INTERACTING BOSE GASES WITH RESTRICTED BOLTZMANN MACHINE

by

Kari Eriksen

THESIS
for the degree of
MASTER OF SCIENCE

Faculty of Mathematics and Natural Science
University of Oslo
March 2020

Abstract

We implement two different methods for solving many-body quantum mechanical systems from scratch, a variational Monte Carlo (VMC) method and a restricted Boltzmann Machine (RBM). We explore a confined Bose-Einstein condensate within a harmonic oscillator potential trap. The VMC uses a single variational parameter in order to describe the wave function and estimate the expectation value for the ground state energy. The RBM uses several tuning parameters such as biases and weights associated with the network to describe the NQS. Both methods use Metropolis sampling technique and gradient descent has been applied as optimization scheme.

Different trap shapes are tested and the interaction strength between particles are also tested. The VMC is able to give good results for different cases of interaction strength and serves well as a method to benchmark from.

The RBM gives decent results for a non-interacting gas, but when a Coulomb force is added the method has some difficulty with estimating the parameters needed to describe the wave function.

Acknowledgements

I would like to thank my supervisor Morten Hjorth-Jensen for much support during the process of writing this thesis. Thank you so much for inspiring me to pursue a degree in computational physics. Through your many courses I have learned a lot more than just programming. Your level of hard work and the enthusiasm you show in science education is an inspiration. And one of the biggest lessons I have learned during my time at Computational Physics is that hard work may take you a long way, but your always gonna need help from others from time to time.

I would also like to give a big thanks to everybody else at the Computational Physics department for great lunches and making the office a fun place to spend time during the last couple of years. And to Morten Ledum, thank you for plenty of good advise and for answering all of my silly questions. I will probably keep em coming.

I don't think it would have been possible to get through the last years without the many excellent group sessions of SiO Athletica, so thank you for creating a superb gym for students and for keeping me in shape.

To all my friends, thank you so much, you are the best <3

But most of all I would like to thank my mom for getting me through everything. I couldn't possibly have finished without you! You are my biggest inspiration in life.

Contents

Chapter 1
Introduction

1.1 Bosonic systems with machine learning

Quantum mechanics is one of the fundamental theories within physics and describes many of the physical phenomena happening around us. From the emitting light of an fluorescent light bulb to the components of a computer, like small transistors. Its importance to the modern world is enormous and every day thousands of researches try to come up with smart solutions to dealing with the wave function. Because of the complexity of the wave function an analytical solutions is usually not tractable and one must use numerical methods. If the wave function is to describe a system of many particles, a so called many-body problem, then the inter-particle correlation effects cause an exponential growth in solution of the wave function. This mean that there is need for new ways of solving these problems, faster computational methods and faster computers.

In quantum mechanics there are many system to explore, however strong correlated systems are becoming more and more a focal point. These systems often require heavier computations as the description of them typically are more complicated than for the weak interacting ones. When the particles move closer to one another the shift in the wave function happens more rapidly. These effects call for more calculations.

Ever since the development of trapping Bose-gases, by use of magnets or laser cooling, this field has exploded. With experimental data available the interest in solving such systems by a numerical approach has grown remarkably. Also these systems are interesting to work with given they are associated with non-complex wave functions, compared to fermionic systems where the wave function often is complex valued. This is something one has to deal with, but avoid when the particles are bosons. In Bose-Einstein condensates, increasing a system is much easier than for other systems given that all particles may occupy the same energy state and a mean-field approach may be applicable. Also the many-body quantum systems are favourable as they may represent the real world case in a better way than systems containing fewer particles.

However, most of the time solving any problem within quantum mechanics is anything but simple. Traditionally methods that have been applied are Monte Carlo simulations. But there is continuous need for new methods. When one goes beyond the stationary cases and look at the time-dependent Schrödinger equation many of the classical numerical techniques are simply ineffective. Hence the development of methods such as Coupled-cluster [1].

With the growing need of computer power there have been in recent years a development in applying neural networks in order to solve quantum mechanical problems.

Today machine learning has invaded almost every field of research and not just the nature sciences, but linguistics, physiology, economics etc. One of the largest growing topics in machine learning is deep neural networks and the development on these have been great. They may be able to extract information and represent data in a larger extent by letting data flow through several layers, and each layer obtaining new information and more complex correlations within the data.

How well an artificial neural networks representative power may be is desirable to know and is research on a lot. A two-layer network may represent any function of any dimensionality unless restricted by a limiting number of parameters (nodes in layer), and a restricted Boltzmann machine has been shown to be a universal approximator of discrete distributions [2] [3]. Perhaps though, one of the most interesting question today is the question of what neural networks and machine learning techniques potentially can describe.

The idea that one may use neural network in order to solve many-body quantum problems is fairly new [4] [5]. Recently it has been published a number of papers on the use of neural network to solve the quantum many-body problem. Like solving boson systems with use of feedforward neural network [6], solving the Ising model by supervised learning, training a network with the use of already Monte Carlo-produced configurations [7], the study of quantum entanglement with use of restricted Boltzmann machine [8]. There have been many contributions, with different results. This is a very interesting field of research as it is fairly new and we do not know the limits of machine learning. And much of the inspiration of this thesis comes from the master thesis of other fellow students as Vilde Flugsrud and Bendik Samseth who have applied restricted Boltzmann machine and neural networks on systems like quantum dot and strong interacting bosons.

1.2 The goal

In this thesis we wish to explore the application of machine learning to quantum mechanical systems. More specific we want to use an unsupervised machine learning technique called the restricted Boltzmann machine and apply it to interacting Bose-systems. In what way may a neural network be able to give an estimate on the quantum mechanical wave function? Can such a method be used to solve the Schrödinger equation and maybe be a leading tool in the field of quantum mechanics? We will use the well established variational Monte Carlo method for benchmarking. The goal is to verify the Boltzmann machine network as a method for solving many-body quantum systems. Can such a method outperform one of the most applied approaches, Monte Carlo simulations? It would be interesting to see how well a neural network may gain information on strongly correlated systems such as the liquid Helium-4 compared to more weakly interacting cases like bosons confined within a trap, mainly since these are more complicated cases. It would tell us something of the extent of machine learning applied to quantum systems.

1.3 Development of code

For this thesis a method for solving correlating systems by use of variational Monte-Carlo and restricted Boltzmann Machine has been developed from scratch. The code has been written in Python and in an object oriented way. Adding new systems may easily be done. All the underlying work, the entire code structures and the files for this document can be found at `https://github.com/KariEriksen`.

When developing the VMC solver I have been inspired by the work of Morten Ledum and Håkon Kristiansen. In the case of the restricted Boltzmann machine solver the code was based upon previous work done in collaboration with fellow students Robert Solli and Geir Ulvik.

Part I

Theory

Chapter 2
Many-body quantum theory

Before Marie Curie's research on radiation, Heinrich Hertz's discovery of the photoelectric effect or Max Planck's solution to of the black-body radiation problem, classical physics; Newton's and Maxwell's equations could basically describe the whole physical world around us. In the early 1900s however many experiments could not be explained by classical physics, and there was the need of a new theory. Quantum theory. This was not a theory made by one individual though, but instead a collaboration of many people; Schrödinger whi suggested electrons can be described by a wave equation, Wolfgang Pauli's invention of the exclusion principle, Max Born's interpretation of the wave equation as a probability distribution and many more. It was Werner Heisenberg who in 1932 was rewarded the Nobel prize in Physics for the creation of quantum physics. Most known possibly for the Heisenberg uncertainty principle. But it was most certainly a work by many. [9]

The development of the field from the early 30s and until today has given us the knowledge of atomic structure, elementary particles, radiation etc. The main problem in quantum physics is determine the electron behavior, and today physicists still work on this, from determining the electron behavior in an atom or molecule at microscale to study of the Bose-Einstein condensate at macroscale. In this chapter we look a the most basic parts of quantum mechanics, like how we may interpret the wave function and what the Schrödinger equation simply is. Most of the theory below comes from [10].

2.1 Classical Mechanics

In classical mechanics a particles state is described by two variables, the particles position and its momenta. For a system of N particles the position and the momentum, $q = (\vec{r}_1, \cdots, \vec{r})_N$ and $p = (\vec{p}_1, \cdots, \vec{p}_N)$, together form a point $\xi(q, p)$ in a two-dimensional *phase space* $\in \mathbb{R}^{2 \cdot n}$. The phase space contains all the possible values for these two variables. The state variables are governed by *Hamilton's equation of motion*

$$\dot{q} = \frac{\partial}{\partial p}\mathscr{H}(q,p), \tag{2.1}$$

$$\dot{p} = -\frac{\partial}{\partial q}\mathscr{H}(q,p), \tag{2.2}$$

where $\mathscr{H}$ is the *Hamiltonian*, an operator that we interpret as the total energy of the system. To see this we write out a special case within the classical Hamiltonian dynamics

$$\begin{aligned} \mathscr{H}(q,p) &= \mathscr{T}(p) + \mathscr{V}(q) + \mathscr{W}(q), && (2.3)\\ &= \frac{1}{2m}\sum_{i=1}^{N}|\vec{p}_i|^2 + \sum_{i=1}^{N} v(\vec{r}_i) + \frac{1}{2}\sum_{i\neq j}^{N} w(r_{ij}). && (2.4) \end{aligned}$$

This is the Hamiltonian for N particles of mass m with inter-particle reaction through the force of a central potential $w(r_{ij}) = |\vec{r}_i - \vec{r}_j|$, moving in an external potential field $v(\vec{r})$. We recognize the first term as the *kinetic energy*, the second term as the *external potential energy*, and the final term as the *interaction energy.*

2.2 Quantum Mechanics

In ordinary quantum mechanics an observable is a linear operator acting on a Hilbert space. Position operator and momentum operator, other observable quantities like angular momentum, energy, and so on, are linear operators constructed out of linear combinations of products of the position and momentum. Atomic units, all equal 1. Hamiltonian

2.2.1 Canonical quantization

In order to move from the classical Hamiltonian description of a particle system to the quantum mechanical description we now look at canonical quantization. In classical mechanic we described the state of a system as a point in phase space. In quantum mechanics however the state is a vector containing all information about the system. The *quantum state*, ψ, is a complex-valued vector state in an infinite *Hilbert space*, i.e. a complete vector space with an inner product. The inner product will then represent a complex number that link two elements together within the vector space.

Where the measurement of variables in the classical case was given through an observable $\omega(q,p)$, the quantum observable is an *Hermitian* (self-adjoint) operator Ω. The operators value in the state ψ is called an *expectation value.* In quantum mechanics we work with expectation values, this because we can not measure a particles quantities accurate. We do not know exact where the particles will be at a given time, so we work with probabilities. We can measure the probability of a particle being at a certain place at a certain time. So the expectation value tells us the likeliest position or momentum of a particles state. For a system of identical particles with positions in $\vec{r}$, the coordinates are given as $x = (\vec{r}, s)$, where the s is called spin. This is an extra internal degree of freedom the particle possesses. Then our state $\psi(x_1, \cdots, x_N)$ is the wave function that depends on all coordinates and $(x_1, x_2, \cdots, x_N) \in X^N$ is a point in the configuration space of N particles

$$\psi = \psi(x_1, x_2, \cdots, x_N).$$

We can obtain all physical properties of the system through the state ψ. So how do we move from the classical case to the quantum mechanical one? Canonical quantization is a procedure where we rewrite the classical coordinates $\xi(q, p)$ to operators $(\hat{q}, \hat{p})$. This is done through the canonical commutation relations

$$\{q_i, p_j\} = \delta_{ij} \Longrightarrow [\hat{x}_i, \hat{p}_j] = i\hbar\delta_{ij}. \tag{2.5}$$

The left side of this equation tells us that our coordinate system in the classical case is canonical[1]. We turn the Poisson brackets into commutation relations to determine the new operators. The new operators are then

$$\hat{x}\psi = x\psi \quad \text{and} \quad \hat{p}\psi = -i\hbar\frac{\partial\psi}{\partial x}.$$

But the Hilbert space is also a part of the solution and is often chosen as the space of square-integrable function, L^2. This is convenient since the state ψ, in quantum mechanics interpreted as the wave equation, has a associated probability given by itself squared

$$P(x_1, \cdots, x_N) = |\psi(x_1, \cdots, x_N)|^2. \tag{2.6}$$

This is Borns statistical interpretation of the wave function where $P(x_1, \cdots, x_N)$ is the probability of the system being in a given state, or configuration. Since the particles has to be somewhere, the integral over the entire phase space must equal one

$$\int_{X^N} |\psi(x_1, \cdots, x_N)|^2 dx_1 \cdots dx_N = 1.$$

2.2.2 Second quantization

Second quantization formalism was designed expressly for calculating matrix elements of operators between wave functions of the form (2.7) [11]. The new bra/ket notation was developed by Paul Dirac, an English theoretical physicist who wan the Nobel prize in Physics 1933. For indistinguishable particles (i.e. identical particles) the wave equation may be written as the product of all single particle states ϕ_i. In the case of bosons the wave function must be symmetric in the interchange of particles

[1] $\{f, g\} = \sum_{i=1}^{N} \left(\frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial g}{\partial q_i} \right)$ with $\{q_i, q_j\} = 0$, $\{p_i, p_j\} = 0$ and $\{q_i, p_j\} = \delta_{ij}$ where δ_{ij} is the Kronecker delta.

$$\psi(r_1, r_2, \cdots, r_N) = c_N \sum_{sym} \phi_1(r_1)\phi_2(r_2)\cdots\phi_N(r_N), \tag{2.7}$$

where c_N is a normalization constant. In second quantization the particles themselves are discrete quanta created and destroyed with creation and annihilation operators. The creation operator $a_q^\dagger$ acts on the state $|\cdot\rangle$ and creates a particle q by

$$a_q^\dagger |p_1 \cdots p_N\rangle = |qp_1 \cdots p_N\rangle .$$

The annihilation operator is the Hermitian adjoint of $a_q^\dagger$. If $q = p_j$ then the annihilation operator removes the already existing particle q

$$a_q |p_1 \cdots p_N\rangle = \left|p_1 \cdots p_{j-1}p_{j+1} \cdots p_N\right\rangle .$$

This new bra/ket notation indicates that $|\Psi\rangle$ is the state and we may find the expectation value of the operator Ω by

$$\langle\Omega\rangle = \langle\psi|\hat{\Omega}|\psi\rangle = \int \psi(x_1, \cdots, x_N)^*[\hat{\Omega}\psi(x_1, \cdots, x_N)]dx_1 \cdots dx_N. \tag{2.8}$$

We only mention this as second quantization is essential in many-body quantum theory. This however is not a formalism we will make use of in this thesis. So we will go no further into the subject.

2.3 The Schrödinger equation

At the center of quantum mechanics stands the Schrödinger equation, a linear partial differential equation that governs the dynamics of the system. The time-dependent Schrodinger equation is a partial differential equation that predicts the development of a quantum system in time

$$i\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = \hat{H}\left|\psi(t)\right\rangle . \tag{2.9}$$

It has its name from Erwin Schrödinger, an Austrian physicist who shared the Nobel Prize in Physics with Dirac in 1933.

For a system in a stationary state, in its ground state, the time-independent Schrodinger equation is sufficient to describe the system. This is a much simpler equation

$$\hat{H}\psi(x_1, x_2, \cdots, x_N) = E\psi(x_1, x_2, \cdots, x_N). \tag{2.10}$$

Here the Hamiltonian is independent of time and the energy E is a constant and an eigenvalue of $\hat{H}$. This is an eigenvalue problem with the wave function being an eigenfunction of $\hat{H}$. By solving this equation we get the eigenfunction containing the information we seek of the system, and its associated eigenvalues. Like we explained before, the wave function can be interpreted as a probability amplitude of the configuration of the particles within the system.

2.4 The quantum Hamiltonian

The Hamiltonian may take many forms, depending on the system associated to it. For the simplest case, a free particle, the only force acting on the particle is the momentum and the Hamiltonian is simply described with the momentum operator

$$H = -\frac{\hbar^2}{2m}\frac{d^2\Psi}{dx^2},$$

and the total energy of the system is given by the kinetic energy. The total equation of motion is the following Schrödinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2\Psi}{dx^2} = E\Psi.$$

However this is a case which have no closed form solution for one particle. As the free particle has no stationary state. It must be contained by a potential in order to fall to ground state. Instead one deal with wave packets, describing the general motion of all particles.

2.4.1 The Harmonic Oscillator

Where the system of the free particle serves more as an fundamental understanding of the wave-like abilities of a particle, the harmonic oscillator is essential in quantum mechanics in that it can describe a lot of cases like a vibrating molecule, motion of atoms in a solid lattice etc.

In this thesis we will be working with a harmonic oscillator potential, so we therefore give a short explanation of the basic principles and characteristics of the Harmonic Oscillator. It is similar to the classical harmonic oscillator, but in the quantum harmonic oscillator the solution of the energy comes in form of quantum levels. Accepted energy states of the particle. The Hamiltonian has the same kinetic term as for the free particle, but an additional term for the potential V of the harmonic oscillator

$$H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}m\omega^2 x^2, \tag{2.11}$$

where ω is the frequency of the oscillator potential. The solution of this system is

$$H = \hbar\omega\left(aa^\dagger - \frac{1}{2}\right),$$

with a being a ladder operator raising the energy from one level to the next, and $a^\dagger$ is its adjoint conjugate. a is given by

$$a = \frac{1}{\sqrt{2\hbar m\omega}}(i\hat{p} + m\omega\hat{x}).$$

The spectrum of the Hamiltonian is given by the energy values per particle

$$E_n = \left(n + \frac{1}{2}\right) \hbar\omega,$$

for $n = 1, 2, 3, ...$ [12]. This is the quantization of the particles.

Chapter 3
Dilute Bose Gases

The area/study of dilute Bose gases has been a field of research with great many discoveries over the last decades. The fact that microscopic quantum phenomena can be observed at a macroscopic level, compared to an atomic one, makes it an interesting subject in itself. With the improvement of laser cooling techniques there has been an explosion of discoveries and verification on theories of dilute Bose gases. Some of which include cold atomic gases of Rubidium-87, Sodium-23 and Lithium, (reference) 1995. Other references: [13] [14].

In dilute gases the scattering length is shorter than the inter-particle spacing. If we compare the molecule density in air at room temperature and atmospheric pressure which is 10^{19} cm^{-3} to the density of a dilute gas at about 10^{14} cm^{-3}, the difference is enormous.

However, in order to observe quantum effects at such low densities, we must require low temperatures. Helium liquid must be cooled to the order of 1 K, i.e. absolute zero in order for effects to occur. This compared to electrons in metals which show quantum effects below the Fermi temperature, $10^4 - 10^5$ K. For the atomic nuclei the density is so high that degeneracy happens at temperature close up to 10^{11} K.

Bose-Einstein condensation in dilute gases occurs when the temperature is so low that λ_T, the thermal de Broglie wavelength

$$\lambda_T = \left(\frac{2\pi\hbar^2}{mkT}\right)^{1/2},$$

is comparable to the mean interparticle spacing, $n^{-1/3}$ [15]. If T is high then λ_T is small, and the gas behaves classically. For low T however, the wave nature of the particle becomes important.

Working with dilute gases gives us the opportunity to approximate the system by a mean-field approach using Hartree-Fock theory. This is the basis of the Gross-Pitaevskii equation, the foundation of the studies of the Bose-Einstein condensate. It is an equivalence to the Schrödinger equation. The wave function it the solution of the Gross-Pitaevskii equation.

Maybe specify some constants if that is necessary.

Application in condensed-matter physics, fluid mechanics, atomic and nuclear physics.

3.1 Bose-Einstein Condensate

The Bose-Einstein condensate (BEC) is the state of a dilute Bose gas at low temperatures at which most of the particles in the gas all resides the same quantum state. This was predicted by Albert Einstein in 1925 after taking the work of Satyendra Nath Bose, who was studying the statistics of photons, further. He considered a non-interacting gas of bosons at low temperature and concluded that the particles could all be found in the lowest-energy single-particle state. Together with Bose he formed the foundation of Bose-Einstein statistics which describes the statistical distribution of bosons. Before looking closer at the statistics in thermodynamics it can be practical to explain what a boson is.

3.1.1 Bosons and fermions

Bosons are particles with integer spin. They can be fundamental particles like the photon or gluon. But also composite particles like mesons or the stable isotope of the helium atom, ^{4}He. They posses the ability to occupy the same quantum state, unlike fermions which are particle of half-integer spin. Fermions include all quarks and leptons, but like bosons they can also be composite particles, like ^{3}He. Fermions underlay the Pauli exclusion principle. It states that two or more identical fermions can not occupy the same quantum state. The typical example being two electrons occupying the same orbital in an atom must have opposite spin quantum number as they have same quantum numbers otherwise. This principle leads to fermions being governed by the Fermi-Dirac statistics and bosons by the Bose-Einstein statistics.

We will come back to this, but first, lets look at one of the most important formulas in statistical mechanics.

3.1.2 Particle statistics

For non-interacting non-distinguishable particles in a box of volume V the *partition function* is given by

$$Z_{tot} = \frac{1}{N!} Z_1^N,$$

where N is the number of particles, $N!$ the number of way one can arrange these within the system, or microstates, and Z_1^N the partition function of each particle, i.e. the partition function is the sum over all states. If the distance between the particles are so large that the probability of two particles wanting to occupy the same single-particle state is very low, then we can ignore quantum effects. Meaning $Z_1 \gg N$ or to say that the number of states is much greater than the number of particles. This is equivalent to the statement that

$$\frac{V}{N} \gg v_Q,$$

which says that particle density is much greater than the quantum volume. However if this is not the case then it matters a great deal if we are dealing with fermions or bosons. The description of the partition function no longer makes sense, because in the case of fermions the particles will fight over the lowest states since they can not occupy the same ones. Bosons however do not have such a restriction and can all fall to the lowest energy state. So the term $N!$ will be different in the two cases. We now look closer at the particle distribution of gases in the non-classical limit for fermions and bosons, we turn to quantum statistics.

Picture a system in thermal equilibrium with a reservoir with temperature T, this system is called the canonical ensemble. Then we know from thermal physics that the probability of finding a system in any particular microstate s with energy $E(s)$ is given by

$$P(s) = \frac{1}{Z} e^{-E(s)/kT}. \tag{3.1}$$

The exponent is the Boltzmann factor, $e^{-E(s)/kT}$, where k is the Boltzmann constant and T the temperature of the reservoir. Now the partition function is the sum over all Boltzmann factors. This is easily seen if we use the fact that the sum of all possibilities must equal 1

$$1 = \sum_s P(s) = \sum_s \frac{1}{Z} e^{-E(s)/kT} = \frac{1}{Z} \sum_s e^{-E(s)/kT}. \tag{3.2}$$

The particle must be in one of the states. For non-interacting particles in thermodynamic equilibrium the mean occupancy number is given by the Boltzmann distribution. This is interpreted as the average number of particles occupying a state with corresponding energy $E(s)$

$$\overline{n}_{Boltzmann} = e^{-(\epsilon-\mu)/kT},$$

where the energy from now on is called ϵ and μ is the chemical potential. If we now look at a system where the system is allowed to exchange particles with the environment as well as the energy, the factor in the exponential changes. And we end up with the Gibbs factor

$$\text{Gibbs factor} = e^{-(\epsilon-\mu N(s))/kT}.$$

We are now looking at the grand canonical ensemble, and the partition function from before is now a grand partition function. Now we can calculate the probability of a system containing N particles being in the state s with corresponding energy ϵ. μ being the chemical potential of the reservoir that is effectively constant, T is also held constant.

$$P_\epsilon(n) = \frac{1}{Z} e^{-n(\epsilon-\mu)/kT}$$

We want to find the average number of particles being in one energy state ϵ, i.e. the distribution of particles over energy states. Lets consider the probability of a particle occupying a single-particle state. If we begin looking at fermions, we know they obey the Pauli exclusion principle, and the grand partition function for the one particles is given as

$$Z = 1 + e^{-(\epsilon-\nu)/kT}.$$

Since the particle can either occupy the state or not we get the following

$$\overline{n} = \sum_n nP_\epsilon(n) = \frac{e^{-(\epsilon-\mu)/kT}}{1+e^{-(\epsilon-\mu)/kT}} = \frac{1}{e^{-(\epsilon-\mu)/kT}+1}.$$

And the average number of fermions in a state is given by

$$\overline{n}_{FD} = \frac{1}{e^{(\epsilon-\mu)/kT}+1}.$$

Since any number of bosons can occupy the same state at the same time, the partition function looks different

$$Z = \frac{1}{1-e^{-(\epsilon-\mu)/kT}}.$$

And the calculation of the average is somewhat more difficult, but using the equality

$$\sum_n -ne^{-nx} = \sum_n \frac{\partial}{\partial x} e^{-nx},$$

we find the average by solving the derivative, where $x \equiv (\epsilon_\nu - \mu)/kT$, and get

$$\overline{n} = \sum_n n\frac{e^{-nx}}{Z} = -\frac{1}{Z}\sum_n \frac{\partial}{\partial x}e^{-nx} = -\frac{1}{Z}\frac{\partial Z}{\partial x}.$$

The average number of bosons being in a state is

$$\overline{n}_{BE} = \frac{1}{e^{(\epsilon-\mu)/kT}-1}, \tag{3.3}$$

and we end up with a different distribution than for fermions.

Figure 3.1 show the distribution for the different particles. For the Fermi-Dirac gas the distribution goes to 1 as temperature decreases, when $\epsilon \approx \mu$. If we were to lower the temperature toward zero, then the Fermi-Dirac distribution becomes a step-function. The Bose-Einstein distribution diverges as the temperature decreases, when $\epsilon - \mu$ goes to zero. All the particles will occupy the ground state. When ϵ is much greater than μ the term ± 1 in the denominator of both expressions becomes irrelevant and they reduce to the Boltzmann distribution, [16].

3.1.3 The ideal Bose gas

Classical case, Boltzmann distribution Uniform density: particles in a box with voloum V. Density inside box: n = N/V

Energy spectrum/particle energy: $\epsilon = \hbar^2 k^2/2m$

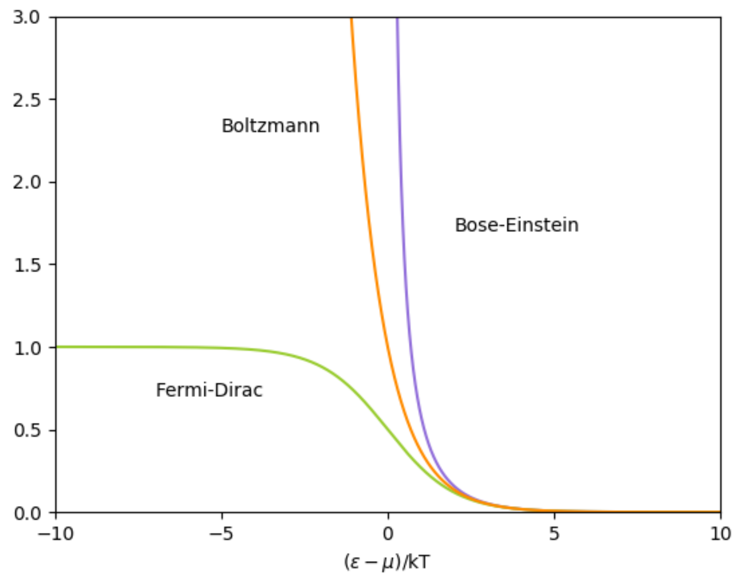

Figure 3.1: The distribution of Boltzmann, Bose-Einstein and Fermi-Dirac gases over energy.

The termodynamics of an ideal Bose gas is best calculated using the grand canonical ensamble.

Above a temperature T_c: classical case Below this temp we must take quantum effects into account.

3.2 Bose gas in a trap

As mentioned above, methods for trapping and cooling alkali atom clouds have developed over the last decades and caused the interest in these systems to grow considerably. Attempts have been made to show explicitly that BEC occurs in different systems of interacting bose gases [17] [18]. We will now look closer at a Bose gas confined within such a trap and look at the properties of the condensate. For the BEC the theoretical framework lies within the Gross-Pitaevskii (GP) equation, derived by Eugene P. Gross and Lev Petrovich Pitaevskii each in their own separately papers, 1961.

