly study the trade-off between exploration and exploitation when training NNs in VMC. In this context, exploit refers to taking advantage of the current model, while explore refers to investigating other solutions. Correlation modelling versus correcting a pre-trained ansatz is then examined. I move on to particle-densities, which I use as a means to discuss correcting the standard VMC approach with an NN. The performance of my methods is then discussed in relation to other QMC and VMC methods, using the ground state energy estimates for two interacting particles in a harmonic oscillator. Lastly, I examine generalization to a one dimensional many-body Calogero-Sutherland model, before summarizing and concluding the thesis.

Part I

Theory: Quantum Mechanics

Chapter 2

Quantum Mechanics

Quantum mechanics describes physics at the very smallest scale in our universe. At this scale, classical physics breaks down, and our intuition often fails us. This first chapter is dedicated to part of the theory relating to the properties of atoms and subatomic particles. The name "quantum mechanics" refers to the fact that many observables, such as energy and momentum, of bound quantum mechanical systems only take discrete values. This phenomenon can be observed through experiments such as the photoelectric effect, documented and published by Einstein in 1905 [14], which showed that light caused electron emission from metals only when the frequency of the light reached some threshold, and not when the intensity of the light was increased. As this thesis is concerned with computational calculation of the ground state energy of non-relativistic quantum mechanical systems, I have chosen to include some selected elementary quantum mechanical topics. These topics are either required or useful to know in order to understand the methods I use in these calculations. Much of the content of this chapter is based on Szabo and Ostlund's book *Modern Quantum Chemistry* [15] and *Introduction to Quantum Mechanics* [16] by Griffiths, and I refer to these books for more details on non-relativistic quantum mechanics.

In the next chapter, I start by introducing and briefly describing the six postulates of quantum mechanics as a stepping stone to the remaining chapter contents. I then move on to the Schrödinger equation and the Hamiltonian, both of which are essential ingredients for the variational Monte Carlo (VMC) methods used in this thesis. Then, to simplify the final computational expressions, I state the electronic Hamiltonian using the Born-Oppenheimer approximation, and introduce atomic units. Lastly, I discuss wave functions and their requirements, which are directly related to the trial wave functions introduced in the chapter on VMC.

2.1 The postulates of quantum mechanics

To what degree does our theory of physics reflect nature? As our understanding of nature hinges on perceiving nature in the first place, our theory is inevitably limited. We use what we perceive to generalize, and in some cases hypothesize, such as was the case for

black holes[1]. However much perception limits our theory, observation is not the only limiting factor. Neither can we directly observe the governing rules of the Universe. A classic example is the theory of relativity, which builds upon two postulates; the principle of relativity and that the speed of light is the same in any frame of reference, no matter the relative motion of the observer to the source of the light. This can be confirmed by experiments, but we cannot necessarily explain why it is so. As such, theory hinges upon some baseline being assumed to be correct, typically based on observation. In the case of quantum physics, this base line consists of the postulates of quantum mechanics [18, 19], which are meant to capture what we know of the world of quantum physics.

First postulate

The state of a quantum mechanical system is completely specified by its wave function, $\Psi(\mathbf{r}, t)$.

The first postulate introduces the concept of a wave function, Ψ, which provides a probability distribution for the observables of the system. An observable is in general a measurable physical quantity, such as momentum, energy, and so forth. In quantum physics, observables are described by operators. The application of an operator corresponds to performing a measurement of the associated observable. These operators work on the wave function and have eigenvalues consistent with all the possible values of the observable. In non-relativistic quantum physics, space and time are considered separate, which is why the wave function is usually expressed as a function of position in space, $\mathbf{r}$, and time, t. The wave function represents the complex single-valued probability amplitude of finding the system in a particular state. This contrasts with the analogous concept in classical physics, where the state of a system is typically taken as its definitive trajectory in space and time. If the wave function is normalized such that

$$\int_{-\infty}^{\infty} d\mathbf{r}\Psi^*(\mathbf{r}, t)\Psi(\mathbf{r}, t) = 1, \tag{2.1}$$

where Ψ^* is the complex conjugate of Ψ, and $\Psi(\mathbf{r}, t)$ is square integrable, the product $\Psi^*(\mathbf{r}, t)\Psi(\mathbf{r}, t)$ yields a probability corresponding to finding the system in $\mathbf{r}$ at time t. The first postulate expresses that if $\Psi(\mathbf{r}, t)$ is specified, so is every possible value of the quantum mechanical observables of the system.

Second postulate

Every quantum mechanical observable has a corresponding linear Hermitian quantum mechanical operator.

[1]Black holes are a theoretical prediction based on the theory of general relativity. The story behind this prediction is described in an entertaining and accessible manner in Stephen Hawking's book *A brief history of time*[17].

As I discussed above, the observables in quantum mechanics are manifested by operators. These operators are linear, meaning that if $\hat{O}$ is such an operator, it obeys

$$\begin{aligned}(\hat{O}_1 + \hat{O}_2)\Psi &= \hat{O}_1\Psi + \hat{O}_2\Psi,\\ \hat{O}(\Psi_1 + \Psi_2) &= \hat{O}\Psi_1 + \hat{O}\Psi_2,\\ \hat{O}\sum c_i\Psi_i &= \sum c_i\hat{O}\Psi_i,\end{aligned}$$

where $c_i \in \mathbb{C}$, and Hermitian. If an operator is Hermitian, it means that $\langle O\rangle^* = \langle O\rangle$ (see 3rd and 4th postulate), which ensures that measurable properties have real ($\mathbb{R}$) values. The quantum mechanical observables and their corresponding operators are listed in table 2.1.

Observable	Symbol	Operator
Total Energy	E	$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(x,y,z)$
Potential Energy	V	$\hat{V} = V(x,y,z)$
Angular momentum	l_x	$\hat{l}_x = -i\hbar\left(y\frac{\partial}{\partial z} - z\frac{\partial}{\partial y}\right)$
Kinetic Energy	T	$\hat{T} = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)$
Position	x	$\hat{x} = x$
	$\mathbf{r}$	$\hat{r} = \hat{x}\mathbf{i} + \hat{y}\mathbf{j} + \hat{z}\mathbf{k}$
Momentum	p_x	$\hat{p}_x = -i\hbar\frac{d}{dx}$
	p	$\hat{p} = -i\hbar\left(\frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z}\right)$

Table (2.1) Quantum mechanical observables and operators for a single particle.

Third postulate

The eigenfunctions [2] *of an operator $\hat{A}$ form a complete and linearly independent set of functions, and a measurement of the observable A will only yield an eigenvalue of $\hat{A}$.*

The third postulate specifies that $\hat{A}\Psi_n = a_n\Psi_n$ where the set $\{a_n\} \in \mathbb{R}$ are the eigenvalues of the eigenfunctions $\{\Psi_n\}$ of $\hat{A}$. This means that if $\Phi = \Psi_i$ and Ψ_i is an eigenfunction of $\hat{A}$, only values of $a \in \{a_i\}$ can be the result of a measurement on Φ. This also also applies if the wave function is a superposition of the eigenfunctions of $\hat{A}$;

$$\Phi = \sum_i^n c_i\Psi_i. \tag{2.2}$$

[2] "*An eigenfunction of an operator $\hat{A}$ is a function f such that the application of $\hat{A}$ on f gives f again, times a constant.*"-[20]

Fourth postulate

For a system with wave function Ψ, the expectation value of the observable corresponding to an operator $\hat{A}$ is given by

$$\langle \hat{A} \rangle = \frac{\int d\mathbf{r} \Psi^* \hat{A} \Psi}{\int d\mathbf{r} \Psi^* \Psi}.$$

The postulate is stated in its most general form; a normalized integral over the entire spatial domain of the system. For a normalized wave function, however, the denominator is unnecessary as it would compute to 1. As in the 3rd postulate assume that $\hat{A}\Psi_n = a_n \Psi_n$, where Ψ_n is an eigenfunction of $\hat{A}$. Assume also that Ψ is normalized. This means that if $\Psi = \Psi_i$, the expectation value of a measurement of A is

$$\langle A \rangle = \int d\mathbf{r} \Psi_i^* \hat{A} \Psi_i = a_i \int d\mathbf{r} \Psi_i^* \Psi_i = a_i, \tag{2.3}$$

which is arguably intuitively appealing because the eigenvalues are the only values which can result from a measurement. So, if the wave function is an eigenfunction of the operator associated with an observable, then there is no uncertainty in that value;

$$\sigma_A = \langle A^2 \rangle - \langle A \rangle^2 = (a_i^2) - (a_i)^2 = 0 \tag{2.4}$$

Note that the 4th postulate relates to *any* quantum mechanical observable.

Fifth postulate

The time-dependent Schrödinger equation

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

governs the time evolution of a wave function.

This equation plays a role analogous to Newton's second law in classical mechanics. However, as this thesis is concerned with stationary states, states that are independent of time, the time-dependent Schrödinger equation is only mentioned here for the sake of completeness.

Sixth postulate

The total wave functions of a fermionic system is required to be antisymmetric under interchange of coordinates.

This sixth postulate has some profound consequences, among them the Pauli Exclusion principle, which will be covered in more detail later. For now, the most important take away is that systems of fermions and systems of bosons must be treated differently.

2.2 The Schrödinger equation and the Hamiltonian

2.2.1 The Schrödinger equation

The non-relativistic time-independent Schrödinger equation,

$$\hat{H}\Psi_n = E_n\Psi_n(\mathbf{x}),$$

binds a system in state n, represented by its wave function $\Psi_n(\mathbf{x})$, $\mathbf{x}$ is left undefined for now and discussed in more detail later, to the corresponding energy, E_n, through the Hamiltonian operator, $\hat{H}$. As opposed to the time-dependent Schrödinger equation, the time-independent equation, which I will refer to as "the Schrödinger equation", yields the energy of stationary states. These states are system states which, unless somehow perturbed, will persist as time evolves, i.e.

$$\frac{\partial}{\partial t}\Psi_n = 0. \tag{2.5}$$

One such state is the ground state of a system, $n = 0$, which is the state with the lowest associated energy. From the first postulate of quantum mechanics, we know that if the wave function of a system is specified, everything else of interest about the system (in that state) is accessible. In the case of the ground state, this enables us to understand the bulk properties of the material or system in question. The ground state is also of interest because that information is a prerequisite for understanding the excited states of a system. In the case of condensates, the ground state is the fundamental state of the system, and in most cases, any system will eventually revert to its ground state. Typically however, the precise wave function for a system is not known, in which case computer aided approximations enter into play. Because of this, the Schrödinger equation is the theoretical starting point for many computational quantum mechanical methods. Before presenting any such methods, and before exploring more about solving the Schrödinger equation itself, a basic walk-through of the most important ingredients involved is in order.

2.2.2 The Hamiltonian operator

The Hamiltonian operator is the total energy operator. Its formulation reflects the energy constraints on a system, such as any external potential acting on the system as well as the interaction between particles. As in classical mechanics, the total energy is the sum of potential and kinetic energy, which is reflected in the operator sum

$$\hat{H} = \hat{T} + \hat{V},$$

where $\hat{T}$ is the kinetic energy operator, and $\hat{V}$ the potential energy operator. This kinetic energy operator may be stated as

$$\hat{T} = \frac{\hat{p}^2}{2m},$$

where $\hat{p}$ is the momentum operator. In one dimension, here the along the x-axis, $\hat{p}_x = -i\hbar d/dx$, and in general terms

$$\hat{p} = -i\hbar\nabla \tag{2.6}$$

$$= -i\hbar\left(\mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}\right). \tag{2.7}$$

In order to concretize the concept of a Hamiltonian, and as a way of motivating the approximations later used when constructing Hamiltonians for the systems I use in this thesis, let us take a look at the Molecular Hamiltonian.

2.2.3 Molecular Hamiltonian

For a system of N electrons and M nuclei, the Hamiltonian consists of five terms, two containing kinetic energy operators and three containing potential energy operators;

$$\hat{H} = \hat{T}_e + \hat{T}_n + \hat{V}_{ee} + \hat{V}_{nn} + \hat{V}_{en}, \tag{2.8}$$

which is known as the known as the molecular Hamiltonian. The kinetic operators [21],

$$\hat{T}_e = \sum_{i=1}^{N} \frac{\hat{p}_i^2}{2m},$$

$$\hat{T}_n = \sum_{A=1}^{M} \frac{\hat{p}_A^2}{2M_A},$$

represent the energy contribution from the electrons and nuclei respectively. Here, m is the electron mass, M_A, the mass of nucleus A, and $\hat{p}_i = -i\hbar\hat{\nabla}_i$ the momentum operator for particle i. The first potential in the molecular Hamiltonian is the Coulomb repulsion between electrons;

$$\hat{V}_{ee} = \sum_{i}^{N}\sum_{j>i}^{N} \frac{1}{4\pi\epsilon_0}\frac{e^2}{r_{ij}},$$

where ϵ_0 is the electric constant, e the elementary charge, and $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ denotes the distance between electron i and j. Similarly,

$$\hat{V}_{nn} = \sum_{A}^{M}\sum_{B>A}^{N} \frac{1}{4\pi\epsilon_0}\frac{Z_A Z_B e^2}{R_{AB}}$$

represents the Coulomb repulsion between nuclei, where Z_A is the charge of nucleus A, and $R_{AB} = |\mathbf{R}_A - \mathbf{R}_B|$ the distance between nuclei A and B. Lastly,

$$\hat{V}_{en} = -\sum_{i}^{N}\sum_{A}^{M} \frac{1}{4\pi\epsilon_0}\frac{Z_A e^2}{r_{iA}},$$

is the Coloumb attraction between electrons and nuclei ($\hat{V}_{ee}$), where $r_{iB} = |\mathbf{r}_i - \mathbf{R}_B|$.

2.2.4 The electronic Hamiltonian

The Born-Oppenheimer approximation

The molecular Hamiltonian (2.8) is arguably rather cumbersome, and in many cases overly detailed. As such, consider a proton with mass M_p and an electron with mass m_e. The ratio of masses, $\mu = M_p/m_e \approx 1836$ [22] show that the the proton is *much* heavier than the electron (and neutrons are even heavier). Accordingly, electrons move much faster than nuclei. Because of this, Max Born and J. Robert Oppenheimer suggested in 1927 that nuclei and electrons could be treated separately. In what is known as the Born-Oppenheimer approximation [15][p.43], the nuclei are considered to be stationary. As a consequence, the kinetic energy of the nuclei is zero, such that $\hat{T}_n$ can be omitted. Additionally, the Coulomb repulsion energy between the nuclei becomes constant as the distances between nuclei are invariant. As constants added to operator eigenvalues has no effect on operator eigenfunctions, $\hat{V}_{nn}$ can also be omitted. Thus, the *electronic Hamiltonian* may be written as

$$\hat{H} = \hat{T}_e + \hat{V}_{en} + \hat{V}_{ee}. \tag{2.9}$$

Finally, in order to generalize this Hamiltonian to represent general N electron systems, the potential energy operator representing the Coulomb attraction between electrons and nuclei is replaced with an unspecified external potential V_{ext};

$$\hat{H} = \hat{T}_e + \hat{V}_{ext} + \hat{V}_{ee}. \tag{2.10}$$

Atomic units, the Hydrogen atom

The hydrogen atom, ^{1}H, consists of a single proton with mass M_p and a single electron with mass m_e. As a consequence, the repulsive electron-electron potential in the electronic Hamiltonian (2.10) is zero. This means that the Hamiltonian for the Hydrogen atom may be written as

$$\hat{H} = \hat{T}_e + \hat{V}_{en}, \tag{2.11}$$

where we defined $\hat{T}_e$ and $\hat{V}_{en}$ the last section. We here introduce the dimensionless variables $x, y, z = \lambda x', \lambda y', \lambda z'$, and chose λ such that

$$-\frac{\hbar^2}{2m_e}\hat{\nabla}^2 - \frac{e^2}{4\pi\epsilon_0 r} = -\frac{1}{2}\hat{\nabla}^2 - \frac{1}{r'}, \tag{2.12}$$

meaning that the Schrödinger equation, (2.2.1), can be written as

$$\left[-\frac{1}{2}\hat{\nabla}^2 - \frac{1}{r'}\right]\Psi = E\Psi, \tag{2.13}$$

which is achieved when

$$\lambda = \frac{4\pi\epsilon_0\hbar^2}{m_e e^2} = a_0. \tag{2.14}$$

This value[22], $a_0 = 5.29177210903 \times 10^{-11}$ m, is the Bohr radius, with the value of one Bohr – the atomic unit of length. It follows that

$$\frac{\hbar^2}{m_e \lambda^2} = \frac{e^2}{4\pi\epsilon_0 \lambda} = E_h, \tag{2.15}$$

where E_h is the atomic unit of energy, the *Hartree*. The Coulomb repulsion term, $\frac{1}{r'}$, in (2.13) can in this case be regarded as an external potential acting on the electron. By using the Born-Oppenheimer approximation to remove all degrees of freedom in the system not related to interaction between the electrons, a similarly simplified but generalized Hamiltonian can applied to a wide range of systems. The Hamilitonan for a system of N particles with two-body interactions (using atomic units $\hbar = c = e = m_e = 1$) is then

$$H = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + V_{ext}(\boldsymbol{r}_i) \right) + \sum_{i<j}^{N} V_{int}(\boldsymbol{r}_i, \boldsymbol{r}_j), \tag{2.16}$$

where V_{ext} is an external potential upon the system and V_{int} is the internal potential arising from particle interactions, and where ∇_k is the vector differential operator acting on particle k, and $\boldsymbol{r}_k$ the coordinates of particle k.

2.3 Wave functions

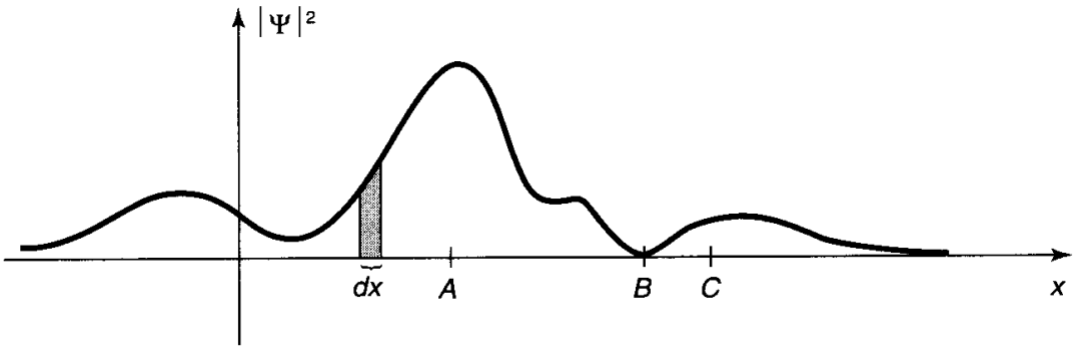


Figure (2.1) An example of the probability distribution of a normalized (single particle) wave function in one dimension. The particle is more likely to be found near A than at B. The shaded area represents the probability of finding the particle in the interval dx. From [16].

So far, I have given a brief description of the core equation in this thesis, the Schrödinger equation, expanded upon the idea of the Hamiltonian, and through approximation obtained a relatively manageable general expression, the electronic Hamiltonian, which applies to wide range of systems. Lastly, I have introduced atomic units which further simplify the Hamiltonian. Throughout all of this, however, I have sidestepped the previously mentioned concept of wave functions. The computational quantum mechanical

methods used in this thesis apply specialized wave function substitutes, meaning that in-depth theoretical knowledge on wave functions is strictly not needed. As such, I will only outline a few related key concepts, and I will save exact wave function formulations for later.

2.3.1 Requirements of Wave Functions

From the postulates of quantum mechanics on can infer some requirements that a wave function must satisfy. Starting with the first postulate, we know that the wave function, Ψ, represents the system state and that it corresponds to a probability magnitude. We also know that $\Psi^*\Psi$ yields the probability distribution of the system state. This means that the wave function must be normalizable, as the total probability must be 1. As such, it must also be a continuous function of space in order to be defined in the entirety of its spatial domain. Furthermore, the wave function must be a solution to the Schrödinger equation and must have continuous first order spatial derivatives (fifth postulate). This basically boils down to having a well behaved and normalizable function. The sixth postulate states that the wave function must fulfil symmetry requirements for identical and indistinguishable particles. Lastly, the Coulomb potential between charged particles present in the electronic Hamiltonian may introduce a singularity when $r_{ij} \to 0$, meaning that the wave function should also satisfy the so called cusp condition[23] which addresses this singularity.

2.3.2 Quantum numbers and orbitals

An orbital is a wave function for a single particle, or a single particle wave function (SPF), expressing the probability of finding the particle in the position with coordinates $\boldsymbol{r} = (x, y, z)$. Orbitals are typically found centered on each atom (atomic orbitals) or can be obtained from a Hartree-Fock procedure (molecular orbitals). An orbital is *occupied* if a particle exists within the orbital, while it is *virtual* if there is no particle in the orbital. In atomic and molecular physics, orbitals are typically described by four quantum numbers, n, l, m_l and m_s, which reflect degrees of freedom. The first number is the principal quantum number, which describes the size of the orbital [24]. It can take on the integer values $1, 2, 3, \cdots$. The second quantum number, $l = 0, 1, 2, ..., n-1$ denotes the angular momentum in an orbital, and specifies its shape. The magnetic quantum number $m_l = -l, ..., 0, ..l$ specifies spatial orientation, and the last quantum number, m_s, is the spin number. All known particles with integer spin are bosons, while particles with half integer spin are fermions. An orbital that does not specifying the spin of a particle is a *spatial orbital*, $\psi(\mathbf{r})$. I will largely be using ψ to denote single particle wave functions, and Ψ to denote total wave functions representing a system.

Spin orbitals

Orbitals for fermionic particles also require a description of particle spin in order to be completely specified. Although this thesis is mainly concerned with systems of bosons,

the methods I use can also be applied to systems of fermions. As such, I have chosen to also include some theory on fermions in this chapter. Informally, the spin of a fermion is said to be either *spin up* $\uparrow$ ($m_s = +1/2$), or to be *spin down* $\downarrow$ ($m_s = -1/2$). Each spatial orbital forms two spin orbitals, one for spin up, and one for spin down;

$$\chi(\mathbf{X}_1) = \begin{cases} = \psi(\mathbf{r}) \uparrow \\ = \psi(\mathbf{r}) \downarrow, \end{cases}$$

where $\mathbf{X}_1 = (\mathbf{r}_1, m_s)$, such that when $\Psi(X_1) = \chi(\mathbf{X}_1)$, The product $|\Psi(\mathbf{X}_1)|^2 d\mathbf{r}$ represents the probability of finding a particle with spin m_s in a volume element $d\mathbf{r}$ surrounding $\mathbf{r}$. As there are two spin functions, K spatial orbitals can be used to form $2K$ different spin orbitals.

2.3.3 Hartree products

Having discussed wave functions for single particles (orbitals), wave functions for systems containing multiple particles is the next natural step. Perhaps the most common and most intuitive method for constructing multi-particle wave functions is using Hartree products. Consider a system of N non-interacting electrons ($\hat{V}_{ee} = 0$). This system is purely artificial, but will nevertheless motivate how to build wave functions for other systems. The Hamiltonian would then be [15][p.48]

$$\hat{H} = \sum_i^N \hat{h}(i), \tag{2.17}$$

where $\hat{h}(i)$ is the Hamiltonian operator of electron i, describing its kinetic energy and potential energy. This means that the set of eigenfunctions and eigenvalues associated with the operator $\hat{h}(i)$ is a set of spin orbitals χ_j and orbital energies ϵ_j, such that

$$h(i)\chi_j(X_i) = \epsilon_j \chi_j(X_i). \tag{2.18}$$

Thus, a possible total trial wave function for the system is a product of these eigenfunctions of $h(i)$;

$$\Psi(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N) = \chi_i(\mathbf{X}_1)\chi_j(\mathbf{X}_2)\ldots\chi_k(\mathbf{X}_N) \tag{2.19}$$

such that

$$\hat{H}\Psi(\mathbf{X}) = E\Psi(\mathbf{X}), \tag{2.20}$$

where $E = \epsilon_i + \epsilon_j + ... + \epsilon_k$, is the eigenvalue. A Hartree product is in other words a total wave function from a product of spin orbitals and an eigenfunction of $\hat{H} = \sum_i^N \hat{h}(i)$. This total wave function represents a system of independent, uncorrelated[3] and *distinguishable* electrons. It specifies which electron is in which orbital. The probability of the system being in a specific state is simply equal to the product of the probabilities of each particle being in the corresponding, specified position. For a system of electrons this is problematic, as electrons in reality are *indistinguishable* – it is not possible to know which electron occupies which orbital.

[3]Uncorrelated in the sense that the motion of one electron has no effect on the motion of another electron

2.3.4 Symmetry requirements

There are two main categories of identical particles, bosons (integer spin) and fermions (half integer spin). All particles with identical internal quantum numbers, as mass, charge and spin, are identical. This means that the particles cannot be meaningfully distinguished, and the wave function must reflect this. In essence, what this boils down to is how the wave function behaves if two coordinates are exchanged. Let $\hat{P}_{ij} = \hat{P}_{ji}$ be an operator (the permutation operator) which interchanges particles i and j, s.t. $\hat{P}_{ij}\hat{P}_{ij} = I$ is the identity operator. Applying the operator to a wave function representing a system with particles i and j results in exchanging the coordinates $\mathbf{X}_i = (\mathbf{r}_i, m_{si})$ with those of $\mathbf{X}_j = (\mathbf{r}_j, m_{sj})$ and vice versa [25]. Let $\Psi(\mathbf{X}_1, ..., \mathbf{X}_i, ..., \mathbf{X}_j, ..., \mathbf{X}_N)$ be an eigenfunction of $\hat{P}_{ij}$. Then,

$$\hat{P}_{ij}\Psi(\mathbf{X}_1, ..., \mathbf{X}_i, ..., \mathbf{X}_j, ..., \mathbf{X}_N) = p\Psi(\mathbf{X}_1, ..., \mathbf{X}_j, ..., \mathbf{X}_i, ..., \mathbf{X}_N), \tag{2.21}$$

where p is an eigenvalue of the $\hat{P}_{ij}$. Applying the operator again, we get

$$\hat{P}_{ij}\hat{P}_{ij}\Psi(\mathbf{X}_1, ..., \mathbf{X}_i, ..., \mathbf{X}_j, ..., \mathbf{X}_N) = p^2\Psi(\mathbf{X}_1, ..., \mathbf{X}_i, ..., \mathbf{X}_j, ..., \mathbf{X}_N), \tag{2.22}$$

implying that $p = 1 \vee -1$. If $p = 1$ the wave function is symmetric, and if $p = -1$, the wave function is antisymmetric. Particles that can be described by symmetric wave functions are bosons, while particles that can be described by antisymmetric wave functions are fermions. Now, assume a system contains two particles with identical coordinates, ie. $X_i = X_j$. Applying the permutation operator then results in

$$\begin{aligned}\Psi(\mathbf{X}_1, ..., \mathbf{X}_i, ..., \mathbf{X}_j, ..., \mathbf{X}_N) &= \Psi(\mathbf{X}_1, ..., \mathbf{X}_j, ..., \mathbf{X}_i, ..., \mathbf{X}_N) \\ &= p\Psi(\mathbf{X}_1, ..., \mathbf{X}_i, ..., \mathbf{X}_j, ..., \mathbf{X}_N),\end{aligned}$$

which means that p must be equal to 1 as the wave function itself is unchanged by the operator. This result is known as the Pauli principle, which states that two fermions may never occupy the same single-particle state. Looking back to the Hartree product, it is clear that this method of obtaining a system description does not satisfy the antisymmetry principle. So, the particle species have different symmetry requirements, which any total wave function representing a system of one of these species must fulfil. When using variational Monte Carlo, trial wave functions (which are the subject of a later chapter) are typically used to represent these systems. In the case of fermionic particles, a trial wave function is often constructed using a Slater determinant in order to meet the requirement of anti-symmetry. For bosonic particles, most trial wave function fulfil the symmetry requirement, but special care is in some cases needed. This is discussed in more detail in the trial wave function chapter.

2.3.5 Cusp Condition

The Coulomb potential (2.2.3) between charged particles, $\sim 1/r_{ij}$, may cause a singularity i.e. a state in which small changes will have large effects, as the distance between

particles i and j, $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j| \to 0$. Typically, this singularity can be cancelled by modifying the wave function such that parts of the kinetic energy term is equal and opposite (for a simplified example using an electron close to a nucleus, see [26]). In the case of two particles of mass m_i and m_j with charges q_i and q_j this results in the requirement [27]

$$\left.\frac{\partial \hat{\Psi}}{\partial r_{ij}}\right|_{r_{ij}=0} = \mu_{ij} q_i q_j \Psi(r_{ij} = 0), \tag{2.23}$$

where $\mu_{ij} = m_i m_j/(m_i + m_j)$, and $\hat{\Psi}$ is the average of Ψ around $r_{ij} = 0$. A solution, as demonstrated in [26], is to formulate the wave function such that $\Psi = \exp(-cr_{ij})$ (which is cusp shaped) when r_{ij} is small, where c depends on the particle composition. The "cusp condition" is also called Kato's cusp condition, after Tosio Kato [23] who first formulated the requirement. Interaction between particles, such as the Coloumb potential between electrons, is typically called particle correlation, which is addressed further in the methods chapter.

2.4 Quantum Mechanics in this thesis

I have so far introduced some of the key concepts in quantum mechanics, but still barely scratched the surface of the field. Before moving on to machine learning, I believe it is prudent to summarize and to some extent contextualize the most important ideas from the previous chapters. To represent a system of identical particles we use a total wave function, which corresponds to the probability of finding the system in a particular state. I have introduced the requirements such a wave function must fulfil, and showed how a tentative total wave function may be constructed from single particle wave functions. By applying atomic units and the Born-Oppenheimer approximation to the electronic Hamiltonian, I discussed how to express a simplified and generalizable Hamiltonian for a system of N particles as

$$H = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + V_{ext}(\boldsymbol{r}_i) \right) + \sum_{i<j} V_{int}(\boldsymbol{r}_i, \boldsymbol{r}_j). \tag{2.24}$$

In the next chapter I will use (2.24) to represent different systems of bosonic particles by specifying an external V_{ext} and internal V_{int} potential. Ultimately, the goal is to use the total wave function(s) and Hamiltonian(s) to estimate the ground state energy, E_0, of these systems. This is explained in more detail later, but hinges on using the Schrödinger equation (2.2.1), and a series of approximations.

Chapter 3

Systems

In this thesis I focus on bosonic particles in three related systems; an ideal harmonic oscillator (HO), a harmonic oscillator with Coulomb interaction (HO+C), and the Calogero–Sutherland model[11] in one dimension (CS). All of these are related by similar external potentials but individually defined by different internal potentials. While the ideal HO is a system with an internal potential of zero, HO+C and CS include different internal potentials which accounts for two-body correlation. Thus, each of the system can be described through the previously introduced generalized and simplified Hamiltonian for N particles with two-body interactions

$$H = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + V_{ext}(\boldsymbol{r}_i) \right) + \sum_{i<j} V_{int}(\boldsymbol{r}_i, \boldsymbol{r}_j), \tag{3.1}$$

by V_{ext} and V_{int}. In this chapter I give a brief definition of each of the three systems, relating them to (3.1) by defining the potentials for each system. I chose these particular systems because the exact ground states are known, making them invaluable for validation and examination of methods. As the systems are closely related, comparison between them may also provide further insight into the the characteristics of feed forward neural networks in variational Monte Carlo.

3.1 Harmonic Oscillator

The isotropic (identical in all directions) harmonic oscillator (HO) potential can be described by

$$V_{ext}(\mathbf{r}_i) = \frac{1}{2}\omega^2 r_i^2, \tag{3.2}$$

where ω is the oscillator frequency, representing the strength of the confinement. Higher values of ω consequently narrows the 'trap', while lower values results in a larger spatial field. Using natural Hartree atomic units $\hbar = c = e = m_e = 1$ and (3.1), the total Hamiltonian of a system with N particles in a harmonic oscillator potential with $V_{int} = 0$ is

$$H = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2 \right). \tag{3.3}$$

In one dimension, the stationary single particle states associated with the principle quantum number n, can be written as [16][p.37-41] [1]

$$\psi_n(x) = N_n(\omega) e^{-\frac{\omega}{2}x^2} H_n(\omega x), \tag{3.4}$$

where N_n[2] is a normalization constant and $H_n(\omega x)$ is the n'th Hermite polynomial [3]. The quantum number n indicates the n'th excited energy level of the system, and is associated with the energy in that dimension by

$$E_n = (n + \frac{1}{2})\omega, \tag{3.5}$$

measured in atomic units. Before moving on, let's contextualize this. This thesis is focused around the variational principle and methods associated with it, both of which will be discussed in more detail later. These methods work towards estimating the lowest energy level a system can inhabit for a given symmetry. This means that we are mainly interested in the ground energy, E_0, and the ground state. The (single particle) ground state, $n = n_x = 0$, for the ideal harmonic oscillator in one dimension can be expressed as

$$\psi_0(x) = N_0(\omega) e^{-\frac{\omega}{2}x^2} H_0(\omega x) = \left(\frac{\omega}{\pi}\right)^{1/4} e^{-\frac{\omega}{2}x^2}, \tag{3.6}$$

with ground energy

$$E_0 = \frac{1}{2}\omega. \tag{3.7}$$

This easily generalizes to multi-dimensional systems through principle quantum numbers for each dimension; $n = n_x + n_y + n_z$. The multi-dimensional single particle wave function is simply the product of each dimension, while the energy is the sum of energies from all the dimensions;

$$E_{n_x,n_y,n_z} = (n_x + n_y + n_z + \frac{3}{2})\omega. \tag{3.8}$$

This means that the ground energy for a D dimensional harmonic oscillator is

$$E_0 = \frac{D}{2}\omega. \tag{3.9}$$

Consequently, a system of N particles will simply have N contributions to the total energy when ignoring particle interaction, meaning that the total energy is

$$E_0 = N\frac{D}{2}\omega. \tag{3.10}$$

[1] The Hamiltonians I use in this thesis are spin independent, and as such I will for the most part not include spin in orbitals or wave functions. Spin can however trivially be appended to the systems if required.

[2] The normalization constant, $N_n = (\frac{m\omega}{\pi\hbar})^{1/4}\frac{1}{\sqrt{2^n n!}}$, and simplifies to $N_n = (\frac{\omega}{\pi})^{1/4}\frac{1}{\sqrt{2^n n!}}$ when using atomic units.

[3] The first three Hermite polynomials, $H_n(x)$, are $H_0 = 1$, $H_1 = 2x$, $H_2 = 4x^2 - 2$.

3.2 HO with Coulomb interaction

In the section above the internal potential, V_{int}, was zero. Let us now instead consider a system of bosons with charge e, confined in a Harmonic Oscillator and interacting according to the Coulomb potential, which I introduced in the section on the Molecular Hamiltonian. For N charged particles, using atomic units, the potential can be expressed by

$$V_{int} = \sum_{i<j} \frac{1}{r_{ij}},$$

where $i = 1, 2..., N$. By including (3.2) in (3.1), the total energy of the system as well as the spatial distribution of particles will be altered. For two particles, M. Taut [28, 29] provides exact values for E_0 in two and three dimensions.

3.3 The one-dimensional Calogero-Sutherland model

The Calogero-Sutherland model (CS)[11] describes a system of bosons trapped in a harmonic oscillator potential, similar to the one discussed earlier, and interacting according to an inverse squared potential. Using atomic units, the internal potential is expressed as

$$V_{int} = \sum_{i<j} \frac{\beta(\beta-1)}{(x_i - x_j)^2}, \tag{3.11}$$

where x_k is the coordinate in one dimension of particle k, and β a parameter governing the interaction. As was done in [11], I use $\beta = 2$. For N bosons in one dimensions, the symmetric exact ground state is

$$\Psi_{exact} = \exp\left(-\frac{1}{2}\sum_{i=1}^{N} x_i^2\right) \prod_{j<k} |x_j - x_k|^{\beta}, \tag{3.12}$$

with associated ground energy

$$E_0 = \frac{N}{2} + \frac{\beta}{2} N(N-1). \tag{3.13}$$

Part II

Theory: Machine Learning

Chapter 4

Supervised learning

The concept of algorithms, which are finite sequences of operations, pre-dates electronics by hundreds, and possibly thousands, of years. The name itself, which comes from the latinization of *Muhammad ibn Mūsa al-Khwarizmī*, a 9th century Persian mathematician, astronomer and geographer [30, 31] is testament to its roots in history. Today, however, the word algorithm if often associated with computers. Solving algorithms is in many ways the foundation of modern data science, where increasingly complex and computationally expensive sequences can be carried out with improved accuracy and speed as the scientific field moves forwards and as the available computational power increases. Whereas many algorithms can be written out in exact detail before being executed, some algorithms include degrees of freedom which leaves room for optimization. *Machine learning* (ML) is the branch of data science which focuses on self improving algorithms – algorithms than either over time, or when supplied with more data, become better by tuning one or more degree of freedom.

This chapter is concerned with the fundamentals of machine learning. First, I introduce terminology related to splitting training data into subsets and the three different machine learning algorithm classes; supervised learning, unsupervised learning and reinforcement learning. Then I use basic regression analysis to motivate and introduce concepts related to all (or many) machine learning methods, such as design matrices and cost functions. I move on to the bias-variance trade-off, which describes two opposing error sources in machine learning, before covering shrinkage methods, which are useful for asserting some control over the bias-variance trade-off. As a natural stage between polynomial regression and the upcoming chapter on artificial neural networks, I discuss qualitative modelling and logistic regression, before ending this chapter with a section on how to optimize machine learning algorithms using gradient based optimization.

4.1 Machine learning algorithms

Machine learning algorithms are used to construct models of some relationship between variables. When we say that an algorithm improves with time or data, what actually

improves, is the quality of the resulting model. In order for the algorithm to learn[1] the relationship between variables, the algorithm needs exposure to a set of data which includes the variables of interest. The set of data used when optimizing, or training, the model is called the *training set*. Independent variables included in the training set are often refereed to as *predictors*, while dependent variables are called *targets*. During training, some degrees of freedom in the algorithm are tuned, based on the predicted output from the training set, for example how the model output compares to corresponding targets. After "training the model", which in all reality means optimizing the algorithm, an unbiased evaluation of the model is made by using it on a hitherto withheld partition of the available data, called the *test set*. The predictors in the test are fed as input to the model, and the resulting model output is typically compared to the test set targets, using some evaluation metric (more on this later). Depending on the model choice, the available data is sometimes partitioned into three, instead of two. The third set, called a *validation set*, is used to tune the hyper-parameters associated with the model.

