UiO : Department of Physics
University of Oslo

FYS4411: Computational Physics II

Project 2

Variational Monte Carlo with Neural Network Quantum States

Jørn Eirik Betten Aleksandar Davidov Nicolai Haug

June 1, 2022

Abstract

The many-body wave function increases exponentially in complexity with the number of particles, and therefore, clever approximations to it is of great interest. The universal approximation theorem states that any function can be approximated to an arbitrary error by a neural network. We will therefore seek to implement an approach based on a machine learning method. An approach based on a neural network for solving the quantum mechanical wave function is still a relatively new, but an increasingly interesting prospect (Saito, 2018). This project analyzes two systems of electrons, a quantum dot in a one-dimensional harmonic oscillator trap and a pair of interacting electrons in an isotropic two-dimensional harmonic oscillator trap. We analyze the systems using an unsupervised learning method, the restricted Boltzmann machine (RBM), to simulate the wave function (known as a Neural-network Quantum State(Carleo and Troyer, 2017)), and generate upper bound estimates to the ground state energies using two Markov Chain Monte Carlo algorithms. We perform some coarse searches in the space of hyper parameters, like learning rate, batch-sizes for optimization and number of neurons in hidden layer. After finding the optimal set of parameters, we find that we approximate the ground state energy for the quantum dot in a single dimension to a high degree of precision. We find the best approximation to the ground state energy to be $E_0 = 0.499999 \pm 3 \cdot 10^{-6}$ a.u, using the RWM sampling algorithm. For the system of two interacting electrons in two-dimensional space we find the best approximation, again via the RWM sampling algorithm, to be $E_0 = 3.059 \pm 0.008$ a.u. Knowing the true ground state energy to be $E_0 = 3.0$ a.u (Taut, 1993), there is still plenty of room for improvement.

Contents

1 Introduction

In recent years, using machine learning to solve quantum mechanical systems has become of great interest for many. The complexity of the many-body quantum wave function increases exponentially with the number of particles, and is therefore computationally very costly, often intractable for practical purposes. Machine learning models may be of practical use in this, as they are designed to find statistical correlations between high-dimensional feature space. Recently, representing wave functions as a Restricted Boltzmann Machine (RBM) has been presented by G. Carleo and M. Troyer where they applied it to quantum mechanical spin lattice system of the Ising model and Heisenberg model (Carleo and Troyer, 2017). They dubbed the approach of using a neural network to represent a quantum state *Neural network Quantum States* (NQS). In this project, we will fit an RBM to two systems; one electron in a one-dimensional harmonic oscillator trap and two interacting electrons confined in an isotropic two-dimensional harmonic oscillator trap. We will be using a reinforcment learning approach using a generative method to evaluate and update the RBM, and utilize the variational principle to find the system's ground state. Specifically, the RBM will act as an approximation to the wave function which we will perform calculations on using different Markov Chain Monte Carlo (MCMC) approaches. This project can be thought of as an extension of (Haug et al., 2022), and for reviews and descriptions of principles regarding the MCMC approaches we will refer to the previous project. We will next (in Section 2) have a look at the systems we are performing calculations on, then look at theoretical applicability of the RBM applied as an NQS. In Section 3 we will display short descriptions of the methods applied in this project, as many of them are very similar to those used in our previous project (Haug et al., 2022). Section 4 will display the results provided by MCMC approach using the NQS as trial wave function and discussions regarding those, where Section 5 will provide conclusions to the approach. Finally, in Section 6 we will look at interesting avenues for future research.

2 Theory

2.1 Systems

We will be considering two system; one electron in a one-dimensional harmonic oscillator trap, and two electrons interacting via the Coulomb interactions in a two-dimensional isotropic harmonic oscillator trap.

A general idealized Hamiltonian for N interacting particles in a d-dimensional isotropic trap is given by

$$H = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 \boldsymbol{r_i} \cdot \boldsymbol{r_i} \right) + \sum_{i<j} \frac{1}{r_{ij}}, \tag{2.1}$$

where we are using the natural units given by $\hbar = c = e = m_e = k_e = 1$. The product $\boldsymbol{r}_i \cdot \boldsymbol{r}_i$ denotes the dot product of the positions of particle i, and all the energies are in atomic units (a.u). The kinetic energy of the electrons is given $\sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2\right)$, the potential energy with respect to the trap potential is given $\sum_{i=1}^{N} \left(\frac{1}{2} D\omega^2 r_i^2\right)$, and the Coulomb interactions are modeled as the inverse Euclidian distance between the electrons ($r_{ij} = \|\boldsymbol{r}_i - \boldsymbol{r}_j\|_2$). Notice that $\sum_{i<j}$ denotes a double sum, such that

$$\sum_{i<j} \frac{1}{r_{ij}} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{1}{r_{ij}}. \tag{2.2}$$

We denote the non-interacting part of the Hamiltonian H_0 and the Coulomb interaction H_1, such that for an interacting system we get $H = H_0 + H_1$. For the system of a single particle in the one-dimensional harmonic oscillator trap, the Hamiltonian becomes

$$H = -\frac{1}{2}\nabla^2 + \frac{1}{2}\omega^2 x^2, \tag{2.3}$$

where x denotes the coordinate with respect to the center of the trap. The general eigenstate of the 1D Hamiltonian is given by

$$\phi_n = AH_n\Big(\sqrt{\omega}x\Big)\exp\Big\{-\omega x^2/2\Big\}, \tag{2.4}$$

where A is a normalization constant, $H_n(\sqrt{\omega}x)$ are the Hermite polynomials with respect to the quantum number n. The corresponding energy eigenvalue of the eigenstate is $E_n = \omega\Big(n + \frac{1}{2}\Big)$ a.u. The ground state of this system occurs when $n = 0$, and is given as

$$\phi_0 = A\exp\Big\{-\omega x^2\Big\}, \tag{2.5}$$

where $A = \left(\frac{\omega}{\pi}\right)^{\frac{1}{4}}$ is such that $\langle\phi_0|\phi_0\rangle = \int_{x\in\mathbb{R}} |\phi_0|^2 \mathrm{d}x = 1$. The energy of the ground state is $E_0 = \omega\frac{1}{2}$ a.u.

The system of two Coulomb interacting particles in a two-dimensional isotropic harmonic oscillator trap, has the Hamiltonian

$$H = -\frac{1}{2}\Big(\nabla_1^2 + \nabla_2^2\Big) + \frac{\omega^2}{2}(\boldsymbol{r}_1 \cdot \boldsymbol{r}_1 + \boldsymbol{r}_2 \cdot \boldsymbol{r}_2) + \frac{1}{r_{12}}, \tag{2.6}$$

and its analytical ground state energy E_0 is 3 a.u., shown in (Taut, 1993). The non-interacting part of the Hamiltonian has the analtyical eigenstate

$$\phi_{n_x,n_y}(x,y) = AH_{n_x}(\sqrt{\omega}x)H_{n_y}(\sqrt{\omega}y)\exp\Big\{(-\omega(x^2+y^2)/2)\Big\}, \tag{2.7}$$

with the corresponding energy eigenvalue of

$$E_{n_x,n_y} = \omega(n_x + n_y + 1),$$

which yields the ground state $n_x = n_y = 0$, $E_{0,0} = \omega$. The two electrons can have the same quantum numbers due to their spin quantum number being different, which does not affect the energy. However, if there are more electrons, the wave functions need to take into account that two fermions cannot occupy the same quantum numbers. We say that the degeneracy of the ground state wave function for the harmonic oscillator is two.