The Gross-Pitaevskii equation form the basis of studies of dilute Bose gases at zero temperature. It gives the ground-state energy of a system of identical bosons. The trap is described by a harmonic oscillator potential. We will consider a dilute Bose gas where the inter-particle distance is much greater than the scattering length. This system can be approximated by a mean field theory, i.e. the Gross-Pitaevskii equation.

3.2.1 The Gross-Pitaevskii equation

The total wave function in the Hartree-Fock approximation of a system of N bosons is represented by the product of single-particle wave functions ϕ

$$\psi(r_1, r_2, \cdots, r_N) = \phi(r_1)\phi(r_2)\cdots\phi(r_N). \tag{3.4}$$

In the fully condensed state all bosons will occupy the same energy state, the ground state. We can therefore write the wave function as in (3.4). The wave function does not contain description of the interaction between neighbouring particles close to one another. This is handled by a pseudopotential represented by an effective interaction, $U_0\delta(r_i - r_j)$ or a mean-field potential. When two particles approach each other the wave function quickly changes. The mean-field approach makes it possible to avoid having to calculate the short-range correlations between particles that move close in spatial space. Since the energy of the gas is so low, we assume zero-temperature, the scattering length is much less then the mean interparticle spacing and we can replace the true potential with a simpler one.

The effective Hamiltonian is given by

$$H = \sum_{i=1}^{N}\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial r_i^2} + V(r_i)\right) + \sum_{i<j}\frac{4\pi\hbar^2 a_s}{m}\delta(r_i - r_j), \tag{3.5}$$

where m is the mass of the boson, V is the external potential, $\delta(r)$ the Dirac delta-function and U_0 contains the boson-boson scattering length a_s.

We can use a variational approach in order to approximate the wave function to the condensate. First we assume the condensate wave function to be $\psi(\mathbf{r}) = \sqrt{N}\phi(\mathbf{r})$. The density of particles is

$$n(\mathbf{r}) = |\psi(\mathbf{r})|^2,$$

and our energy function may be written

$$E(\psi) = \int d\mathbf{r}\left[\frac{\hbar^2}{2m}|\nabla^2\psi(\mathbf{r})|^2 + V(r)|\psi(\mathbf{r})|^2 + \frac{2\pi\hbar^2 a_s}{m}|\psi(\mathbf{r})|^4\right]. \tag{3.6}$$

Under the condition that the number of particles be conserved

$$N = \int d\mathbf{r}|\psi(\mathbf{r})|^2,$$

we minimize the energy functional under variations of ψ and ψ^* using the Lagrangian multiplier [1] μ

[1]The method of Lagrange multiplier gives us the maxima and minima of a function with one or more constraints. If $f, g : \mathbb{R}^m \to \mathbb{R}$ are two functions with continuous partial derivatives and $x = (x_1, x_2, ..., x_m)$ is a local min. or max. point for f, then either $\Delta g(x) = 0$ or there is a constant μ that satisfy $\nabla f(x) = \mu\nabla g(x)$, [19]

$$\frac{\delta}{\delta\psi^*}(E(\psi) - \mu N(\psi)) = 0. \tag{3.7}$$

Using equation (3.7) we get the Euler-Lagrange equation which is the Gross-Pitaevskii equation

$$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial r^2} + V(r) + \frac{4\pi\hbar^2 a_s}{m}|\psi(r)|^2\right)\psi(r) = \mu\psi(r). \tag{3.8}$$

The first term in the Hamiltonian we recognise as the kinetic energy term, $U_0 = \frac{4\pi\hbar^2 a_s}{m}$ is the effective interaction between two particles at low energy, a constant in the momentum representation. In coordinate space this corresponds to a contact interaction $U_0\delta(\mathbf{r} - \mathbf{r'})$. For a uniform Bose gas, the GP equation is very simple

$$\mu = U_0|\psi(\mathbf{r})|^2 = U_0 n.$$

3.2.2 Basic scattering theory

In the theory of the GP equation the interaction between particles in a trapped gas is handled by an effective interaction potential. However it can be important to know something about what happens when two particles in a low-energy system as a cold trapped bose gas approaches one another. We will describe some simple two-body scattering theory.

As mentioned above we use a mean-field approach on the wave function (3.4). By applying a correlation operator to the wave function we can in a simple way take the scattering effect to account.

We can describe scattering between two particles with no internal degree of freedom as

$$\psi = e^{ikz} + \psi_{sc}(\mathbf{r}),$$

where we have set the incoming plane wave to be in the z-directon [11]. At low energies s-wave [2] scattering is sufficient to consider the interaction between atoms and the wave function can be written as

$$\psi = e^{ikz} + f(\theta)\frac{e^{ikr}}{r},$$

where $f(\theta)$ is called the scattering angle amplitude and depends only on the angle between the two particles after scattering. It reaches a constant value for low energies, and is simply the scattering length mentioned above. We can say that the potential for a hard-sphere boson is the scattering length

$$\psi = 1 - \frac{a}{r}. \tag{3.9}$$

[2]The s-wave takes a spherical form and is called an s-wave with the s referring to the atomic s-orbital with angular momentum quantum number $l = 0$.

For identical bosons the scattering cross section is simply $\sigma = 8\pi a^2$.

3.2.3 The ground-state energy

For a non-interacting Bose-gas confined in a trap the two contributions to the energy is the kinetic and the potential energy, and they go as the following

$$V_{trap}(\mathbf{r}) = \frac{1}{2}m\omega_0^2 r^2, \quad K(\mathbf{r}) = \frac{\hbar^2}{2mr^2},$$

where ω_0^2 is the trap potential strength. The total energy has a minimum when the kinetic energy and the potential energy are the same. Setting the two expressions equal we can calculate at which radius the gas is at an equilibrium state

$$a_{osc} = \left(\frac{\hbar}{m\omega_0}\right)^{1/2}.$$

a_{osc} is the quantum-mechanical length scale of the harmonic oscillator. Given that our trap potential is the oscillator potential, this makes perfect sense. In appendix A.1.1 we show that the expectation value of the total energy per particle is given by

$$\frac{E}{N} = \frac{3\omega_0}{2}.$$

Now lets take interaction between the particles into account and the expression of the total energy in such a case. The particle density goes of the order $n \sim N/r^3$, and the interaction energy depends on the scattering length typically like na. With increasing value of the interaction energy, the importance of the kinetic energy decreases. As before we may find the equilibrium radius, now by minimizing the sum of the interaction energy and the potential energy, neglecting kinetic energy and we get

$$R \sim a_{osc}\left(\frac{Na}{a_{osc}}\right)^{2/5}.$$

The result we may use to approximate the total energy of the steady state of the gas

$$\frac{E}{N} \sim \hbar\omega_0\left(\frac{Na}{a_{osc}}.\right)^{2/5}$$

Here a/a_{osc} is a dimensionless quantity that describes the interaction strength between the particles. As mentioned in the previous section, a is the scattering length, for hard-sphere bosons, the atom size. Which makes a/a_{osc} simply the atom size to trap size ratio.

3.3 Strong interacting Bose gases

When it comes to strong interacting Bose gases, the idea of the single-particle description of the wave function breaks down. Due to higher densities the mean-field approach is inapplicable and one must turn to other methods.

Systems with strong correlations are often more difficult to solve but of great importance. There have been many studies exploring the physic behind ultracold gases of bosons and fermions over the last years [20] [21]. One of the system of high interest being liquid Helium, particularly Helium-4. Liquid Helium is an exception to the rule that liquids solidify at low temperature. When cooled sufficiently, below 2.2 K, or under pressure liquid Helium becomes a superfluid, called helium II. This is not a subject for this thesis, but instead lets look at Helium-4 as a non-superfluid liquid, the helium I.

3.3.1 Liquid Helium-4

Stable helium comes in form of two isotopes, the rare helium-3 and helium-4. Most helium in the world is of the latter. Helium-4 consists of two protons and two neutrons in the core. It is a spin 1 atom, i.e. it is a boson.

In 1956 Penrose and Osanger gave a theoretical proof that Bose-condensate indeed occurs in liquid helium-4 [22]. And the study of Helium-3 (spin 1/2 fermion), where no condensate happens, confirmed the theory that particle statistics is important in such systems [23].

But as mentioned, in the case of strong correlation between the atoms, a single-particle wave function with a correlation factor is not a sufficient representation for such a system. The dominating forces are short-range two-body or possibly three-body interactions, and a mean-field approach for the potential falls apart. The typical density of Rubidium-87 in a trap is of the order $10^{12} - 10^{14}$ atoms/cm^3 [24] where as for liquid Helium-4 the density is of the order 10^{22} atoms/cm^3. With the increasing density of the system we are no longer dealing with a dilute weak-interacting gas.

For the liquid helium-4 one can assume a Hamiltonian given as

$$\hat{H} = \sum_{i=1}^{N} -\frac{\hbar}{2m}\nabla_i^2 + \sum_{i<j} V(r_{ij}). \tag{3.10}$$

In the absence of an external field the dominating forces of the system are the Van der Waals forces, an attractive force between the atoms, and the repulsive force between the electrons, coming from the overlapping of electron orbitals. So we use the two-body Lennard-Jones potential to describe the external forces acting on each of the particles.

Lennard-Jones potential

The Lennard-Jones potential is a mathematical model for describing the total interaction between neutral atoms, both the attraction and the repulsion are due to the different

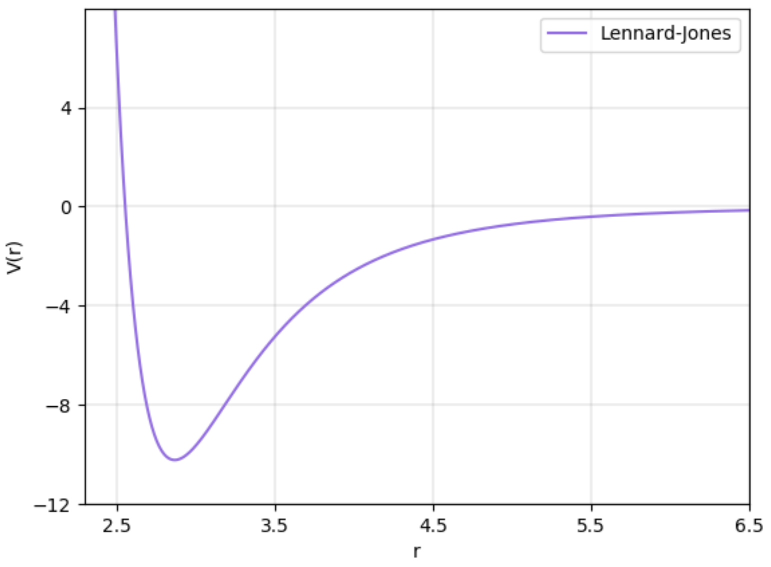

Figure 3.2: The Lennard-Jones potential.

components in the atom.

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right]. \tag{3.11}$$

The first term represents the short range repulsive force and the second term is the attractive long-range Van Der Waals force. ϵ is the depth of the potential well and σ determines the particle distance at which the potential is zero, Figure 3.2. We should mention that this potential has no theoretical background, but is an approximation derived from experiments and perturbation theory [25].

For this system a suggestion for the wave function has been given by McMillian [26],[27] with good results

$$\psi_T = \exp\left(-\frac{1}{2} \sum_{i<j} \left(\frac{\beta}{r_{ij}} \right)^{5} \right). \tag{3.12}$$

It is a good choice as it is small for large value of the potential, i.e. small values of r. And for small value of the potential the wave function reaches a constant value.

Chapter 4
Quantum Monte Carlo

Monte Carlo methods may be extended to solve quantum mechanical problems. In section 2.2.1 we explained how the wave function is given by the probability distribution function (2.6), it tells us the likeliness of a particle being in a certain place at a certain time. And more often than not, we do not know the exact wave function. So how can we know anything about how the particles move over time? Quantum Monte Carlo (QMC) is a range of methods that deal with this problem, i.e. solving the Schrödinger equation, by use of random walkers.

As quantum mechanics possesses some randomness, whether this randomness be of apparent nature or an intrinsic on[1], it is there. What QMC does is exploiting the statistical features of Monte Carlo simulation in order to handle this randomness through different sampling techniques. In this thesis we have used one of the most common methods for solving quantum systems, the Variational Monte Carlo (VMC). In the following chapter we will explain how we solve the eigenvalue problem that is the Schrödinger equation, more specific for the stationary state of the Hamiltonian, by use of the variational principle.

What we seek is the expectation value of the Hamiltonian $\langle\psi|\hat{H}|\psi\rangle$ which is given by (2.8). For an increasing number of particles this becomes a complex multi dimensional integral that can not be solved analytical or by a conventional numerical approach. So we turn to VMC.

Much of the content of this next chapter has been gathered from the lecture notes in Computational Physics by Morten Hjorth-Jensen [28].

4.1 Variational Monte Carlo

Monte Carlo simulations are widely used methods in numerical science, that employs random walkers in order to typically solve complicated integration problems. Stochastic Monte Carlo simulations are well suited for problems where the system at hand can be described with a probability distribution function (PDF).

[1]This is a philosophical question in quantum theory, concerning our knowledge of a system (its initial state) in order to predict its evolvement in time. Does the randomness come from the fact that we do not have *all knowledge* of the system, or is it an inherent randomness?

If the PDF is known to us then we can sample directly from it. Usually we do not have it and must find it using random draw-out methods, like the Metropolis algorithm which we will look closer at in the following sections. Once we have the samples we can evaluate the sampled PDF by some optimization method, or in this case, a minimization method.

The Variational Monte Carlo (VMC) is a quantum Monte Carlo method that exploits the variational principle in order to find the ground-state energy of a system by use of variational parameters. So the minimization is done on the energy of the system. We begin by stating the variational principle.

Theorem 1 *(Variational principle) Consider the expectation value functional defined by*

$$\mathcal{E}(|\psi\rangle) \equiv \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle}. \tag{4.1}$$

Let $|\psi\rangle$ be given. Then $E_* = \mathcal{E}(|\psi\rangle)$ is a stationary value of $\mathcal{E}$ with respect to all infinitesimal variations $|\psi\rangle + \epsilon\,|\mu\rangle$ (with ϵ a small number and $\langle\mu|\mu\rangle = 1$) if and only is

$$\hat{H}\,|\psi_*\rangle = E_*\,|\psi_*\rangle\,.$$

The variational principle in its simplest form states that the ground-state energy E_0 is the minimum of the expectation value of the Hamiltonian [10]

$$E_0 \leq \langle H\rangle. \tag{4.2}$$

In VMC we take advantage of Theorem 1 and rewrite the expectation value problem of the Hamiltonian to a Monte Carlo solvable problem. From quantum mechanics we know the PDF is given by the wave function, like in (2.6). Since we do not know the exact wave function we rely on a trail wave function, $\Psi_T(\mathbf{r};\alpha)$, and let it be dependable on a variational parameter α. The PDF can be written

$$P(\mathbf{r};\alpha) = \frac{|\Psi_T(\mathbf{r};\alpha)|^2}{\int |\Psi_T(\mathbf{r};\alpha)|^2 d\mathbf{r}}. \tag{4.3}$$

If $\Psi_T(\mathbf{r};\alpha)$ is an eigenfunction from the solution of the time-independent Schrödinger equation (2.10), then we know that the expectation value of the Hamiltonian is the eigenvalue E, i.e. the energy. Lets rewrite the expression (4.1)

$$\langle\hat{H}\rangle = \frac{\int d\mathbf{r}\Psi_T^*(\mathbf{r})\hat{H}\Psi_T(\mathbf{r})}{\int d\mathbf{r}\Psi_T^*(\mathbf{r})\Psi_T(\mathbf{r})} = \frac{\int d\mathbf{r}|\Psi_T(\mathbf{r})|^2\frac{\hat{H}\Psi_T}{\psi_T}}{\int d\mathbf{r}|\Psi_T(\mathbf{r})|^2},$$

$$= \int d\mathbf{r}P(\mathbf{r})\frac{\hat{H}\Psi_T(\mathbf{r})}{\psi_T(\mathbf{r})} = \int d\mathbf{r}P(\mathbf{r})\hat{E}_L(\mathbf{r}).$$

In the last expression we have defined a new operator, $\hat{E}_L$, called the local energy

$$\hat{E}_L(\mathbf{r};\alpha) = \frac{1}{\Psi_T(\mathbf{r};\alpha)}\hat{H}\Psi_T(\mathbf{r};\alpha). \tag{4.4}$$

Now the variational energy can be estimated as the expectation value of the local energy. This is done by the Monte Carlo integration method through

$$E_v \approx \langle \hat{E}_L \rangle = \int P(\mathbf{r})\hat{E}_L d\mathbf{r} \approx \frac{1}{N}\sum_{i=1}^{N} \hat{E}_L(x_i). \tag{4.5}$$

This is what the Variational Monte Carlo method bases itself on. Given a probability distribution we can evaluate the wave function and look for a local minimum in the energy. Which is practical given we avoid an analytical integration involving the wave function. However we are highly reliable on a good guess on the trail wave function. It should contain all of the important features of the exact wave function since all observables are evaluated with respect to the PDF.

4.1.1 Monte Carlo Integration

One of the most simple examples of Monte Carlo calculations is numerical integration. In standard numerical discretization methods, like Euler method or Runge-Kutta, one uses evenly spaced sub-intervals to solve the integral whereas in Monte Carlo integration (MCI) one uses random numbers. MCI estimates the definite integral I by drawing N random samples over the interval $[a, b]$, with $\frac{b-a}{N}$ indicating the step length

$$I = \int_a^b f(x)dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i).$$

With the random variable x_i we introduce the associated PDF, $p(x_i)$, containing information of the frequency of the variables occurring over the sampling interval. From before we know that the expectation value, or the average of the function f, is given by

$$\langle f \rangle = \frac{1}{N}\sum_{i=1}^{N} p(x_i)f(x_i),$$

for $0 \leq x_i \leq 1$ when $p(x_i)$ is normalized. It is possible to solve problems for other than the uniform distribution, but the interval $x_i \in [0, 1]$ is sufficient for this thesis. With the PDF being normalized to one, the expression above simply gives the estimation of I

$$I = \int_0^1 f(x)dx \approx \langle f \rangle. \tag{4.6}$$

4.2 Sampling technique

The typical process of a Monte Carlo simulation involves the following steps.

Basic Monte Carlo algorithm

- Generate a random number
- Take/don't take a new random step based on the random number
- Repeat trail

Obviously the way of generating a random number can be done in many different ways and is something to be thought of. But here we will focus on the step involving the selection of accepting a new random move in the configuration space. This process governs the evolvement of the system into the state we seek. Since the true wave function is unknown to us we cannot sample from the PDF directly. Therefore we are in need of a good way of selecting new steps, as one is in all Monte Carlo methods. In this thesis we will make us of both the Metropolis algorithm and importance sampling, which both use Markov processes. When we get to the machine learning part we will look at another type of sampling, Gibbs sampling, also based on Markov processes. So lets look closer at what these are.

4.2.1 Markov chains

A Markov chain is a stochastic process in which the conditional probability of the transition from one event to another $\mathbf{r}_k$ depends solely on the previous event $\mathbf{r}_{k-1}$

$$P(\mathbf{r}_k|\mathbf{r}_{k-1}, ..., \mathbf{r}_1) = P(\mathbf{r}_k|\mathbf{r}_{k-1}).$$

So for a particle e.g., moving through the sequence of positions $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2)$, the probability of this transition happening is given by

$$P(\mathbf{r}_2, \mathbf{r}_1, \mathbf{r}_0) = P(\mathbf{r}_2|\mathbf{r}_1)P(\mathbf{r}_1|\mathbf{r}_0)P(\mathbf{r}_0).$$

Compared to a standard stochastic process with no Markov chain the same probability is given as

$$P(\mathbf{r}_2, \mathbf{r}_1, \mathbf{r}_0) = P(\mathbf{r}_2|\mathbf{r}_1|\mathbf{r}_0)P(\mathbf{r}_1|\mathbf{r}_0)P(\mathbf{r}_0).$$

The probability distribution $P(\mathbf{r}_j|\mathbf{r}_i)$ is called the transition probability. It gives the probability of moving from state $\mathbf{r}_i$ to state $\mathbf{r}_j$ and is a *stochastic matrix*.

Now we assume there exists a stationary state $\rho(\mathbf{r})$ of the transition probability representing the true probability. What we hope for is that after some time sampling $P(\mathbf{r}_j|\mathbf{r}_i)$ we will reach an equilibrium. That what we in the end are sampling from is the stationary distribution $\rho(\mathbf{r})$, our target probability distribution. For this to be the case we demand $\rho(\mathbf{r})$ to be an eigenvector of $P(\mathbf{r}_j|\mathbf{r}_i)$ and the transition distribution be normalized (which it is being a stochastic matrix). Given all possible new steps, the particle must move to one of them

$$\int d\mathbf{r}_j P(\mathbf{r}_j|\mathbf{r}_i) = 1.$$

However there are two more important conditions to be made in order for the sampling to give the 'true' distribution. We want the walker to be able to move to any configuration within the space that the distribution function spans. So we require that the sampling be *ergodic*. And finally the last condition on the Markov chain. The condition of *detailed balance.*

$$P(\mathbf{r}_j|\mathbf{r}_i)\rho(\mathbf{r}_i) = P(\mathbf{r}_i|\mathbf{r}_j)\rho(\mathbf{r}_j)$$

This implies the Markov chain be reversible. The probability flux between the two states $\mathbf{r}_i$ and $\mathbf{r}_j$ to be the same in both directions [29].

4.2.2 The Metropolis Algorithm

In the paper by Metropolis [30] he suggested a symmetric proposal matrix in order to satisfy the condition of detailed balanced. We will describe closer how this works.

What the Metropolis algorithm does it creating a Markov chain whose stationary distribution is our target distribution $\rho(\mathbf{r})$. To do so we begin with the transition distribution P, or $P(\mathbf{r}_j|\mathbf{r}_i)$, the probability of finding the system in state j given state i. The main idea is to suggest a new move $\mathbf{r}'_j$ in configuration space and find the likeliness of this transition taking place. Given some conditions we either accept the move, meaning $\mathbf{r}_j = \mathbf{r}'_j$, or we reject the suggested move, i.e. $\mathbf{r}_j = \mathbf{r}_i$.

Now we introduce two probabilities; $A(\mathbf{r}_j|\mathbf{r}_i)$ as an acceptance probability and $T(\mathbf{r}_j|\mathbf{r}_i)$ is a proposal probability, the one mentioned in the beginning of this section. We suggest the transition probability be constructed by these two probabilities in the following way

$$P(\mathbf{r}_j|\mathbf{r}_i) = A(\mathbf{r}_j|\mathbf{r}_i)T(\mathbf{r}_j|\mathbf{r}_i).$$

For a suggested new move $\mathbf{r}'_j$ we can write the transition probability as the following

$$P(\mathbf{r}_j|\mathbf{r}_i) = A(\mathbf{r}_j|\mathbf{r}_i)T(\mathbf{r}_j|\mathbf{r}_i) + [1 - \int d\mathbf{r}'_j A(\mathbf{r}'_j|\mathbf{r}_i)T(\mathbf{r}'_j|\mathbf{r}_i)]\delta(\mathbf{r}_i - \mathbf{r}_j).$$

This equation ensure all the conditions mentioned before in order for the Markov chain to reach an stationary state. The proposal probability $T(\mathbf{r}_j|\mathbf{r}_i)$ is a stochastic matrix, the acceptance probability $A(\mathbf{r}_j|\mathbf{r}_i)$ however is not. It must be selected so as to fulfill the condition of detailed balance. One possibility is

$$\frac{A(\mathbf{r}_j|\mathbf{r}_i)}{A(\mathbf{r}_i|\mathbf{r}_j)} = \frac{\rho(\mathbf{r}_j)T(\mathbf{r}_i|\mathbf{r}_j)}{\rho(\mathbf{r}_i)T(\mathbf{r}_j|\mathbf{r}_i)}.$$

By sampling enough times we suggest that these ratios are equal. The fact that we are only dealing with ratios makes it less complicated handling the probability density $\rho(\mathbf{r})$, there is no need to calculate its normalization factor. We also notice that if $T(\mathbf{r}_j|\mathbf{r}_i)$ where to be a symmetric matrix then it would drop out of the equation all together. However, the only constraint on this probability is that it must be a stochastic matrix.

Now we want to push the system towards regions in space where the density of the transition probability is high, in that way for the system to converge to the desired stationary distribution. This is the machinery of the algorithm, it maximizes the acceptance probability

$$A(\mathbf{r}_j|\mathbf{r}_i) = \min\left(1, \frac{\rho(\mathbf{r}_j)T(\mathbf{r}_i|\mathbf{r}_j)}{\rho(\mathbf{r}_i)T(\mathbf{r}_j|\mathbf{r}_i)}\right). \tag{4.7}$$

Now the general Metropolis algorithm takes the following procedure.

1. Generate initial values $\mathbf{r}_0$
2. Generate a trail value $\mathbf{r}'_j$ with probability $T(\mathbf{r}_j|\mathbf{r}_i)$
3. Accept the trail value with the probability $min\{1, A(\mathbf{r}_j|\mathbf{r}_i)\}$
4. In case of acceptance, set the new value $\mathbf{r}_j = \mathbf{r}'_j$
5. Repeat 1. - 4.

The easiest choice of $T(\mathbf{r}_j|\mathbf{r}_i)$ is a uniform distribution which region is that of a small cube with $\mathbf{r}_i$ as the center and side length Δ. If we use this then a new suggested step is given as

$$\mathbf{r}'_j = \mathbf{r}_i + \Delta\mathbf{r}\ \chi,$$

where χ is a random number from the uniform distribution between $[-1, 1]$. A sketch of the full algorithm of the Metropolis may be seen in Algorithm 1.

4.2.3 Importance Sampling

Now we move to an extension of the Metropolis algorithm, the Importance sampling. This method provides us with a better way of suggesting new moves. Where the Metropolis algorithm may suggest new moves in every direction with the same probability, importance sampling pushes the configuration towards higher probability densities by the use of a drift force. The expression for the new move, y, is given by the solution of the Langevin equation. This equation is originally used to solve Brownian motion in molecular dynamics. It give the time evolution of particles experiencing small variations in variable change

$$\frac{\partial x(t)}{\partial t} = DF(x(t)) + \xi,$$

where ξ is a gaussian random variable, D is the diffusion coefficient which is $\frac{1}{2}$ and Δt is the time step which take values between $[0.001, 0.01]$. Solving this partial differential equation we get the selection of the new steps

```
number of Monte-Carlo cycles $N_{MC}$;
initialize $\mathbf{r}$;
calculate $|\Psi_T(\mathbf{r})|^2$;
for $i$ in $[0, N_{MC}]$ do
    $\mathbf{r}_p = \mathbf{r} + r * \Delta\mathbf{r}$;
    calculate $\omega = \frac{|\Psi_T(\mathbf{r}_p)|^2}{|\Psi_T(\mathbf{r})|^2}$;
    if $q \leq \omega$ then
        accept new move;
    else
        reject new move;
    end
    update energy, $\mathbf{E}_L$;
end
```

Algorithm 1: The full algorithm for Monte Carlo with Metropolis-Hastings sampling of configurations.

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t},$$

with the drift force $F(x)$ given by the gradient of the wave function

$$F = 2\frac{1}{\Psi_T}\nabla\Psi_T.$$

It is the drift force that ensures us we move particles towards regions of configuration space where the trail wave function is large. Now that we have the equation for selecting steps we only need the new proposal probability $T(\mathbf{r}_j|\mathbf{r}_i)$. A good choice is the solution of the Fokker-Planck equation[28]

$$\frac{\partial T(\mathbf{r},t)}{\partial t} = D\frac{\partial}{\partial x}\left(-\frac{\partial}{\partial x} + F(x(t))\right)T(\mathbf{r},t),$$

which is the Green's function

$$G(y,x,\Delta t) = \frac{1}{(4\pi D\Delta t)^{3N/2}}\exp\bigl(-(y - x - DF(x)\Delta t)^2/4D\Delta t\bigr).$$

We must calculate the drift force in order to find new moves and the Green's function to accept/reject these moves. With ξ being a random number drawn from a Gaussian PDF we are much more likely to draw from the exact distribution. The reason for this being that our system is not represented by a uniform distribution, but more likely shaped as a Gaussian.

