Machine learning for quantum many-body problem

Restricted Boltzmann and Quantum Boltzmann Machine

A.J Luis & P.V Joseph
University of Oslo

March 10, 2025

Abstract

In this second project we work with a system composed by two fermions into the harmonic oscillator. As in the previous project, the initial calculations will be performed using Metropolis algorithm with and without repulsive interaction. A new wave function, neural network quantum state (NQS), will replace our previous wave function, which implies that we must optimize a Restricted Boltzmann Machine (RBM) with several parameters (biases and weights). Furthermore, we will include a study on Quantum Boltzmann Machine with the helped of Qiskit.

Contents

1 Introduction

Currently, restricted Boltzmann machines (BMs) are a useful tool in the field of machine learning, offering robust capabilities for unsupervised learning tasks. RBMs are of vital importance, since they allow us to study the quantum many-body problem [1, 2, 3].

Therefor, for this project we will combine two courses oriented to the Restricted Boltzmann Machine (RBM) and Quantum Boltzmann Machine (QBM): Computational Physics II (FYS4411) and Quantum computing and quantum machine learning (FYS5419).

The **first aim for this project is to study the RBM** in a system with many bodies, where we must calculate the ground state energy through modifications in our previous code on Variational Monte Carlo (VMC). The old wave function will be replaced by the neural network quantum state (NQS), which contains the parameters W_{ij}, a_i and b_j, introducing the weights and visible and hidden biases. The several parameters problem will be solved through the stochastic gradient descent, which will allow us to optimize them.

Note that it will be necessary to calculate the local energy and the gradient of the local energy with respect to the parameters, these calculations will be shown in the Appendix A and B section. You can notice that the workflow will be very similar to project 1 (FYS4411), this will allow using the Metropolis algorithm either force brute or importance sampling. In addition, we will use blocking method to perform a statistical analysis.

The initial calculations will be to perform for a system with one particle ($N = 1$), one-dimension ($D = 1$) and with two hidden variables (h = 2) without repulsive interaction. We will study the system by modifying the learning rate and the number of hidden variables. This will be calculated with Metropolis algorithm (force brute). Later the same analysis will be carried out for a system with two particles ($N = 2$), two-dimension ($D = 2$) and including the repulsive interaction. For this second case, we will use both algorithms, force brute and importance sampling.

The **second aim will be to study the QBM**, how the training of weights should be performed through the use of a quantum neural network and quantum circuits. We will basically represent the workflow and the ingredients that make up this process.

For this we will analyze two examples taken from different libraries. The first taken from Qiskit Machine Learning, where we will work with a real database and the second taken from Prabh Simran's repository, where we will work with a artificial database (autogenerate).

In the **section 2**, it will be included RBM. There we will briefly describe the VMC and RBM formalism. Subsections will also be included for code implementation, results, analysis and discussions about the topic.

Section 3 will be referred to QBM, also with subsections for formalism, results, analysis and discussions. While in **section 4**, it will be included the conclusions for both approaches.

2 Restricted Boltzmann Machines (RBMs)

This section is focus in the classical RBMs in the context of variational Monte Carlos method (VMC), we are going to start making a theoretical background, then saying something about code implementation and analisys of the results. In the final of this section is an overview of BMs.

2.1 Formalism

Here, there are the different theoretical components to study the BMs in the environment of VMC. Making a discussion of variational Monte Carlos method, Markov chains, metropolis, importance sampling and the definition RBMs. At the end, some necessary concepts used in VMC calculation.

2.1.1 Variational Monte Carlo Method (VMC)

The use of the probability distribution makes VMC a very useful tool, its definition can be known from quantum mechanics through the wave function.

$$P(R,\alpha) = \frac{|\Psi_T(R,\alpha)|^2}{\int |\Psi_T(R,\alpha)|^2 dR} \tag{2.1}$$

where the positions of the particles are related to $R = (R_1, \dots, R_N)$ and α is in general a variational parameter becomes to represent the visible, hidden layer and interaction between them in BMs. This relation between the probability distribution and wave function allows us to find a local minimum, with which local energy can be defined.

$$E_L(R,\alpha) = \frac{1}{\Psi_T(R,\alpha)} H\Psi_T(R,\alpha) \tag{2.2}$$

The expectation value of the local energy is given by

$$\langle E_L \rangle = \int P(R) E_L dR \approx \frac{1}{N} \sum_{i=1}^{N} E_L(x_i) \tag{2.3}$$

Herein N is the number of Monte Carlo cycles.

2.1.2 Markov chains and Metropolis algorithm

The Markov chains (MC) is a special type of stochastic process in which the probability of an event occurring depends only on the immediately preceding event, this especial process is an ally to find the so-called Metropolis algorithm this helps us find a better way to calculate the integral using MCM.

To find a Monte Carlos Metropolis algorithm (MCMa) we need defining PDF but also a transition probability distribution function (TPDF), using the MC we know the transition probability only has to be depended on the probability immediately preceding event if we start to assume a discrete probability how show figure below

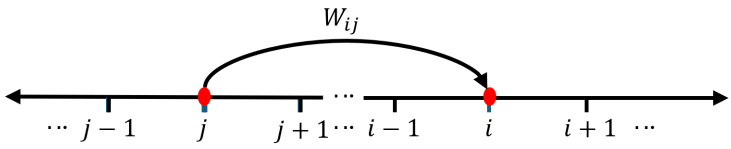


Figure 1: Illustration of the Markov chains process

In the **Figure** 1 we have that $i \longrightarrow w_i(t)$ is the PDF in us case is a square wave function, and it is mean to find the system in the state i at the time t, then defining the TPDF $W_{ij}(t)$

$$w_i(t) = \sum_j W_{ij}(t) w_j(t) = \sum_j W_{(j \longleftarrow i)}(t) w_j(t)$$

it gives us the TPDF to pass the state $w_j(t)$ to $w_i(t)$.

The problem comes that we do not have the TPDF, but we can do some subjection about that because has to satisfy some properties since

- $0 \leq w_i(t) \leq 1$
- $\sum_i w_i = 1$
- $0 \leq W_i j(t) \leq 1$
- the sum of some columns or rows $\sum_i W_{ij}(t) = \sum_j W_{ij}(t)$
- The eigenvalues $\|\lambda(W_i j)\| \leq 1$ because TPDF must be a stochastic matrix, the $\|\lambda_{max}\| = 1$

for simplicity we can do other assumption that $W_{ij}(t)$ time independent the transition or jumping the state i to j are instantaneously. We can find the probability after the time t,

$$w_i(t_0 + \varepsilon) = \sum_j W_{(j \longrightarrow i)} w_j(0) = W w_j(0)$$

$$w_i(t_0 + 2\varepsilon) = \sum_j W_{(j \longrightarrow i)} w_j(\varepsilon) = W w_j(\varepsilon) = W^2 w_j(0)$$

$$w_i(t_0 + 3\varepsilon) = \sum_j W_{(j \longrightarrow i)} w_j(2\varepsilon) = W w_j(2\varepsilon) = W^3 w_j(0)$$

if we follow with this idea after n steps

$$w_i(t_0 + n\varepsilon) = \sum_j W_{(j \longrightarrow i)} w_j((n-1)\varepsilon) = W^n w_j(0)$$

let $t_0 = 0$ and $n\varepsilon = t$, we found the probability at the time t

$$w_i(t) = W^{t/\varepsilon} w_j(0)$$

then we consider W has eigenvalues $\{\lambda_0, \lambda_1, ..., \lambda_k ...\lambda_F\}$ where the largest value is $\{\lambda_0 = 1 \leq \lambda_1 \leq ... \leq \lambda_k ... \leq \lambda_F\}$ and eigenvectors $\{v_0, v_1, ..., v_k, ...v_F\}$, this way we can expand $w_j(0)$ as a lineal combination of v_k

$$W w_j(0) = W \sum_k^F \alpha_i v_k = \sum_k^F \alpha_i \lambda_k v_k$$

introducing a variables called dumping time $\tau_k = -\frac{1}{\ln \lambda_k}$, then we can write the PDF for $w_i(t)$ as

$$w_i(t) = \sum_k^F \alpha_k v_k \exp\left(-t/\tau_k\right) = \alpha_0 v_0 + \sum_{k=1}^F \alpha_k v_k \exp\left(-t/\tau_k\right) \quad [2.4]$$

after the so large time $t \longrightarrow \infty$ the **Equation** 2.4 becomes

$$\lim_{t\longrightarrow\infty} w_i(t) = \alpha_0 v_0 + \lim_{t\longrightarrow\infty} \sum_{k=1}^F \alpha_k v_k \exp\left(-t/\tau_k\right)$$

$$\lim_{t\longrightarrow\infty} w_i(t) = w_i = \alpha_0 v_0 \quad [2.5]$$

Equation 2.5 says for long time PDF becomes to be constant, this state is called the more likely state or stationary state.

The next step it is constructing the Metropolis algorithm. We do not TPDF, therefor we need to make a model is there when Metropolis algorithm helps ours. We are going to start with the TPDF can be written as product of $W_{(j\Longrightarrow i)} = T_{(j\Longrightarrow i)} A_{(j\Longrightarrow i)}$ where $T_{(j\Longrightarrow i)}$ and $A_{(j\Longrightarrow i)}$ gives us the probability to do a transition (jumping) and the probability to accept the transition (jumping) from j to i respectively

$$w_i(t) = \sum_j W_{(j\longrightarrow i)} w_j(t-\varepsilon) = \sum_j \left\{ w_j(t-\varepsilon) T_{(j\longrightarrow i)} A_{(j\longrightarrow i)} + w_i(t-\varepsilon) T_{(i\longrightarrow j)} \left(1 - A_{(i\longrightarrow j)}\right) \right\} \quad [2.6]$$

we have two terms in the right hand, the first is easy to interpret because is the probabilities jumping from j to j and the second term maybe little more difficult to interpret, but it is the probability just staying $w_i(t)$ if we already are in i not jumping, for this reason appear $(1 - A_{(i\longrightarrow j)})$ we neglected the probability to jumping.

we need to assume $T_{(i\longrightarrow j)}$ and $A_{(i\longrightarrow j)}$ have to be normalized then we can rewrite **Equation** 2.6 as

$$w_i(t) = w_i(t-\varepsilon) + \sum_j \left\{ w_j(t-\varepsilon) T_{(j\longrightarrow i)} A_{(j\longrightarrow i)} + w_i(t-\varepsilon) T_{(i\longrightarrow j)} A_{(i\longrightarrow j)} \right\} \quad [2.7]$$

now, if we take the limit when time a large quantity says infinite the PDF becomes to be constant, therefor **Equation** 2.7 can be written as

$$0 = \sum_j \left\{ w_j T_{(j\longrightarrow i)} A_{(j\longrightarrow i)} + w_i T_{(i\longrightarrow j)} A_{(i\longrightarrow j)} \right\}$$

$$w_j T_{(j\longrightarrow i)} A_{(j\longrightarrow i)} + w_i T_{(i\longrightarrow j)} A_{(i\longrightarrow j)} = 0$$

or

$$\frac{w_i}{w_j} = \frac{T_{(j\longrightarrow i)} A_{(j\longrightarrow i)}}{T_{(i\longrightarrow j)} A_{(i\longrightarrow j)}} \quad [2.8]$$

the problem still do not finish because we do not what it exactly $T_{(...)}$ and $A_{(...)}$, so we can do some restriction like that the transition probability has a reversal symmetric property that meaning $T_{(j\longrightarrow i)} = T_{(i\longrightarrow j)}$

$$\frac{w_i}{w_j} = \frac{A_{(j\longrightarrow i)}}{A_{(i\longrightarrow j)}} \quad \Rightarrow \quad \textbf{Metropolis algorithm} \tag{2.9}$$

this algorithm has an advantage in the relation with VMC. In MCM that wave functions needs to be normalized, but to do this we need to find integral of norm and obviously is very time consuming, but the **Equation** 2.9 we do not need because this factor is cancelled.