2.2 Neural Networks

Artificial Neural Networks, or simply just Neural Networks (NN), are computer systems which are modeled after the biological neural networks of animal brains. Just like biological brains, Neural Networks consists of neurons which either activate or or not depending on if a

given threshold is reached. Each connection of a Neural Network can be compared with the synapses of a biological brain. They transmit signals to other neurons. However, biological neural networks learn through a process called synaptic plasticity, while artificial while learn by adjusting it's weights and biases in order to minimize a given cost function. The weights of the Neural Network are the connections between two neurons. In Graph Theory, we would refer to them as (weighted) edges between nodes. The bias term shifts the input of a neuron. The general approach to neural networks is to train it using gradients of a *cost function*, i.e. a distance between the ground truth labels and the prediction yielded. Then perform a gradient-based approach to update the parameters of the neural network.
A Neural Network consists of an input layer, a hidden layer, or multiple hidden layers as in the case of deep neural networks, and an output layer. An example of a neural network architecture with three hidden layers and three outputs is displayed in Figure 2.1.

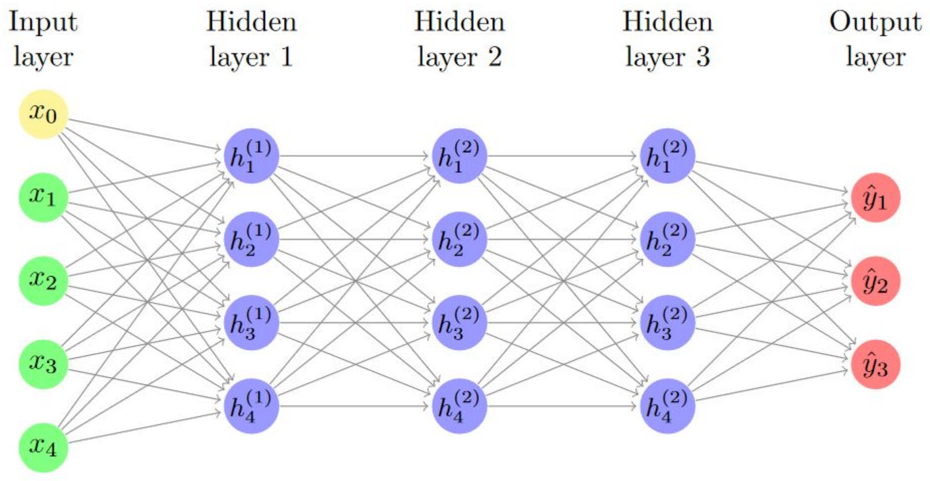

Figure 2.1: *A simple Feed-forward Neural Network with 3 input neurons, 3 output neuron and 3 layers of hidden layers consisting of 4 neurons each.*

Generally, we can group machine learning in three general categories; supervised, unsupervised and reinforcement learning. If you have a data set $\mathscr{D}$ with known ground truth labels and train the neural network to learn a function for the features of the network that produces a general output equal that corresponds well to a general data set of same distribution as $\mathscr{D}$, we have supervised learning. In the case of a classification problem, the Neural Network will activate one of the output neurons corresponding to the label of the classified object, while in a regression problem, the single output neuron will return the predicted output value. Unsupervised learning takes a data set without known ground truth labels and tries to find correlations between the data, or learns a distribution using some other criterion than a cost function including ground truth labels. Reinforcement learning is performed when we do not have ground truth labels and use some kind of penalty and reward system to evaluate generated output. In our case we train the network based on the variational principle (see (Haug et al., 2022)), which states that any approximation to the expectation value of energy is greater or equal to the ground state energy.

2.3 Boltzmann machines

A *Boltzmann machine* (BM) is an undirected probabilistic graphical model with stochastic continuous, or discrete, nodes. It consists of one visible and one hidden layer, which are both inter- and intraconnected, meaning that there are weight matrices that represents the strength of the interactions between the nodes (both within the same layer and connections to the nodes in the other layer). The Boltzmann machine is a generative model as it allows generating new samples from a learned distribution. These properties are useful in our search for a ground state wave function, however, the Boltzmann machine requires a large computational cost to train. We will therefore employ a *restricted Boltzmann machine* (RBM) to do our bidding. Figure 2.2 is a simple representation of a BM network, where all the nodes are connected.

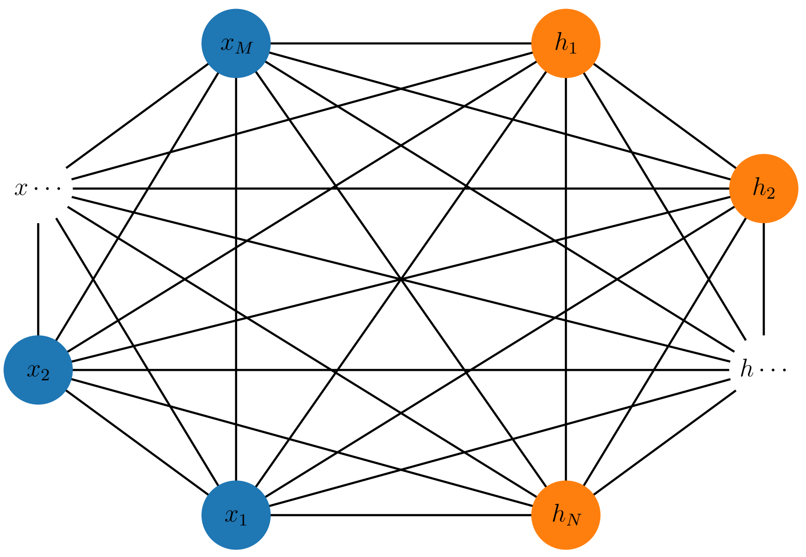

Figure 2.2: *Simple visualization of a BM network with a visible layer of n nodes and a hidden layer of m nodes. The links between the nodes are weighted, and they are all contained within a weight matrix, W. The BM network is fully connected between all nodes. Notice that there are no intraconnections between the layers. Every link is undirected, as the connections go both ways.*

2.4 Restricted Boltzmann Machines

The visible and hidden layer of the RBM is interconnected, but not intraconnected. The connections between the hidden layer, $\boldsymbol{h} \in \mathbb{R}^N$, and the visible layer, $\boldsymbol{x} \in \mathbb{R}^M$, are weighted by the weight matrix $W \in \mathbb{R}^{M\times N}$, represented by the edges in Figure 2.3. The layers can both have biases of equal length. We denote the bias for the visible layer, $\boldsymbol{a}$, and for the hidden layer, $\boldsymbol{b}$. The most common type of an RBM is the so-called *binary-binary RBM*, but to be able to represent the continuous configuration space of the particles, we will use a *Gaussian-binary RBM* implementation.

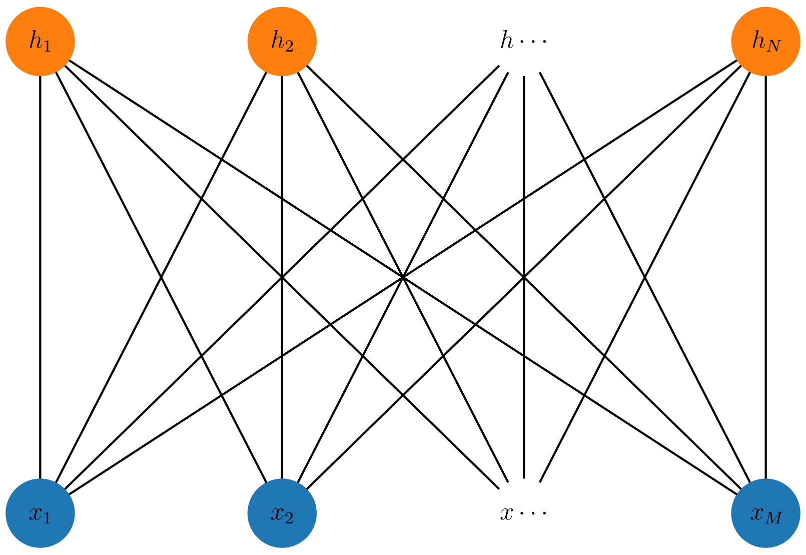

Figure 2.3: *Simple visualization of an RBM network with a visible layer of n nodes and hidden layer layer of m nodes. The links between the nodes are weighted, and they are all contained within a weight matrix, W. The RBM network is fully connected between the layers. Every link is undirected, as the connections go both ways.*

2.4.1 Representing the wave function using a neural network.

Approximating a high dimensional wave function to a high degree of certainty requires a large computational cost, and often becomes practically intractable for large system. We want an RBM to learn the wave function (our system has a wave function fully defined in the real domain) of the quantum mechanical system we want to simulate, and generate samples using the RBM as our system. The RBM representation of the wave function was first done by (Carleo and Troyer, 2017) and they coined the general approximation of a wave function using a neural network method a *neural network quantum state*, or NQS. When applied to large system of particles, the NQS may offer a lower computational cost compared with traditional methods.