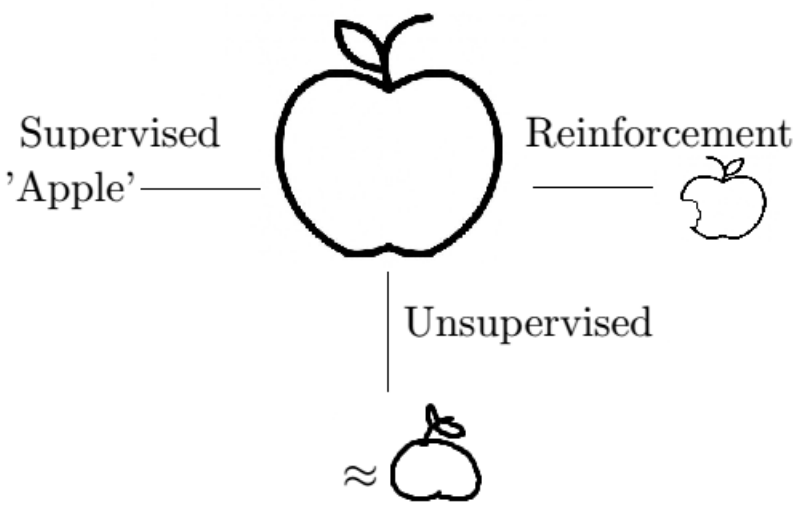


Figure (4.1) The 'Apple example'. Illustrates the type of output that supervised learning (a label), unsupervised learning (a pattern) and reinforcement learning (favourable interaction) could produce from the sketch of an apple.

Machine learning algorithms are usually divided into three classes; supervised learning, unsupervised learning, and reinforcement learning. To help exemplify each class I will use the "apple example" borrowed from the MIT course "Introduction to Deep Learning" [32] [lect 5.], illustrated in figure 4.1, where the training data is a sketch of an apple. Supervised learning algorithm are aimed towards specific model outputs based on variable correlation, for example labels of fruit types, such as 'apple' or 'pear', for a data set of fruit sketches. The term "supervised" fits well, because the training is indeed supervised through well defined targets and quantifiable model error. The characteristics of the dependent variable(s) are reflected in the output of the model, which is either continuous or discrete. Typically, models with discrete outputs are called classifiers.

[1] In this context, "to learn" really means inferring a function.

This means that the possible values of the dependent variable can be labelled – such as types of fruits. A concrete example of continuous modelling using supervised learning (regression analysis) is shown later in this chapter.

In contrast to supervised learning algorithms, unsupervised ones do not use targets. These algorithms aim instead to create a model which maps the independent variables, meaning that it trains only on predictors. This could mean quantifying how similar the apple sketch is to other fruit sketches in the training set. This could mean that the model would associate the sketch of a pear with the sketch of an apple without considering anything else than the structure of the data. Popular unsupervised learning algorithms include techniques such as clustering [6] [p.10] and dimensionality reduction [6] [p.11].

The last class, reinforcement learning algorithms (RL), is concerned with targets, but in a way different from supervised learning. The resulting model does not involve specific target values, and instead only infer the quality of an output. This is done by evaluating the accumulated output over many steps, and optimizing such that it favours steps that eventually leads to a desirable outcome. Simply put, this could be to learn to "eat" the apple, and thereby avoid starvation. More realistically, this could involve training autonomous cars or AI in computer games, such that they avoid collision and reach their the destination, or win the computer game.

Later in this thesis, I will introduce the specific machine learning algorithms I use to produce my results by computer simulations. As these models generate the data while the model is optimized, they can be considered a mixture of supervised learning and reinforcement learning. The approach to optimization in variational Monte Carlo is however mostly consistent with supervised learning, which is why I mainly focus on this class of algorithms. In the following chapter I introduce and use the supervised learning method polynomial regression to exemplify the process of employing machine learning algorithms to produce data models. Whilst doing so, I also bring into acquaintance important concepts and terminology as they arise, such as linear algebra in ML, shrinkage methods and cost functions. Although the regression methods themselves do not play a pivotal role in this thesis, they are well suited as a stepping stone into neural networks as the two share many common aspects. I then discuss the trade-off between bias and variance, shrinkage methods (ways to control bias and variance), qualitative modelling, and lastly how one can optimize a machine learning model using gradient based optimization.

4.2 Regression analysis

Regression analysis is a collection of methods that aim to estimate the relation between independent and dependent variables. Choosing a specific regression model is usually motivated by either insight into the origin of the data and/or by studying the data itself. For instance, if the data in in the training set appears to have an approximately linear relationship, linear regression can be used to model the targets. Through optimizing the model using that data set, it will hopefully generalize well to other data from the same

population. For a set of observations $\{x_i, z_i\}_{i=1}^n$, the linear response, $\tilde{z}_i$, can be modelled by

$$\tilde{z}_i + \epsilon_i = \beta_0 + \beta_1 x_i + \epsilon_i = z_i, \tag{4.1}$$

where x_i is the i'th observation of the independent variable, z_i the associated target, β_0, β_1 are real valued coefficients, and ϵ_i is the residual error between the i'th response and target. This means that the difference between the modelled response for observation i, $\tilde{z}_i$, and the target, z_i, is the error, ϵ_i of the model for this observation. If the relationship between predictors and targets in the data set is indeed approximately linear, there should exists a β_0 and β_1 such that ϵ_j is small for all observations j. Linear regression is a first order model, but the principle can easily can be extended to higher orders, in which case it's called polynomial regression (modelling by power law). A polynomial approximation of order $m-1$ can be expressed as

$$\tilde{z} + \epsilon_j = \sum_{j=1}^{m} \beta_j x^{j-1} + \epsilon_j = z_j. \tag{4.2}$$

How close $\tilde{z}$ is to z, or how small the resulting ϵ is, depends on the values of the coefficients, β_j. A data set of n data points, $\{x_i, z_i\}_{i=1}^n$, typically represents an overdetermined system ($n > m$). In such cases, one tries to find values for the β coefficients, such $\tilde{z_1}, \tilde{z_2}, .., \tilde{z_n}$ fits $z_1, z_2, ..., z_n$ well. How "well" is typically measured will be addressed later. In order to deal with the system of n equations for a $m-1$ degree polynomial regression, it is common practice to bring the system of equations to a vector-matrix form, which increases efficiency and readability. The n values of x are then arranged in an $n \times m$ *design matrix* with elements $X_{i,j} = x_i^{j-1}$ for $i = 1, 2, ..., n$ and $j = 1, 2, ..., m$;

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{m-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^{m-1} \end{bmatrix}. \tag{4.3}$$

The design matrix typically reflects the model choice, and in this case the geometric progression makes it a so called *Vandermonde Matrix* [33][p. 147-148]. By using $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_m]^T$, $\boldsymbol{z} = [z_1, z_2, ..., z_n]^T$, $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, ..., \epsilon_n]^T$, and the design matrix X, the system of equations associating the model to the data set can be expressed as

$$\boldsymbol{z} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}. \tag{4.4}$$

4.3 Cost functions and optimizing of models

After having chosen a parametrized model such as (4.4), to be fitted to some data set, the next step is to estimate the parameters, $\boldsymbol{\beta}$, of the model. In the case of an $m-1$ order polynomial regression, this would correspond to finding the right values of $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_m]^T$. In order to do so, a *loss function* or *cost function*, $C(\boldsymbol{\beta})$, is

introduced. The exact formulation of the cost function depends on the model, but it always quantifies the error between projected model prediction and target by a single real number. Minimizing the cost function with respect to $\boldsymbol{\beta}$ will fit the model to the data. The exact minimizing values may in some cases be obtainable, while they in other cases may only be estimated, depending on the data set, choice of model, and cost function.

4.3.1 Ordinary least squares

Ordinary least squares (OLS) is a method commonly used for finding parameter values of linear regression models like the ones discussed. The method fits the model by minimizing the sum of the squares of the errors between the model outputs and targets. Assuming n data points and $\boldsymbol{\beta}$ parameters as previously, this results in the cost function

$$C(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n}(z_i - \tilde{z}_i)^2 = \frac{1}{2n}(\boldsymbol{z} - X\boldsymbol{\beta})^T(\boldsymbol{z} - X\boldsymbol{\beta}), \tag{4.5}$$

which can be stated as the least squares problem[2]

$$\min_{\boldsymbol{\beta}} \frac{1}{2}||X\boldsymbol{\beta} - \boldsymbol{z}||_2. \tag{4.6}$$

The necessary conditions for this minimum is achieved by

$$\frac{\partial}{\partial\beta_k}\frac{1}{2}\sum_{i=1}^{n}\left(z_i - \sum_{j=1}^{m}x_{i,j}\beta_j\right)^2 = 0, \tag{4.7}$$

for $k = 1, 2, .., m$. This can be rewritten [33][p.144] as

$$(X^TX)\boldsymbol{\beta} = X^T\boldsymbol{z}, \tag{4.8}$$

which is a system of m linear equations in m unknowns, called *normal equations*. Fitting the model thus corresponds to solving the normal equations for $\boldsymbol{\beta}$. Ideally, these values are such that $C(\beta) = 0$, or $\boldsymbol{z} - XT\boldsymbol{\beta} = 0$ – although this is generally not possible. If X^TX is invertible, which corresponds to $\text{rank}(X) = m$ [33] [p.144] (full column rank i.e. linearly independent columns), the normal equations may be solved for the (unique) global minima, giving;

$$\boldsymbol{\beta} = (X^TX)^{-1}X^T\boldsymbol{z}. \tag{4.9}$$

This matrix inversion may be carried out using for example SVD or LU decomposition. Alternatively, minimizing (4.5) may be accomplished using orthogonal transformation with QR decomposition. Examples and a brief discussion can be found in the appendix.

[2] $||A||_2 = ||\lambda_{max}(\boldsymbol{A}^*\boldsymbol{A})||_2$ where $\lambda_{max}(\boldsymbol{B})$ is the largest value of matrix $\boldsymbol{B}$, and A^* is the conjugate transpose of $\boldsymbol{A}$.

4.4 Bias-variance tradeoff

A supervised learning algorithm ideally results in a model that predicts well on data not included in the training set and is able to learn dependencies present in the training set. Model complexity is in most cases a requirement for learning anything more than the simplest dependencies between variables, or data regularities, in the training set. However, while increased model complexity could enable composite mapping, it may also lead to *overfitting*. When a model is overfitted, the model is too exactly tuned towards its training data such that it suffers loss of generalization. This manifests itself in model output that is predominantly consistent with training set targets, regardless of input. Conversely, an *underfitted* model ignores most of the data regularities in the training set and will by and large not produce outputs consistent with the training targets. Both cases usually result in poor prediction quality on the test set.

The expected error from an unseen sample can be decomposed into three different categories. Suppose that there is a set of training data, $D = \{x_i, z_i\}_{i=1}^{i=n}$, and some regression model $z_i = \beta_0 + \beta_1 x_i + \epsilon_i = \tilde{z}_i + \epsilon_i$ with random noise reflected in the residual error by the (scalar) expectation value $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and variance $\mathrm{Var}[\boldsymbol{\epsilon}] = \sigma^2$. The expected mean square error (MSE) of the model may then be written as(see appendix 1),

$$\mathbb{E}\left[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2\right] = \frac{1}{n}\sum_i (z_i - \mathbb{E}\left[\tilde{\boldsymbol{z}}\right])^2 + \frac{1}{n}\sum_i (\tilde{z}_i - \mathbb{E}\left[\tilde{\boldsymbol{z}}\right])^2 + \sigma^2 \tag{4.10}$$

$$= \mathrm{Bias}^2(\tilde{\boldsymbol{z}}) + \mathrm{Var}(\tilde{\boldsymbol{z}}) + Error. \tag{4.11}$$

The first error term, Bias [3], corresponds to the mean deviation between the average prediction, $\mathbb{E}\left[\tilde{\boldsymbol{z}}\right]$, and the true response, z_i. The second term is the model *variance*, or the squared expected difference of the predictions about their mean. The third term is the irreducible error from data noise.

Figure 4.2 illustrates low and high bias and variance, with the center of the targets representing correct prediction value. Bias measures how far the average prediction is from the correct value, or how "biased" the model is to some specific solution. In the figure 4.2, this is reflected by how far the center of the clusters of predictions (blue dots) are from the target center. On the other hand, variance is a measure of the spread of predictions. Low variance corresponds to consistent predictions, or a tight cluster of blue dots, while high variance is associated with less consistency and a larger spread.

Reducing the bias error usually means enhancing model fit in order to more precisely capture data trends. This can be achieved through increased model complexity and/or training time, and may yield highly accurate predictions on the training set and achieve a low bias error on test data. However, as model complexity increases, small perturbations in input data may result in drastic changes to predictions, or increased variance following transition from underfitting to overfitting. The *bias-variance trade-off* is a model property describing the conflict of wanting to minimize the two opposing error sources bias and variance – as one decreases, the other generally increases. Because of this, one generally seeks a middle ground in which the total error is minimized.

[3] First of the Magi, first apprentice to Juvens [34].

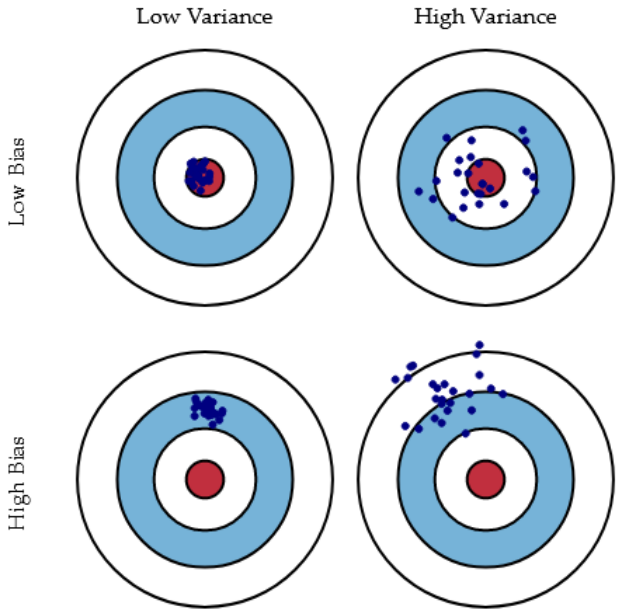


Figure (4.2) Visualization of bias-variance. Figure is taken from Scott Forman-Roe[35]

4.5 Regularization

Regularization, or shrinkage methods are examples of ways to exert some control over the bias-variance trade-off. In general, this is done by introducing an extra term to the cost function which includes the model parameters, $\boldsymbol{\beta}$, and a penalty parameter γ which effectively shrinks the parameters.[4] This is intended to decreased the impact of variable perturbations and ultimately reduce variance. Below, two such methods are briefly discussed, Ridge Regression and Lasso Regression, exemplified using linear regression and OLS.

4.5.1 Ridge Regression

The Ridge Regression cost function [37] [p.22] minimizes the square of the residual sum with a quadratic penalty term:

$$C(\boldsymbol{\beta}) = \frac{1}{2n}(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta}) + \gamma\boldsymbol{\beta}^T\boldsymbol{\beta}, \tag{4.12}$$

where $\gamma \geq 0$ is the penalty constant. As is evident from the equation, $\gamma = 0$ results in a solution equal to OLS (4.5), while increased values of γ results in increased shrinkage, or lower $\boldsymbol{\beta}$ values. While applying OLS to a regression model yields one set of coefficients, every value of γ is associated with a different set of coefficients[38][p.215], meaning that optimal use of ridge regression may require more thorough optimization. If the

[4]Originally, the shrinkage parameter γ was introduced to address rank deficiency [36] [p.64] – which it does, as the method adds a non-zero constant to the diagonal elements.

relationship between predictors and response is close to linear, OLS may result in a model with low bias but high variance [38] [p.218]. In such scenarios, minor perturbations in training data may cause large changes in the OLS estimated coefficients. For such cases, increased shrinkage using ridge regression may reduce the variance at the cost of a small increase in bias. The cost function (4.12) results in the following expression for the regression coefficients;

$$\boldsymbol{\beta}^{\text{Ridge}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{z}. \tag{4.13}$$

4.5.2 Lasso Regression

Lasso Regression is similar, but not identical, to Ridge Regression. Instead of a quadratic penalty term, Lasso Regression uses the absolute value of the coefficients [39];

$$C(\boldsymbol{\beta}) = \frac{1}{n}(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda||\boldsymbol{\beta}||. \tag{4.14}$$

The minimization problem based on this cost function has solutions which are non-linear to $\boldsymbol{z}$, and there is, in contrast to Ridge Regression, no closed form solution [36] [p.68]. Thus, to determine the model coefficients by Lasso Regression, a computing algorithm must be used. In addition, while Ridge Regression does not set any coefficients to exactly zero [36][p.68], Lasso Regression may do exactly that, which means that Lasso Regression may eliminate certain input parameters. This means that Lasso Regression produce *sparse* models – models which might only include a subset of the variables in the data set [38] [p.219]. A consequence of this is that Lasso Regression generally produces models that are easier to interpret than Ridge Regression and OLS.

In-depth comparisons between Ridge and Lasso may be studied further in the literature, such as [38] [p.223-224], on which the following is based. Simply put, Ridge Regression leads to better prediction when performed on data where there is only a small number of predictors of roughly the same importance. However, for a large set of predictors, where a small subset of predictors are significantly stronger linked to the response than the rest, Lasso performs better. Especially so when the response has low dependency on the remaining subset. In this regard it is important to note that when analyzing a real data set, the number of significant predictors is unknown a priori. This means that in order to determine the best suited method of regression, one often has to test different methods and compare the outcomes.

4.6 Qualitative modelling and logistic regression

A qualitative model, often called a classifier, is a function which assigns class labels to data points. Although closely related to some qualitative methods, the regression methods discussed so far in this chapter were examples of quantitative modelling, and are ill suited for discrete outputs. Using for example least squares for categorical prediction would typically lead to problems of ordering of responses and category gaps (explained in detail in [38] [p.130]). Because of this, special methods are needed for discrete variables.

An example of qualitative modelling is logistic regression, which models the probability that a qualitative response falls within a category. In the case of two mutually exclusive categories or classes, such as "True" ($z = 1$) and "False" ($z = 0$), this translates to modelling the relationship between the probability of $z = 1$ given X, as $P(z = 1|X; \boldsymbol{\beta}) = p(X; \boldsymbol{\beta})$, using the *logistic function* [38] [p.132]

$$p(z = 1|X, \boldsymbol{\beta}) = p(\boldsymbol{X}; \beta) = \frac{e^{\boldsymbol{\beta}^T \boldsymbol{X}}}{1 + e^{\boldsymbol{\beta}^T \boldsymbol{X}}}, \tag{4.15}$$

where $\boldsymbol{\beta} = [\beta_0, \beta_1, ...\beta_p]$, $\boldsymbol{X} = [1, X_1, X_2, ...X_p]$, and p is the number of predictors that the the binary response z is modelled upon. Due to mutual exclusivity, it models $z = 0$ as a function of X, and $P(z = 0|X; \boldsymbol{\beta}) = p(X'; \boldsymbol{\beta})$, as $1-p(\boldsymbol{X}; \boldsymbol{\beta})$. When fitting the model, or estimating the β coefficients, to a data set $\mathcal{D} = \{z_i, \boldsymbol{x}_i\}_{i=1}^n$, of n observations, the desired fit is the one where the predicted probabilities of $p(\boldsymbol{x}_i)$ most precisely corresponds to z_i. In other words, one seeks the model which has the highest probability of predicting the data set, which is achieved by utilizing the *maximum likelihood* method (4.16) [38] [p.133]

$$\ell(\boldsymbol{\beta}) = \prod_{i:y_i=1} p(\boldsymbol{x}_i; \boldsymbol{\beta}) \prod_{i':y_{i'}=0} (1 - p(\boldsymbol{x}_{i'}; \boldsymbol{\beta})). \tag{4.16}$$

Taking the log of the maximum likelihood, one obtains the log-likelihood. The (reordered) negative log-likelihood constitutes the cost function, $\mathcal{C}(\hat{\beta})$ [36] [p.120] (in Hastie et al, the log-likelihood is maximized) of the logistic regression, or the *cross entropy*,

$$\mathcal{C}(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \left(y_i(\boldsymbol{\beta}^T \boldsymbol{x}_i) - log(1 + e^{\boldsymbol{\beta}^T \boldsymbol{x}_i}) \right). \tag{4.17}$$

Given a data set, the cross entropy is exclusively a function of the regression coefficients, $\boldsymbol{\beta}$. This means that the desired values of $\boldsymbol{\beta}$ can be found by minimizing the cross entropy with respect to the coefficients.

4.7 Gradient based optimization

The minimization of a cost function may in some cases be carried out analytically, for example when using least squares error. For some cost functions, however, the analytic expression may be inconvenient to use due to factors such as the size of a matrix which requires inverting. In other cases, close form solutions are entirely unavailable. For problems such as these, numerical methods for obtaining the desired model fit are applied instead. One such method is the *gradient descent* (GD) (4.18) [6] [p.247], also known as steepest descent. The [concept] idea is to makes an initial guess for $\boldsymbol{\beta}$ and then evaluating the gradient, $\boldsymbol{g}$, of the cost function in order to obtain an estimate closer to the desired minimizing value. This process is then iteratively repeated until the sufficient convergence of coefficients is reached. In order not to over-adjust the parameters, a *learning rate* of η is applied to the gradient – which may also serve to lengthen the step

in an iteration. The value of the learning rate can either be fixed, usually by trial and error, such that it works well for the problem at hand, or it may be adapted to each step;

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta_k \boldsymbol{g}(\boldsymbol{\beta}_k). \tag{4.18}$$

A regularization parameter, γ, is often added to the gradient in an extra term, similar to the regularization approach I discussed for OLS. This effectively modifies the gradient descent by adding shrinkage which may inhibit overfitting (see bias-variance trade off) to training data. This could for example mean modifying (4.18) to

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta_k \boldsymbol{g}(\boldsymbol{\beta}_k)(1 + \gamma). \tag{4.19}$$

A common improvement to GD is the *stochastic gradient descent* (SGD), which involves splitting the data into mini batches, then randomly selecting one batch at a time from which to compute the gradient. More detailed information is available in literature such as Kevin P. Murphy's book on machine learning [6] [p.262]. Special "momentum" based algorithms, which introduce further improvements on the SGD method are common in machine learning. As basic gradient descent may get "stuck" in local minima, momentum methods take the gradient of previous optimization iterations into account in order to carry the optimization through the local minima. An example of this is the ADAM algorithm [8], which uses adaptive and individual learning rates for each parameter.

Chapter 5

Neural Networks

Neurons, or nerve cells, are the building blocks of the biological brain. Although each neuron is relatively simple[1], the composite structure of neurons in a network enable the vastly complex patterns and signal mappings that constitute all human sensory and cognitive activity. Each neuron has a lipid bilayer membrane with channels and pores which regulate the composition of ions inside the neuron by type-specific exchange with the intercellular medium of the brain. Because ions have electric charge, this results in an electric potential across the membrane, typically around -65 mV (inside the cell) when at rest [40] [p.13-15]. If the membrane potential increases above some threshold, the neuron will "activate" or fire an action potential, involving the opening and closing of specific channels and pores, resulting in a swift decrease of the membrane potential. This neuronal firing occurs at one spot inside the neuron and propagates along the membrane by inducing new action potentials in adjacent segments of the neuronal membrane. Starting in the main body of the pre-synaptic neuron, the action potential propagates through the axon (see figure 5.1) before translating into stimuli of other neurons through synaptic connections. The strength and sign of a post-synaptic response depends on the characteristics of the individual synapse. Some synapses change in strength over time through synaptic plasticity, and long-lasting changes to synapses are hypothesized to substantially contribute to memory and learning [40] [ch.7.5].

In the context of data science, artificial neural networks (ANNs), or more commonly just neural networks (NNs)[2], is a sub-field of machine learning inspired by biologic neurological capacity for analysis and learning from experience. These networks offer great flexibility in modelling range, and can be designed either as supervised, unsupervised, or reinforcement learning algorithms. NNs are based on interconnected nodes somewhat similar to neurons with synapses in the biological brain. Nodes are usually arranged in layers with connections between between layers as in figure 5.2 (which shows a feed forward neural network), the notation used in the figured is explained in more detail in the next section. The first layer is commonly called the input layer, while the last

[1] Neurons are simple in the sense that neurons are understood in detail in contrast to the brain as a whole.

[2] The following chapter is loosely based on the informative and well written book by Michael Nielsen [5], lectures notes in FYS-STK4155 [42] and "Machine Learning" by Kevin P. Murphy [6].

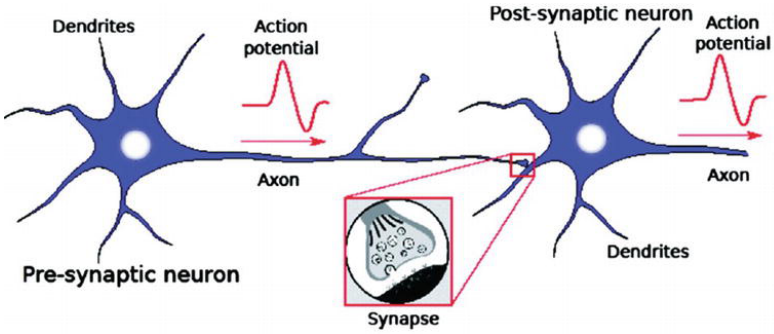


Figure (5.1) Biological neuron with synapse, taken from [41][ch.12]

layer is known as the output layer. Layers which are neither first nor last are called hidden layers. There are many different types of neural networks with different applications and characteristics. Convolutional neural networks (CNN)[6] [p.564] are networks which use filters to pool information from a selection of nodes in order to create feature maps. These are typically used for problems such as image analysis. Recurrent NNs [43] (RNN) include inter layer connections which are not strictly sequential, meaning that they feature feed backwards between layers. RNNs are commonly used when the ordering of data is important, such as language translation. Conversely, feed forward neural networks (FFNN), which is the type of NNs used in this thesis, use strictly sequential nodal connections between layers, meaning that there is no cyclical flow of information. FFNNs with even a single hidden layer have been shown by Hornik et al. [7] to be universal approximates. In their paper, they established that a standard multilayer feed forward network is capable of mapping any Borel measurable function [44] from one finite space to another with arbitrary accuracy given a sufficient amount of nodes. Although this is true for *shallow* networks, NNs with few hidden layers as well as *deep* networks, i.e. NNs with many hidden layers, it has been shown [45] that deep networks in some cases give improved performance over shallow NNs. One of the reasons for this is that a deep network will have more weights than a shallow network with an equal number of units, and thus more flexibility.

5.1 Feed Forward Neural Networks

The networks used in this thesis are based on feed forward neural networks (FFNNs) with some modifications. Every concept required to understand FFNNs are present in other types of NNs, and their computational potential and relative simplicity keep them relevant in the field of supervised learning in scientific programming. FFNNs are artificial neuron networks consisting of an ordered series of regression models with connected inputs and outputs [6] [p.563]. Each node in the network is similar to a neuron with synapses to the nodes in the next layer, and the synaptic strength of each connection is represented by regression coefficients. The network consists of $L + 1$ layers, with M neurons (regression models), or nodes, in each layer. The neurons are not necessarily of the same type, as one can for example use logistic regression models for all nodes except

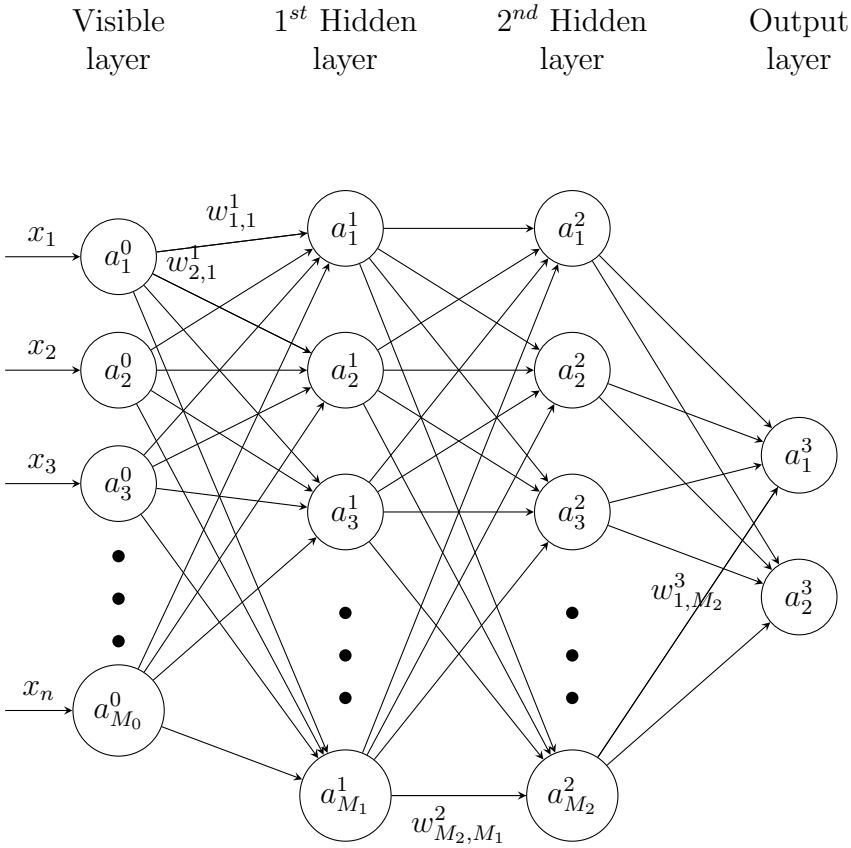


Figure (5.2) A feed forward neural network with an input layer (n variables), two hidden layers and an output layer ($L = 3$), with M_l nodes in layer $l = 0, 1, 2, 3$.

nodes in the output layer, $l = L$, which could be linear regression models if the problem is a regression problem. The set of outputs,

$$a^0 = \{a_i^0\}_{i=1}^{M_0}, \tag{5.1}$$

of the M_0 nodes in layer $l = 0$, the input layer, are simply the independent variables. a^0 are fed to the nodes in layer $l = 1$, which outputs some transformation on a^0, and so on, until the outputs, a^{L-1}, from layer $l = L - 1$ is fed to layer $l = L$, the output layer. Outputs from layer L, a^L, are the network's outputs or model predictions. The propagation of a signal from inputs to output through the architecture of the network is known as *feed foward*, hence the name feed forward neural network. The $L_h = L - 2$ layers in-between layer 0 and layer L, are deterministic functions of the input, and are called *hidden layers*. Every node has a unique weight associated with each of its inputs, similar to the coefficients corresponding to the inputs used in regression analysis. Node j in layer l will for example have M^{l-1} weights, denoted $w_{j,i}^l$, for $i = 1, 2, ..., M^{l-1}$. Each node also has its own *bias*, b_j^l, which regulates the output tendency of neuron j in layer

l. The sum of the weighted inputs and the bias,

$$z_j^l = \sum_i^{M^{l-1}} w_{j,i}^l a_i^{l-1} + b_j^l, \tag{5.2}$$

is usually called the activation of the neuron. The output of the neuron

$$a_j^l = f^l(z_j^l), \tag{5.3}$$

is the value of the *activation function* $f^l(z)$ of layer l, which performs a final transformation of the neuron activation – more on this later. This facilitates optimization of the network by evaluation of the output error through the process of *back propagation*, which yields activation gradients as functions of the network's weights and biases.

5.1.1 Feed forward

The propagation of neuron outputs in the network, resulting in prediction, is called feed forward. The process is well described in literature such as Michael Nielsen's book [5]. It is common to arrange the nodal outputs matrices in a way similar to what I demonstrated in the section on regression analysis, such that the mathematical operators involved in optimizing and predicting can be carried out efficiently. I have previously refer to a^l as a set, but will from now on refer to a^l as a matrix $\mathbb{R}^{M_l \times n}$. This means that the output of the input layer, a^0 (the design matrix), is on the form $M_0 \times n$, where n is the number of data points of $p = M_0$ independent variables from the training set. Subsequently, a^l for $l = 2, 3, .., L$ can be written as

$$a^l = f^l(z^l) = f^l(w^l a^{l-1} + b^l),$$

where $f^l(z)$ represents an element-wise application of the activation function for layer l, where

$$w^l = \begin{bmatrix} w_{1,1}^l & w_{1,2}^l & \dots & x_{1,M_{l-1}}^l \\ w_{2,1}^l & w_{2,2}^l & \dots & x_{2,M_{l-1}}^l \\ \dots & \dots & \dots & \dots \\ w_{M_l,1}^l & w_{M_l,2}^l & \dots & x_{M_l,M_{l-1}}^l \end{bmatrix}, \tag{5.4}$$

and where

$$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \dots \\ b_{M_l}^l \end{bmatrix}. \tag{5.5}$$

This results in the network predictions $\tilde{z} = a^L$ on the form $M_L \times n$, which corresponds to n predicted values of M_L variables.

5.1.2 Back propagation

Once the output, a^L, is obtained through the feed forward process, the network has made predictions for the dependent variables, z, for each of the n observations. However, finding an initiation of the weights and biases such that $a^L \approx z$ with sufficient accuracy is unlikely. The quality of these predictions are quantified by the cost function C, which facilitates optimization similar to what has been exemplified earlier in this chapter. There are two assumptions about the cost function that need to hold true [5]. It should be an average over the n observations, and it should be written as a function of the outputs of the neural networks, a^L,

$$C = \frac{1}{n} \sum_{x_i} C(a^L(x_i)), \tag{5.6}$$

where x_i is the i'th observation of the p independent variables. An example of such a cost function is ordinary least squares, which was introduced in the section on regression analysis. As with other ML algorithms, the cost function enables optimization of the model parameters, in this case the weights and biases of the network. When optimizing FFNNs, this is done through back propagation [5]. The process consists of calculating the error of each layer, δ_l, starting with the output layer, $l = L$. The error of the j'th output layer,

$$\delta^L = \nabla_a C \odot f'(z^L), \tag{5.7}$$

is calculated using $\nabla_a C$, which is a vector with elements $\partial C / \partial a_j^L$, where j is the output of the j'th node in L, and where $\odot$ is the *Hadamard product*, denoting the element-wise product of two vectors. $f'(z^L)$ is a vector containing the derivatives of $f^L(z^L)$ similarly ordered as the cost function gradient. Knowing this, it is clear that δ^L is an application of the chain rule on the derivative of $C(a^L)$, which is the basis for the entire back propagation. Having obtained δ^L, the error of layer $L-1$ can be calculated by the same principle using

$$\delta^l = \delta^{l+1}(w^{l+1})^T \odot f'(z^l). \tag{5.8}$$

The process is then iteratively repeated layer by layer, going backwards. Based on the errors, the layer-wise gradients of the cost function with respect to the biases can be calculated using

$$\frac{\partial C}{\partial b^l} = \delta^l, \tag{5.9}$$

and similarly for the weights of l by

$$\frac{\partial C}{\partial w^l} = a^{l-1}\delta^l. \tag{5.10}$$

After calculating the gradients involving the model parameters, some gradient based optimization method is applied in order to obtain new values for the weights and biases. Once the weights and biases are updated, a new feed forward process is initiated, followed by back propagation and corresponding adjustments of parameters until the cost function is sufficiently minimized or the maximum number of iterations (feed forward + back propagation) is reached.

5.1.3 Activation functions

While describing the forward feed and backpropagation, The activation function represents the activation threshold in biological neurons described earlier. Usually the activation function is the same for all nodes in a layer. The inclusion of activation functions also allows for optimization of the network by *back propagation*. Given an input, the gradient of the activation function will include both weights and biases, allowing for parameter updates through using for example gradient descent. If the activation function of multiple layers is the identity function $f(z) = z$, that part of the model collapses to a linear model [36] [p.394], which limits the predictive power of the network. Because of this $f(z)$ is usually a non-linear function. Furthermore, without activation functions, the output of the neuron would be unbounded, while activation function introduce a way to control the output range, such that the activation does not blow up. There are a large number of activation functions to chose from, and usually it is difficult to know beforehand which ones will suit the task at hand best. Because of this, it is common practice to use trial and error combined with insights into the data type and network output. I have chosen to include a small selection of activation functions, visualized in figure 5.3 and discussed in brief below. There are, however, many more activation functions, such as the adaptive piecewise linear (APL) activation unit [46], which has performed well on CIFAR sets and in high-energy physics modelling, and Maxout [47] which performed well in deep networks on a range of image recognition benchmarks.

Sigmoid function

The sigmoid function,

$$f(z) = \frac{1}{1 + e^{-z}},$$

is monotonic with range $(0, 1)$, and is typically one of the first activations encountered when learning about NNs. As can be seen in B.1, the function is very sensitive around $z = 0$, but quickly lose responsiveness as the output tends towards 0 or 1. In the context of parameter updates, the latter leads to very small gradients, known as *vanishing gradients*, and consequently little to no learning for the network. Because of this, extra care is required when initializing the weights and biases. Despite this, the activation function is still popular, especially for binary classification [48].

Tanh

The tangent hyperbolic function, or tanh

$$f(z) = \frac{2}{1 + e^{-2z}} - 1,$$

is in many respects similar to the sigmoid function above, but unlike the sigmoid, it is centred on $f(z) = 0$ instead of $f(z) = 0.5$, and has range $(-1, 1)$. This ensures that the outputs are closer to zero and more centred, facilitating improved learning tendencies

compared to the sigmoid [48]. Like the sigmoid, tanh also imposes the issue of vanishing gradients, but despite this, it is popular, especially for nodes in hidden layers.

ReLU and leaky ReLU

Proposed by V. Nair et al. in 2010 [49], Rectified Linear Unit (ReLU),

$$f(z) = \max(0, \mathrm{z}) = \begin{cases} z, & z > 0 \\ 0, & \text{else} \end{cases},$$

is an activation function which is mostly used in hidden layers. It introduces non-linearity by its piecewise definition. One of its advantages is that it eliminates vanishing gradients at a computational discount compared to the sigmoid or tanh. However, as the function is unbounded for positive values, it runs the risk of blowing up. The function is mostly differentiable, with a gradient of 0 at $z < 0$ and 1 at $z > 0$. Meanwhile, at $z = 0$, it is undifferentiable. This may be remedied by fixing the gradient at $z = 0$ to some number. The gradient for $z < 0$ is nonetheless an issue, as nodes may be become indefinitely stuck in an inactive state ($f(z) = 0$), a problem called "dying ReLU". *Leaky ReLU* addresses this issue by multiplying the activation with a small constant α (~ 0.01) in the inactive region;

$$f(z) = \max(0, \mathrm{z}) = \begin{cases} z, & z > 0 \\ \alpha z, & \text{else} \end{cases}.$$

Exponential Linear Unit

Exponential Linear Unit (ELU)[50] is an adaptation of ReLU with exponential activation for negative values. Using ELU brings the activations closer to zero, compared with RELU, without adding much computational cost. This means that ELU decreases the propagated variation in data, making the activation more noise-robust.

$$f(z) = max(0, z) = \begin{cases} z & z > 0 \\ \alpha(exp(z) - 1) & \text{else} \end{cases}.$$

Exponential activation

The pure exponential activation

$$f(z) = \max(0, \mathrm{z}) = \exp(\mathrm{z}),$$

is less common than the other activation functions. However, when the target has explicit exponential dependency on the predictors, using it for the output layer is an easy way of ensuring that the network reflects this. However, using pure exponential activation for hidden layers is less appealing as even small inputs might result in very large outputs.

