Project 2 - Restricted Boltzmann Machines applied to the Quantum many-body Problem

FYS4411 - Computational Physics II

Hannes Kneiding
Marianne Bjerke

May 31, 2019

Link to GitHub - Repository

Abstract

A Restricted Boltzmann Machine (RBM) approach has been developed to determine ground state energies of confined Bose systems with different numbers of particles and dimensions. Brute-force and importance Metropolis-Hastings sampling have been implemented as well as Gibbs sampling. A non-interacting as well as an interacting system have been examined. Optimization of network parameters has been done using the stochastic gradient descent method. The results have been evaluated using the statistical method of blocking.
For the non-interacting system the analytical results could be reproduced with almost exact precision using analytic expressions for gradients, Laplacians and parameter derivatives. Importance as well as Gibbs sampling proved to be more efficient than simple brute-force sampling as it was expected. Furthermore, the effect of the different RBM parameters (learning rate, number of hidden nodes, variance) on the convergence behaviour of the simulation has been investigated.
The obtained results (ground state energies) of the interacting system (two particles in two dimensions) were in accordance with the analytical result shown in [1], but showed non-negligible deviations. Already the first decimal place was off compared to the analyitcal values.

Contents

1 Introduction

A recent publication on solutions of the quantum many-body problem with Markvo-Chain-Monte-Carlo (MCMC) simulations proposed a novel approach using the theoretical and practical advancements in machine learning. Instead of taking a standard variational approach to the optimization of the wave function, so called Restricted Boltzmann Machines (RBM) have been used in order to represent the wave function. These wave function networks are called Neural Network Quantum State (NQS). The results for quantum mechanical spin lattice systems obtained with this method were very promising and indicate a general applicability of this method for quantum mechanical studies with MCMC methods. [2, 3]

In this project the proposed method was applied to a system of two interacting electrons confined in a harmonic potential. Conclusions about the applicability to quantum state many-body problems have been drawn. Therefore, the non-interacting as well as the interacting case have been investigated. Aside from standard Metropolis sampling also Gibbs sampling has been implemented which helps in sampling from a multivariate probability distribution. For the statistical analysis and the determination of error estimates the method of blocking has been used.

1.1 Problem definition

The Hamiltonian is given by

$$\hat{H} = \hat{H}_0 + \hat{H}_1 \tag{1}$$

where $\hat{H}_0$ is the non-interacting part of the Hamiltonian

$$\hat{H}_0 = \sum_i^N \left(-\frac{1}{2}\bigtriangledown_i^2 + \frac{1}{2}\omega^2 r_i^2 \right) \tag{2}$$

and $\hat{H}_1$ is the interacting part of the Hamiltonian

$$\hat{H}_1 = \sum_{i<j}^N \frac{1}{r_{ij}} \tag{3}$$

with natural units $\hbar = e = c = m_e = 1$.

The trial wave function for the ground state with N atoms is given by

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

where X_i are the visible nodes, a_i the visible biases, b_i the hidden biases and w_{ij} the weights characterizing the connection between node i and j (between different layers).

1.2 Benchmarks

The exact, analytical energies are known for the non-interacting case and are given by $E_L = \frac{1}{2} P \cdot D$ (assuming natural units) where P and D denote the number of particles and dimensions respectively. Analytical solutions for the interacting case are generally not known, however for the special case of two electrons in two dimensions the exact, analytical energy has been shown to be 3. [1]

2 Methods

This project employs a Monte-Carlo approach in order to determine the ground state energies of different bosonic systems. Monte-Carlo simulations for quantum mechanical systems operate on the basis of the variational principle:

Given a trial wave function Ψ_T and a Hamiltonian $\hat{H}$, then the expectation value $\langle \hat{H} \rangle$ defined as

$$E[\hat{H}] = \langle \hat{H} \rangle = \frac{\int d\mathbf{R} \Psi_T^*(\mathbf{R}, \boldsymbol{\alpha}) \hat{H}(\mathbf{R}) \Psi_T(\mathbf{R}, \boldsymbol{\alpha})}{\int d\mathbf{R} \Psi_T^*(\mathbf{R}, \boldsymbol{\alpha}) \Psi_T(\mathbf{R}, \boldsymbol{\alpha})} \tag{5}$$

with variational parameters $\boldsymbol{\alpha}$, constitutes an upper bound to the exact ground state energy [4]:

$$E_0 \leq E[\hat{H}] \tag{6}$$

By varying the variational parameters $\boldsymbol{\alpha}$ and thereby obtaining various trial wave functions, approximations to the exact ground state energy can be made when analytical solutions to the ground state energy are not available.
However, the dimensionality of the integrals involved, directly scales with the number of particles and spatial dimensions, making the analytic calculation of the expectation value $\langle \hat{H} \rangle$ unfeasible for realistic systems.

2.1 Monte Carlo integration

Relief to this issue can be achieved by applying Monte Carlo integration for the evaluation of the integrals. Monte Carlo integration is based on the idea that an integral is nothing more than the area contained between the function itself and the axes or some kind of boundary. Therefore, an approximation to the exact integral can be obtained by randomly picking a series of points within the range of interest and then averaging over the corresponding function values. [5]
Let $f(\mathbf{x})$ be a function then the corresponding integral on the surface $\boldsymbol{\Omega}$ is defined as

$$I = \int_{\boldsymbol{\Omega}} f(\mathbf{x}) \, d\mathbf{x} \tag{7}$$

which can be approximated as

$$I \approx \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) \tag{8}$$

where samples are uniformly drawn from $\boldsymbol{\Omega}$

$$\mathbf{x}_i \in \boldsymbol{\Omega} \tag{9}$$

The accuracy of the approximation obviously scales with the number of random points sampled (N). In the limit $N \to \infty$ the approximation becomes exact. In practice however, the number of samples needed to obtain a good approximation is unfeasibly high since all samples are generated from a uniform distribution and "important" areas may not get sampled enough.