The Boltzmann distribution describes the probability of a system being in a certain state as a function of energy and temperature. It is widely used in statistical physics and mathematics. Mathematically, it can be stated as

$$p_i = \frac{e^{-\frac{1}{T}E_i}}{Z}, \tag{2.8}$$

where E_i is the energy of state $|i\rangle$, T is the 'temperature' of the system and Z denotes the *partition function*. The partition function has the form

$$Z = \sum_{j=0}^{J-1} e^{-\frac{1}{T}E_j},$$

where J is the number of possible states, and is the normalization condition for the probability p_i. Now, if we define the *energy function* of the neural network to be a function of the visible and hidden layers, $E(\boldsymbol{x},\boldsymbol{h})$, we can use the Boltzmann distribution to define a joint probability function

$$p(\boldsymbol{x},\boldsymbol{h}) = \frac{e^{-\frac{1}{T}E(\boldsymbol{x},\boldsymbol{h})}}{Z}. \tag{2.9}$$

The partition function, Z, will then be the sum of all possible configurations of visible and hidden nodes,

$$Z = \int\int e^{-\frac{1}{T}E(\boldsymbol{x},\boldsymbol{h})}\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{h}. \tag{2.10}$$

Generally, calculating the partition function is intractable (or just inconvenient), and is luckily not needed. The reason is that we use Markov chain Monte Carlo (MCMC) methods that only need the *ratio* between states to generate new samples for the chain. To see this, say $p = p(\boldsymbol{x},\boldsymbol{h})$ is the joint probability for state $\boldsymbol{x}$ and $p' = p(\boldsymbol{x'},\boldsymbol{h})$ is the joint probability for state $\boldsymbol{x'}$, we get the ratio

$$\frac{p'}{p} = \exp\Big\{-\frac{1}{T}E(\boldsymbol{x'},\boldsymbol{h}) + \frac{1}{T}E(\boldsymbol{x},\boldsymbol{h})\Big\}.$$

Further, we will set the 'temperature' parameter equal to 1.

2.5 Binary-Binary RBM

The Binary-Binary RBM discrete configuration space. Here, we use binary units in both the visible and hidden layer where the binary values most commonly taken on by the nodes are 0 and 1. The corresponding energy function is defined as follows:

$$E(\mathbf{x},\mathbf{h}) = -\sum_i^M x_i a_i - \sum_j^N b_j h_j - \sum_{i,j}^{M,N} x_i w_{ij} h_j,$$

or in vector notation as

$$E(\mathbf{x},\mathbf{h}) = -\mathbf{x}_i\mathbf{a}_i - \mathbf{b}_j\mathbf{h}_j - \mathbf{x}_i\mathbf{w}_{i,j}\mathbf{h}_j.$$

2.6 Gaussian-Binary RBM

The Gaussian-Binary RBM can represent continuous configuration space, such as our systems of interest. Another advantage it has, is that the Gaussian part of the wave function can easily be approximated by the Gaussian part of the RBM, and thus it should be a suitable choice of wave function. The energy function $E(\boldsymbol{x},\boldsymbol{h})$, is given as

$$E(\boldsymbol{x},\boldsymbol{h}) = \sum_i^M \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_j^N b_j h_j - \sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2}$$

or, in vector notation, as

$$E(\boldsymbol{x},\boldsymbol{h}) = \frac{\|\boldsymbol{x}-\boldsymbol{a}\|^2}{2\sigma^2} - \boldsymbol{b}^T\boldsymbol{h} - \frac{\boldsymbol{x}\boldsymbol{W}\boldsymbol{h}}{\sigma^2}. \tag{2.11}$$

σ^2 will refer to the expected variance in the Gaussian layer of the model, which will be treated as a hyper parameter in this project.

2.7 Optimizing the RBM

We will treat the expectation value of the local energy as the cost function of our RBM and try and minimize it, as according to the variational principle.

2.7.1 Derivations of local energy

We want the NQS marginal probability distribution over the visible layer to mimic the wave function (alternatively, the square of the wave function). Therefore,

$$p(\boldsymbol{x}) = \sum_{j=1}^{N} \frac{\mathrm{e}^{-\frac{1}{T}E(\boldsymbol{x},\boldsymbol{h})}}{Z},$$

will act as our trial wave function, $\Psi_T(\boldsymbol{x})$. For a Gaussian-Binary RBM, the marginal PDF can be expanded into

$$\Psi_T(\boldsymbol{x}) = \frac{\mathrm{e}^{-\frac{\|\boldsymbol{x}-\boldsymbol{a}\|^2}{2\sigma^2}}}{Z} \prod_{j}^{N} \left(1 + \mathrm{e}^{b_j + \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}}\right).$$

To compute the kinetic energy of the system (K), we will apply the formula

$$K = -\frac{1}{2} \sum_{i}^{M} \left(\left(\frac{\partial \ln \Psi}{\partial x_i} \right)^2 + \frac{\partial^2 \ln \Psi}{\partial x_i^2} \right),$$

which is derived in (Haug et al., 2022). The trial wave function in the logarithmic domain is

$$\ln \Psi_T(\boldsymbol{x}) = -\frac{\|\boldsymbol{x}-\boldsymbol{a}\|^2}{2\sigma^2} + \sum_{j}^{N} \ln\left(1 + \mathrm{e}^{b_j + \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}}\right) - \ln Z.$$

We will omit the partition function in the derivatives, because

$$\frac{\partial \ln \Psi_T}{\partial x} = \frac{1}{\Psi_T} \frac{\partial \Psi_T}{\partial x},$$

which leads to the ability to factorize away Z. Thus the first partial derivative with respect to input x_i becomes

$$\begin{aligned}
\frac{\partial \ln \Psi_T(\boldsymbol{x})}{\partial x_i} &= -\frac{x_i - a_i}{\sigma^2} + \sum_{j}^{N} \frac{W_{ij}}{\sigma^2} \left(\frac{\mathrm{e}^{b_j + \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}}}{1 + \mathrm{e}^{b_j + \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}}} \right) \\
&= -\frac{x_i - a_i}{\sigma^2} + \sum_{j}^{N} \frac{W_{ij}}{\sigma^2} \frac{1}{\mathrm{e}^{-b_j - \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}} + 1},
\end{aligned}$$

while the second partial derivative is

$$\begin{aligned}
\frac{\partial^2 \ln \Psi_T(\boldsymbol{x})}{\partial x_i^2} &= -\frac{1}{\sigma^2} + \sum_{j}^{N} \frac{W_{ij}^2}{\sigma^4} \frac{\mathrm{e}^{-b_j - \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}}}{\left(\mathrm{e}^{-b_j - \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}} + 1\right)^2} \\
&= -\frac{1}{\sigma^2} + \frac{1}{\sigma^4} \sum_{j}^{N} W_{ij}^2 \frac{\mathrm{e}^{b_j + \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}}}{\left(1 + \mathrm{e}^{b_j + \left[\frac{\boldsymbol{x}}{\sigma^2}\right]^T W_{*j}}\right)^2}.
\end{aligned}$$

The local energy can then be calculated, $E_L = K + P$, where the potential, P, depends on the system. We perform our calculations in an isotropic harmonic oscillator trap with an angular frequency ω and, if the particles are interacting, a Coulomb interacting potential between the electrons, such that the local energy becomes

$$E_L = \frac{1}{2}\sum_i^M \left(-\left[\frac{\partial \ln \Psi_T}{\partial x_i}\right]^2 - \frac{\partial^2 \ln \Psi_T}{\partial x_i^2} + \omega x_i^2 \right) + \sum_{k<l} \frac{1}{r_{kl}}, \tag{2.12}$$

where r_{kl} is the Euclidian distance between particle k and j in configuration space.