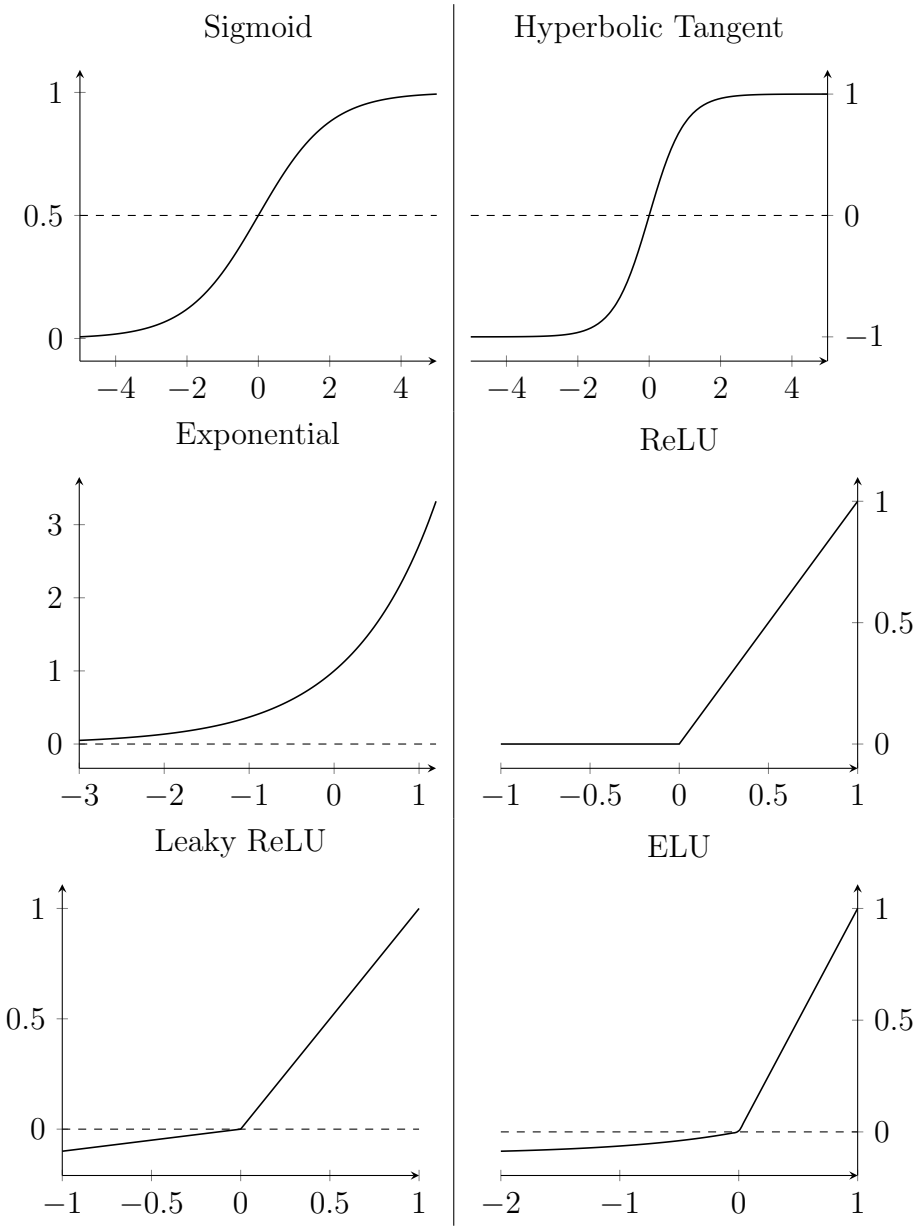


Figure (5.3) Visualization of the activation functions described in 5.1.3. From [9].

5.2 Boltzmann Machines

Although this thesis focuses on feed forward neural networks, I have chosen to include this short section as an example of another type of neural networks which are commonly used in Variational Monte Carlo (VMC). Boltzmann Machines[3] (Ackley et al. 1985) are

[3]I have based the following section, and the more detailed description in an appendix, on the book "Machine Learning" by Murphy[6] [Ch. 27], as well as the informative and well written thesis of Nordhagen [51] which focuses on Boltzmann machines in VMC simulations.

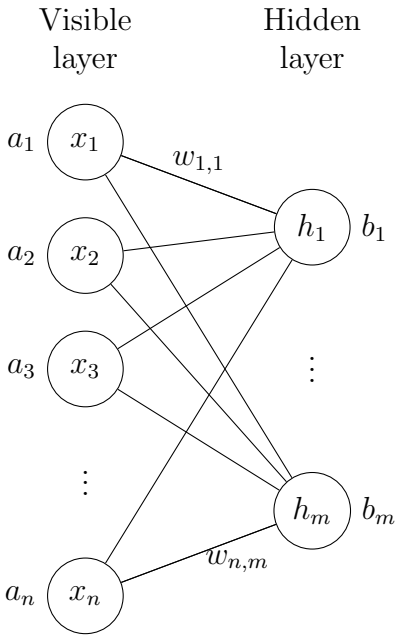


Figure (5.4) A restricted Boltzmann machine (RBM) with $1, .., i, ...n$ visible nodes and $1, .., j, ...m$ hidden nodes, where x_i, h_j are the values of node i and j in the visible and hidden layer respectively, and a_i, b_j the associated biases.

a class of stochastic and *generative* artificial neural networks. Generative models aim at learning a probability distribution, as opposed to for example feed forward neural networks, which are *discriminative models*, meaning that FFNNs predict an output value. Restricted Boltzmann machines (RBMs) are a special case of Boltzmann Machines that only include connections between nodes in different layers, meaning it has no intra-layer connections. Such networks commonly only consist of two layers, a hidden layer and a *visible layer*, which in the context of VMC typically has nodes representing the particle positions (one for each particle in each dimension). As was the case for FFNNs, the layers of an RBMs consist of nodes with weights and biases, but it does not include an output layer. Instead, the visible layer acts both as input and output, and the hidden layer works as a latent variable. This is covered in more detail in its own appendix chapter.

Part III

Methods

Chapter 6

Variational Monte Carlo

(a) Stanislaw "Stan" Ulam (1909–1984), Professor in Mathematics at Princeton University.

(b) John von Neumann (1903–1957), Professor in Mathematics at Princeton University.

(c) Major General Leslie Richard Groves (1896–1970).

(d) J. Robert Oppenheimer (1904–1967), American theoretical physicist.

(e) Edward Teller (1908–2003), known as the father of the hydrogen bomb.

(f) Nicholas Metropolis (1915–1999), American mathematician and physicist.

(g) Enrico Fermi (1901–1954), Italian physicist, Nobel laureate.

Figure (6.1) Some of the most famous researches vital to the development of the Monte Carlo method. Published in [52].

During World War II, the first programmable electronic computer [53] was designed by physicist John Mauchly and engineer Presper Eckbert at the University of Pennsylvania in Philadelphia. Mauchly had realized that if one could build electronic circuits capable of counting, the computer would also be able to do arithmetic, and thus many more complex tasks such as solving differential equations, with hitherto unrivalled speed. The ENIAC (Electronic Numerical Integrator and Computer), as it was called, was built with the aim of calculating firing solutions for guns using ballistic arcs, a task that was currently being carried out using mechanical desk calculators, which comparatively took

ages. Although the ENIAC was originally conceived for a limited range of intended application, its final build was with excessive computational power, which meant that it could be put to a wide array of tasks. As such, its first task was calculations for constructing a hydrogen bomb[54].

A number of mathematicians and physicists (see figure 6.1), among them Neumann, Metropolis, Ulam, and Richtmeyer, were involved with the ENIAC, and various discussions and correspondences on the subject eventually led to the development of a groundbreaking statistical method, the Monte Carlo method. Inspired by the famous casino, its name was suggested by Metropolis [53], and in his own words *not unrelated to the fact that Stan had an uncle who would borrow money from relatives because he "just had to go to Monte Carlo".* Although the ENIAC was not mentioned because the machine was classified until 1959 [52], Metropolis and Ulam published a paper on the Monte Carlo method in 1949 [55]. These advances in both statistics and electronics quickly lead to developments in the field of quantum mechanics, as solving the Schrödinger equation could now be carried out through electronic computation as a stochastic problem [56]. Since then, a large number of quantum mechanical computation methods based on Monte Carlo have been proposed, commonly called Quantum Monte Carlo, or QMC.

In this chapter, I introduce the Monte Carlo method and contextualize it for computational quantum mechanics through the Variational Monte Carlo method (VMC). I then discuss the variational principle, a quantum mechanical theorem that gives an upper bound on the system energy through variational energy. Subsequently, I show how Monte Carlo integration can be used to approximate the variational energy based on local energy sampling, and introduce two Monte Carlo sampling algorithms; the Metropolis algorithm and Metropolis-Hastings algorithm (importance sampling).

6.1 The intended use of VMC in this thesis

Before discussing the specifics of Variational Monte Carlo (VMC), I wish to give a simplified overview of the methodology used in this thesis. The overarching goal is to use computer code to run simulations which are capable of producing an estimate of the ground state energy, E_0, of a system of bosonic particles. I use the Variational Monte Carlo approach, which I will discuss shortly, to calculate the variational energy, E, which for sufficiently large simulations should approach E_0. The variational energy is calculated based on the mean of M values of the local energy, E_L, which is sampled from the energy space of the system[1]. By iteratively proposing spatial configurations of the particles in the system and evaluating the likelihood of each proposed move, the state of the system is altered such that it gradually moves towards regions of higher probabilities, and associated lower local energy.

In order to evaluate the probability and the local energy, a wave function is required. As the exact wave function of the system is unknown, a modifiable *trial wave function* (TWF) is used. After a "burn in" or thermalization period where the samples are

[1] A system of particles in VMC is an estimation of a physical system. I will however generally refer to the VMC system state as simply the system state for the sake of brevity.

discarded, the simulation should be set up such that the system is sampling from regions of (relatively) low local energy based on the trial wave function under the constrains imposed by the Hamiltonian. By using the obtained samples, the trial wave function can now be modified such that it minimizes E, and more closely approximates the ground state. If the TWF optimizes correctly, this means that the local energy samples in the next run will result in E closer to E_0. After many iterations, the uncertainty of the final energy estimate can be evaluated using the variance of the local energy average, a method which will be discussed more later.

6.2 Monte Carlo

The driving idea behind the Monte Carlo (MC) method is to sequentially propose M changes to the state of the system based on stochastic calculations for the previous state. This proposed state is then either rejected or accepted according to some set of rules. If the proposed new state, or *trial state*, is accepted, then this state is the new system state. If the trial state is rejected, the system retains it state. After all M iterations of the algorithm is completed, the set of states, $\{C_k\}_{k=1}^{M}$, where C_k is the state at iteration i, which depends on C_j when $j = k + 1$, constitutes a Markov chain in the phase space of the system. Defining $P_I(t)$ as the probability at time t for the system to be in state I, a finite Markov process can be described by

$$\frac{dP_I(t)}{dt} = \sum_{I \neq I} \left[P_J(t) W(J \to I) - P_I(t) W(I \to J) \right], \tag{6.1}$$

where $W(I \to J) \geq 0$ is the transition rate from I to J, and where

$$\sum_{B} W(I \to J) = 1 \tag{6.2}$$

for all I and J. [57] The impact on the Markov chain is arguably intuitive; the probability of I increases if the probability of J times the transition ratio from from J to I increases, and vice versa. For a Monte Carlo algorithm to execute successfully, it must fulfil two requirements. It needs to be able to sample all non-zero contributions of the phase space to the partition sum, and secondly it has to be set up such that it does not get stuck in a particular cycle of states[2]. Let us now for a moment assume we are dealing with a system of classical material particles defined by the Boltzmann distribution. The first requirement, known as *ergodicity*, maintains that the algorithm should eventually be able to access any state with a non-zero *Boltzmann weight*[3] when the initial state has a non-zero Boltzmann-weight. This ensures that the entire phase space is accessible. The

[2] This condition can be circumvented by a few specific algorithms [57]

[3] "*..., the probability that a system in thermal equilibrium occupies a state with the energy E is proportional to* $\exp(-E/(k_B T))$*, where T is the absolute temperature and k_B is the Boltzmann constant. Of course, $k_B T$ has a dimension of energy. The exponential* $\exp(-E/(k_B T))$ *is called the Boltzmann weight.* From Springer's "Encyclopedia of Mathematics" [58].

second requirement is known as the *condition of detailed balance*. If we let $T(I \to J)$ be the probability of a move $I \to J$ being proposed, and $A(I \to J)$ be the probability of such a move being accepted, the transition rate from I to J can be expressed as

$$W(I \to J) = T(I \to J)A(I \to J). \tag{6.3}$$

Instead of imposing (6.1) on the Monte Carlo algorithm, the Markov chain can be constructed such that

$$P_I \cdot A(I \to J) \cdot T(I \to J) = P_J \cdot A(J \to I) \cdot T(J \to I), \tag{6.4}$$

for every pair of states I and J. This is a sufficient, but not necessary, condition for a stationary distribution of states [59]. In addition, to ensure that the ratios and probabilities involved in the algorithm are correct, they have to be chosen such that the probability is equal to the Boltzmann weight,

$$P_I = \frac{e^{-\beta E_I}}{Z}, \tag{6.5}$$

where $\beta = 1/(k_b T)$ (which is scaled to 1), after thermalization [57]. Because of this, the ratio of probabilities can be expressed as

$$\frac{P_I}{P_J} = \frac{W(I \to J)}{W(J \to I)} = \frac{T(I \to J)A(I \to J)}{T(J \to I)A(J \to I)} = e^{-(E_I - E_J)}. \tag{6.6}$$

This will be put to use later when introducing specific Variational Monte Carlo sampling algorithms, although the systems in this thesis are governed by wave functions and not the Boltzmann distribution.

6.3 Variational Monte Carlo

Quantum Monte Carlo (QMC) is a collection of computational methods that use the Monte Carlo method to calculate the multi-dimensional integrals involved in complex quantum mechanical systems. One of the most well-known such method is called the Variational Monte Carlo (VMC) method, owing to its reliance on the *variational principle* when calculating the ground state energy of the system. As will be discussed later, the variational principle can be leveraged in combination with the Monte Carlo method to effectively solve the time-independent Schrödinger equation for (potentially) any system. There are many different algorithms based on VMC, but they all use the same general approach to calculating the ground state energy of a system, which may be broken up into three steps. First, a Hamiltonian for the particular system in question must be defined. This may in some cases require both experimental and theoretical work, but is not within the scope of this thesis. Secondly, an *ansatz*, or initial guess, on the ground state of the system in the form of a *trial wave function* with some variational parameters is proposed. Finally, these parameters are fine tuned towards energy minimization under the constraint of the Hamiltonian, using the variational principle.

6.4 The Variational Principle

The variational principle [16] [Ch. 7] (proof [60])

$$E_0 \leq \langle \Psi | H | \Psi \rangle \equiv \langle H \rangle, \tag{6.7}$$

states that given a time-independent Hamiltonian, H, the expectation value of the Hamiltonian provides an upper bound to the ground state energy, E_0, of the associated system. This theorem holds for any normalizable function Ψ, meaning that one does not actually need to know the exact ground state to get information about the ground-state energy. The closer the function Ψ is to the exact ground state, Ψ_0, the closer the resulting expectation value of $\langle H \rangle$ will be to E_0. In practice $\langle H \rangle$ is usually approximated by finite sampling using a wave function ansatz. This approach gives an indication of how far the approximation is from E_0 by using the variance of the samples, which is described in more detail in the section on error estimation. The ansatz, or trial wave function (TWF), is a function of one or more variational parameters. Using the gradient of the parameters with respect to the approximation of $\langle H \rangle$, the TWF can be iteratively tuned such that it approximates Ψ_0 increasingly well. Given a sufficiently good approximation to $\langle H \rangle$, the iterative adjustments to the TWF will result in estimates which converge towards E_0. In order for this to work, the trial wave function should typically fulfil two requirements. It should obey the wave function requirements discussed in the first part of this thesis, and it should be able to approximate the exact ground state to a high degree. Typically, the TWF is based on some insights into the system, or on a standardized method. In practice, I use Monte Carlo integration to approximate $\langle H \rangle$, and the product of basic standardized trial wave functions and a feed forward neural network to construct a trial wave function.

6.4.1 Monte Carlo integration and local energy

The trial wave function, $\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})$ is a function of $\boldsymbol{R} = (\boldsymbol{r}_1, \boldsymbol{r}_2, ...\boldsymbol{r}_n)$ which is the positions of all n particles, and have variational parameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2..., \alpha_m)$.[4] Given a Hamiltonian and a suitable trial wave function, the last ingredient in the VMC approach is estimating $\langle H \rangle$, corresponding to the variational energy, E, which can be written as [59]

$$\begin{aligned} E[H, \boldsymbol{\alpha}] &= \langle H \rangle && (6.8)\\ &= \frac{\langle \Psi_T | \, H \, | \Psi_T \rangle}{\langle \Psi_T | \Psi_T \rangle} && (6.9)\\ &= \frac{\int d\boldsymbol{R} \Psi_T^*(\boldsymbol{R}, \boldsymbol{\alpha}) H(\boldsymbol{R}) \Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})}{\int d\boldsymbol{R} \Psi_T^*(\boldsymbol{R}, \boldsymbol{\alpha}) \Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})} && (6.10)\\ &= \int d\boldsymbol{R} P(\boldsymbol{R}) E_L(\boldsymbol{R}), && (6.11) \end{aligned}$$

[4] Direct dependence on $\boldsymbol{\alpha}$ is often omitted, but is included here to show that the energy also depends on the variational parameters.

where

$$E_L = \frac{1}{\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})} H(\boldsymbol{R}) \Psi_T(\boldsymbol{R}, \boldsymbol{\alpha}) \tag{6.12}$$

is the *local energy*, and

$$P(\boldsymbol{R}) = \frac{|\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})|^2}{\int d\boldsymbol{R} |\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})|^2} \tag{6.13}$$

is the *normalized probability density*. It is worth noting that this formulation allows for wave functions that are not normalized, because

$$\frac{\Psi_T(\boldsymbol{R})}{\sqrt{\int d\boldsymbol{R} \Psi_T^*(\boldsymbol{R}) \Psi_T(\boldsymbol{R})}}$$

is normalized.

By using the average of the local energies as an *estimator* (more on this in the section on error estimates) of the variational energy, i.e.

$$E \approx \bar{E}_L = \frac{1}{M} \sum_{i=1}^{M} E_L(\boldsymbol{R}_i), \tag{6.14}$$

the integral can be approximated by a sum. This method, called Monte Carlo integration [61] [Ch. 3], is a stochastic method aimed at evaluating deterministic and high dimensional integrals – although higher dimensionality requires more samples for an acceptable approximation. Averaging over M values of E_L as $M \to \infty$, sampled according to the probability density function $p(\boldsymbol{R})$, the expectation value

$$\langle \frac{1}{M} \sum_{i=1}^{M} E_L(\boldsymbol{R}_i) \rangle \approx \int d\boldsymbol{R} P(\boldsymbol{R}) E_L(\boldsymbol{R}). \tag{6.15}$$

The probability density function $P(\boldsymbol{R})$ is however unknown. This is where the Monte Carlo method described earlier comes into play. A random walker in the phase space of the system is implemented through an algorithm to generate a Markov Chain of M local energy samples. The Markov chain converges to the desired probability density if the the transition between states, which is governed by the sampler, fulfils the requirements of ergodicity and detailed balance. It is usually constructed such that the probability of sampling each state is stationary by first allowing the random walker to thermalize in a "burn-in" period. An example of such an algorithm is the Metropolis algorithm, which I introduce in the next section.

6.5 Sampling algorithms

Markov chain Monte Carlo (MCMC) use sampling algorithms which impose certain rules on the transitions between states in order to generate a sequence of samples according

to a probability distribution – exactly what is required to carry out the Monte Carlo integration introduced above. The Markov Chain consists of a sequence of samples, and the algorithms must be formulated such that the requirements of ergodicity and detailed balance is fulfilled. Below, I introduce the Metropolis algorithm, and the Metropolis-Hastings (importance sampling) algorithm which is an improvement to the Metropolis algorithm. There are, however, many other sampling (rules) algorithms, such as Gibbs sampling, which I discuss briefly in the next chapter.

6.5.1 The Metropolis algorithm

The (plain) Metropolis algorithm [57, 61] is a highly adaptable Monte Carlo sampling method, and applicable in virtually all cases, be it lattice or off-lattice, discrete or continuous, short-range or long-range interactions. The Metropolis algorithm fulfils detailed balance by proposing small changes to a state I, resulting in a state J, that the reverse transition is just as probable. Specifically, the Metropolis algorithm assumes that the proposed moves are such that

$$T(I \to J) = T(J \to I) \tag{6.16}$$

for all I, J. Recalling that (see (6.4))

$$P_I \cdot A(I \to J) \cdot T(I \to J) = P_J \cdot A(J \to I) \cdot T(J \to I), \tag{6.17}$$

means that the acceptances probability $A(I \to J)$ and $A(J \to I)$ can be arbitrarily scaled in the range zero to one. To speed up the algorithm, it is common to scale these probabilities such that the highest of the two is equal to one. We can then express the evaluate the ratio of the probability of P_J and P_I as

$$\frac{P_J}{P_I} = \frac{T(I \to J)A(I \to J)}{T(J \to I)A(J \to I)} \tag{6.18}$$

$$\frac{P_J}{P_I} = A(I \to J), \tag{6.19}$$

From the fact that probabilities are not higher than one, it can be induced that the acceptance probability of the proposed move from I to J is

$$A(I \to J) = \min\left[1, \frac{P_J}{P_I}\right]. \tag{6.20}$$

This means that when $P_J > P_I$, the transition from I to J is always accepted. However, to ensure ergodicity and because the entire process is aimed at sampling a probability density function, stochasticity must be introduced. One way to do this is to draw a random number q between zero and one, and accept the transition $I \to J$ if $A(I \to J) \geq q$. This ensures that in the end, any states J with $P_J > 0$ accessible from I, will be sampled when the number of samples M is sufficiently large. In the context of

the real-space VMC calculations used in this thesis, one cycle in VMC algorithm with Metropolis sampling consists of randomly selecting a particle, then moving that particle from its current position $\boldsymbol{r}_i$ to a new proposed position

$$\boldsymbol{r}_i^* = \boldsymbol{r}_i + l\boldsymbol{\Delta}\mathbf{r}, \tag{6.21}$$

where l is the maximum step length, and $\boldsymbol{\Delta}\mathbf{r}$ a vector with elements $\mathcal{U}[-0.5, 0.5]$. The acceptance probability

$$A(\boldsymbol{R} \to \boldsymbol{R}^*) = \min\left[1, \frac{\Psi(\boldsymbol{R}^*)}{\Psi(\boldsymbol{R})}\right] \tag{6.22}$$

is then evaluated, and a random number $q \sim \mathcal{U}[0, 1]$ is generated. If $A(\boldsymbol{R} \to \boldsymbol{R}^*) \geq q$, the new particle position and associated system state is accepted, i.e. $\boldsymbol{R} = \boldsymbol{R}^*$, if not $\boldsymbol{R} = \boldsymbol{R}$. The local energy, E_L evaluated at $\boldsymbol{R}$ is then stored in order to later calculate the average, before a new cycle initiated until M cycles have been executed. After M cycles the energy estimate $\bar{E} = 1/(M)\sum E_L$ is obtained. The following shows the Metropolis algorithm adapted to suit the simulations I use in this thesis, using MT cycles to thermalize, where $T \in (0, 1)$ is the burn-in fraction, which is required in order for the particles to be distributed according to the sampling rules instead of the initialization.

VMC with Metropolis sampling

1. Initialize algorithm
   - Set the number of cycles, M step length, l and burn-in fraction T
   - Define a TWF, $\Psi_T(\boldsymbol{\alpha}, \boldsymbol{R})$
   - Initiate $i = 1, 2..., n$ particle positions as $\mathbf{r}_i = \mathbf{0} + \boldsymbol{\Delta r}$
   - Set $\boldsymbol{R} = (\boldsymbol{r}_1, ..., \boldsymbol{r}_i, ...\boldsymbol{r}_n)$, and $k = 0$
2. Propose new configuration
   - Select random particle i
   - Generate a new $\boldsymbol{\Delta r}$
   - Calculate $\boldsymbol{r}_i* = \boldsymbol{r}_i + l\boldsymbol{\Delta r}$
   - Set $\boldsymbol{R}* = (\boldsymbol{r}_1, ..., \boldsymbol{r}_i*, ...\boldsymbol{r}_n)$
3. Evaluate proposal
   - Calculate $w = \frac{P(\mathbf{R}*)}{P(\mathbf{R})} = \frac{|\Psi_T(\mathbf{R}*)|^2}{|\Psi_T(\mathbf{R})|^2}$
   - Generate $q = \mathcal{U}[0, 1]$.
   - If $w \geq q$, accept proposal and set $\boldsymbol{R} = \boldsymbol{R}*$, $k \to k + 1$
   - If $k > int(MT)$, Calculate and store $E_L(\boldsymbol{R})$
4. If the $k < M + MT$, go to step 2. Else calculate $E = 1/(M)\sum E_L$ and end algorithm.

6.5.2 Importance sampling: Metropolis-Hastings

An issue associated with the Metropolis algorithm described above is that there is a possibility of repeatedly sampling configurations from regions of relatively low probability, or in other words, wasting MC cycles. A method that can be used to reduce the inefficiency is the variance reducing technique *importance sampling*. Instead of suggesting moves at random, importance sampling suggests moves based on the current configuration of the system. Importance sampling in VMC is motivated by a time dependent density in a diffusion process following the Fokker-Planck equation [62]. Hammond et al. [62] describe the line of reasoning leading to particle trajectories (Brownian motion) given by the Langevin equation

$$\frac{\partial x(t)}{\partial t} = DF(x(t)) + \eta, \tag{6.23}$$

where

$$F = 2\frac{1}{\Psi_T}\nabla\Psi_T \tag{6.24}$$

is the *drift force*[5], d, the diffusion constant (in atomic units, $d = 1/2$), and η Gaussian distributed random noise with zero mean and variance $2d$. By numerical integration over a short time interval/step, Δt, a new particle position can be calculated by

$$\boldsymbol{r}^* = \boldsymbol{r} + DF(\boldsymbol{r})\Delta t + \boldsymbol{\xi}\sqrt{\Delta t} \tag{6.25}$$

where $\boldsymbol{\xi}$ is a vector of Gaussian random variables. This approach suggests particle moves which result in large Ψ_T^2, but also energy estimates which deviate increasingly from E_0 as Δt increases [62] [p.54]. The latter, however, can be corrected by adjusting how the Metropolis algorithm evaluates the proposed particle move. The modified algorithm is called the Metropolis-Hastings algorithm, and uses the acceptance probability

$$A(\boldsymbol{R} \to \boldsymbol{R}^*) = \min\left[1, \frac{G(\boldsymbol{R}, \boldsymbol{R}^*)|\Psi_T(r^*)|^2}{G(\boldsymbol{R}^*, \boldsymbol{R})|\Psi_T(r)|^2}\right], \tag{6.26}$$

instead of (6.22), where

$$G(\boldsymbol{R}, \boldsymbol{R}^*, \Delta t) = \frac{1}{(4\pi d\Delta t)^{DN/2}} \exp\left(-\frac{(\boldsymbol{R} - \boldsymbol{R}^* - d\Delta t F(\boldsymbol{R}^*))^2}{4d\Delta t}\right), \tag{6.27}$$

where D is the dimensionality of the system. This corresponds to the Fokker-Planck equation, which governs the diffusion process from which this approach is modelled. Because the acceptance probability, (6.26), is a ratio, many of the common factors cancel out. The only additonal computational complexity comes from evaluating the exponential in (6.27).

[5]The diffusion is normally represented as D, but in this thesis, D represents the dimensionality of the system.

Chapter 7

Trial Wave functions

I have now covered most of the ingredients required to carry out a variational Monte Carlo simulation, except how to specifically formulate and optimize a trial wave function (TWF), which is the subject of this chapter. The exact formulation of a specific TWF can have severe impact on the quality and validity of a VMC simulation. Factors such as erroneous assumptions on the form of the wave function, computability and level of complexity are all important. Too simplistic, and the TWF cannot sufficiently approximate the system, too complex, and the TWF becomes cumbersome to optimize. Traditionally, TWFs are comprised of single particle contributions and particle correlations. Assumptions about the correlation modelling or choice of single particle wave function may lead to irreducible errors in the ground state approximation and consequently also in the ground state energy estimate.

In this chapter I discuss how to optimize wave functions in general and explicitly list which calculations the TWF is involved in in order to carry out a VMC simulation. Then, I introduce Slater Jastrow type TWFs before moving on to the main focus of this thesis, which is neural network quantum states. I discuss how to use the feed forward neural network as part of a trial function as well as some considerations regarding the weights of the network and symmetry of the resulting TWF. Before going into details on TWF optimization, let's review the iterative VMC approach used to optimize the TWF and calculate a final energy estimate in an algorithmic format.

The VMC approach

1. Given a Hamiltonian H, chose a TWF $\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})$ with variational parameters $\boldsymbol{\alpha}$ for N particles.
2. Set the number M of iterations and samples per iteration.
3. Use a sampling algorithm (such as Metropolis) to get generate Markov chains of M samples of E_L and $\nabla_{\boldsymbol{\alpha}}\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})$.
4. Use the samples of E_L and $\nabla_{\boldsymbol{\alpha}}\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})$ from the last VMC execution to optimize Ψ_T w.r.t. $\boldsymbol{\alpha}$, such that E_L is minimized.
5. Repeat 3. and 4. according to the number of iteration.
6. Calculate a final estimate energy $E = 1/M \sum E_L$

In the algorithm above, I introduced a new quantity, $\nabla_{\boldsymbol{\alpha}}\Psi_T(\boldsymbol{R}, \boldsymbol{\alpha})$, which is required to optimize the trial wave function between each iteration. This I will cover in more detail in the next section. If M is large enough to provide a sufficient basis for optimization, the final energy estimate should now in general be closer to E_0 than an estimate produced after fewer iterations.

7.1 Optimizing the trial wave function

In order to estimate the ground state energy, adjustments to the trial wave function between VMC simulations is necessary. The variational parameters will realistically not be initialized with optimal values, and will therefore require updating. As the VMC method is implemented through algorithms, and the aim is that the algorithm improves the energy estimate over time, this can be viewed as machine learning (ML) problem. VMC can generally be regarded as a supervised learning problem, but has no pre-defined data set. Instead, the data is generated while the model optimizes. This means that the quality of each training sequence of the model, or VMC iteration, is dependent on how large a data set it has been allowed to generate with its current variational parameters. If the TWF is updated based on many samples, we can in general expect the update to be more precise than it would have been with fewer samples. However, because each sample requires calculations (which takes time), the model cannot be given arbitrarily large sample sizes to update on. When using a gradient based optimization scheme, the TWF can also be expected to require many updates in order to be sufficiently trained. The trade-off between exploring the current model versus exploiting it in order to quickly update it is especially relevant when dealing with complex trial wave functions. Too greedy exploitation, and the TWF may become unstable. Too much exploration, and the TWF will never be fully optimized. I will address this trade-off in relation to my own experiments later. For now, let us discuss how the TWF is updated in practice.

7.1.1 The cost function

In VMC we seek to minimize the energy estimate $E = \langle H \rangle \approx \langle \bar{E}_L \rangle$. By using the energy as the cost function, the TWF will eventually reach its best approximation of the ground state under the constraint of a given Hamiltonian, given sufficient samples for each update. As such, we define the cost function as

$$C(\boldsymbol{\alpha}) = \langle \bar{E}_L \rangle, \tag{7.1}$$

where the variational parameters $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_i, ..., \alpha_m)$ of the trial wave function are the model parameters. Another viable cost function in a VMC scenario is variance optimization [63]

$$\sigma^2 = \langle (E_L - \bar{E}_L)^2 \rangle, \tag{7.2}$$

which minimizes the parameters with respect to the variance, σ, as it can be expected to decrease as $E \to E_0$ (see the section on error estimation). Having defined a cost function, a gradient descent (GD) method can be used to update the algorithm. When using the energy optimization approach, the gradients of the cost function w.r.t. $\boldsymbol{\alpha}$ is required in order to use GD, i.e.

$$g(\boldsymbol{\alpha}_i) = \nabla_{\alpha_i} \langle E_L \rangle, \tag{7.3}$$

where $\nabla_{\alpha i}$ is the gradient of the variational parameters α_i. By applying the chain rule to the local energy equation (6.11), the following expression can be obtained [63] for parameter α_i;

$$\nabla_{\alpha_i} \langle H \rangle = 2 \left\langle \frac{\Psi_{T,\alpha_i}}{\Psi_T} \left(E_L - \langle E_L \rangle \right) \right\rangle, \tag{7.4}$$

where Ψ_{T,α_i} is the derivative of Ψ_T with respect to parameter α_i. Using

$$\nabla(f)/f = \nabla(\ln f), \tag{7.5}$$

the gradient can be rewritten as

$$\nabla_\alpha \langle H \rangle = 2 \left(\langle E_L \nabla_{\alpha_i} \ln \Psi_T \rangle - \langle E_L \rangle \langle \nabla_{\alpha_i} \ln \Psi_T \rangle \right). \tag{7.6}$$

Similarly to $\langle E_L \rangle$, the quantities $\langle E_L \nabla_{\alpha_i} \ln \Psi_T \rangle$ and $\langle E_L \rangle \langle \nabla_{\alpha_i} \ln \Psi_T \rangle$ can be calculated using Monte Carlo integration. As higher numbers of cycles are expected to bring $\bar{E}_L$ closer to E_0, they should in general also produce more accurate $\boldsymbol{\alpha}$ gradients.

7.1.2 Calculations involving the TWF

A VMC algorithm requires certain calculations involving the chosen trial wave function. These calculations are in practicality the only footprint of the trial wave function, and are callable in the implementation.

1. In order to compare new configuration proposals, a *trial wave function evaluation* given some specific particle configuration $\mathbf{R}$ is required, $\Psi(\mathbf{R})$. It is worth noting that it is often possible to work around a direct TWF evaluation by for example simplifying the ratios of the trial wave function for two different configuration when calculating acceptance probability.

2. As the TWF optimization scheme above showed, the *gradient with respect to the variational parameters* is also required, i.e.

$$\nabla_{\alpha_i} \ln \Psi_T. \tag{7.7}$$

3. The TWF is the basis for estimating the variational energy through the local energy evaluation. This includes the kinetic term of the Hamiltonian, involving the *Laplacian of the wave function*;

$$\sum_i \frac{1}{\Psi(\boldsymbol{R})} \nabla_i^2 \Psi(\boldsymbol{R}). \tag{7.8}$$

4. Lastly, when using importance sampling, the *gradient with respect to spatial configuration* for particle k, $\nabla_k \Psi$, is needed to calculate the drift force

$$F_k = 2\frac{1}{\Psi} \nabla_k \Psi. \tag{7.9}$$

7.2 Slater-Jastrow

A popular method for constructing a trial wave functions is using the so called *Slater-Jastrow* approach. A Slater-Jastrow wave function is composed of two factors, S and J;

$$\Psi_{SJ} = SJ \tag{7.10}$$

where J is the *Jastrow factor* which models particle correlations, and S the *Slater factor* which is the single-particle contribution to the wave function.

7.2.1 The Slater Factor

The Slater factor [15] [p.49-53] is a determinant (or permanent) of basis functions from a chosen basis set. The basis set usually consists of either single particle wave functions (SPFs), such as atomic orbitals, or simple multi-particle functions such as molecular orbitals (obtained from the Hartree procedure). Typically, Slater determinants are used for multi-fermionic systems in order to construct a wave function that fulfils the Pauli pricinple, while a system of bosons is usually represented by a permanent (i.e. a product of single particle functions – SPFs). This thesis is mainly concerned with bosons, but I have chosen to include the following on Slater determinants as the methods I use should be adaptable to systems of fermions.

For a system of two non-interacting (interaction is as mentioned addressed through the Jastrow factor) fermions with spin orbitals $\chi_1(\mathbf{X})$ and $\chi_2(\mathbf{X})$, and coordinates $\mathbf{X}_1 = (r_1, m_{s1})$ and $\mathbf{X}_2 = (r_2, m_{s2})$, the total wave function of the system, $\Psi(\mathbf{X})$ can be defined as the Slater determinant of the system, $S(\mathbf{X})$,

$$\Psi(\mathbf{X}) = S(\mathbf{X}) = \frac{1}{\sqrt{2}} \begin{vmatrix} \chi_1(\mathbf{X}_1) & \chi_2(\mathbf{X}_1) \\ \chi_1(\mathbf{X}_2) & \chi_2(\mathbf{X}_2) \end{vmatrix} \tag{7.11}$$

$$= \frac{1}{\sqrt{2}} \left[\chi_1(\mathbf{X}_1)\chi_2(\mathbf{X}_2) - \chi_2(\mathbf{X}_1)\chi_1(\mathbf{X}_2) \right]. \tag{7.12}$$

This wave function clearly changes sign if the particle positions are exchanged, and $\Psi(\mathbf{X}) = \Psi(\mathbf{X}_1, \mathbf{X}_2) = 0$ when $\mathbf{X}_1 = \mathbf{X}_2$, meaning that the wave function satisfies the antisymmetry requirement. This method is generalizable to a system of N particles and N functions;

$$S(\mathbf{X}) = \frac{1}{\sqrt{n!}} \det(\boldsymbol{\psi}), \tag{7.13}$$

where $\boldsymbol{\psi}$ is the $n \times n$ matrix consisting of all permutations of i, j in $\psi_i(\mathbf{X}_j)$. The determinant can be factorized into a spin up and a spin down matrix, i.e. two independent anti-symmetric systems;

$$S(\boldsymbol{X}) = \frac{1}{\sqrt{n!}} S_\uparrow(\boldsymbol{X}_\uparrow) S_\downarrow(\boldsymbol{X}_\downarrow), \tag{7.14}$$

where the spin up determinant (and similarly for spin down, $\downarrow$) is on the form (here in the 2×2 case)

$$S_\uparrow(\boldsymbol{X}_\uparrow) = \begin{vmatrix} \psi_1(\boldsymbol{r}_1)m_{s1} & \psi_2(\boldsymbol{r}_1)m_{s1} \\ \psi_1(\boldsymbol{r}_2)m_{s2} & \psi_2(\boldsymbol{r}_2)m_{s2} \end{vmatrix}.$$

which, because the particles in the determinant share the same spin, simplifies to

$$S_\uparrow(\boldsymbol{X}_\uparrow) = \begin{vmatrix} \psi_1(\boldsymbol{r}_1)\uparrow & \psi_2(\boldsymbol{r}_1)\uparrow) \\ \psi_1(\boldsymbol{r}_2)\uparrow & \psi_2(\boldsymbol{r}_2)\uparrow \end{vmatrix}.$$

This means that the spin factors of the spin orbitals can be factorized out of the determinants. Anti-symmetry for the system as whole is lost when factorizing into two determinants, but this is not necessarily a problem. None of the operators used in this thesis (such as the Hamiltonians) are spin-dependent, and it can be shown [64] p[39-40] that the expectation value of a spin independent operator working on the TWF is unaltered when splitting the determinant. .