2.2 Markov chains and Monte-Carlo Sampling

A more efficient approach is the employment of Markov-Chain-Monte-Carlo (MCMC) simulations for the sampling process. Markov chains are sequences of states, where each state $\mathbf{x}_i$ is only dependent on its previous state $\mathbf{x}_{i-1}$. In Monte Carlo simulations, Markov chains are generated by proposing a new state $\mathbf{x}'$ following some proposal rule and then deciding whether to accept or reject the trial state. [5] [6] If accepted, the proposed trial state $\mathbf{x}'$ is appended to the Markov chain; if rejected, the current state $\mathbf{x}_i$ will be appended to the Markov chain, thus building a chain of states. [5]

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ ... \ \mathbf{x}_{n-1}, \mathbf{x}_n\} \tag{10}$$

Each state is sampled from a probability distribution (PDF) $p(\mathbf{x}_{i+1}|\mathbf{x}_i)$ on the basis of the previous state. This PDF defines the transition probability from state $\mathbf{x}_i$) to state $\mathbf{x}_{i+1}$) and should be easy to sample from. MCMC thereby allows us to sample from a model instead of the original distribution from which direct sampling is difficult. [5, 6]

The basic MCMC process looks as follows:

1. Randomly initialize starting state x_0.
2. Repeat until convergence criterion is satisfied:
   (a) Propose trial state x' according to a specified proposal rule.
   (b) Accept or reject trial state according to a specified acceptance scheme.

2.2.1 Metropolis algorithm

By defining the local energy

$$E_L(\mathbf{R}) = \frac{1}{\Psi_T(\mathbf{R})}\hat{H}\Psi_T(\mathbf{R}), \tag{11}$$

and the PDF

$$P(\mathbf{R}) = \frac{|\Psi_T(\mathbf{R})|^2}{\int |\Psi_T(\mathbf{R})|^2 d\mathbf{R}} \tag{12}$$

the integral to be evaluated (equation 5) can be rewritten to

$$\langle E_L \rangle = \int P(\mathbf{R}) E_L(\mathbf{R}) d\mathbf{R} \tag{13}$$

This integral can now be approximated by Monte-Carlo integration

$$\langle E_L \rangle \approx \frac{1}{N} \sum_{i=1}^{N} P(\mathbf{R}_i) E_L(\mathbf{R}_i) \tag{14}$$

In order to obtain an expectation value for the local energy we need to sample states from P and accumulate the local energy values. For the MCMC sampling process an acceptance scheme is necessary. By splitting up the transition probability P into acceptance and proposal probability

$$p(\mathbf{x}_{i+1}|\mathbf{x}_i) = p_{acc}(\mathbf{x}_i \to \mathbf{x}_{i+1}) p_{prop}(\mathbf{x}_i \to \mathbf{x}_{i+1}) \tag{15}$$

the popular Metropolis acceptance scheme can be derived

$$p_{acc}(\mathbf{x}_i \to \mathbf{x}_j) = \min\left\{1,\ \frac{\mu(\mathbf{x}_j)p_{prop}(\mathbf{x}_j \to \mathbf{x}_i)}{\mu(\mathbf{x}_i)p_{prop}(\mathbf{x}_i \to \mathbf{x}_j)}\right\} \tag{16}$$

where $\mu(\mathbf{x}_i)$ is the probability of finding the system in state $\mathbf{x}_i$ and $p_{prop}(\mathbf{x}_i \to \mathbf{x}_j)$ is the probability of suggesting the specified trial position. [5, 6]
Assuming symmetric proposal probabilities

$$p_{prop}(\mathbf{x}_i \to \mathbf{x}_j) = p_{prop}(\mathbf{x}_j \to \mathbf{x}_i) \tag{17}$$

the acceptance scheme reduces to

$$p_{acc}(\mathbf{x}_i \to \mathbf{x}_j) = \min\left\{1,\ \frac{\mu(\mathbf{x}_j)}{\mu(\mathbf{x}_i)}\right\} \tag{18}$$

Inserting P into equation 18 yields the Metropolis acceptance scheme for this problem

$$p_{acc}(\mathbf{R}_i \to \mathbf{R}_j) = \min\left\{1,\ \frac{|\Psi_T(\mathbf{R_j})|^2}{|\Psi_T(\mathbf{R_i})|^2}\right\} \tag{19}$$

where the hard-to-compute integrals cancel out. As a proposal rule a uniform (and therefore symmetric) updating scheme has been chosen

$$\mathbf{R}' = \mathbf{R} + \gamma\mathbf{r} \tag{20}$$

where $\mathbf{r}$ is a vector of random numbers $\in [0, 1]$ according to the number of dimensions and γ is a chosen step size.

2.2.2 Importance sampling

The efficiency of Monte-Carlo sampling can be increased by replacing the brute-force Metropolis with a scheme where the random walk is biased by the trial wave function. [5] This approach is based on the Langevin and Fokker-Planck equations.
In one dimension the Langevin equation is given by

$$\frac{\partial x(t)}{\partial t} = DF(x(t)) + \eta \tag{21}$$

where D is the diffusion coefficient and η is a random variable. F denotes the drift vector defined as

$$F = 2\frac{1}{\Psi_T}\nabla\Psi_T \tag{22}$$

which can be derived from the Fokker-Planck equation. Integrating this differential equation with Euler's method yields an update equation for a trial position

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t} \tag{23}$$

where Δt is a chosen time step and ξ is a Gaussian random variable. The drift vector pushes the trial particles to regions where the wave function is large so that the proposal probabilities become asymmetric.