2.7.2 Gradients needed for gradient descent-based optimizer.

We also need the gradients with respect to the bias of the visible layer, $\boldsymbol{a}$, to be able to use a Gradient Descent (GD) approach in the update of the parameters. We need to find the gradient with respect to the expectation value of the energy yielded by the Monte Carlo method applied. The gradient of the expectation value of the energy with respect to an arbitrary parameter $\boldsymbol{\lambda}$ is derived in (Haug et al., 2022) and is

$$\boldsymbol{\nabla}_{\boldsymbol{\lambda}} \langle E \rangle = 2(\langle E \boldsymbol{\nabla}_{\boldsymbol{\lambda}} \ln \Psi_T \rangle - \langle E \rangle \langle \boldsymbol{\nabla}_{\boldsymbol{\lambda}} \ln \Psi_T \rangle). \tag{2.13}$$

To evaluate the gradient of the expectation value of the energy, we need the gradient with respect to the trainable parameter. We therefore need the gradient with respect to the bias of the visible layer, the bias of the hidden layer and the interactions between these layers, the kernel weights.

The gradient of the trial wave function with respect to the bias of the visible layer, $\boldsymbol{a}$, is

$$\boldsymbol{\nabla}_{\boldsymbol{a}} \ln \Psi_T = \frac{\|\boldsymbol{x} - \boldsymbol{a}\|}{\sigma^2}, \tag{2.14}$$

and for the hidden layer bias it is

$$\boldsymbol{\nabla}_{\boldsymbol{b}} \ln \Psi_T = \frac{\mathrm{e}^{\boldsymbol{b} + \frac{1}{\sigma^2}\boldsymbol{x}^T W}}{\left(1 + \mathrm{e}^{\boldsymbol{b} + \frac{1}{\sigma^2}\boldsymbol{x}^T W}\right)} \tag{2.15}$$

$$= \frac{1}{\left(\mathrm{e}^{-\boldsymbol{b} - \frac{1}{\sigma^2}\boldsymbol{x}^T W} + 1\right)}. \tag{2.16}$$

For the kernel weights we find the gradients of the trial wave function to be

$$\boldsymbol{\nabla}_W \ln \Psi_T = \frac{\boldsymbol{x}}{\sigma^2\left(\mathrm{e}^{-\boldsymbol{b} - \frac{1}{\sigma^2}\boldsymbol{x}^T W} + 1\right)}, \tag{2.17}$$

which is N gradients of length M. The expectation value of the gradients along with the expectation value of the gradients times the local energy is used to find the gradient of the expectation value of the energy, as shown in Equation 2.13.

2.8 Markov Chain Monte Carlo

Our previous project, (Haug et al., 2022), contains informations about the theory behind our MCMC approaches. We have two approaches, the Random Walk Metropolis (RWM) and the Langevin Metropolis-Hastings (LMH).

3 Methodology

3.1 Previous work

This project is a continuation of our past work (Haug et al., 2022) and will along with the theory presented above also include the methods presented in this section.

3.2 Variational Monte Carlo

Theory and description for both our Monte Carlo approaches, brute-force and importance sampling, can be accessed through our previous work (Haug et al., 2022). The variational part of this project, i.e approximating the upper bound for the ground state energy of our system, will be based on our previous implementation. In short, our results showed that the importance sampling implementation yielded the most accurate sampling results. The importance sampling approach requires relatively few numbers of MC cycles, M, for it to yield sufficient results, even for more complex systems than studied here.

3.3 Gradient-based optimization

Our gradient descent approach can also be found in our previous work (Haug et al., 2022). In principle, we have as many datapoints as we want to generate. We will therefore use a Stochastic Gradient Descent (SGD) approach to find the upper limit of ground state energy. Although a basic SGD could provide us with decent results, we have implemented a standard gradient descent and ADAM, and the ADAM optimizer is what we utilize in the our optimization runs.

Unlike previously, we will treat the local energy, E_L as the cost and differentiate with regard to the variational parameters, $\boldsymbol{\lambda}$. In order to obtain the wave function that is an eigenstate of the ground state energy, we will optimize the $\boldsymbol{\lambda}$ value containing the RBM parameters. In general, the premise is the same as in our previous work, except that we are working with multiple parameters, being the biases of the visible and hidden layers, and the weights of the interactions between them, of the RBM.

The gradient of the loss function, in our case the expectation value of the energy, with respect to an arbitrary parameter, $\boldsymbol{\lambda}$, of the RBM is given by

$$\boldsymbol{\nabla}_{\boldsymbol{\lambda}} \langle E \rangle = \boldsymbol{\nabla}_{\boldsymbol{\lambda}} \langle E(\boldsymbol{\lambda}) \rangle$$

which is found via Equation 2.13. For every step, displaying the standard gradient descent method, we update the value of $\boldsymbol{\lambda}_k$ by subtracting by the expectation value of the energy which then is multiplied by a learning rate, η in the following way,

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k - \eta \boldsymbol{\nabla}_{\boldsymbol{\lambda}_k} \langle E(\boldsymbol{\lambda}_k \rangle \qquad \text{for } k \geq 0,$$

where $\boldsymbol{\lambda}_k$ and $\boldsymbol{\lambda}_{k+1}$ represent the kth and $(k+1)$th values of the trainable parameters, respectively. This procedure is repeated until we get convergence (minimization in the cost function) for all trainable parameters.

3.4 ADAM

In addition of keeping track of the running average of the first momentum, the ADAM (ADAptive Momentestimation) algorithm also does the same for the second moment of the gradient. Complete description of this method can be found in our previous work (Haug et al., 2022).

3.5 Blocking

In order to find the sampling error in our energy estimations, blocking will be used. The complete outline and mathematical background of the blocking method used in our implementation, can be found in our previous work (Haug et al., 2022) and is based on Jonnson's work (Jonsson, 2018).

3.6 Parameters and General Outline

We have in this project looked at two MCMC methods. The scale parameter Random Walk Metropolis (RWM) and the Langevin Metropolis-Hastings (LMH) algorithm where the former is a brute-force implementation and the latter uses importance sampling. Complete outline of these methods can be found in our previous work (Haug et al., 2022). For the single electron in one-dimension we found a suitable scale parameter for the RWM algorithm used in generating proposal states to be 3.0, which yielded an acceptance ratio of $\sim 30\%$, while for the LMH algorithm we found the 'time-step', scale$= \sqrt{dt}$, to be 1.3 with an acceptance ratio of $\sim 60\%$. For the system of two interacting particles we used the scale parameters scale$= 1.0$ for the RWM algorithm and $scale = 1.0$ for the LMH algorithm which yielded, respectively, the acceptance ratios of $\sim 30\%$ and $\sim 60\%$.

The parameters of the RBM that we will train are the biases of the visible and hidden layers, as well as the weights describing the strength of the connections between the visible and hidden neurons. The biases are initialized according to $\mathrm{N}(\mu = 0, \sigma = 0.001)$ and the weight matrix, to combat vanishing and exploding gradients, is initialized according to $\mathrm{N}\Big(\mu = 0, \sigma = \sqrt{1/M}\Big)$, with M being the number of visible neurons.