The next steps it is illustrated how works **Equation** 2.9, the answer is in the fact we always want to maximize the probability the better way to see it thinking in a Gaussian distribution **Figure** 2 the value central is which maximized the probability, so we need to choose the value of $A_{(j\longrightarrow i)}$ it is actually computational Metropolis algorithm did.

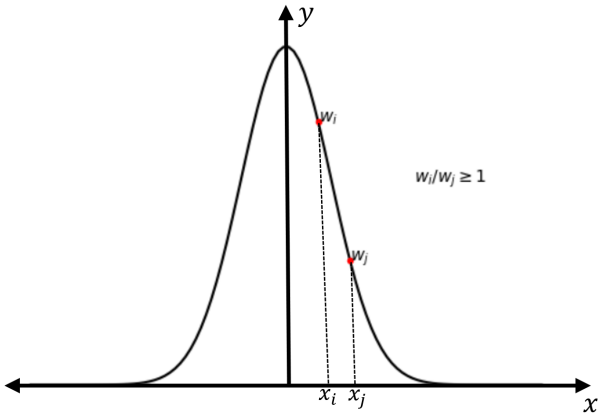


Figure 2: Illustration of Metropolis algorithm

normally this values it is chosen $A_{(j\longrightarrow i)} = 1$ and then we are going to have $\frac{A_{(j\longrightarrow i)}}{A_{(i\longrightarrow j)}} \geq 1$ and $\frac{w_i}{w_j} \geq 1$. It gives us the recipe to maximize the PDF.

2.1.3 Importance Sampling

The important sampling is a way to improve the MCMa. We need to know more about the transition probability, this allows us to find a better selection (see **Equation** 2.8) in jumping from w_j to w_i. This approach is based on the Fokker-Planck equation and the Langevin equation for generating a trajectory in coordinate space.

For a diffusion process characterized by a time-dependent probability (transition probability) density $P(x,t)$ in one dimension the Fokker-Planck equation for one particle or walker is given

$$\frac{\partial P}{\partial t} = D\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} - F\right)P(x,t), \tag{2.10}$$

where is F a drift quantum force term and contained derivative of the trial wave function $\mathbf{F} = 2\frac{1}{\Psi_T}\nabla\Psi_T$, and D is the diffusion coefficient.

The new positions in coordinate space are given as the solutions of the Langevin equation using Euler's method, and from the Langevin equation we can show

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

where η is a random variable, then this give us a new position

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t}, \tag{2.11}$$

where ξ is a Gaussian random variable and Δt is a chosen time step. the quantity D can be found through the so called fluctuation-dissipation theorem and it is given $D = 1/2$. Other important aspect we have to be into account the time here it just a parameter in the best option to choose for a good convergence to value between $\Delta t \in [0.001, 0.01]$.

The solution the Fokker-Planck equation give us a transition probability which has as a solution the Green's function

$$G(y, x, \Delta t) = \frac{1}{(4\pi D\Delta t)^{3N/2}} \exp\left(-(y - x - D\Delta t F(x))^2/4D\Delta t\right) \tag{2.12}$$

which means we have found a transition probability $T(...)$ for the Metropolis algorithm, so now we have

$$q(y, x) = \frac{G(x, y, \Delta t)|\Psi_T(y)|^2}{G(y, x, \Delta t)|\Psi_T(x)|^2} \tag{2.13}$$

where q can be interpreted like a reason between probability distribution to be in x and y respectively. look back in the discrete case we have

$$q(y, x) \equiv \frac{T_{(i\longrightarrow j)}}{T_{(j\longrightarrow i)}}\frac{w_i}{w_j} = \frac{A_{(j\longrightarrow i)}}{A_{(i\longrightarrow j)}} \tag{2.14}$$

and

$$T_{(k\longrightarrow l)} \equiv G(x_k, x_l, \Delta t) = \frac{1}{(4\pi D\Delta t)^{3N/2}} \exp\left(-(x_k - x_l - D\Delta t F(x))^2/4D\Delta t\right)$$

we have found an Importance sampling algorithm, and then we can use this new condition in ordered to find a Monte Carlo Importance Sampling algorithm (MCISa). On the other hand, it is not difficult to see the improvement is coming for the fact that has walker has a restriction where can jumping (see **Equation** 2.11) and obviously for the knowledge transition probability.

2.1.4 Neural network quantum state (NQS) wave function

For study our different systems, we must use a Hamiltonian. For this project the Hamiltonian will be described by two terms, a harmonic oscillator term and a coulomb interaction term.

$$H = \sum_{i=1}^{N}\left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2\right) + \sum_{i<j}\frac{1}{r_{ij}} \tag{2.15}$$

Where ω is the oscillator frequency, N the number of particles and r_{ij} is the distance between the particles given by $r_{ij} = |r_1 - r_2|$. The wave function is given from the energy method of the restricted Boltzmann machine, which is composed by visible (x) and hidden (h) nodes.

$$F_{rbm}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z}e^{-\frac{1}{T_0}E(\mathbf{x},\mathbf{h})}. \tag{2.16}$$

The marginal probability $F_{rbm}(x) = \sum_h F_{rbm}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z}\sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}$ is used to represent the wave function.

2.1.5 Gaussian-Binary RBM

This method is an extension of the Boltzmann Machine, it is an energy-based generative model which include hidden and visible variables. The restricted term indicates that there is connection between layers, but no between nodes and layers. The energy related to a configuration of the nodes is given by

$$E(\mathbf{x},\mathbf{h}) = \sum_{i}^{M} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{j}^{N} b_j h_j - \sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2}. \tag{2.17}$$

For our project we want to work with Gaussian-binary, that means that the visible and hidden layer take a continuous value. then the marginal probability becomes to be

$$\Psi(X) = F_{rbm}(x) \tag{2.18}$$

$$= \frac{1}{Z} \exp\left(- \sum_{i}^{M} \frac{(x_i - a_i)^2}{2\sigma^2} \right) \prod_{j}^{N} \left(1 + \exp\left(b_j + \sum_{i}^{M} \frac{x_i \omega_{ij}}{\sigma^2} \right) \right) \tag{2.19}$$

where Z is the partition function, X_i are the visible nodes, a_i and b_j are the visible and hidden biases and ω_{ij} is a MxN matrix holding the weights connecting the visible with the hidden nodes.

The **Equation** 2.19 is for one particle. For many-particles system, the wave function ansatz for N particles is given,

$$\Psi_T(X) = \prod_{j}^{N} \Psi_j(X) \tag{2.20}$$

We should note that in this representation there is no take into account the symmetric properties of the fermions, that meaning the wave function can not model the system in a currency way. The good notice is for a system of two electrons can be demonstrated does not change the energy ground state of the system [4], but we need to take care if we increase the number of the electrons.

Note, as well that the visible variables represent the movement of particles. The partition function is given by

$$Z = \int \int e^{-\frac{1}{T_0} E(x,h)} dx dh \tag{2.21}$$

Herein E shown us the relation between the visible and hidden nodes.

2.1.6 Gradient Descent

Gradient descent is an optimization of the first derivative of a function. Unlike the project 1, where we only have optimized one parameter, here we must optimize several parameters $\alpha = \{a, b, w\}$. These parameters increase rapidly for very large systems.

Through of this method, the expectation value of the energy local is minimized.

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta E(\theta_i) \tag{2.22}$$

For our case the equation is given by

$$\alpha_{i+1} = \alpha_i - \gamma \nabla_\alpha \langle E_L(\alpha_i) \rangle \tag{2.23}$$

where γ is the learning rate.

2.1.7 Blocking Method

Blocking method is a tool to perform a statistical analysis of the information obtained. The basic idea is to separate our data in different blocks and calculating the mean of each block. Then we can calculate the total mean and variance. This method we us allow knowing how accurate are our results, giving an estimation of the error.

The blocking method has become a standard technique for estimating variance $V(\hat{\theta})$ for a specific estimator $\hat{\theta}$, specifically $\hat{\theta} = \overline{X}$.

Assume $n = 2^d$ for some integer $d > 1$, and consider $X_1, X_2, \dots, X_n$ as a stationary time series. Furthermore, assume that the time series is asymptotically uncorrelated. We use vector notation by arranging $X_1, X_2, \dots, X_n$ into a n-tuple $\hat{X} = (X_1, X_2, \cdots, X_n)$.

We now define the blocking transformations. Starting with the vector $\vec{X}$, we form a new vector $\vec{X}_1$ by taking the mean of each subsequent pair of elements. By continuing this process with $\vec{X}_1$ taking the mean of each subsequent pair of its elements, we obtain $\vec{X}_2$. This process is repeated iteratively. Define $\vec{X}_i$ recursively by:

$$(\vec{X}_{i+1})_k \equiv \frac{1}{2}\Big((\vec{X}_i)_{2k-1} + (\vec{X}_i)_{2k}\Big) \qquad \text{for all} \qquad 1 \leq i \leq d-1 \tag{2.24}$$

The quantity $\vec{X}_k$ is subject to k blocking transformations. We now have d vectors $\vec{X}_0, \vec{X}_1, \cdots, \vec{X}_{d-1}$ containing the subsequent averages of observations. It turns out that if the components of $\vec{X}$ is a stationary time series, then the components of $\vec{X}_i$ is a stationary time series for all $0 \leq i \leq d-1$.