Therefore, they have to be considered in the Metropolis acceptance scheme. The solution to the Fokker-Planck equation yields the required proposal probabilities expressed in terms of Green's function

$$G(y, x, \Delta t) = \frac{1}{(4\pi D\Delta t)^{3N/2}} \exp(-(y - x - D\Delta t F(x))^2 / 4D\Delta t) \tag{24}$$

Therefore, the Metropolis choice is replaced by the Metropolis-Hastings article

$$p_{acc}(\mathbf{R}_i \to \mathbf{R}_j) = \min\left\{1,\ \frac{G(\mathbf{R}_i, \mathbf{R}_j, \Delta t)|\Psi_T(\mathbf{R}_j)|^2}{G(\mathbf{R}_j, \mathbf{R}_i, \Delta t)|\Psi_T(\mathbf{R}_i)|^2}\right\} \tag{25}$$

Cancelling out the prefactors in Green's function and merging the exponentials yields a simplified and less expensive to compute expression

$$p_{acc}(\mathbf{R}_i \to \mathbf{R}_j) = \min\left\{1,\ \Lambda\frac{|\Psi_T(\mathbf{R}_j)|^2}{|\Psi_T(\mathbf{R}_i)|^2}\right\} \tag{26}$$

with

$$\Lambda = \exp\left(0.5(\mathbf{R}_i - \mathbf{R}_j)(\mathbf{F}(\mathbf{R}_i) + \mathbf{F}(\mathbf{R}_j)) + \frac{D\Delta t}{4}([\mathbf{F}(\mathbf{R}_i)]^2 - [\mathbf{F}(\mathbf{R}_j)]^2)\right) \tag{27}$$

2.2.3 Gibbs sampling

Another popular sampling technique is the so-called Gibbs sampler, which is specifically designed to solve problems with multi-dimensional target distributions. The Gibbs sampler achieves this by sequentially drawing from the individual univariate conditional distributions, for which there often times analytical expressions are available. [5]
Assume that $\boldsymbol{X} = (X_1, \ldots, X_p)^T$, then the basic Gibbs sampler proceeds as follows:

1. Select initial values $\boldsymbol{x}^{(0)}$ and set $t = 0$.

2. Sample sequentially:

$$\begin{aligned}
X_1^{t+1} &\sim f(x_1|x_2^{(t)}, \ldots, x_p^{(t)}) \\
X_2^{t+1} &\sim f(x_2|x_1^{(t+1)}, x_3^{(t)}, \ldots, x_p^{(t)}) \\
&\vdots \\
X_{p-1}^{t+1} &\sim f(x_{p-1}|x_1^{(t+1)}, \ldots, x_{p-2}^{(t+1)}, x_p^{(t)}) \\
X_p^{t+1} &\sim f(x_p|x_1^{(t+1)}, \ldots, x_{p-1}^{(t+1)})
\end{aligned}$$

3. Increment t and return to step 2.

where $f(x_i|\cdot)$ denotes the univariate conditional distribution. This procedure is also called deterministic Gibbs scan where one iteration of step 2 is called a cycle. [5]

In this project the X_i values are the vectors representing the values of the visible and hidden nodes, giving that $\boldsymbol{X} = (\boldsymbol{x}, \boldsymbol{h})^T$. By using a two step Gibbs sampling procedure we can sample from the joint probability of $\boldsymbol{x}$ and $\boldsymbol{h}$. [3] The sampling is done according to the univariate conditional distributions

$$P(\boldsymbol{X}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{X}; \boldsymbol{a} + \boldsymbol{W}\boldsymbol{h}, \sigma^2) \tag{28}$$

and

$$P(\boldsymbol{H}|\boldsymbol{x}) = \prod_j^N \frac{e^{(b_j + \frac{\mathbf{x}^T \mathbf{w}_{*j}}{\sigma^2})H_j}}{1 + e^{b_j + \frac{\mathbf{x}^T \mathbf{w}_{*j}}{\sigma^2}}} \tag{29}$$

which means for the binary hidden values:

$$P(H_j = 1|\mathbf{x}) = \frac{1}{1 + e^{-b_j - \frac{\mathbf{x}^T \mathbf{w}_{*j}}{\sigma^2}}} \tag{30}$$

$$P(H_j = 0|\mathbf{x}) = \frac{1}{1 + e^{b_j + \frac{\mathbf{x}^T \mathbf{w}_{*j}}{\sigma^2}}} \tag{31}$$

2.3 Gradient descent

A crucial step in simulating a bosonic system with a RBM approach is finding the optimal values for all weights of the neural network denoted by $\boldsymbol{\alpha}$. Due to the large number of parameters in a RBM, employing a brute-force search is unfeasible. Instead a more intelligent scheme known as gradient descent can be used to solve the optimization problem. The gradient descent algorithm uses the gradient in order to iteratively update the parameters. [7] It is defined by the update step

$$\boldsymbol{\alpha}_{i+1} = \boldsymbol{\alpha}_i - \gamma \nabla_\alpha \mathbf{F}(\boldsymbol{\alpha}_i) \tag{32}$$

where $\mathbf{F}$ is the function to be optimized and γ is the step size, which scales the gradient and determines the speed of adaptation. [7] The gradient can be updated during the optimization to avoid overshooting and slow convergence. The optimization starts from some randomly initialized point $\boldsymbol{\alpha}_0$ and ends when a convergence criterion is satisfied (typically gradient is below a certain threshold). [7]
In this case the function to optimize is the expectation value of the local energy $\langle E_L(\boldsymbol{\alpha})\rangle$. To determine its gradient with respect to $\boldsymbol{\alpha}$ define

$$\bar{E}_{\boldsymbol{\alpha}} = \frac{d\langle E_L(\boldsymbol{\alpha})\rangle}{d\boldsymbol{\alpha}} \tag{33}$$

and

$$\bar{\Psi}_T = \frac{d\Psi_T(\boldsymbol{\alpha})}{d\boldsymbol{\alpha}} \tag{34}$$

By applying the chain rule and using the hermicity of the Hamiltonian it can be shown that

$$\bar{E}_{\boldsymbol{\alpha}} = 2\left(\left\langle \frac{\bar{\Psi}_T}{\Psi_T(\boldsymbol{\alpha})} E_L(\boldsymbol{\alpha})\right\rangle - \left\langle \frac{\bar{\Psi}_T}{\Psi_T(\boldsymbol{\alpha})}\right\rangle \langle E_L(\boldsymbol{\alpha})\rangle\right) \tag{35}$$

During the optimization we therefore need to keep track of

$$\frac{\bar{\Psi}_T}{\Psi_T(\boldsymbol{\alpha})} E_L(\boldsymbol{\alpha}) \tag{36}$$

and

$$\frac{\bar{\Psi}_T}{\Psi_T(\boldsymbol{\alpha})} \tag{37}$$

to be able to compute the expectation values needed for the derivative. Every $10^4 - 10^5$ Monte Carlo cycles a new gradient step is computed and the parameter $\boldsymbol{\alpha}$ is updated.