7.2.2 The Jastrow Factor

The Jastrow factor is commonly also called the correlation term, as it accounts for the particle correlations as discussed in relation to Kato's cusp condition. It is typically formulated as

$$J = \exp\left(\sum_{i<j} U(r_{ij})\right), \tag{7.15}$$

where $U(r_{ij})$ is a function of the distance r_{ij} between particles i and j. As discussed previously, this exponential form ensures that the wave function cancels out the singularity associated with particles in close proximity. The Jastrow factor typically includes variational parameters, as the exact form of the correlations are usually not known. The commonly called "simple Jastrow" factor is perhaps the most straightforward inter-particle function. It can be expressed as

$$U_{ij} = \beta_{ij} r_{ij}, \tag{7.16}$$

where β_{ij} is a variational parameter. The "simple Jastrow" includes both variational parameters and meets the required shape with respect to cusp conditions. It does not however take particle spin into account. A common improvement with increased complexity is called *Padé-Jastrow*;

$$U_{ij} = \frac{\alpha_{ij} r_{ij}}{1 + \beta r_{ij}}, \tag{7.17}$$

where β is a variational parameter and a_{ij} a constant which depends on the spins of particle i and j as well as the dimensionality of the system [65].

7.3 The Neural network ansatz

Most ansatze, such as Slater-Jastrow, advance assumptions on a specific form. These are usually based on a hypothesis on the form of the wave function, which is not always easy. In order for the ansatz to be cogent, insight into the system is required. Sufficient insight is not always a given, meaning that formulating an ansatz which reflects the system to a large enough degree is not always possible. The validity of the ansatz might break down under certain conditions or limit the range of simulations, especially where high flexibility is required. The assumptions on which the ansatz is based, may prove to be erroneous to some degree or another and introduce systematic (and therefore irreducible) errors in the energy estimate. A reasonable alternative is to make no assumptions on form, and instead maximize flexibility by making use of neural network quantum states. NN quantum states involve using neural networks to represent either the entire TWF or a factor in a composite TWF. As neural networks are generally computationally intensive, this option has become more viable in the recent decades due to technological advances. In their article from 2017, Carleo and Troyer [66] suggested that one may view the wave function as a computational black box, which for a specific system configuration

outputs a value corresponding to the wave function amplitude. This approach opens up for any number of neural network applications. Feed forward neural networks was used by Saito [11] (2018) to obtain the ground states of few-body systems of bosons, using many of the same techniques as in this thesis. In an article published this year (2021) Adams et al. [13] (2021) successfully benchmarked a neural network quantum state ansatz for binding energies and point-nucleon densities against Jastrow functions and Green's function Monte Carlo in their paper. In the following section I introduce how feed forward neural networks can be used in trial wave functions, and some of the related challenges and options.

7.3.1 Feed forward neural network trial wave function

FFNNs provide high flexibility and in theory require no assumptions about the functional mapping. A feed forward neural network can hypothetically be implemented as a trial wave function rather straightforwardly by using the particle positions (and possibly all quantum numbers) as network inputs such that each degree of freedom for each particle is one input, and by defining the network output as the wave function amplitude;

$$\Psi_{FFNN} = a^L. \tag{7.18}$$

Whether such a TWF can be optimized to approximate the ground state of any given system to an arbitrary accuracy is however not certain. Artificial neural networks (ANNs), including shallow networks, are as previously mentioned, universal approximators. I make no claims on whether mapping from the coordinate space of the particles to real valued wave function amplitude is a Borel measurable function. Instead, I assume that FFNNs are capable of efficiently representing the ground state, which has been shown [67] to be the case for most states and many-body Hamiltonians. However, if the universal approximation theorem holds for the intended application, a lack of success in exact approximation can be attributed to an insufficient number of hidden units, too little training or a deficient deterministic relationship between variables [7]. In the context of VMC, only the two first factors should theoretically be an issue, as long as the dependencies of the Hamiltonian are reflected in the trial wave function. This being said, let's review some of the practical implications of using an FFNN as a TWF.

In each Monte Carlo cycle, the particle positions are used to obtain a trial wave function amplitude through the feed forward process. This can then be used to generate M values of E_L and $\nabla_{\alpha_i} \ln \Psi_T$, such that the weights and biases of the network can be optimized through minimization of $\langle \bar{E}_L \rangle$, using back propagation. This covers the first and second calculations that are required when implementing TWFs. The third and fourth quantities are based on differentiating the spatial components of the particles, which are represented as individual inputs in the ANN architecture. Because of this the desired quantities boil down to differentiating the network output with respect to input j (which is one of the coordinates of a particle[1]), or $\partial \Psi_{FFNN}/\partial x_j$ and $\partial^2 \Psi_{FFNN}/\partial x_j^2$.

[1]Referring to the particles in a simulation, not *real* particles.

Starting with the first order derivative, $\partial\Psi_{FFNN}/\partial x_j = \partial a^L/\partial x_j$, and following the notations introduced in the section on feed forward NNs

$$\frac{\partial a^L}{\partial x_j} = \frac{\partial f^L(z^L)}{\partial z^L}\frac{\partial z^L}{\partial x_j} \tag{7.19}$$

$$= f^{L'}(z^L)\frac{\partial}{\partial x_j}(W^L a^{L-1} + b_L) \tag{7.20}$$

$$= f^{L'}(z^L)W^L\frac{\partial a^{L-1}}{\partial x_j}. \tag{7.21}$$

This is similar to the thought process used for back propagation, by using the chain rule going back layer by layer. For node i in layer $l \neq 0$, with $i = 1, 2, ..., M^l$,

$$\frac{\partial a_i^l}{\partial x_j} = \frac{\partial f_i^L(z_i^l)}{\partial z_i^l}\frac{\partial z_i^l}{\partial x_j} \tag{7.22}$$

$$= f_i^{l'}(z_i^l)W_{i,k}^l\frac{\partial a_k^{l-1}}{\partial x_j}. \tag{7.23}$$

The input layer, $l = 0$, does not contain any activation functions or weights, and thus reduces to

$$\frac{\partial a_i^{l=0}}{\partial x_j} = \frac{\partial x_i}{\partial x_j} = \delta_{ij}. \tag{7.24}$$

Using the same process, the second order derivatives for $l = 1, 2, .., L$ can be expressed as

$$\frac{\partial^2 a_i^l}{\partial x_j^2} = \frac{\partial}{\partial x_j}\left(f_i^{l'}(z_i^l)W_{i,k}^l\frac{\partial a_k^{l-1}}{\partial x_j}\right) \tag{7.25}$$

$$= f_i^{l''}(z_i^l)\left(W_{i,k}^l\frac{\partial a_k^{l-1}}{\partial x_j}\right)^2 + f_i^{l'}(z_i^l)\left(W_{i,k}^l\frac{\partial^2 a_k^{l-1}}{\partial x_j^2}\right), \tag{7.26}$$

where

$$\frac{\partial^2 a_i^{l=0}}{\partial x_j^2} = \frac{\partial^2 x_i}{\partial x_j^2} = 0. \tag{7.27}$$

This covers the calculations required for a neural network to function as a TWF in VMC. There are, however, many considerations left, such as the choice of activation function for layer l, how the weights are initialized, the number of hidden layers, $L_h = L - 2$, and the number of nodes, n_h, in each layer. In my simulations I explore a few combinations of L_h and n_h and their effects on estimation quality and network stability. Based on the methods in [11] and [68], I mainly use $f^l(z) = \tanh(z)$ in the hidden layers and $f^L(z) = \exp(z)$ for the output layer. The activation for the output layer should however in some cases be modified to better reflect the problem, which I briefly discuss later by comparing simulations for two different Hamiltonian with dissimilar correlation factors in their exact ground states.

Weight initialization

The weights should in general be initialized at small random values, such that the output of the network is not too high, resulting in energy divergence. The correct scale of the random number is however dependent on the choice of activation function and number of nodes [69]. This may seem like a trivial detail, but if the weights are initiated at an incorrect scale, a network which is otherwise capable of converging on the correct underlying mapping of variables, may not converge. For $f(z) = \tanh(z)$, the most common initialization scheme is *Xavier initialization*[69], which for layer L with M^l nodes is

$$w_{ij} = \frac{1}{\sqrt{M^{l-1}}}\mathcal{N}[0,1]. \tag{7.28}$$

For other activation functions, this scheme is not as well suited. For $f(z) = ReLU(z)$ for example, this should be multiplied by a factor of $\sqrt{2}$ [69]. This paper also describes a general approach based on Taylor expansion around $z = 0$ for finding the correct scale for an arbitrary activation function differentiable at 0. By using this approach (eq.(12) in [69]), it follows that $f(z) = \exp(z)$ should be initialized using (7.28), multiplied by a factor $1/\sqrt{2}$.

FFNN in TWF for modelling correlatios vs. correction

Because an FFNN is initiated with random weights and biases, the initial network output is practically impossible to predict. This means that the energy sampling of $\Psi = \Psi_{FFNN}$ may diverge, resulting in unstable optimization conditions and in practice, an untrainable network. In his thesis on learning correlation using neural networks, Samseth [68] suggested that a more practical application of FFNNs in VMC is to use the networks in concoction with another TWF. He suggested two options for bootstrapping the network with an existing wave function. Either train the network to produce the same output as another wave function before optimizing the network in a VMC scheme, or use the network as an extra TWF factor. Because pre-training the network requires running VMC simulations beforehand, and because of possible instabilities relating to cusp conditions and symmetry requirements, he concluded that the latter of the two options was more appealing. In this thesis I use trial wave functions on the form

$$\Psi_T = \Psi_{GTWF}\Psi_{FFNN}, \tag{7.29}$$

where Ψ_{GTWF} is a "guide" trial wave function (GTWF) intended to bootstrap the TWF to non-divergent energies while the network trains. The network factor, Ψ_{FFNN}, can be used to model correlations if the GTWF is a single particle permanent. If the GTWF also models correlations, however, such as a Jastrow factor, Ψ_{FFNN} can possibly also function as a correctional factor to the GTWF.

7.3.2 Symmetry

The requirements of wave functions also apply to trial wave functions. For the most part this is uncomplicated. TWFs are generally constructed so as to be well-behaved and

continuous w.r.t. spatial parameters, as these properties are also required by the VMC scheme. However, for systems of indistinguishable particles, the symmetry requirements are not necessarily met without taking special care. In the case of systems of bosons[2], spatial wave functions, such as the ones used in this thesis, are required to be symmetric under particle exchange. In the context of using neural networks for trial wave functions, Calcavecchia et al. [3] argue that for Hamiltonians without degeneracy and real-valued eigenstates, the state with lowest associated energy is symmetric, meaning that neural networks, given sufficient optimization, should approximate the (symmetric) ground state. If the network for some reason settles on non-symmetric local minima, reference [68] suggests two possible solutions. Either to pool the output or sorting the input in order to impose symmetry on the network. The term "pooling" refers to operations similar to those used in convolutional neural networks. In the context of NNs in VMC, it involves defining the network output by a correlation order, for example by a two-body interaction, which would imply defining the wave function as a sum of all permutations of i and j of $a^L(x_i, x_j)$. This solution suffers from having to choose between computational efficiency ($O(N^2)$ in the case of two-body interaction) and limitations on correlation order the network can learn[68]. On the other hand, sorting is computationally less expensive ($O(N)$), and involves ordering the inputs to the neural network according to the distance between the particles (not the individual coordinates) and the system origin. The latter solution was used by [11] to impose symmetry on a Calogero-Sutherland type model, which features degenerate states near the ground state.

7.4 Error estimation

I have now covered how an estimate for the ground state energy can be obtained using the VMC approach, and how neural networks can be used as (part of) a trial wave function in VMC. I have, however, not yet covered how the error of the energy estimate can be estimated, which is the subject of the following section. When discussing the variational principle, I introduced the sample mean of the local energy, $\bar{E}_L$, as an estimator for the variational energy E, which is an upper bound for the ground state energy E_0. This methodology introduces two types of errors [59]:

- *Systematic error*, associated with erroneous assumptions, such as the formulation of the trial wave function.
- *Statistical uncertainty*, which is attributable to finite sampling of E_L.

The systematic error can generally not be quantified, unless if the exact solution were known. On the other hand, statistical uncertainty can be estimated, and may be used to infer some judgement on the quality of the estimate. In order to later quantify

[2] Antisymmetric ground states generally have higher energies that symmetric ground states, meaning that if an FFNN was to be used for fermionic systems, it is likely to learn a symmetric state, and consequently not the ground state of the fermionic system. This challenge could potentially be solved with Slater determinants, but this is not a subject for this thesis.

the statistical uncertainty of the energy sampling, a brief discussion of some general concepts of statistics is in order. For a random variable X with a probability density function $p(x)$, the expectation value of X is defined as

$$\mu = \langle X \rangle = \int xp(x)dx. \tag{7.30}$$

The expected squared deviation of X from μ is called the variance, and can be expressed as

$$\text{var}(X) = \sigma^2 = \langle X^2 \rangle - \langle X \rangle^2 \tag{7.31}$$

However, as is the case in VMC, when sampling X from an unknown probability density function, (7.30) is not possible to calculate. Instead, given $x_1, ..., x_n$, observations of X, the sample mean, $\bar{x}$ of x_i, can be used as an *estimator* for μ;

$$\mu = \langle X \rangle \approx \frac{1}{n} \sum_{i=1}^{n} x_i = \bar{x}, \tag{7.32}$$

assuming that x_i are independently sampled (and from the same pdf), meaning that the sample mean of x_i infers the value of $\langle X \rangle$. If the quantity of interest is instead a function of X, $F_n(X)$, an estimator for $F_n(X)$ can be defined as

$$F_n = \frac{1}{n} \sum_{i=1}^{n} f_n(x_i), \tag{7.33}$$

with expectation value

$$\langle F_n \rangle = \langle \frac{1}{n} \sum_{i=1}^{n} f_n(x_i) \rangle = \frac{1}{n} \sum_{i=1}^{n} \langle f_n(x_i) \rangle, \tag{7.34}$$

meaning that F_N and f_n have the same expectation value, as is the case for μ and its estimator, i.e. $\langle \bar{x} \rangle = \mu$. The variance of the sample mean, $\bar{x}$, can be written as [59]

$$\text{var}(\bar{x}) = \text{var}\left(\frac{1}{n} \sum_{i=1}^{n} x_i \right) = \frac{1}{n^2} \sum_{i=1}^{n} \text{var}(x_i) = \frac{\sigma_X^2}{n}, \tag{7.35}$$

assuming $x_1, ..., x_n$ are independently sampled from $p(x)$ with variance σ_X^2. This means that the variance of the sample mean, $\text{var}(\bar{x})$, decreases with more samples. When sampling from an unknown pdf, however, σ_X is unknown, but can be estimated unbiased by [70] [p. 377]

$$\sigma_X^2 \approx S^2(x) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2. \tag{7.36}$$

7.4.1 Statistical uncertainty of the energy estimate

In VMC the local energy samples $E_L(\boldsymbol{R})$ are generated using the Monte Carlo scheme to sample from the probability density function $p(\boldsymbol{R})$, which is unknown. This means that the statistical uncertainty of the energy estimate, $\bar{E}_L$, cannot be calculated, but must be inferred. Using the concepts above, an estimate for the statistical uncertainty of $\bar{E}_L$ can be expressed by the standard deviation of its Gaussian distribution by

$$\sigma(\bar{E}_L) = \sqrt{\text{var}(\bar{E}_L)} \approx \sqrt{\frac{S^2(\bar{E}_L)}{M}}, \tag{7.37}$$

meaning that the expectation value of E_L, which is the same as the expectation value of the variational energy, E, has a 68.3% probability of being within $[\bar{E} - \sigma, \bar{E} + \sigma]$. Assuming that the samples are independent of each other, the law of large numbers[3] implies that the difference between the sample mean and the true mean decreases as the number of samples increase. For a sufficiently large number of Monte Carlo cycles, this can reduce the statistical uncertainty arbitrarily close to zero as long as $\mu_{\bar{E_L}} = \langle E \rangle$, which depends on having chosen the correct variational parameters. If one further assumes that $\mu_{\bar{E_L}} = \langle E \rangle = E_0$, the statistical uncertainty of the energy samples, $\sigma(\bar{E}_L)$, can be directly compared to the variance of the ground state energy. In general, $H\Psi = E_0\Psi$ when Ψ is the exact ground state (the same holds for any eigenpair) of the system. This means that $\langle H^n \rangle = \langle \Psi | H^n | \Psi \rangle = E^n$, which also means that the exact wave function has variance of zero;

$$\text{var}(E) = \langle H^2 \rangle - \langle H \rangle^2 = E^2 - (E)^2 = 0. \tag{7.38}$$

In other words, $\sigma(\bar{E}_L)$ should be (machine precision) equal to the energy variance, which is zero, if the variational parameters are correct, the trial wave function is "perfect" and the number of cycles sufficiently large. If the parameters are not correct, then $\sigma(\bar{E}_L)$ should decrease as the variational parameters are optimized. However, as the samples of E_L are correlated due to the sampling scheme, the magnitude of $\sigma(\bar{E}_L)$ will not be correct. This means that $\sqrt{\frac{S^2(\bar{E}_L)}{M}}$ will *underestimate* $\sigma(\bar{E}_L)$ by inferring a narrower range for $\mu_{\bar{E_L}}$ than is the case.

7.4.2 The "blocking" method

When using the VMC approach, the coordinates of the simulated particles, $\boldsymbol{R}$, other than the initial, are directly dependent on the previous configuration of the system through the use of a random walker. Because of this, the associated energy samples are correlated. The uncertainty estimate in (7.37) is made with the assumption that the samples are uncorrelated, and will as such underestimate the sample spread. In order to remedy this, I have chosen to use the "blocking" or "bunching" method introduced by Flyvbjerg and Petersen [71], which benefits from being more accurate the larger the sample size, unlike for example bootstrapping. As the series of energy samples from the VMC process is ordered as a function of time, it is a time series of finite length. A time

[3] $lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} x_i p(x_i) = \mu_x$.

series of n samples can in general be used to form an n-vector or n-tuple, x, where (x_i) denotes the i'th sample in the series. .

With the goal of estimating the variance of the sample mean, var($\bar{x}$), the "blocking" method divides the time series into blocks of increasing size. By combining these transformations with theorems on strictly stationary time series, an automated "blocking" method (explained in detail in [72]) may be applied to a sample series (such as the local energies in VMC), $\boldsymbol{X}$ of length 2^p for $p \in \mathbb{Z}$. Starting with one block per sample, followed by two samples per block and so on, the sample variance, $\hat{\sigma}^2$ is calculated. As these blocks increase in size, $\hat{\sigma}^2$ of the time series gradually separates from the sample covariance. From $x_0 = x$, the subsequent "blocking" transformation, $x_0 \to x_1$, produces a new time series X_1 with length $n_1 = n/2$ [4]. The subsequent "blocking" number k continues in the same fashion, where each transformation is achieved through

$$(x_i)_k = \frac{1}{2}(x_{2i-1})_{k-1} + (x_{2i})_{k-1} \tag{7.39}$$

for $i = 1, .., n_k$, where n_k is the length of the $\boldsymbol{x}_k$. These transformations are carried out until only one block remains. The estimate for the variance of the sample mean can be expressed as $\text{var}(\bar{X}) = \frac{\sigma_k}{n_k} + \epsilon_k$, where ϵ_k is the truncation error, which after a certain number of transformations is expected to become constant. Finding the lowest k, such that $\epsilon_k = \epsilon_{k+1}$, is the goal of the last step. By calculating M (7.41), where d comes from $n = 2^d$, the smallest k such that $M_k \leq q_{d-k}(1-\alpha)$ gives the value of k to use when calculating $\hat{var}(\bar{x}) = \sigma_k^2/n_k$ to use as an estimate for the sample mean[5], with confidence interval $(1-\alpha)$. Because M_k requires both σ_k and $\gamma_k(1)$, these quantities should be calculated after each transformation.

$$\hat{\gamma}(h) = \frac{1}{n}\sum_{i=1}^{n-h}(x_i - \bar{x})(x_{i+h} - \bar{x}) \tag{7.40}$$

which is a measure of the h-order ($h = |i-j|$) covariance of a time series series with length n.

$$M_j = \sum_{k=j}^{d-1} \frac{n_k\left[(n_k-1)\hat{\sigma}_k^2/(n_k^2) + \hat{\gamma}_k(1)\right]^2}{\sigma_k^4} \tag{7.41}$$

This methodology can be applied for with a sample series of local energies in order to produce an uncertainty estimate which takes sample correlation into account;

$$\hat{\sigma}(\bar{E}_L) = \sqrt{\frac{\hat{\sigma}_k^2}{n_k}} \tag{7.42}$$

Which leads to the the automatic "blocking" algorithm [6];

[4] In the case where a time series is not divisible by two, a random sample is omitted.
[5] A value for $q_{d-k}(1-\alpha)$ may be found in a table for the χ distribution.
[6] E_i is the time series of energy samples after i "blocking" transformations.

Algorithm: The automated "blocking" method

1. Initialize algorithm
   - Set α
   - Estimate $var(\bar{E})$
   - Set $i = 0$
2. Iterative "blocking" while size of $E_{i+1} \geq 2$:
   - If $n_i \% 2 \neq 0$, remove a random energy sample from E_i
   - Compute $\hat{\sigma}_i^2, \hat{\gamma}_i(1)$
   - Transform data $\mathbf{E}_i \rightarrow \mathbf{E}_{i+1}$, and set $i = i + 1$
3. Calculate final values
   - Compute M_j from $\hat{\sigma}_j^2, \hat{\gamma}_j(i)$
   - Find smallest k such that $M_k \leq q_{d-k}(1 - \alpha)$
   - Calculate $\hat{\sigma}(\bar{E}) = \sqrt{\hat{\sigma}_k^2 / n_k}$

Part IV

Implementation and Results

Chapter 8

Implementation

8.1 QFLOW

I have chosen to use the self-contained library QFLOW[9] as the primary tool for executing Variational Monte Carlo (VMC) simulations. The library was developed by Bendik Samseth for his master thesis [68], and specifically designed for feed forward neural networks in VMC. His thesis includes a thorough and detailed explanation of the library, which is why I have chosen to only give a brief description. QFLOW is an object oriented Python package with a C++ backend, meaning that setting up the simulation can be carried out in Python while calling on C++ for the computationally expensive calculations, such as the trial wave function optimization and energy sampling. The library supplies all the basic building blocks required to execute variational Monte Carlo simulations, including Hamiltonians, trial wave function and samplers. Among these trial wave functions are feed forward neural networks and restricted Boltzmann machines, among others. Its most notable feature is however that it is set up so that it can optimize and sample from trial wave function products. This means QFLOW is well suited for simulations which use FFNNs and guide trial wave functions to construct a TWF. I have modified and added some functionality to a private branch of library, such as the Calogero-Sutherland model and some activation functions and schemes for weight initialization, but most of the core code remains unchanged.

8.1.1 Parallelization

QFLOW is written such that is can be run in parallel to a large degree with the standardized message passing interface (MPI). This allows for multiple threads to be used simultaneously, meaning that more than one process can be executed at the same time. The effect is an almost linear speed up of the VMC simulations for larger systems. Calculating the gradients used in energy optimization (7.6) and sampling local energy are two example of processes that benefit greatly from this. The a expectation values are estimated by taking the mean of M samples, which are generated by $n_{threads}$ threads. Each thread generates $M/n_{threads}$ samples before summing up the relevant quantities individually and broadcasting the summed quantities to all threads for a final sum over

$n_{threads}$. The same approach can be applied to most aspects of VMC simulations, which is why VMC is sometimes labelled as "embarrassingly trivial to parallelize".

8.1.2 Classes

Within object oriented programming, classes are templates for creating objects. Sub classes can be defined to inherit from a parent class, meaning that sub-classes will have the same functionality and properties as the parent class, but not the other way around. This is useful when several similar classes which share some traits, but still differ in some ways, are required. QFLOW is based around four base classes; Hamiltonians, Wave functions, Samplers and Optimizers, but includes other classes as well. These base classes provide an indiscriminate framework in which their sub classes can be combined in order to stage specific simulation.

Wavefunctions

The QFLOW class `qflow.wavefunctions.Wavefunction` provides functionality for returning a TWF evaluation, calculating the Laplacian, drift force, and relevant gradients of the TWF. The sub classes are specific TWF formulations, such as `qflow.wavefunctions.SimpleGaussian` ($\prod_i^N \exp(-\alpha \boldsymbol{r}_i^2)$) which inherits functionality from `qflow.wavefunctions.Wavefunction`. The sub classes are however defined in such a way that they must provide individual implementations for the functions that depend on the antsatz (defined in the methods chapter), such as the Laplacian of the TWF. This is also where the neural networks are defined, with functionality to add layers and activation functions.

Hamiltonians

The Hamiltonians provided by QFLOW are defined with the same logic as the Wavefunctions. `qflow.hamiltonians.Hamiltonian` defines functionality for returning kinetic, external and internal energy as well as optimizing a `qflow.wavefunctions.Wavefunction` object. The individual sub classes, such as `qflow.hamiltonians.HarmonicOscillator`, are required to specify the relevant potentials. When sampling for optimization or energy estimation, the Hamiltonian class uses a burn-in $0.2M$ (introduced in themethods chapter).

Samplers

`qflow.sampler` provides functionality for proposing and evaluating new configurations. A matrix which holds particle positions is passed to the sampler object, and perturbed (moved), evaluated, and returned, typically as part of energy sampling by a Hamiltonian class. The sub classes, such as `qflow.samplers.ImportanceSampler`, are required to define the associated acceptance probability, system perturbation and so on.

Optimizers

The base class `qflow.optimizer.SgdOptimizer` is used by `qflow.hamiltonians.Hamiltonian` to optimize the weights from `qflow.wavefunctions.Wavefunction`. It is, as the name suggests, an implementation of stochastic gradient descent, meaning that the sub classes are other gradient-based optimization functions, such as the ADAM optimizer [8].

8.2 Summary

VMC simulations in QFLOW use a matrix to represent the system, which contains the particle positions of N particles, $\boldsymbol{R} = \boldsymbol{r}_1, \boldsymbol{r}_2, ...\boldsymbol{r}_N$, in D dimensions. A `qflow.wavefunctions.Wavefunction` object and system matrix is passed to a `qflow.sampler` object, which populates the system matrix with initial positions for the particles. The initial positions are generated according to the specific sampler class, `qflow.samplers.ImportanceSampler` for example offsets each coordinate $\mathcal{U}[0,1]\cdot\delta t$ from the origin, while `qflow.samplers.GibbsSampler` offsets each particle $\mathcal{N}(0,\sigma)$. Energy sampling and wave function optimization is carried out by the `qflow.hamiltonians.Hamiltonian` object, calling on the sampler and wave function. The library also provides a metric for measuring the symmetry of a wave function by approximation through testing and evaluating a number of permutations.

8.3 Verifying implementation integrity

In the following section I document tests aimed at verifying that the core methods in this thesis work as intended through the qflow library. First, I use a pure isotropic harmonic oscillator potential for one particle, $N = 1$, in one dimension, $D = 1$, with Hamiltonian (3.3)

$$H = -\frac{1}{2}\frac{\partial^2}{\partial x^2} + \frac{1}{2}\omega^2\boldsymbol{r}^2,$$

with oscillator frequency, $\omega = 1.0$. The ground energy of this system (3.8) is $E_0 = 0.5$ a.u. Using the Gaussian basis set, the single-particle ansatz for the system of one boson is

$$\psi_{gaussian} = exp(-\alpha\boldsymbol{r}^2),$$

where $\boldsymbol{r} = (x\boldsymbol{i})$ is the position of the particle, and α the variational parameter. This ansatz is on the same form as the closed form stationary state (3.6). As such, $\alpha = -\omega/2 = 0.5$ should yield $\bar{E} \approx E_0 = 0.5$, and α close to 0.5 should result in local energy samples from regions close to 0.5 a.u.

8.3.1 Sampling local energy

Using both the Metropolis ($l = 1.0$) and Metropolis-Hastings ($\Delta t = 1.0$) algorithms[1] (samplers) I ran a simulation of $M = 2^{10}$ Monte Carlo cycles, using $\alpha = 0.45$ ($\alpha = 0.5$ results in exclusive sampling of $E_L = 0.5$) in order to present an easy to understand visualization of the sampling process. Figure 8.1 shows the results, and indicates that both samplers produce samples from regions close to $E_0 = 0.5$, as is required to perform the approximation $E \approx \bar{E} = 1/M \sum E_L$.

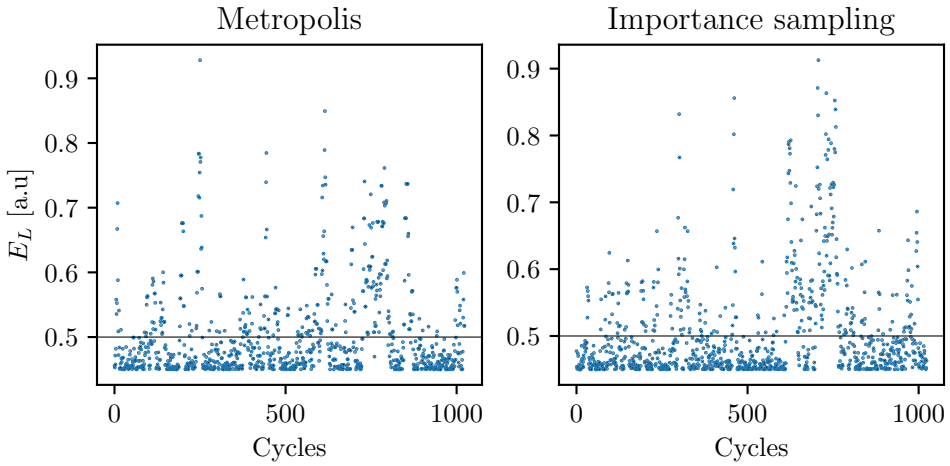


Figure (8.1) $N = 1$ particle in a $D = 1$ harmonic oscillator. Shows sampling sequence of local energies, E_L, over $M = 2^{10}$ cycles using the Metropolis algorithm (left) and Metropolis-Hastings/Importance sampling (right), produced with the singe particle ansatz $\psi_{gaussian}(\alpha = 0.45)$

8.3.2 Energy and uncertainty estimation

I reran the harmonic oscillator (HO) simulation, using $l = 1.0$ (Metropolis) and$\Delta t = 0.5$(Metropolis-Hastings) with $M = 2^{16}$ cycles, in order to verify that both sampling methods produce an energy estimate that converges towards $E_0 = 0.5$ as the number of cycles move towards M. Figure 8.2 shows $\bar{E}$ as a function of MC cycles. The figure indicates that both sampling methods produce energy estimates which converge towards E_0. It is clear that $\alpha = 0.45$ results in $\bar{E} > E_0$, which is to be expected for $\alpha \neq 0.5$ considering the variational principle and the approximations used in obtaining the estimate. In the first few thousand cycles, we see large deviations in $\bar{E}$ for both samplers. This can likely be attributed to the invalidation of the approximation $\bar{E} \approx E$ as the estimator relies on the law of large numbers.

[1]These values for l and Δt should be reasonable values to verify the implementation. An in-depth study of these sampler parameter can be found in the thesis by Flugsrud [73].

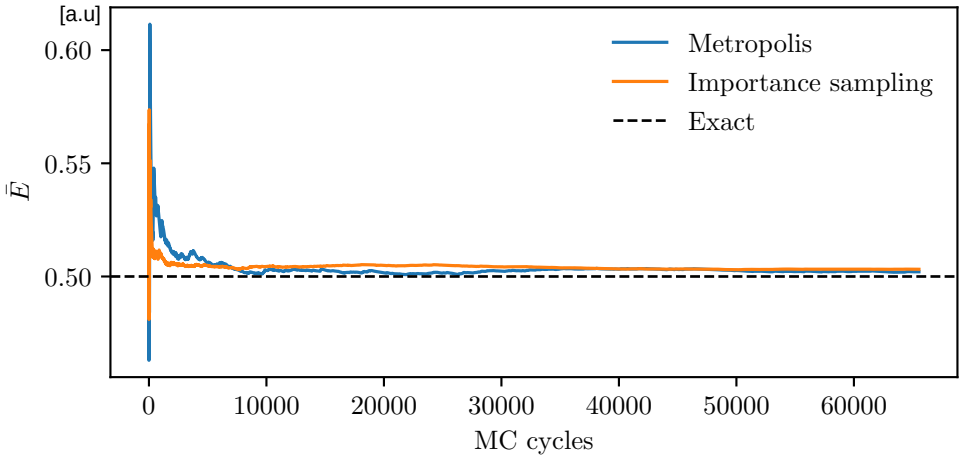


Figure (8.2) $N = 1$ particle in a $D = 1$ harmonic oscillator. Energy estimate $\bar{E}$ as a function of MC cycles (M) using Metropolis sampling with and without importance sampling for $\psi_{gaussian}$ ($\alpha = 0.45$)

Table 8.1 shows $\bar{E}$ to decimal precision dictated by the estimated statistical uncertainty both with, $\hat{\sigma}$, and without, σ, blocking. after $M = 2^{21}$ cycles. The table also includes the number of independent samples, $\hat{M}$, sampling time t[s], and acceptance rate, A. As the local energy samples are correlated, σ will be an underestimate (see section on statistical uncertainty) of the real standard deviation of the mean. Comparing $\hat{\sigma}$ for the two methods, it is also evident that a higher number of independent samples, $\hat{M}$[2], gives a lower uncertainty estimate, as is to be expected. Without importance sampling, the simulation resulted in significantly lower $\hat{M}$ and consequently higher $\hat{\sigma}$, meaning that although the simulation which used importance sampling is computationally slower ($t = 0.59$ vs $t = 0.34$), it can produce a better energy estimate. The sampling time, t, is of course dependent on the hardware and implementation. Using importance sampling, the acceptance rate, A, was higher, which is consistent with proposing changes to the system state based on configuration instead of at random. It is worth noting that the metrics are dependent on choice of l and Δt, meaning that A, $\bar{M}$ and so on can to some degree be controlled by adjusting the sampling parameters. As an example, if Δt is too large, the sampler will more frequently propose 'long' moves into regions of (relatively) low probability, meaning that the the acceptance rate will decrease.

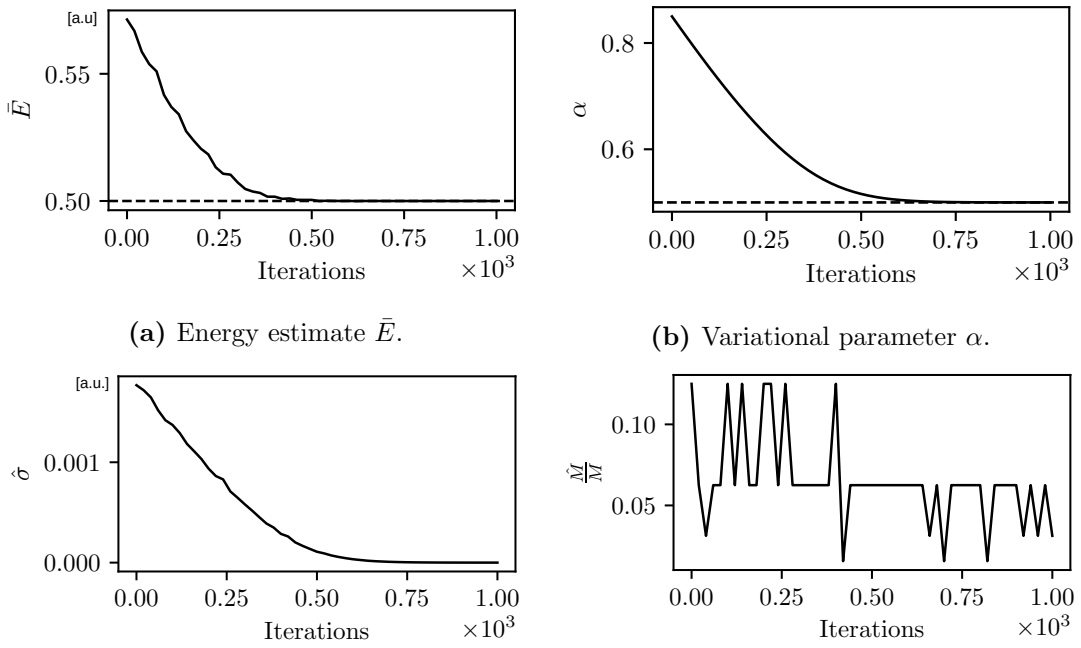
[2]The number of independent samples, $\hat{M}$, corresponds to the number of blocks, n_k, from the blocking method. See eq. (7.42) in the section statistical uncertainty.

Table (8.1) $N = 1$ particle in a $D = 1$ harmonic oscillator. Comparison of energy estimate, $\bar{E}$, standard deviation with blocking, $\hat{\sigma}$, and without blocking, σ, the number of independent samples, $\hat{M}$, total sampling time, t, and acceptance rate, A, for Metropolis sampling $l = 1.0$ and Metropolis-Hastings (importance sampling) with $\Delta t = 0.5$.