4.3 Optimization

The goal in VMC is finding the ground state energy of a system, or come as close to E_0 as possible. And the way to search for E_0 is to vary the variational parameter α by using an optimization algorithm. The job of the algorithm is to adjust the parameter using a cost function. The cost function tells us something about how close our estimation are to the actual solution. In our case, how close our estimation on the energy is. Since we seek the ground state energy we want the system to move towards lower energy states. Our cost function may therefore be the energy gradient ∇E.

The simplest and most famous optimization gradient method is the gradient descent.

4.3.1 Gradient Descent

We want to push the system towards lower energy states. Meaning we want to move in the direction of the negative of the energy gradient. In standard gradient descent we iteratively adjust the parameter θ according to

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta E(\theta_i). \tag{4.8}$$

The new parameter value θ_{t+1} is given by the previous one plus the learning rate η_t times the energy gradient. Usually the first parameters are given random variables, or one may give a guess. The learning rate controls how large steps we take for each iteration. The easiest way to implement the learning rate is by giving it a constant value. A to large learning rate may cause the model to diverge, a to low learning rate causes the method iterating to slowly, and one ends up using unnecessary CPU time. But given an optimal value or close to one the gradient descent will converge towards the minima. One can however introduce a changing learning rate. This can help us avoid problems like overshooting or oscillating between two points. Such adaptive step size methods can be Newton's method, backtracking line search, Cauchy or Barzilai and Borwein. Algorithm 2 shows the algorithm of the regular gradient descent, running over gradient iterations, then over Monte-Carlo cycles.

Gradient descent has some drawbacks, one being that it can get stuck in one local minima and never reaching the correct one. Another is that it is sensitvive to initial conditions, so what values we give the parameters in the begining matters. It is also somewhat computational expensive. A solution to this is stochastic gradient descent.

4.3.2 Stochastic gradient descent

In regular gradient descent the algorithm simply updates the parameter θ_{i+1} according to (4.8). A way to avoid correlation in the data and to speed up the process of optimization is to use the stochastic gradient descent. This method, indicated by its name, introduces stochastic behavior. If the amount of data we are to analyse is n data points of some sort, we can divide the data into smaller minibatches of size M creating n/M batches. We then

```
1 number of gradient descent $N$;
2 number of Monte-Carlo cycles $N_{MC}$;
3 initialize $\alpha$;
4 for $i$ in $N$ do
5     for $j$ in $N_{MC}$ do
6         calculate $\nabla E_L$
7     end
8     $\alpha(i+1) = \alpha(i) - \gamma \nabla E_L$
9 end
```

Algorithm 2: Regular gradient descent for updating variational parameter α by use of (4.8) where $\nabla E_L = \frac{\partial E_L}{\partial \alpha}$. After each entire Monte-Carlo cycle we use the newest estimation of the gradient to update α.

denote these minibatches B_k running from $k = 1...n/M$. Our energy gradient can then be rewritten in the following way

$$\nabla_\theta E(\theta) = \sum_i^n \nabla_\theta e_i(\mathbf{x}_i, \theta) \longrightarrow \sum_{i \in B_k} \nabla_\theta e_i(\mathbf{x}_i, \theta),$$

In stead of going through the entire data, the new energy gradient comes from the calculation over one minibatch B_k

$$\nabla_\theta E^{MB}(\theta) = \sum_{i \in B_k}^M \nabla_\theta e_i(\mathbf{x}_i, \theta).$$

We can now solve the gradient descent for the new minibatches and update the parameter according to

$$\theta_{i+1} = \theta_{\mathbf{i}} - \eta \nabla_\theta E^{MB}(\theta). \tag{4.9}$$

We cycle over all minibatches by random and a full cycle is called an epoch.

4.4 Statistical Errors

In this entire thesis we deal with random numbers a lot, and calculation and estimation using random numbers give rise to statistical errors. Especially one very important property to evaluate is the variance. From the central limit theorem we get the variance of the probability dependent on a stochastic variable x. The variance is given as

$$\begin{aligned}
\sigma_x^2 &= \mathrm{Var}(x) = \left\langle (x - \langle x \rangle)^2 \right\rangle = \int (x - \langle x \rangle)^2 p(x) dx, \\
&= \int (x - 2x \langle x \rangle + \langle x \rangle)^2 p(x) dx, \\
&= \left\langle x^2 \right\rangle - 2x \langle x \rangle \langle x \rangle + \langle x \rangle^2, \\
&= \left\langle x^2 \right\rangle - \langle x \rangle^2 .
\end{aligned}$$

It is the average of the squared difference of the mean (4.10). The square root of the variance, $\sigma_x^2 = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}$ is called the standard deviation of $p(x)$. If the probability is given by uncorrelated events then the estimated standard deviation, when sampling $p(x)$ with N samples, may be written

$$\sigma_N \approx \sqrt{\frac{1}{N}\left(\langle x^2 \rangle - \langle x \rangle^2 \right)} = \frac{\sigma_x}{\sqrt{N}}.$$

In our case, sampling the probability density $P(\mathbf{r})$ from a sequence of points $\mathbf{r}_i$ through the Metropolis algorithm makes these non-independent (i.e. correlated) as each new sample is selected based on the previous one. See Eq. (4.3). Dealing with correlated variables the calculations of the standard deviation is not so straight forward. We will have to make use of some statistical techniques in order to solve the dependencies. We will describe one of the methods in the next section.

4.4.1 Blocking Method

The theory of blocking and the code used in this thesis both come from the work of Marius Jonsson [31]. The idea behind blocking is to give a statistical analysis of the estimated values post run-time. We are dealing with randomized systems and ideally we would run sampling and make calculations infinitely many times to get the best possible estimate of the energy. This is time-consuming and something we wish to avoid. Hence the blocking method. It can estimate the error of our calculations by use of one simple sequence of data $\mathbf{x} = (x_1, x_2, x_3, \cdots, x_n)$, compared to using many sequences.

The most typical estimator of the expectation value, as we talked about in previous chapters, is the mean value

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{4.10}$$

with x_i being one sampled variable or observable, and n is the total number of samples. We repeat the sample variance from the last section

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2.$$

If our samples were uncorrelated we could use this expression to find the standard error, or the variance. But our samples are not independent of one another. This is where we use blocking. Lets say $n = 2^d$ with d being an integer value. We can then apply blocking transformation to the data according to

$$(x_0)_k \equiv (x)_k,$$
$$(x_{i+1})_k \equiv \frac{1}{2}\left((x_i)_{2k-1} + (x_i)_{2k}\right),$$

where k indicates the number of blocking transformations and $0 \leq i \leq d-1$. The covariance between one block i and another block j, with $h = |i - j|$, can then be calculated by

$$\gamma_{k+1}(h) = \frac{1}{4}\gamma_k(2h-1) + \frac{1}{2}\gamma_k(2h) + \frac{1}{4}\gamma_k(2h+1).$$

If $\gamma_k(0) = \sigma_k^2$ then the variance may be found by

$$\mathrm{var}(\bar{x}_k) = \frac{\sigma_k^2}{n_k} + \frac{2}{n_k}\sum_{h=1}^{n_k-1}\left(1 - \frac{h}{n_k}\right)\gamma_k(h) = \frac{\sigma_k^2}{n_k} + e_k.$$

It can then be shown that $\mathrm{var}(\bar{x}_i) = \mathrm{var}(\bar{x}_j)$ for all $0 \leq i \leq d-1$ and $0 \leq j \leq d-1$.

4.4.2 One-body density

In quantum mechanics a nice way to illustrate how the system acts is by calculating the one-body density function. This is done by integrating over all particles but one

$$\rho(\mathbf{r}_1) = \int d\mathbf{r}_2 \ldots d\mathbf{r}_N |\Psi(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)|^2. \tag{4.11}$$

This is a probability density function for particle 1. If we did this for all particles we could get a sense of how the system looks like as $\rho(\mathbf{r})$ would give the relative distribution of all particles. This however is a difficult property to solve as (4.11) is a multidimensional integral. There is a way to avoid the integral but still get the property $\rho(\mathbf{r})$.

Lets say we are dealing with a system of particles in 3 dimensions. We create a sphere around the particles with the center in origo. If we section this sphere into several rings with equal distance Δr from one another, we can call these sections bins, we then count how many particles are inside each bin. We can then visualizing this by use of a histogram and that would tell us how the probability distribution function looks like.

4.5 Numerical differentiation

Lastly in this chapter we mention the method of numerical differentiation as it has been used as a backup method for the derivatives used in the expression of the local energy. To evaluate the local energy of the system it is necessary to compute the second derivative of the trial wave function ψ_T. The method of finite difference approximates derivatives by an iteration scheme. In case of the first derivative we have used the forward difference scheme

$$f'(x) = \frac{f(x+h) - f(x)}{h},$$

with a truncation error going as $\mathcal{O}(h)$. And the second-order central in case of the second derivative

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2},$$

with a truncation error going as $\mathcal{O}(h)$. This method however comes at highly expensive computational costs. It would be preferable to use other expressions for the derivatives, which we shall.

Chapter 5
Machine Learning

What is machine learning? First of all it is a fairly wide term. It can be everything from speech recognition [32] to medical diagnosis [33], or lately even algorithms that can solve complicated classical statistical problems within physics, like the Ising model [34].

Machine learning (ML) is a subfield of artificial intelligence (AI), a field that evolved from the study of the human brain where the idea is to make the computer learn, or remember, by constructing a network of neurons linked to each other. Much like how synaptic connections in the brain link neurons together. These connection are either strong or weak dependent on how important they are. This process is in machine learning handled by what we call activation functions.

In essence ML is to learn from, and make predictions about, data. Basically we have some data that we want to analyse, and given a function for measuring, the computer is able to find patterns in the data. One distinguish between estimating and predicting new data. Most common is probably estimating. Nevertheless we can describe a general approach to apply ML to a problem.

Suppose we are looking at a system with a given observable quantity $\mathbf{x}$. Related to this quantity is a parameter $\mathbf{w}$ and a model $p(\mathbf{x}|\mathbf{w})$ which describes the probability of $\mathbf{x}$ given $\mathbf{w}$. We do not know the model, i.e. we do not know the relationship between the two variables. Now we are given a dataset $\mathbf{X}$. What we wish to do is fit the model to these data, meaning we seek a $\hat{\mathbf{w}}$ that can increases the probability of us observing the data. This can be done using some maximization function $\hat{\mathbf{w}} = \text{argmax}_{\mathbf{w}} p(\mathbf{X}|\mathbf{w})$. In machine learning the parameter $\hat{\mathbf{w}}$ is called the weights and they are to be adjusted so that the model fit the data. If this was an estimation problem we would want to find the most accurate $\hat{\mathbf{w}}$. Where it a prediction problem on the other hand, we would be concerned with the accuracy of the model $p(\mathbf{x}|\mathbf{w})$ being able to predict new observations.

Some ML techniques are closely related to computer science, other have strong resemblance to classical statistics. Usually one talks of three main groups of techniques in ML; supervised, unsupervised and reinforcement learning. However it may be difficult to separate them from one another sometimes. But here are the main characteristics.

- Supervised learning: are discriminative models, typically used for classification or regression

- Unsupervised learning: for clustering, looks for similarities in data
- Reinforcement learning: a search method using trail-and-error

ML is a field of research which is getting more and more significance. With the amount of data produced in the world today, we need better and faster methods to analyse them. It is also getting more attention in businesses and in industry, becoming a topic of interest. This mainly because ML comprises methods that can be applied to several branches of science; finance, physics, chemistry, medicine, etc., and is already of great importance in areas such as cosmology, engineering and biophysics.

Although ML may seem like something that has been discovered during the recent decade its history actually stretches back to the 40s when neurophysiologist Warren McCulloch and mathematician Walter Pitts suggested representing neurons in the brain by electric circuits [35]. Their neuron model consisted of a set of inputs with connected weights (corresponding to synapses), an adder summing up the input signals, and an activation function (threshold function) determine whether the neuron fires or not. This was the beginning of neural nets.

Then in 1958 Frank Rosenblatt designed the first artificial neural network, the Perceptron. A collection of McMulloch and Pitts neurons putt together.

However the first computer program that could learn as it ran was developed by Arthur Samuel in 1952 working at IBM [36]. It was a program that could play checkers. Some years later he was probably the first to use the term "machine learning" in a paper where he described the theory behind the program, as a look-ahead procedure [37].

Through the next couple of decades the field had a period of low activity and with little progress. Then the interest started picking up again during the 80s, and by the 90's many exiting achievements where done, like Deep Blue, a computer program that could play chess with the goal to beat the world chess champion at the time, Garry Kasparov. It did so in 1997, not without accusation of cheating from the defeated man him self. The next year, 1998, AT& T Bell Laboratories detecting handwritten postcodes from the US Postal Service. Since then machine learning has become a field of enormous interest in almost every part of the scientific world.

In this thesis we are dealing with the restricted Boltzmann machine, a reinforcement learning method based on the Perceptron. This is a network reinvented by Geoffrey Hinton in 80s. But before we go further into the restricted Boltzmann machine we'll look a bit closer on some other types of ML techniques, and start with one of the most easiest to understand, linear regression. Its a supervised learning method.

5.1 Supervised learning

In supervised learning we must require data, otherwise we could not supervise the learning process to know whether it was learning correct or not. So the entire method is based on the presumption that we have labeled data, an input to feed the algorithm. Lets say these data come in a dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is a matrix of independent variables and $\mathbf{y}$ is a vector of dependent variables, also called the output or the response. Now the model is a function that tries to predict an output θ given the input, $f(\mathbf{x};\theta)$. Our goal in the training procedure involves getting the algorithm to generalise the response correctly for all input. In order to tell if the model actually predicts correct we use a cost function, $\mathcal{C}(\mathbf{y}, f(\mathbf{X};\theta))$ that tells us something about how much the prediction deviates from the actual output. What we hope for is that it is as close as possible, meaning we want to minimize the cost function. Fitting the model is therefore done by minimizing the cost function.

Typically one would divide the dataset into two, a training set $\{\mathbf{X}_{train}, \mathbf{y}_{train}\}$, and a test set $\{\mathbf{X}_{test}, \mathbf{y}_{test}\}$. Obviously the first one is used to train the model and then one can use the rest of the data to evaluate the model, $\mathcal{C}(\mathbf{y}_{test}, f(\{\mathbf{X};\theta)\})$.

5.1.1 Linear Regression

Linear regression is one of the simplest machine learning techniques there is and a well known method in statistics. The method is much based upon the setup from last section. Our dataset, $\{(y_i, \mathbf{x}^{(i)})\}_{i=1}^{n}$, consists of of n samples where $\mathbf{x}^{(i)}$ is the i'th vector of the independent variables, and y_i is the corresponding response. We assume there is a linear relationship between the independent variable and the response. The function we seek, the function that describes the response given the data, we assume to be continuous

$$y_i = f(\mathbf{X}) = f(\mathbf{x}_i;\mathbf{w}) + \epsilon_i, \tag{5.1}$$

where $\mathbf{X}$ is called the design matrix, $\mathbf{X} \in \mathbb{R}^{n\times p}$, containing all the samples and the columns of the design matrix contains the features, $X^\top = \{X_0, X_1, X_2, X_3, ..., X_p\}$. $\mathbf{w}$ is the parameters we wish to obtain, $\mathbf{w} \in \mathbb{R}^p$, a vector to hold the coefficients explicitly describing the function we seek. A design matrix is given in (5.2) where the first column only containing one's corresponds to the intercept in $\mathbf{w}$ vector

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np}. \end{bmatrix} \tag{5.2}$$

If the input variables are linear then we can write one of the equations from (5.1) as the following

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + w_4 x_{i4} + ... + w_p x_{ip}.$$

These are the equations we wish to solve using the design matrix, and depending on the matrix having full column rank and the properties $\mathbf{X}^\top\mathbf{X}$ being positive definite we can find the solution. One method for solving this matrix issue is by least square regression.

The least square method selects the parameters in $\mathbf{w}$ so that residual sum of squares (RSS) is minimized

$$\mathrm{RSS}(w) = \sum_{i=1}^{N} (y_i - x_i^\top w)^2.$$

Searching the minimum of the RSS, we take its derivative wrt. w. And we get

$$\hat{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

Hat indicating it is the estimated value. This is the expression we use in the ordinary least square-method in order to find the optimal w-values. This method depends on the properties $\mathbf{X}^\top \mathbf{X}$ being positive definite in order to be able to calculate its inverse. In case it is not one must use other methods.

5.1.2 The Perceptron

In the previous section we looked at problems where the output is continuous data. Classification problem however can not be solved the same way. Here the output comes in form of discrete variables. Many problems in supervised learning are categorizing problems. Like making a network distinguish between pictures of dogs and cats e.g. The simplest way to solve this would be to put a sign function on the output data and map it to discrete variables like $\{0, 1\}$[1] and that way separate the categories. This method is called the Perceptron.

Before we begin to look at neural networks it can be motivating to understand the basic principles of these models, and possible the best way is by studying the Perceptron. Even the most complicated Artificial Neural Networks (ANN) are based on the model of the simple artificial neuron, which was first discovered when studying signal processing in the brain. The brain consists of millions of neurons connected to each other. These neurons, dependent on the input they receive from other neurons can either be inactive and do nothing or they can be activated and fire an output. These are processes dominating the activity in the brain. One can model these neurons in a simple way. Which is what McCulloch and Pitts did in 1943 [38].

The McCulloch and Pitts neuron must exceed an activation threshold in order to yield an output, i.e. to fire. If this threshold is not made then the neuron will stay inactive and give zero output. This is dependent on the input it gets

$$h = \sum_{i=1}^{m} w_i x_i.$$

The neural model consists of an adder that sums the input signals (x_i is the input and w_i its corresponding weight, together they form the signal), and an activation function

[1]The classification of dogs and cats is a typical Logistic Regression problem, a method similar to the Perceptron, but instead of the simple sign function the activation function comes in the form of sigmoid (logistic) function. Which has a simple derivative, making it very suitable when minimizing the cost function.

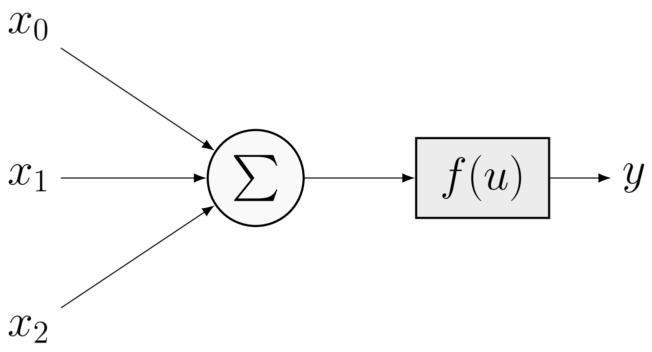

Figure 5.1: An illustration of a McCulloch-Pitts neuron model where the input goes through an adder, an activation function and results in an output y.

$g(h)$ which decides whether the neuron fires or not, depending on the input is above the threshold, θ or not

$$g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta. \end{cases} \tag{5.3}$$

There are of course many other choices of activation function than the step function (5.3), like the sigmoid function or the hyperbolic tangent function. So a more general mathematical description of the model is

$$y = f\left(\sum_{i=1}^{m} w_i x_i\right) = f(u). \tag{5.4}$$

The McCulloch and Pitts neural model is the most simple way to simulate a neuron, but one neuron does not do much and this model is not very useful for even the simplest problems within machine learning. We need more complex models.

The Perceptron is a collection of McCulloch and Pitts neurons connected together in the way visualised in Figure 5.2. The neurons are now formed together in a layer. The nodes take the input values and then they connect to the each neuron with a set of weights. If we compare with the simple neuron model above the weights now come in form of a matrix w_{ij} with the element w_{23} connecting the second input node with the third neuron. With the new layer of neurons Eq. (5.4) can be rewritten into an output function for the Perceptron

$$y_j = f\left(\sum_{i=1}^{m} w_{ij} x_i\right) = f(u). \tag{5.5}$$

The number of input nodes m is given by the data, and the number of neurons n depends on the number of features we wish the Perceptron to find. This is the goal, to make the Perceptron reproduce a particular target $\mathbf{t}$.

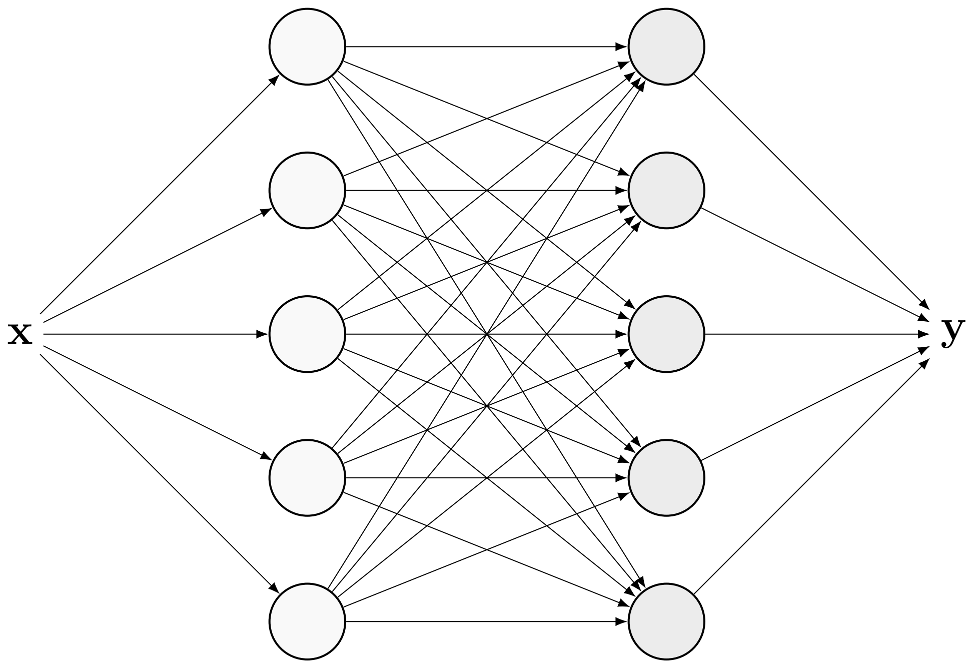

Figure 5.2: The Percetron is a collection of several McCulloch-Pitts neurons put together.

Given the target we can use an error function in order to fix the weights according to the difference between the output and the correct target values

$$w_{ij} = w_{ij} - \eta(y_j - t_j)x_i. \tag{5.6}$$

Here y_j is the output, t_j is the target and x_i the input. η is called the learning rate. It controls how fast the model is learning. The algorithm of the Perceptron consists of first initializing the weights and then train by computing (5.5) and adjusting weights through (5.6). This is done until the error is acceptably small.

5.1.3 Artificial Neural Networks (ANN)

Now lets look into some of the more advanced machine learning methods. There are several ways of constructing a neural networks depending on the problem that is to be solved. And we should mention that ANN in general can be used for other purposes than supervised learning, typically we can divide neural networks into four categories [39];

- general networks for supervised learning
- networks designed for image processing, most common one being Convolutional Neural Networks (CNN)
- networks for sequential data such as Recurrent Neural Networks (RNN)
- networks for unsupervised learning such as Deep Boltzmann Machine (DBM)

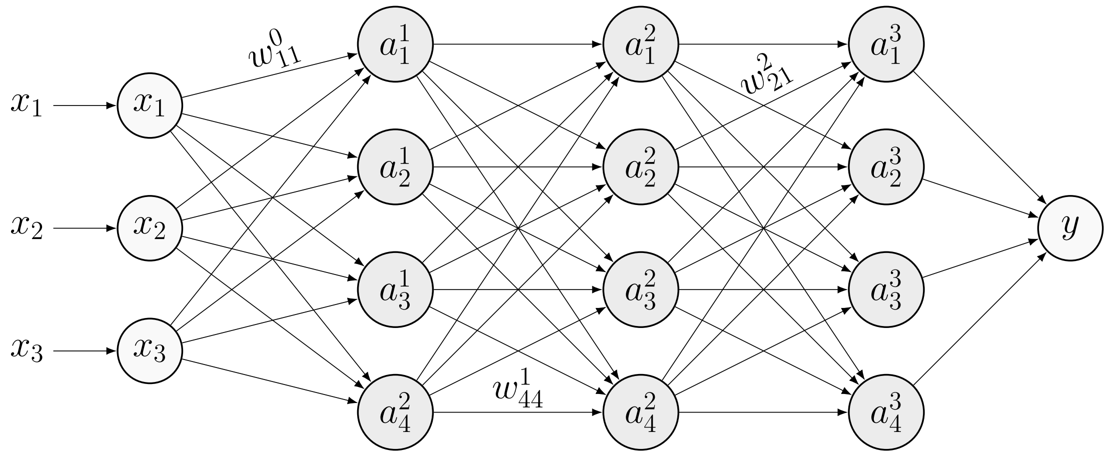

Figure 5.3: The Feedforward Neural Network consists of several layers where information goes only one way, forward in the network.

The architecture of the different networks look very different and for simplicity we will disregard networks like CNN and RNN as they have little resemblance to the main topic of this thesis, which is the restricted Boltzmann Machine. And we shall just describe the basics of ANN by use of the simplest model, the feedforward neural network (FNN).

In general a neural network consist of an input layer, one or more hidden layers and an output layer. The number of nodes in the input layer is usually dependent on the observable values. The number of nodes in the hidden layer however we are to choose our self. There is no connection between the input layer and the output layer. They are separated by the hidden layers and what connects each layer are weights that can be adjusted, which is the training process. Just like we saw for the Perceptron.

A "plain vanilla" network is called a Multi Layer Perceptron (MLP) and is, obvious from its name, based on the Perceptron and contains more than the one layer of neurons. A full FNN is simply the MLP with backpropagation, a concept we will explain shortly.

5.1.4 Feedforward Neural Network

The FNN, which indicated by its name, only propagates information forward in the network. [2] There is no information going back in the network, as we shall see is the case of the Boltzmann Machine.

The algorithm of a FNN with several layers is a complicated one as one has to deal with more than one weight matrix, and with the addition of layers one has to focus in order to keep track of all indexes. But lets begin. The input to each node i in the input layer is a component from a vector $\mathbf{x} = (x_1, x_2, ..., x_d)$. Just as for the Perceptron, the inputs are weighted and summed

[2]Compared to other networks like RNN e.g., that can handle time dependent data and the weights may keep a memory of previous states.

$$z = \sum_{i=1}^{m} w_{ij} x_i + b. \tag{5.7}$$

Here w_{ij} is the matrix containing all the weights connecting the nodes from the input layer and the neurons in the first hidden layer, b is called the bias. Just like before we need an activation function to tell us if the weighting of the inputs make the receiving neuron fire or not. In this case we want the activation function to push the value of z into a number between 0 and 1, so instead of the step function from before we can use the sigmoid function, or another function. The bias serves as a value in which the input must reach in order for the neuron to become active. We see that if the weighted sum is smaller than the bias the sigmoid function will return a number close to zero, indicating a low activity in the neuron. But in the reversed case it returns a number close to 1, meaning the neuron is active.