3.7 Overview of the Implementation

We have implemented the discussed formalism in a Python package with a high-level API that can be found here https://github.com/nicolossus/FYS4411-Project2. The closed-form expressions that are involved in the cost function and the sampling algorithms are written in a highly vectorized manner by using functionality offered by NumPy (Harris et al., 2020). Furthermore, we have included the possibility of using an automatic differentiation routine by instead using a JAX (Bradbury et al., 2018) backend. The RBM. $F_{\mathrm{RBM}}(\boldsymbol{x})$ can be chosen to represent the wave function either as $\psi(\boldsymbol{x}) = F_{\mathrm{RBM}}(\boldsymbol{x})$ or $|\psi(\boldsymbol{x})|^2 = F_{\mathrm{RBM}}(\boldsymbol{x}) \implies \psi(\boldsymbol{x}) = \sqrt{F_{\mathrm{RBM}}(\boldsymbol{x})}$. Note that for complex valued wave functions, the only representation that can be used is $\psi(\boldsymbol{x}) = F_{\mathrm{RBM}}$. Even though this flexibility in representation is incorporated into our package, we will only use the $\psi(\boldsymbol{x}) = F_{\mathrm{RBM}}$ in this study.

```python
import nqs

# Config
nparticles = 1     # number of particles
dim = 1            # dimensionality
```

```
nhidden = 2        # hidden neurons
scale = 3.0        # Scale of proposal distribution
sigma2 = 1.0       # Variance of Gaussian layer in the RBM

# Instantiate the model
system = nqs.NQS(nparticles,
                 dim,
                 nhidden=nhidden,
                 interaction={False, True},  # Repulsion or not
                 mcmc_alg={'rwm', 'lmh'},    # MCMC algorithm
                 nqs_repr={'psi', 'psi2'},   # RBM representation
                 backend={'numpy', 'jax'}    # Closed-form or AD
                 log=True,                   # Show logger?
                 logger_level="INFO",        # Logging level (see docs)
                 rng=None                    # Set RNG engine (optional)
                 )

# Initialize parameters; biases and weight matrix are set automatically
system.init(sigma2=sigma2, scale=scale)

# Train the model
system.train(max_iter=500_000,                # No. of training iterations
             batch_size=1_000,                # No. samples used in one update
             gradient_method={'gd', 'adam'},  # Optimization algorithm
             eta=0.05,                        # Learning rate
             beta1=0.9,                       # ADAM hyperparameter
             beta2=0.999,                     # ADAM hyperparameter
             epsilon=1e-8,                    # ADAM hyperparameter
             mcmc_alg=None,                   # Set MCMC algo. (optional)
             seed=None                        # Set for reproducibility
             )

# Sample variational energy
df = system.sample(int(2**18),                # No. of energy samples
                   nchains=4,                 # No. of Markov chains
                   mcmc_alg=None,             # Set MCMC algo. (optional)
                   seed=None                  # Set for reproducibility
                   )

# Results are returned in a pandas.DataFrame. Save directly with
system.to_csv('filename.csv')
```

Listing 1: *Example usage of the NQS framework.*

4 Results and Discussion

4.1 Quantum Dot

We start with the simplest system of a single one-dimensional quantum dot, i.e., a system without repulsive interactions, in order to both validate the implementation and compare different settings for the training of the RBM. As mentioned in Section 2.1, the exact ground state energy for this system is 1/2 a.u. Throughout this study, the common variance of the Gaussian-binary RBM's visible layer is set to unity, the number of variational energy samples are held fixed at 2^{18} samples and in gradient descent optimization we use the ADAM algorithm.

The Effect of the Learning Rate

In the following we use a RBM with 2 hidden neurons. Figure 4.1 shows the effect of the learning rate η with different numbers of training iterations. The training, or update of parameters, are done in batches of 1000 iterations, meaning that the x-axis tick labels correspond to the number of updates during training. Each point is the average of 8 Markov chains, each with expectation value of the energy, $\langle E \rangle$, the sampling error, σ_b, found via the blocking method, and the variance $\mathrm{Var}(E)$. In the figure, we show both results obtained by using RWM and LMH sampling algorithms which are color coded as blue and orange, respectively. Although not visible in most plots, the shaded areas around the points are the standard error of the means computed across the Markov chains. From the figure we see that $\eta = 0.5$ is the fastest to converge, in a descending manner, to minimum values for all quantities, i.e., $\langle E \rangle$, σ_b and $\mathrm{Var}(E)$, with both the RWM and LMH sampler. Keeping in mind the different scales in the plots, the smaller learning rates also are able to achieve accurate minimum values of the quantities, although with a detour toward ascending estimates. The take-away message is that the gradient descent optimization of the RBM parameters needs a relatively large number of training iterations, or MCMC cycles, in order to converge, and the speed of the convergence depend on the learning rate. Here, we have only carried out an extremely coarse search for the optimal learning rate, and a different learning rate might have faster convergence properties for this particular system.

Figure 4.2 is similar to the preceding figure, only here the update of parameters are done in batches of 5 000 training iterations. The rationale behind using a larger batch size is that the expectation values are then estimated from a larger sample size, thus increasing the accuracy of the estimate. With the same number of total training iterations as in Figure 4.1 for each point, the increased batch size means that the number of updates during training is reduced correspondingly. We again see the fastest convergence with $\eta = 0.5$, and that neither of smaller learning rates seem to have converged in terms of minimum values across all quantities, $\langle E \rangle$, σ_b and $\mathrm{Var}(E)$. As such, this result indicates that an increase in batch size for the same number of total training iterations, will make the convergence properties of the optimization procedure more dependent on using an optimal learning rate. However, one should be wary about using to small batch sizes, as then the estimates of the expectation values will be less accurate. Hence, there is a trade-off between estimation accuracy and computational cost we need to take into account when designing the optimization procedure.