Method	$\bar{E}$ [a.u]	σ	$\hat{\sigma}$	$\hat{M}$	t (s)	A
Metropolis	0.5028	5.2×10^{-5}	0.00022	8.2×10^{3}	0.34	0.87
Imp. Sampling	0.50280	5.2×10^{-5}	7.1×10^{-5}	1.6×10^{4}	0.59	0.93

8.3.3 Optimization

With the goal of testing that the implementation indeed optimizes a trial wave wavefunction, I ran 1000 iterations (MC simulations) using 2^{16} samples per iteration, using the Adam optimizer to adjust the parameters between iterations. This was done using Metropolis-Hastings ($\Delta t = 0.5$), at an initial $\alpha = 0.85$. Figure 8.3 shows $\bar{E}$ (a), α (b), $\hat{\sigma}$ (c), and the ratio of independent samples to samples $\hat{M}/M$ (d). The figure shows convergence of $\bar{E} \to E_0 = 0.5$ as $\alpha \to 0.5$, which shows that the trial wave function was optimized with respect to α. The plots also show that $\hat{\sigma}$ strictly decreases as a function of the iterations, while $\bar{M}/M$ fluctuates in $[10^{-2}, 10^{-1}]$. $\bar{M}/M$ is lower in the second half of iterations than the first, meaning fewer independent samples and hence lower n_k. As $\hat{\sigma} = \sqrt{\hat{\sigma}_k/n_k}$, this means that $\hat{\sigma}_k$ decreased as $\bar{E} \to E_0$, suggesting that $\hat{\sigma}$ can indeed be used as an indication on how close the energy is to E_0 (see section energy and uncertatinty estimation). Using the final α value, $\alpha = 0.5000047$, I ran an additional 2^{21} cycles which produced $\bar{E} = 0.500000005(6)$ (the number in the parenthesis stand for the estimated statistical uncertainty, $\hat{\sigma}$, of the last digit in the energy estimate) and $\bar{M} = 32768$. Comparison with the values in table 8.1 (which are also from 2^{21} cycles), shows that the TWF has been optimized to a satisfactory degree. The relatively low $\bar{M}/M$ associated with $\bar{E}$ in vicinity of E_0 may be the consequence of a higher frequency of suggested moves to regions of higher energy (which are likely to be rejected) as the system occupies states close to E_0.



(a) Energy estimate $\bar{E}$.

(b) Variational parameter α.

(c) Uncertainty estimate $\hat{\sigma}$.

(d) Ratio of independent samples $\hat{M}/M$.

Figure (8.3) $N = 1$ particle in a $D = 1$ harmonic oscillator. Selected metrics (see text and sub figures a through d) during 1000 optimization iterations (2^{16} cycles per iteration) of $\psi_{gaussian}$ using importance sampling. Resolution: 50 with 2^{16} samples per data point

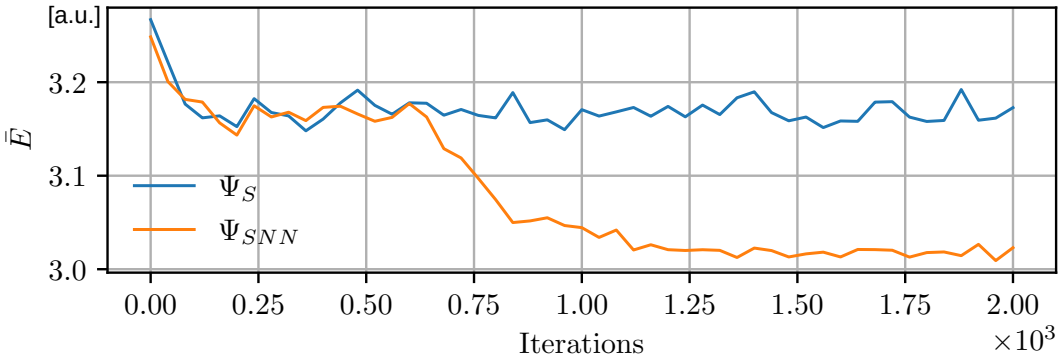
8.3.4 Feed forward neural network factor

For N non-interacting bosons in a harmonic oscillator (HO), the single particle permanent of Gaussian SPFs, $\Psi_S(\alpha)$, includes the ground state of the system (given the right value of α). As demonstrated above, optimization of $\psi_{gaussian}$ results in $\bar{E} \approx E_0$. This means that such a system is unsuited to verify if the inclusion of a neural network TWF factor, Ψ_{FFNN}, can yield improved energy estimation. Because of this, I added the repulsive Coulomb potential (3.2) (particle interaction) to the Hamiltonian. This means that Ψ_S no longer includes the ground state. Using two particles ($N = 2$) in two dimensions ($D = 2$), means that the results can be compared to the analytical solution[28] of $E_0 = 3.0$ a.u. for oscillator frequency $\omega = 1.0$. I used 2000 training iteration of 2^{16} cycles to optimize Ψ_S using importance sampling ($\Delta t = 0.5$) with the intention of having a basis of comparison for the neural network trial wave function

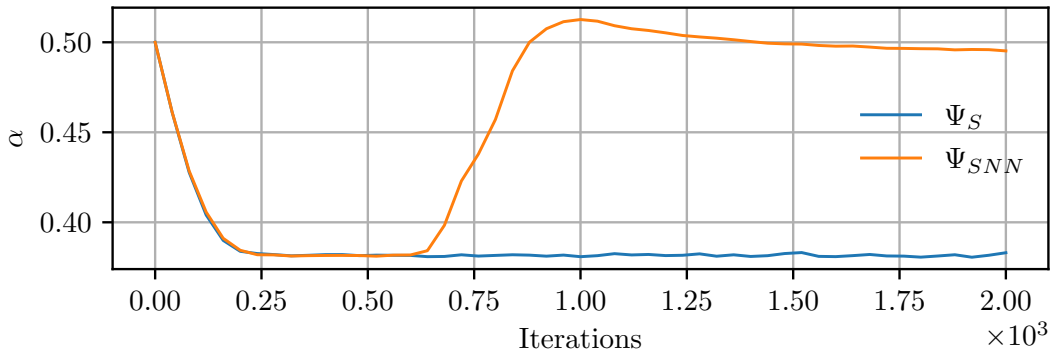
$$\Psi_{SNN} = \Psi_S \cdot \Psi_{FFNN} = \prod_i^N \exp(-\alpha \boldsymbol{r}_i^2) \cdot a^L, \tag{8.1}$$

which I from here on will abbreviate as SNN to large extent. Using the same set up as for Ψ_S (with interaction), I optimized SNN with $L = 2$, meaning one hidden layer of 12 hidden units, using the activation functions $f^1(z)$=tanh(z) and $f^{L=2}(z) = exp(z)$. The

weights were initiated with random values $\sim 10^{-43}$ in order to ensure that the network had minimal impact on the energy (avoiding divergence) and could "piggyback" Ψ_S in the initial iterations.



(a) Energy estimate $\bar{E}$



(b) Variational parameter α.

Figure (8.4) $N = 2$ interacting particles in a $D = 2$ harmonic oscillator. Energy estimates, $\bar{E}$ [a.u.] and variational parameter α from the single particle factor, from 2000 optimization iterations of 2^{16} cycles using Ψ_S and Ψ_{SNN} (see text), using importance sampling ($\Delta t = 0.5$). Resolution: 50.

Figure 8.4 shows $\bar{E}$ (a) and α (b) (parameters of FFNN not shown) for Ψ_S and SNN during the training iterations. The variational parameter of Ψ_S, α, was initially ($\sim$ 400 iterations) corrected through optimization from the starting value $\alpha = 0.5$ to $\alpha \approx 0.38$ for both TWFs, matched by a decrease in $\bar{E}$. As both TWF exhibited the same training trajectory during the first portion of iterations, it is likely that the network had little impact in, and was indeed "piggybacking" Ψ_S, which reached its full potential for optimization relatively quickly due to its lack of flexibility. After around 500 iterations, α from the SP permanent in SNN increased to $\alpha \approx 0.52$ with approximately the same rate as in the initial correction. The new training trajectory was accompanied by a gradual decrease of $\bar{E}$ at a diminishing rate. This was likely a consequence of the

[3]When I later present the results of this thesis, I use the initialization schemes described in the methods section.

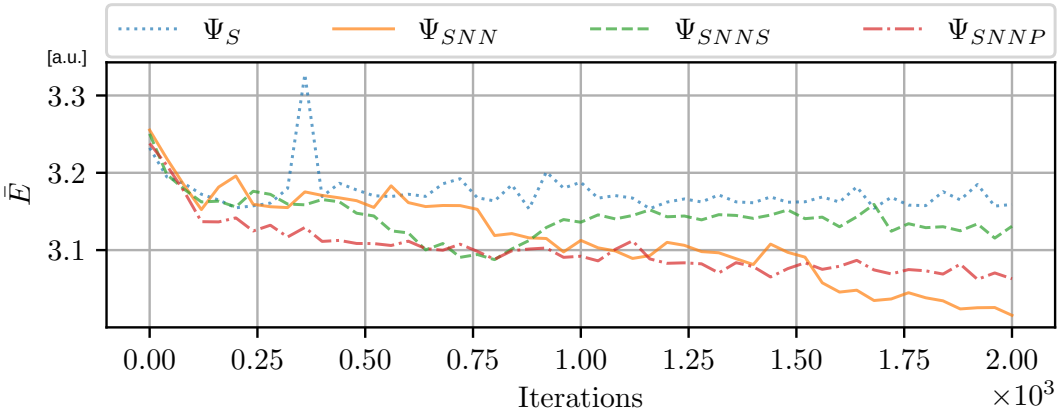
network "kicking in", meaning that the weights and biases are reaching values where Ψ_{FFNN} begins to model particle correlation, which again offsets α. After the second rapid period of training for SNN, α slowly decreased in tandem with $\bar{E}$. Table 8.2 shows $\bar{E}$, $\bar{M}/M$ and the training time (which of course is dependent on the hardware), for both TWFs based on 2^{21} samples. The table shows clearly the trends seen in figure 8.4, namely that both TWFs started off similarly, but that SNN produced a final $\bar{E}$, which was considerably closer ($\sim 10^{-2}$) to $\bar{E}_0 = 3.0$, at a large increase in computational time. $\bar{M}/M$ was substantially higher after $3k$ iterations than before optimization in the case of SNN, which is in contrast to what can we observed in figure 8.3 where it decreases as $\bar{E} \to E_0$. This could mean that the network was not able to model the TWF amplitude consistently, even though it accessed more samples of lower energies than Ψ_S, which suggests that the network was not fully optimized.

Table (8.2) $N = 2$ interacting particles in a $D = 2$ harmonic oscillator. Comparison of $\bar{E}$ in a.u., $\bar{M}/M$ and training time (s), based on 2^{21} samples before and after optimizing using 2000 iterations of 2^{16} cycles for Ψ_S and Ψ_{SNN}. See text for abbreviations.

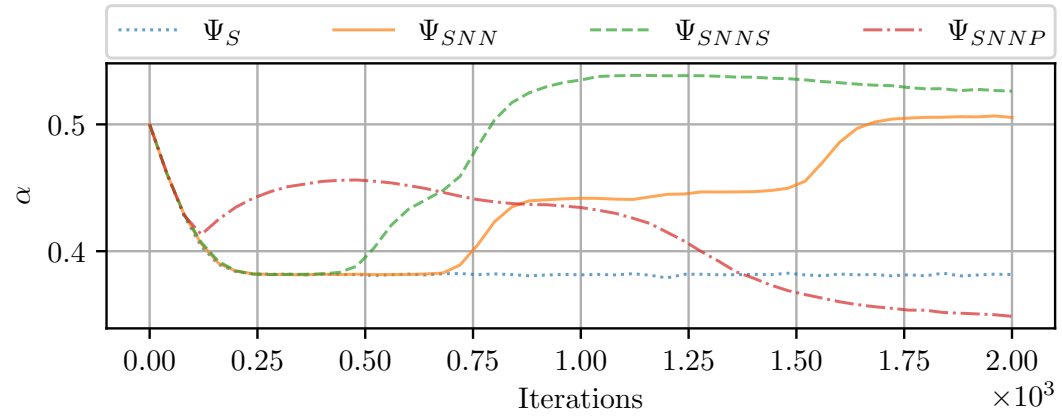
Ψ (# iterations)	$\bar{E}[a.u]$	$\bar{M}/M$	time (s)
$S(0)$	3.252(2)	0.0078125	0
$S(2k)$	3.169(1)	0.03125	70
$SNN(0)$	3.251(2)	0.03125	0
$SNN(2k)$	3.016(1)	0.25	2700

8.3.5 Symmetry

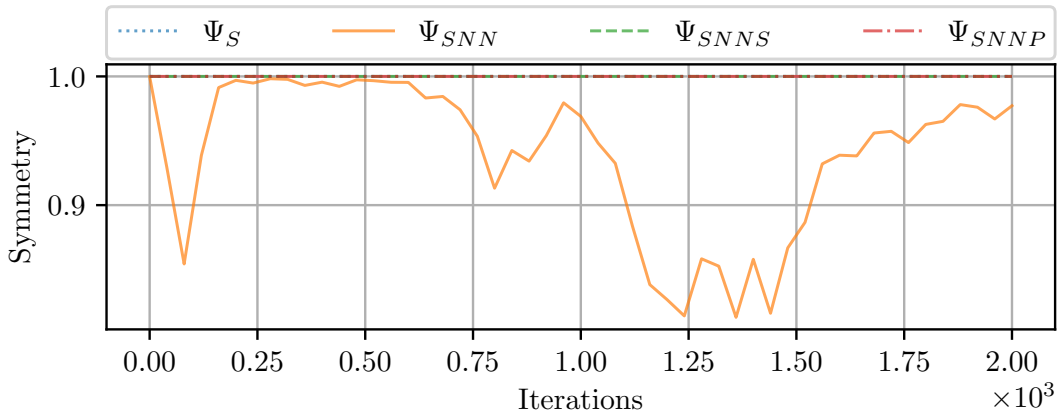
As the trial wave function has to meet the relevant symmetry requirement, it is important to verify that the trial wave function with a neural network factor is in fact capable of learning symmetry, as discussed in the methods section, for a system of bosons. As the NNs I use for the Calogero-Sutherland model, which we return to later, benefits from imposing symmetry, it is also prudent to test that these methods work as intended. I re-used the set up for the FFNN test above and ran simulations using SNN, including pooling (SNNP), input sorting (SNNS) as well as Ψ_S as a basis for comparison as it is symmetric by definition.



(a) Energy estimate $\bar{E}$



(b) Variational parameter α.



(c) Symmetry

Figure (8.5) $N = 2$ interacting particles in a $D = 2$ harmonic oscillator. Energy estimates, $\bar{E}$ and variational parameter α from the single particle permanent, shown for Ψ_S, Ψ_{SNN}, Ψ_{SNNS}, and Ψ_{SNNP} (see text), after 2000 iterations of $M = 2^{16}$ using importance sampling ($\Delta t = 0.5$). Resolution: 50.

Figure 8.5 show $\bar{E}$, α (from the Gaussian single particle basis), and symmetry measured by $S(\Psi)$ (see description of QFLOW), for the four TWFs. Ψ_S, SNNS, SNNP, exclusively scored $S(\Psi) = 1$ (this may be somewhat difficult to discern from plot (c) due to overlay, but was controlled during simulation). The plot also shows that $S(\Psi_{SNN})$ is the vicinity of 1, but not strictly 1 ($[0.8, 1.0]$), meaning that the TWF is not symmetric, but tending towards it. In the training period 250 to 600 iterations, it produced $S(\Psi) \approx 1$, which is consistent with the hypothesis that the network is practically inactive in the early stages of training due weight initialization, meaning that $exp(-\alpha x^2)$ dominated the TWF in this region. After ≈ 500 iterations, $S(\Psi_{SNN})$ decreased. A possible explanation for this is that as the network "kicks in" (i.e. weights reach threshold where the network influences the TWF sufficiently), which the trajectory of α suggests (as previously discussed). The network's lack of imposed symmetry then becomes increasingly visible as the network output grows in magnitude. After ~ 1400 iterations, $S(\Psi_{SNN})$ increased again, which indicates the network beginning to learn symmetry as a part of energy minimization, as discussed in the methods section. The experiment clearly indicated that the symmetry imposing methods work as intended, and also suggest that a TWF with a NN factor is indeed capable of learning symmetry.

8.3.6 Other observations from validation

Figure 8.5 shows that α plateaus from iteration ≈ 800 to ≈ 1500, which did not occur in the last experiment (figure 8.4), before reaching values $\alpha \approx 0.5$. This does not, however, mean that the network is not being optimized, because $\bar{E}$ decreases in the same region. The different training trajectories could be the result of unalike weight initialization, but it is also possible that the plateauing is caused by small dissimilarities in optimization due to random sampling, or a combination of the two.

Table (8.3) $N = 2$ interacting particles in a $D = 2$ harmonic oscillator. Comparison of $\bar{E}$ in a.u., $\bar{M}/M$ and training time (4 threads), based on $M = 2^{21}$ before and after 2000 iterations of $M = 2^{16}$ for Ψ_{SNN} with sorting Ψ_{SNNS} and pooling Ψ_{SNP}. See text for abbreviations.

$\Psi(iteration)$	$\bar{E}$	$\bar{M}/M$	time
$SNNS(0)$	3.251(2)	0.03125	0 s
$SNNS(2k)$	3.1228(7)	0.25	1100 s
$SNNP(0)$	3.251(2)	0.03125	0 s
$SNNP(2k)$	3.069(1)	0.25	3400 s

Table 8.5 shows $\bar{E}$, $\bar{M}/M$ and training time for $SNNS$ and $SNNP$ before and after training. These values were comparable to the ones in table table 8.2 for Ψ_S and SNN, and are therefore not shown. Final $\bar{E}$ for $SNNS$ and $SNNP$, as well as the energy trajectories in figure 8.5 suggest that both $SNNS$ and $SNNP$ have lower optimization rates than SNN. With sorting, the optimization appears to be hampered to a degree,

as $\bar{E}$ for $SNNS$ is one order of magnitude higher than for SNN and $SNNP$. Of all the four TWFs, the pooled network output, Ψ_{SNNP}, resulted in the most rapid convergence towards $E_0 = 3.0$ during the first half of the training. However, it also had a lower optimization rate than SNN after initial correction, and produced a final estimate $\sim 5 \times 10^{-2}$ higher. As table 8.3 was produced using 4 parallel threads, the time values cannot be directly compared with those in 8.2. As a basis of comparison for the symmetry imposing methods, SNN used $t = 1000$ and Ψ_s used $t = 30$. This means that although the pooling method performed decently, at least compared to the sorting method, the increased computation cost (see description of QFLOW) is evident as it used thrice as many seconds as $SNNS$ and $SNNP$ on 4 threads. All in all these simulations suggests that symmetry imposing methods should only be employed if the TWF is unable to learn symmetry, because they appear to be associated with a lower rate of training and increased computational time. As all of these observations were based on comparisons between single experiments, this is however only suggestive.

8.3.7 Assessment of implementation

The experiments in this chapter have demonstrated that the implementation is capable of energy sampling and producing an estimate close to E_0 if the TWF is sufficiently close to the ground state. The implementation was also shown to optimize TWFs such that the final energy estimate $\bar{E}$ approached E_0. The TWF with a FFNN factor, $\Psi_{SNN} = \Psi_S \Psi_{FFNN}$ (SNN), resulted in an a lower $\bar{E}$ for the interacting system of 2 bosons, than the SPF ansatz Ψ_S. This demonstrates that the trial wave function factors Ψ_{FNNN} works as intended, and is trainable when used as a correlation factor. Lastly, the verification tests also indicated that the network optimizes towards symmetry, and that the symmetry imposing methods of pooling and sorting work as intended by ensuring symmetry.

Chapter 9

Results: Optimizing neural networks with VMC

In the following chapter I study the optimization of neural networks as factors in a trial wave function by comparing multiple experiments of identical networks. I generally optimized the trial wave functions (TWFs) for $\sim$ 5000 iterations, with the intention of identifying promising candidates, both with respect to optimization and architecture for closer study using more iterations, which is done in the next chapter. The integral approximation used in VMC can sometimes lead to estimates which are slightly lower than the ground state energy, E_0. This means that lower energy estimates are not always strictly better estimates. With this in mind, I have chosen to conduct the experiments on systems to which the exact ground state energy is known. In this chapter I use a harmonic oscillator potential and Coulomb interaction (HO+C) described in the chapter on systems. For systems of $N = 2$ particles in $D = 2$ and $D = 3$ dimensions, the exact ground state energy is known for given oscillator frequencies, ω. These values[28, 29] are shown in table 9.1. In this chapter I exclusively used $\omega = 1.0$ for $D = 2$ and $\omega = 0.5$ for $D = 3$. In the next chapter I do however also investigate NN performance for the two other values.

Table (9.1) Exact ground state energy for $N = 2$ charged particles in a Harmonic Oscillator (HO) with oscillator frequency ω, interacting according to the Coulomb potential (C).

D	ω	E_0 a.u.
2	1/6	2/3
	1.0	3.0
3	0.1	0.5
	0.5	2.0

I used the Metropolis-Hastings sampling scheme, with $\delta t = 0.5$ based on the findings in [73], as the sampler can be expected to result in more independent samples than the

Metropolis sampler. For the purpose of training the TWFs, I used the Adam sampler, which is a gradient based optimization method. The neural networks (NNs) are used as factors (see methods) in a total trial wave function. The total TWFs in this chapter also include a single particle permanent of Gaussian functions, Ψ_S, such that the NN TWF can be written as

$$\Psi_{SNN} = \Psi_S \cdot \Psi_{FFNN} = \prod_i^N \exp(-\alpha \boldsymbol{r}_i^2) \cdot a^L., \tag{9.1}$$

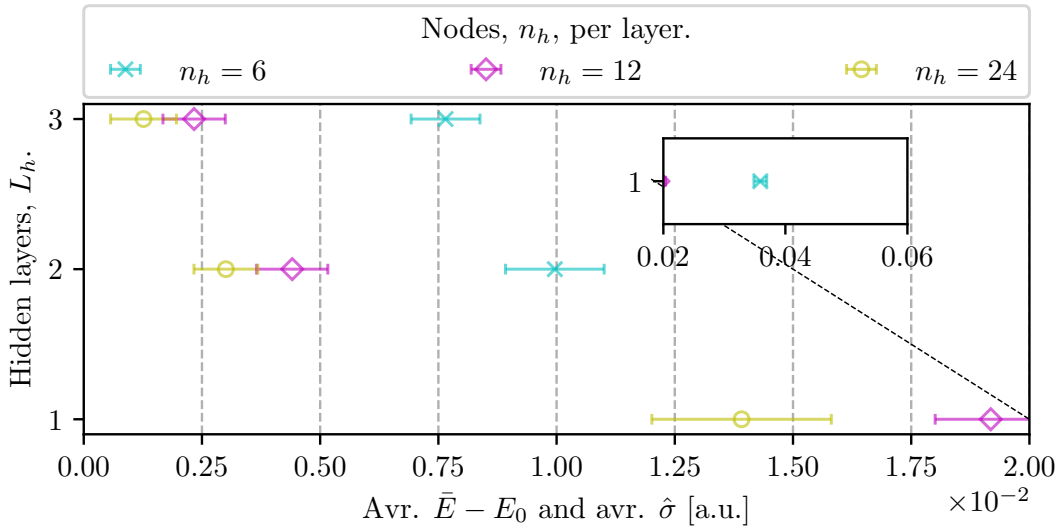
which I generally refer to as SNN. The activation functions in the hidden layers were $\tanh(z)$ and $exp(z)$ for the last layer. Instead of setting a static range for random weight initialization as was done during validation, I used Xavier initialization[69] for the hidden layers, and the corresponding range for the output activation (see methods section). Other activations were considered in the initial stages of experimentation, but none matched the performance of $tanh(z)$ and $exp(z)$. These functions were also used by [68] and [11] in their simulations involving FFNNs in VMC.

9.1 Network architecture

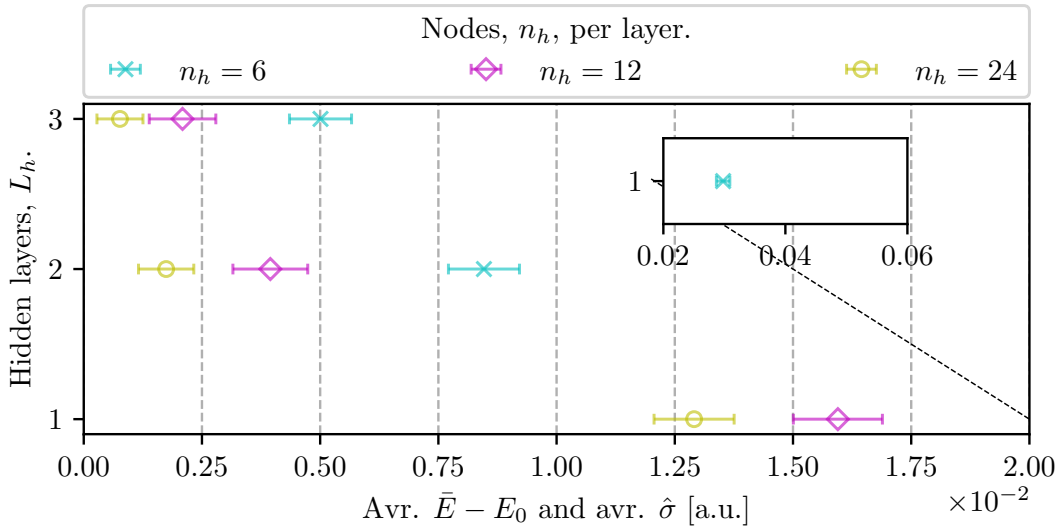
Deep neural networks, meaning networks with many hidden layers, have been shown to outperform shallow networks (see theory section on neural networks). To gauge how different NN depths and widths affect the optimization of the TWF with respect to energy estimation, I ran simulations using 9 different NN architectures. By different architectures I mean different numbers of hidden layers, L_h, and hidden nodes, n_h, per layer (identical in each hidden layer). Each architecture was applied in 10 individual experiments with different seeds [1] to ensure that each experiment resulted in different initialization and Markov Chains – and hence also independent optimization. The following results are based on 5000 training iterations, using both $M = 2^{10} = 1024$ and $M = 2^{11} = 2048$ samples per iteration in order to also discuss the effects of M.

[1] A seed is a number used to initialize a pseudorandom number generator. The random number generator will output the same sequence of numbers every time if the seed is the same, but can be expected to output different numbers for different seeds.

9.1.1 Network averages



(a) Distribution of average $\bar{E} - E_0$ with avrerage $\hat{\sigma}$ for $M = 2^{10}$



(b) Distribution of average $\bar{E} - E_0$ with avrerage $\hat{\sigma}$ for $M = 2^{11}$

Figure (9.1) $N = 2$ interacting particles in a $D = 2$ dimensional harmonic oscillator. Comparison of NN architectures for SNN. Shows the distribution of average $\bar{E} - E_0$ with average $\hat{\sigma}$ based on 10 experiments per L_h, n_h, for $L_h = 1, 2, 3$ hidden layers, with $n_h = 6, 12, 24$ nodes per layer (identical in all layers). The visualized estimates were based on 2^{21} samples after $5k$ optimization iterations of $M = 2^{10}$ (a) and $M = 2^{11}$ (b) samples per iteration.

Table (9.2) $N = 2$ particles interacting particles in a $D = 2$ dimensional harmonic oscillator. Total number of weights in a NN architecture and average difference between estimate and exact energy $\bar{E} - E_0$ (average $\hat{\sigma}$) based on $M = 2^{21}$ samples from SNN ($ND = 4$ inputs) with $L_h = 1, 2, 3$ hidden layers, of $n_h = 6, 12, 24$ nodes per layer, after training for $5k$ iterations of M samples.

L_h	n_h	#weights	average $\bar{E} - E_0$ a.u. $M = 2^{10}$	$M = 2^{11}$
1	6	30	0.036(1)	0.030(1)
	12	60	0.019(1)	0.0159(9)
	24	120	0.014(2)	0.0129(8)
2	6	66	0.010(1)	0.0085(8)
	12	204	0.0044(8)	0.0039(8)
	24	696	0.0030(7)	0.0017(6)
3	6	102	0.0077(7)	0.0050(7)
	12	348	0.0023(7)	0.0021(7)
	24	1272	0.0013(7)	0.0008(5)

Figure 9.1 shows the energy difference averaged over 10 experiments for each network, with the average uncertainty estimate $\hat{\sigma}$, for all 9 networks when optimized with $M = 2^{10}$ (a) and $M = 2^{11}$ (b) samples per iteration. Table 9.2 shows the values visualized in figure 9.1 [2].

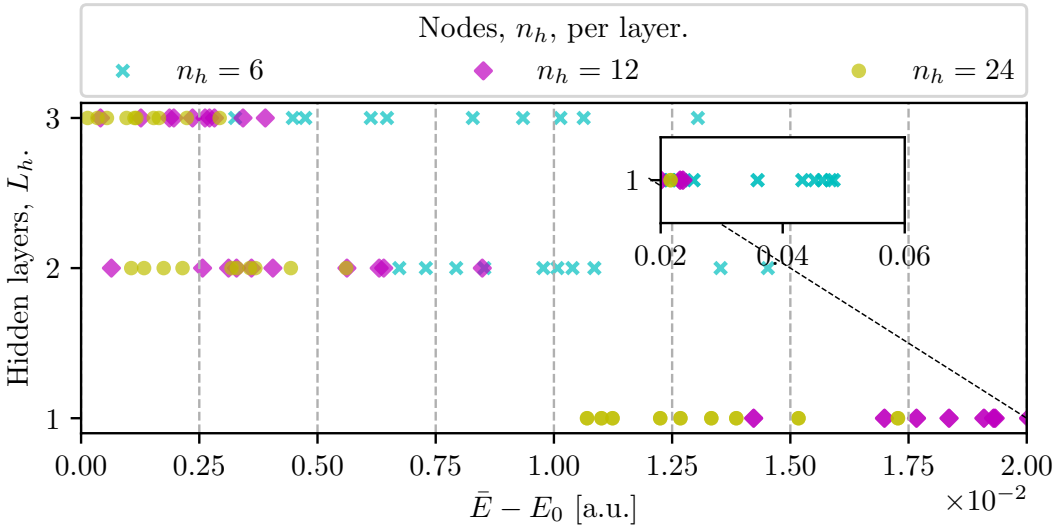
To start with, I will address the trends that can be observed in both $M = 2^{10}$ and $M = 2^{11}$ respectively. It is clear that the energy estimates relied on both the number of hidden layers, L_h, and nodes n_h. For a given L_h, average $\bar{E} - E_0$ was lower for higher n_h, and the same holds for a given n_h when L_h is increased. This general trend of more layers, and more nodes per layer, resulting in better estimates is to be expected. More layers mean more nodes, and more nodes means more weights, thus increasing the capacity of the network to correctly map variables when sufficiently optimized. In the context of NNs in TWFs, correct mapping means to associate the particle coordinates such that the magnitude of the trial wave function approximates that of the ground state. As the estimates were closer to E_0 for $M = 2^{11}$, we can assume that the networks are not fully optimized, meaning that the approximation theorem arguably does not enter into the picture. We do, however, see that more complex networks appear to optimize faster.

Although the magnitude of the average estimates appeared to rely to some degree on the number of total weights, this is clearly not the only factor. The network with $L_h = 1, n_h = 12$ has roughly the same number of nodes as $L_h = 2, n_h = 6$, but the more shallow ($L_h = 1, n_h = 12$) of the two networks produced higher average estimates by a factor $\sim 0.4 - 0.5$ depending on M. This pattern of higher estimates for shallow networks compared to deeper networks with as many or fewer weights was also observed for $L_h = 1, n_h = 24$ and $L_h = 3, n_h = 6$, as well as $L_h = 2, n_h = 24$ and $L_h = 3, n_h = 12$,

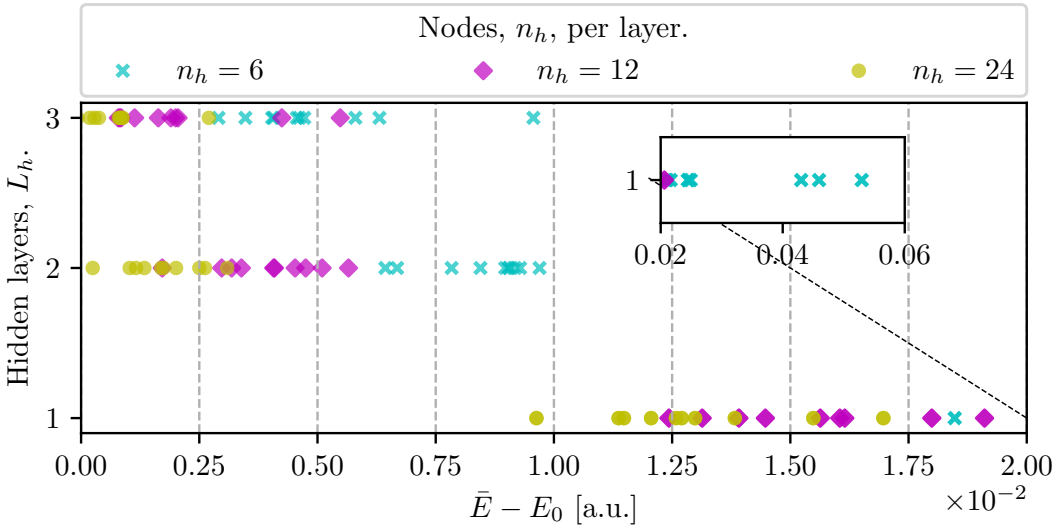
[2] Table C.1 and figure C.1 in the appendix) show the results for $D = 3$ using $\omega = 0.5$ for $M = 2^{10}$

in which case the number of weights are twice(!) as many for the shallower of the two network. However, this is not the case for $L_h = 3, n_h = 6$ (102 weights) compared with $L_h = 2, n_h = 12$ and 24 (which have 204 and 696 weights respectively). These observations suggest that some, but not all, deeper networks can be optimized such that they produce estimates which are as good, and in some cases better, than shallower networks with as many, and in some cases more weights. For a given n_h, the average difference between estimate and exact energy was approximately half (or better) as a new layer of nodes was introduced in almost all the cases. For the $n_h = 6$ architectures, this was not the case. When optimized with $M = 2^{11}$ samples, the average $\bar{E} - E_0$ produced by the networks with $L_h = 2$ and $L_h = 3$ layers of $n_h = 6$ nodes can only be said to be halved if the estimates are at the upper bounds, considering $\hat{\sigma}$, and lower bounds respectively. So, $M = 2^{11}$ is either strictly halving average $\bar{E} - E_0$, or close to, for a given n_h when adding a layer, while $M = 2^{10}$ is not. This might either indicate statistical outliers or that (comparatively) few weights in narrow networks are more difficult to optimize if the sample size is low. I will return to considerations on this when I address network stability later. For any L_h, n_h, training sets of $M = 2^{11}$ samples resulted in lower average energy estimates than for $M = 2^{10}$. Although this may seem intuitive, some of the results presented later suggests that this is not a general trend for NNs in VMC. The average of the uncertainty estimate $\hat{\sigma}$ generally decreased for larger networks, but did not improve significantly with more samples per iteration when training. Instead, average $\hat{\sigma}$ generally decreased as average $\bar{E} - E_0$ decreased. This means that for the tested architectures, no particular structure appeared to yield any advantages with respect to uncertainty estimation, other than by associated with lower energies. The uncertainty estimate is after all predominantly reliant on sample size used in energy estimation (which was 2^{21}), and the proximity of these samples to E_0.

9.1.2 Network stability



(a) $\bar{E} - E_0$ for 10 experiments per NN architecture. $M = 2^{10}$.



(b) $\bar{E} - E_0$ for 10 experiments per NN architecture. $M = 2^{11}$.

Figure (9.2) $N = 2$ interacting particles in a $D = 2$ dimensional harmonic oscillator. Comparison of architectures used in SNN. Shows distribution of $\bar{E} - E_0$ for each of the 10 experiment per L, n_h, for $L_h = 1, 2, 3$ hidden layers, with $n_h = 6, 12, 24$ nodes per layer (identical in all layers). The estimates were produced based on 2^{21} samples after $5k$ optimization iterations of $M = 2^{10}$ (a) and $M = 2^{11}$ (b) samples per iteration.

Figure 9.2 shows the 10 individual experiments for each of the 9 different NN architectures, for $M = 2^{10}$ (a) samples per iteration and $M = 2^{11}$ (b). The spread of the energy estimates for each NN is interesting for two reasons. Firstly, it gives an indication of

the convergence rate, meaning how fast the networks find a stable solution. Secondly, it can also give and indication on the robustness of neural networks as part of a TWF ansatz. Robustness is paramount as the TWF is supposed to approximate a specific ground state, which means that 'different' solutions are unwanted. It is clear that the spread of experiments were generally related to the number of nodes and the number of layers. As the number of experiments were only 10 per network, the observed spread of samples are not necessarily representative for the particular architectures, but we can still use the results as approximate guidelines.

$M = 2^{11}$ samples appeared to generally result in a smaller spread of experiments, meaning that sample size is an important factor in stability and/or convergence rate. This was however not particularly prominent for $L_h = 1$. For both values of M, the NN $L_h = 1$, $n_h = 6$ decidedly showed the largest spread of estimates of all the networks. The lowest estimates were close to the two other $L_h = 1$ architectures, while the highest were more than two times larger. This is probably because small networks lack flexibility, causing small adjustments to a single weight to have a large impact on the output, leading to instability while optimizing. This observation holds for both values of M, as well as in the $D = 3$ case shown in figure C.1 (b) in the appendix). As $L_h = 1, n_h = 24$ (120 weights) had a larger spread of estimates than $L_h = 2, n_h = 6$ (66 weights), this is clearly not just a matter of weights, but also of shallowness. Based on of this, I believe that when using narrow single layer networks in TWFs for VMC, special care must be taken when optimizing and sampling. This was also observed to be the case for deep networks (as I briefly discuss in the next chapter), when the number of particles was high.

The cases where a network has fewer weights but more layers than others are particularly interesting in the discussion on average energy estimation. Comparisons between these architectures may also give some indication on depth versus width with respect to convergence rate. $L_h = 3, n_h = 6$ (fewer than $L_h = 2, n_h = 12, 24$) and $L_h = 3, n_h = 12$ (fewer than $L_h = 2, n_h = 24$ and $L_h = 1, n_h = 24$), as well as $L = 2, n_h = 6$ (fewer than $L = 1, n_h = 24$), are therefore particularly worth examining. The narrowest of the deepest networks, $L_h = 3, n_h = 6$, does not appear to have a particularly smaller spread than than either $L_h = 2, n_h = 12$ or $L_h = 2, n_h = 24$. This observation also holds for $L_h = 3, n_h = 12$ compared to the shallower $L_h = 2, n_h = 24$ for $M = 2^{10}$, but not necessarily $M = 2^{11}$. When optimized with more samples per iteration, $L_h = 3, n_h = 12$ resulted in two estimates with comparatively high values – even higher than the highest produced by the same network when using $M = 2^{10}$. Outlier values can also be observed for the other $L_h = 3$ networks when the number of samples was increased. This may be a matter of exploitation versus exploration (which I discuss later), in combination with random weight initialization and/or random sampling. The narrowest $L_h = 2$ network with $n_h = 6$ hidden nodes is largely comparable to the $L_h = 1, n_h = 12, 24$ networks for $M = 2^{10}$, and appears smaller than the $L_h = 1$ networks for $M = 2^{11}$. This could be related to the general lack of robustness observed in all $L_h = 1$ NNs, and the observation that the spread of estimates appear to decrease for the more complex networks as training sample size increases. Considering all these "special cases", it would appear that the share number of weights is the most important factor regarding NN robustness.