But we are not just dealing with the one layer, we have several ones. And the general expression of the input-output function is given as

$$a_j^l = \sigma\left(\sum_k w_{jk}^{l-1} a_k^{l-1} + b_j^l\right) = \sigma(z_j^l).$$

a_j^l indicates the activation of the jth neuron in the lth layer, which is given by the connecting weights between layer l and $l-1$, the activation of the previous layer a^{l-1} and the bias. It is meaningful to define the weighted sum

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l.$$

The way to train a neural network is by something called backpropagation. This is an algorithm that computes the gradients of the weights with respect to a cost function. This makes it possible to use gradient descent to update the weights to minimize loss. For categorical data it is common to use the cross-entropy for the cost function. For continuous data the mean squared error is usually applied

$$\mathcal{C}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - a^L)^2. \tag{5.8}$$

Either way $\mathcal{C}$ depends on the activation of the neurons in the output layer L, meaning it depends on the weights and biases as well. The backpropagation begins with deriving the gradient of the cost function for the a layer l w.r.t the activation function from the previous layer $l-1$. We define the error of the jth neuron in a layer l as

$$\Delta_j^l = \frac{\partial \mathcal{C}}{\partial z_j^l} = \frac{\partial \mathcal{C}}{\partial a_j^l}\sigma'(z_j^l).$$

Then we propagate backward through the network computing the error of the cost function, w.r.t the activation function of the subsequent layer $l+1$

$$\Delta_j^l = \frac{\partial \mathcal{C}}{\partial z_j^l} = \sum_k \frac{\partial \mathcal{C}}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_j^l} = \left(\sum_k \Delta_j^{l+1} w_{kj}^{l+1} \right) \sigma'(z_j^l).$$

When we have calculated all the errors we need the derivatives of the cost function w.r.t the weights and biases, so we know how to adjust these variables

$$\Delta_j^l = \frac{\partial \mathcal{C}}{\partial z_j^l} = \frac{\partial \mathcal{C}}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} = \frac{\partial \mathcal{C}}{\partial b_j^l},$$

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^l} = \frac{\partial \mathcal{C}}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1}.$$

This adjustment is done through the method of gradient descent as we discussed in the chapter of VMC. Or it may be done by the more improved method of Stochastic Gradient Descent (SGD).

FNN may in its simplest form be looked upon as a nonlinear transformation of the input into the output, dependent on the weights and biases. If we compare with the simple linear regression model, the updating or adjustment of weights according to the FNN cost function would be the same as minimizing the RSS.

At last in this section we mention the universal approximation theorem which states that a feed-forward network with a single layer containing a finite number of neurons can approximate continuous function of n real variables.

5.2 Unsupervised learning

We have looked at some methods for solving problems where we assume to have labeled data that can be used to train with. When it comes to unsupervised learning this is no longer the case, and therefore these are methods for solving other types of problems.

Since there is no known correct output value to check up against, unsupervised learning is applied to find similarities in data. Typically they are used to find patterns, group data together (clustering) or for reduction of dimension. Big data pools can be decreased in size, which can be very useful with the amount of data in our world today. Each day the amount of data produced goes as 10^{18} [40], so these methods are important for reduction.

This is not a subject we will explore further in this thesis. Instead we move on to the main topic of this chapter, reinforcement learning.

5.3 Reinforcement learning

Reinforcement learning can be said to lie somewhere in between supervised and unsupervised learning. We do not have the same knowledge of the output as one does with supervised learning, what we do have though is some knowledge of the *environment* the problem we try to solve is located in.

This form of machine learning is described by having an *agent* which acts in the environment. By the use of trail-and-error the agent makes a possible action, e.g. moving from one *state* to another, and based on some *reward function* derived from the environment it makes the move or not.

This trail-and-error way of solving a problem is also called "learning with a critic". The algorithm is only let know when it is doing wrong, but not how to fix it. So it must explore the state space and is rewarded when it is moving towards the correct state. This is a method used in game theory and navigation algorithms for robots.

Often a reinforcement learning process may be described in a way such that the requirements of Markov process are met. If that is the case then we may use sampling techniques like the ones we described in 4.2.2 and 4.2.3 and train by lowering the Markov chain cost.

5.3.1 Energy based models

Problem solving in physics often involve the calculation of the energy of a system. Thermodynamics tells us that a system will try to reach lower energy states, an equilibrium. This is something we take advantage of in VMC. And it is also something that can be applied in machine learning. Therefore energy based models are closely related to the methods in statistical mechanics, like Monte Carlo. They deal with an associated energy function of a system and bases the cost function on this function.

Typically these systems have a connecting probability distribution that contains information of the behavior. The important difference between discriminative models and energy based models is that where discriminative models need data to train on, energy based models may draw entirely new examples from a probability distribution. They do not use backpropagation to adjust the weights, but instead samples from the probability distribution. Methods that are able to do so are called generative models.

In the aspects of reinforcement learning, a Hamiltonian may describe the environment and the probability distribution the state. The energy function would then serve as a reward function to be minimized.

5.3.2 Generative learning

In this thesis we will be looking closer at the Restricted Boltzmann Machine, which is a energy-based generative model. Meaning it can take a probability distribution and sample from it. This is very convenient since problems in quantum mechanics try to solve a probability distribution, in effect, the wave function. This however is easier said then done considering probability distributions often contain a difficult partition function, or in case of a wave function, a high-dimensional complex integral. So when dealing with complicated probability distributions it is often much easier to learn from the relative weights of different states or data points than absolute probabilities [39]. Take the Boltzmann distribution in the grand-canonical ensemble,

$$p(x_1) = \frac{1}{Z} e^{-\beta(E(x_1) - \mu N(x_1))}.$$

If we were to calculate the probability of the system being in state x_1 we would require to find the partition function, which depends on all states $\mathbf{x}$. The relative probability of two configurations, x_1 and x_2, is proportional to the difference between their Boltzmann weights

$$\frac{p(x_1)}{p(x_2)} = e^{-\beta(E(x_1) - E(x_2))}.$$

So we see that this is a much easier calculation than the one above, since the partition function cancels out. This is very useful as we will see when we get to the Boltzmann Machine where the partition function contains a three part energy term that sums over all visible and hidden nodes. The number of possible configurations becomes a very large number that we would like to avoid having to handle.

The algorithm concerning the rested Boltzmann Machine is bases upon Bayesian statistics. We therefore mention some aspects of this.

Bayes' theorem states that

$$P(h|x) = \frac{P(x|h)P(h)}{P(x)},$$

where $P(h|x)$ is the conditional probability of a hypothesis h being true given some data x [41]. $P(x)$ and $P(h)$ are the probabilities of the data and the hypothesis, the latter being something we do not have. We the use the marginal probability

$$P(x) = \sum_h P(x,h),$$

where $P(x,h)$ is the joint probability distribution

The conditional probability is

$$P(h|x) = \frac{P(x,h)}{P(x)}.$$

5.4 Restricted Boltzmann Machine

Now to the main part of machine learning theory, the method we have used in our calculations. The Restricted Boltzmann Machine (RBM). This is a generative neural network that bases it self on an energy function. It is an unsupervised machine learning method where the training is done by an algorithm called Contrastive Divergence (CD), an algorithm created by Hinton and colleagues [38].

The RBM is an extension of the the Boltzmann Machine, a stochastic Hopfield network with hidden units. A visualization of the Hopfield network can be seen in Figure 5.4. Here

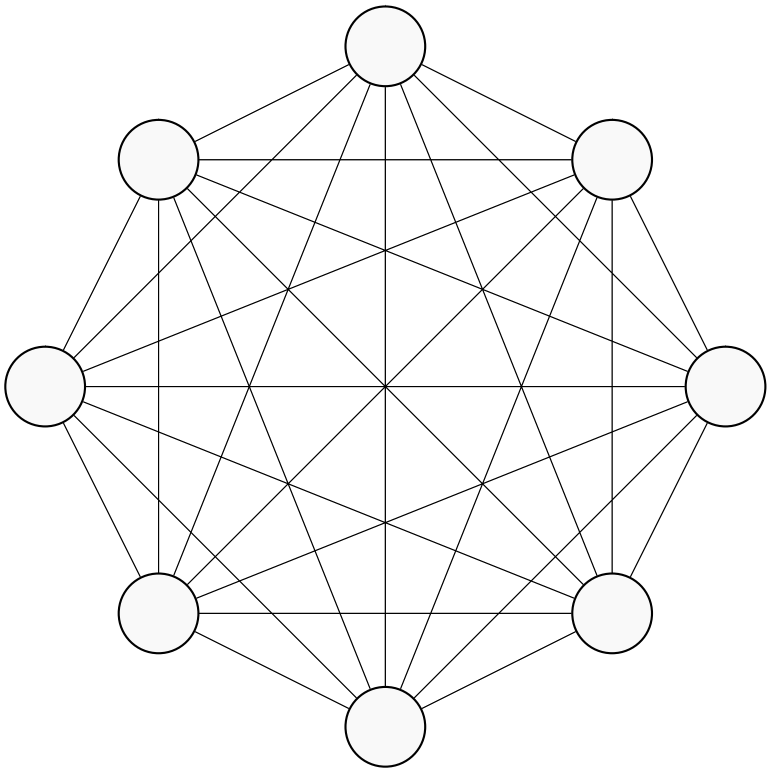

Figure 5.4: An illustration of the Hopfield network, a fully connected network of McCulloch and Pitts neurons. Each neuron is connected to all the other neurons with symmetric weights, and all neurons are visible.

all neurons $\mathbf{v}$ are connected to each other and all neurons are visible. The network does much of the same job as for the MLP, transformation of data by use of a binary threshold. But with a continuous Hopfield network we can add a continuous activation function and sample from a probability distribution

$$P(\mathbf{v}) = \frac{1}{Z} \exp\left(-E(\mathbf{v})\right). \tag{5.9}$$

This distribution givens the probability of the system being in a state $E(\mathbf{v})$. So instead of minimizing the energy, as one does for most other networks, we approximate the probability distribution that matches the energy function [38]

$$E(\mathbf{v}) = -\sum_i b_i v_i - \frac{1}{2} \sum_i \sum_j w_{ij} v_i v_j. \tag{5.10}$$

This network is very suitable for solving the Ising model. The reason for this is that the energy function that comes form this network takes the same shape as the energy of a system of (± 1) spins on a lattice. From Figure 5.4 we see that by multiplying each visible node with each other and their connecting weights we get the second term in (5.10). The factor $\frac{1}{2}$ comes from the fact that each connection get summed over twice. The first term is simply each node multiplied with its bias.

With the Boltzmann Machine however we have both a visible and a hidden layer. The visual layer takes information from the environment and the hidden neurons serves the same purpose as before. The units are given stochastic features, introducing randomness to the system. We see that (5.9) is the Boltzmann distribution we talked about in previous chapters, hence the name of this network. The shape of the network is the same as for the Hopfield network, but now some of the nodes are visible and the rest serves as hidden nodes.

The way to train the Boltzmann machine is by maximizing the log likelihood of the weights representing the correct output. Still this is a network difficult to train, as one must sample two distributions, the likelihood of the hidden neurons given the data and weights, and the likelihood of the visible nodes given the hidden neurons and the weights. Also since all neurons are connected to one another, there will be correlation between the two distributions.

Enters the RBM. In this network there are no connections between neurons within the same layer, see Figure 5.5. This makes it possible to manipulate conditionally independent probability distributions to sample from. This is the basis of the CD. This was a break trough within the field of neural networks causing the interest of ANN to grow.

One thing we did not talk about in the section on FNN was the computational power of the hidden layers. The reason to use hidden layers is to add complexity to a network and increase its ability to describe problems to a greater extent. Each hidden node or variable may extract interaction between the visible units of higher order [39].

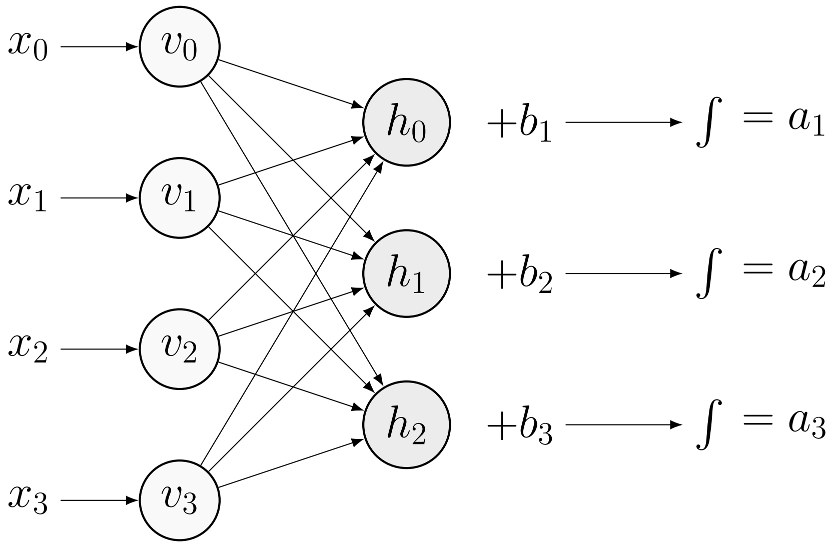
x_0
x_1
x_2
x_3
v_0
v_1
v_2
v_3
h_0
h_1
h_2
$+b_1 \longrightarrow \int = a_1$
$+b_2 \longrightarrow \int = a_2$
$+b_3 \longrightarrow \int = a_3$

Figure 5.5: The RBM consist of visible layer and a hidden layer. It looks very similar to a regular FNN with just one layer. However the network has stochastic units. Just like before we have not visualized the weights, but the are what connects the visible and hidden nodes.

5.4.1 The Network

The RBM network looks quite similar to the FNN, and it is the same if the FNN has one layer and uses a logistic activation function. This is also assuming the hidden nodes of the RBM carry binary values (we will look closer at this soon). The RBM is a bipartite graph, which means that the nodes in the same layer are not connected to each other and can be dealt with separately. In other words, they are independent of one another. The nodes are only connected to every node in another layer. An illustration is given in Figure 5.5. Input is given to the first layer keeping the visible nodes, v_i. Weights, kept in a two dimensional matrix w_{ij}, connect the first layer to the next one, keeping the hidden nodes, h_j. Just like every other network we have looked at before the hidden nodes have associated biases. However, now the visible nodes have biases as well.

5.4.2 The energy function

The energy function for the RBM is similar to the one for the Boltzmann Machine, except from the fact that we now have separated the visible and hidden layer. A new term enters and we have a joint energy function for both visible and hidden nodes

$$E(\mathbf{v},\mathbf{h}) = -\sum_i a_i(v_i) - \sum_j b_j(h_j) - \sum_{ij} w_{ij} v_i h_j. \tag{5.11}$$

Here $a_i(v_i)$ and $b_j(h_j)$ are bias functions of the visible and hidden nodes respectively. Since we no longer multiply each connection twice the factor $\frac{1}{2}$ vanishes. And the reason all terms are negative is that we expect the energy of the network to decrease.

The bias functions can take different forms depending on the units of the variables in the layers. In (5.11) both layers are assumed Bernoulli, i.e. both have binary units. This is called a binary-binary RBM. In case of a Gaussian-binary type the bias functions take the following form

$$a_i(v_i) = \frac{v_i^2}{2\sigma_i^2}, \quad b_j(h_j) = b_j h_j.$$

Here the biases of the visible nodes are given in **a** and the visible units take a continuous form $v_i \in \mathbb{R}$. This way the visible variables can represent moving particles. The biases of the hidden nodes are given in **b** and the hidden units take a binary form $h_j \in [0, 1]$. This causes the energy function changing into

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{ij} \frac{w_{ij} v_i h_j}{\sigma_i^2}. \tag{5.12}$$

Given our new energy function the new probability distribution becomes a joint distribution function, depending on both visible and hidden variables and the partition function Z equal (5.14)

$$P_{rbm}(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}, \tag{5.13}$$

$$Z = \int\int \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})} d\mathbf{v} d\mathbf{h}. \tag{5.14}$$

The joint probability distribution with the full expression of the energy function is

$$P_{rbm}(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp\left(-\sum_i^M \frac{(v_i - a_i)^2}{2\sigma_i^2}\right) \prod_j^N \exp\left(h_j b_j + \sum_i^M \frac{w_{ij} v_i h_j}{\sigma_i^2}\right). \tag{5.15}$$

What we are searching for is the probability distribution over the visible units, given the hidden units. To do so we find the marginal distribution of the visible units.

Marginal distribution

From (5.13) we can marginalize over all the hidden units and get the distribution over the visible units

$$\begin{aligned} P_{rbm}(\mathbf{v}) &= \sum_{\mathbf{h}} P_{rbm}(\mathbf{v}, \mathbf{h}), \\ &= \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}. \end{aligned}$$

This is essentially the distribution we want to sample

$$P_{rbm}(\mathbf{v}) = \frac{1}{Z} \exp\left(-\sum_i^M \frac{(v_i - a_i)^2}{2\sigma_i^2}\right) \prod_j^N \left(1 + \exp\left(b_j + \sum_i^M \frac{v_i w_{ij}}{\sigma^2}\right)\right). \tag{5.16}$$

Conditional distribution

When we come to the sampling part if the RBM we are going to need the conditional probability density functions of both the hidden and visible units. These can be calculated from (5.15) by use of Bayes' theorem. We have gathered these from [42]. The conditional probability of a binary hidden unit h_j takes the form of a sigmoid function

$$p_{GB}(h_j = 1|\mathbf{x}) = \frac{1}{1 + e^{-b_j - (\frac{x}{\sigma^2})^T \mathbf{w}_{*j}}}, \tag{5.17}$$

and tells the probability of it being an activated neuron. The conditional probability of continuous $\mathbf{v}$ has another form

$$p_{GB}(x_i = 1|\mathbf{h}) = \mathcal{N}(v_i|a_i + \mathbf{w}_{i*}^T \mathbf{h}, \sigma_i^2). \tag{5.18}$$

The name Gaussian-binary now makes sense, the last conditional probability takes shape as a normal distribution with mean a_i and variance σ_i^2, and it given the probability of the neuron being on.

5.5 Gibbs Sampling

Gibbs sampling is a sampling technique quite similar to Metropolis. The proposal probability we talked about in 4.2.2 is now selected to be the conditional probabilities from the section above. This causes the acceptance probability to equal 1. Thus we are accepting all suggested states. This is different from Metropolis and importance sampling.

The fact that the nodes in each layer are independent of one another makes it possible to calculate (5.17) and (5.18).

5.5.1 Algorithm

Like mentioned, since the units in one layer are independent of each other we may sample the conditional probabilities like

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}),$$

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}).$$

The way the algorithm goes is that we iteratively sample from the conditional distributions $\mathbf{h}_{t+1} \sim p(\mathbf{h}|\mathbf{v}_t)$ and $\mathbf{v}_{t+1} \sim p(\mathbf{v}|\mathbf{h}_{t+1})$. When we calculate the distribution of the hidden node we clamp the nodes in the visible layer, i.e. keep the values stationary. Then we use (5.17) to get the new values of each h_j. When this is done we do the same with the hidden nodes, clamp their values, and calculate the new v_i's by (5.18).

The distributions are guarantied to converge for $t \to \infty$, but drawing that many samples is computationally expensive. Instead we may draw samples just for a few iterations and then use an optimization method to estimate the outcomes. This is basically the Contrastive Divergence method suggested by Hinton.

The general Gibbs sampling algorithm take the following structure.

1. Initialize $\mathbf{v}$
2. Sampling $\mathbf{h}$ through $P(\mathbf{h}|\mathbf{v})$ with acceptance $= 1$
3. Sampling $\mathbf{v}$ through $P(\mathbf{v}|\mathbf{h})$ with acceptance $= 1$
4. Iterate step 2 and 3

5.6 Cost function

In machine learning in general we have seen that what governs the training process is the cost function. This makes sure that the adjusting of weights minimizes the difference between the predictions and the actual correct answer. With reinforcement learning the case is different since we do not know the answer. Instead our cost function is the energy of the system we seek. Just as we did in VMC, seeking the minimum in local energy, we do the same with the RBM.

Our minimization of choice is the gradient descent and the stochastic gradient descent. How we will make use of these methods and how to find the term we need, i.e. the gradient of the energy ∇E_L, we will explain later.

Part II

Implementation and results

Chapter 6
Implementation: Structure

In VMC there are a lot of methods to implement and many of them rely on each other. Writing a VMC code can become very disorganized. Therefore structuring is important. Also, solving many-particle quantum systems numerical involve calculations on a vast number of variables. Even for bosons, with just three degrees of freedom, keeping track of movements can become cumbersome . Therefore how we implement thing become important.

We have decided to use Python as programming language. The reason for this being it is a very dynamical language and easy to write, compared to languages such as C++ where one must deal with pointers and typesetting. These is something Python deals with automatically [43].

6.1 Object Oriented Programming

Object oriented programming (OOP) has its history back to the 50s and 60s.

OOP makes it possible to collect things that belong together in one place, in a class. Python has from the beginning been an object oriented language. Therefore using classes in Python is extremely simple. But what is an object? And what is a class?

Classes

A class is a piece of code with associated variables and methods. It is callable and when called upon an object is created, comprised of the class elements. A class may be called several times, each time creating a unique object of the class, an instance.

There are many features of classes, one class may inherent from a base class, but as this is not something that has been taken advantage of in this thesis, we will not dwell upon the subject.

```python
class MeaningOfLife:
    """A simple class called MeaningOfLife"""

    def __init__(self):
    """This is an instance of class"""

        self.a = 42

    def simple_answer(self):
    """This is a method in class"""

        return self.a
```

An example of a class is given above, which is callable by the statement x = MeaningOfLife(). This creates a new instance of the class and puts it to the local variable x, where x now is a module object.

Instances

An instance is, as mentioned before, a unique, individual object. In the example above an __init__ method was implemented. This is a special method in classes that is automatically called first when a new instance of a class is created. Typically this creates an initial state. So each time the class is called upon, a new initial state is set.

Methods

Once we have created a new object we may access the methods within the class. This is done by dot notation, x.simple_answer(), where x is the module object and the function simple_answer() is an attribute of it, will access simple_answer() and do what the function says. Which is return the value 42. There are two types of attributes to a class, data attributes and method attributes. The one described above is of the last type. But we could get the value 42 by accessing the data attribute. This is simply done by x.a, without a parenthesis since it is not a function, but a variable.

6.2 Structure of code

In Figure 6.1 an illustration of how the VMC program is build up and how the different classes are connected to one another is shown. The arrows indicate information flow. So the Metropolis class uses methods from all of the other classes except for Optimizer, which is only used in the VMC-file (which technically is not a class, but a regular Python file). And indirectly gets information from Hamiltonian class by the Sampler class.

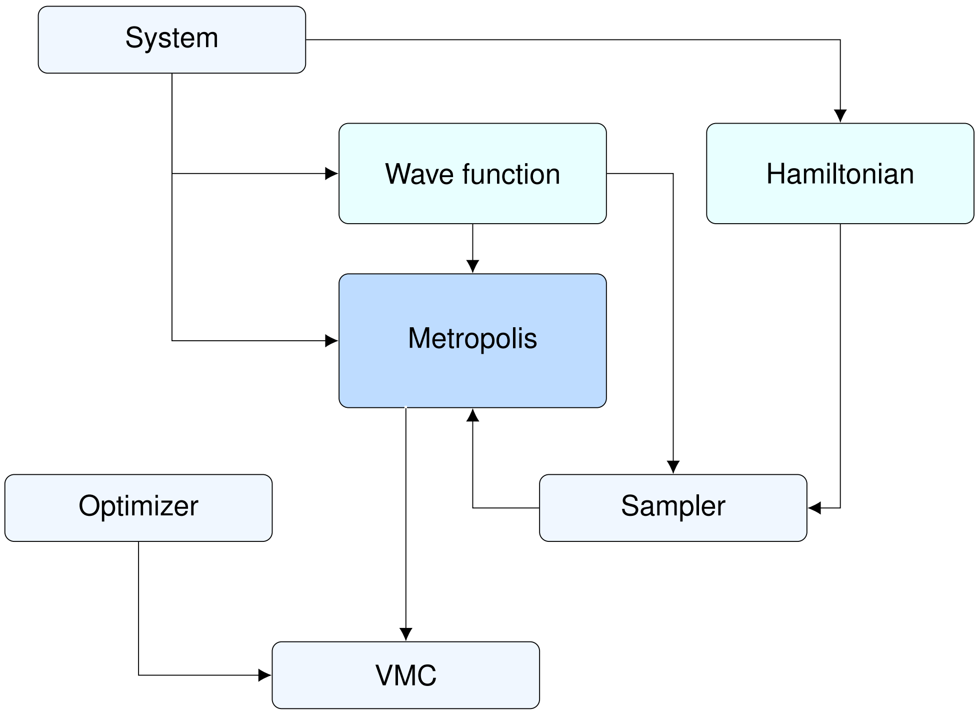

Figure 6.1: Each box represents a class, Metropolis being the largest one both in number of methods and number of lines. Next is Wavefunction and Hamiltonian. The arrows show which classes are used/send information to which classes.

The information that is being sent around is a position matrix keeping track of all particles position in the configuration space. The initialization of positions happens inside the Metropolis class, this is machinery of the entire VMC. Then Metropolis uses methods from the other classes to solve the VMC.

6.3 System class

We have created a class called System for the one purpose of keeping all values of the distance between the particles. The reason for this is that calculating the distance between each particle requires to loop over all particles, twice. The expression of the distance between particle k and particle j is given as follows

$$r_{kj} = \sqrt{|x_k - x_j|^2 + |y_k - y_j|^2 + |z_k - z_j|^2}. \tag{6.1}$$

Obviously this is somewhat expensive and something we wish to avoid. Especially since this is an expression that appears a lot of places in the calculations. In the expression of Jastrow factor of the wave function, the gradient and laplacian of the wave function, the Lennard Jones potential and derivative of the McMillian wave function wrt. the variational parameter. Needless to say, we do not want to calculate (6.1) over and over again. It is

$$\begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} \\ r_{2,1} & r_{2,2} & r_{2,3} & r_{2,4} \\ r_{3,1} & r_{3,2} & r_{3,3} & r_{3,4} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix}$$

Figure 6.2: The distance matrix contains the value of the distance between each particle, where the entrance $r_{2,3}$ is the distance between particle number 2 and 3. Since this matrix is symmetric we are only interested in the upper diagonal.

$$\begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} \\ r_{2,1} & r_{2,2} & r_{2,3} & r_{2,4} \\ r_{3,1} & r_{3,2} & r_{3,3} & r_{3,4} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix}$$

Figure 6.3: When updating the distance matrix it is not sufficient to update one row or one column. We must update both in order for all values to be correct. Even though we are only using the upper diagonal.

much more convenient to store the values and simply apply them when needed.

Keeping in mind that $r_{2,3}$ necessarily must be equal $r_{3,2}$ we are only in need if the upper diagonal of the matrix, Figure 6.2. The diagonal will be zero, since this is the distance between one particle and it self.

Another aspect of this is, our sampling methods only move one particle at a time, meaning there is no need to calculate the entire matrix each time one particle is moved.