2.3.1 Stochastic gradient descent

The *stochastic* gradient descent method works based on the same principles, however instead of using the exact gradient during the update step, an estimation of the gradient is used. Such an estimate for the gradient may for example be obtained by only considering a subset of the full data and thusly increasing variability in the result. The main advantage of the stochastic gradient descent its improved capability of leaving local minima compared to the standard gradient descent method, while still maintaining good convergence behaviour. Furthermore, also the required computational time is less compared to the gradient descent. [7]

2.4 Blocking

Monte Carlo methods typically produce long sequences of data points, where each data point can be interpreted as an experimental measurement. The obtained data points should be statistically analyzed and evaluated to determine the expectation value of the quantity of interest and its estimated error margin, which can be calculated from the variance. [3]
The variance for an uncorrelated sample

$$var(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{38}$$

can be easily computed, but gives a too optimistic approximation of the true variance and therefore the error. The reason for this is that Monte Carlo simulations produce correlated samples because the individual steps are linked together in a Markov chain. [3]
For correlated samples the sample error is defined as

$$err_x^2 = \frac{1}{N}cov(x) \tag{39}$$

with the covariance

$$cov(x) = \frac{1}{N}\sum_{i,j}^{N}(x_i - \mu)(x_j - \mu) \tag{40}$$

Rewriting this expression using the autocorrelation function κ_d yields

$$err_x^2 = \frac{\tau}{N}\, var(x) \tag{41}$$

where τ is the autocorrelation time

$$\tau = 1 + 2\sum_{d=1}^{N-1}\kappa_d \tag{42}$$

with

$$\kappa_d = \frac{f_d}{var(x)} \tag{43}$$

and

$$f_d = \frac{1}{n-d}\sum_{k=1}^{n-d}(x_k - \mu)(x_{k-d} - \mu) \tag{44}$$

The computation of the autocorrelation time is very expensive since it contains the evaluation of a double sum over all particles and therefore has a complexity of $\mathcal{O}(N^2)$.
The calculation of this quantity can be circumvented by iteratively applying a so-called blocking transformation proposed in [8], that repeatedly halves the length of the data set. [8] The transformation is given as

$$x_i' = \frac{1}{2}(x_{2i-1} + x_{2i}) \tag{45}$$

so that

$$n' = \frac{1}{2}n \tag{46}$$

The sample error err_x^2 is invariant under this operation, which leads to the inequality [8]

$$err_x^2 \geq \left\langle \frac{f_0}{n-1} \right\rangle \tag{47}$$

It can be shown that there exists a fixed point for which the equality holds. [8]
Equation (47) is evaluated for decreasing block size according to the blocking transformation by approximating the expectation value with the fraction $\frac{f_0}{n-1}$ itself. This process is repeated until the block size is $n' = 2$. (Because the block size is halved in each iteration it is advisable to have data sets with lengths that are powers of 2.) The values for $\frac{f_0}{n-1}$ will increase with decreasing block size. As soon as a fixed point is reached the values will converge and form a plateau with constant values within an error margin. The highest value of this plateau is the estimation of the sample error err_x^2. [8]

In this project an automated approach to the blocking method proposed in [9] is used.

2.5 Restricted Boltzmann Machines

In recent years a novel approach to the solution of the quantum many-body problem has become popular in the scientific community. Instead of applying all physical knowledge one might have about the system in question when specifying a wave function, a completely agnostic approach is taken by representing the wave function by a self-learning neural network. [2]
Supervised learning approaches, mainly used for classification problems, can not be used since they require labeled data, which are not at all or not abundantly available. Instead generative networks are used which can learn a probability distribution from their input training data. The most common type of generative networks are so called Boltzmann Machines (BM). BMs are energy-based models, meaning that they are not trained using the backpropagation algorithm for optimization (like most of the networks used for supervised learning), but instead use Markov-Chain-Monte-Carlo methods to simulate an energy term. This energy term acts in the same way as a cost function in supervised learning and is minimized during training using gradient methods. Therefore, this approach is in fact very similar to the traditional

Variational Monte Carlo (VMC) approach. [3, 7]
In standard BMs all nodes are connected with each other, not only between layers but also within layers, which drastically increases the networks complexity and the computational time needed to train it. Therefore, a streamlined variation of the BM is used, the Restricted Boltzmann Machine (RBM). A RBM consists of one visible and one hidden layer, where each node of the visible layer is connected with each node of the hidden layer (figure 1). [3, 7]

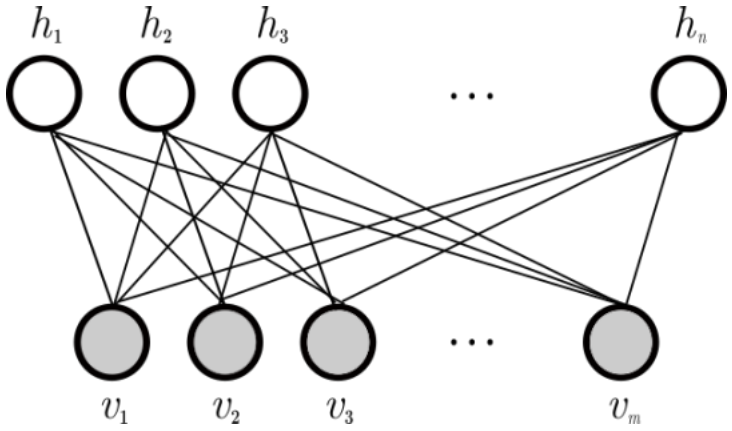

Figure 1: Architecture of a Restricted Boltzmann Machine. [10]

Furthermore, each node is connected to a bias weight. The visible layer represents the input as well as the output the network might give. In the case of quantum mechanical studies this means that the visible layer represents the positions of all particles. The size of the visible layer should be chosen to reflect this. The hidden layer is used to model the expressive power of the network. The weighted connections between the visible and the hidden layer then define the complex relationships between visible and hidden nodes. [3]
The optimizable parameters are the visible and hidden biases and the weights between the layers, so that the total number of weights is $N_{tot} = N_{vis} + N_{hid} + N_{vis} \cdot N_{hid}$. [7]