Next, we compute the autocovariance, variance, sample mean, and number of observations for each i. Let γ_i, σ_i^2, and $\overline{X}_i$ represent the autocovariance, variance, and average of the elements of $\vec{X}_i$, respectively, and let n_i denote the number of elements in $\vec{X}_i$. By induction, we find that $n_i = n/2^i$.

We can obtain variance alter blocking transformation (V) fulfills that $V(\overline{X}_i) = V(\overline{X}_0) = V(\overline{X})$ for all $0 \leq i \leq d-1$ where $\overline{X}_i$ is referred to the mean between X_i. As a consequence,

$$V(\overline{X}) = \frac{\sigma_k^2}{n_k} + e_k \qquad \text{for all} \qquad 0 \leq k \leq d-1. \tag{2.25}$$

where e_k is called the truncation error $e_k = \frac{2}{n_k}\sum_{h=1}^{n_k-1}\left(1 - \frac{h}{n_k}\right)\gamma_k(h)$, while $h = |i-j|$ is related with the distribution of the data.

2.2 Code implementation

In the code implementation, we use some approximation to construct the fermions wave function, we drop the antisymmetric properties, that means the result work very well for two electrons, if we want extended for more than two fermions we have to make some modification. On the other hand, for the bosons the perfect can be extended until the computation tool allows it.

The code description can be found in github repository. here you will find the enteri progran with some jopyter notebook to facilitate the undertanding as it can be used.

2.3 Results and analysis

Herein, we will to make an analysis of the data result. In all calculations, we have used 10^4 Monte Carlo cycles with 0.01 as time step for the importance sampling. Furthermore, the number of sampling is 2^{10} to do a statistic analysis using a blocking method. The reason we used blocking lies in the fact in Monte Carlo simulations successive samples are often correlated, especially if configurations are generated using a Markov Chain Monte Carlo (MCMC) method. These correlations can distort statistical analysis and lead to underestimation of errors.

Note when in the result talking about analytic energy is referred to the energy without interaction, the simple harmonic oscillator.

2.3.1 Metropolis algorithm (Force brute)

For the first simulation with two hidden variables, where the case is for a system with one particle and one-dimensional without coulomb interaction, besides the analytical energy is 0.5 a.u. From the **Figure** 3 we can notice that the convergence is faster when we increase the learning rate. If we zoom the **Figure** 3, we can notice that the learning rate $\gamma = 0.4$ is better than the others (**Figure** 4).

From **Table** 1, We can notice that changes in hidden variables and learning rates only affect the variance. It also allows us to notice that when we increase the number of hidden variable, the time consumption increases.

So that the report is not too long, here we will only show the plot for the learning rate $\gamma = 0.4$ with different hidden variables (**Figure** 5). Others plots for different learning rates and hidden variables will be included in github repository.

γ	Hidden	Mean $\langle E_L \rangle$ (a.u)	σ^2	CPU time (min)
0.001	1	0.50	1.95×10^{-13}	2.21
0.01	1	0.50	4.68×10^{-16}	2.10
0.1	1	0.50	3.63×10^{-20}	2.12
0.2	1	0.50	3.77×10^{-20}	2.08
0.4	1	0.50	2.45×10^{-19}	2.12
0.8	1	0.50	1.03×10^{-19}	2.10
0.001	2	0.50	2.60×10^{-13}	2.75
0.01	2	0.50	3.39×10^{-14}	2.72
0.1	2	0.50	8.67×10^{-20}	2.67
0.2	2	0.50	1.74×10^{-19}	2.62
0.4	2	0.50	1.73×10^{-20}	2.80
0.8	2	0.50	5.09×10^{-19}	2.70
0.001	4	0.50	1.79×10^{-12}	3.90
0.01	4	0.50	1.95×10^{-13}	3.82
0.1	4	0.50	1.63×10^{-19}	3.81
0.2	4	0.50	6.74×10^{-19}	3.84
0.4	4	0.50	7.48×10^{-19}	3.81
0.8	4	0.50	4.06×10^{-19}	3.85
0.001	6	0.50	1.11×10^{-11}	5.44
0.01	6	0.50	1.07×10^{-12}	5.86
0.1	6	0.50	6.50×10^{-19}	4.92
0.2	6	0.50	1.78×10^{-18}	6.15
0.4	6	0.50	5.08×10^{-20}	8.72
0.8	6	0.50	8.34×10^{-19}	4.99
0.001	10	0.50	8.57×10^{-13}	8.39
0.01	10	0.50	3.71×10^{-13}	7.69
0.1	10	0.50	1.90×10^{-18}	6.97
0.2	10	0.50	9.25×10^{-18}	7.37
0.4	10	0.50	4.21×10^{-18}	7.61
0.8	10	0.50	6.92×10^{-18}	7.11

Table 1: Results for a system with $N = 1$ and $D = 1$ without interaction, where the analytical energy is 0.5 a.u. Herein, we also include the calculations for different hidden variables. We have used 100 iterations.

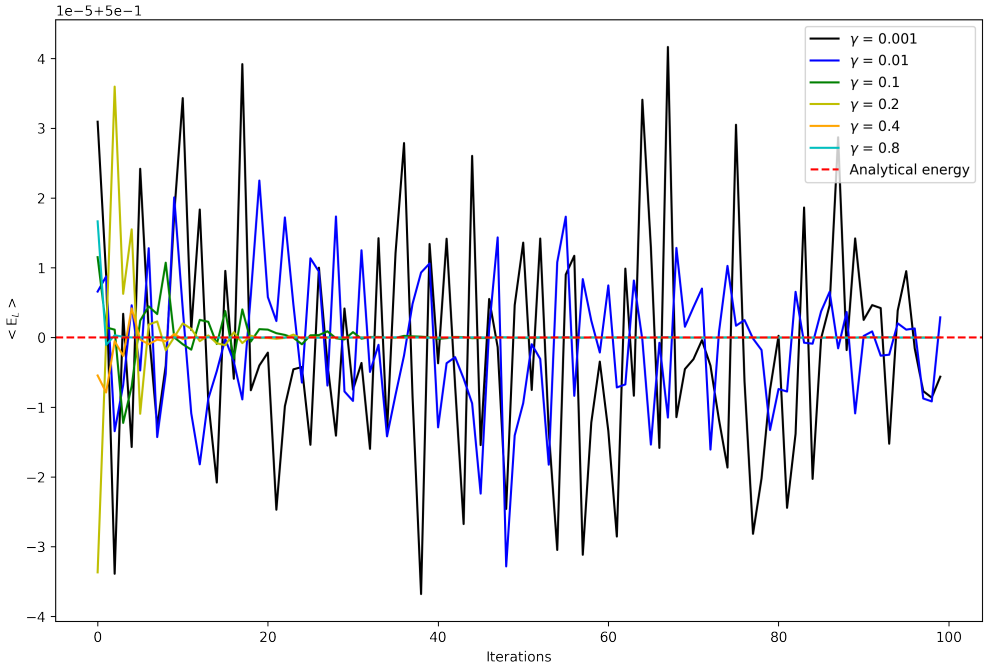


Figure 3: Comparison between different learning rates for a system with $N = 1$, $D = 1$ without interaction and 2 hidden variables.

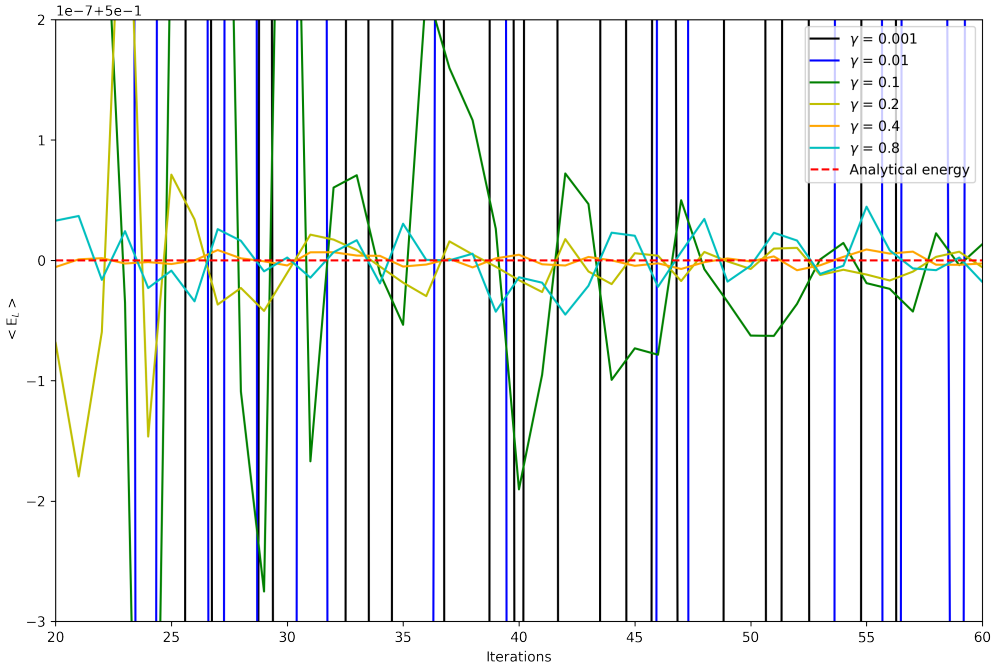


Figure 4: Zoom to the **Figure** 3.

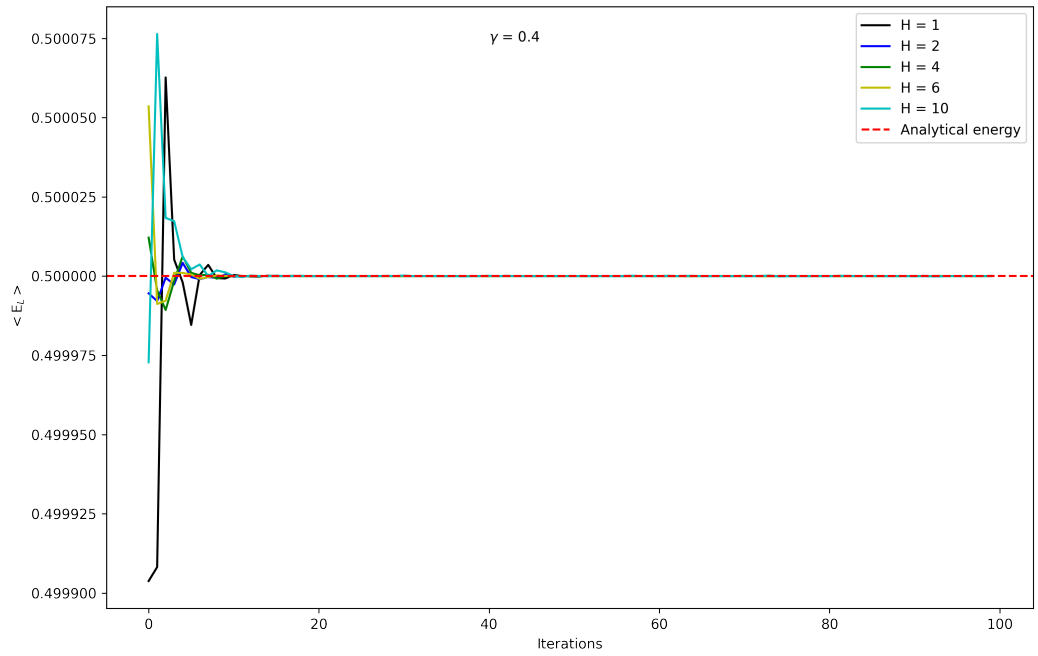


Figure 5: Comparison to $\gamma = 0.4$ with different hidden variables.