Instead it makes more sense just updating the matrix by calculating the new distances between the moved particle and the rest. This reduces our previous two loops to one. However we must update the symmetric entries as well in order for every entry in the upper diagonal to be correct. Lets say we move particle number 2. This means that only entries with a 2 in them in the upper diagonal must be updated. If we look at Figure 6.3 we see that by updating to second row (green row), only $r_{2,3}$ and $r_{2,4}$ are correct. But if we symmetries the row and column (purple column), $r_{1,2}$ is now also updated.

```
def positions_distances(self, positions):
    """Calculate the distances between particles"""

    for i in range(self.num_p):
        for j in range(i, self.num_p-1):
            r = 0.0
            for k in range(self.num_d):
                ri_minus_rj = (positions[i, k] -
                               positions[j+1, k])
                r += ri_minus_rj**2
            self.distances[i, j+1] = math.sqrt(r)
```

Inside the metropolis method we first initialize the distance matrix and we have to run over all particles twice. When we begin sampling however we can simply call the distance_update method and update the necessary entries.

```
def distances_update(self, positions, i):
    """Update the distance matrix for movement of one particle"""
    """particle i = particle index"""

    for j in range(self.num_p):
        if j != i:
            r = 0.0
            for k in range(self.num_d):
                ri_minus_rj = positions[i, k] - positions[j, k]
                r += ri_minus_rj**2
            self.distances[i, j] = math.sqrt(r)
            self.distances[j, i] = self.distances[i, j]
```

Looking at the jastrow function we get an example of use of the distance matrix from the system class. It is important to remember that the elements we want are the i'th and $j+1$'th. This comes from the definition of the Jastrow factor.

```python
def jastrow_factor(self, positions):
    """Calculate correlation factor."""
    f = 1.0

    for i in range(self.num_p):
        for j in range(i, self.num_p-1):

            distance = self.s.distances[i, j+1]
            if distance > self.a:
                f = f*(1.0 - (self.a/distance))
            else:
                f *= 0.0
    return f
```

6.4 Periodic Boundary Conditions

Lastly in this chapter we mention some theory on periodic boundary conditions as we will need it later on. This is often done to eliminate surface effects on account of the fact that one wishes to work on systems of infinite size. This however is computationally expensive, or frankly impossible. So one simulate such a system by adding many similar boxes next to each other. The periodic boundary conditions causes the particles to stay inside an original box with dimension L, with the center of the cube being at $[L/2, L/2, L/2]$. When on particle moves outside the box it reappears on the other side of the box. This way we can imagine an infinite large system by use of one simple box. The particles that move outside the system reappears on the other side.

```
def periodic_boundary_conditions(self, positions, index):
    """Apply periodic boundary conditions"""
    """for the case of strong interaction between particles"""

    moved_particle = positions[index, :]
    x = moved_particle[0]
    y = moved_particle[1]
    z = moved_particle[2]
    L = 5.0

    if(x > L):
        moved_particle[0] = x - L
    if(y > L):
        moved_particle[1] = y - L
    if(z > L):
        moved_particle[2] = z - L

    if(x <= 0.0):
        moved_particle[0] = x + L
    if(y <= 0.0):
        moved_particle[1] = y + L
    if(z <= 0.0):
        moved_particle[2] = z + L

    return moved_particle
```

We simply check each coordinates if they are greater than L or lesser than zero, and move them to the other side of the box if they are. But if we are to use periodic boundary conditions we must take to account the distances between the particles. This is a quantity we deal with a lot, as we mentioned above. Two particles far away from one another inside the box, their distance is not trivial. So we will explain shortly the concept of minimum image convention.

6.4.1 Minimum image convention

The simple approach of simulating a large system by use of periodic boundary conditions cause us to think twice about the quantity r_{kj}. If two particles are on either side of the box then the distance between them is no longer given by (6.1). Each particle has an image of it self in every other copied box. One particle positioned at (x, y, x) has a relative position at $(x+L, y, x)$ for instance. What we do in minimum image is finding the minimum distance between two particles. If the distance between two particles k and j is greater than half the size of the box then there exists a shorter distance, which is found by subtracting the length of the box with the position coordinates, $|x_k - x_j| = L - |x_k - x_j|$. This way we use the distance between the image of the particles.

Chapter 7
Implementation: VMC

We will now give an outline of the Variational Monte-Carlo method. The entire framework consists of 4 different classes inside the src file, Metropolis, Sampler, System, Optimizer, as well as two folders containing classes for the Hamiltonians and different Wave functions. The Metropolis class contains code doing most of the important work, whereas the Sampler, System and Optimizer classes only controls independent variables, like average values and distance matrices.

Inside the Wave function and Hamiltonian classes we find important properties of the systems, like the gradients and laplacian of the wave function, local energy function etc. We will look closer at these classes, but first lets get a overview of the fundamental structure of the VMC. As explained in 4.1 we can rewrite the expectation value of the Hamiltonian of our system so that what are dealing with is a minimization problem that can be solved by MCI.

We repeat the expectation value of the local energy

$$E_V \approx \langle \hat{E}_L \rangle = \int P(\mathbf{r}) \hat{E}_L d\mathbf{r} \approx \frac{1}{N} \sum_{i=1}^{N} \hat{E}_L(x_i),$$

since our main goal is estimating this property in order to check the validity of the given PDF, or the trail wave function we are testing. The way to approach this is by sampling the PDF $P(\mathbf{r})$ by use of a sampling technique, like Metropolis, N times, where N being the number of Monte Carlo cycles. This means that without knowing the exact wave function of our system we can, by selecting a good enough trail wave function, we can possibly get close to the exact one. This strongly depends on us selecting a trail wave function that can possibly describe the exact one. Remembering that the wave function must contain a variational parameter that can be changed as we search for the ground state. This variational parameter can be optimized by gradient descent or some other optimization algorithm.

This complete VMC method is implemented in the file called vmc.py. It imports all classes and contains function that separates different cases of systems.

```python
from metropolis import Metropolis
from optimizer import Optimizer
from Hamiltonian.non_interaction import Non_Interaction
from Hamiltonian.weak_interaction import Weak_Interaction
from Hamiltonian.lennard_jones import Lennard_Jones
from Wavefunction.mcmillian import McMillian_Wavefunction
from Wavefunction.wavefunction import Wavefunction
from sampler import Sampler
from system import System

"""
Variational Monte Carlo with Metropolis Hastings algorithm for
selection of configurations. Optimizing using Gradient descent.
"""

step_metropolis = 1.0
step_importance = 0.01
learning_rate = 0.01
gradient_iterations = 1000

opt = Optimizer(learning_rate)
```

This is an example from the vmc.py fil, where all classes are imported and the needed parameters are given, such as the number of steps for each method, learning rate etc. These are parameters we do not wish to change, but they are instead optimal values we use through out the thesis. Since many of the methods depend on one another it is probably easiest to look at Figure 6.1 to get a sence of how the implementation is done. But below we have given an example of how the gradient method runs with a call for almost all the other classes.

```python
def non_interaction_case(monte_carlo_cycles, num_particles,
   num_dimensions,
                         alpha):
    """Run the variational monte carlo for the non-interactive
    case."""
```

Within the same file, we have created functions or methods for different cases, such as the non-interacting gas, the weak-interacting gas, gas with deformation of trap potential etc. Here the size of the system may be selected, and also how many Monte-Carlo simulations we wish to do. And also the start value of the variational parameter α. In each case the optimizing method is some variant of gradient descent. An example of this is given below.

```python
sys = System(num_particles, num_dimensions)
for i in range(gradient_iterations):

    if abs(d_El) > 1e-20:

        # Call wavefunction class in order to set new alpha
          parameter
        wave = Wavefunction(num_particles, num_dimensions,
                            parameter, beta, a, sys)
        # Run with analytical expression of local energy = true
        hamilton = Non_Interaction(omega, wave, sys, 'true')
        met = Metropolis(monte_carlo_cycles, step_metropolis,
                         step_importance, num_particles,
                         num_dimensions, wave, hamilton, sys)

        d_El, energy, var = met.run_metropolis()

        new_parameter = opt.gradient_descent(parameter, d_El)
```

For each gradient iteration we must update the variational parameter α, which both the Wavefunction class and the Hamiltonian class is dependent on, so these must be updated. The Metropolis class indirectly depends on α through Wavefunction and Hamiltonian class, so this must be updated as well. We also see that the method run_metropolis() gives an output, or several. Estimated values related to the local energy, (4.5). These are the important values we mentioned in the section on gradient descent, 4.3.1. Other cases, such as blocking and one-body density are also calculated through the vmc.py. We will come back to these later.

7.1 Hamiltonian

The Hamiltonian class is really three different classes for each of the three different cases; non-interacting Bose-gas, the weak-interacting Bose-gas and the strong-interacting liquid helium-4. The reason for this being that these systems all have different Hamiltonian's with all different derivative. So a systematic way is just to separate them.

```python
class Weak_Interaction:
    """Calculate variables regarding the Hamiltonian of given
       wavefunction."""

    def __init__(self, omega, wavefunction, system, analytical):
        """Instance of class."""
        self.omega = omega
        self.w = wavefunction
        self.s = system
        self.analytical = analytical
```

The class Weak_Interaction has both Wavefunction and System as inputs, since the Hamiltonian depends on the derivative of the wave function. Methods inside this class includes the laplacian, which may be calculated both by use of an analytical expression and by numerical differentiation. Also the local energy function is found here. Lets look at the Hamiltonian for the different cases.

7.1.1 Weak interacting gas

In Chapter 3 we explained how the weak interacting system may be described by the Gross-Pitaevskii equation (3.8) and the wave function for such a system is the solution to this equation. We are studying a system of BEC confined in a harmonic oscillator trap and the Hamiltonian may be written [44]

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + V_{ext}(\mathbf{r}) \right) + \sum_{i<j} V_{int}(\mathbf{r}_i, \mathbf{r}_j), \tag{7.1}$$

with V_{ext}, the external trap potential and V_{int}, the internal potential given by

$$V_{ext}(\mathbf{r}) = V_{trap}(\mathbf{r}) = \frac{1}{2} m(\omega_{ho}^2 x^2 + \omega_{ho}^2 y^2 + \omega_{ho}^2 z^2), \tag{7.2}$$

$$V_{int}(|\mathbf{r}_i - \mathbf{r}_j|) = \begin{cases} \infty & |\mathbf{r}_i - \mathbf{r}_j| \leq a \\ 0 & |\mathbf{r}_i - \mathbf{r}_j| > a. \end{cases} \tag{7.3}$$

The first sum in (7.1) is the standard harmonic oscillator part and the last is the interacting term describing the inter-particle interaction by use of a pairwise hard-core potential. N represent the number of particles, ω_{ho} is the oscillator frequency of the trap and r_i is the position of particle i, whereas r_{ij} is the distance between the particles and given as $r_{ij} = |\mathbf{r_i} - \mathbf{r_j}|$. a is the strength of the interaction between the particles. If we use $\omega_\perp = \omega_{ho}$ as the trap frequency in the perpendicular xy-plane and ω_z the frequency in the z-direction, we may take into account the anisotropy of the trap potential and shape the trap in other than the spherical form.

$$V_{ext}(\mathbf{r}) = \begin{cases} \frac{1}{2}m\omega_\perp^2 r^2 & (S) \\ \frac{1}{2}m[\omega_\perp^2(x^2+y^2)+\omega_z^2 z^2] & (E), \end{cases} \tag{7.4}$$

In the case of $\omega_\perp = \omega_z$, the shape of the trap is spherical indicated by the S in (7.4). E indicates an elliptical trap potential. If we use $\lambda = \omega_z/\omega_\perp$ we can rewrite the external potential for the elliptical trap to the following

$$\frac{1}{2}m[\omega_\perp^2(x^2+y^2)+\omega_z^2 z^2] = \frac{1}{2}m\omega_\perp^2(x^2+y^2+\lambda^2 z^2),$$

which is the potential for the spherical trap, but with a factor λ^2 in the z-direction. This is something we will take advantage of later.

An example of a method within the non_interaction class is given below. This show laplacian_analytical(), which solves the ∇^2 of the wave function. We give the laplacian of non-interactive case as the expression of its derivative is much shorter than that of the interactive case which is given by (A.22). The reason for using this monstrous expression is that it is faster to use the analytical solution than using numerical differentiation to solve the derivatives.

```
def laplacian_analytical(self, positions):
    """Analytic solution to laplacian for non-interacting case
    and symmetric potential"""
    """Assumes beta = 1.0 and scattering length = a = 0.0"""

    c = 0.0
    d = self.w.num_d
    n = self.w.num_p
    for i in range(self.w.num_p):
        x = positions[i, 0]
        if d == 1:
            c += x**2
        elif d == 2:
            y = positions[i, 1]
            c += x**2 + y**2
        else:
            y = positions[i, 1]
            z = positions[i, 2]
            c += x**2 + y**2 + z**2

    laplacian_analytic = -2*d*n*self.w.alpha + 4*(self.w.alpha**2)*
        c

    return laplacian_analytic
```

Units

We will use natural units, $\hbar = m = 1.0$. The dimension of the trap is given in Å, where $a_{ho} = (\hbar/m\omega_{ho})^{1/2} \approx 10^4$Å defines the length of the trap. And the energy is given in $\hbar\omega_\perp$.

7.1.2 Strong interacting gas

When we look at the strong interacting gas, the liquid helium, the Hamiltonian consists of the kinetic part and the Lennard-Jones potential. Since the potential is given in units of Kelvin, we must convert the factor in kinetic term so the units match.

Units

With the parameters for the Lennard-Jones potential given as the following [45]

$$\epsilon/k_B = 2.556K \text{ and } \sigma = 10.22\text{Å},$$

we must rewrite the factor $\frac{\hbar}{2m}$ in the kinetic term so the energy also is given in Kelvin. We simply divide it by the Boltzmann factor k_B which has units JK^{-1} and get

$$\hat{H} = \sum_{i=1}^{N} -\frac{\hbar^2}{2mk_B}\nabla_i^2 + \sum_{i<j} 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right].$$

The mass m of one helium-4 atom is $\approx 6.6423 \times 10^{-27}$ kg, and the factor in front of the laplacian is ≈ 6.06, with units in Å^2K. As mentioned before, with strong correlation between particles we must apply periodic boundary conditions in order. This is first of all because we wish to simulate an infinite system, meaning the number of particles $N \to \infty$. But since this is not possible to do we will have to resemble such a system. The size of the simulated box is given by the volume $\Omega = L^3$ with L being the length of the sides of the box and the density within the box is $\rho = N/\Omega$. We will keep the density constant, $\rho_0 = 0.02185$ atoms/Å^3 which is the density of liquid helium-4 at zero pressure [46].

```
def laplacian_analytical(self, positions):
    """Analytical solution to the laplacian"""

    sum1 = 0.0
    sum2 = np.zeros(self.w.num_d)
    for k in range(self.w.num_p):
        rk = positions[k, :]
        for j in range(self.w.num_p):
            rj = positions[j, :]
            if(j != k):
                r_kj = rk - rj
                rkj = self.s.distances[k, j]

        rkj1 = 1.0/(self.w.alpha*rkj)
        rkj2 = self.alpha2/rkj**2
        rkj6 = rkj2*rkj2*rkj2

        sum1 += rkj6*rkj1
        sum2 += r_kj

    sum_squared = (sum1*sum1)*np.dot(sum2, sum2)
    laplacian = self.fraction**sum_squared - 10.0*sum1

    return laplacian
```

Example shows the laplacian inside hamiltonian class for the strong interacting case, where the calculations for the laplacian comes from (7.10).

7.2 Representing the wave function

When choosing a trial wave function there are several thing that must be considered. First of all the trial wave function's characteristic description must be close to the exact wave function that we are trying to approximate. It has to have the same general shape in order to even be able to approximate the real one. Also it must be a continuous function, it must be differentiable. Otherwise it is not of much use to us.

7.2.1 Weak interacting gas

Our starting point for selecting a trial wave function for the weak interacting gas confined within a harmonic oscillator trap is the Gross-Pitaevskii equation 3.8. We begin with the single-particle wave function, or the mean-field wave function

$$\psi_{MF}(\alpha,\beta,\mathbf{r}_i) = \exp\{[-\alpha(x_i^2+y_i^2+\beta z_i^2)]\}, \tag{7.5}$$

where the variational parameter α is given as a factor in the exponential and the factor determining the formation of the trap, β, is incorporated in the wave function. We can see this if we put $\lambda^2 = \beta$ in (7.4). We will express the interchanging forces between particles by a two-body correlation function. The simplest correlation function comes in Jastrow form

$$F(1,\cdots,N) = \prod_{i<j} f(a,|\mathbf{r}_i-\mathbf{r}_j|).$$

We will use the two-body correlation function (3.9) that was the solution of the Schrödinger equation of two atoms interaction via a hard-core potential with diameter a, at low energy.

$$f(a,|\mathbf{r}_i-\mathbf{r}_j|) = \begin{cases} 0 & |\mathbf{r}_i-\mathbf{r}_j| \le a \\ (1-\frac{a}{|\mathbf{r}_i-\mathbf{r}_j|}) & |\mathbf{r}_i-\mathbf{r}_j| > a. \end{cases} \tag{7.6}$$

This will cause the wave function to flatten out when it is activated, when two particles come closer together than the scattering length a. The smaller the value of a, the slower the system is. So this parameter determines the activity of the gas.

The point of choosing this particular wave function is that its derivatives cancels out the effects of the external and internal potentials. The external potential goes as r^2 and the internal as $1/r^2$. The laplacian of the wave function (i.e. the kinetic energy) cancel out these potentials, which is what we want. We are looking for the ground state energy, where kinetic energy and potential energy is at an equilibrium.

The complete trial wave function now reads

$$\Psi_T(\mathbf{r}) = \Psi_T(\mathbf{r}_1,\mathbf{r}_2,\dots\mathbf{r}_N,\alpha,\beta) = \prod_i \psi_{MF}(\alpha,\beta,\mathbf{r}_i)\prod_{i<j} f(a,|\mathbf{r}_i-\mathbf{r}_j|). \tag{7.7}$$

This wave function has been used in several papers on the system we are dealing with [24] [47] [48]. One of the main advantages of using this wave function is that the short-range repulsion between the particles are incorporated into the wave function. So the internal potential (7.3) may be disregarded as these effects are taking into account in Ψ_T. The wave function guarantees that the particles will not overlap.

Now that we have the trial wave function there are some important properties that we will need in order to use the different methods. The calculation of the gradient and the laplacian for this wave function is quite cumbersome, and the entire derivation is given in A.1.1. For the gradient descent we will need the derivative of $\Psi_T(\mathbf{r})$ with respect to α. This is a simple derivation so we give it here. The term $\frac{1}{\Psi_T}\frac{d\Psi_T}{d\alpha}$ is fairly easy given the wave function we are considering

$$\begin{aligned}\frac{1}{\Psi_T}\frac{d\Psi_T}{d\alpha} &= \frac{-\Psi_T\sum_i^N(x_i^2+y_i^2+\beta z_i^2)}{\Psi_T},\\ &= -\sum_i^N(x_i^2+y_i^2+\beta z_i^2).\end{aligned} \tag{7.8}$$

Since the correlation function $f(a, r_{ij})$ of the wave function is independent of the variational parameter α, this expression holds for the interactive case as well.

```
def alpha_gradient_wavefunction(self, positions):
    """Calculate derivative of wave function divided by wave
       function."""
    """This expression holds for the case of the trail wave
       function
    described by the single particle wave function as a the
       harmonic
    oscillator function and the correlation function
    """
    deri_psi = 0.0

    for i in range(self.num_p):
        x = positions[i, 0]
        if self.num_d == 1:
            deri_psi += x*x
        elif self.num_d == 2:
            y = positions[i, 1]
            deri_psi += (x*x + y*y)
        else:
            y = positions[i, 1]
            z = positions[i, 2]
            deri_psi += (x*x + y*y + self.beta*z*z)

    return -deri_psi
```

The calculation of (7.8) is done inside the Wavefunction class and used in the Sampler class to calculate (7.13). The Wavefunction class contains many methods such as the ratio needed for Metropolis sampling, the gradient of Ψ_T needed for importance sampling and of course the wave function itself.

7.2.2 Strong interacting gas

In case of strong interaction our trial wave function is the McMillian wave function (3.12). Again we need the gradient and laplacian of the wave function in order to calculate properties such as the local energy and quantum force. We repeat the McMillian wave function

$$\psi_T = \exp\left(-\frac{1}{2}\sum_{i<j}\left(\frac{\beta}{r_{ij}}\right)^5\right).$$

Lets call the inside of the sum

$$u(r_{ij}) = \left(\frac{\beta}{r_{ij}}\right)^5,$$

to make the calculations easier to both read and do.

The Gradient

We get the gradient through

$$\nabla_k\psi_T = \exp\left(-\frac{1}{2}\sum_{i<j}\left(\frac{\beta}{r_{ij}}\right)^5\right)\sum_{k<j}-\frac{1}{2}\nabla_k u(r_{kj}).$$

Where we have used the (A.15). The gradient of $u(r_{kj})$ with respect to the k'th particle is

$$\begin{aligned}\nabla_k u(r_{kj}) &= u'(r_{kj})\frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}},\\ &= -\frac{5\beta^5(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}^7},\end{aligned}$$

where we have used (A.19), and $u'(r_{kj}) = -5\beta^5/r_{kj}^6$. Now we can write the total gradient

$$\nabla_k\psi_T = \frac{5}{2}\sum_{k<j}\frac{\beta^5(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}^7}\exp\left(-\frac{1}{2}\sum_{i<j}\left(\frac{\beta}{r_{kj}}\right)^5\right).$$

But the expression we will use is the gradient divided by the wave function, and it is

$$\frac{1}{\psi_T}\nabla_k\psi_T = \frac{5}{2}\sum_{k<j}\frac{\beta^5(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}^7}. \tag{7.9}$$

The Laplacian

Now we find the expression for the laplacian of the trail wave function, and we begin with

$$\nabla_k^2\psi_T = \nabla_k\left(\exp\left(-\frac{1}{2}\sum_{i<j}u(r_{ij})\right)\right)\sum_{k<j}-\frac{1}{2}\nabla_k u(r_{kj})$$
$$+\exp\left(-\frac{1}{2}\sum_{i<j}u(r_{ij})\right)\sum_{k<j}-\frac{1}{2}\nabla_k^2 u(r_{kj}).$$

The laplacian of $u(r_{kj})$ we get from (A.21) and we get the following

$$\begin{aligned}\nabla_k^2 u(r_{kj}) &= u''(r_{kj}) + u'(r_{kj})\frac{2}{r_{kj}},\\ &= \frac{30\beta^5}{r_{kj}^7} - \frac{10\beta^5}{r_{kj}^6}\frac{1}{r_{kj}},\\ &= \frac{20\beta^5}{r_{kj}^7}.\end{aligned}$$

And the total expression for the laplacian of the wave function divided by the wave function becomes

$$\frac{1}{\psi_T}\nabla_k^2\psi_T = \frac{25}{4}\left(\sum_{k<j}\frac{\beta^5(\mathbf{r}_k-\mathbf{r}_j)}{r_{kj}^7}\right)^2 - 10\sum_{k<j}\left(\frac{\beta^5}{r_{kj}^7}\right). \tag{7.10}$$

We immediately see the difficulty with the vector subtraction, $\mathbf{r}_k - \mathbf{r}_j$, inside the sum of the first expression. This term in itself is no problem, it is the fact that the entire sum is squared that causes difficulties.

We will have to know the derivative of the wave function w.r.t the variational parameter as well, so we list that as well

$$\frac{1}{\psi_T}\frac{\partial\psi_T}{\partial\beta} = -\frac{5}{2}\beta^4\sum_{i<j}\left(\frac{1}{r_{ij}}\right)^5. \tag{7.11}$$

The properties connected to the McMillian wave function are kept in a separate class called McMillan_Wavefunction but keep the same functions as the Wavefunction class.

7.3 Sampling

Now we come to the class doing the main work of the VMC, the Metropolis class. This is a class with many methods and collecting a lot of information from other classes. From Figure 6.1 we see it gets input from both Wavefunction, System, Sampler and Hamiltonian (indirectly through Sampler). The Sampler class does all the sampling of expectation values such as $\langle E_L \rangle$ and $\langle \nabla_\alpha \Psi_T \rangle$.

```
def sample_values(self, positions):
    """Sample important values"""
    """From Hamiltonian and Wavefunction class"""

    self.local_energy = self.h.local_energy(positions)
    self.alpha_gradient_wf = self.w.alpha_gradient_wavefunction(
        positions)
    self.accumulate_energy += self.local_energy
    self.accumulate_energy_sq += self.local_energy*self.
        local_energy
    self.accumulate_psi_term += self.w.alpha_gradient_wavefunction(
        positions)
    self.accumulate_both += self.local_energy*self.
        alpha_gradient_wf
```

Sampling with Metropolis is not to be confused with the Sampler class, which samples expectation values and averages. The methods within Metropolis selects new suggested states of a system by using the algorithm we explained in 4.2.2 and 4.2.3.

And for each Monte Carlo cycle we propose a new configuration $\mathbf{r}_p$ for the system

$$\mathbf{r}_p = \mathbf{r} + r * \Delta\mathbf{r}.$$

with r being a random number drawn from the distribution [-1,1] and $\Delta\mathbf{r}$ is the step length of the Metropolis method. The choice behind the distribution [-1,1] is that the particles may move in every direction. Our choice of the proposal probability in (4.7) is the wave function,

$$\frac{\rho(\mathbf{r}_j)T(\mathbf{r}_i|\mathbf{r}_j)}{\rho(\mathbf{r}_i)T(\mathbf{r}_j|\mathbf{r}_i)} = \frac{|\Psi_T(\mathbf{r}_j')|^2}{|\Psi_T(\mathbf{r}_i)|^2}, \tag{7.12}$$

which in our case is positive definite since our system consists of bosons. And therefore it may be considered as a PDF. Below we have given a short piece of the code from metropolis_step() is given. Inside this method new configurations are suggested, and accepted by use of the acceptance ratio calculated in Wavefunction class. Epsilon is some small random number.

```
if acceptance_ratio > epsilon:
    positions = new_positions
    self.s.distances_update(positions, random_index)
    self.c += 1.0

else:
    pass
```

In importance sampling the equation 7.12 from Metropolis algorithm changes to

$$\frac{\rho(\mathbf{r}_j)T(\mathbf{r}_i|\mathbf{r}_j)}{\rho(\mathbf{r}_i)T(\mathbf{r}_j|\mathbf{r}_i)} = \frac{G(\mathbf{r}_i, \mathbf{r}_j, \Delta t)|\Psi_T(\mathbf{r}'_j)|^2}{G(\mathbf{r}_j, \mathbf{r}_i, \Delta t)|\Psi_T(\mathbf{r})|^2}.$$

One main reason we may use importance sampling instead of Metropolis is that our wave function is not a uniform distribution. So a Gaussian distribution is better, which is what this sampling method does.

```
def run_metropolis(self):
    """Run the naive metropolis algorithm."""

    # Initialize the posistions for each new Monte Carlo run
    positions = np.random.rand(self.num_p, self.num_d)
    # Initialize the distance matrix
    self.s.positions_distances(positions)
    # Initialize sampler method for each new Monte Carlo run
    self.sam.initialize()

    for i in range(self.mc_cycles):
        new_positions = self.metropolis_step(positions)
        positions = new_positions
        self.sam.sample_values(positions)

    self.sam.average_values(self.mc_cycles)
    energy = self.sam.local_energy
    d_El = self.sam.derivative_energy
    var = self.sam.variance
    self.print_averages()
    return d_El, energy, var
```

The way the metropolis sampling is done, same goes for importance, is that a method called run_metropolis calls the metropolis_step() which does the selection of moves and returns the new position. Here the distance-matrix is also updated. We see in the run-method that the call on the step-method is done for each Monte Carlo step. Positions are updated here as well and then the values are sampled within the Sampler. The output of the metropolis is the estimated values of local energy, its variance and also the derivative of E_L needed for optimization.

Other methods within the Metropolis class is the use of boundary conditions in the case of the strong interacting liquid helium-4, we have methods for blocking and for calculating the one-body density.

```python
def one_body_density(self, positions):
    """Run one-body density count."""

    num_radii = 41
    density = np.zeros(num_radii)
    r_vec = np.linspace(0, 4, num_radii)
    step = r_vec[1] - r_vec[0]

    # Calculate the distance from origo of each particle
    radii = np.zeros(self.num_p)
    for i in range(self.num_p):
        r = 0
        for j in range(self.num_d):
            r += positions[i, j]*positions[i, j]
        radii[i] = math.sqrt(r)

    # Check in which segment each particle is in
    for i in range(self.num_p):
        dr = 0.0
        for j in range(num_radii):
            if(dr <= radii[i] < dr+step):
                density[j] += 1
                break
            else:
                dr += step

    return density
```

Here is an example of how to find one-body density. This method also has a connected run-method that looks similar to the run_metropolis. We have decided the segmentation of the sphere to be divided into 40 pieces. Then we run over all segments/bins and add the particles positioned in each of the bins.

7.4 Optimization

As energy minimization method we have used the simple gradient descent. This is sufficient in the case of the VMC when we are only dealing with one variational parameter. But, as we will see for the RBM, where the number of parameters to optimize can increase quickly, the use of stochastic gradient descent becomes vital.

First of all the gradient descent rely on us knowing the gradient of the term for the local energy with respect to the variational parameter α. The expression for the first derivative of the local energy is given by the following equation [49]

$$\frac{dE_L}{d\alpha} = 2\left(\left\langle \frac{1}{\Psi_T}\frac{d\Psi_T}{d\alpha}E_L \right\rangle - \left\langle \frac{1}{\Psi_T}\frac{d\Psi_T}{d\alpha} \right\rangle \langle E_L \rangle \right). \tag{7.13}$$

To get the energy gradient (7.13) we will need to calculate the expectation value of the local energy as well as the partial derivative of the wave function, and their product. The partial derivatives we calculated above, for both the single particle wave function as well as the McMillian.