There are different types of RBMs with differences in the type of nodes and the energy function. Most common are the Binary-Binary and the Gaussian-Binary RBMs. In Binary-Binary RBMs both the visible as well as the hidden nodes can only take binary values. This variant is unfeasible for the quantum many-body problem since we want to model the positions using the RBM. Instead the Gaussian-Binary RBM is used in which the visible layer takes Gaussian values while the hidden layer takes binary values, so that the visible nodes can be used for the positional data of the particles in the quantum mechanical system. The corresponding energy function for that type of RBM is given by

$$E(\boldsymbol{x}, \boldsymbol{h}) = \sum_{i}^{M} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{j}^{N} b_j h_j - \sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2} \tag{48}$$

where M and N are the number of visible and hidden nodes respectively, $\boldsymbol{x}$ and $\boldsymbol{h}$ are the visible and hidden nodes respectively, $\boldsymbol{a}$ and $\boldsymbol{b}$ are the visible and hidden biases respectively, $\boldsymbol{w}$ is the weight matrix and σ is the chosen standard deviation of the Gaussian model. [3]

From this the joint distribution of $\boldsymbol{x}$ and $\boldsymbol{h}$ can be modeled using the Boltzmann distribution as

$$F_{rbm}(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z} e^{-\frac{1}{T} E(\boldsymbol{x}, \boldsymbol{h})} \tag{49}$$

where the temperature value T is set to 1 for simplicity. The wave function represents the probability amplitude with dependence on $\boldsymbol{x}$. Therefore, we further derive the marginal distribution of $\boldsymbol{x}$

$$F_{rbm}(\boldsymbol{x}) = \sum_{\boldsymbol{h}} F_{rbm}(\boldsymbol{x}, \boldsymbol{h}) = \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-\frac{1}{T} E(\boldsymbol{x}, \boldsymbol{h})} \tag{50}$$

Inserting the expression for the energy finally leads to an expression of the wave function, the so called neural network quantum state (NQS)

$$\Psi(\boldsymbol{x}) = \frac{1}{Z} e^{-\sum_i^M \frac{(x_i - a_i)^2}{2\sigma^2}} \prod_j^N (1 + e^{b_j + \sum_i^M \frac{x_i w_{ij}}{\sigma^2}}) \tag{51}$$

Since we have no training data available to train the model parameters, we instead make use of the variational principle. By simulating the local energy with Monte-Carlo cycles and minimizing the expectation value of the local energy, we try to obtain parameter values that produce the ground state energy of the system. For the minimization process gradient based methods such as stochastic gradient descent or the Adam optimizer can be used. The expression for the gradient with respect to the model parameters is the same as in any standard VMC calculation and is given by (35). [3]

3 Implementation

The RBM algorithm has been implemented in *Python* using an object-oriented approach. For random number generation and vector operations the package *numpy* is used. In the following all classes and their functionalities are briefly outlined:

- **NeuralQuantumState** - Static class representing the trial wave function that contains analytic expressions for gradients, Laplacians and parameter derivatives of the respective wave function.
- **Hamiltonian** - Class that represents the chosen Hamiltonian by defining expressions for the kinetic, potential and (if present) interaction energies. Needs to be initialized with a specified instance of the NeuralQuantumState class.
- **Particle** - Custom data type for a particle containing its position and methods to propose a trial position.
- **System** - Custom data type for a system of particles containing a list of all particles and functions to evaluate the wave function, local energy and drift force. Needs to be initialized with a specified instance of the Hamiltonian class.
- **Simulation** - Main class for running a RBM simulation. Contains all major, high level functionalities (MC sampler, gibbs sampler, stochastic gradient descent method). Needs to be initialized with a specified instance of the System class.
- **Observables** - Custom data type for handling energy expectation values during the simulation.

Furthermore, an independent script for the calculation of the error using the method of blocking[1] was used.

3.1 How to run

To start a simulation first all required parts need to be initialized. First a list of Particle objects should be created where every Particle is instantiated with a specified position that takes the form of an array with equal length to the dimension. Then a NeuralQuantumState object should be instantiated, specifying the structure of the neural network (number of nodes). Using the instantiated NeuralQuantumState object the Hamiltonian can be build by specifying whether to include the interaction potential or not. Then a System object can be instantiated passing the Hamiltonian and the list of particles to the constructor. Lastly, a simulation object can be instantiated using the System object.

The code furthermore provides an additional script (*run.py*), that automatically performs the initial setup and then runs a simulation. All relevant parameters can be directly configured in the script.

[1] This script was provided by the supervisor [3].

4 Results and Discussion

4.1 Non-interacting system

As a proof of concept for the RBM approach, first the non-interacting case has been analyzed where closed form expressions are available. This has been done for one particle in one dimension for all three different sampling techniques. The expected local energy for the non-interacting case is given by

$$E_L = \frac{1}{2} P \cdot D \tag{52}$$

where P and D denote the number of particles and dimensions respectively.

4.1.1 Brute-force sampling

For brute-force sampling multiple runs with different learning rates and the parameters specified in table 1 have been conducted. All runs converged to parameterization of the RBM that lead to reasonable estimates of the local energy.

Table 1: Parameter values for brute-force Monte-Carlo simulation of the non-interacting system.

Parameter	Value
mass m	1
oscillator frequency ω	1
Monte-Carlo iterations N_{MC}	1000
step size d	0.45
learning rate γ	$0.1 - 0.9$
RBM variance σ^2	1
number of hidden units n_h	4
optimization cycles N_{opt}	1000

Figure 2 shows the comparison of the performances of three different learning rates. It becomes apparent that the best learning rate is the highest one, whereas the results obtained with lower learning rates show higher variance in the trajectories.
Using RBM variance values different from $\sigma^2 = 1$ seemed to worsen the results and was detrimental to the overall convergence behaviour.