For this part, we will also show the case for two particles and two-dimensions with interaction. The reason for this calculation is to be able to compare it with the case where we will use the Important Sampling and parse the accuracy of the calculations. The results are shown in **Table 2**. The plots for this system can be found in github repository.

γ	Hidden	Mean $\langle E_L \rangle$ (a.u)	σ^2	CPU time (min)
0.001	1	3.24	1.91×10^{-6}	4.27
0.01	1	3.24	1.56×10^{-6}	4.06
0.1	1	3.23	1.37×10^{-6}	4.32
0.2	1	3.23	1.25×10^{-6}	4.19
0.4	1	3.23	1.29×10^{-6}	4.34
0.8	1	3.23	1.27×10^{-6}	4.25
0.001	2	3.24	2.04×10^{-6}	5.86
0.01	2	3.24	2.16×10^{-6}	5.83
0.1	2	3.23	1.48×10^{-6}	5.72
0.2	2	3.23	1.44×10^{-6}	5.69
0.4	2	3.23	1.01×10^{-6}	5.87
0.8	2	3.23	1.25×10^{-6}	5.86
0.001	4	3.24	1.91×10^{-6}	8.43
0.01	4	3.24	1.49×10^{-6}	8.80
0.1	4	3.23	1.33×10^{-6}	8.70
0.2	4	3.23	1.45×10^{-6}	8.70
0.4	4	3.23	1.27×10^{-6}	8.66
0.8	4	3.23	1.15×10^{-6}	8.63
0.001	6	3.24	1.80×10^{-6}	12.02
0.01	6	3.24	1.74×10^{-6}	11.84
0.1	6	3.24	1.46×10^{-6}	11.65
0.2	6	3.23	1.52×10^{-6}	11.37
0.4	6	3.23	2.01×10^{-6}	11.70
0.8	6	3.23	1.23×10^{-6}	11.27
0.001	10	3.24	2.24×10^{-6}	16.78
0.01	10	3.24	1.89×10^{-6}	16.85
0.1	10	3.23	1.33×10^{-6}	16.82
0.2	10	3.23	1.20×10^{-6}	16.91
0.4	10	3.23	1.31×10^{-6}	16.74
0.8	10	3.23	1.27×10^{-6}	16.72

Table 2: Results for a system with $N = 2$ and $D = 2$ with interaction, where the analytical energy is 3 a.u. Herein, we also include the calculations for different hidden variables. We have used 100 iterations.

2.3.2 Importance sampling

We will now add the importance of sampling to the above calculations. Our calculations will be for a system with two particles and two-dimensions with interaction, where their analytical energy is 3 a.u.

Table 3 shows us the results for different hidden variables. We can notice that changes in hidden variables and learning rates do not have a great impact on the mean expectation value of the local energy.

From **Figure** 6, we can see the oscillation of the values with a larger energy with respect to the analytical energy for all learning rates, which it is expected because the energy increase as consequence of interaction part (Coulomb repulsion). It can also be seen in **Figure** 7. The plots

for the others cases show a same analysis, they will be included in github repository.

γ	Hidden	Mean $\langle E_L \rangle$ (a.u)	σ^2	CPU time (min)
0.001	1	3.26	7.00×10^{-6}	6.30
0.01	1	3.26	6.53×10^{-6}	6.54
0.1	1	3.25	6.46×10^{-6}	6.82
0.2	1	3.24	4.65×10^{-6}	6.64
0.4	1	3.25	4.26×10^{-6}	6.37
0.8	1	3.24	4.88×10^{-6}	6.32
0.001	2	3.27	7.53×10^{-6}	8.69
0.01	2	3.26	7.08×10^{-6}	8.62
0.1	2	3.25	4.80×10^{-6}	8.93
0.2	2	3.24	4.27×10^{-6}	8.58
0.4	2	3.24	4.84×10^{-6}	8.56
0.8	2	3.24	4.57×10^{-6}	8.51
0.001	4	3.26	7.16×10^{-6}	12.75
0.01	4	3.26	8.00×10^{-6}	12.81
0.1	4	3.26	6.56×10^{-6}	12.57
0.2	4	3.24	3.27×10^{-6}	12.63
0.4	4	3.24	4.35×10^{-6}	12.66
0.8	4	3.24	4.70×10^{-6}	12.69
0.001	6	3.26	8.62×10^{-6}	17.56
0.01	6	3.26	7.69×10^{-6}	17.28
0.1	6	3.26	6.98×10^{-6}	16.56
0.2	6	3.24	4.43×10^{-6}	16.63
0.4	6	3.24	5.57×10^{-6}	16.69
0.8	6	3.24	4.78×10^{-6}	16.64
0.001	10	3.26	7.06×10^{-6}	24.36
0.01	10	3.26	7.16×10^{-6}	24.68
0.1	10	3.26	7.32×10^{-6}	26.14
0.2	10	3.24	4.83×10^{-6}	25.33
0.4	10	3.24	4.54×10^{-6}	24.85
0.8	10	3.24	3.98×10^{-6}	24.94

Table 3: Results for a system with $N = 2$ and $D = 2$ with interaction, where the analytical energy is 3 a.u. We have used 100 iterations.

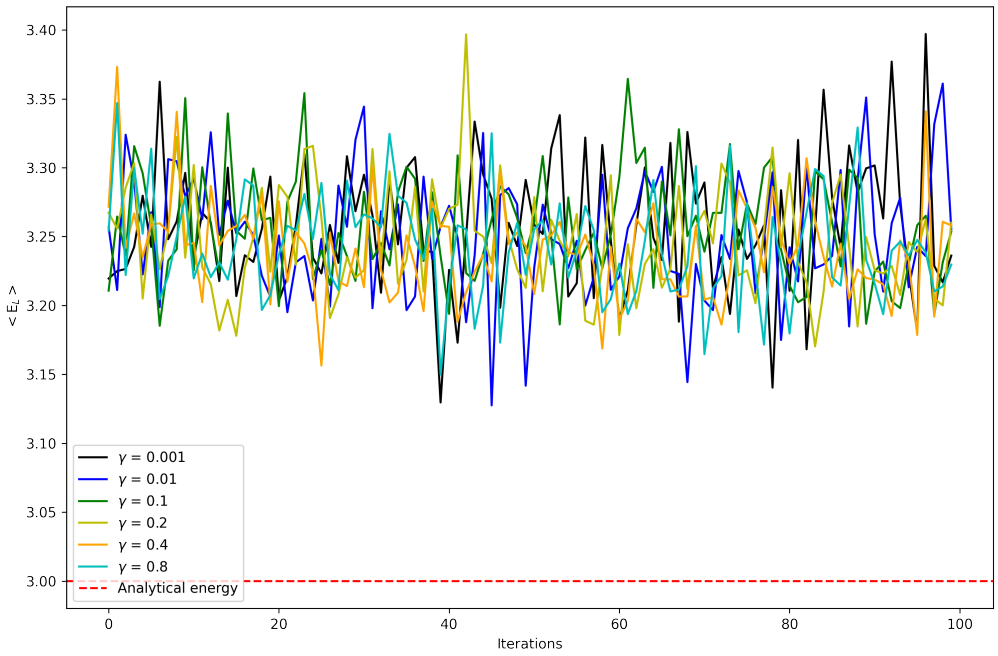


Figure 6: Comparison between different learning rates for a system with $N = 2$, $D = 2$ with interaction and 2 hidden variables.

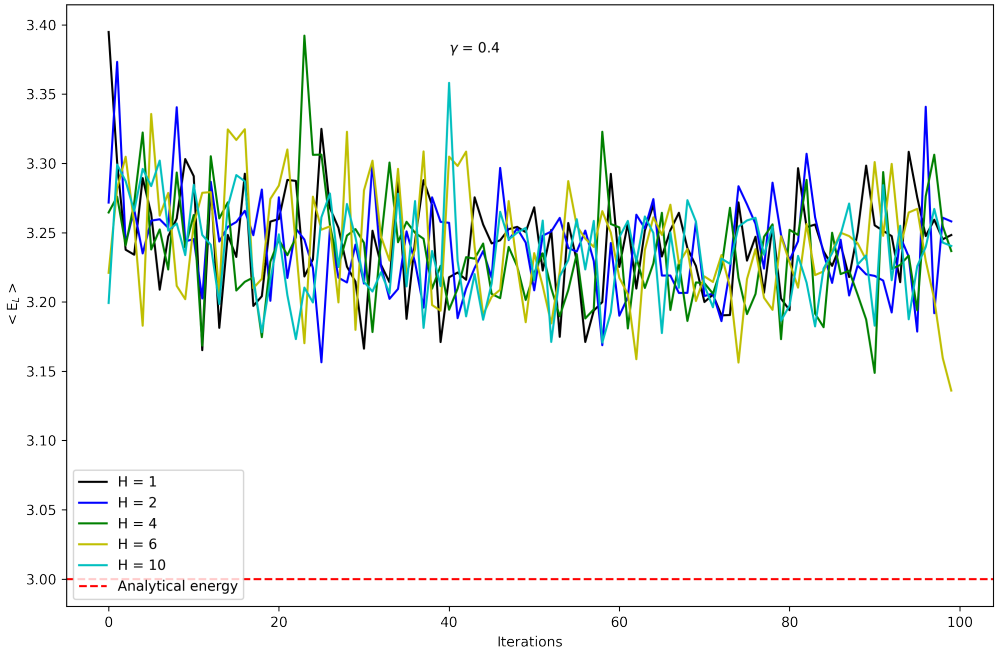


Figure 7: Comparison to $\gamma = 0.4$ with different hidden variables.