9.2 Effects of single particle factor

The total trial wave function $\Psi_{SNN} = \Psi_S \Psi_{FFNN}$ includes variational parameters from both wave function factors; α from $\Psi_S(\alpha)$ and the weights and biases of $\Psi_{FFNN}(\{\boldsymbol{w}^l, \boldsymbol{b}^l\}_{l=2}^{l=L})$. As $\Psi_S(\alpha)$ is a single particle (SP) permanent for an HO, the optimal value of α for a given oscillator frequency ω is easily obtained (see the validation chapter), meaning that the SP permanent can be expressed exactly. The same goes for many other quantum mechanical systems, and in those cases where the SP contribution is not known exactly, it can often be numerically determined with good precision by use of for example VMC. When optimizing a total TWF for a system with particle interaction that includes both an SP permanent and a correlation factor we can chose to optimize all of the parameters simultaneously. An alternative is to remove the parameter(s) that are known exactly (or approximately, from the optimization scheme and fix it at the 'optimal' value. In the following section I explore how the optimization of a TWF with an FFNN correlation factor is affected by fixing or not fixing the SP permanent. In order to have a basis for comparison, I also use a TWFs on the form

$$\Psi_{SPJ} = \Psi_S \Psi_{PJ} = \prod_i^N \left(\exp(-\alpha \boldsymbol{r}_i^2)\right) \cdot \exp\left(\sum_{i<j} \frac{\alpha_{ij} r_{ij}}{1+\beta r_{ij}}\right) \tag{9.2}$$

where Ψ_{PJ} is a Padé-Jastrow correlation factor with variational parameter[3] β. The TWF Ψ_{SPJ} is a simple ansatz in the sense that it only has two variational parameters. Because of this, the TWF can be expected reach an effective optimization limit after relatively few iterations. As Ψ_{SPJ} is often regarded as the standard VMC approach, I will sometimes refer to it as SPJ. The distinction between the TWF and the methodology of variational Monte Carlo should be clear from the context.

Table (9.3) N = interacting particles in a $D = 2$ dimensional harmonic oscillator. Average $\bar{E} - E_0$ with average $\hat{\sigma}$ (10 experiments per TWF) based on 10 of $M = 2^{21}$ samples using the total TWFs $\Psi_{SPJ} = \Psi_S(\alpha)\Psi_{PJ}$ (SPJ) and $\Psi_{SNN} = \Psi_S(\alpha)\Psi_{FFNN}$ (SNN) with $L_h = 3$ hidden layers of $n_h = 12$ and 24 hidden nodes, after $5k$ iterations of $M = 2^{10}$ training samples (Metropolis-Hastings $\delta t = 0.5$). The total TWFs were trained with the single particle permanent being either [a] optimized, [b] fixed at 0.5 or [c] fixed at 0.55.

Total TWF	n_h	Average $\bar{E} - E_0$ a.u.		
		[a] α optimized	[b] α fixed (0.5)	[c] α fixed(0.55)
VMC		0.00127(4)	0.00147(4)	0.0092(2)
SNN	12	0.0023(7)	0.0013(6)	0.0030(6)
SNN	24	0.0014(7)	0.0008(4)	0.0021(7)

[3] α_{ij} in (9.2) is not a variational parameter, but depends on particle spins[65], for $N = 2$ particles, the correlation of two electrons with unalike spin will be similar to that of two charged bosons, meaning that $\alpha_{ij} = (1/D - 1)$.

In table 9.3 we can see that SPJ resulted in energy estimates marginally closer to the ground state energy (by a factor ~ 0.9) when the variational parameter α from the SP factor was optimized, and not fixed at exact value. Figure C.2 in the appendix, shows that Ψ_{SPJ} (SPJ) optimized slightly faster when α was not fixed, but that both approaches resulted in practically identical TWFs after less than 1000 iterations of $M = 2^{10}$ samples. As such, I assume that SPJ reached its effective optimization limit before 5000 iterations, and that it is largely unaffected by whether the SP contribution is fixed or not, due to the simplicity of the total ansatz. In the case of SNN, table 9.3 shows that both networks (three layers of $n_h = 12$ and $n_h = 24$) produced significantly lower estimates when the SP permanent was fixed. Compared to the values in column [b], the average estimates for the TWFs with NNs in column [a] were closer to the ground state energy by a factor ~ 0.6. This means that when the correlations were modelled with an NN, fixing the exact single particle contributions of the TWF resulted in improved estimates, which was not the case when the correlations were modelled with a Jastrow factor. As the Jastrow factor makes explicit assumptions on the correlation form and has limited flexibility, it is reasonable to assume that it introduces irreducible systematic errors to the TWF (see methods chapter) which could explain why it cannot be arbitrarily optimized using a given sampling method. The networks in SNN on the other hand, especially $n_h = 24$ due to large number of weights, is unlikely to be near its effective optimization limit, which is confirmed in the results presented later. By isolating the SP contribution from the optimization scheme, the network is forced to exclusively minimize energy by modelling correlations. When α is not fixed, this parameter is likely trained towards values far from the exact one in the initial stages of optimization to compensate for the untrained network (as can be observed in figure 8.4 in the implementation chapter), which could possibly lead to a pendulum effect in the initial iterations of the optimization. Because the parameters of the network are gradually optimized by changing the weight and bias values through gradient descent, it would make sense that the network optimizes slower if the SP contribution changes during training, which would explain why the best estimates were obtained with α fixed at the exact value.

9.2.1 Correcting systematic errors in SP factor

Table 9.3 show that all tested TWFs produced significantly higher average estimates when α was fixed at an inaccurate value (column [c]), which is not surprising. The magnitude of the TWF evaluated at a newly proposed state (meaning spatial configuration of the system) $\boldsymbol{R}*$ largely decides whether $\boldsymbol{R}*$ is accepted or not. Because of this, any systematic errors, such as an inaccurate SP permanent, will over time push the random walker out of the regions with the lowest local energy, unless corrected by other factors in the TWF. When α was fixed at 0.55 (column [c]), the resulting average $\bar{E} - E_0$ was ~ 6.3 larger than the lowest values produced by SPJ. The corresponding factor for SNN with $n_h = 12$ was ~ 2.3 and ~ 2.6 with $n_h = 24$. The Jastrow term assumes a specific functional form and has only one variational parameter, which limits its expressive power. The comparatively large difference between columns [a],[b] and [c] for SPJ could

very well be a consequence of this, as the Jastrow factor is unable to effectively correct the systematic error introduced in the inaccurate SP permanent. The neural networks however have high expressive power and flexibility, which means that it can more easily model both the correlations and correct the errors in the total TWFs. As such, table 9.3 suggests that NN are especially relevant in cases where the single particle states of the system must be approximated. The results were similar for 3 dimensions, see figure C.3 in the the appendix, albeit with less instability, which can most likely be attributed to more weights in the first hidden layer.

9.2.2 Uncertainty estimates

The estimated uncertainty of $\bar{E}$ (and consequently $\bar{E} - E_0$), $\hat{\sigma}$ was one order of magnitude lower for SPJ compared to SNN, even when the average estimates from SPJ were significantly higher. Column [c] is an exception to this, but this is to be expected as it represent conditions for which the Jastrow factor is particularly unsuited, and even in this case, the average $\hat{\sigma}$ was lower than those of SNN. This is probably because SPJ maps $\boldsymbol{R}$ to TWF magnitude very concisely, even with little training, due to its simplicity. This means that whenever a move to regions of higher local energy is accepted the random walker will quickly find its way back. The neural networks however are likely to be less consistent as they are unlikely to have converged after 5000 iterations. Because the networks are a lot more complex than the Jastrow factor, they can be expected to have high variance before convergence. As such, SNN will tend to produce less homogeneous sample data and accordingly higher a higher standard deviation, $\hat{\sigma}$, even if the majority of samples have lower energies than those from SPJ.

9.3 Exploitation versus exploration

Table (9.4) HO+C ($N = 2$, $D = 2$). Average $\bar{E} - E_0$ with average $\hat{\sigma}$ from 10 experiments per TWF. Each estimate is based on $M = 2^{21}$ samples (Metropolis-Hastings $\delta t = 0.5$) using the total TWFs $\Psi_{SPJ} = \Psi_S(\alpha)\Psi_{PJ}$ (SPJ) and $\Psi_{SNN} = \Psi_S(\alpha = 0.5)\Psi_{FFNN}$ (SNN) with $L_h = 3$ hidden layers of $n_h = 12$ and 24 hidden nodes.

Total TWF	n_h	$\bar{E} - E_0$ a.u.			
		5k iterations		8k iterations	
		$M = 2^{10}$	$M = 2^{11}$	$M = 2^{10}$	$M = 2^{11}$
SPJ	-	0.00127(4)	0.00126(4)	0.00128(4)	0.00128(4)
SNN	12	0.0013(6)	0.002(1)	0.0011(7)	0.001(2)
SNN	24	0.0008(4)	0.0005(5)	0.0002(5)	0.0004(5)

Table 9.4 shows that the average estimates were approximately the same whether SPJ was optimized $5k$ or $8k$ iterations, or whether those iterations consisted of $M = 2^{10}$

samples or $M = 2^{11}$. This strongly suggests that SPJ reached its effective optimization limit after $5k$ iterations of $M = 2^{10}$ (or before that). When optimized with a $\delta t = 0.1$, SPJ resulted in estimates one order of magnitude lower, which is shown in the next chapter. The neural networks were however largely unaffected by decreasing δt to 0.1 (for this particular system).

SNN ($L_h = 3, n_h = 12$) did not show any improvements going from 5000 to 8000 iterations. This narrow network did however result in higher average variance as the number of iterations, and the number of samples, were increased. The wider network of $L_h = 3, n_h = 12$ on the other hand seem to benefit from more iterations, but did not show any improvement moving from $M = 2^{10}$ to $M = 2^{11}$ samples when trained for 8000 iterations. In other words, both the wide and the narrow networks resulted in higher estimates (with variance taken into considerations) when trained with more samples after 8000 iterations. To examine this further, let us now inspect the individual experiments.

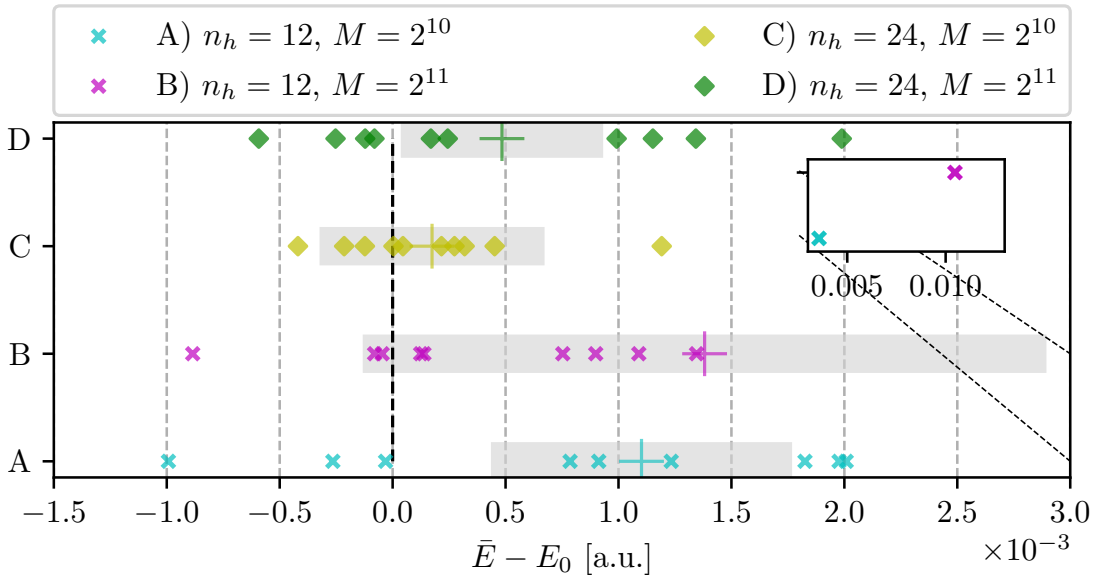


Figure (9.3) HO+C, $N = 2$, $D = 2$. Distribution of $\bar{E} - E_0$ for 10 experiments per per architecture and sample size, M, using SNN where the SP permanent was fixed at exact value, and where the networks had $L_h = 3$ hidden layers of n_h nodes per layer. The estimates are based on $M = 2^{21}$ samples after $8k$ iterations of $M = 2^{10}$ and $M = 2^{11}$, using Metropolis-Hastings($\delta t = 0.5$). Averages are represented with '+' with average $\hat{\sigma}$ indicated in gray.

Figure 9.3 shows the distribution of $\bar{E} - E_0$ for the experiments that the average estimates for SNN in the 8000 column of table 9.4 are based on. For the narrow network (teal and cyan in figure 9.3), most of the experiments resulted in estimates closer to E_0 when M was increased, with the exception of the outlier at ~ 0.01 $\bar{E} - E_0$ a.u. The outlier increased the average estimate considerably, as it was two order of magnitude

larger than the majority of the outcomes for this network. This was likely also the cause of the observed increase in $\hat{\sigma}$. For the wider network (yellow and green), the majority of the estimates moved further from E_0 as M was increased. This means that the networks, in contrast to the Jastrow factor, did not reach an effective optimization limit. It does however show that the networks are sensitive to the sample size when optimizing, and not exclusively in a positive way with respect to energy estimation. The Markov chain produced during one MC iteration is likely to contain more samples with a low acceptance probability when M is increased. Because of this, larger values of M when optimizing can arguably be described as increased exploration. Usually, exploration is associated with finding promising new 'solutions' that the model is not yet optimized towards. In VMC this translates to finding possible energy minima which have lower probability because the TWF is currently favouring other minima. In this specific case, it does however seem that increased exploration either hindered or slowed down the optimization of the largest of the two networks. A possibility is that the wider network is training towards mapping a larger range of values when M is higher, which would indeed slow down optimization. I examine this further in the next chapter. The pay off may be increased accuracy in the ground state, and especially in the tails, meaning regions of relatively low probability. Consequently, it would seem that greedy exploitation, meaning few samples per iteration, results in faster convergence towards E_0. This may however not be the case for more complex systems or systems with more particles.

9.4 Validity and summary

The discussions in this chapter were largely based on observation from simulations of 5000 to 8000 iterations of $M = 2^{10}$ and $M = 2^{11}$ samples. This optimization was clearly insufficient for the larger networks to converge, which means that the findings may not be applicable if the number of iterations or samples is much higher. The observations should however be more representative for the initial optimization of such networks. Although the total number of experiments in this chapter was relatively high, the number of experiments per network are too few to decisively confirm the observations. This is especially true for the observations regarding robustness, as by-eye comparison is hardly precise. However, as many of observed trends are based on patterns which are recognizable for more than one network, their validity is arguably strengthened, especially relating to the observations on estimation averages and effects of the SP permanent, due to the concise quantities. The number of possible architectures are virtually endless, especially considering choices of the activation functions. Other choices in this regard may very well result in different outcomes. The networks may also optimize differently for larger and/or more complex systems.

The most substantial assessment with regards to obtaining energy estimates closer to E_0 is that deeper and narrower networks appeared to result in better trial wave function optimization, compared with shallower and wider networks with as many, or more, weights. $L_h = 3$ layers of $n_h = 12$ nodes per layer (348 weights total) and

$L_h = 2, n_h = 24$(696 weights), resulted in approximately the same average difference between estimated energy and exact ground state energy ($\bar{E} - E_0 \approx 0.2 \times 10^{-2}$ a.u.) when considering the uncertainty estimates. The stability of TWFs with $L > 1$ networks was observed to rely mostly on number of weights. It follows that a network used in VMC calulations may be put to best use when the it is deep and narrow, rather than shallow and wide. All of the networks with 3 hidden layers (except the most narrow of them), were however observed to produce outlier and/or increased spread of estimates when M was increased from $\sim 1k$ to $\sim 2k$. This suggests that the networks converge faster towards E_0 when exploitation is emphasized over exploration. TWFs with a neural network factor ($L_h = 3, n_h = 12, 24$) were also observed to produce lower estimates (by a factor ~ 0.6) when the SP permanent was fixed at a pre-trained value. This suggests that neural networks also can be used to correct pre-trained TWFs which have converged, and may possibly be used to improve ground state energy estimates from other methods.

Chapter 10

Results: Ground state energy estimation using neural networks

In the previous chapter I compared the outcome from multiple experiments using the same simulation set up for relatively few iterations (5000). We saw that the neural networks (NNs) had not converged, as the estimate were sensitive to increased optimization and because we observed many outlier values when estimating energy. In this chapter, I present results from single experiment simulations involving more iterations while utilizing some of the assessments made earlier. Four different architectures were used, one *wide* and less deep, one *deep* and narrow, one *balanced* – meaning of intermediate depth and width, and one *funnel* 'shaped' – meaning that it has more weights in the first layers than in the subsequent layers. These networks all have $\sim 600 - 700$ weights, and use the same initialization scheme and activations as in the previous chapter. The exact architectures can be found in table C.2 in the appendix. The following holds unless stated otherwise; symmetry was measured to be at least 0.99 (where 1.0 means full permutation symmetry) for all the presented estimates. All the estimates are based on $M = 2^{22}$ samples, using the Metropolis-Hastings sampler ($\delta t = 0.5$), which was also used when training. Optimization was done using the ADAM optimizer. For $N = 2$ particles and 30k iterations, the *CPU time* (total time to run the simulation on a central processing unit) was 2 to 4 hours depending on the number of dimensions.

10.1 Correlation modelling versus correcting TWFs

In the previous chapter we observed that SNN resulted in estimates closer to E_0 when the single particle permanent was fixed at the exact value. This can be regarded as correcting the SP ansatz, which does not model particle correlation. This approach did not results in significantly better estimates after $30k$ iterations, as can be observed in table 10.1 for $D = 2$ dimensions. In the case of $D = 3$ however (table 10.2), all architectures resulted in better estimates, mostly by one order of magnitude, when the NNs were used to correct the SP permanent. As $D = 3$ represents two extra inputs, it is likely that the networks require more iterations before learning to associate the coordinates of the

particles. This means that the initial pendulum effect from the unfixed SP permanent (discussed in the previous chapter) may be more persistent. Going from $D = 2$ to $D = 3$ also increases the number of weights by a factor ~ 0.05 to 0.15. All in all, this means that the networks may require more iterations before covering when $D = 3$ compared to $D = 2$, which could mean that the 'unfixed' TWFs have not caught up with the fixed TWFs.

Table (10.1) Harmonic oscillator with interaction, $N = 2$ particles in $D = 2$ dimensions. Difference between energy estimate and exact ground state energy, $\bar{E} - E_0(\hat{\sigma})$ [10^{-4} a.u.] for the following TWFs; $\Psi_S(\alpha)\Psi_{FNN}$(SNN), $\Psi_S\Psi_{PJ}$ (VMC), and $\Psi_S\Psi_{PJ}\Psi_{FFNN}$(NN+SPJ). See text for details on architectures. Subscript f indicates that the TWF factor was fixed at pre-trained values and not optimized. Estimates are based on $M = 2^{22}$ samples after $30k$ iterations of $M = 2^{10}$ samples.

TWF Ψ	$\bar{E} - E_0$ [10^{-4} a.u.]				
		Network architecture			
		Wide	Balanced	Funnel	Deep
SNN		−8(4)	2(5)	64(3)	−3(3)
S_fNN		−8(2)	−3(2)	−10(2)	−4(6)
VMC	64(5)				
NN+SPJ		−0.00(1)	0.18(1)	0.075(9)	0.09(9)
NN+SPJ$_f$		0.01(3)	0.05(3)	0.07(2)	0.10(3)

Table (10.2) Harmonic oscillator with interaction, $N = 2$ particles in $D = 3$ dimensions. Difference between energy estimate and exact ground state energy, $\bar{E} - E_0(\hat{\sigma})$ [10^{-4} a.u.] for SNN, VMC, NN+SPJ, see figure 10.1 for abbreviations and text for details on architectures. Subscript f indicates that the TWF factor was fixed at pre-trained values and not optimized. Estimates are based on $M = 2^{22}$ samples after $30k$ iterations of $M = 2^{10}$ samples.

TWF Ψ	$\bar{E} - E_0$ [10^{-4} a.u.]				
		Network architecture			
		Wide	Balanced	Funnel	Deep
SNN		0.9(3)	1.1(4)	1.0(3)	0.7(3)
S_f NN		0.5(3)	0.4(3)	0.2(3)	0.4(3)
VMC	1(2)				
NN+SPJ		0.07(1)	0.09(1)	0.08(1)	0.08(1)
NN+SPJ$_f$		0.8(3)	0.5(3)	0.4(3)	0.5(3)

I also applied the method of 'fixing' to TWFs comprising of a SP permanent and

Padé-Jastrow factor (SPJ)[1]. When NNs were applied to correct SPJ_f (the pre-trained and fixed ansatz), the result was, however ,estimates further from the exact energy, than when SPJ was not fixed, especially for $D = 3$. The addition of the Padé-Jastrow factor is likely to reduce the pendulum effect (discussed in the previous chapter) as SPJ optimizes swiftly and clearly separates SP contributions from correlations. This also means that the NNs are bootstrapped to a solution which is already relatively minimized with respect to energy, meaning that the networks 'observe' samples from regions of high probability much earlier, and may therefore also optimize faster. Comparing SNN and NN+SPJ (both fixed and not), it is clear that NN+SPJ generally resulted in lower estimates, especially for $D = 2$. This can probably be partially contributed to the considerations I have already discussed for SPJ. Another important factor is that the networks in NN+SPJ are not required to model correlations. Instead, the networks optimize towards correcting SPJ. This means that the output range of the NNs is likely to be a lot narrower, and only really important when the particles are very close together, as this is when the (interactive) potential is strongest. This means that the expressive range of the network can be tuned towards a smaller numerical region which should give better accuracy when 'predicting' TWF magnitudes. For each TWF and number of dimensions, the different architectures largely resulted in estimates of the same order of magnitude with a few exceptions. The funnel network contributed the most to these exceptions, followed by the balanced network. I will not attempt to speculate on any particular reason for this, but it worth noting that both of these architectures had slightly more weights than the two others. Because of this, and in order to reduce the amount of data and simulations, I have chosen to only use the 'Wide' and 'Deep' networks for the remainder of the thesis.

10.1.1 Optimization

Increasing the number of iterations from 30000 to 40000 resulted is a slightly lower energy $\bar{E} - E_0 = -0.038(4) \times 10^{-4}$ a.u. for NN+SPJ (deep) compared to 0.09(9). The biggest pay off was however in the estimated uncertainty, which was reduced by one order of magnitude (from $\sim 10^{-6}$ to $\sim 10^{-7}$).

[1]The ansatz SPJ (and only SPJ) was optimized using $\delta t = 0.1$, not $\delta t = 0.5$ as this resulted estimates one order of magnitude lower. This goes for all estimates and figures for SPJ in this chapter.

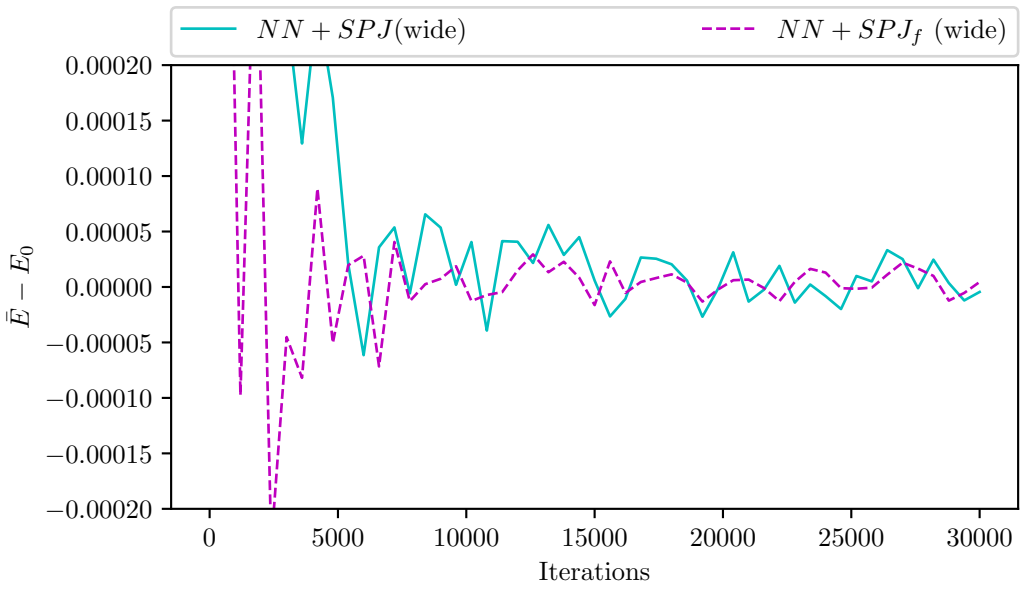


Figure (10.1) Convergence of between estimated and exact ground state energy, $\bar{E} - E - 0$, with respect to training iterations for both fixed and unfixed SPJ factors multiplied by a wide neural network. The values are based on $M = 2^{15}$ samples. Resolution 50 data point per 30000 iterations.

Figure 10.1 shows the optimization of "wide" networks when the SPJ factor was fixed, and when it was not fixed. When SPJ was fixed, the total TWF sampled energies close ($\sim 10^{-4}$) to E_0 after fewer iterations than when SPJ was not fixed, which is consistent with the earlier findings. After $\sim$ 6k iterations, the fixed and unfixed approaches did however largely follow the same trajectories.

10.2 Particle densities

Up to this point, I have mostly considered ground state energy estimation. In order to both discuss the system(s), and effectively discuss some previous observations, such as the one on exploitation versus explorations, I would now like to focus on particle densities. This provides a means of visualizing the approximated ground state, rather than just the energy. The one body density (OBD), $\rho(r)$ corresponds to finding a particle at a distance r from the origin. Computationally, this is done by dividing the spatial distance of the system into bins[2], then counting the number of particles in each bin while the system is perturbed before normalizing to 1(one). Similarly, the two-body density (TBD), $\rho(r_i, r_h)$ corresponds to the probability of finding particle i at a distance r_i from the origin given r_j. In contrast to the OBD, TBD explicitly visualizes how the particles are distributed in relation to each other. However, as the the TBD is of a

[2]I use $M = 2^{27}$ samples distributed in 200 and 200^2 bins for OBD and TBD respectively.

higher dimensionality it is usually more difficult to use for the purpose of comparing similar solutions.

10.2.1 One-Body density

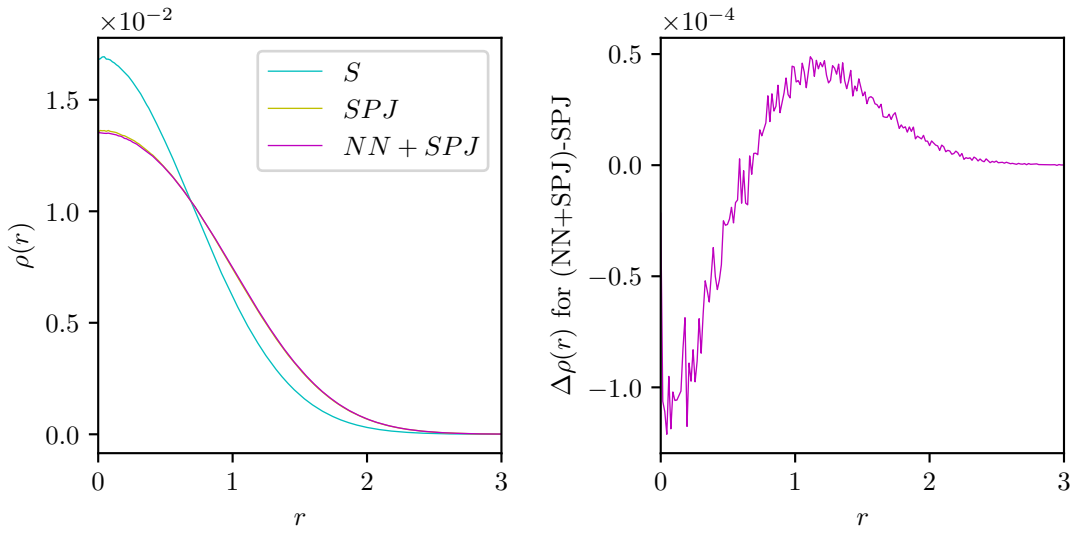


Figure (10.2) $N = 2$ interacting particles in a $D = 2$ dimensional harmonic oscillator. LH: $\rho(r)$ for a system of non-interacting bosons in a HO($\omega = 0.5$) sampled using the exact WF (teal), compared to when Coulomb interaction is included, sampled with the TWFs SPJ(yellow) and 'NN+SPJ'(magenta) - after $40k$ iterations of $M = 2^{10}$. See text for TWF abbreviations. RH: Shows the difference in OBD, $\Delta\rho(r)$, between NN+SPJ and SPJ, negative values corresponds to amplitude of 'NN+SPJ' being lowest. $\rho(r)$ was estimated from 2^{28} samples of r, using 200 bins in $r = (0, 3)$

Figure 10.2 (LH) shows how the OBD of a harmonic oscillator (HO) changes when an interactive potential is introduced. When no interaction is included in the Hamiltonian (shown in teal color in figure 10.2), the particles are more likely to be found near the center of the HO trap, with exponentially decaying probability towards the tail end of the TWF. By including the Coulomb potential, the particles tend more towards regions further from the center. This is because the repulsive force is inversely dependent on particle distance, meaning that the particles are likely to maintain a distance between them. I included the OBD sampled using both the TWF SPJ and NN+SPJ (after energy convergence), which as we saw earlier, resulted in $\bar{E} - E_0$ up to four orders of magnitude different. The OBD are however very difficult to discern by the eye. As such, I plotted the difference in OBDs, $\Delta\rho(r)$ for the two TWF (RH). The shape of said difference in OBD clearly indicates a systematic dissimilarity (as opposed to random noise) between SPJ and NN+SPJ – which is unsurprising considering that the latter generally resulted

in lower energies. For $r < 0.75$ (roughly), NN+SPJ resulted in lower OBD than SPJ. For larger values of r, the relationship was opposite, which is interesting as $r \sim 0.75$ is where the OBD from the non-interacting system cross-sections the OBDs from the interacting systems. We can from this assume that SPJ underestimates the importance of particle correlations, and accepts moves towards the origin too frequently when a particle is already in the vicinity of the origin. Adding an NN factor to SPJ consequently adjusts for this tendency to some degree.

Exploitation versus exploration re-visited

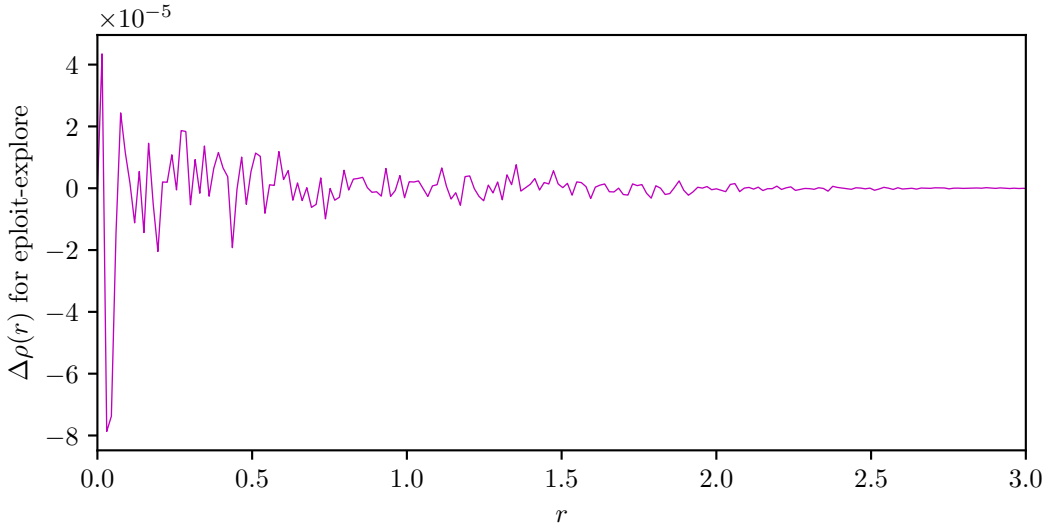


Figure (10.3) Difference in OBD, $\Delta\rho(r)$ between NN+SPJ (deep) optimized using $M = 2^{10}$ (exploit) and $M = 8 \times 2^{10} = 2^{13}$ (explore). $\rho(r)$ estimated from 2^{28} samples of r, using 200 bins in $r = (0, 3)$, after 30k iterations.

In the section on exploitation versus exploration in the previous chapter, I discussed the effects of increasing the sampling size, M, when optimizing. Increasing M was observed to either result in instability or slower convergence – which were both possible explanations considering the NNs had not converged after 50000 iterations. To further examine this I optimized two identical TWFs (deep NN+SPJ) with NN factors. One was optimized with $M = 2^{10}$ samples per iterations, and the second with $M = 2^{13}$, meaning that it was optimized using a 8 times larger sample size per update, which represents considerably more exploration. The final result for the second TWF was $\bar{E} - E_0 = 0.06(2) \times 10^{-4}$ a.u., which is comparable to the that of the the first TWF (see table 10.1). This suggests that the ground state energy estimate for a TWF with an NN factor is largely invariant to sample size when sufficiently optimized. An observation which is in contrast to what we saw for fewer iterations previously. The next question is therefore whether increased exploration has any effect on the ground state approximation. Figure

10.3 (RH) shows the difference in OBD, $\Delta\rho$ for the two TWFs. It is clear that the difference in $\rho(r)$ is smaller ($\sim 10^{-5}$ or less) than it was when I compared SPJ to NN+SPJ ($\sim 10^{-4}$). The plot bears no resemblance to neither $rho(r)$ (10.3 (LH)) nor the cross section between ρ with and without interaction. As such, I believe that the small deviations can largely be attributed to the noise associated with random sampling. The ground state approximation is generally reflected in the ground state energy, but it is feasible that we could have seen some improvement in tail end magnitudes without very large impact on $\bar{E}$ due to the general infrequency of such samples. It consequently seems that the TWFs are largely insensitive to increases in sample size (at this scale), both with respect to ground state energy and ground state approximation.

10.2.2 Two-Body density

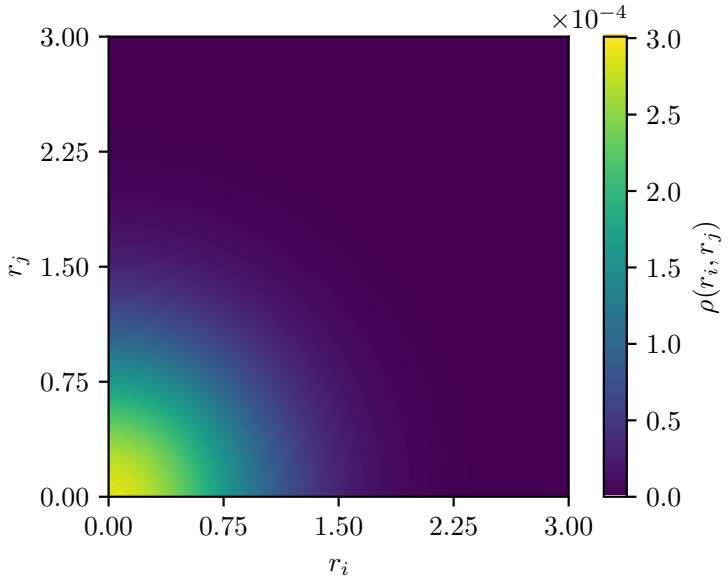
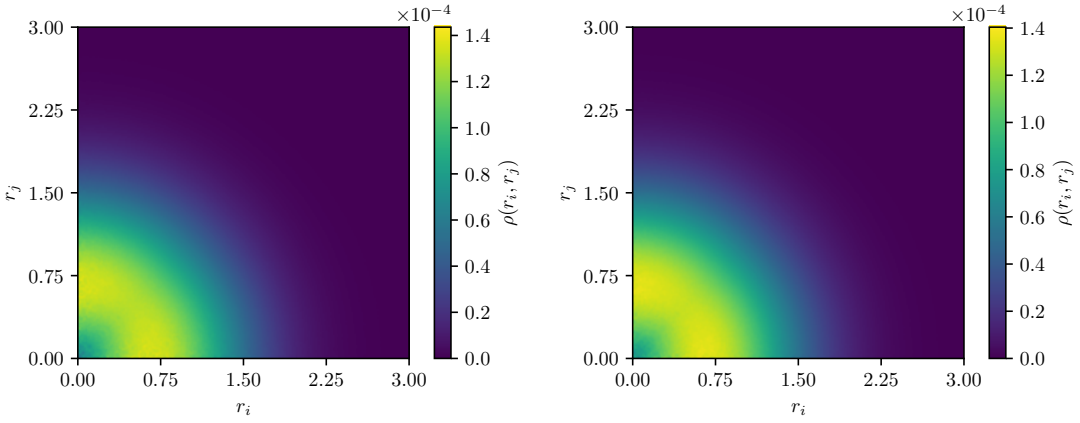


Figure (10.4) HO, $N = 2, D = 2$. Two body density, $\rho(r_1, r_2)$ for pure (isotopic) harmonic oscillator ($\omega = 1.0$).

Without interaction, the particles in a HO generally tends to occupy space close to the center of the system, as we saw when examining OBDs. This is also clearly reflected in the TBD, shown in figure 10.4. As $\rho(r_i, r_j)$ has the highest magnitude in the lower left corner, we can see that the most likely configuration is for both particles to be near the center. When the interaction is included this changes. Figure 10.5 illustrates this, and also shows that it is very difficult to discern any difference between SPJ and NN+SPJ, even though they resulted in $\bar{E} - E_0$ of different orders of magnitude. As TBD is a two dimensional measurement (two radial distances), the number of samples required to obtain high resolution with somewhat representative distribution is higher than for OBDs. As the OBDs that were sampled with these TWFs were barely discernible, it

follows that direct comparison of TBDs from identical systems is often may not be worth while unless done with a very high sample size. Therefore, let us instead discuss what we can surmise in general from the TBD when interaction is included in the system.