In figure 3 the results of different numbers of hidden nodes for brute-force Monte-Carlo sampling are displayed. It can be observed that two, four and six hidden nodes perform fairly similar in the limit of $i \to \infty$. However, the smaller the number of hidden nodes the higher the convergence speed which is most likely due to the fact that if there are fewer hidden nodes there also are less parameters to optimize. Using ten hidden nodes results in a highly volatile energy trajectory.

4.1.2 Importance sampling

Importance sampling has been done with the parameterization specified in table 2. The overall result was similar to the result obtained from brute-force sampling, that is the local energy could reliably be

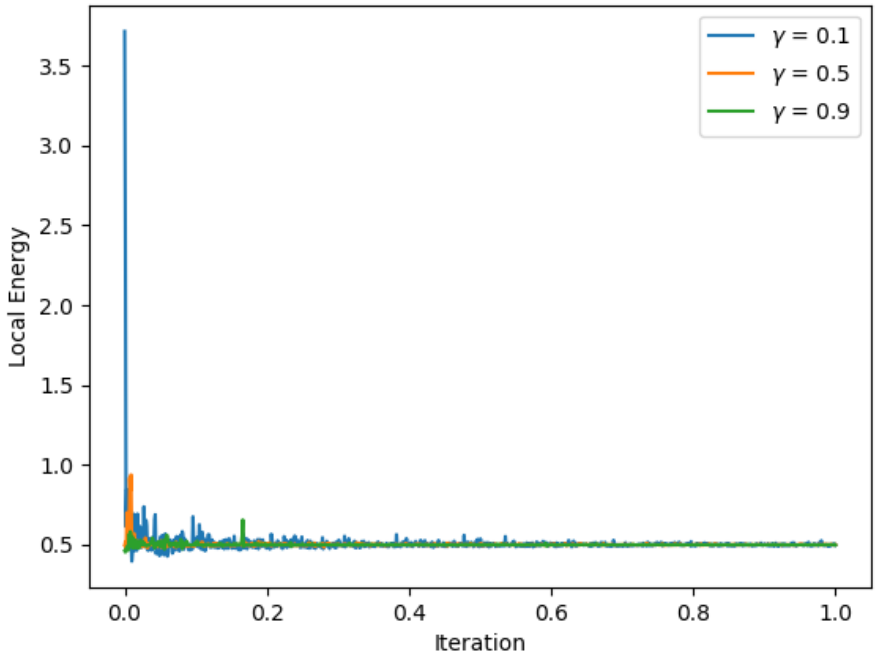

Figure 2: Performance of the brute-force Monte-Carlo sampler for different learning rates.

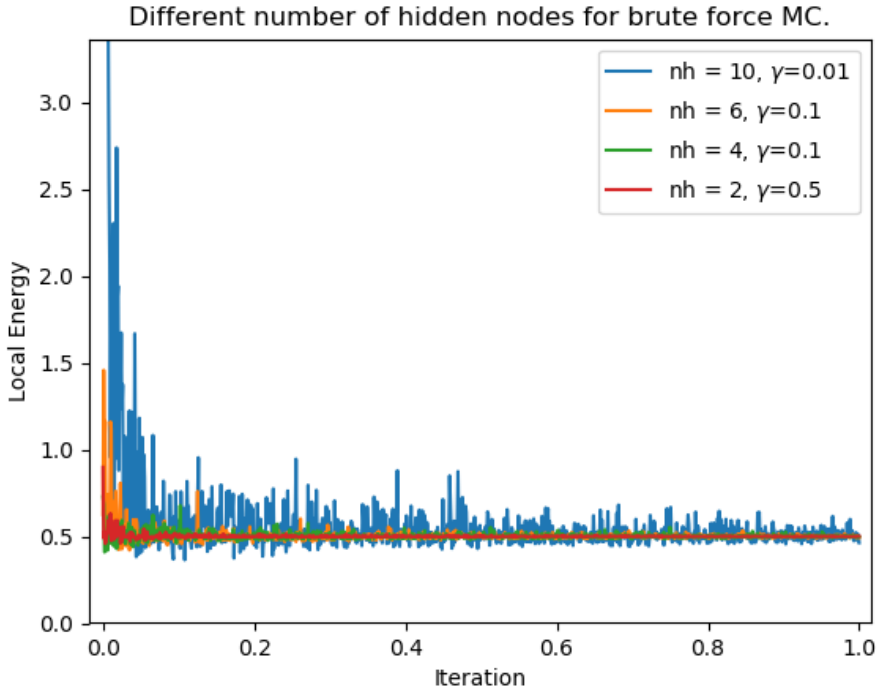

Figure 3: Performance of the brute-force Monte-Carlo sampler for different numbers of hidden nodes.

reproduced by the algorithm. One main difference however was, that it was beneficial to the convergence behaviour to use smaller learning rates (~ 0.1) instead of the high values of 0.9 that have been used for brute-force sampling.
The convergence behaviour and its dependency on the parameter values was very similar to the results obtained for simple brute-force sampling.

Table 2: Parameter values for importance Monte-Carlo simulation of the non-interacting system.

Parameter	Value
mass m	1
oscillator frequency ω	1
Monte-Carlo iterations N_{MC}	1000
time step Δt	0.45
diffusion coefficient D	0.5
learning rate γ	0.1
RBM variance σ^2	1
number of hidden units n_h	4
optimization cycles N_{opt}	1000

4.1.3 Gibbs sampling

Gibbs sampling proved to show very smooth convergence behaviour as can be seen in figure 4. The plot also shows that by using the Gibbs sampler the local energy converges to the correct, analytical value.
In figure 5 the convergence behaviour of the Gibbs sampler with different numbers of hidden nodes is displayed. It can be observed that two, four and six hidden nodes converge more or less to the same extent after 200 optimization iterations (while slightly favoring four hidden nodes). When using two hidden nodes the algorithm converges slightly faster which is due to the fact that less parameters need to be optimized in that case so that convergence can be achieved faster. When using ten hidden nodes the convergence behaviour is significantly worse compared to all other configurations which is probably due to the fact that the learning rate had to be adapted in that case for the algorithm to run properly (numerical problems arose).
Figure 6 shows the convergence behaviour of four different values of σ. A value of 1.0 seems to be the best. Higher values let the algorithm converge to incorrect values, whereas smaller values lead to a noisy trajectory.