2.3.3 Comparison

Herein, we will compare the results between force brute and importance sampling for the case where the system has $N = 2$ and $D = 2$ with interaction. We will use the data from the previous calculations.

From **Figure** 8, we can notice that there is some stability in the results for both methods, in general cases the importance sampling has a better behaviour, but this has a trick because depended on the learning rate use. for this instance, we are not able to affirm which has better response.

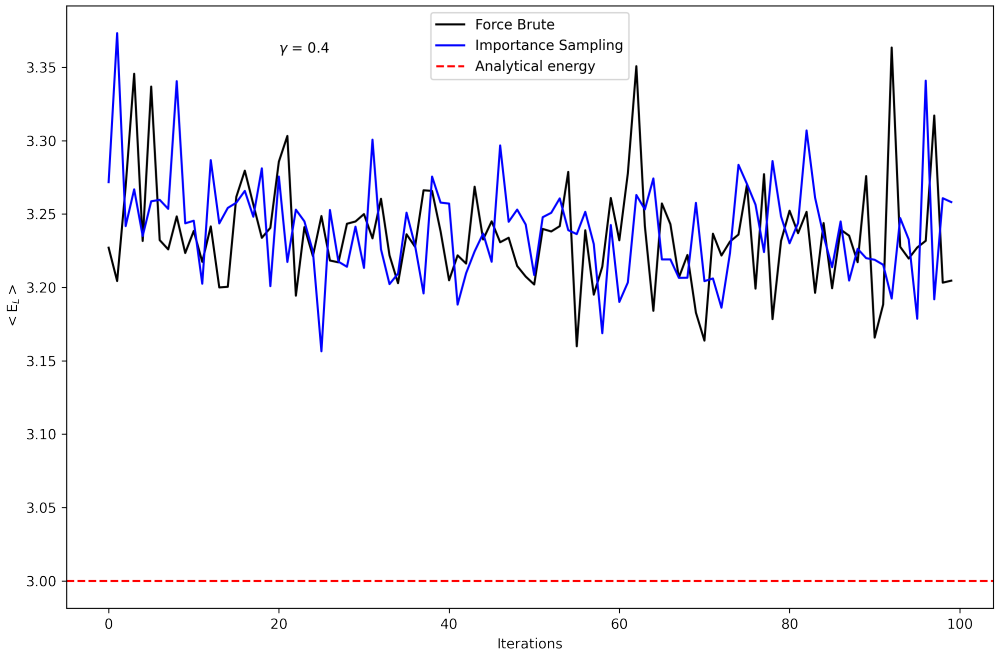


Figure 8: Comparison between the force brute and importance sampling for a system with $N = 2$ and $D = 2$ with interaction. The number of hidden variables is 2 and $\gamma = 0.4$.

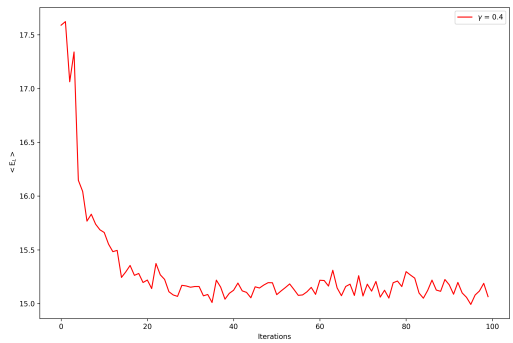
2.3.4 Extending for bosons

Herein we will extend the system for many bosons. We will use a two-dimensional case with 4 hidden variables, $\gamma = 0.4$ and including the interaction. The results can be seen in **Table** 4.

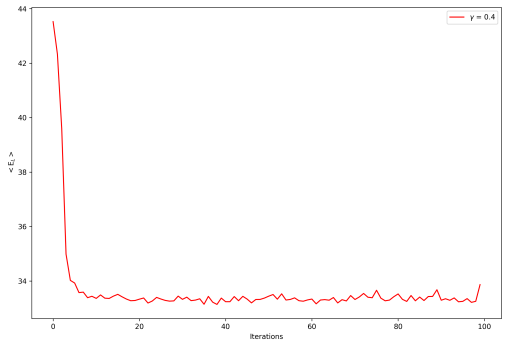
N	Mean $\langle E_L \rangle$ (a.u)	σ^2	CPU time (min)
5	15.16	1.63×10^{-5}	30.93
8	33.89	2.08×10^{-5}	51.64
10	48.76	5.16×10^{-5}	64.76
20	160.05	3.1×10^{-4}	154.78

Table 4: Results for a system with many bosons. We have used 100 iterations.

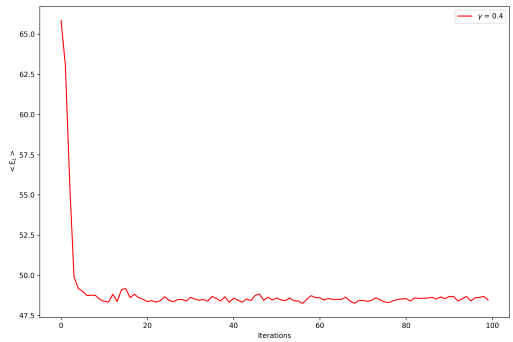
The plots for the information from **Table** 4 is shown in the **Figures** 9a, 9b, 9c and 9d.



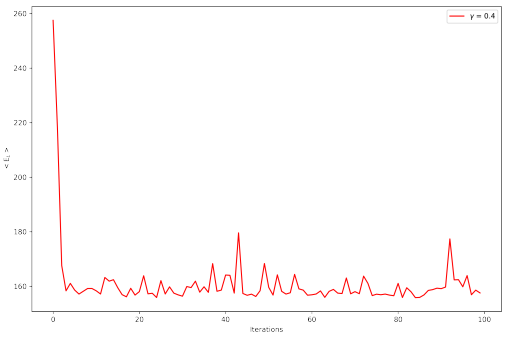
(a) System with $N = 5$ and $D = 2$.



(b) System with $N = 8$ and $D = 2$.



(c) System with $N = 10$ and $D = 2$.



(d) System with $N = 20$ and $D = 2$.

Figure 9: Different systems with interaction. The number of hidden variables is 4 and $\gamma = 0.4$.

2.4 Some important discussions

For our initial calculations, where the system is composed of one to two fermions, using the metropolis algorithm (force brute) and importance sampling, we are able to estimate a good results since they reached the convergence value. They also showed a good stability. without and with interaction.

In cases where we have included many bosons, the time consumption also increases. for the largest bosons configuration, the stability became decrease. that can be improved by increasing the number of hidden layer or a more precise learning rate.

2.5 Overview of Boltzmann Machines

In many-body physics, BMs have become a powerful tool for modelling and understanding complex quantum systems. These neural networks are particularly adept at capturing the intricate correlations and interactions among a large number of particles. By leveraging their ability to represent complex probability distributions, BMs can efficiently approximate the ground state wave functions of quantum systems, even in high-dimensional spaces [5]. This is particularly useful in studying phenomena like quantum phase transitions, where traditional methods may struggle due to the exponential growth of the state space with the number of

particles [6]. Additionally, BMs can be employed to model thermal states, providing insights into the thermodynamic properties of many-body systems. Their flexibility and adaptability make them valuable in simulating and exploring a wide range of physical systems, from strongly correlated electron systems to lattice models in condensed matter physics [1, 7, 8].

3 Quantum Boltzmann Machines (QBMs)

The Quantum Boltzmann Machines (QBMs) is a type of computer model that uses the rules of quantum mechanics to improve how it learns and processes information, that meaning the QBMs is hastily to say a translation from BMs in the quantum world or quantum computing scheme. It makes them faster and more efficient at finding solutions. This helps them solve complex problems and perform tasks in machine learning more quickly and accurately [9].

In this section, we will argue about the QBMs, making a theoretical background, some analysis with Qiskit packages and import comments.

3.1 Formalism

In this place, we are going to make an introduction to theoretical QBMs, stating by an introduction a quantum computing, and then a variational QBMs.

3.1.1 A short introduction to quantum computing

The quantum computing (QC) is an advanced area of computing that leverages the principles of quantum mechanics to process information in fundamentally different ways from classical computers. The QC is well known, it uses quantum bit (qubit) $|0\rangle$ and $|1\rangle$ where are called ket zero and one respectively instead of simplest classic bit 0 and 1. The qubit has a big different from a classic bit, the first can be in so-called superposition in the most general case is represented by $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where α and β are in general complex by postulate of quantum mechanic have to be normalized $\|\alpha\|^2 + \|\beta\|^2 = 1$. A qubit normally is shown in the geometric representation (see **Figure** 10) in the known Block sphere.

In the qubit there are "many" state in fact there are infinite (continuous), if it thinks each state as information meaning that the qubit can "process" more that the simplest classic bit just can be in two state 0 or 1 [10]. This is not the only advantage, there are more like a reversibility, entanglement between other [11].

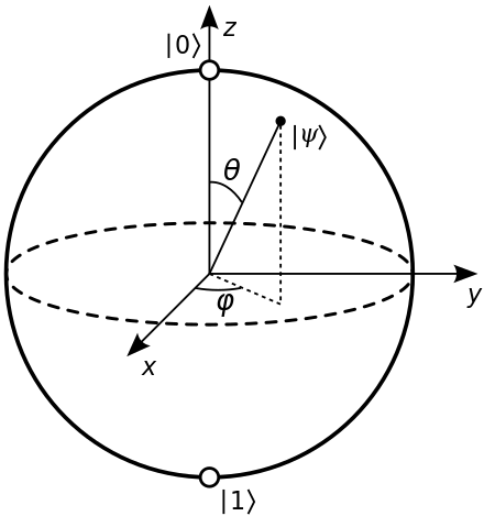


Figure 10: The geometrical representation of a qubit in the Block sphere. A qubit can take a position anywhere on the Bloch sphere of the following form $|\Psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$.