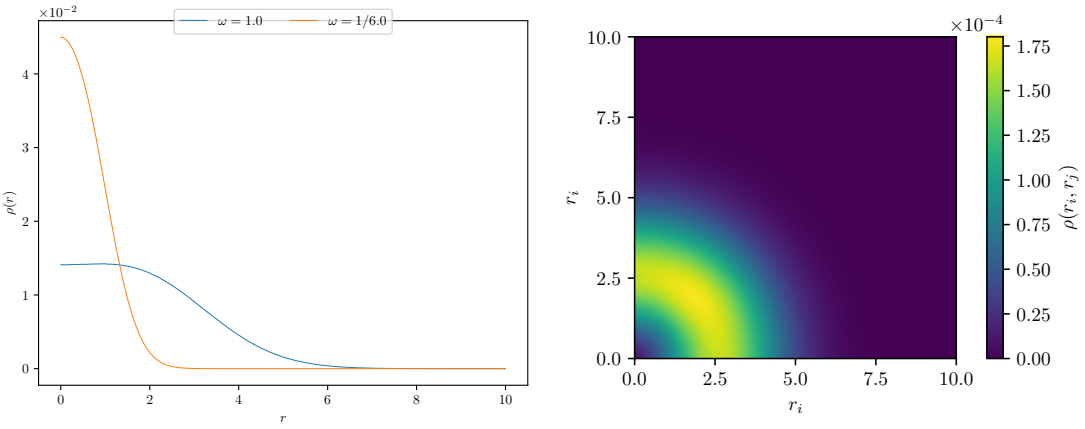
The TBD was generally highest when $r_i \approx 0$, r_j is ≈ 0.75, and vice versa. This means that when a particle is found near the origin, the other particle tends to be elsewhere due to the repulsive potential. The system still favours distributions in which one particle is close to the center, but not both. Along the diagonal $r_i = r_j$ the TBD is slightly lower than in the adjacent regions, indicating that the particles are less likely to be found at an equal distance from the origin. When particle i is near the origin, particle j can be found at a wide range of coordinates with the same distance to the origin. The range of coordinates which satisfies $r_i \approx r_j$ without resulting in a large repulsive force is however much smaller due to the strength of the HO ($\omega = 1.0$) – which after all pulls the particle towards the origin. We will revisit both OBDs and TBDs when considering the strength of the harmonic oscillator potential, and when examining the Calogero-Sutherland model.



(a) TBD for SPJ **(b)** TBD for NN+SPJ.

Figure (10.5) $N = 2$ interacting particles in a $D = 2$ dimensional harmonic oscillator. Two body density, $\rho(r_i, r_j)$. (a) for SPJ and (b) NN+SPJ, after 30k iterations, $M = 2^{10}$.

10.3 Trap strength



(a) $\rho(r)$ for $\omega = 1.0$ and $\omega = 1/6$

(b) $\rho(r_i, r_j)$ for $\omega = 1/6$.

Figure (10.6) $N = 2$ interacting particles in a $D = 2$ dimensional harmonic oscillator (HO+C). (a) one-body density, $\rho(r)$, in a narrow, and wide, harmonic oscillator (with particle repulsion). (b) Two-body density, $\rho(r_i, r_j)$, for the wide, $\omega = 1/6$ HO+C. Sampled from SPJ, after 30000 iterations of $M = 2^{10}$.

The oscillator frequency, ω, represents the strength of the confinement. Higher values of ω narrows the 'trap', while lower values results in a larger spatial field. This is reflected in the one body density for $\omega = 1/6$ compared to the one for $\omega = 1.0$, shown in figure 10.6 (a). Because the potential of the HO is weaker, the repulsive potential is more prominent. This means that the particles tend to move further away from each other, and consequently also further from the origin. The particle are however unlikely to scatter completely due to the HO potential, which still pulls the particles towards the center, albeit not as strongly. In the previous section, we could observe that $r_i \approx r_j$ has slightly lower $\rho(r_i, r_j)$ than the surrounding areas (figure 10.5). This is not the case when $\omega = 1/6$, which we can see from 10.6 (b). At this frequency, the trap is wide enough to accommodate $r_i \approx r_j$ without strong repulsion.

Let us now turn our attention towards energy minimization. We have previously seen (section correlation vs correcting) that fixing the SPJ factor in NN+SPJ resulted in worse estimates for $D = 3$, and relatively similar estimates for $D = 2$ – at the very least for the wide and deep architectures. These trends were not observable when ω was reduced[3] – neither for $D = 2$ nor $D = 3$. Table 10.3 shows that fixing the SPJ factor generally resulted in estimates one order of magnitude lower than when not fixing. SPJ without correction from a NN also resulted in estimates closer to those that included a correctional NN factor than what was observed for $\omega = 1.0$ ($D = 2$) and $\omega = 0.5$

[3] I have chosen to only focus on two values for each of the two cases $D = 2$, $D = 3$, because these are the values for which we know the exact energies. See table 9.1 in the previous chapter.

($D = 3$). This is likely connected to the observations regarding $\rho(r)$ for 'VMC' versus NN+SPJ from the section on one-body density (figure 10.2). There we saw that SPJ can be assumed to be less inaccurate as the distance between the particles increase, and especially so in regions where ρ with interaction was larger than without. As such, we would expect SPJ to perform closer to NN+SPJ when the trap strength is weaker. Why the NN+SPJ$_f$ resulted in lower estimates than NN+SPJ in this scenario is not clear. It could be a matter of convergence, meaning that the TWFs take longer to converge when the correlations are less prominent, but estimates produced after more than $30k$ iterations would be required to confirm it.

Table (10.3) $N = 2$ interacting particles in a $D = 2$ and $D = 3$ dimensional harmonic oscillator with oscillator frequency ω. Difference between energy estimate and exact ground state energy, $\bar{E} - E_0(\hat{\sigma})$ [10^{-4} a.u.] for the following TWFs; $\Psi_S(\alpha)\Psi_{FNN}$(SNN), $\Psi_S\Psi_{PJ}$ (VMC), and $\Psi_S\Psi_{PJ}\Psi_{FFNN}$(NN+SPJ). See beginning of chapter for details on architectures. Subscript f indicates that the TWF factor was fixed at pre-trained values and not optimized. Estimates are based on $M = 2^{22}$ samples after 30000 iterations of $M = 2^{10}$ samples.

TWF Ψ	D	ω	$\bar{E} - E_0$ [10^{-4} a.u.]	
			Network architecture	
			Wide	Deep
SPJ	2	1/6	6.6(7)	
NN+SPJ			0.08(3)	0.02(3)
NN+SPJ$_f$			0.02(2)	0.03(2)
SPJ	3	0.1	2.0(3)	
NN+SPJ			0.04(2)	0.9(1)
NN+SPJ$_f$			0.01(2)	0.03(1)

10.4 Comparative performance

Table 10.4 shows a the difference between the estimated ground state energy and exact ground state energy $\bar{E} - E_0$ ($\hat{\sigma}$) for the deep networks of this thesis, NN+SPJ, compared to those of two other methods. The shown estimates are for a system of $N = 2$ identical and charged particles in a $D = 2$ and $D = 3$ dimensional harmonic oscillator with Coulomb repulsion. For this final benchmarking, I optimized NN+SPJ ($L_h = 3$ hidden layers of $n_h = 16$ nodes per layer) for 40k iterations of $M = 2^{10}$ samples per iteration, and estimated the energy from $M = 2^{22}$. I discuss why I used 'wide' here instead of 'deep' in the validity section at the end of the chapter. The first alternative method uses Restricted Boltzmann Machine (RBM) with a Padé Jastrow factor (RBM+SPJ), which is similar to the approach in this thesis – albeit with another type of neural network. RBMs are covered briefly in the the chapter on machine learning. The thesis by Nordhagen[51], where the energy estimates for RBM+SPJ are from, describes the

subject in more depth. The other method is Diffusion Monte Carlo (DMC), which is different from Variational Monte Carlo, in that is does not use the variational principle, and instead relies on solving the Schrödinger Equation as a diffusion problem. DMC is one of the most precise methods available, and can in theory result in energy estimates which is numerically exact to the ground state energy if allowed sufficient run time. For more on DMC, see Høgberget[74], which is where the DMC estimates shown in this section is from. I also included energy estimates from the 'standard' VMC approach, meaning a single particle permanent and a Padé-Jastrow factor (VMC).

Table (10.4) HO(ω)+C, $N = 2$ charged particles. Difference between estimated ground state energy and exact ground state energy[28, 29], $\bar{E} - E_0(\hat{\sigma})$ [10^{-4} a.u.] for $D = 2$ and $D = 3$ dimensions. Values for DMC are taken from Høgberget[74], and RBM+SPJ from Nordhagen[51]. NN+SPJ was optimized for 40000 iterations of $M = ^{10}$ samples, with final estimates based on 2^{22} samples. See text for details on the methods.

Method	$\bar{E} - E_0$ [10^{-4} a.u.]			
	$D = 2$		$D = 3$	
	$\omega = 1/6$	$\omega = 1.0$	$\omega = 0.1$	$\omega = 0.5$
NN+SPJ	0.08(3)	−0.02(2)	0.04(2)	0.5(3)
NN+SPJ$_f$	0.02(2)	0.01(2)	0.01(2)	0.06(1)
RBM+SPJ	4.8(6)	−4.13(5)	0.80(6)	−0.88(5)
DMC		0.0(1)	−0.03(3)	0.00(2)
VMC	6.8(2)	12.0(2)	1.45(9)	2.11(7)

We have previously seen that NN+SPJ$_f$ resulted in slightly worse estimates than NN+SPJ in some of the simulations. When optimized for 40k iterations however, NN+SPJ$_f$ gave slightly better estimates, although the two methods were mostly similar. The differences between the two approaches have already been thoroughly discussed. We can nonetheless close the discussion for now by surmising that given sufficient optimization, a neural network appears to be perform no worse when used to correct a pre-trained ansatz, than when the two are optimized together.It is clear that NN+SPJ outperformed RBM+SPJ and VMC, whether the SPJ factor was fixed or not. I have previously discussed energy estimation for NN+SPJ versus SPJ and refer to the previous sections in this chapter for such considerations. It is however worth mentioning that the CPU time of SPJ was very low; ~ 0.005 seconds per iteration compared to ~ 0.3 for NN+SPJ. A possible reason for why NN+SPJ outperformed RBM+SPJ is that RBMs make explicit assumptions on the TWF form, which the FFNN does not. Another factor is that the RBMs in [51] used 2 hidden nodes per input, while NN+SPJ used 24, meaning that NN+SPJ has a lot more flexibility and expressive capacity. Even though the number of parameters was higher, the CPU time for NN+SPJ (~ 0.3s per iteration) was considerably lower than for RBM+SPJ (~ 7.2s per iteration). This is because NN+SPJ was able to optimize effectively with fewer samples per iteration. Direct comparison

with DMC is more difficult because the method is quite different from NN+SPJ. Instead we will consider it as an indicator of how well NN+SPJ performed, because DMC is considered to be highly accurate. NN+SPJ$_f$ resulted in better estimates and variance than DMC for two of the three cases, and in the last case similar magnitude albeit less exact. Diffusion Monte Carlo generally requires preparation and physical insight into the system, and is often used to improve on the estimate of a pre-trained ansatz. This means that approaches such as NN+SPJ may be effectively faster as the application of neural networks requires few calculations and almost no planning– at least if a general approach can be shown to work for a variety of systems. Because of the comparative estimation quality to DMC, we can assume that NN+SPJ$_f$ at the very least works well for the harmonic oscillator system with two interacting particles. Whether this is generalizable to larger systems remains to be seen, but the results in [2] suggests that this may be the case.

10.5 The one dimensional Calogero-Sutherland model

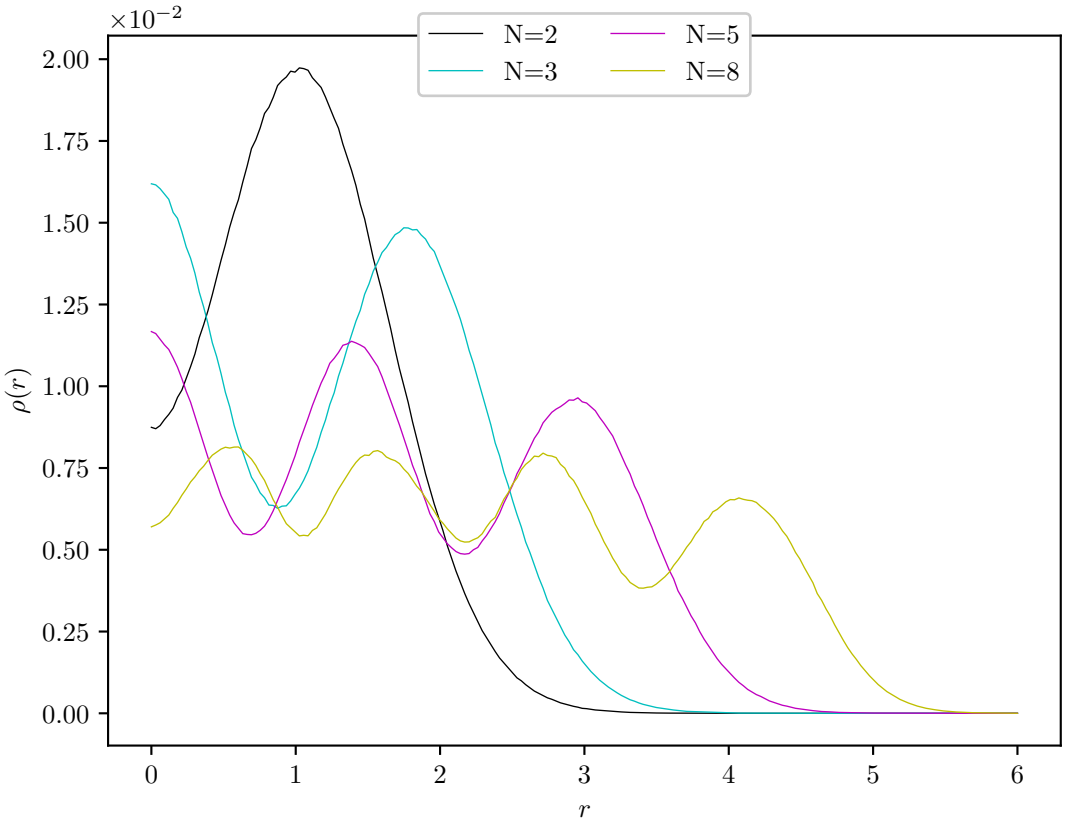


Figure (10.7) $1D$ Calogero-Sutherland with N particles. One-body densities, $\rho(r)$, based on 2^{22} samples obtained after optimizing a deep ($L_h = 5$ hidden layers of $n_h = 12$ nodes) NN for 40k optimization steps of $M = 2^{10}$ samples. Resolution: 200 bins.

The Calogero-Sutherland model represents an opportunity to test the methods of this thesis for a system of "many" interacting bosons. I chose this system for two reasons. Firstly, its exact energies are known (see table 10.5), which means that it is well suited for evaluating new methods. Secondly, [11] covers shallow (one hidden layer) neural networks used to estimate these energies, which means that I can relate the results of this thesis to said article. As was done in reference [11], I introduced the repulsive potential gradually in order to thermalize the system without energy divergence. This is because the repulsive potential here is stronger than the Coulomb potential we have dealt with so far, and because we are now dealing with more particles. This was done by defining the internal potential as

$$H_{int} = min\left[(an)^2, \sum_{j<k} \frac{\beta(\beta-1)}{(r_j - r_k)^2}\right], \tag{10.1}$$

Table (10.5) Exact ground state energies[11], E_0, for N particles in a 1 dimensional Calogero-Sutherland model.

N	2	3	5	8	10	12	14
E_0 [a.u]	3	7.5	22.5	60	95	138	189

where $a = 0.00001$, n the number of perturbations to the system, r_i the position of particle i, and $\beta = 2$ is the interaction parameter. I used the same neural network trial wave function as for the system as previously. Namely the fixed, exact single particle contribution, $\Psi_S(\alpha = 0.5)$, with a neural network factor to model the correlations;

$$\Psi_{TWF} = \Psi_S(\alpha = 0.5) \cdot \Psi_{FFNN} = \prod_i^N \exp\left(-\frac{1}{2}r_i^2\right) \cdot a^L, \tag{10.2}$$

where r_j as this is a one dimensional system. I used the two NN architectures that performed the best in the previous sections on the harmonic oscillator system, the *deep* and *wide* (see the appendix for details) network. These networks are both by definition 'deep' as they have more than one hidden layer, but the first is deeper than the second network. I also applied a "shallow" network of 30 hidden nodes in a single layer to see whether this matched the performance seen in reference [11], where this exact architecture is used, although direct comparison is difficult as said article presents the results in a plot.

I found that the quality of the estimates on the ground state energy estimates of the Calogero-Sutherland (CS) model were sensitive to the choice of time step δt in the Metropolis-Hastings sampling. I used $\delta t = 0.1$ when obtaining the results for $N = 2, 3, 5$. This means that I used a lower δ for Calogero-Sutherland (CS) model than for the other systems. This is likely a consequence of the stronger repulsive force in CS, compared to the harmonic oscillator with Coulomb interaction we have seen so far. Long particle moves (towards other particles), which we can generally associated with high δt, tends to result in a strong repulsive force, and as such, a simulated state with relatively low probability. This means that many moves are likely will be rejected, resulting in a a small effective sample size. The low probability of finding two particles in close proximity is reflected in the sharpness of the peaks in the OBDs, which I discuss in more detail below.

The one-body and two-body densities, as well as the energy estimates for $N < 10$, were based on 2^{22} samples, obtained after optimizing the deep NN for 40000 optimization steps of $M = 2^{10}$ samples. For $N = 10, 12, 14$ the ground state energy estimates tended to diverge (rather than converge towards E_0), which I addressed by first optimizing for 40000 iterations with $\delta t = 0.5$, then 40000 more iterations with $\delta t = 0.1$. This is discussed in more detail in the section on simulations on "many" particles. I used input sorting (discussed in the methods chapter, which improved the results considerably. This is because the CS model has degenerate states near the ground sate[11].

Before examining ground state energy estimation, let us first discuss the system by examining the one body densities (OBD) for $N = 2, 3, 5, 8$ particles, shown in figure 10.7. We see that more particles are associated with more "peaks" in the OBD. These peaks indicate regions with relatively high probability of finding a particle. We also see the overall amplitude decreases with N. This is due to normalization, such that $\rho(r)$ integrates to 1. For even numbers of N, the first peak can found at a distance from the origin, indicating that we are likely to find the particles pairwise equidistant at either side of origin. For odd numbers N, the first peak is at $r = 0$, as we are likely to find one particle at the origin and the remaining even number of particles distributed equidistantly at either side of the center. As the Calogero-Sutherland model is a one dimensional system, we must keep in mind that there is only one degree of freedom for the bosons. This means that we can generally 'count' two particles in each OBD peak that is not at the origin, and one in the peak at the origin – two particles will rarely be found together due to the "strong" repulsive potential. The peaks are in general sharper and higher for low r. This is because the repulsion from the particle further away from the origin (but at the same side) effectively stack to push the ones near the center together, which is counter-acted by repulsion from the particles at the opposite side of the origin. We can also recognize these patterns in the two-body densities (TBD), $\rho(r_i = 0, r_j)$ (figure 10.8). The 'hot spot' of $r_i \approx r_j$ has a considerably lower $\rho(r_i, r_j)$ than when $r_i \neq r_j$ for all r_i, which confirms pairwise distribution.

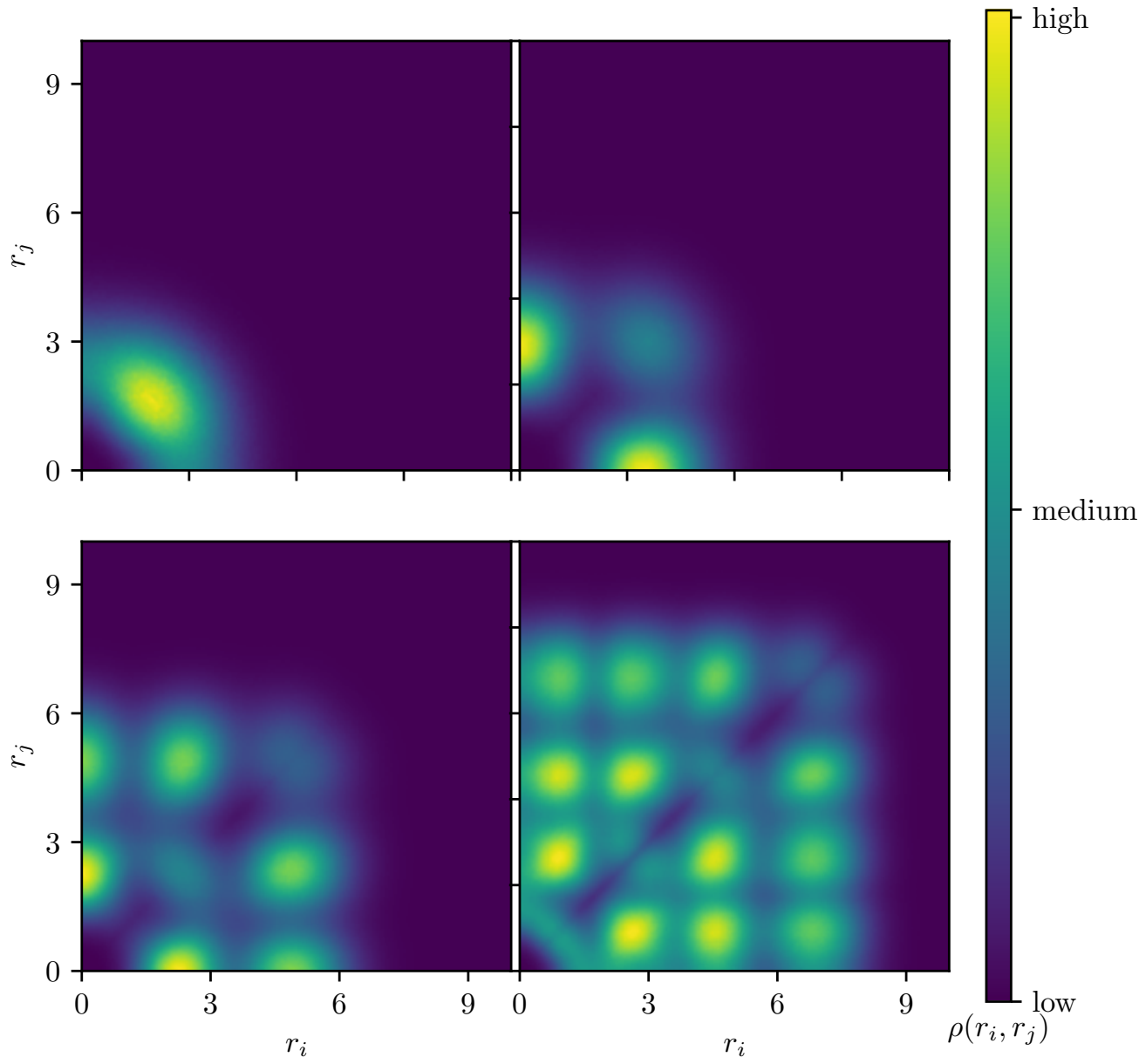


Figure (10.8) Two-body densities, $\rho(r_i, r_j)$ for $N = 2$ (upper left), 3 (upper right), 5 (lower left), and 8 (lower right) particles. Resolution: 200 by 200.

Ground state energy estimation

In the following I discuss the estimation quality of the different networks in terms of the absolute difference between estimated energy and ground energy, per particle (see table 10.6), which I will refer to as $\Delta E/N$ (measured in a.u.). For $N = 2, 3, 5$ particles, the different architectures performed largely the same. All except one of these $\Delta E/N$ values were within the same order of magnitude. For $N = 3$, the "deep" network produced an estimate one order of magnitude lower, which could be possibly be attributed to

the assessment made previously in this chapter on higher convergence rates for deeper networks. For $N = 8$ particles however, the "shallow" resulted in the lowest ΔE. This suggests that "deep" networks are not always strictly better, and that shallow networks of sufficient width can perform equally well. For $N = 2, 3, 5$ I used a different optimization scheme than for $N = 8, 10, 12, 14$, as I have previously mentioned. Simulations of "many" particles is also discussed further in the next section. We can see that the shallow network performed better than the two other architectures for $N = 8$ particles. For the "deep" network, $\Delta E/N$ at $N = 8$ did not conform to the general progression of ΔE as N increased for this network. For $N = 10$ and $N = 12$ however, the 'deep' network outperformed the shallow, and for $N = 14$, the "wide" network resulted in $\Delta E/N$ one order of magnitude lower than the two others. In reference [11], the author reports that estimation precision was observed to saturate as the number of units increased, which may be why we cannot discern any specific patterns regarding architecture performance. The issues related to simulating system of $N = 8, 10, 12, 14$ particles may also explain why we cannot discern if any stable value for $\Delta E/N$ can be expected for SNNs, and why comparing the architectures is not feasible. The variance (see section on blocking) does however appear to stabilize at 10^{-3}. Due to the reasons discussed above, the following is largely speculative. I believe that $N = 10$ and $N = 12$ may signify systems of sufficient complexity for networks with lower expressive capacity to begin to "fall off" in terms of ΔE, as the "deep" network outperformed the two other. For $N = 14$ however, ΔE for the deep network increased significantly compared to values for lower N. I believe that this is due to an issue of width in the first layer. Because the "deep" network has $n_h = 12$ nodes per hidden layer, $N = 14$ particles and hence 14 inputs means that the number of variables is higher than the number of nodes in the first layer. It would be reasonable to assume that this hamper optimization as the networks are strictly feed forward. We can also see that the "wide" network. Further study of the Calogero-Sutherland model for "many" particles is required to make any substantial assessments regarding architecture and viability of the SNN ansatz for this system.

Table (10.6) 1 dimensional Calogero-Sutherland with N particles. Absolute difference between estimated ground state energy and exact ground state energy[11] per particle, $\Delta E/N = \text{abs}(\bar{E} - E_0)/N$ [a.u.]. Estimates were based on $M = 2^{22}$ samples obtained using SNN (shallow), SNN (deep), and SNN (wide) – see text for abbreviations. For $N = 2, 3, 5$, the networks were optimized for 40k iterations of $M = 2^{10}$ with time step $\delta t = 0.1$. For $N = 8, 10, 12, 14$ the networks were optimized for $40k$ iterations with $\delta = 0.5$, then $40k$ iterations with $\delta = 0.1$ (marked *), see next text for more on this.

N	$\Delta E/N$		
	Shallow	Wide	Deep
2	0.0011(3)	0.0027(4)	0.00504(2)
3	0.0304(4)	0.0274(2)	0.0012(1)
5	0.029(1)	0.0337(5)	0.033(1)
8*	0.05(3)	0.137(4)	0.18(8)
10*	0.122(8)	0.10(1)	0.03(1)
12*	0.78(3)	1.8(4)	0.09(2)
14*	1.33(4)	0.180(8)	1(5)

On simulation of "many" particles.

I found that when increasing N (especially > 8), a trade-off between step-length, iterations and a – the parameter which initially dampens the interaction occurred. By increasing δt of the Metropolis-Hastings sampler (see the methods chapter) the particles are moved further at every sampling step. If δt is too high, the acceptance rate decreases, which hampers optimization. This issue scales with N because of how 'small' the preferred regions of occupancy are, as discussed in relation to the one-body density. When δt is low, this is not an issue. The challenge when δt is small is instead in the initial thermalization phase while $(an)^2$ dominates H_{int} (10.1). This is because the particles are unable to move far enough from each other before the potential is fully introduced, especially when N is large because each sampling step only moves one particle selected at random. During verification of the implementation, the simulations suggested that the networks were largely "inactive" during the first few hundred iteration (although this may change with architecture). If the particles are not sufficiently spread out before the networks become "active", we are likely to experience instability, and possibly invalidation due to the cusp condition (see the chapter onquantum mechanics). This issue is further complicated by the low dimensionality of the system. Altering the initialization scheme remedied this somewhat. By decreasing a by one order of magnitude, and first optimizing with $\delta = 0.5$, then $\delta = 0.1$, more particles could be simulated without energy divergence – which requires more CPU time. I was however not able however able to work out a general optimization scheme which worked for all N, meaning that the results above are unlikely to be representative of the full potential of the different networks. I believe that it is possible to find a more general and elegant solution, for

example by altering how the particles are initiated, or how the system is thermalized between iterations, but this is left for future work.

Chapter 11

Conclusions and Future Work

11.1 Summary and conclusions

I have in the two previous chapters discussed the performance and optimization of (feed forward) neural networks (NNs) in relation to estimating ground state energies, E_0, of different systems in a variational Monte Carlo (VMC) framework. This was done by simulating systems of two interacting particles in a two and three dimensional harmonic oscillator (HO), and at the end of the last chapter, also by examining the one dimensional Calogero-Sutherland model[11] for up to 14 interacting bosons. This means that most of the following observations and assessments are not necessarily generalizable to larger or more complex systems. The NNs I applied had various numbers of hidden layers and hidden nodes – or architectures. In the first chapter on results, I focused on comparing multiple experiments for each architecture in order to discuss stability and convergence rate. In that chapter, the trial wave functions were comprised of a single particle contribution of Gaussian functions and a neural network – denoted SNN. I will explicitly state which of the following pertains to the Calogero-Sutherland model.

In the introduction, I stated that one of the goals was to discuss how the optimization of NNs in VMC are affected by certain factors, which I address piecemeal below. Deep NNs, meaning networks with more than one hidden layer, were in reference [45] described to outperform shallow (one hidden layer) networks and also optimize faster on fewer samples with lower sample complexity. This was also observed to be the case in this thesis, with a few exceptions – specifically when examining the Calogero-Sutherland model. When examining different architecture for SNN used to model the HO system with two interacting particles, I found that deeper networks generally resulted in a higher convergence rate and energy estimates,$\bar{E}$, closer to the exact ground state energy, E_0, than those of shallow networks. I also found that deeper and narrower networks resulted in more accurate estimates than more shallow and wide networks with as many or more weights.

Discussing whether NNs can effectively model correlations was another of my main goals, and as such I required a basis for comparison. For this, I used what is considered the standard ansatz in VMC; a product of single particle functions and a Padé-Jastrow

factor, abbreviated as "VMC". Generalization to the many body Calogero-Sutherland model is discussed later. Using few, $\sim$ 1000, samples per iteration, SNN was shown to outperform "VMC" by resulting in energy estimating one order of magnitude closer to E_0, even with as few as 5000 iterations. This was also demonstrated to be the case after 30000 iterations. These results suggested that deep NN are capable of modelling correlations as well, and likely better, than the standard "VMC" ansatz, although this was only confirmed for two interacting particles in a harmonic oscillator.

The network in the ansatz SNN models particle correlation, which appeared to be a viable application of the networks. The NNs were however also used to correct the "VMC" ansatz, which is arguable a different type of application, and one of the aspects I originally set out to examine. This was done by defining a total trial wave function comprised of the single particle and Padé-Jastrow contributions with variational parameters fixed at pre-trained values (NN+SPJ$_f$) or optimized simultaneously as the network (NN+SPJ). This approach of correcting a pre-trained ansatz was also applied to the single particle contribution of SNN. Fixing all other factors than the NNs was generally demonstrated to result in a faster convergence rate, which was attributed to more stable conditions for the NN in the initial stages of optimization. The ansatz NN+SPJ (both fixed and not) was shown to result in energy estimates which were two orders of magnitude closer to E_0 than "VMC", both for high and low HO frequencies. Comparison of one-body densities indicated that the NNs corrected the correlations modelled by SPJ in all spatial regions of the system. This comparison also suggested that "VMC" systematically underestimated the importance of correlations, regardless of how strong the external potential was.

The performance of the trial wave functions with NN factors were observed to be largely invariant to increasing sample sizes while optimizing. The final energy estimates were of the same order of magnitude when the sample size was doubled. This suggests that the exploit-explore trade off normally associated with this type of machine learning problem may be in the favour of exploitation when NNs are used in VMC. Further study using more computational power is however required to confirm this. In this context, exploit refers to taking advantage of the current model, while explore refers to investigating other solutions.

Lastly, the SNN approach was applied to a N body Calogero-Sutherland model in one dimension, for N up to fourteen. The results demonstrated that the approach can estimate the ground state energy with a relative discrepancy $\sim 10^{-3}$ per particle or less (for N up to fourteen). This thesis was however unable to robustly assess whether deep networks outperform shallow networks in this regard (and for this model). This was due to a trade-off between step length in the sampling process, the gradual introduction of the repulsive potential, and the number of iterations. Further study with a more refined optimization and sampling scheme is required to make any final conclusions on this. For the harmonic oscillator with Coulomb repulsion however, deep networks consistently resulted in faster convergence than shallow networks.

One of the most promising assessment of this thesis is the previously mentioned application of NNs in correcting pre-trained trial wave functions. If this can be shown to work for other systems and ansatze, the method may offer a viable alternative to Diffu-

sion Monte Carlo for obtaining accurate ground energy estimates. This was exemplified by comparing the energy estimates from NN+SPJ$_f$ with those obtained from DMC for two interacting particles in a harmonic oscillator in two and three dimensions, for three different oscillator frequencies. The NN methodology studied in this thesis was shown to result in equally close, and for one of the frequencies, marginally closer, energy estimates to exact ground state energy. DMC may in some cases have shorter computational time than neural networks, but the application of NNs require little planning and physical insight. This means that the effective work load and time required to improve the ground state energy estimate of a pre-trained ansatz may be lower when using NNs than DMC.

11.2 Related research and future work

In this thesis we have seen some possible applications of deep neural networks in variational Monte Carlo. There is however a multitude of alternative approaches available. This includes different architectures, pooling layers such as the ones seen in convolutional neural networks, activation functions and optimization schemes. Some of these alternatives have been explored in other research. One such example can be found in reference [2], where the relative distances between particles was given as input to the network as well as the individual positions. In said article, the authors describe the use of neural networks to as the total trial wave function, as opposed to what I have done in this thesis.

The neural network approach also needs to be expanded to larger systems, and most interestingly those involving fermions. This is discussed in reference [3], which describes methods related to those in this thesis [1], where their methods are also applied to systems of fermions. In reference [3], the authors note that there is no computationally feasible general method of ensuring anti-symmetry using networks. As such, they go on to describe that the solution depends on application of Slater determinants. In reference [2], neural network ansätze were used to construct Slater determinants, without the use of standard ansatze, obtaining lower ground state energies than those of other Quantum Monte Carlo (QMC) methods.

The neural network ansätze in this thesis were shown to be reasonably effective with respect to computational time. This was the case because the networks were trainable with relatively few samples per iteration. Whether this is the case for more complex system was not examined. The computations may also be executed more efficiently using GPUs (Graphical Processing Units). Reference [2] does however suggest that memory may be the primary limitation with respect to system size for neural networks in VMC due to large numbers of weights.

Other optimization techniques for the neural network should also be considered. In this thesis, I have exclusively used the ADAM[8] optimizer, but other methods could possibly yield better results. One example of this is the Kronecker-factored Approximate Curvature method (K-FAC), which was used for NNs in VMC in reference [2].

[1] Articles [2] and [3] were published after work on this thesis had started.

Bibliography

[1] Bo Zhou. "New trial wave function for the nuclear cluster structure of nuclei". eng. In: 2018.4 (2018). ISSN: 2050-3911.

[2] David Pfau et al. "Ab initio solution of the many-electron Schrödinger equation with deep neural networks". In: *Physical Review Research* 2.3 (Sept. 2020). ISSN: 2643-1564. DOI: 10.1103/physrevresearch.2.033429. URL: http://dx.doi.org/10.1103/PhysRevResearch.2.033429.

[3] Jan Kessler, Francesco Calcavecchia, and Thomas D Kühne. "Artificial Neural Networks as Trial Wave Functions for Quantum Monte Carlo". eng. In: *Advanced theory and simulations* 4.4 (2021), n/a. ISSN: 2513-0390.

[4] Francesco Calcavecchia et al. "Sign problem of the fermionic shadow wave function". eng. In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 90.5 (2014). ISSN: 1539-3755.

[5] Michael A. Nielsen. *Neural Networks and Deep Learning*. eng. Determination Press, 2015.

[6] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. eng. Adaptive computation and machine learning. Cambridge: MIT Press, 2012. ISBN: 0262018020.

[7] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". eng. In: *Neural networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080.

[8] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". eng. In: (2014).

[9] *QFLOW*. https://github.com/bsamseth/qflow. Accessed: 2021-10-12.

[10] Claudia Filippi and C. J Umrigar. "Multiconfiguration wave functions for quantum Monte Carlo calculations of first-row diatomic molecules". eng. In: *The Journal of chemical physics* 105.1 (1996), pp. 213–226. ISSN: 0021-9606.

[11] Hiroki Saito. "Method to Solve Quantum Few-Body Problems with Artificial Neural Networks". eng. In: *Journal of the Physical Society of Japan* 87.7 (2018), p. 74002. ISSN: 0031-9015.

[12] Ioan Kosztin, Byron Faber, and Klaus Schulten. "Introduction to the diffusion Monte Carlo method". In: *American Journal of Physics* 64.5 (1996), pp. 633–644.

[13] "Variational Monte Carlo Calculations of A¡=4 Nuclei with an Artificial Neural-Network Correlator Ansatz". eng. In: *Physical review letters* 127.2 (2021), pp. 022502–022502. ISSN: 1079-7114.

[14] A Einstein. "Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt [AdP 17, 132 (1905)]". eng. In: *Annalen der Physik* 14.S1 (2005), pp. 164–181. ISSN: 0003-3804.

[15] Attila Szabo. *Modern quantum chemistry : introduction to advanced electronic structure theory*. eng. New York: Macmillam, 1982. ISBN: 0029497108.

[16] David J Griffiths. *Introduction to quantum mechanics*. Prentice Hall, 1995. ISBN: 0131244051.

[17] Stephen Hawking. *A Brief History of Time: From the Big Bang to Black Holes*. London: Bantam, 1988.

[18] R Nave. *Notes on Postulates of Quantum Mechanics*. http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/qm.html. Accessed: 2021-08-20. 2000.

[19] D. Sherrill. *Notes on Postulates of Quantum Mechanics*. http://vergil.chemistry.gatech.edu/notes/intro_estruc/node4.html. Accessed: 2021-08-20. 2003.

[20] D. Sherrill. *Eigenfunctions and Eigenvalues*. http://vergil.chemistry.gatech.edu/notes/quantrev/node15.html. Accessed: 2021-11-17. 2006.

[21] P. Ehrenfest. "Bemerkung über die angenäherte Gültigkeit der klassischen Mechanik innerhalb der Quantenmechanik". In: *Zeitschrift für Physik* 45.7 (1927), p. 455.

[22] *2018 CODATA recommended values*. https://physics.nist.gov/cuu/Constants/index.html. Accessed: 2021-10-27.

[23] Tosio Kato. "On the eigenfunctions of many-particle systems in quantum mechanics". In: *Communications on pure and applied mathematics* 10.2 (1957), pp. 151–177. ISSN: 0010-3640.