Given the drawbacks of the simple gradient descent method we have decided to use the two-point step size method by Barzilai and Borwein [50] to hopefully speed up the algorithm and avoid some of the problems related to the gradient descent. This method makes adjustment to the learning rate based on an iteration scheme according to

$$\gamma_k = \frac{\langle \Delta\alpha, \Delta\alpha \rangle}{\langle \Delta\alpha, \Delta g \rangle}$$

where

$$\Delta\alpha = \alpha_k - \alpha_{k-1} \text{ and } \Delta g = g_k - g_{k-1},$$

and $g_k = \nabla E_L(\alpha_k)$. The upgraded learning rate is then calculated through

$$\gamma_k = \frac{\alpha_k - \alpha_{k-1}}{\big\langle \nabla E_L(\alpha_k) \big\rangle - \big\langle \nabla E_L(\alpha_{k-1}) \big\rangle}. \tag{7.14}$$

We begin by making a guess on the parameter θ, then we follow Eq. (4.8). Copy from VMC chapter (The last step of the algorithm is to check if the absolute value of the gradient squared is smaller or larger than some given small number ϵ. If larger then we continue the process, if smaller we consider our result good enough.)

```python
def gradient_descent_barzilai_borwein(self, alpha,
   derivative_energy, k):
    """gradient descent with Barzilai-Borwein update on learning
       rate"""

    if k == 0:
        new_alpha = alpha - self.learning_rate*derivative_energy
        self.old_alpha = alpha
        self.old_gradient = derivative_energy

    else:

        diff_alpha = alpha - self.old_alpha
        diff_derivative_energy = derivative_energy - self.
           old_gradient
        new_gamma = diff_alpha/diff_derivative_energy

        new_alpha = alpha - new_gamma*derivative_energy

    self.old_alpha = alpha
    self.old_gradient = derivative_energy

    return new_alpha
```

In the gradient descent with applied Barzilai-Borwein calculation of the learning rate the algorithm goes much like the general case, but we must check if our calculation is the first iteration, in that case we use the start value of the learning rate. Otherwise the (7.14) is used.

7.5 Testing of code

We use the non-interacting system as a test case, to check if methods used for sampling and optimization work as they should. In Table 7.1 and 7.2 we have compared the Metropolis algorithm with the improved importance sampling technique.

N	α	$\langle E_L \rangle$	$\langle E_L \rangle_{exact}$	$\sigma^2_{E_L}$	acceptance rate	CPU time
1	0.5	1.5	1.5	$4.17 \cdot 10^{-14}$	0.56	223.2
2	0.5	3.0	3.0	$1.78 \cdot 10^{-15}$	0.56	338.0
5	0.5	7.5	7.5	$1.52 \cdot 10^{-10}$	0.56	706.5
10	0.5	15.0	15.0	$6.74 \cdot 10^{-12}$	0.56	1127.7
50	0.5	75.0	75.0	$5.36 \cdot 10^{-11}$	0.56	12454.4

Table 7.1: Run with Metropolis algorithm, for the non-interacting case with 10^4 Monte Carlo runs. We use the method of gradient descent with 100 iterations. Start value of α is 0.1 and learning rate is set to 0.01. The step length $\Delta r = 1.0$ was selected.

N	α	$\langle E_L \rangle$	$\langle E_L \rangle_{exact}$	$\sigma^2_{E_L}$	acceptance rate	CPU time
1	0.5	1.5	1.5	$6.22 \cdot 10^{-15}$	0.99	537.8
2	0.5	3.0	3.0	$9.24 \cdot 10^{-14}$	0.99	982.8
5	0.5	7.5	7.5	$1.11 \cdot 10^{-12}$	0.99	2906.8
10	0.5	15.0	15.0	$2.06 \cdot 10^{-11}$	0.99	9565.6
50	50.65	-2.4	75.0	$1.95 \cdot 10^{-11}$	0.0	185418.6

Table 7.2: Run with importance sampling algorithm, for the non-interacting case with 10^4 Monte Carlo runs. Again we use the method of gradient descent of 100 iterations, α is 0.1 and learning rate is 0.01. Notice how the importance sampling cannot handle 50 particles when we only use 10^4 MC runs. The step length $\Delta t = 0.01$ was selected.

From Table 7.1 and 7.2 we see that both methods end up with reasonably low variance, but the CPU-time is very different. The importance sampling takes over twice the amount of time for 1 particle compared to metropolis, and over 8 times the amount of time if we increase the system to 10 particles. Figure 7.2 give us a sense of how the two methods work for different variational paramaters. Metropolis is generally more stable, but for correct α the importance sampling give a better estimate on the energy, with zero variance. From Figure 7.1 we see how slow the Metropolis algorithm moves towards the stable configura-

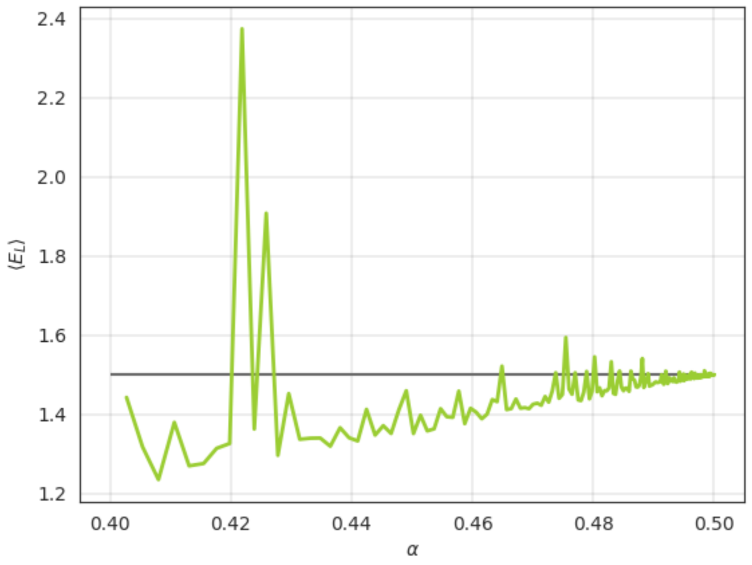

Figure 7.1: The convergence of expectation value of the energy for 1 particle i 3 dimensions. The Metropolis algorithm was chosen and the regular gradient descent as optimization method. The gray line indicated the true ground state energy at 1.5.

tion. This is not the case for the importance sampling, which is more unstable but reaches the equilibrium state much faster.

The step length for both methods has been set so the acceptance rate is according to an ideal percentage. For regular Metropolis this rate is 50%. In this case the step length $\Delta r = 1.0$ has been selected. Importance sampling requires a higher acceptance rate given that the suggestion of new states now is based upon a more thought trough algorithm and is as high as $\sim$90 %. The step length $\Delta t = 0.01$ was used.

Because of the instability of the importance algorithm the Metropolis sampling technique was used for further calculations.

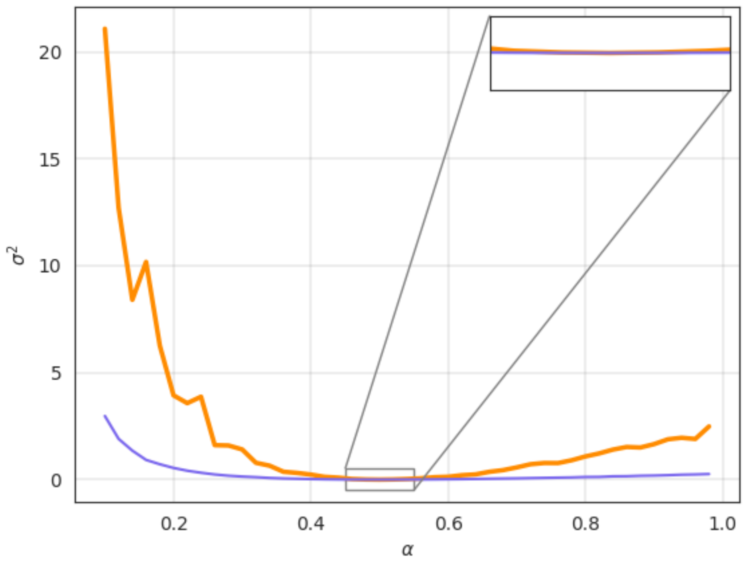

Figure 7.2: The variance of the expectation value of the local energy. Calculated with Metropolis algorithm (purple line) and importance sampling (yellow line) using 10^4 Monte Carlo cycles for 1 particle in 3 dimensions. We have used brute force to variate the α parameter.

In Figure 7.3 we see the behavior of standard gradient descent with two different learning values and the Barzilai-Borwein method. The Barzilai-Borwein method is sensitive to start values, and may deviate in some cases. If the method is able to locate the global minimum it converges mush faster than regular gradient descent however. But is suffers from much of the same problems as with standard gradient descent. Given some initial values it may get stuck in a local minimum.

Regular gradient descent with different learning rates behave differently as well. The higher the learning rate the faster the convergence, but may also deviate for some initial states.

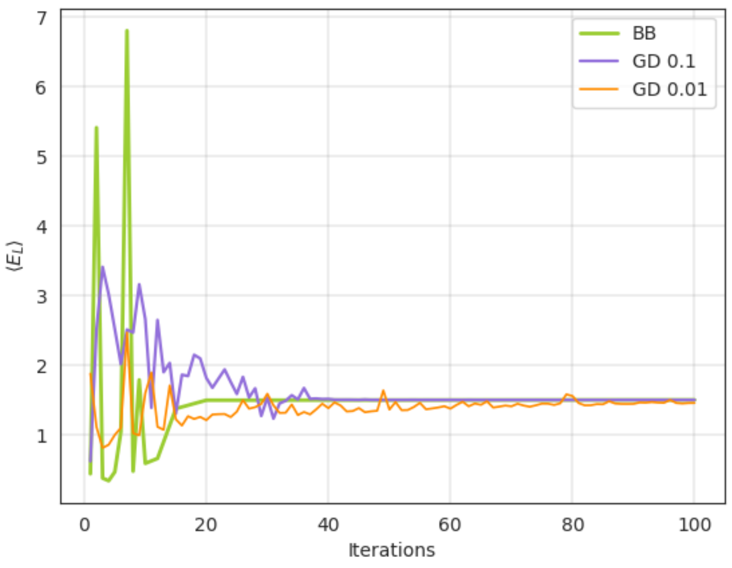

Figure 7.3: The expectation value of the local energy for different number of gradient descent iterations. We have used 10^4 Monte-Carlo runs for one particle.

Chapter 8
Implementation: RBM

In Chapter 5 we explained the theory behind the restricted Boltzmann machine and also laid the ground work of the Gaussian-binary RBM, the network and its energy function. Here we will explain further how we choose the trial wave function for the RBM and the structure of the method.

The structure of the RBM program is very similar to the VMC, seen in Figure 6.1. The reason for this is the similarities of the way of using RBM to solve many-body quantum problems compared to Monte Carlo calculations, which we explained some in Chapter 5. We still have a Hamiltonian class, a Wavefunction class, a Sampler class and an Optimizer class. Everything is run inside a program called rbm.py. What we do not have is a System class. Instead all calculations of distances between particles are done locally where they are needed.

8.1 Hamiltonian

In the case of the RBM our wave function does not contain a constant to adjust the strength of the interaction and we must implement the two-body interaction potential V_{int}

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2 \right) + \sum_{i<j} V_{int}(\mathbf{r}_i, \mathbf{r}_j), \tag{8.1}$$

in order to capture the interaction between particles. But the internal potential describing interaction between neutral particles such as bosons is given as zero for bosons far apart and infinitely strong if they get to close, (7.3). This was incorporated into the trial wave function in the VMC. This potential is not possible to use, as the optimization method then would have to search for the a minimum within energy space where there exists a point where the energy is infinitely strong. We therefore use another potential as a trail potential

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

This is the Coulomb potential describing the interaction between charged particles. It will not give us the correct ground state energy, but it may give us some insight in how well the RBM may estimate the true wave function for our system at hand.

Our expression of the local energy is still given by (4.4). In B.1 we show that

$$E_L = \sum_i^M \frac{1}{2}\left(-\left(\frac{\partial}{\partial X_i}\ln\Psi\right)^2 - \frac{\partial^2}{\partial X_i^2}\ln\Psi + \omega^2 X_i^2\right) + \sum_{k<l}\frac{1}{r_{kl}}$$

with the derivatives of Ψ are given in (B.1) and (B.2).

8.2 Representing the wave function

Like we explained in the theory of generative models what we seek is a probability distribution. And in our case more precisely the quantum mechanical wave function. In [51] the NQS (neural-network quantum state) was suggested as a representation of a trial wave function that could be trained to describe the wave function for the Ising model. The NQS

$$\Psi(X) = P_{rbm} = \frac{1}{Z}\exp\left(-\sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2}\right)\prod_j^N\left(1 + \exp\left(b_j + \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)\right), \tag{8.2}$$

comes from the energy of the restricted Boltzmann machine, (5.12). From the marginal probability of the joint energy functional, (5.16), we get our wave equation.

However this wave function has not been normalized and can no longer be interpreted as a regular probability distribution function. Instead the NQS gives us an amplitude and phase. For our purpose this is enough, but it is something to remember. In Gibbs sampling though we use the wave function given as

$$\psi(x) = \sqrt{(P_{rbm})}, \tag{8.3}$$

rather than on the form above.

8.3 Sampling

We may sampler the NQS with use of Metropolis or importance sampling. But as we explained in the theory of the RBM in Chapter 5, because of the structure of the RBM and the fact that the visible and hidden units are independent of one another, we may use a new sampling technique, the Gibbs sampling. The Gibbs sampling method is similar to the previous sampling methods and is also based on Markov chains. All methods are kept in the same class which we have decided to keep the name Metropolis for reasons being mentioned in the beginning of this chapter.

```python
def gibbs_step(self, positions):
    """Calculate new Gibbs step."""

    h_j = np.zeros(self.w.N)
    sigma = 1
    sigma2 = 1

    for j in range(self.w.N):
        sum = 0.0
        for i in range(self.w.M):
            sum += positions[i]*self.w.W[i, j]/sigma2

        h_j[j] = 1/(1 + math.exp(-self.w.b[j] - sum))

    for i in range(self.w.M):
        sum = 0.0
        for j in range(self.w.N):
            sum += self.w.W[i, j]*h_j[j]

        mu = self.w.a[i] + sum
        # acceptance with probability of 1
        positions[i] = np.random.normal(mu, sigma)

    return positions
```

The gibbs_step uses the expressions (5.17) and (5.18) to find the new state for the hidden nodes and the visible node respectively. We begin with finding the hidden nodes h_j where we keep the values of the visible nodes fixed, run through the sum over all v_i and calculate each h_j. Then we do the same with the visible nodes. Clamp the h_j's and find the new v_i's which are now the new positions r_i. This means that we accept each new suggested move with the probability of one. There is no need to calculate the ratio of a probability distribution function like we must for Metropolis and importance.

Here we see a very good reason to use hidden variables. With the wave function exposing an exponentially growing correlation problem, the hidden units are able to find these correlations by integrating out the degree of freedom of the system [39]. When clamping the units we can get information of the remaining variables, the unclamped units. This are the benefits of the RBM.

However, sampling in RBM requires calculating more vales as the wave function is now represented by several parameters $\alpha = \alpha_1, \alpha_2, \cdots$, or biases and weights. If we look at the example code below we see all the extra properties that must be updated inside the Sampler class when we work with the RBM. Since our optimization method is some form of gradient descent method we still need to calculate (7.13) which depends on all the parameters. And we must optimize all of these.

```python
def sample_values(self, positions, gibbs):
    """Get the local energy from Hamiltonian class"""
    """Sample important values"""

    if gibbs == 'true':
        self.local_energy = self.h.local_energy_gibbs(positions)
        self.accumulate_energy += self.h.local_energy_gibbs(
           positions)
    else:
        self.local_energy = self.h.local_energy(positions)
        self.accumulate_energy += self.h.local_energy(positions)
    # self.local_energy = self.h.local_energy_numerical(positions)
    # self.accumulate_energy += self.h.local_energy_numerical(
       positions)
    gradient_wf_a = np.zeros(self.w.M)
    gradient_wf_b = np.zeros(self.w.N)
    gradient_wf_W = np.zeros((self.w.M, self.w.N))

    gradient_wf_a = self.w.gradient_wavefunction_a(positions)
    gradient_wf_b = self.w.gradient_wavefunction_b(positions)
    gradient_wf_W = self.w.gradient_wavefunction_W(positions)

    self.accumulate_psi_term_a += gradient_wf_a
    self.accumulate_psi_term_b += gradient_wf_b
    self.accumulate_psi_term_W += gradient_wf_W
    self.accumulate_both_a += gradient_wf_a*self.local_energy
    self.accumulate_both_b += gradient_wf_b*self.local_energy
    self.accumulate_both_W += gradient_wf_W*self.local_energy
```

Before we start sampling expectation values we do a check of what sampling technique we are using since the NQS is not the same in the case of Gibbs sampling and Metropolis and importance sampling, as we explained in 8.2.

8.4 Optimization

As mentioned in the section above, for the RBM we must optimize more than just the one variational parameter we did in VMC. Our NQS depends on the biases a_i and b_j as well as all the weights in the matrix w_{ij}. In the case of the NQS represented by (8.2), its gradient with respect to each parameter is given as follows

$$
\begin{aligned}
\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial a_k} &= \frac{1}{2\sigma^2}(X_k - a_k), \\
\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial b_n} &= \frac{1}{2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}}{\sigma^2}\right)\right)^{-1}, \\
\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial w_{kn}} &= \frac{X_k}{2\sigma^2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}}{\sigma^2}\right)\right)^{-1}.
\end{aligned}
$$

These equations must be found in order to optimize the biases and weights through the gradient descent. The parameters grow large for large systems. As we increase the number of particles in the system the number of visible nodes grow with, since they necessary represent the particles positions. This means we must increase the number of hidden nodes in order for the NQS to be able to represent the true wave function. With the dimension of biases and the weights depending on the number of nodes in the visible and hidden layers, we immediately see that the sampling of all expectation values and the optimization of parameters demands a lot of CPU time. Hence the stochastic gradient descent becomes important for speeding up this process.

In the case of using Gibbs sampling the NQS takes the form of $\sqrt{F_{rbm}(x)}$. We must keep in mind that the derivative the NQS with respect to the parameters are different

$$
\begin{aligned}
\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial a_k} &= \frac{1}{2\sigma^2}(X_k - a_k), \\
\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial b_n} &= \frac{1}{2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}}{\sigma^2}\right)\right)^{-1}, \\
\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial w_{kn}} &= \frac{X_k}{2\sigma^2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}}{\sigma^2}\right)\right)^{-1}.
\end{aligned}
$$

And when using Gibbs sampling we must use these expressions.

8.5 Testing of code

From Table 8.1 and 8.2 we have listed the ground state energy for the non-interactive system of bosons. We have used the same number of nodes in the hidden layer as the number of particles within the system. The expectation value of the energy and the variance is calculated with blocking method.

N	γ	$\langle E_L \rangle$	$\langle E_L \rangle_{exact}$	$\sigma^2_{E_L}$	acceptance rate	CPU time
1	0.2	1.4997	1.5	$5.24 \cdot 10^{-4}$	0.51	0.004
2	0.2	3.0033	3.0	$3.26 \cdot 10^{-3}$	0.51	0.014
5	0.2	7.5047	7.5	$6.07 \cdot 10^{-3}$	0.51	0.044
10	0.02	15.56	15.0	$1.05 \cdot 10^{-1}$	0.51	0.215

Table 8.1: Run with Metropolis algorithm, $\Delta r = 1.0$, and regular gradient descent by use of different learning rates γ. The number of hidden nodes is equal the number of particles.

N	γ	$\langle E_L \rangle$	$\langle E_L \rangle_{exact}$	$\sigma^2_{E_L}$	acceptance rate	CPU time
1	0.4	1.5001	1.5	$5.45 \cdot 10^{-4}$	0.51	0.004
2	0.4	3.0025	3.0	$1.47 \cdot 10^{-3}$	0.51	0.014
5	0.04	7.549	7.5	$1.71 \cdot 10^{-2}$	0.51	0.044
10	0.04	15.020	15.0	$3.84 \cdot 10^{-2}$	0.51	0.213

Table 8.2: The same methods used in Table 8.1, but with different learning rates.

We notice that the magnitude of the learning rate does not effect the results for the smaller systems. As the size of the system increases however, a large γ causes an increase in the variance. But looking at Figure 8.1 we see that a very low γ causes the gradient method to converge much more slowly than for larger γ. The magnitude of the learning rate certainly matters a great deal.

When dealing with importance sampling we have illustrated the behavior of the importance sampling for time steps Δt equal 0.1 and 0.01. In the first case, the method shows difficulty locating the global minimum and is highly dependent on the learning rate. In the second case we experience a more stable convergence. The variance however is greater than for the Metropolis sampling.

The number of nodes in the hidden layer seems to have little influence on the RBMs ability to represent the correct wave function looking at Figure 8.3. As long as it is a

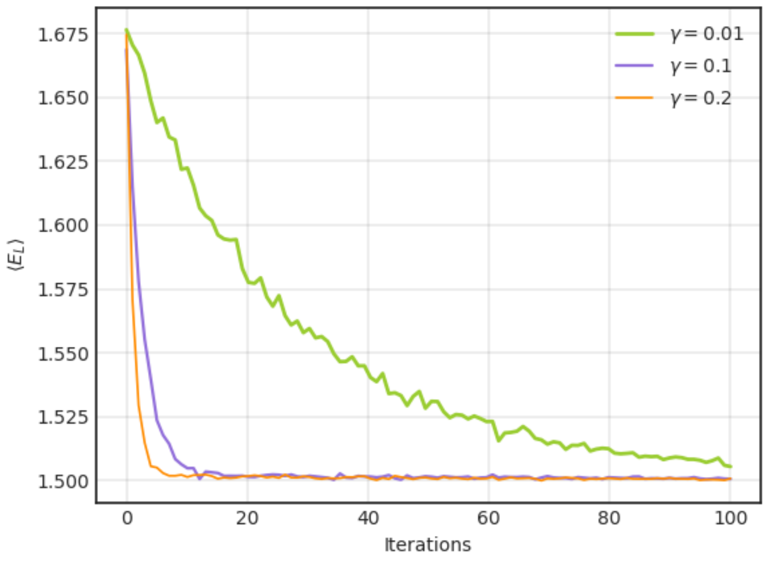

Figure 8.1: The expectation value of the local energy for different learning rates γ with Metropolis sampling algorithm. We have used 10^4 Monte-Carlo runs for one particle.

sufficiently large number this makes no difference, with a sufficiently large number seeming to be the same as the number of particles. When we add nodes this necessarily comes at a larger computational cost, simply because the number of parameters that needs solving increases. It would seem beneficial to keep the number of nodes as small as possible.

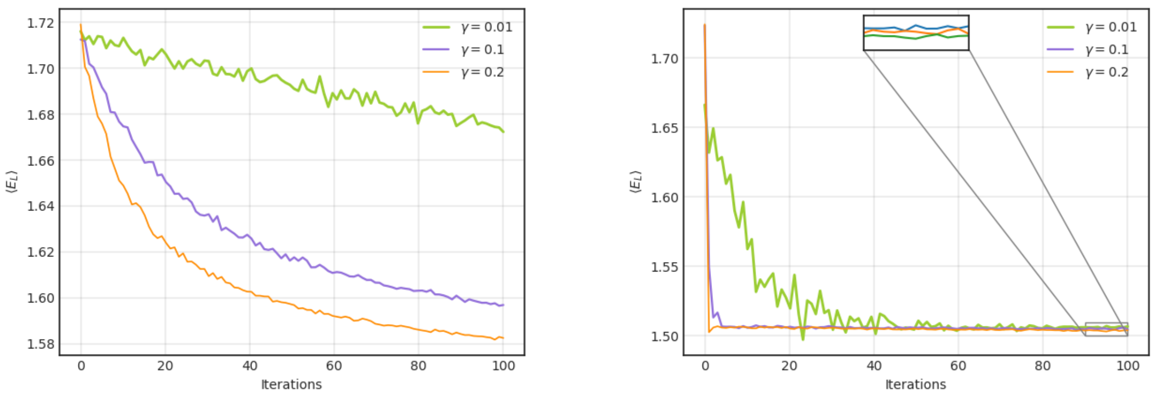

Figure 8.2: The expectation value of the local energy for different learning rate with importance sampling. We have used 2^{17} Monte-Carlo. The first figure with $\Delta t = 0.1$ and in the second $\Delta t = 0.01$.

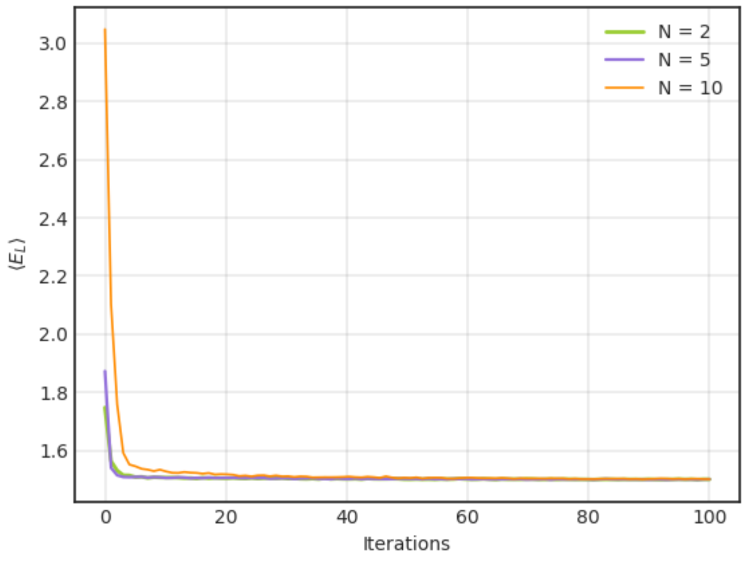

Figure 8.3: The expectation value of the local energy for different number of hidden nodes in the hidden layer.

Chapter 9
Results

In this chapter we represent some of the results that was obtained from the two different variational methods, VMC and RBM. The sampling techniques that has been used is based upon the verification of the different methods from the two previous chapters. In both cases the Metropolis algorithm seemed like the better choice as this was the most stable one. The step length was tuned so as the acceptance rate was optimal for the algorithm, 50%, and set to 1.0.

N	a	λ	α	$\langle E_L \rangle$	$\langle E_L \rangle_{ref}$	$\sigma^2_{E_L}$
2	0.00433	1	0.4997	3.003474	3.00346	$9.30 \cdot 10^{-6}$
3	0.00433	1	0.4994	4.5103	4.510036	$1.45 \cdot 10^{-5}$
5	0.00433	1	0.4988	7.5345	7.53443	$5.75 \cdot 10^{-5}$
10	0.00433	1	0.4973	15.151	15.1537	$2.37 \cdot 10^{-3}$
20	0.00433	1			30.640	
2	0.433	1	0.468	3.392	3.3831	$5.42 \cdot 10^{-4}$
3	0.433	1	0.455	5.573	5.553	$1.05 \cdot 10^{-3}$
5	0.433	1	0.443	10.61	10.577	$2.02 \cdot 10^{-3}$
10	0.433	1			26.22	
20	0.433	1			66.9	

Table 9.1: VMC results for the weak interaction case with different interaction strength a using Metropolis sampling and blocking method, reference energy from [52]. (Reference values comes from DMC calculations.)

We begin with the VMC and the weak interacting Bose-gas trapped within a harmonic oscillator potential. In Table 9.1 we have listed the ground state energy for systems with different system sizes and inter-particle interaction strength a. The optimal α parameter was obtained by use of 10^6 Monte Carlo runs and 1000 gradient descent iterations. The expectation value of the ground state and its variance was then calculated by blocking method with 2^{20} Monte Carlo runs.