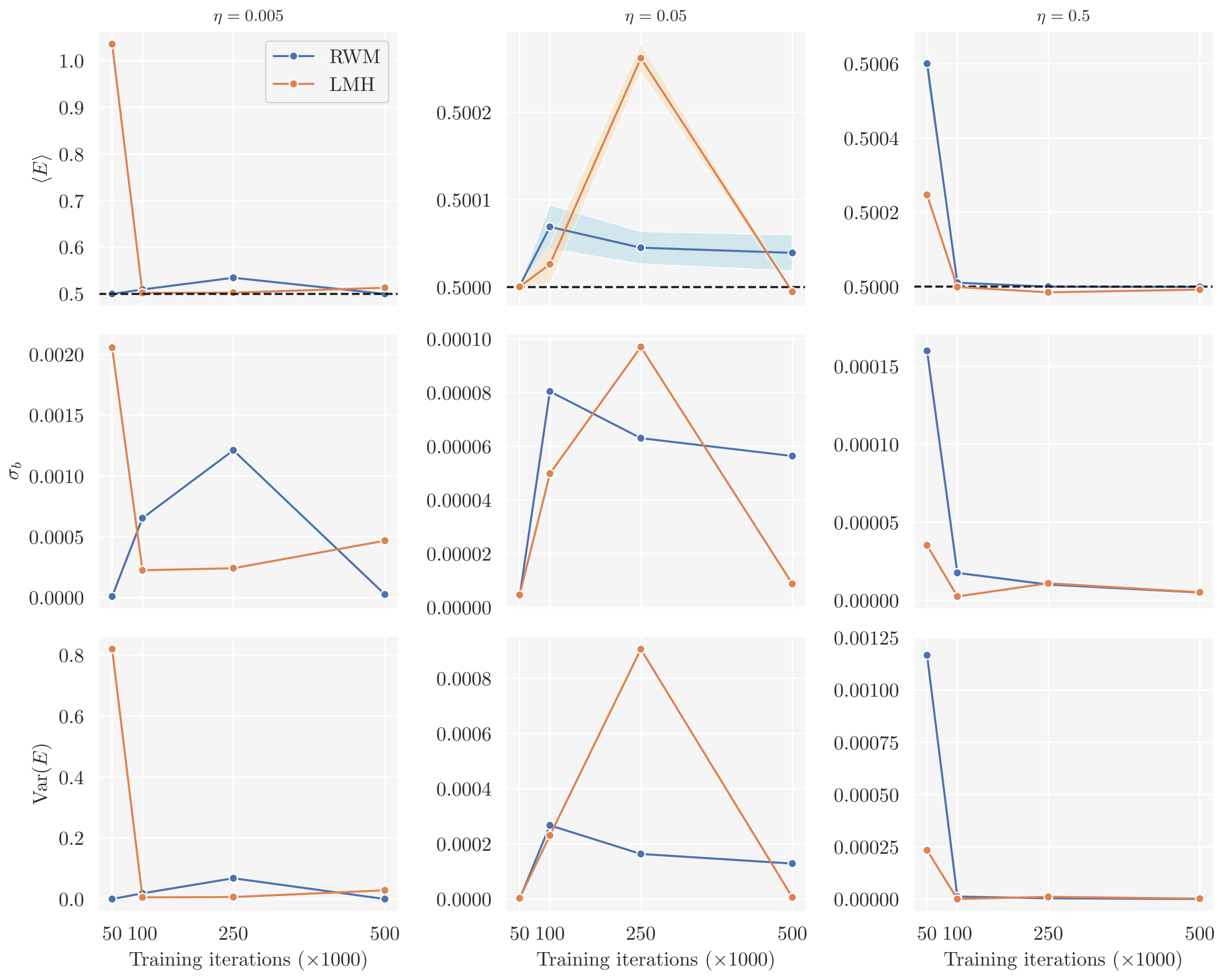

Figure 4.1: *The effect of the learning rate η, with the different values stated in each column's title, and the number of training iteration. Each point is the mean across 8 Markov chains, and the shaded areas (generally not visible) the standard error of the mean. The figure shows the expectation value of the energy, $\langle E \rangle$, the sampling error, σ_b, found via the blocking method, and the variance $\text{Var}(E)$. The blue lines and regions display the results from the RWM algorithm, while the orange lines show the corresponding results from the LMH algorithm. The training, or update of parameters, are done in batches of 1000 iterations, meaning that the x-axis tick labels correspond to the number of updates during training, or epochs.*

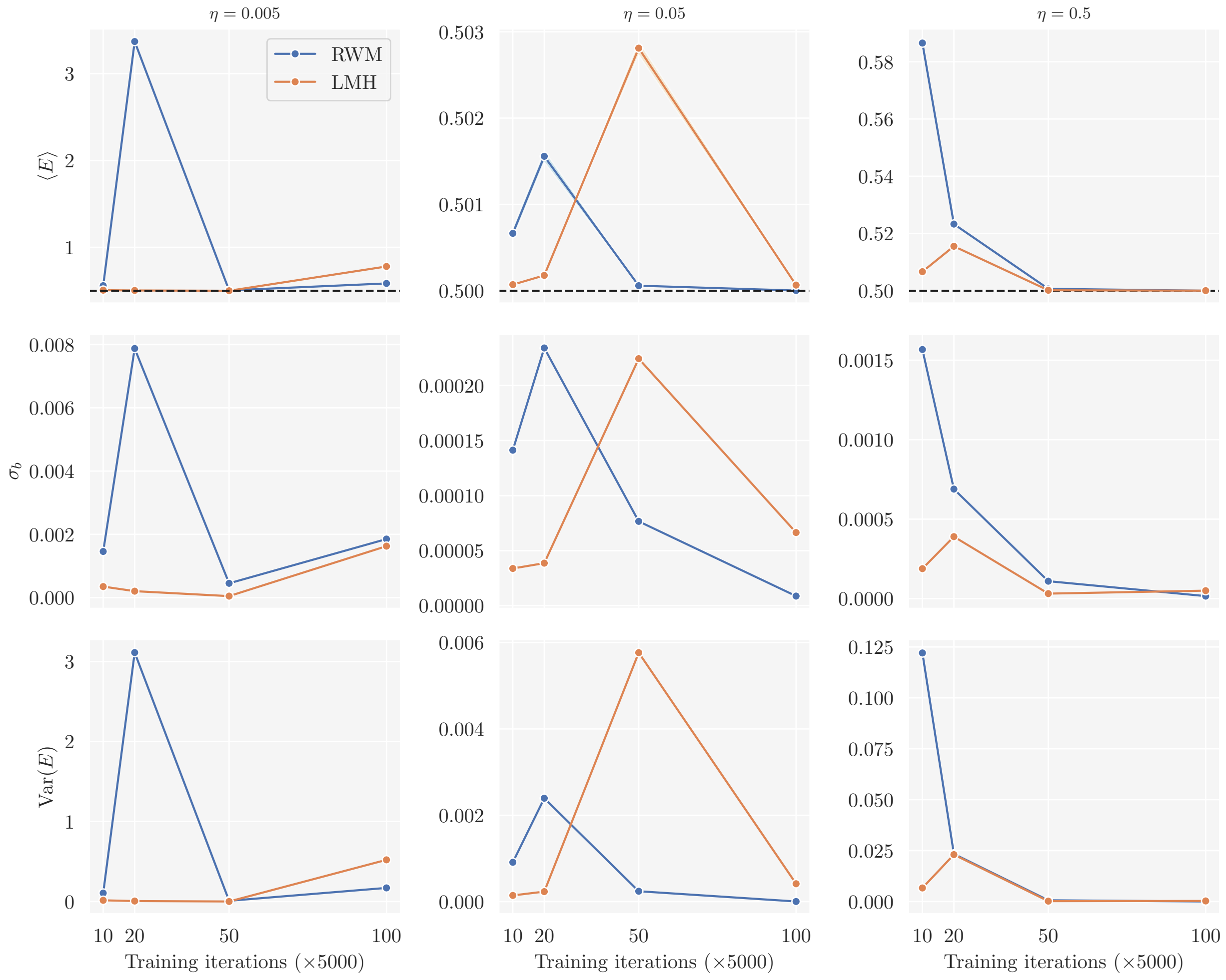

Figure 4.2: *The effect of the learning rate η and the number of training epochs, here with batches of 5 000 samples. See the description in Figure 4.1 for more details.*

The Number of Hidden Neurons

Figure 4.3 displays comparisons of RMBs with different number of hidden neurons, $N_{\text{hidden}} = \{1, 2, 3, 4\}$, with their calculated energies (upper plots) where the left one is calculated with a batch size of $1,000$ cycles, and the right calculated with a batch size of $5,000$ cycles. The bottom plots shows the corresponding variance in the energy. The optimal number of hidden neurons for the single one-dimensional quantum dot is 2, where the lowest error for both batch-sizes is. We find that the best approximation occurs when the batch-size is $1,000$ cycles and using the RWM sampling algorithm. We find the best approximation to the value of the ground state energy to be $E_0 = 0.499999 \pm 3 \cdot 10^{-6}$ a.u, which is equal to the analytical ground state energy $E_0 = 0.5$ a.u, with a precision of order 10^{-6}.