For two particles in two dimensions, Gibbs sampling seems to result in a very volatile start, but does eventually converge to the actual value, close to a local energy of a bit above $E_L = 2$. The variable learning rate was introduced to potentially avoid any local minima.

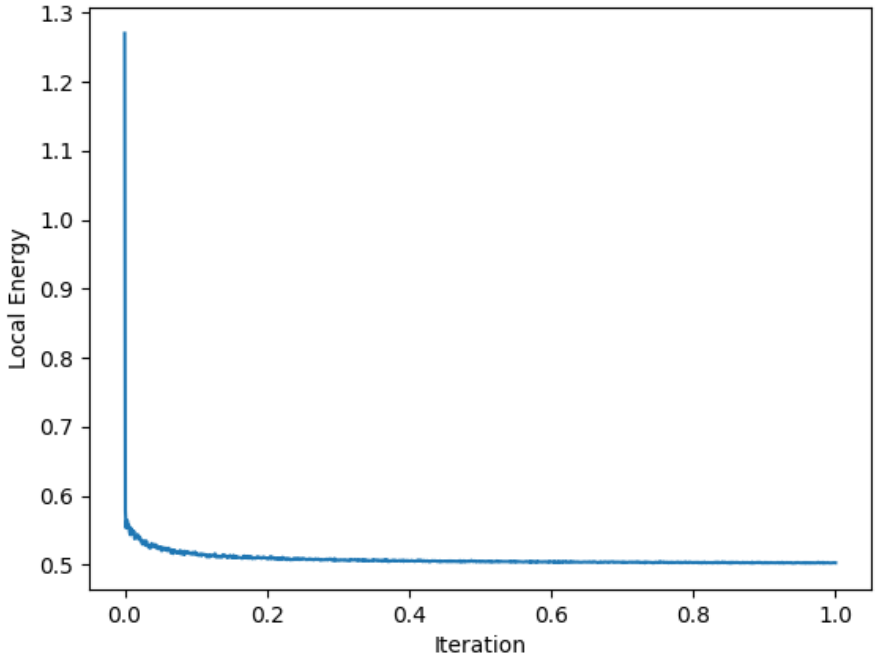

Figure 4: Convergence behaviour of the Gibbs sampler.

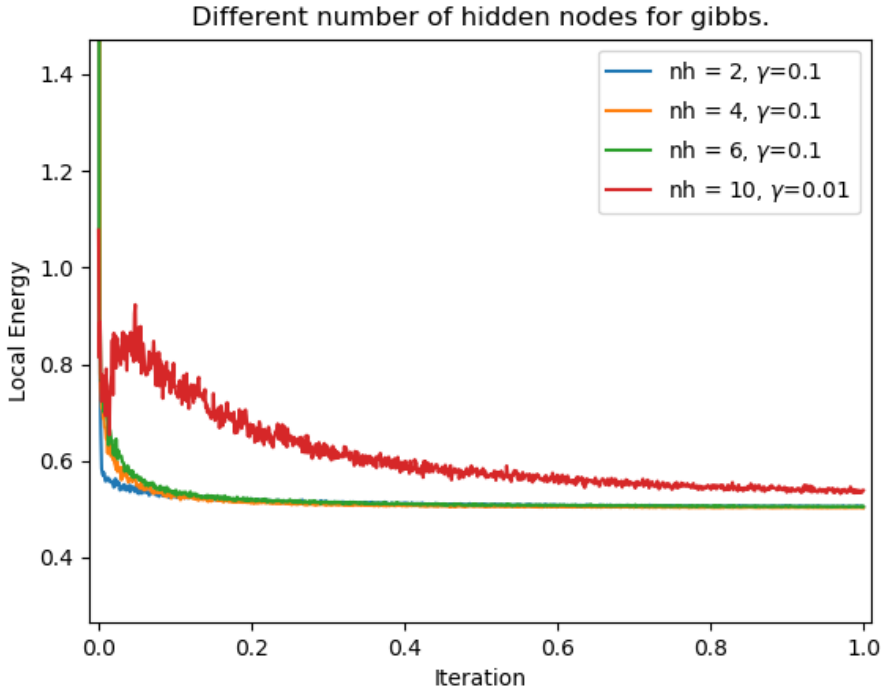

Figure 5: Performance of the Gibbs sampler for different numbers of hidden nodes.

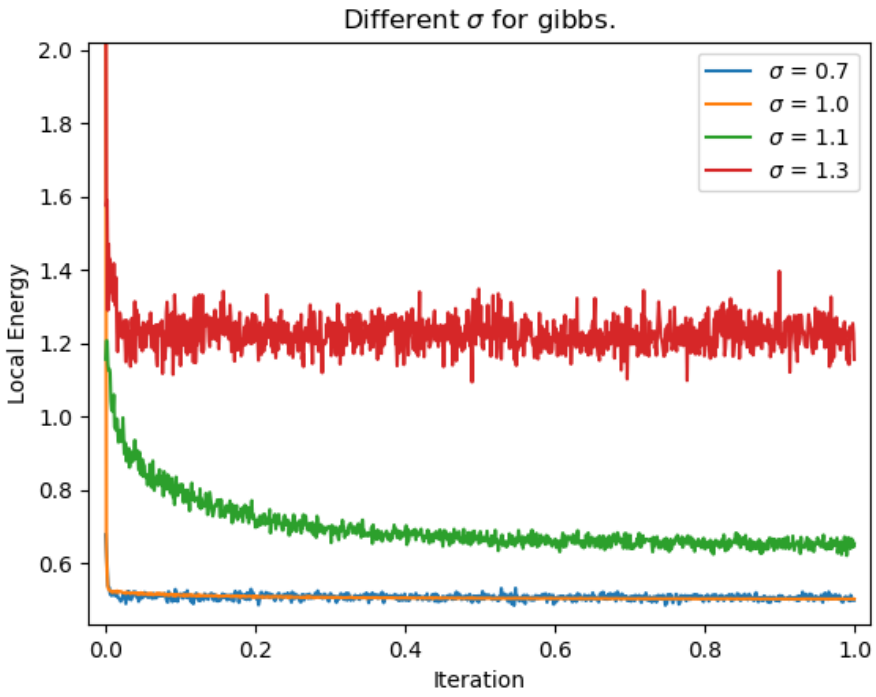

Figure 6: Performance of the Gibbs sampler for different values of the RBM standard deviation.