It is a vector representation of one qubit, but if we have more than one $|\Psi\rangle$ becomes easier to use a matrix representation and other import reason is because in QC the operator (quantum gate) as quantum mechanic could be represented as matrix. For instance one qubit,

$$|\Psi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} \tag{3.1}$$

where in this representation it easy to see that

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \qquad \text{and} \qquad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

for many qubits the full states is defined as a tensor product of the different single qubit state,

$$|\Psi\rangle = |\Psi_0\rangle \otimes |\Psi_1\rangle \cdots \otimes |\Psi_{n-1}\rangle \equiv |\Psi_0\Psi_1 \cdots \Psi_{n-1}\rangle \tag{3.2}$$

for example a system of two qubits, with $|\Psi_0\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ and $|\Psi_1\rangle = \beta_0|0\rangle + \beta_1|1\rangle$

$$|\Psi\rangle = |\Psi_0\rangle \otimes |\Psi_1\rangle = \begin{bmatrix}\alpha_0\beta_0\\ \alpha_0\beta_1\\ \alpha_1\beta_0\\ \alpha_1\beta_1\end{bmatrix} = \alpha_0\beta_0\begin{bmatrix}1\\0\\0\\0\end{bmatrix} + \alpha_0\beta_1\begin{bmatrix}0\\1\\0\\0\end{bmatrix} + \alpha_1\alpha_0\begin{bmatrix}0\\0\\1\\0\end{bmatrix} + \beta_1\beta_1\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{3.3}$$

the **Equation** 3.3 can be written in more compact way as

$$|\Psi\rangle = \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle \tag{3.4}$$

then

$$|00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad , \quad |01\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad , \quad |10\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \text{and} \quad |11\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

3.1.2 Quantum circuit and gate

A quantum circuit is a scheme that makes possible to perform computation in terms QC, it is analogous to a classical circuit to make in classical computation instead of uses bit and classical gates it works with qubits and quantum gates [11].

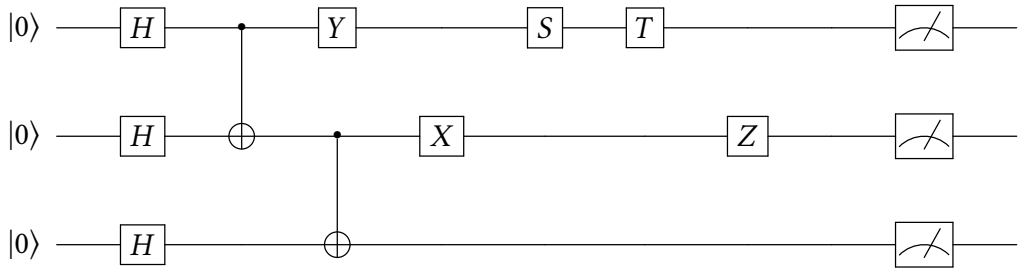


Figure 11: Scheme of a quantum circuit and also the common quantum gate.

the quantum gate has been mentioned, but it has not been defined. What is a quantum gate? the quantum gate is the form to build blocks of quantum algorithms, enabling tasks such as state preparation, entanglement creation and operation over qubits. Common examples include the Hadamard gate, Pauli gates (X, Y, Z), phase gates (S, T), and the CNOT gate, each essential for different quantum computational processes, in the **Table** 5 show a resume of them.

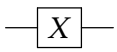
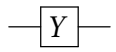
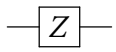
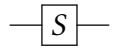
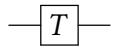
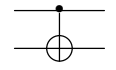
Name	Symbol	Matrix Representation
Pauli-X (NOT)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard-H		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase-S		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi/4} \end{bmatrix}$
T		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
CNOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Table 5: The most common quantum gate, name, symbol of the circuit representation and matrix representation

Now it is illustrated as some quantum gate work presenting a one qubit X, Y, Z, H and two qubits CNOT operation gates,

$$X|\Psi\rangle = \beta|0\rangle + \alpha|1\rangle \quad [3.5]$$

$$Y|\Psi\rangle = -i(\beta|0\rangle - \alpha|1\rangle) \qquad Z|\Psi\rangle = \alpha|0\rangle - \beta|1\rangle \quad [3.6]$$

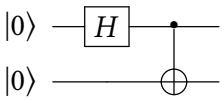
$$H|\Psi\rangle = \alpha\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \beta\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \quad [3.7]$$

In words X-gate switch the probability amplitudes, Y-gate introduce the imaginary unit and switch the amplitudes, Z-gate switch this sign. H-gate make a superposition.

On the other hand, the CNOT is a two qubits operation gate, it is very import because normally is used to make an entanglement state, this works

$$\text{CNOT}(|x\rangle \otimes |y\rangle) \longrightarrow |x\rangle \otimes |y + x\rangle \quad [3.8]$$

where x and y are binary numbers. In words, the CNOT-gate as name indicate controlled one qubit with respect to the state of the other one. The entanglement state is made in two-step. First applying a H-gate in one of the qubits by convention in the first qubit, creating a superposition, second step is a CNOT-gate. These states are very import in QC and have a special name called Bell State $|\Phi^{\pm}\rangle$.



$$|\Phi^{\pm}\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle) \qquad [3.9]$$

There are other very import quantum gate in special when it is talked about variational quantum algorithms (VQA) [8, 12]. It is the rotation gate R_x, R_y, and R_z allow for any single-qubit to make a unitary transformation, then combining these rotations it can achieve any possible single-qubit state on the Bloch sphere (see **Figure** 10).

Name	Symbol	Matrix Representation
Rotation-X	$R_x(\theta_1)$	$\begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$
Rotation-Y	$R_y(\theta_2)$	$\begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$
Rotation-Z	$R_z(\theta_3)$	$\begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$

Table 6: Rotation quantum gates

3.1.3 Variational quantum Boltzmann machines (VarQBMs)

A QBMs are defined by a general parameterized Hamiltonian H_θ,

$$H_\theta = \sum_{i=0}^{p-1} \theta_i h_i \qquad \longrightarrow \qquad h_i = \otimes_{j=0}^{n-1} \sigma_{j,i} \qquad [3.10]$$

where θ_i is a vector with dimension $\mathbb{R}^p$, and $\sigma_{j,i} \in I, X, Y, Z$ acting on the j^{th} qubit. Normally, the QBMs are represented by so-called Ising model, which it is a direct way to map onto it [13, 14]. QBM is not limited, in principle can be applied to any type of Hamiltonian compatible with Boltzmann distribution doing a Jordan-Wigner transformation or normally called decomposition [15].

In comparison with BMs, the visible layer in QBM consists of qubits that represent the input data directly, capturing the observable features of the dataset. On the other hand, The hidden layer is interpreted as qubits that are not directly observable but interact with the visible qubits, and weight is the Hamiltonian itself because this makes possible the interaction between visible and hidden layer [16].

the probability to measure a configuration v of the visible qubits is defined with respect to a projective measurement $\Lambda_v = |v\rangle\langle v| \otimes I$ using quantum Gibbs distribution,

$$\rho^{\text{Gibbs}} = \frac{e^{-H_\theta/k_B T}}{Z} \qquad [3.11]$$

with $Z = \text{Tr}(\frac{e^{-H_\theta/k_B T}}{Z})$, for instance, the probability to measure $|v\rangle$ is calculated as

$$P_v^{QBM} = \text{Tr}(\Lambda_v \rho^{\text{Gibbs}}) \qquad [3.12]$$

where Λ_v refers to projective measurements with respect to the computational basis of the visible qubits.

To train the Hamiltonian parameters θ_i in QBM, our objective is to adjust these parameters so that the sampling probabilities of the corresponding Gibbs state $\rho^{\text{Q-Gibbs}}$ accurately reflect the probability distribution of the given classical training data. It is needed to minimize the so-called loss function or objective function,

$$L = -\sum_v p_v \log(P_v^{\text{QBM}}) \tag{3.13}$$

For efficient training, we need to calculate the derivative of with respect to the Hamiltonian parameters. Unlike current QBM implementations, VarQBM enables the use of analytic gradients for the loss function (**Equation** 3.13). To make this processed is normally used an intermediate step called variational quantum imaginary time evolution (VarQITE). VarQITE is important in VarQBMs because it provides an approximation to the Gibbs distribution, but also a robust and efficient method for parameter optimization, improving convergence process. we will not go deep in it, but for complete view (see the reference [8]). In the figure **Figure** 12 shows scheme of different processes,

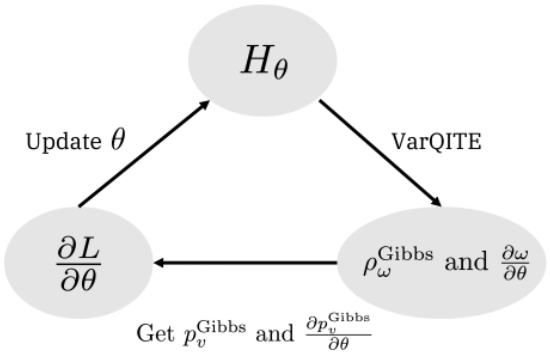


Figure 12: The VarQBM training of different process. It is needed to fix the Pauli terms for H_θ and choose initial suggestion for θ. Then, distribution Gibbs states $\rho_\omega^{\text{Gibbs}}$ are approximated using VarQITE, and end we can find $\frac{\partial L}{\partial \theta}$ to make an optimization of the Hamiltonian parameters with a classical optimizer like in the BMs [8].

the gradient of the loss function since a data tray (p^{data}) is given for

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial}{\partial \theta_i}\left(-\sum_v p_v^{\text{data}} \log(P_v^{\text{QBM}})\right) = -\sum_v p_v^{\text{data}} \frac{\partial P_v^{\text{QBM}}/\partial \theta_i}{P_v^{\text{QBM}}} \tag{3.14}$$

using in **Equation** 3.14 that,

$$\frac{\partial P_v^{\text{QBM}}}{\partial \theta_i} = \sum_k \frac{\partial P_v^{\text{QBM}}}{\partial \omega_k(\tau)} \frac{\partial \omega_k(\tau)}{\partial \theta_i} \tag{3.15}$$

where $\frac{\partial P_v^{\text{QBM}}}{\partial \omega_k(\tau)} = \frac{\partial}{\partial \omega_k(\tau)} \text{Tr}(\Lambda_v \rho_\omega^{\text{Gibbs}})$ and $\frac{\partial \omega_k(\tau)}{\partial \theta_i}$ is evaluated using VarQITE (see [8]). An import point in the VarQITE is used to approximate this evolution within a variational subspace

spanned by a parameterized trial state. The parameters are adjusted to minimize the norm of the difference between the exact and variational evolution. In rash way, the QBMs is a way to rewrite the classical BMs doing some transformation (decomposed) that making possible a suitable condition to implement it in a quantum computing.

In the quantum neural network (QNN), the general circuit implementation can be represented in the **Figure** 13, which show the general process for a successfully implemtation of it.