First of all we immediately see that the Jastrow factor has a huge impact on the systems energy. The larger the inter-particle repulsion the larger the energy grows. With an increase in the system size the more difficulties the method has in finding optimal α as the exact wave function becomes more complex. This again causes a larger variance in the estimation of the energy, causing a large variance.

We also notice that as the particles interact more strongly with each other, larger a, the more difficulty the method has estimating the energy. This may come of the fact that our sampling method may suggest many wrong moves before the system stabilizes. The more the particles push each other around the longer the time before a stable configuration is reached.

N	a	λ	α	$\langle E_L \rangle$	$\langle E_L \rangle /N$	$\sigma^2_{E_L}$
10	0.00433	$\sqrt{8}$	0.48	21.643	2.1643	$1.73 \cdot 10^{-6}$
50	0.00433	$\sqrt{8}$	0.48	111.066	2.2213	$1.76 \cdot 10^{-6}$
100	0.00433	$\sqrt{8}$	0.48	228.682	2.2868	$1.76 \cdot 10^{-6}$

Table 9.2: Results from VMC for an elliptical shaped trap, $\lambda = \sqrt{8}$.

In Table 9.2 we have formed the trap in an elliptical shape by adjusting the λ parameter. The total energy of the system is not very interesting as we do not have any reference values to compare with in this case, but looking at the energy pr. particle we see that the energy increases as the trap in deformed. Compared to the energy in Table 9.1 the energy pr. particle is lower. When the trap is pushed into a elliptical shape the density of particles is increase in the center. The particle will be pushed closer together and the repulsion between them is activated more often. We also observe that as the system size grows so does the energy. This makes sense, as the number of particles goes up they interaction energy should go up as well. The more particles pushing each other the larger the energy. The variance is also acceptable, compared to Table 9.1.

The one-body density for a boson system consisting of 3 particles can be seen in Figure 9.1, where cases of non-interaction and weak interacting with two different correlation strength has been applied. For the non-interacting case, with $a = 0.0$, the distribution of particles is normalised round the center of the trap. As interaction is turned on the distribution stretches out as the particles now pushes each other away from the trap center. When the strength between the interacting particles increases by a factor 100 we immediately see a larger effect on the distribution. They move further and further away from one another and closer to the outer regions of the trap.

In Figure 9.2 we have applied the Coulomb potential for a system of 2 particles in 2 dimension. The reason for this being that we have a analytic solution to the wave function for two electrons in this case [53]. The ground state energy is 3.0 a.u. For low learning rate the RBM is not able to reach the correct value of the expectation value for 500 gradient descent iterations. But with a larger γ the method gives a more acceptable estimate. Table 9.3 shows the energy for the same case, and the estimate is not near exact and the variance

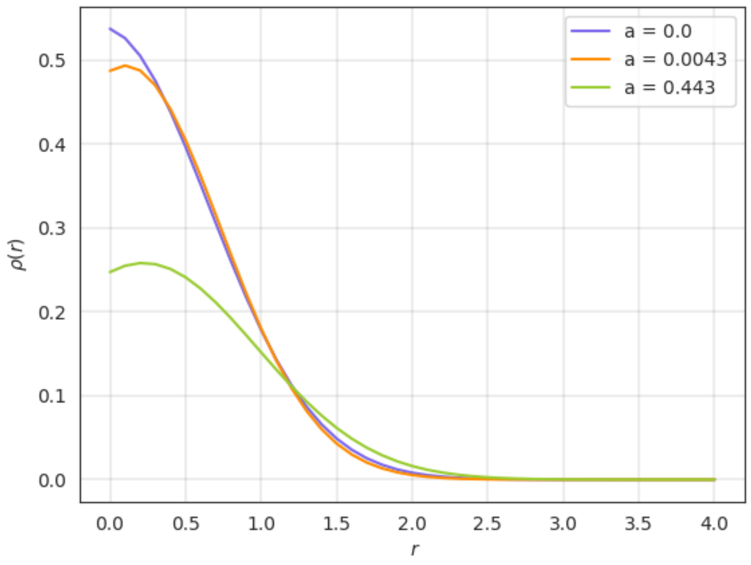

Figure 9.1: The one-body density for different cases of interaction strength. System of 3 particles in 3 dimensions.

N	λ	γ	$\langle E_L \rangle$	$\langle E_L \rangle_{ref}$	$\sigma^2_{E_L}$
2	1.0	0.3	3.104	3.0	$4.11 \cdot 10^{-2}$
2	1.0	0.4	3.100	3.0	$4.07 \cdot 10^{-2}$
2	1.0	0.5	3.106	3.0	$4.15 \cdot 10^{-2}$

Table 9.3: The ground state energy for a system of 2 particles in 2 dimensions with an interacting Coulomb force.

is higher than for the non-interacting case. Possibly running for longer than 500 gradient iterations and with a lower learning rate could decrease the variance.

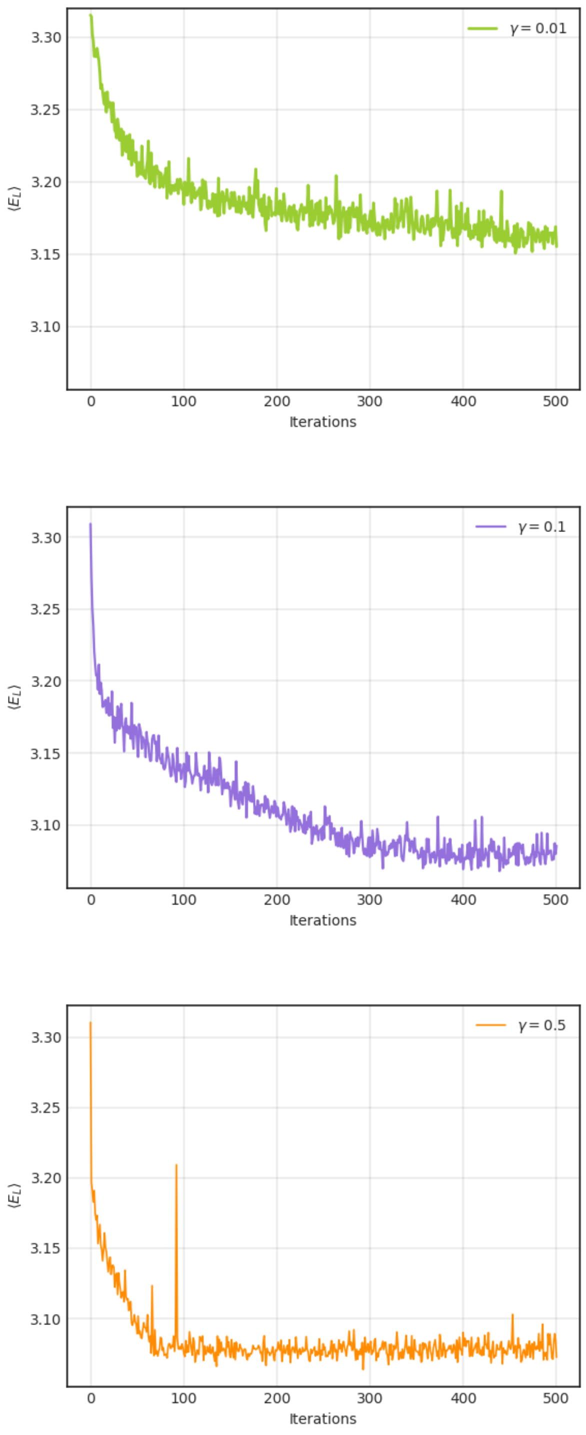

Figure 9.2: The convergence of the energy as training goes by for the interactive case. We have used 2^{17} Monte-Carlo runs.

Part III

Conclusions and future work

Chapter 10
Sum-up and Conclusion

Through out this thesis we have been studying Bose-Einstein condensates and tried to solve a system of weak interacting trapped bosons. We first used a variational Monte Carlo method to estimate the ground state energy of the system as a function of the variational parameter α. Then a RBM was applied in order to make a prediction on the exact wave function and ground state energy by use of a NQS as a trial wave function and the parameters a_i, b_j and w_{ij}.

For both methods we were able to get good results for the non-interacting case. The VMC outperformed the machine learning algorithm with close to zero variance. Both the Metropolis and the importance sampling gave good estimates. None of the sampling techniques converged any faster than the other, but importance sampling reached a lower variance.

When it came to the RBM with Metropolis sampling the ground state energy was found but with greater variance than for the VMC. Attempts were made to improve the method by testing of optimal hyper parameters of the network such as learning rate and number of hidden nodes. The RBM with importance sampling experienced some instability. Here several time steps were tested.

In the case of the VMC we where able to get good results for different interaction strength, with a one-body density that illustrated the effects on the system.

The RBM gave less good results when a repulsion force between particles was added. The methods had difficulties with reaching optimal parameters with just a simple gradient descent method as optimization scheme. With many parameters to adjust and check this was an interesting method to work with and we would have hoped to explore this network some more.

We did not test for larger systems. But the implementation was done correctly and if we had more time larger systems could have been tried out. Also other type of potential would have been interesting to check as well, like the strong interacting case of liquid Helium-4 described in 3.3.1. In this case the one-particle model falls apart and one needs a two-body potential to describe the dominating repulsive forces. If the network could pick up such strong correlations could have been very interesting to explore.

The goal of this thesis has been to explore the possibilities of using machine learning as an approach to solve many-body quantum systems. We have proven that it certainly does, but other and more updated optimization methods would have given better results, like stochastic gradient descent or the now well used Adam optimization scheme [54]. And the use of unsupervised machine learning seems like an promising field of research, as machine learning in general is a growing research area. The large variation of methods makes this an endless field of opportunities.

Future work

As mentioned, the use of machine learning on quantum mechanical systems is possible only at its beginning. In the future other type of network could be interesting to explore. Such as a convolutional neural network or the recurrent neural network. This are typically supervised learning algorithms, but perhaps in combination with traditional Monte Carlo or with modifications these networks could serve as quantum solvers.

With a growing computational power of new computers, larger and more complex systems may be explored. Also deeper networks, like the deep Boltzmann Machine or other deep neural networks could be able to see correlation in the data to a greater extent than a simple one-layer RBM.

Chapter A
Appendix A

Here we go through the analytical solution for the local energy in the case of the VMC where the trial wave function is the product of the single-particle wave function and the simple Jastrow factor.

Also an analytical description of the drift force is given.

A.1 Derivations

A.1.1 Local energy

The local energy is given by

$$E_L(\mathbf{r}) = \frac{1}{\Psi_T(\mathbf{r})}\hat{H}\Psi_T(\mathbf{r})$$

with trial wave equation

$$\Psi_T(\mathbf{r}) = \prod_i g(\alpha, \beta, \mathbf{r_i}) \prod_{i<j} f(a, |\mathbf{r_i} - \mathbf{r_j}|)$$

The Hamiltonian becomes

$$\hat{H} = \sum_i^N \left(\frac{-\hbar^2}{2m}\nabla_i^2 + V_{ext}(\mathbf{r_i}) \right) + \sum_{i>j}^N V_{int}(\mathbf{r_i}, \mathbf{r_j})$$

where

$$V_{ext}(\mathbf{r_i}) = \begin{cases} \frac{1}{2}m\omega_{ho}^2 r^2 & (S) \\ \frac{1}{2}m[\omega_{ho}^2(x^2+y^2) + \omega_z^2 z^2] & (E) \end{cases}$$

and

$$V_{int}(|\mathbf{r_i} - \mathbf{r_j}|) = \begin{cases} \infty & |\mathbf{r_i} - \mathbf{r_j}| \leq a \\ 0 & |\mathbf{r_i} - \mathbf{r_j}| > a \end{cases}$$

First we solve only the harmonic oscillator (a=0) and we use $\beta = 1$ for one particle in 1D. And for one particle the trial wave equation, Hamiltonian and local energy becomes

$$\Psi_T(\mathbf{r}) = g(\alpha, x) = e^{-\alpha x_i^2}, \tag{A.1}$$

$$\hat{H} = \left(\frac{-\hbar^2}{2m}\nabla^2 + \frac{1}{2}m\omega_{ho}^2 x^2 \right) \tag{A.2}$$

$$E_L = \frac{1}{e^{-\alpha x^2}} \left(\frac{-\hbar^2}{2m}\nabla^2(e^{-\alpha x^2}) + \frac{1}{2}m\omega_{ho}^2 x^2(e^{-\alpha x^2}) \right) \tag{A.3}$$

We then need to compute the laplacian of the trial wave-function.

$$\nabla^2(e^{-\alpha x^2}) = \frac{d^2 e^{-\alpha x^2}}{dx^2} \tag{A.4}$$

$$= \frac{d}{dx}\left(\frac{de^{-\alpha x^2}}{dx} \right) \tag{A.5}$$

$$= \frac{d}{dx}(-2\alpha x \cdot e^{-\alpha x^2}) \tag{A.6}$$

$$= (4\alpha^2 x^2 - 2\alpha)e^{-\alpha x^2} \tag{A.7}$$

Finally, the local energy becomes

$$E_L = \frac{e^{-\alpha x^2}}{e^{-\alpha x^2}} \left(\frac{-\hbar^2}{2m}(4\alpha^2 x^2 - 2\alpha) + \frac{1}{2}m\omega_{ho}^2 x^2 \right) \tag{A.8}$$

$$E_L = \frac{-\hbar^2}{2m}(4\alpha^2 x^2 - 2\alpha) + \frac{1}{2}m\omega_{ho}^2 x^2 \tag{A.9}$$

Solving the same problem for one particle in 2D and 3D we get something similar,

$$E_L = (\frac{-\hbar^2}{2m})4\alpha^2(x^2 + y^2) - 4\alpha + \frac{1}{2}m\omega_{ho}^2(x^2 + y^2), \tag{A.10}$$

$$E_L = (\frac{-\hbar^2}{2m})4\alpha^2(x^2 + y^2 + z^2) - 6\alpha + \frac{1}{2}m\omega_{ho}^2(x^2 + y^2 + z^2) \tag{A.11}$$

We can now easily see that the local energy energy for N number of particles in 1D, 2D and 3D becomes

$$\begin{aligned}
&\text{1D: } E_L = \sum_i^N (\frac{-\hbar^2}{2m})[4\alpha^2(x^2) - 2\alpha] + \frac{1}{2}m\omega_{ho}^2(x^2),\\
&\text{2D: } E_L = \sum_i^N (\frac{-\hbar^2}{2m})[4\alpha^2(x^2+y^2) - 4\alpha] + \frac{1}{2}m\omega_{ho}^2(x^2+y^2) \qquad \text{(A.12)}\\
&\text{3D: } E_L = \sum_i^N (\frac{-\hbar^2}{2m})[4\alpha^2(x^2+y^2+\beta^2z^2) - 4\alpha - 2\alpha\beta] + \frac{1}{2}m\omega_{ho}^2(x^2+y^2+\beta z^2)
\end{aligned}$$

since the energy is simply the sum of the derivatives of each particle.
Now we can try to solve the complete problem. The trial wave function is now

$$\Psi_T(\mathbf{r}) = \prod_i g(\alpha,\beta,\mathbf{r_i}) \prod_{i<j} f(a, |\mathbf{r_i} - \mathbf{r_j}|)$$

and first we rewrite it using

$$g(\alpha,\beta,\mathbf{r_i}) = \exp -\alpha(x_i^2 + y_i^2 + \beta z_i^2) = \phi(\mathbf{r_i})$$

and

$$f(r_{ij}) = \exp\left(\sum_{i<j} u(r_{ij})\right)$$

getting

$$\Psi_T(\mathbf{r}) = \prod_i \phi(r_i) \exp\left(\sum_{i<j} u(r_{ij})\right).$$

The local energy for this problem becomes:

$$E_L = \frac{1}{\Psi_T(\boldsymbol{r})}\left(\sum_i^N\left(\frac{\hbar^2}{2m}\nabla_i^2\Psi_T(\mathbf{r}) + \frac{1}{2}m\omega_{ho}^2 r^2\Psi_T(\mathbf{r})\right) + \sum_{i<j}^N V_{int}(\mathbf{r_i},\mathbf{r_j})\Psi_T(\mathbf{r})\right). \qquad \text{(A.13)}$$

The difficulty in (A.13) is solving the derivatives of the wave equation given the complexity of the exponential. We begin with the first derivative.

$$\nabla_i^2 \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right) = \nabla_i \cdot \nabla_i \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right)$$

The first derivative of particle k:

$$\nabla_k \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right) = \nabla_k \left(\prod_i \phi(\mathbf{r_i})\right) \exp\left(\sum_{i<j} u(r_{ij})\right) + \nabla_k \left(\exp\left(\sum_{i<j} u(r_{ij})\right)\right) \prod_i \phi(r_i)$$

$$\begin{aligned}
\nabla_k \left(\prod_i \phi(\mathbf{r_i})\right) &= \nabla_k (\phi(r_1)(\phi(r_2)...(\phi(r_k)..(\phi(r_N)) \\
&= \nabla_k \left(e^{-\alpha(x_1^2+y_1^2+z_1^2)} e^{-\alpha(x_2^2+y_2^2+z_2^2)} ... e^{-\alpha(x_k^2+y_k^2+z_k^2)} ... e^{-\alpha(x_N^2+y_N^2+z_N^2)}\right) \\
&= \nabla_k \phi(r_k) \left[\prod_{i \neq k} \phi(\mathbf{r_i})\right]
\end{aligned}$$

$$\nabla_k \exp\left(\sum_{i<j} u(r_{ij})\right) = \nabla_k \exp\big(u(r_{12}) + u(r_{13}) + ... + u(r_{23}) + ... + u(r_{kj}) + .. + u(r_{N-1,N})\big) \tag{A.14}$$

$$= \exp\left(\sum_{i<j} u(r_{ij})\right) \sum_{i \neq k} \nabla_k u(r_{kj}) \tag{A.15}$$

And the first derivative of the trial wave equation is

$$\nabla_k \Psi_T(\mathbf{r}) = \nabla_k \phi(r_k) \left[\prod_{i \neq k} \phi(\mathbf{r_i})\right] \exp\left(\sum_{i<j} u(r_{ij})\right) + \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right) \sum_{j \neq k} \nabla_k u(r_{kj}).$$

Now we find the second derivative of the wave function.

$$\frac{1}{\Psi_T(\mathbf{r})}\nabla_k^2\Psi_T(\mathbf{r}) = \frac{1}{\nabla_k \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right)} \left(\nabla_k \left(\nabla_k \phi(r_k) \left[\prod_{i\neq k} \phi(\mathbf{r_i})\right]\right) \cdot \exp\left(\sum_{i<j} u(r_{ij})\right)\right.$$

$$+ \nabla_k \phi(r_k) \left[\prod_{i\neq k} \phi(\mathbf{r_i})\right] \cdot \nabla_k \left(\exp\left(\sum_{i<j} u(r_{ij})\right)\right)$$

$$+ \nabla_k \left(\prod_i \phi(\mathbf{r_i})\right) \cdot \exp\left(\sum_{i<j} u(r_{ij})\right) \sum_{j\neq k} \nabla_k u(r_{kj})$$

$$+ \prod_i \phi(\mathbf{r_i}) \cdot \nabla_k \left(\exp\left(\sum_{i<j} u(r_{ij})\right)\right) \sum_{j\neq k} \nabla_k u(r_{kj})$$

$$\left. + \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right) \cdot \nabla_k \left(\sum_{j\neq k} \nabla_k u(r_{kj})\right)\right)$$

Solving these equations separately makes it easier.

$$\frac{\nabla_k^2 \phi(r_k) \left[\prod_{i\neq k} \phi(\mathbf{r_i})\right] \cdot \exp\left(\sum_{i<j} u(r_{ij})\right)}{\nabla_k \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right)} = \frac{\nabla_k^2 \phi(r_k)}{\phi(r_k)}$$

$$\frac{\nabla_k \phi(r_k) \left[\prod_{i\neq k} \phi(\mathbf{r_i})\right] \cdot \nabla_k \left(\exp\left(\sum_{i<j} u(r_{ij})\right)\right)}{\nabla_k \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right)} = \frac{\nabla_k \phi(r_k)}{\phi(r_k)} \sum_{j\neq k} \nabla_k u(r_{kj})$$

$$\frac{\nabla_k \left(\prod_i \phi(\mathbf{r_i})\right) \cdot \exp\left(\sum_{i<j} u(r_{ij})\right) \sum_{j\neq k} \nabla_k u(r_{kj})}{\nabla_k \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right)} = \frac{\nabla_k \phi(r_k)}{\phi(r_k)} \sum_{j\neq k} \nabla_k u(r_{kj})$$

$$\frac{\prod_i \phi(\mathbf{r_i}) \cdot \nabla_k \left(\exp\left(\sum_{i<j} u(r_{ij})\right)\right) \sum_{j\neq k} \nabla_k u(r_{kj})}{\nabla_k \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right)} = \sum_{i\neq k} \nabla_k u(r_{ki}) \sum_{j\neq k} \nabla_k u(r_{kj})$$

$$\frac{\prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right) \cdot \nabla_k \left(\sum_{j\neq k} \nabla_k u(r_{kj})\right)}{\nabla_k \prod_i \phi(\mathbf{r_i}) \exp\left(\sum_{i<j} u(r_{ij})\right)} = \sum_{j\neq k} \nabla_k^2 u(r_{kj})$$

Putting them together again we get the following

$$\frac{1}{\Psi_T(\mathbf{r})}\nabla_k^2\Psi_T(\mathbf{r}) = \frac{\nabla_k^2\phi(r_k)}{\phi(r_k)} + 2\frac{\nabla_k\phi(r_k)}{\phi(r_k)}\sum_{j\neq k}\nabla_k u(r_{kj}) + \sum_{i\neq k}\nabla_k u(r_{ki})\sum_{j\neq k}\nabla_k u(r_{kj}) + \sum_{j\neq k}\nabla_k^2 u(r_{kj}) \tag{A.16}$$

We solve the first and second derivatives of $u(r_{kj})$.

$$\nabla_k u(r_{kj}) = \left(\vec{i}\frac{\partial}{\partial x_k} + \vec{j}\frac{\partial}{\partial y_k} + \vec{k}\frac{\partial}{\partial z_k}\right) u(r_{kj}) \tag{A.17}$$

From Rottmann p. 128 we have that

$$\begin{aligned}
\frac{\partial u(r_{kj})}{\partial x_k}\vec{i} &= \frac{\partial u(r_{kj})}{\partial r_{kj}}\frac{\partial r_{kj}}{\partial x_k}\vec{i} \\
&= u'(r_{kj})\frac{\partial\sqrt{|x_k - x_j|^2 + |y_k - y_j|^2 + |z_k - z_j|^2}}{\partial x_k}\vec{i} \\
&= u'(r_{kj})\frac{1}{2}2|x_k - x_j|\frac{1}{r_{kj}}\vec{i} \\
&= \frac{u'(r_{kj})|x_k - x_j|}{r_{kj}}\vec{i} \\
\frac{\partial u(r_{kj})}{\partial y_k}\vec{j} &= \frac{u'(r_{kj})|y_k - y_j|}{r_{kj}}\vec{j} \\
\frac{\partial u(r_{kj})}{\partial z_k}\vec{k} &= \frac{u'(r_{kj})|z_k - z_j|}{r_{kj}}\vec{k}
\end{aligned}$$

where we have used the fact that $r_{kj} = \sqrt{|x_k - x_j|^2 + |y_k - y_j|^2 + |z_k - z_j|^2}$. Now equation (A.17) becomes

$$\nabla_k u(r_{kj}) = u'(r_{kj})\frac{(|x_k - x_j|\vec{i} + |y_k - y_j|\vec{j} + |z_k - z_j|\vec{k})}{r_{kj}} \tag{A.18}$$

$$= u'(r_{kj})\frac{(\boldsymbol{r_k} - \boldsymbol{r_j})}{r_{kj}}. \tag{A.19}$$

And for the second derivative have that

$$\nabla_k^2 u(r_{kj}) = \left(\frac{\partial^2}{\partial x_k^2} + \frac{\partial^2}{\partial y_k^2} + \frac{\partial^2}{\partial z_k^2}\right) u(r_{kj}) \tag{A.20}$$

and use Rottmann p. 128 again and see that

$$\begin{aligned}
\frac{\partial^2 u(r_{kj})}{\partial x_{kj}^2} &= \frac{\partial^2 u(r_{kj})}{\partial r_{kj}^2}\left(\frac{\partial r_{kj}}{\partial x_{kj}}\right)^2 + \frac{\partial u(r_{kj})}{\partial r_{kj}}\frac{\partial^2 r_{kj}}{\partial x_{kj}^2} \\
&= u''(r_{kj})\frac{(x_k-x_j)^2}{r_{kj}^2} + u'(r_{kj})\frac{\partial^2 r_{kj}}{\partial x_{kj}^2} \\
\frac{\partial^2 r_{kj}}{\partial x_{kj}^2} &= \frac{\partial^2\sqrt{|x_k-x_j|^2+|y_k-y_j|^2+|z_k-z_j|^2}}{\partial x_{kj}^2} \\
&= \frac{\partial\frac{x_k-x_j}{\sqrt{|x_k-x_j|^2+|y_k-y_j|^2+|z_k-z_j|^2}}}{\partial x_{kj}} \\
&= \frac{r_{kj}-\frac{1}{2}(x_k-x_j)\cdot\frac{2(x_k-x_j)}{r_{kj}}}{r_{kj}^2} \\
&= \frac{1}{r_{kj}} - \frac{(x_k-x_j)^2}{r_{kj}^3}.
\end{aligned}$$

These equations put together and solving with respect to y and z we get

$$\begin{aligned}
\frac{\partial^2 u(r_{kj})}{\partial x_{kj}^2} &= u''(r_{kj})\frac{(x_k-x_j)^2}{r_{kj}^2} + u'(r_{kj})\left(\frac{1}{r_{kj}} - \frac{(x_k-x_j)^2}{r_{kj}^3}\right), \\
\frac{\partial^2 u(r_{kj})}{\partial y_{kj}^2} &= u''(r_{kj})\frac{(y_k-y_j)^2}{r_{kj}^2} + u'(r_{kj})\left(\frac{1}{r_{kj}} - \frac{(y_k-y_j)^2}{r_{kj}^3}\right), \\
\frac{\partial^2 u(r_{kj})}{\partial z_{kj}^2} &= u''(r_{kj})\frac{(z_k-z_j)^2}{r_{kj}^2} + u'(r_{kj})\left(\frac{1}{r_{kj}} - \frac{(z_k-z_j)^2}{r_{kj}^3}\right).
\end{aligned}$$

Now we can add them all together and equation (A.20) becomes

$$\begin{aligned}
\nabla_k^2 u(r_{kj}) &= \frac{u''(r_{kj})}{r_{kj}^2}((x_k-x_j)^2+(y_k-y_j)^2+(z_k-z_j)^2) \\
&+ u'(r_{kj})\left(\frac{3}{r_{kj}} - \frac{(x_k-x_j)^2+(y_k-y_j)^2+(z_k-z_j)^2}{r_{kj}^3}\right) \\
&= \frac{u''(r_{kj})r_{kj}^2}{r_{kj}^2} + u'(r_{kj})\left(\frac{3}{r_{kj}} - \frac{r_{kj}^2}{r_{kj}^3}\right) \\
&= u''(r_{kj}) + u'(r_{kj})\frac{2}{r_{kj}} \qquad (A.21)
\end{aligned}$$

Now we can write out the complete second derivative, equation (A.16)

$$\frac{1}{\Psi_T(\mathbf{r})}\nabla_k^2\Psi_T(\mathbf{r}) = \frac{\nabla_k^2\phi(r_k)}{\phi(r_k)} + 2\frac{\nabla_k\phi(r_k)}{\phi(r_k)}\sum_{j\neq k}\frac{(\boldsymbol{r_k}-\boldsymbol{r_j})}{r_{kj}}u'(r_{kj})$$
$$+\sum_{ij\neq k}\frac{(\boldsymbol{r_k}-\boldsymbol{r_i})}{r_{ki}}\frac{(\boldsymbol{r_k}-\boldsymbol{r_j})}{r_{kj}}u'(r_{ki})u'(r_{kj}) + \sum_{j\neq k}\left(u''(r_{kj}) + \frac{2}{r_{kj}}u'(r_{kj})\right).$$