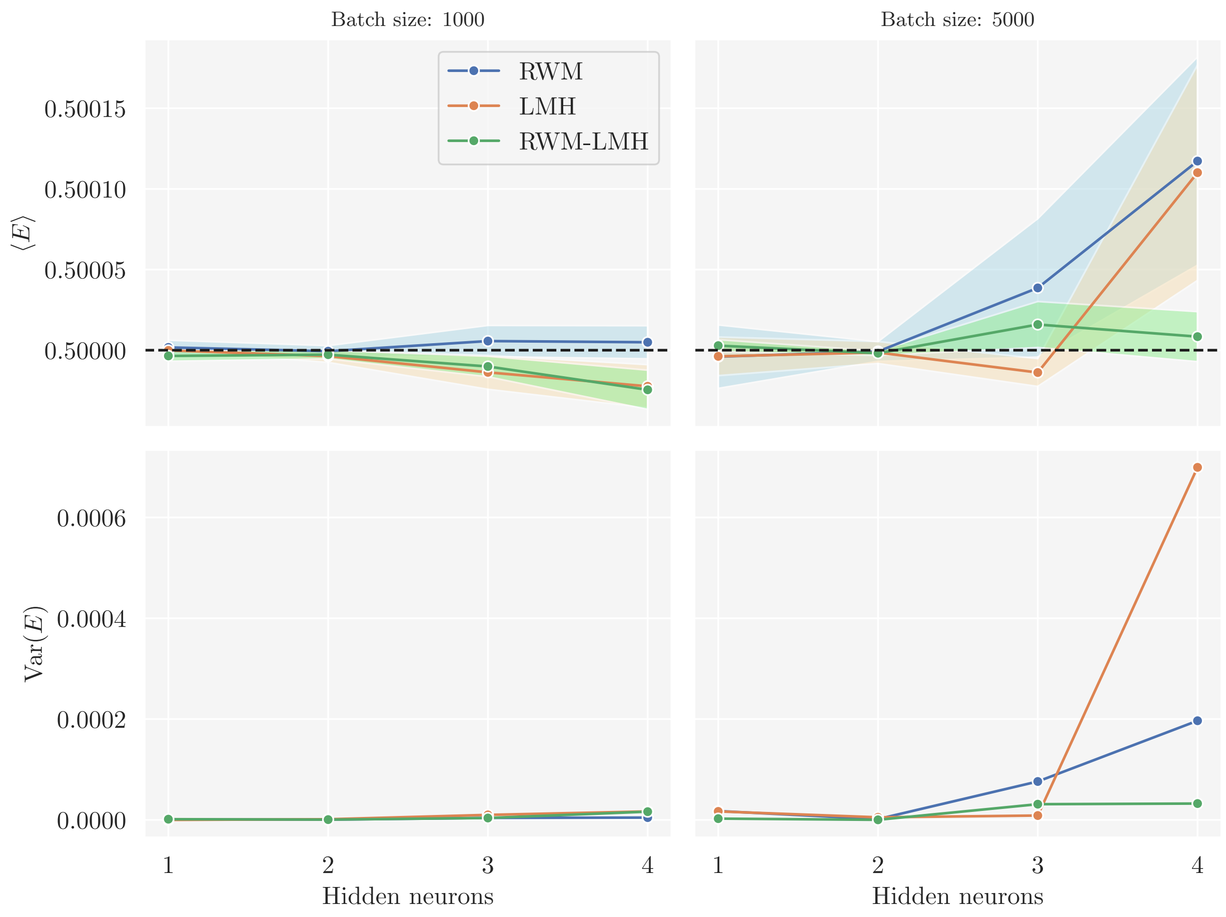

Figure 4.3: *Calculated approximation of the ground state energy of the quantum dot in the one-dimensional harmonic oscillator trap against the number of neurons in the hidden layer, for batch-size* $1,000$ *(left) and* $5,000$ *right, with their corresponding variances displayed below.*

Comparing Closed-Form Gradients and Automatic Differentiation

Figure 4.4 compares the runtimes of as a function of the number of training iterations for the RWM and LMH algorithms, where calculations are performed using closed-form (CF) expressions and an automatic differentiation (AD) implementation using JAX (Bradbury et al., 2018). The difference in the implementations using the CF expressions are small, but the RWM algorithm is, as expected, due to fewer calculations in the proposal steps and acceptance probability, a little faster. When comparing with the implementations using AD, we observe that the RWM algorithm is approximately a factor of 2 faster on closed-form, and the LMH algorithm is approximately a factor of 3 faster on closed-form.

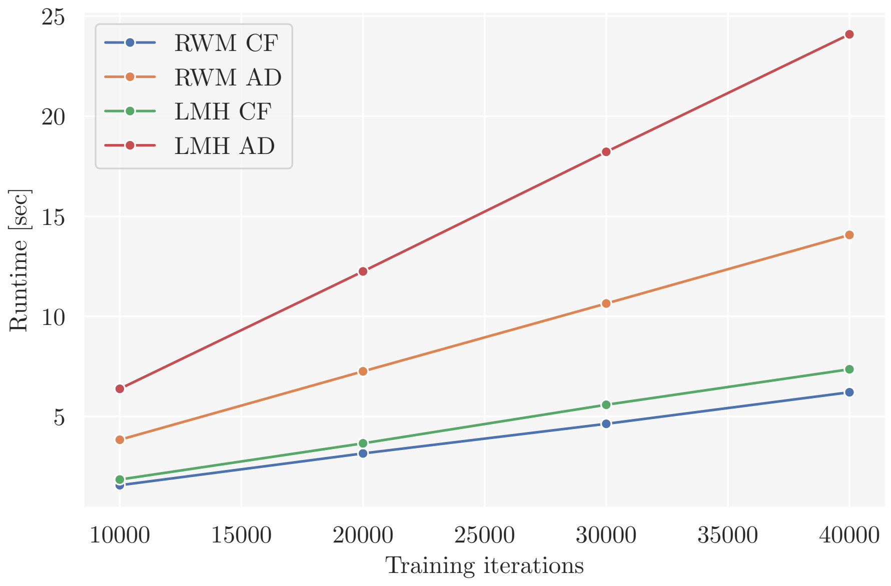

Figure 4.4: *Runtimes against training iterations for four different sampling approaches on the same system. The closed-form (CF) expressions are utilized for calculating the next step using the RWM and LMH sampling algorithms for the lines labeled RWM CF and LMH CF, while the needed gradients are found through automatic differentiation (AD) for the same algorithms for the lines labeled RWM AD and LMH AD.*

4.2 Interacting Quantum Dots

We now move on to the more interesting case of a system of a couple of two-dimensional electrons with repulsive interactions.

Figure 4.5 displays the computed expectation values of the ground state energy against the number of hidden neurons for the two-dimensional system of two interacting quantum dots, for three different learning rates, $\eta \in \{0.01, 0.05, 0.5\}$ using the RWM sampling algorithm. The optimal values for the learning rate and the number of hidden nodes are $\eta^* = 0.05$ and with 6 hidden neurons. For these hyper parameters the calculated ground state energy is found to be $E_0 = 3.059 \pm 0.008$ a.u.

Figure 4.6 shows a comparisons between in the calculated expectation values for the ground state energy of the RWM and LMH sampling algorithms against the same number of hidden neurons as in Figure 4.5, with a learning rate of $\eta = 0.05$. The best approximation to the ground state using the RWM algorithm is $E_0 = 3.059 \pm 0.008$ a.u, while for the LMH algorithm the best approximation occurs when there are 2 hidden neurons and yields $E_0 = 3.089 \pm 0.004$ a.u. The figure shows that the LMH algorithm performs more evenly for different number of hidden neurons, but that RWM has better approximations when the number of hidden neurons are $2, 3, 4, 6$ and 7.

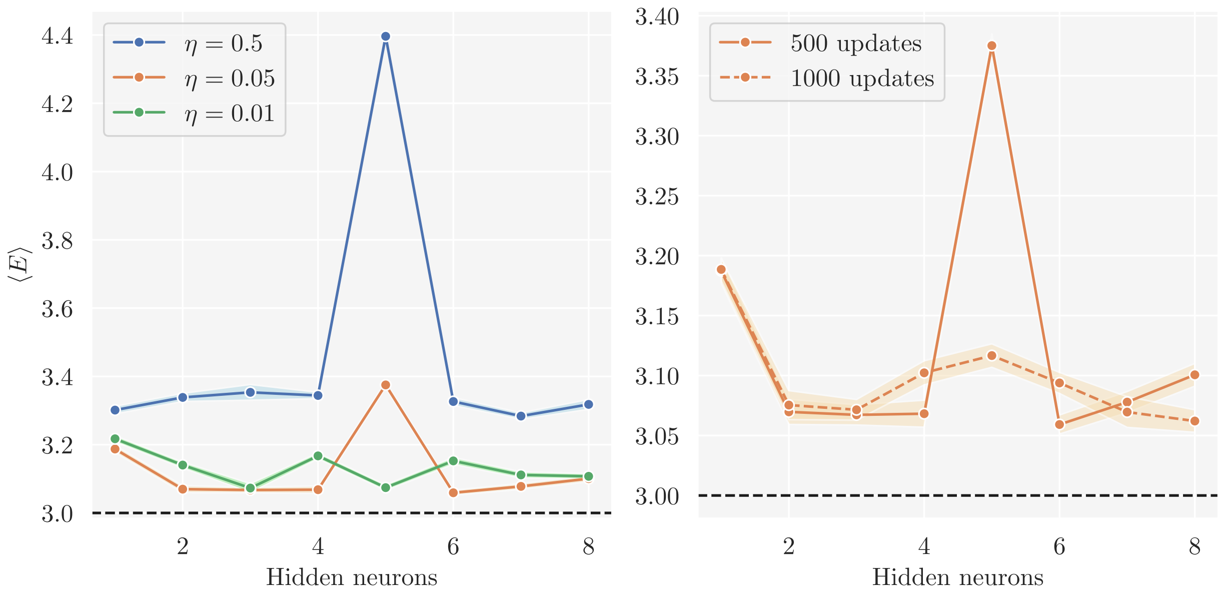

Figure 4.5: *The effect of the number of neurons in the hidden layer on the approximated upper bound on the ground state energy is plotted for three different learning rates (left plot), while the right plot displays the approximation with a learning rate of* 0.05 *after* 500*(solid line) and* 1000*(dashed line) updates in the training. The RWM sampling algorithm was used for all points.*