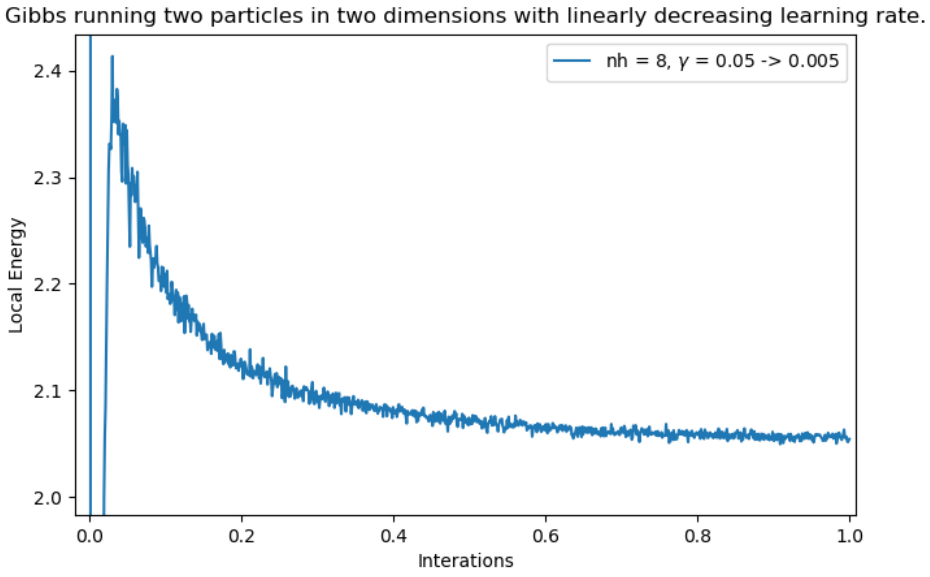

Figure 7: Gibbs with two particles and two dimensions with a linearly decreasing learning rate from 0.05 down to ten percent of the original value. Here, it is also zoomed in a little bit to better show the evolution. The lowest value of the "dip" at 0.01 is around $E_L = -10$.

4.2 Interacting system

For the analysis of the interacting system of two particles in two dimensions the importance sampling procedure has been chosen. A local energy estimate of $\hat{E}_L = 3.268$ with a standard error of $\sigma = 0.016$ could be obtained by running a simulation with parameters specified in table 3.
Comparing these results to the analytical result for the local energy of $E_L = 3$ ([1]), it can be observed that the local energy obtained with the RBM approach constitutes a somewhat reasonable estimate for the local energy of the interacting case. However, it has to be noted that the precision is not even given up to the first decimal place.

Table 3: Parameter values used for the simulation of the interacting system.

Parameter	Value
mass m	1
oscillator frequency ω	1
Monte-Carlo iterations N_{MC}	1000
time step Δt	0.45
diffusion coefficient D	0.5
learning rate γ	0.1
RBM variance σ^2	1
number of hidden units n_h	6
optimization cycles N_{opt}	1000

The other two sampling methods (brute-force, Gibbs) were also able to reproduce the digit before the decimal place, but gave worse results with higher variability overall.

5 Conclusion

The analysis of the non-interacting system showed that the RBM algorithm sampled correctly and produced almost the exact analytical results. The three different sampling techniques (brute-force, importance, Gibbs) that have been implemented produced comparable results with small deviations in the energy estimates. However, brute-force sampling had substantially lower efficiency in comparison with importance and Gibbs sampling, which is due to the higher rejection rate of this naive approach.

The obtained results of the interacting system (two particles in two dimension) were in accordance with [1]. The analytical result could be reproduced using any one of the three implemented sampling techniques, but importance sampling gave the best results. Correct precision could not even be obtained for the first decimal. Longer optimization runs or the employment of a more involved optimization algorithm such as Adam might however able to achieve this.

Overall it could be shown that RBMs constitute a valid and powerful approach for the solution of many-body quantum problems, which does not require extensive *ab initio* knowledge about the physical system in question. The usage of neural networks together with gradient based optimization methods proved to be fit for such an agnostic approach as illustrated in this project.

The programming language *python* proved to be a simple and flexible means for the implementation of a scientific application like this RBM algorithm. The possibility for object-oriented programming enables the creation of structured and maintainable code. However, when it comes to performance and required CPU time *python* cannot quite compete with low level programming languages such as *C++* that excel at executing loops and simple mathematical operations. This problem can be counteracted by using *python* libraries such as *numba* that allows just-in-time compilation of *python* code into fast machine code.

References

[1] M. Taut. Two electrons in an external oscillator potential: Particular analytic solutions of a coulomb correlation problem. *Phys. Rev. A*, 48:3561–3566, Nov 1993.

[2] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

[3] Morten Hjorth-Jensen. Lecture notes computational physics ii: Quantum mechanical systems, 2019.

[4] Walter Kauzmann. *Quantum chemistry: an introduction*. Elsevier, 2013.

[5] G.H. Givens and J.A. Hoeting. *Computational statistics*. Wiley series in probability and statistics. Wiley-Interscience, 2005.

[6] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[7] Stephen Marsland. *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[8] Henrik Flyvbjerg and H.G. Petersen. Error estimates on averages of correlated data. *The Journal of Chemical Physics*, 91, 07 1989.

[9] Marius Jonsson. Standard error estimation by an automated blocking method. *Physical Review E*, 98, 10 2018.

[10] LazyProgrammer. Deep learning tutorial: Deep belief networks. `https://lazyprogrammer.me/tag/restricted-boltzmann-machines`, June 2015.

List of Figures