For the full analytical solution of the interacting problem we solve each term by it self.

$$\begin{aligned}\frac{1}{\phi(r_k)}\nabla_k^2\phi(r_k) &= \frac{\nabla_k^2\ exp(-\alpha(x_k^2+y_k^2+\beta z_k^2))}{exp(-\alpha(x_k^2+y_k^2+\beta z_k^2))}\\ &= ((2\alpha(2\alpha x_k^2-1)) + (2\alpha(2\alpha y_k^2-1)) + (2\alpha\beta(2\alpha\beta z_k^2-1)))\cdot\frac{\phi(r_k)}{\phi(r_k)}\\ &= -4\alpha^2 - 2\alpha\beta + 4\alpha^2(x_k^2+y_k^2+\beta z_k^2)\end{aligned}$$

$$\begin{aligned}\frac{1}{\phi(r_k)}\nabla_k\phi(r_k) &= \frac{\nabla_k\ exp(-\alpha(x_k^2+y_k^2+\beta z_k^2))}{exp(-\alpha(x_k^2+y_k^2+\beta z_k^2))}\\ &= (2\alpha x_k\vec{i} + 2\alpha y_k\vec{j} + 2\alpha\beta z_k\vec{k})\cdot\frac{\phi(r_k)}{\phi(r_k)}\\ &= (2\alpha x_k\vec{i} + 2\alpha y_k\vec{j} + 2\alpha\beta z_k\vec{k})\end{aligned}$$

Wolfram alpha gives to solution the first and second derivatives of function $u(r_{kj})$.

$$\begin{aligned}u'(r_{kj}) &= \frac{d(\ln f(r_{kj}))}{dr_{kj}}\\ &= \frac{d\ \ln\left(1-\frac{a}{(r_k-r_j)}\right)}{dr_{kj}}\\ &= -\frac{a}{ar_{kj}-r_{kj}^2}\end{aligned}$$

$$\begin{aligned}u''(r_{kj}) &= \frac{d^2(\ln f(r_{kj}))}{dr_{kj}^2}\\ &= \frac{a(a-2r_{kj})}{r_{kj}^2(a-r_{kj})^2}\end{aligned}$$

Putting all of this together we end up with the following expression for the second derivative of the wave equation divided by it self:

$$\begin{aligned}
\frac{1}{\Psi_T(\mathbf{r})}\nabla_k^2\Psi_T(\mathbf{r}) &= -4\alpha - 2\alpha\beta + 4\alpha^2(x_k^2 + y_k^2 + \beta z_k^2) \\
&+ 2((2\alpha x_k\vec{i} + 2\alpha y_k\vec{j} + 2\alpha\beta z_k\vec{k}))\sum_{j\neq k}\left(\frac{(x_k - x_j)\vec{i} + (y_k - y_j)\vec{j} + (z_k - z_j)\vec{k}}{r_{kj}}\left(\frac{-a}{ar_{kj} - r_{kj}^2}\right)\right. \\
&+ \sum_{j\neq k}\left(\frac{(x_k - x_i)\vec{i} + (y_k - y_i)\vec{j} + (z_k - z_i)\vec{k}}{r_{ki}}\right)\left(\frac{(x_k - x_j)\vec{i} + (y_k - y_j)\vec{j} + (z_k - z_j)\vec{k}}{r_{kj}}\right) \\
&* \left(\frac{-a}{ar_{ki} - r_{ki}^2}\right)\left(\frac{-a}{ar_{kj} - r_{kj}^2}\right) \\
&+ \sum_{j\neq k}\left(\frac{a(a - 2r_{kj})}{r_{kj}^2(a - r_{kj})^2} + \frac{2}{r_{kj}}\left(\frac{-a}{ar_{kj} - r_{kj}^2}\right)\right) \qquad (A.22)
\end{aligned}$$

A.1.2 Drift force

As for the drift force we simply write down the solutions for one particle in 1D, 2D and 3D as they are just the first derivative of the wave function,

$$F = \frac{2\nabla\Psi_T}{\Psi_T}.$$

And are easily solved;

$$F = -4\alpha x,$$

$$F = -4\alpha(x + y),$$

$$F = -4\alpha(x + y + \beta z).$$

For N-particles we must solve $\nabla_k\Psi_T$. In 1D this is

$$\nabla_k\Psi_T = -2\alpha x_k\prod_{k\neq i}^{N} e^{-\alpha x_i^2},$$

$$F_k = -4\alpha x_k\frac{\prod_{k\neq i}^{N} e^{-\alpha x_i^2}}{\prod_{i}^{N} e^{-\alpha x_i^2}} = -4\alpha x_k\frac{1}{e^{-\alpha x_k^2}},$$

$$F = \sum_{i}^{N} -4\alpha x_i \frac{1}{e^{-\alpha x_i^2}}.$$

And for 2D and 3D;

$$F = \sum_{i}^{N} -4\alpha(x_i + y_i)\frac{1}{e^{-\alpha(x_i^2+y_i^2)}},$$

$$F = \sum_{i}^{N} -4\alpha(x_i + y_i + \beta z_i)\frac{1}{e^{-\alpha(x_i^2+y_i^2+\beta z_i^2)}}.$$

Finally we can solve for N-particles with interaction.
We have already solved the first derivative of the wave equation when we dealt with the local energy problem.

$$\nabla_k \Psi_T(\mathbf{r}) = \nabla_k \phi(r_k)\left[\prod_{i\neq k}\phi(\mathbf{r_i})\right]\exp\left(\sum_{i<j}u(r_{ij})\right) + \prod_i \phi(\mathbf{r_i})\exp\left(\sum_{i<j}u(r_{ij})\right)\sum_{j\neq k}\nabla_k u(r_{kj})$$

$$\begin{aligned}
\nabla_k \phi(r_k) &= \frac{\partial}{\partial x_k}e^{(-\alpha(x_k^2+y_k^2+\beta z_k^2))}\vec{i} \\
&+ \frac{\partial}{\partial y_k}e^{(-\alpha(x_k^2+y_k^2+\beta z_k^2))}\vec{j} \\
&+ \frac{\partial}{\partial z_k}e^{(-\alpha(x_k^2+y_k^2+\beta z_k^2))}\vec{k} \\
&= (-2\alpha x_k\vec{i} - 2\alpha y_k\vec{j} - 2\alpha\beta z_k\vec{k})e^{(-\alpha(x_k^2+y_k^2+\beta z_k^2))} \\
\nabla_k u(r_{kj}) &= u'(r_{kj})\frac{(\mathbf{r_k} - \mathbf{r_j})}{r_{kj}} \\
&= -\frac{a(\vec{r_k} - \vec{r_j})}{ar_{kj}^2 - r_{kj}^3}
\end{aligned}$$

$$\nabla_k \Psi_T(\mathbf{r}) = (-2\alpha x_k\vec{i} - 2\alpha y_k\vec{j} - 2\alpha\beta z_k\vec{k})\Psi_T(\mathbf{r}) + \sum_{j\neq k}\left(-\frac{a(\vec{r_k} - \vec{r_j})}{ar_{kj}^2 - r_{kj}^3}\right)\Psi_T(\mathbf{r})$$

$$F = (-4\alpha x_k\vec{i} - 4\alpha y_k\vec{j} - 4\alpha\beta z_k\vec{k}) + \sum_{j\neq k}\left(-\frac{a(\vec{r_k} - \vec{r_j})}{ar_{kj}^2 - r_{kj}^3}\right)$$

A.2 Green's function ratio

$$\frac{G(x,y,\Delta t)}{G(y,x,\Delta t)} = \frac{\exp\bigl(-(x-y-DF(y)\Delta t)^2/4D\Delta t\bigr)}{\exp\bigl(-(y-x-DF(x)\Delta t)^2/4D\Delta t\bigr)} \tag{A.23}$$

$$\begin{aligned}(x-y-DF(y)\Delta t)^2 &= x^2 - xy - xDF(y)\Delta t - xy - y^2\\ &\quad + yDF(y)\Delta t - xDF(y)\Delta t + yDF(y)\Delta t + (DF(y)\Delta t)^2\\ &= x^2 + y^2 - 2xy - 2yDF(y)\Delta t + 2xDF(y)\Delta t + (DF(y)\Delta t)^2 \quad (i)\end{aligned}$$

$$(y-x-DF(x)\Delta t)^2 = y^2 + x^2 - 2xy - 2yDF(x)\Delta t + 2xDF(x)\Delta t + (DF(x)\Delta t)^2 \quad (ii)$$

$$\begin{aligned}\frac{G(x,y,\Delta t)}{G(y,x,\Delta t)} &= \exp\left(\frac{-(i)+(ii)}{4D\Delta t}\right)\\ &= \exp\left(\frac{-(DF(y)\Delta t)^2 + 2DF(y)\Delta t(x-y) + (DF(x)\Delta t)^2 + 2DF(x)\Delta t(x-y)}{4D\Delta t}\right)\\ &= \exp\left(\frac{2D\Delta t(x-y)(F(x)+F(y)) + (D\Delta t)^2(F(x)^2 - F(y)^2)}{4D\Delta t}\right)\\ &= \exp\left(\frac{(x-y)(F(x)+F(y)) + 2D\Delta t(F(x)^2 - F(y)^2)}{2}\right)\end{aligned}$$

We then make use of the following equality.

$$(F(x)^2 - F(y)^2) = (F(x)+F(y))(F(x)-F(y)) \tag{A.24}$$

The final result for the Green's function ratio becomes

$$\begin{aligned}\frac{G(x,y,\Delta t)}{G(y,x,\Delta t)} &= \exp\left(\frac{(x-y)(F(x)+F(y)) + 2D\Delta t(F(x)+F(y))(F(x)-F(y))}{2}\right)\\ &= \exp\left(\frac{(F(x)+F(y))((x-y) + 2D\Delta t(F(x)-F(y))}{2}\right)\end{aligned}$$

Chapter B
Appendix B

In this chapter we go through some of the elements need in the RBM method, such as an analytic solution of the local energy and the gradients of the parameters associated with the network.

B.1 Local energy

Local energy with $\psi(x) = P_{rbm}$
Given the Hamiltonian in (8.1) the expression for the local energy is given by the following.

$$\begin{aligned} E_L &= \frac{1}{\Psi}\hat{H}\psi \\ &= \sum_{k}^{P}\frac{1}{2}\left(-\frac{1}{\Psi}\nabla_k^2\Psi + \omega^2 r_k^2\right) + \sum_{k<l}\frac{1}{r_{kl}} \end{aligned}$$

We wish to find the Laplacian of the NQS wave function. Since our visible nodes represents the particle positions in Cartesian coordinates the Laplace operator takes the second partial derivative with respect to each independent variable in the vector space. It can therefore be written as

$$\nabla_k^2\Psi = \sum_{l}^{D}\frac{\partial^2}{\partial x_{kl}^2}\Psi.$$

$$E_L = \sum_{k}^{P}\sum_{l}^{D}\frac{1}{2}\left(-\frac{1}{\Psi}\frac{\partial^2}{\partial x_{kl}^2}\Psi + \omega^2 r_k^2\right) + \sum_{k<l}\frac{1}{r_{kl}}$$

$$E_L = \sum_{i}^{M}\frac{1}{2}\left(-\frac{1}{\Psi}\frac{\partial^2}{\partial X_i^2}\Psi + \omega^2 X_i^2\right) + \sum_{k<l}\frac{1}{r_{kl}}$$

In the above equation we have used the fact that the number of visible nodes are $M = P \cdot D$. This reduces the number of sums in the first term to one. Also we see that it would be an advantage to solve the derivatives of $\ln\Psi$ since our wave function consist of sums in the exponential and also removes the Ψ in the denominator. Thus the identity below is helpful.

$$\frac{1}{\Psi}\frac{\partial^2}{\partial X_i^2}\Psi = \left(\frac{\partial}{\partial X_i}\ln\Psi\right)^2 + \frac{\partial^2}{\partial X_i^2}\ln\Psi$$

$$E_L = \sum_i^M \frac{1}{2}\left(-\left(\frac{\partial}{\partial X_i}\ln\Psi\right)^2 - \frac{\partial^2}{\partial X_i^2}\ln\Psi + \omega^2 X_i^2\right) + \sum_{k<l}\frac{1}{r_{kl}}$$

Our local energy can be rewritten and we must now solve both the first and second derivative of the wave function. Our wave function is represented by the marginal probability F_{rbm}.

$$\begin{aligned}\Psi(X) &= P_{rbm}(X)\\ &= \frac{1}{Z}\exp\left(-\sum_i^M \frac{(X_i-a_i)^2}{2\sigma^2}\right)\prod_j^N\left(1+\exp\left(b_j+\sum_i^M\frac{X_i\omega_{ij}}{\sigma^2}\right)\right)\end{aligned}$$

$$\ln\Psi = -\ln Z - \sum_i^M \frac{(X_i-a_i)^2}{2\sigma^2} + \sum_j^N \ln\left(1+\exp\left(b_j+\sum_i^M\frac{X_i\omega_{ij}}{\sigma^2}\right)\right)$$

Since Z is the normalization constant its derivative is zero and can be removed. We end up with the following when calculating the derivative of particle k.

$$\begin{aligned}\nabla_k\ln\Psi &= \nabla_k\left(-\sum_i^M \frac{(X_i-a_i)^2}{2\sigma^2} + \sum_j^N \ln\left(1+\exp\left(b_j+\sum_i^M\frac{X_i\omega_{ij}}{\sigma^2}\right)\right)\right)\\ &= \frac{\partial}{\partial X_k}\left(-\sum_i^M \frac{(X_i-a_i)^2}{2\sigma^2} + \sum_j^N \ln\left(1+\exp\left(b_j+\sum_i^M\frac{X_i\omega_{ij}}{\sigma^2}\right)\right)\right)\\ &= \frac{-2(X_k-a_k)2\sigma^2}{(2\sigma^2)^2} + \sum_j^N \frac{1}{1+\exp\left(b_j+\sum_i^M\frac{X_i\omega_{ij}}{\sigma^2}\right)}\left(\frac{\omega_{kj}}{\sigma^2}\right)\exp\left(b_j+\sum_i^M\frac{X_i\omega_{ij}}{\sigma^2}\right)\\ &= \frac{-(X_k-a_k)}{\sigma^2} + \frac{1}{\sigma^2}\sum_j^N\frac{\omega_{kj}}{1+\exp\left(-b_j-\sum_i^M\frac{X_i\omega_{ij}}{\sigma^2}\right)} \qquad \text{(B.1)}\end{aligned}$$

Given the first derivative we now find the second derivative. And we have solved the analytic case of the local energy.

$$\begin{aligned}
\nabla_k^2 \ln \Psi &= \nabla_k^2 \left(-\sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^N \ln \left(1 + \exp \left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right) \right) \right) \\
&= \frac{\partial}{\partial X_k} \left(\frac{-(X_i - a_i)}{\sigma^2} + \frac{1}{\sigma^2} \sum_j^N \frac{\omega_{ij}}{1 + \exp \left(-b_j - \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right)} \right) \\
&= -\frac{1}{\sigma^2} + \frac{1}{\sigma^2} \sum_j^N \frac{-\omega_{kj}}{\left(1 + \exp \left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right) \right)^2} \left(\frac{-\omega_{kj}}{\sigma^2} \right) \exp \left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right) \\
&= -\frac{1}{\sigma^2} + \frac{1}{\sigma^4} \sum_j^N \frac{\left(\omega_{kj} \right)^2}{\left(1 + \exp \left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right) \right)^2} \exp \left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right)
\end{aligned} \tag{B.2}$$

B.2 Gradient

Gradients with respect to RMB parameters, $\psi(x) = P_{rbm}$ To find the local energy minimum of the NQS wavefunction the gradient of the local energy with respect to the variational parameter $\alpha = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$ needs to be computed. For each variational parameter $\alpha_i \in \alpha$ the gradient is defined as in equation B.3

$$G_i = \frac{dE_L}{d\alpha_i} = 2 \left(\left\langle \frac{1}{\Psi_T} \frac{d\Psi_T}{d\alpha_i} E_L \right\rangle - \left\langle \frac{1}{\Psi_T} \frac{d\Psi_T}{d\alpha_i} \right\rangle \langle E_L \rangle \right) \tag{B.3}$$

$$\Psi(\mathbf{X}) = \frac{1}{Z} \exp \left(-\sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} \right) \prod_j^N \left(1 + \exp \left(b_j + \sum_i^M \frac{X_i w_{ij}^{\mathbf{T}}}{\sigma^2} \right) \right) \tag{B.4}$$

We now find it useful to use the identity

$$\frac{d}{dx} \ln f(x) = \frac{1}{f(x)} \frac{d}{dx} f(x) \tag{B.5}$$

Taking the logarithm of the NQS wavefunction then gives

$$\ln\big(\Psi(\mathbf{X})\big) = \ln\frac{1}{Z} - \sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^N \ln\left[1 + \exp\left(b_j + \sum_i^M \frac{X_i w_{ij}^{\mathbf{T}}}{\sigma^2}\right)\right] \tag{B.6}$$

The final result for the gradient of the components of α_i is then

$$\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial a_k} = \frac{1}{\sigma^2}(X_k - a_k) \tag{B.7}$$

$$\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial b_n} = \left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}^{\mathbf{T}}}{\sigma^2}\right)\right)^{-1} \tag{B.8}$$

$$\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial w_{kn}} = \frac{X_k}{\sigma^2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}^{\mathbf{T}}}{\sigma^2}\right)\right)^{-1} \tag{B.9}$$

B.3 Local energy

Local energy with $\psi(x) = \sqrt{P_{rbm}}$ In the case of the Gibbs sampling we represent the wave function as

$$\begin{aligned}\Psi(X) &= \sqrt{P_{rbm}(X)} \\ &= \sqrt{\frac{1}{Z}\exp\left(-\sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2}\right)\prod_j^N \left(1 + \exp\left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right)\right)}.\end{aligned}$$

Which is simply reduced to a constant in front of the term when taking the natural logarithm of the wave function.

$$\ln\Psi = -\frac{1}{2}\left(\ln Z - \sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^N \ln\left(1 + \exp\left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right)\right)\right)$$

Now we can easily see that the derivatives of the logarithm of the nqs wave function becomes,

$$\nabla_k \ln \Psi = \nabla_k \left(-\sum_i^M \frac{(X_i - a_i)^2}{4\sigma^2} + \frac{1}{2}\sum_j^N \ln \left(1 + \exp \left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right) \right) \right)$$

$$= \frac{-(X_k - a_k)}{2\sigma^2} + \frac{1}{2\sigma^2} \sum_j^N \frac{\omega_{kj}}{1 + \exp\left(-b_j - \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)},$$

$$\nabla_k^2 \ln \Psi = \nabla_k^2 \left(-\sum_i^M \frac{(X_i - a_i)^2}{4\sigma^2} + \frac{1}{2}\sum_j^N \ln \left(1 + \exp \left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2} \right) \right) \right)$$

$$= -\frac{1}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_j^N \frac{\left(\omega_{kj}\right)^2}{\left(1 + \exp\left(-b_j - \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)\right)^2} \exp\left(-b_j - \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2} \right)$$

and are the derivative to be replaced in the expression of the local energy when we employ the Gibbs sampling.

B.4 Gradients

Gradients wrt. RMB parameters, $\psi(x) = \sqrt{P_{rbm}}$ We also need to update the gradients to be used in the gradient descent method. These are also easily found, a simple constant i front of each term.

$$\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial a_k} = \frac{1}{2\sigma^2}(X_k - a_k) \tag{B.10}$$

$$\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial b_n} = \frac{1}{2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}}{\sigma^2} \right) \right)^{-1} \tag{B.11}$$

$$\frac{\partial \ln\big(\Psi(\mathbf{X})\big)}{\partial w_{kn}} = \frac{X_k}{2\sigma^2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}}{\sigma^2} \right) \right)^{-1} \tag{B.12}$$

Bibliography

[1] Isaiah Shavitt and Rodney J Bartlett. *Many-body methods in chemistry and physics: MBPT and coupled-cluster theory.* Cambridge university press, 2009.

[2] Yoav Freund and David Haussler. “Unsupervised learning of distributions on binary vectors using two layer networks”. In: *Advances in neural information processing systems.* 1992, pp. 912–919.

[3] Nicolas Le Roux and Yoshua Bengio. “Representational power of restricted Boltzmann machines and deep belief networks”. In: *Neural computation* 20.6 (2008), pp. 1631–1649.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[5] Xun Gao and Lu-Ming Duan. “Efficient representation of quantum many-body states with deep neural networks”. In: *Nature communications* 8.1 (2017), pp. 1–6.

[6] Hiroki Saito. “Method to solve quantum few-body problems with artificial neural networks”. In: *Journal of the Physical Society of Japan* 87.7 (2018), p. 074002.

[7] Junwei Liu et al. “Self-learning monte carlo method”. In: *Physical Review B* 95.4 (2017), p. 041101.

[8] Dong-Ling Deng, Xiaopeng Li, and S Das Sarma. “Quantum entanglement in neural network states”. In: *Physical Review X* 7.2 (2017), p. 021021.

[9] A. Raklev, T. Engeland, M. Hjorth-Jensen, S. Viefers. *Kompendium i FYS2140, Kvantefysikk.* Lecture Notes. Aug. 2015. URL: `https://www.uio.no/studier/emner/matnat/fys/FYS2140/v15/kompendium/kompendium.pdf`.

[10] S. Kvaal. *Lecture notes for FYS-KJM4480.* Lecture notes. Nov. 2017. URL: `http://theory.rutgers.edu/~giese/notes/DFT.pdf`.

[11] Christopher J Pethick and Henrik Smith. *Bose-Einstein condensation in dilute gases.* Cambridge university press, 2008.

[12] David J Griffiths. “Introduction to quantum mechanics”. In: *2nd, Pearson, Chapter2. The time-independent schrodinger equation* (2005), pp. 70–73.

[13] Frederic Chevy and Christophe Salomon. “Strongly correlated Bose gases”. In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 49.19 (2016), p. 192001.

[14] Immanuel Bloch, Jean Dalibard, and Wilhelm Zwerger. “Many-body physics with ultracold gases”. In: *Reviews of modern physics* 80.3 (2008), p. 885.

[15] C.J Pethick and H. Smith. *Bose-Einstein Condensation in Dilute Gases.* Cambridge university press, 2009.

[16] Daniel V. Schroeder. *An Introduction to Thermal Physics.* Pearson, 2002.

[17] E Buffet, Ph de Smedt, and JV Pulé. “The condensate equation for some Bose systems”. In: *Journal of Physics A: Mathematical and General* 16.18 (1983), p. 4307.

[18] Joris Lauwers, André Verbeure, and VA Zagrebnov. “Bose–Einstein condensation for homogeneous interacting systems with a one-particle spectral gap”. In: *Journal of statistical physics* 112.1-2 (2003), pp. 397–420.

[19] T. Lindstrøm and K. Hveberg. *Flervariabel analyse med lineær algebra.* Gyldendal akademiske, 2011.

[20] RJ Wild et al. “Measurements of Tan’s contact in an atomic Bose-Einstein condensate”. In: *Physical review letters* 108.14 (2012), p. 145305.

[21] Igor Ferrier-Barbut et al. “Observation of quantum droplets in a strongly dipolar Bose gas”. In: *Physical review letters* 116.21 (2016), p. 215301.

[22] Oliver Penrose and Lars Onsager. “Bose-Einstein condensation and liquid helium”. In: *Physical Review* 104.3 (1956), p. 576.

[23] WF Vinen. “The physics of superfluid helium”. In: (2004).

[24] JL DuBois and HR Glyde. “Bose-Einstein condensation in trapped bosons: A variational Monte Carlo analysis”. In: *Physical Review A* 63.2 (2001), p. 023602.

[25] Kaoru Ohno, Keivan Esfarjani, and Yoshiyuki Kawazoe. *Computational materials science: from ab initio to Monte Carlo methods.* Springer, 1999.

[26] William Lauchlin McMillan. “Ground state of liquid He 4”. In: *Physical Review* 138.2A (1965), A442.

[27] MH Kalos et al. “Modern potentials and the properties of condensed he 4”. In: *Physical Review B* 24.1 (1981), p. 115.

[28] M. Hjorth-Jensen. *Computational Physics.* Lecture notes. Aug. 2015. URL: `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf`.

[29] Julien Toulouse, Roland Assaraf, and Cyrus J Umrigar. “Introduction to the variational and diffusion Monte Carlo methods”. In: *Advances in Quantum Chemistry.* Vol. 73. Elsevier, 2016, pp. 285–314.

[30] Nicholas Metropolis et al. “Equation of state calculations by fast computing machines”. In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[31] Marius Jonsson. “Standard error estimation by an automated blocking method”. In: *Physical Review E* 98.4 (2018), p. 043304.

[32] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[33] Maryellen L Giger. “Update on the potential of computer-aided diagnosis for breast cancer”. In: *Future Oncology* 6.1 (2010), pp. 1–4.

[34] Alan Morningstar and Roger G Melko. “Deep learning the ising model near criticality”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 5975–5991.

[35] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[36] Hugo Mayo et al. *History of Machine Learning*. 2018. URL: `https://www.doc.ic.ac.uk/~jce317/history-machine-learning.html#top` (visited on 11/07/2019).

[37] Arthur L Samuel. “Some studies in machine learning using the game of checkers. II—recent progress”. In: *Computer Games I*. Springer, 1988, pp. 366–400.

[38] Stephen Marsland. *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2014.

[39] Pankaj Mehta et al. “A high-bias, low-variance introduction to machine learning for physicists”. In: *Physics Reports* (2019).

[40] Bernard Marr. *How much data do we create every day? The mind-blowing stats everyone should read*. 2018. URL: `https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#10d310be60ba` (visited on 2020).

[41] Devinderjit Sivia and John Skilling. *Data analysis: a Bayesian tutorial*. OUP Oxford, 2006.

[42] Vilde Moe Flugsrud. “Solving Quantum Mechanical Problems with Machine Learning”. MA thesis. 2018.

[43] The Python Software Foundation. *The Python Tutorial Classes*. 2020. URL: `https://docs.python.org/3/tutorial/classes.html#class-objects` (visited on 02/03/2020).

[44] Franco Dalfovo et al. “Theory of Bose-Einstein condensation in trapped gases”. In: *Reviews of Modern Physics* 71.3 (1999), p. 463.

[45] J De Boer and A Michels. “Quantum-mechanical calculation of the second virial-coefficient of helium at low temperatures”. In: *Physica* 6.5 (1939), pp. 409–420.

[46] Daniel Schiff and Loup Verlet. “Ground state of liquid helium-4 and helium-3”. In: *Physical Review* 160.1 (1967), p. 208.

[47] AR Sakhel, JL DuBois, and HR Glyde. “Bose-Einstein condensates in 85 Rb gases at higher densities”. In: *Physical Review A* 66.6 (2002), p. 063610.

[48] JK Nilsen et al. “Vortices in atomic bose-einstein condensates in the large-gas-parameter region”. In: *Physical Review A* 71.5 (2005), p. 053610.

[49] D Ceperley, GV Chester, and MH Kalos. “Monte Carlo simulation of a many-fermion study”. In: *Physical Review B* 16.7 (1977), p. 3081.

[50] Jonathan Barzilai and Jonathan M Borwein. “Two-point step size gradient methods”. In: *IMA journal of numerical analysis* 8.1 (1988), pp. 141–148.

[51] Giuseppe Carleo and Matthias Troyer. “Solving the quantum many-body problem with artificial neural networks”. In: *Science* 355.6325 (2017), pp. 602–606.

[52] Doerte Blume and Chris H Greene. “Quantum corrections to the ground-state energy of a trapped Bose-Einstein condensate: A diffusion Monte Carlo calculation”. In: *Physical Review A* 63.6 (2001), p. 063601.

[53] áM Taut. “Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem”. In: *Physical Review A* 48.5 (1993), p. 3561.

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