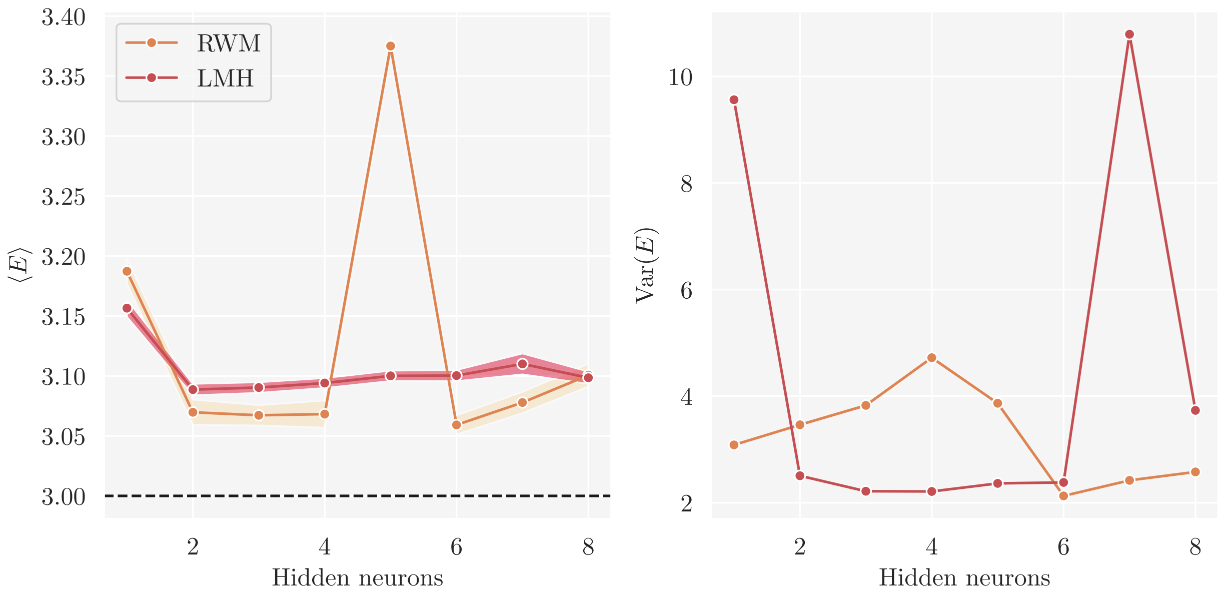

Figure 4.6: *Sampled upper bound approximation to the ground state energy as a function of neurons in the hidden layer for both the RWM and LMH sampling algorithms, with a learning rate of* 0.05 *after* 500 *updates in the parameters of the RBM.*

5 Conclusion

Inspired by Carleo and Troyer, 2017, we have implemented a formalism using the unsupervised machine learning method *restricted Boltzmann machine* (RBM) to study interacting many-particle systems. In particular, we have have used Gaussian-binary RBMs to model the wave function. In order to learn the parameters of the model, we have used a variational

Monte Carlo (VMC) approach together with gradient descent optimization. For the VMC approach we consider the two Markov chain Monte Carlo (MCMC) sampling algorithms *random walk Metropolis* , which is more of a brute-force sampling routine, and the *Langevin Metropolis-Hastings*, which uses importance sampling. For gradient descent we have used the ADAM optimizer. We were able to accurately estimate the ground state energy for the simple system with a single one-dimensional particle, and found that, in this case, the RBM was more lenient when it came which settings produced accurate results. We found the best approximation to the ground state energy to be $E_0 = 0.499999 \pm 3 \cdot 10^{-6}$ a.u with ground truth $E_0 = 0.5$ a.u, using he RWM sampling algorithm. For the more complicated case with a couple of two-dimensional particles with repulsive interaction, finding optimal settings was more difficult. Here, we were able to obtain an energy estimate within one decimal point accuracy, $E_0 = 3.059 \pm 0.008$ a.u, of the ground truth, $E_0 = 3.0$ a.u, for specific settings, but others yielded estimates with significantly larger discrepancy. Our cost function is based solely on minimization of the expectation value of the energy, $\langle E \rangle$, and it might be beneficial to construct a cost function based on minimizing the variance, $\mathrm{Var}(E)$, or on minimizing both $\langle E \rangle$ and $\mathrm{Var}(E)$.

6 Future Work

It is clear that machine learning approaches have proven themselves to be prominent in computational quantum mechanics and they will undoubtedly keep getting more and more popular in the future. The first step forward that comes to mind would be to analyze more complex systems with more than two particles. But as only two particles can occupy the same state, as we know from the Pauli exclusion principle, we would need to make changes to our implementation. The Gaussian energy function, $E(\mathbf{x}, \mathbf{h})$ is symmetric, but for dimensions of particles greater than two, we need to ensure that the total wave function is anti-symmetric when factoring in spin. Using the theory of Slater determinants together with the Gaussian energy function, could bring a solution to this. Therefore an extension to a general system of fermions would require more complications, and may be a good extension of our NQS trial wave function implementation. But to accomplish this, it would be a good idea to implement deep RBMs as they have the ability to model more complex systems. Or we could all-together drop using RBMs and test with (Deep) Neural Networks and analyze the differences. Besides that, experimenting with other sampling techniques like Gibbs sampling can also be of great interest. With more complex methods comes also a greater price in computational cost.

In our NQS framework we included procedures for automatic differentiation via JAX. We have also implemented a RBM model using the JAX-based machine learning framework Flax, see `https://github.com/nicolossus/FYS4411-Project2/blob/main/nqs/models/rbm_flax.py`, which can be used as basis for a flexible (deep) RBM framework in the future.

References

Saito, Hiroki (July 2018). “Method to Solve Quantum Few-Body Problems with Artificial Neural Networks”. In: *Journal of the Physical Society of Japan* 87.7, p. 074002. DOI: 10.7566/jpsj.87.074002. URL: https://doi.org/10.7566%2Fjpsj.87.074002 (cit. on p.).

Carleo, Giuseppe and Matthias Troyer (Feb. 2017). “Solving the quantum many-body problem with artificial neural networks”. In: *Science* 355.6325, pp. 602–606. DOI: 10.1126/science.aag2302. URL: https://doi.org/10.1126%2Fscience.aag2302 (cit. on pp. 1, 5, 17).

Taut, M. (Nov. 1993). “Two electrons in an external oscillator potential: Particular analytic solutions of a Coulomb correlation problem”. In: *Phys. Rev. A* 48 (5), pp. 3561–3566. DOI: 10.1103/PhysRevA.48.3561. URL: https://link.aps.org/doi/10.1103/PhysRevA.48.3561 (cit. on p. 2).

Haug, N., J.E. Betten, and A. Davidov (2022). “A Variational Monte Carlo Analysis on Bose-Einstein Condensation in a Trapped Bose Gas.” In: (cit. on pp. 1, 3, 7–10).

Jonsson, Marius (2018). “Standard estimation by an automated blocking method”. In: *Physical Review* (cit. on p. 10).

Harris, Charles R. et al. (2020). “Array programming with NumPy”. In: *Nature* 585.7825, pp. 357–362 (cit. on p. 10).

Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. URL: http://github.com/google/jax (cit. on pp. 10, 15).