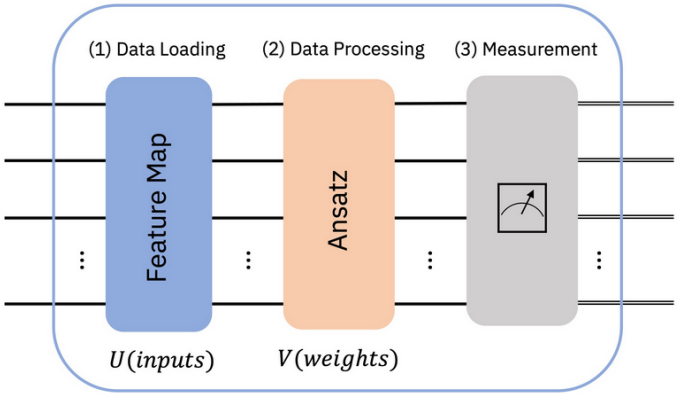


Figure 13: The figure shown the 3 principal processes in the QNN, the first step is to pass the input data. The second is about the trynig processe which can made using a identical proceses like VarQBMs. Third is just about the mesaure the qubit satate. This also show U (inputs) the "visible layer" and W (weights) wich is related with the energy and conect the "visible layer" with "hidden layer".

3.2 Code implementation

The codes for both examples have been taken from Qiskit Machine Learning and Prabh Simran's repository.

The code for the first case have been taken in its entirety, without modifications, that is why we cite the link here. Explication about the workflow will be given in Results and analysis section.

The code for the second case only had to be updated according to the pyquil package recommendations. It also necessary the installation of Rigetti Quantum Virtual Machine (QVM). You can check the updated version in github repository.

3.3 Results and analysis

Since the implementation of the code for QBM or special cases like Quantum restricted Boltzmann machine (QRBM) requires starting from the scratch, which is a challenging, we have decided to use packages from qiskit and github repositories. Implementation is not an impossible challenge, but it requires a lot of time.

The experiments analyzed have been taken from Qiskit Machine Learning and Prabh Simran's repository. The analysis in this project will be given based on our understanding of the process.

3.3.1 Using a real database

Herein we will work with a real database obtained from scikit-learn. If we check the database we can see that it has 4 features, which will define the number of qubits. We should also

mention that for comparison purposes these data will be divided into two groups: a training dataset and a test dataset.

Our first step will be to ensure that the data is encoded as qubits, this process is refer as data loading and it will be worked by the feature map. This process is unique of the quantum machine learning.

This is where the parameterized quantum circuit comes into play, where we must to into account the weights (parameters). These weights will be optimized for minimize the objective function. So far, if we compare it with the workflow of project 1 we can see that is similar.

For this example will use the ZZFeatureMap as feature map, which is represented in **Figure** 14.

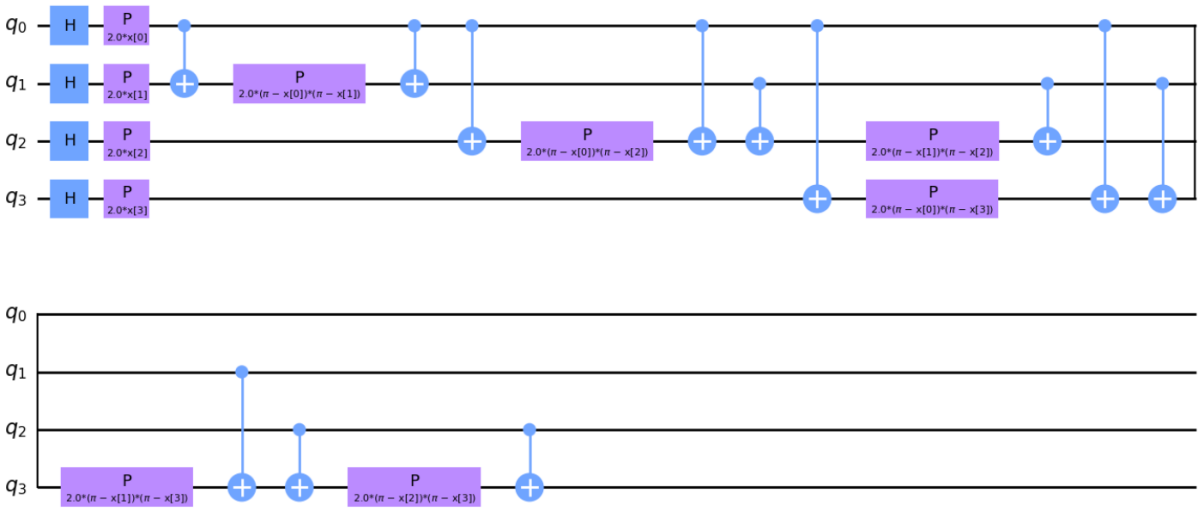


Figure 14: Feature map. The circuit have been decompose.

From **Figure** 14 we can note the weights (parameters), they are four. Now we must define our ansatz, shown in **Figure** 15.

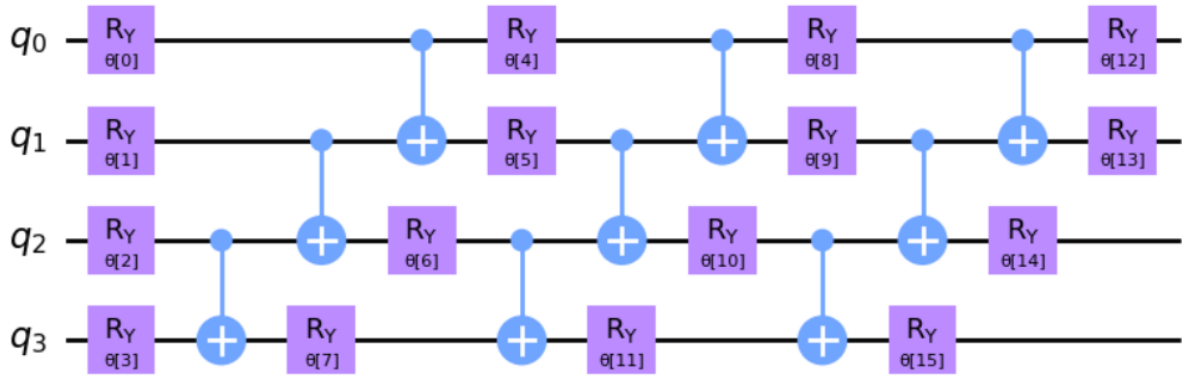


Figure 15: Ansatz for our calculations, composed of rotational and CNOT gates.

All these new parameters in **Figure** 15 are trainable weights of a classifier. The classifier used for this case is a Variational Quantum Classifier (VQC).

The function of the VQC is to build a quantum neural network using the previous ingredients (feature map and the ansatz). Now we can train our weights through a variational manner.

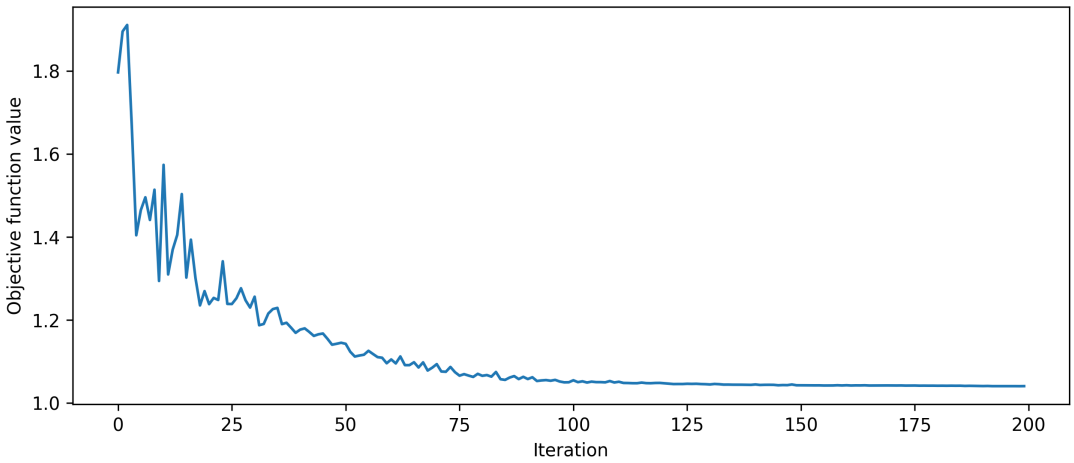


Figure 16: Feature map. The circuit have been decompose.

From **Figure** 16 we can note that the objective function value decrease rapidly. If we compare our results of both groups (**Table** 7), we can note that the value of the trained group fits quite well with the value of the test group, which shows us that the model can predict or find hidden data.

Training group	Test group
0.87	0.80

Table 7: Results for both groups.

3.3.2 Coin Flip Experiment

A special and exciting case due to its simplicity is the coin flip, since finding either side (heads or tails) has the same probability. We can apply this to training with a QRBM.

Herein the heads will be represented as 0, while the tails as 1. This will help us generate an underlying hidden distribution of our data. It is also necessary to generate artificial data where we will hide information that the QRBM algorithm must decode. That information will be encoded within a set of four numbers, for example: 0 will be represented as $(1, 1, -1, -1)$, while 1 as $(-1, -1, 1, 1)$. From the previous information we can notice that we will have to consider 4 visible variables, while only 1 hidden variable. Furthermore, It is also worth mentioning that probabilities can be calculated through a quantum circuit (Bell state) shown in **Figure** 17.

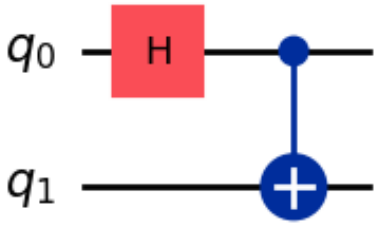


Figure 17: Quantum circuit (Bell state).

Analyzing our example we can note that the model is able to predict either 0 or 1 after 50 epochs. Its prediction is completely accurate, since the probability shown is 1 (**Table** 8).

QRBM Probability	Original Coin Value
0.00	1.00
0.00	1.00
1.00	0.00
0.00	1.00
1.00	0.00
1.00	0.00
1.00	0.00
0.00	1.00
0.00	1.00
0.00	1.00
0.00	1.00
0.00	1.00
1.00	0.00
1.00	0.00
0.00	1.00
1.00	0.00
1.00	0.00
1.00	0.00
1.00	0.00
1.00	0.00

Table 8: Training the weights, after 50 epochs is able to predict the value 0.

3.4 Some important discussions

The previous examples are very important for understanding both the workflow and the ingredients to perform calculations using QBM.

We can see that the vital ingredient for this kind of calculations is get a database, this can be a real database like the first example or a artificial (fictitious) database like the second example.

And then, based on this database, we can take certain characteristics relevant to the study we want to carry out. Another relevant point is the use of classifiers focused on machine learning, here their function is to build the quantum neural network.