[24] Kevin A. Boudreaux. *Quantum Numbers, Atomic Orbitals, and Electron Configurations*. https://www.angelo.edu/faculty/kboudrea/general/quantum_numbers/Quantum_Numbers.htm. Accessed: 2021-10-27. 2021.

[25] M. Hjorth-Jensen. *Many-body Physics FYS4480: Many-body Hamiltonians, linear algebra and second quantization*. 2020. URL: https://manybodyphysics.github.io/FYS4480/doc/pub/secondquant/html/secondquant.html.

[26] *Claudio Attaccalite: Cusp Conditions*. https://www.attaccalite.com/PhDThesis/html/node62.html. Accessed: 2021-09-07.

[27] C. R Myers et al. "Fock's expansion, Kato's cusp conditions, and the exponential ansatz". In: *Physical review. A, Atomic, molecular, and optical physics* 44.9 (1991), pp. 5537–5546. ISSN: 1050-2947.

[28] M Taut. "Two electrons in an external oscillator potential : particular analytic solutions of a Coulomb correlation problem". eng. In: *Physical review. A, Atomic, molecular, and optical physics* 48.5A (1993), pp. 3561–3566. ISSN: 1050-2947.

[29] M Taut. "Two electrons in a homogeneous magnetic field: particular analytical solutions". eng. In: *Journal of physics. A, Mathematical and general* 27.3 (1994), pp. 1045–1055. ISSN: 0305-4470.

[30] *History of Information, Al-Khwârizmî Invents the Algorithm*. https://www.historyofinformation.com/detail.php?id=202. Accessed: 2021-08-23.

[31] *Encyclopedia Britannica, Al-Khwârizmî*. https://www.britannica.com/biography/al-Khwarizmi. Accessed: 2021-08-23.

[32] Alexander Amini and Ava Soleimany. *6.S191 Introduction to Deep Learning*. 2021. URL: http://introtodeeplearning.com/ (visited on 09/06/2021).

[33] Uri M. Ascher and Chen Greif. *Linear Least Squares Problems*. Society for Industrial and Applied Mathematics, 2011, pp. 141–166. ISBN: 9780898719970.

[34] Joe Abercrombie. *The First Law: The Blade Itself*. Gollancz, 2006. ISBN: ISBN 978-0575077867.

[35] *Understanding the Bias-Variance Tradeoff*. http://scott.fortmann-roe.com/docs/BiasVariance.html. Accessed: 2021-08-23.

[36] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. eng. Second Edition. Springer Series in Statistics. New York, NY: Springer New York, 2009. ISBN: 9780387848570.

[37] Pankaj Mehta et al. "A high-bias, low-variance introduction to Machine Learning for physicists". eng. In: *Physics Reports* 810 (2019), pp. 1–124. ISSN: 0370-1573.

[38] Gareth James. *An Introduction to statistical learning : with applications in R*. eng. Corrected at 8th printing 2017. Springer texts in statistics. New York: Springer, 2017. ISBN: 978-1-4614-7137-0.

[39] Morten Hjorth-Jensen. *Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis*. Sept. 2019.

[40] David Sterratt et al. *Principles of Computational Modelling in Neuroscience*. eng. Cambridge: Cambridge University Press, 2011. ISBN: 9780521877954.

[41] Alex Pappachen James. *Memristor and Memristive Neural Networks*. eng. IntechOpen, 2018. ISBN: 9535140094.

[42] Morten Hjorth-Jensen. *(UiO) Lectures Notes in FYS-STK4155. "From Stochastic Gradient Descent to Neural networks", and "Tensor flow and Deep Learning, Convolutional Neural Networks"*. Oct. 2020.

[43] *IBM: Recurrent Neural Networks*. https://www.ibm.com/cloud/learn/recurrent-neural-networks. Accessed: 2021-10-28.

[44] *Borel function, encyclopedia of math*. https://encyclopediaofmath.org/wiki/Borel_function. Accessed: 2021-10-28.

[45] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. "When and Why Are Deep Networks Better than Shallow Ones?" In: AAAI'17 (2017), pp. 2343–2349.

[46] Forest Agostinelli et al. "Learning Activation Functions to Improve Deep Neural Networks". In: (2014). arXiv: 1412.6830 [cs.NE].

[47] Ian J. Goodfellow et al. *Maxout Networks*. 2013. arXiv: 1302.4389 [stat.ML].

[48] Jianli Feng and Shengnan Lu. "Performance Analysis of Various Activation Functions in Artificial Neural Networks". eng. In: *Journal of physics. Conference series* 1237.2 (2019), p. 22030. ISSN: 1742-6588.

[49] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: ICML'10 (2010), pp. 807–814.

[50] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". eng. In: (2015).

[51] Even Marius Nordhagen. *Master thesis: Studies of Quantum Dots using Machine Learning*. eng. 2019.

[52] Dobriyan M Benov. "The Manhattan Project, the first electronic computer and the Monte Carlo method". eng. In: *Monte Carlo methods and applications* 22.1 (2016), pp. 73–79. ISSN: 0929-9629.

[53] N. Metropolis. "The Beginning of the Monte Carlo Method". In: *Los Alamos Science Special Issue* 15 (1987), pp. 125–130.

[54] Paul A. Freiberger and Michael R. Swaine. *ENIAC*. eng. 2020.

[55] Nicholas Metropolis and S Ulam. "The Monte Carlo Method". eng. In: *Journal of the American Statistical Association* 44.247 (1949), pp. 335–341. ISSN: 0162-1459.

[56] Michel Mareschal. "The early years of quantum Monte Carlo (1): the ground state". In: *European physical journal H* 46.1 (2021). ISSN: 2102-6459.

[57] J.-C Walter and G.T Barkema. "An introduction to Monte Carlo methods". eng. In: *Physica A* 418 (2015), pp. 78–87. ISSN: 0378-4371.

[58] Springer Verlag GmbH, European Mathematical Society. *Encyclopedia of Mathematics, Boltzmann weight*. http://encyclopediaofmath.org/index.php?title=Boltzmann_weight&oldid=50336. Accessed on 2021-08-02.

[59] Julien Toulouse, Roland Assaraf, and C. J Umrigar. "Introduction to the variational and diffusion Monte Carlo methods". eng. In: 73 (2016), p. 285.

[60] R. Fitzpatrick. *Variational principle*. https://phys.libretexts.org/Bookshelves/Quantum_Mechanics. Accessed: 2021-10-28. 2021.

[61] A. Weisse H. Fehske R. Schneider. *Computational Many-Particle Physics*. Berlin, Heidelberg, 2008.

[62] B.L Hammond. *Monte Carlo methods in Ab Initio quantum chemistry*. eng. Singapore, 1994.

[63] C J Umrigar and Claudia Filippi. "Energy and variance optimization of many-body wave functions". eng. In: *Physical review letters* 94.15 (2005), pp. 150201–150201. ISSN: 0031-9007.

[64] D Nissenbaum. "The Stochastic Gradient Approximation: an application to Li nanocluster". In: *PhD disseration, Northeastern University Boston, Massachusetts* (2008).

[65] *Nichols A. Romero: Slater-Jastrow wave function*. https://courses.physics.illinois.edu/phys466/sp2013/projects/1999/team4/webpage/local_energy/node1.html. Accessed: 2021-09-23.

[66] Giuseppe Carleo and Matthias Troyer. "Solving the quantum many-body problem with artificial neural networks". eng. In: 355.6325 (2017), pp. 602–606. ISSN: 0036-8075.

[67] Xun Gao and Lu-Ming Duan. "Efficient representation of quantum many-body states with deep neural networks". eng. In: *Nature communications* 8.1 (2017), pp. 662–662. ISSN: 2041-1723.

[68] Bendik Samseth. *Learning Correlations in Quantum Mechanics with Neural Networks*. nor. 2019.

[69] Siddharth Krishna Kumar. "On weight initialization in deep neural networks". eng. In: (2017).

[70] Jay L Devore, Kenneth N Berk, and Matthew A Carlton. *Modern Mathematical Statistics with Applications*. 3rd ed. 2021. Springer Texts in Statistics. Cham: Springer International Publishing AG, 2021. ISBN: 9783030551551.

[71] H Flyvbjerg and H. G Peterson. "Error estimates on averages of correlated data". eng. In: *The Journal of chemical physics* 91.1 (1989), pp. 461–466. ISSN: 0021-9606.

[72] Marius Jonsson. "Standard error estimation by an automated blocking method". eng. In: *Physical review. E* 98.4 (2018). ISSN: 2470-0045.

[73] Vilde Moe Flugsrud. *Master thesis: Solving Quantum Mechanical Problems with Machine Learning*. eng. 2018.

[74] Jørgen Høgberget. *Quantum Monte-Carlo Studies of Generalized Many-body Systems*. eng. 2013.

[75] Michael S. Floater. *UiO MAT3110: Lecture 3: Cholesky and QR*. Sept. 2017.

[76] Michael S. Floater. *UiO MAT3110: Lecture 4, Gram-Schmidt, QR solution to least squares*. Sept. 2017.

Appendices

Appendix A

OLS

A.1 OLS solution examples: QR factorization and Cholesky factorization

Assuming that a Vandermonde matrix X, such as the one described earlier, has linearly independent columns, it may be factorized as $QR = X$, where Q is an orthogonal $\mathbb{R}^n$ matrix and R an upper $\mathbb{R}^n$ triangular matrix [75]. Additionally, and generally, the vector $\mathbf{x} \in \mathbb{R}^{\mathbf{m}}$ minimizes $||A\mathbf{x} - \mathbf{b}||$ IFF it minimizes $||\Omega A\mathbf{x} - \mathbf{b}||$ for any orthogonal matrix Ω [76]. Using this, the minimization

$$||X\boldsymbol{\beta} - \boldsymbol{z}||^2 = ||Q^T X\boldsymbol{\beta} - Q^T\boldsymbol{z}||^2 = ||R - Q^T\boldsymbol{z}||^2 = || \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \boldsymbol{\beta} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} ||^2 \tag{A.1}$$

Thus solving the minimization corresponds to solving $R_1\mathbf{x} = c_1$, which, since R_1 is a non-singular, upper triangular matrix, may be accomplished by backward substitution. An alternative method for obtaining the coefficients in $\boldsymbol{\beta}$ is solving the normal equations

$$X^T X\boldsymbol{\beta} = X^T\boldsymbol{\beta}, \tag{A.2}$$

using a Cholesky factorization. The 2-norm is a convex function, with a unique minimum, found by taking the gradient of the norm. The matrix $B = X^T X$ is a symmetric $n \times n$ matrix, which allows the application of the Cholesky factorization [75]: $B = R^T R$, where $R \in \mathbb{R}^n$ is a lower triangular matrix. Thus (A.2) may be written as $RR^T\boldsymbol{\beta} = X^T\boldsymbol{z}$. In order to solve for $\mathbf{x}$, define $R^T\mathbf{x} = \mathbf{w}$, and solve $R\mathbf{w} = A^T\mathbf{b}$ for $\mathbf{w}$ by forward substitution. $R^T\mathbf{x} = \mathbf{w}$ is then solved by backward substitution.

The two applied methods are dissimilar in computational cost, measured in floating point operations (flops) - especially in the case where $m << n$. The normal equation solution uses $mn^2 + \frac{1}{3}n^3$ flops [33][p.146], while the QR decomposition requires $2mn^2 - \frac{2}{3}n^3$ flops [33][p.155]. Which means that the normal equations method is significantly faster than the QR approach. The conditioning, or stability with respect to input perturbation, are measured in conditioning number of the matrix representing the problem, $K(A)$. Higher $K(A)$ means less stable.

$$K(A) = ||A^{-1}|| \times ||A|| \tag{A.3}$$

In order to acquire the normal equations, the product of A and it's transpose is used. Hence, it's conditioning number goes as $K(A^TA) = K(A)^2$. This squaring of the conditioning number is not required for the QR approach, which means that the QR method has a lower conditioning number, and is thus the more stable method. When handling well-conditioned Vandermonde matrices, the normal equations solution with Cholesky is faster than the QR method, and not necessarily particularly unstable. However, the larger the Vanermonde matrix is, the less well-conditioned it becomes. For higher order fittings, one may therefor expect the normal equations approach to be less well suited, as the solution will tend towards instability. On the other hand, the QR method is stable for a much larger variety of problems, and should therefore be employed when the polynomial degree (or number of predictors) is large.

Appendix B

RBM

B.0.1 Boltzmann Machines

Although this thesis focuses on feed forward neural networks, I have chosen to include this short section as an example of another type of neural networks which are commonly used in VMC. Boltzmann Machines[1] (Ackley et al. 1985) are a class of stochastic and *generative* artificial neural networks. Generative models aim at learning a probability distribution, as opposed to for example feed forward neural networks, which are *discriminative models*, meaning that FFNNs predict an output value. Restricted Boltzmann machines (RBMs) are a special case of Boltzmann Machines that only include connections between nodes in different layers, meaning it has no intra-layer connections. Such networks commonly only consist of two layers, a hidden layer and a *visible layer*, which in the context of VMC typically has nodes representing the particle positions (one for each particle in each dimension). As was the case for FFNNs, the layers of an RBMs consist of nodes with weights and biases, but it does not include an output layer. Instead, the visible layer acts both as input and output, and the hidden layer works as a latent variable. Figure B.1 shows the architecture of an RBM with n visible nodes and H hidden nodes, where N would be the number of particles times the dimensionality of a quantum mechanical system.

The driving concept behind Boltzmann machines is to use the Boltzmann distribution to find the probability of a particular system state, which is reflected by the network's *energy function*, $E(\boldsymbol{x}, \boldsymbol{h})$ (not to be confused with the energy of the system). Using this, the marginal probability of a vector of visible nodes, $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, can be written as

$$P(\boldsymbol{X}) = \frac{1}{Z} \sum_{h} e^{-E(\boldsymbol{X}, \boldsymbol{h})}, \tag{B.1}$$

where Z is a partition function[2], and where $\boldsymbol{h} = \{h_j\}$ contains the binary hidden nodes

[1] I have based the following section on the book "Machine Learning" by Murphy[6] [Ch. 27], as well as the informative and well written thesis of Nordhagen [51] which focuses on Boltzmann machines in VMC simulations.

[2] The partition function Z is a special case of a normalization constant, and $\boldsymbol{h} = (h_1, ..., h_m)$ is a

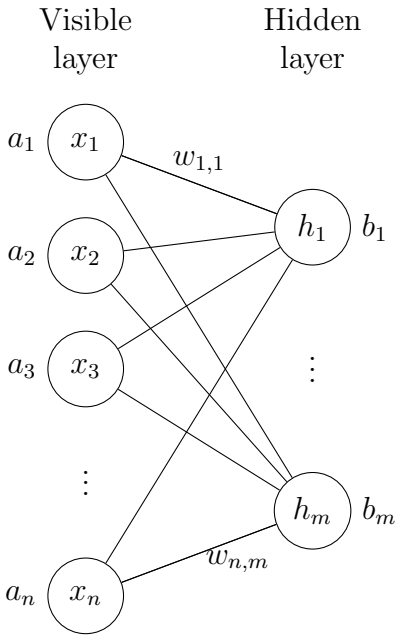


Figure (B.1) A restricted Boltzmann machine (RBM) with $1, .., i, ...n$ visible nodes and $1, .., j, ...m$ hidden nodes, where x_i, h_j are the values of node i and j in the visible and hidden layer respectively, and a_i, b_j the associated biases.

$(h_j = 1 \vee 0)$. RBMs predominantly have binary hidden nodes and binary visible nodes. However, as the network will be used to model the probability distribution of a quantum mechanical system state in terms of particle coordinates, only the hidden nodes should be binary. There are various forms of RBMs with different energy functions, specialized to different types of data. The one used here, which allows for continuous values in the visible nodes, is known as a *Gaussian-Binary RBM*. Using its associated energy function, and representing each particle position as $\boldsymbol{x}_i$, such that $i = 1, 2, .., n$ where $n = ND$ (the number of particles time the dimensionality of the system), results in the following trial wave function expression

$$\begin{aligned}\Psi_{RBM}(\mathbf{X}) &= F_{rbm}(\mathbf{X}) \\ &= \sum_{\{h_j\}} exp - \sum_{i}^{n} \frac{(x_i - a_i)^2}{2\sigma^2} + \sum_{j}^{m} b_j h_j + \sum_{i,j}^{n,m} \frac{X_i w_{ij} h_j}{\sigma^2} \\ &= exp(-\sum_{i}^{n} \frac{(x_i - a_i)^2}{2\sigma^2}) \prod_{j}^{m} (1 + e^{v(j)}),\end{aligned} \tag{B.2}$$

vector of the hidden nodes. As Ψ_{RBM} is evaluated as part of a ratio, and the derivatives of $ln\Psi_{RBM}$ also remove the Z factor, I will simply omit it from here on.

where σ^2 is the variance of the distribution, and

$$v(j) = b_j + \sum_{i=1}^{m} \frac{x_i w_{ij}}{\sigma^2}. \tag{B.3}$$

The derivatives needed to optimize the network and the derivatives with respect to the input, can be found in Appendix A: RBM.

B.0.2 Gibbs Sampling

Gibbs sampling [6] [Ch. 24.2] is a Monte Carlo algorithm (or sampler), analogous to coordinate descent in a Monte Carlo framework. Here, the two-way feed forward network, the RBM, is used to sample the joint probability of $\boldsymbol{X}$ and $\boldsymbol{h}$ in a two step process. First, $P(\boldsymbol{h}|\boldsymbol{X})$ is used to determine the state of the binary nodes $\boldsymbol{h}$, for the current system state $\boldsymbol{X}$. Then, after having updated $\boldsymbol{h}$, $P(\boldsymbol{h}|\boldsymbol{X}))$ is used to sample a new system state. In contrast to the Metropolis sampler or the Metropolis-Hastings sampler, the probability of accepting this proposed system state equals always equals one [73], ie. every proposal will be accepted. The conditional probabilities are given by

$$\begin{aligned} P(h_j = 1|\boldsymbol{X}) &= \frac{1}{1 + e^{-b_j - \sum_i^H \frac{X_i W_{ij}}{\sigma^2}}} = \text{logistic}(-(v(j)) \\ P(h_j = 0|\boldsymbol{X}) &= \text{logistic}((v(j)) \end{aligned} \tag{B.4}$$

$$P(\boldsymbol{h}|X_i) = \mathcal{N}(X_i; a_i + \boldsymbol{W}_{i*}\boldsymbol{h}, \sigma^2) \tag{B.5}$$

In order to sample $\Psi_{RBM,Gibbs}(\boldsymbol{X})$ s.t $|\Psi_{RBM,Gibbs}(\boldsymbol{X})|^2$ can be interpreted as a probability, $\Psi_{T,Gibbs}(\boldsymbol{X}) = \sqrt{F_{rbm}(\boldsymbol{X})}$, under the assumption that the wave function is positive definite. This slightly changes the expressions the derivatives w.r.t. inputs and gradients (see appendix A).

B.1 RBM: Derivatives w.r.t RBM parameters

$$\nabla_\alpha \langle H \rangle = 2 \left\langle \frac{\Psi_{RBM,\alpha_i}}{\Psi_{RBM}} (E_L - \langle E_L \rangle) \right\rangle \tag{B.6}$$

For parameter k:

$$\frac{1}{\Psi_{RBM}} \frac{\partial \Psi_{RBM}}{\partial \beta_k} = \frac{\partial}{\partial \beta_k} \log \Psi_{RBM} \tag{B.7}$$

for $\beta_k = \boldsymbol{a}$:

$$\frac{\partial}{\partial a_k} \log \Psi_T = \frac{X_k - a_k}{\sigma^2} \tag{B.8}$$

for $\beta_k = \boldsymbol{b}$:

$$\frac{\partial}{\partial b_k} \log \Psi_T = \frac{1}{1 + e^{-v(\boldsymbol{X},k)}} \tag{B.9}$$

for $\beta_k = \boldsymbol{W}$:

$$\frac{\partial}{\partial W_{kl}} \log \Psi_T = \frac{X_k e^{v(\boldsymbol{X},l)}}{(1+e^{v(\boldsymbol{X},l)})\sigma^2} = \frac{X_k}{(1+e^{-v(\boldsymbol{X},l)})\sigma^2} \tag{B.10}$$

B.2 RBM: Derivatives w.r.t input

First derivative:

$$\begin{aligned}
\frac{1}{\Psi_{rbm}} \nabla_k \Psi_{rbm} &= \nabla_k \ln \Psi_{rbm} \\
&= \nabla_k \big(\ln \frac{1}{Z} - \sum_i^N \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^H \ln\Big(1 + e^{b_j + \sum_i^N \frac{X_i w_{ij}}{\sigma^2}}\Big)\big) \\
&= -\frac{(X_k - a_k)}{\sigma^2} + \sum_j^H w_{kj} \frac{\exp\Big(b_j + \sum_i^N \frac{x_i w_{ij}}{\sigma^2}\Big)}{1 + e^{b_j + \sum_i^N \frac{X_i w_{ij}}{\sigma^2}}} \\
&= -\frac{(X_k - a_k)}{\sigma^2} + \sum_j^H \frac{w_{kj}}{\sigma^2} \frac{1}{1 + e^{-v(\boldsymbol{X},j)}}
\end{aligned} \tag{B.11}$$

Second derivative:

$$\begin{aligned}
\nabla_k^2 \ln \Psi_{rbm} &= \nabla_k \big(-\frac{(X_k - a_k)}{\sigma^2} + \sum_j^H \frac{w_{kj}}{\sigma^2} \frac{1}{1 + e^{-b_j - \sum_i^N \frac{X_i w_{ij}}{\sigma^2}}} \big) \\
&= -\frac{1}{\sigma^2} + \sum_j^H \frac{w_{kj}^2}{\sigma^4} \frac{e^{-v(\boldsymbol{X},j)}}{(1 + e^{-v(\boldsymbol{X},j)})^2}
\end{aligned} \tag{B.12}$$

or first derivative:

$$\nabla \ln \Psi_{rbm} = -\frac{(X_k - a_k)}{\sigma^2} + \sum_j^H \frac{w_{kj}}{\sigma^2} \text{logistic}(-v(j)) \tag{B.13}$$

and second derivative:

$$\nabla^2 \ln \Psi_{rbm} = -\frac{1}{\sigma^2} + \sum_j^H \frac{w_{kj}^2}{\sigma^4} \exp(-v(j)) \text{logistic}^2(-v(j)) \tag{B.14}$$

where

$$v(j) = b_j + \sum_i^H \frac{X_i W_{ij}}{\sigma^2} \tag{B.15}$$

and

$$\text{logistic}(x) = \frac{1}{1 + e^x} \tag{B.16}$$

B.3 RBM: Gibbs Sampling gradients

Using

$$\Psi_{RBM,Gibbs} = \sqrt{F_{RBM}(\boldsymbol{X})} = \sqrt{\Psi_{rbm}}, \tag{B.17}$$

which changes the first order derivative w.r.t inputs

$$\nabla \ln \Psi_{rbm} = \frac{1}{2} \nabla \ln \Psi_{rbm} \tag{B.18}$$

and the second order derivative w.r.t inputs

$$\nabla^2 \ln \Psi_{rbm} = \frac{1}{2} \nabla^2 \ln \Psi_{rbm} \tag{B.19}$$

and lastly, the parameter gradients, specifically for parameter number k of $\boldsymbol{\beta}$;

$$\frac{1}{\Psi_{T,Gibbs}} \frac{\partial \Psi_{T,Gibbs}}{\partial \beta_k} = \frac{1}{2} \frac{\partial}{\partial \beta_k} \log \Psi_{rbm} \tag{B.20}$$

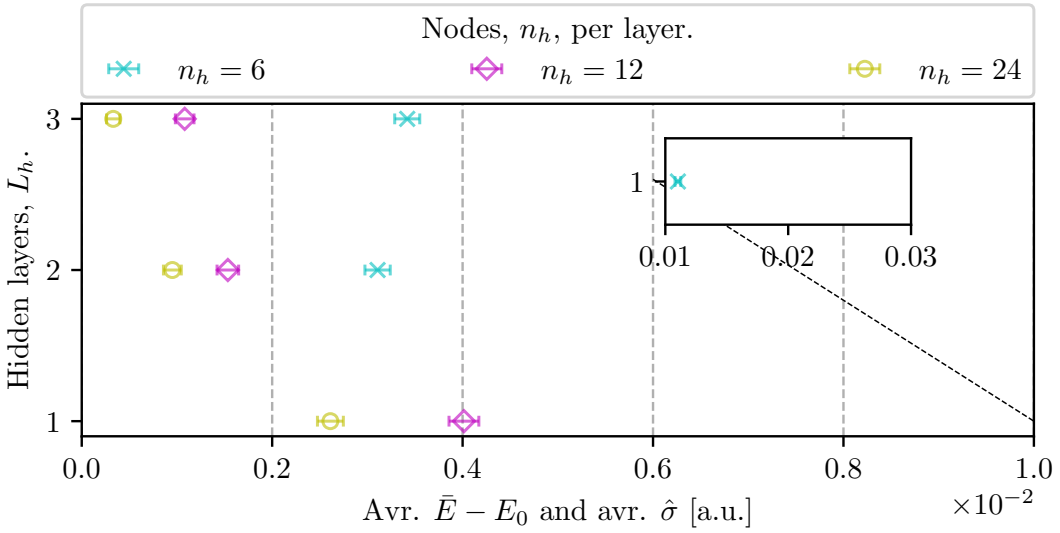
Appendix C

Simulations

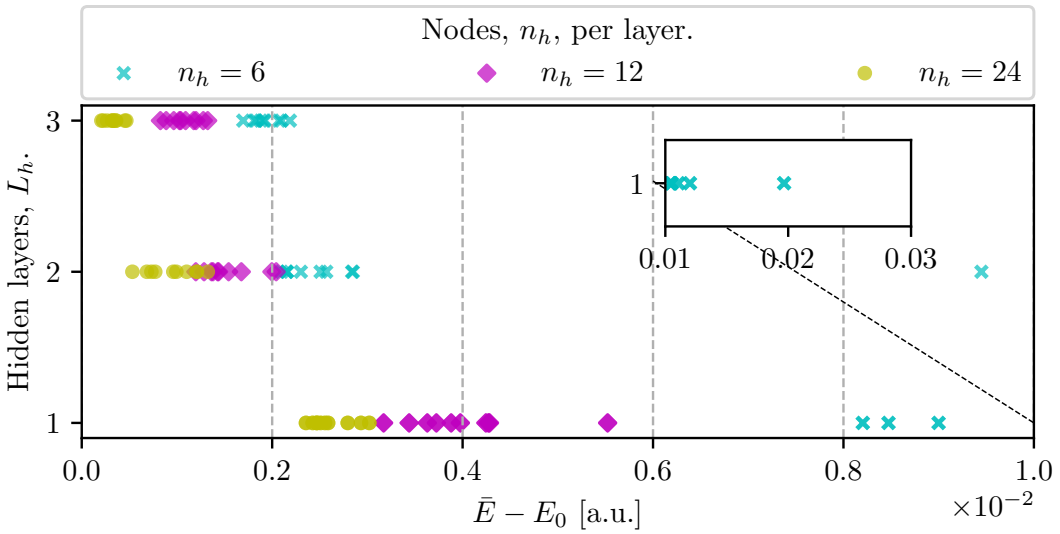
C.1 Results Ch. 1: Network architecture

Table (C.1) HO+C,$N = 2$, $D = 3$. Comparison of network architectures for . Average difference between estimate and exact energy $\bar{E} - E_0$ and number of weights for $L_h = 1, 2, 3$ hidden layers, of $n_h = 6, 12, 24$ nodes per layer, for a neural network with $PD = 6$ inputs. $5k$ iterations of $M = 2^{10}$

L_h	n_h	#weights	Average $\bar{E} - E_0(\hat{\sigma})$ $\times 10^{-2}$ $M = 2^{10}$
1	6	42	1.10(2)
	12	84	0.40(2)
	24	168	0.26(1)
2	6	78	0.31(1)
	12	204	0.15(1)
	24	744	0.095(9)
3	6	114	0.34(1)
	12	372	0.11(1)
	24	1320	0.033(7)



(a) Average of 5 experiments per NN, avr. $\bar{E} - E_0$ with avr. $\hat{\sigma}$

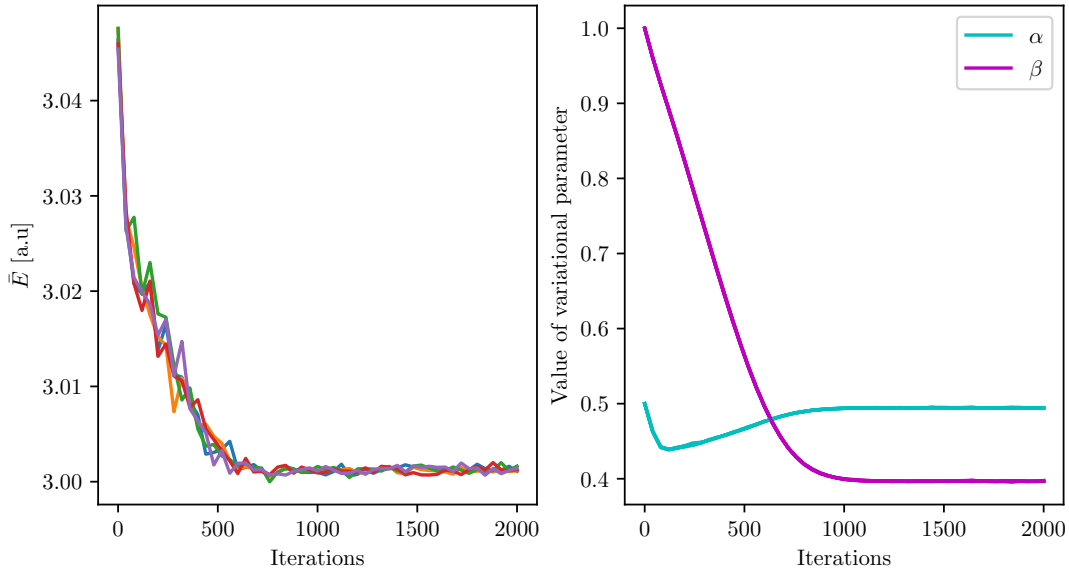


(b) Results for 5 experiments per NN, $\bar{E} - E_0$

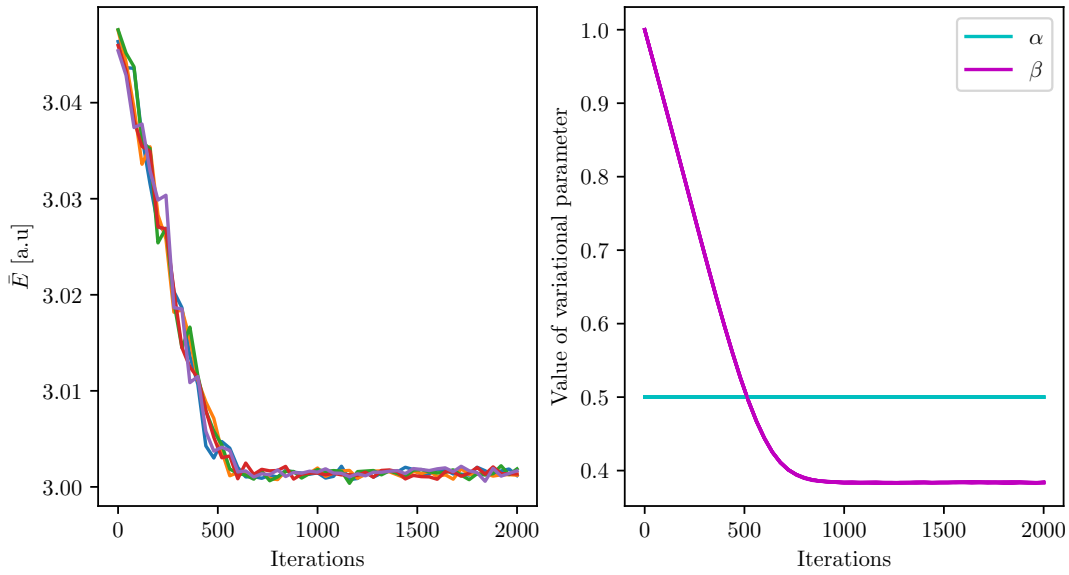
Figure (C.1) HO+C,$N = 2$, $D = 3$. Comparison of network architectures for , Ψ_{SNN}. Shows the distribution of $\bar{E} - E_0$ for $L_h = 1, 2, 3$ hidden layers, with $n_h = 6, 12, 24$ nodes per layer (identical in all layers). Based on 2^{21} samples after $5k$ optimization iterations of $M = 2048$ cycles. (a) shows average $\bar{E} - E_0$ with average $\hat{\sigma}$ based on 5 experiments per network architecture. (b) shows $\bar{E} - E_0$ for each experiment.

C.2 Results Ch. 1: SP permanent effects on optimization

C.2.1 Training Padé-Jastrow



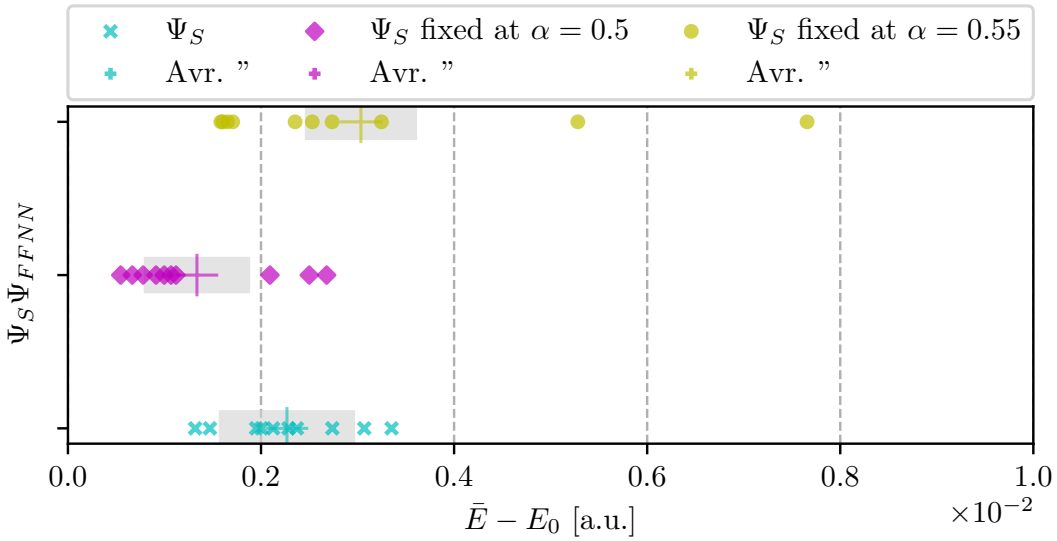
(a) Both WF factors optimized at the same time.



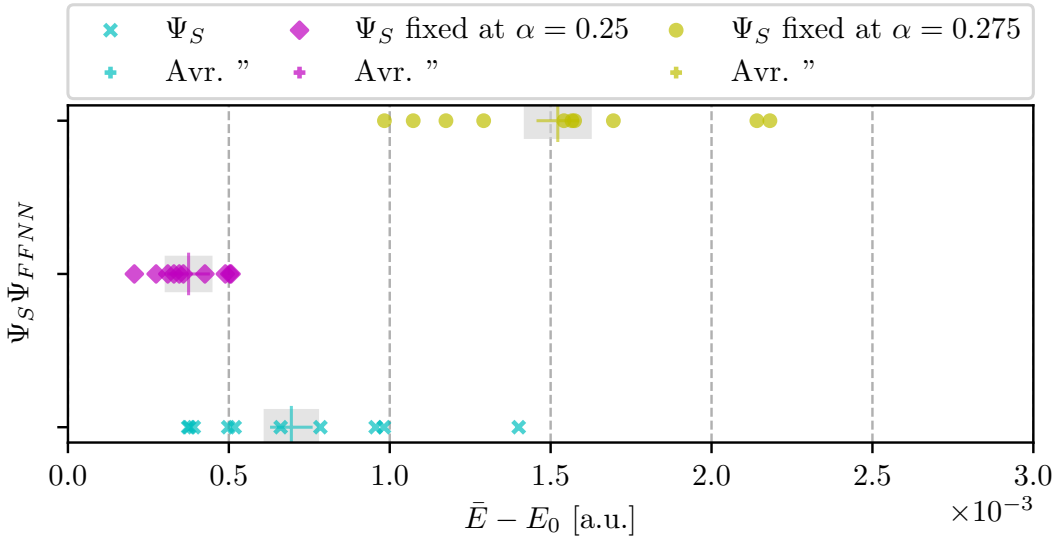
(b) Ψ_S fixed at $\alpha = 0.5$.

Figure (C.2) HO+C,$N = 2$, $D = 2$. $\bar{E}$ and variational parameters for $\Psi_{SPJ} = \Psi_S(\boldsymbol{R}, \alpha)\Psi_{PJ}(\boldsymbol{R}, \beta)$ (5 experiments) during $2k$ iterations of $M = 2^{10}$ samples using IS($\delta t = 0.5$). (a) shows Ψ_{SPJ} optimized with respect to α and β. (b) shows Ψ_{SPJ} optimized w.r.t β, and Ψ_S locked at $\alpha = 0.5$.

C.2.2 Stability 2D and 3D



(a) $D = 2$. $\bar{E} - E_0$ for 10 experiments per TWF.



(b) $D = 3$. $\bar{E} - E_0$ for 10 experiments per TWF.

Figure (C.3) HO+C, $N = 2$. (a) $D = 2$, (b) $D = 3$. Distribution of $\bar{E} - E_0$ and average $\bar{E} - E_0$ ($\hat{\sigma}$ (grey)), for 10 experiments per $\Psi_{SNN} = \Psi_S\Psi_{FFNN}$ using $L_h = 3$ hidden layers of $n_h = 12$ nodes per layer, optimized over 5k iterations of $M = 2^{10}$ samples. Variational parameter α of the single particle permanent, Ψ_S, either optimized simultaneously as Ψ_{FFNN} (cyan), not optimized and instead fixed at exact value (magenta), or not optimized and fixed at an inaccurate value (yellow). Shown estimates are based on $M = 2^{21}$ samples.

C.3 Results Ch. 10: Network architectures

The network architectures used in the chapter 10 are presented in table C.2. All networks used the $tanh(z)$ activation function in the hidden layers, and $exp(z)$ on the final layer. Weights were initialized using the initialization scheme in the methods section.

	Nodes, n_h^l, per layer.					Total number of weights	
l	1	2	3	4	5	$D = 2$	$D = 3$
Wide	16	16	16			592	624
Balanced	14	14	14	14		658	698
Funnel	20	14	12	10		658	686
Deep	12	12	12	12	12	636	660

Table (C.2) NN architectures used in chapter 10.

C.4 Iterations and cycles