It is also possible to make improvements in our training, we could change the feature map and the ansatz for others shown in the libraries or others that we can create, as well as the replacement of the optimizers with much more complex ones. These factors will increase the complexity of the calculation but will give us better results. We must also take into account that the greater the complexity, the consumption time will increase.

4 Conclusions

The BMs show a powerful tool to calculate the ground state of one and two ferions with and without interaction. Which is something very interacting because the wave function construct using BMs has not any physical real condition to use it. For bosons system the result are very equivalent regarding convergence point view of the ground state. The algorithm shown us that can be used to study the correlated many bosson problem using a mix between variational Monte Carlo and Restricted Boltzmann Machine.

On the other hand, the result show for value of learning rate 0.4 is found a better convergence for instance problem, and for no large particles (fermions or bosons) the changed in the number of the hidden layer it is not very important, while for case the large number of bosons can be improved the convergence value. n

On QBM, the complexity of the application on a physical system requires a deep review of its principles and implementation or use of the codes (libraries). Our review through the examples studied shows that quantum neural networks can be implemented into quantum machine learning workflow. Furthermore, the trainable parameters (weights) play a supremely important role to achieve the minimization of the objective function.

A self-criticism of our work, in the QBM section, is not having been able to implement it in a physical system. For example, a very studied model with this tool is the Ising model.

A Local energy

Calculations will be perform using the wave function

$$\psi(x) = F_{rbm} \tag{A.1}$$

Energy local is given by

$$E_L = \frac{1}{\Psi_T} H \Psi_T \tag{A.2}$$

and the hamiltonian is

$$H = \sum_{k=1}^{P} \left(-\frac{1}{2}\nabla_k^2 + \frac{1}{2}\omega^2 r_k^2 \right) + \sum_{k<l} \frac{1}{r_{kl}} \tag{A.3}$$

Therefore, replacing

$$E_L = \sum_{k}^{P} \frac{1}{2} \left(-\frac{1}{\Psi}\nabla_k^2 \Psi + \omega^2 r_k^2 \right) + \sum_{k<l} \frac{1}{r_{kl}} \tag{A.4}$$

Now we must to calculate the laplacian of the wave function, which can write as

$$\nabla_k^2 \Psi = \sum_{l}^{D} \frac{\partial^2}{\partial x_{kl}^2} \Psi. \tag{A.5}$$

replacing

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

As the number of visible nodes is $M = P \times D$, we can written again

$$E_L = \sum_{i}^{M} \frac{1}{2} \left(-\frac{1}{\Psi} \frac{\partial^2}{\partial X_i^2} \Psi + \omega^2 X_i^2 \right) + \sum_{k<l} \frac{1}{r_{kl}} \tag{A.6}$$

Keep in mind that

$$\Psi(X) = \frac{1}{Z} \exp\left(-\sum_{i}^{M} \frac{(X_i - a_i)^2}{2\sigma^2} \right) \prod_{j}^{N} \left(1 + \exp\left(b_j + \sum_{i}^{M} \frac{X_i \omega_{ij}}{\sigma^2} \right) \right) \tag{A.7}$$

the NQS wave function has exponentials and is in the denominator for the calculation of the local energy. A smart way to solve this situation is consider the $ln\Psi$ using the identity

$$\frac{1}{\Psi} \frac{\partial^2}{\partial X_i^2} \Psi = \left(\frac{\partial}{\partial X_i} \ln \Psi \right)^2 + \frac{\partial^2}{\partial X_i^2} \ln \Psi \tag{A.8}$$

replacing the **Equation** A.8 in **Equation** A.6

$$E_L = \sum_{i}^{M} \frac{1}{2} \left(-\left(\frac{\partial}{\partial X_i} \ln \Psi \right)^2 + \frac{\partial^2}{\partial X_i^2} \ln \Psi + \omega^2 X_i^2 \right) + \sum_{k<l} \frac{1}{r_{kl}} \tag{A.9}$$

and $ln\Psi$ is given by

$$\ln \Psi = -\ln Z - \sum_{i}^{M} \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_{j}^{N} \ln\left(1 + \exp\left(b_j + \sum_{i}^{M} \frac{X_i \omega_{ij}}{\sigma^2} \right) \right) \tag{A.10}$$

Now we must solve the derivatives, the first derivative is given by

$$\begin{aligned}
\nabla_i \ln \Psi &= \nabla_i \left(-\sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^N \ln\left(1 + \exp\left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right)\right)\right) \\
&= \frac{\partial}{\partial X_i}\left(-\sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^N \ln\left(1 + \exp\left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right)\right)\right) \\
&= \frac{-2(X_i - a_i)2\sigma^2}{(2\sigma^2)^2} + \sum_j^N \frac{1}{1 + \exp\left(b_j + \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)} \left(\frac{\omega_{ij}}{\sigma^2}\right) \exp\left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right) \\
&= \frac{-(X_i - a_i)}{\sigma^2} + \frac{1}{\sigma^2} \sum_j^N \frac{\omega_{ij}}{1 + \exp\left(-b_j - \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)}
\end{aligned} \tag{A.11}$$

while the second derivative is

$$\begin{aligned}
\nabla_i^2 \ln \Psi &= \nabla_i^2 \left(-\sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^N \ln\left(1 + \exp\left(b_j + \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right)\right)\right) \\
&= \frac{\partial}{\partial X_i}\left(\frac{-(X_i - a_i)}{\sigma^2} + \frac{1}{\sigma^2} \sum_j^N \frac{\omega_{ij}}{1 + \exp\left(-b_j - \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)}\right) \\
&= -\frac{1}{\sigma^2} + \frac{1}{\sigma^2} \sum_j^N \frac{-\omega_{ij}}{\left(1 + \exp\left(-b_j - \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)\right)^2} \left(\frac{-\omega_{ij}}{\sigma^2}\right) \exp\left(-b_j - \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right) \\
&= -\frac{1}{\sigma^2} + \frac{1}{\sigma^4} \sum_j^N \frac{\left(\omega_{ij}\right)^2}{\left(1 + \exp\left(-b_j - \sum_i^M \frac{X_i\omega_{ij}}{\sigma^2}\right)\right)^2} \exp\left(-b_j - \sum_i^M \frac{X_i \omega_{ij}}{\sigma^2}\right)
\end{aligned} \tag{A.12}$$

B Gradients with respect to RMB parameters

Gradient is define as

$$G_i = \frac{\partial \langle E_L \rangle}{\partial \alpha_i} = 2\left(\langle E_L \frac{1}{\Psi} \frac{\partial \Psi}{\partial \alpha_i} \rangle - \langle E_L \rangle \frac{1}{\Psi} \frac{\partial \Psi}{\partial \alpha_i} \right) \tag{B.1}$$

we must derive with respect to $\alpha = \{a, b, W\}$, using the NQS

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

taking the logarithm

$$\ln(\Psi(\mathbf{X})) = \ln \frac{1}{Z} - \sum_i^M \frac{(X_i - a_i)^2}{2\sigma^2} + \sum_j^N \ln\left[1 + \exp\left(b_j + \sum_i^M \frac{X_i w_{ij}^{\mathrm{T}}}{\sigma^2}\right)\right] \tag{B.3}$$

A useful identity for this calculations is

$$\frac{d}{dx} \ln f(x) = \frac{1}{f(x)} \frac{d}{dx} f(x) \tag{B.4}$$

by calculating we can obtain

$$\begin{aligned}
\frac{\partial \ln(\Psi(\mathbf{X}))}{\partial a_k} &= \frac{1}{\sigma^2}(X_k - a_k) \\
\frac{\partial \ln(\Psi(\mathbf{X}))}{\partial b_n} &= \left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}^{\mathrm{T}}}{\sigma^2}\right)\right)^{-1} \\
\frac{\partial \ln(\Psi(\mathbf{X}))}{\partial w_{kn}} &= \frac{X_k}{\sigma^2}\left(1 + \exp\left(-b_n - \sum_i^M \frac{X_i w_{in}^{\mathrm{T}}}{\sigma^2}\right)\right)^{-1}
\end{aligned} \tag{B.5}$$

References

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

[2] 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.

[3] Guido Montúfar. Restricted Boltzmann Machines: Introduction and Review. *ArXiv*, abs/1806.07066, 2018.

[4] Jules W. Moskowitz and M. H. Kalos. A new look at correlations in atomic and molecular systems. I. Application of fermion monte carlo variational method. *International Journal of Quantum Chemistry*, 20(5):1107–1119, 1981.

[5] Chae-Yeun Park and Michael J. Kastoryano. Geometry of learning neural quantum states. *Phys. Rev. Res.*, 2:023232, May 2020.

[6] Hiroki Saito. Method to Solve Quantum Few-Body Problems with Artificial Neural Networks. *Journal of the Physical Society of Japan*, 87(7):074002, 2018.

[7] Roger G. Melko, Roger G. Melko, Giuseppe Carleo, Juan Felipe Carrasquilla, and Juan Ignacio Cirac. Restricted Boltzmann machines in quantum physics. *Nature Physics*, 15:887 – 892, 2019.

[8] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Variational quantum Boltzmann machines. *Quantum Machine Intelligence*, 3:1–15, 2021.

[9] Mohammad H. Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum Boltzmann Machine. *Phys. Rev. X*, 8:021050, May 2018.

[10] R. Horodecki. Quantum Information. *Acta Physica Polonica A*, 139(3):197–2018, March 2021.

[11] Andrew Steane. Quantum computing. *Reports on Progress in Physics*, 61(2):117, feb 1998.

[12] Onno Huijgen, Luuk Coopmans, Peyman Najafi, Marcello Benedetti, and Hilbert J Kappen. Training quantum Boltzmann machines with the $\beta - variationalquantumeigensolver$. *Machine Learning: Science and Technology*, 5(2):025017, April 2024.

[13] Jing Gu and Kai Zhang. Thermodynamics of the Ising Model Encoded in Restricted Boltzmann Machines. *Entropy*, 24(12), 2022.

[14] Rodrigo Veiga and Renato Vicente. Restricted Boltzmann Machine Flows and The Critical Temperature of Ising models, 2022.

[15] C. D. Batista and G. Ortiz. Generalized Jordan-Wigner Transformations. *Phys. Rev. Lett.*, 86:1082–1085, Feb 2001.

[16] Yuta Shingu, Yuya Seki, Shohei Watabe, Suguru Endo, Yuichiro Matsuzaki, Shiro Kawabata, Tetsuro Nikuni, and Hideaki Hakoshima. Boltzmann machine learning with a variational quantum algorithm. *Phys. Rev. A*, 104:032413, Sep 2021.