Project 2 - Studying Quantum Mechanical Systems with Restricted Boltzmann Machines

FYS4411 at University of Oslo

Simen Løken

June 2023

Contents

1 Abstract

In this project, we will examine the how we can estimate the wavefunction of a quantum mechanical system using a Restricted Boltzmann Machine. We find that it can sufficiently model a system of two fermions with two degrees of freedom, without interaction, with very good energy estimates that are very close in value to the nominal $2.0a.u.$ for such a system. We then extend the system to also include the interactive energy between the fermions, with somewhat lackluster results. Lastly, we quickly examine how a Restricted Boltzmann Machine could be replaced with an equivalent deep network, and see what kind of results it can yield.

2 Introduction

When working with quantum mechanics, traditionally we use wavefunctions to the describe and model the quantum states of the system. This is fine when working with small systems, but as the size and computational magnitude of the system increases, the resources required to represent and work with wavefunctions increases exponentially. This in turn makes it harder to study, explore and analyze quantum mechanical systems.
But all is not lost - in recent years the field of machine learning has seen significant advancement in terms of both deep learning and neural networks. One such method, and the method we're going to be studying today, is the Restricted Boltzmann Machine (RBM). It turns out RBMs provide a compact and efficient representation of quantum wavefunctions. The efficiency, expressive power and learning capabilities make them the an ideal choice when studying quantum mechanical systems
In this project, we will therefore examine a simple quantum mechanical system of limited complexity, and use an RBM to model the appropriate wavefunction to see if we can approach the known analytical solutions to the energies of such a system, for both an uninteractive and a interactive case, respectively. Lastly, we will try to see if we can replace the RBM with a deep network to see if it also can estimate the wavefunction of the system.

3 Theory and Method

3.1 The System

In this project we will be studying a two-particle system of fermions. The fermions will be limited to two-dimensional movement. We will study the system both in terms of an interacting and a non-interacting case. That is to say, when measuring the energy of the system we will be, we will look at both purely the kinetic and potential energy of the fermions, and also at the interacting energy between the two particles.

3.1.1 The non-interacting case

Firstly, we will study the non-interacting case, simply because it is much simpler than than its interacting counterpart and also because such a system has a known energy. The energy of such a system can be expressed in atomic units (a.u.) by:

$$E = N \cdot D \tag{1}$$

where N is the number of particles in the system and D is the degrees of freedom of the particles (in this case 1, 2 or 3 for 1D, 2D and 3D respectively). Additionally, the system is modeled by a Hamiltonian:

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

For the sake of simplicity, we let all natural units be 1, and set $\omega = 1$. This will allow us to work with atomic units, which let's us easily compare with the known analytical energy given by Eq. [1].

3.1.2 The interacting case

Now for the more interesting case. We alluded to earlier that we would only be examining two fermions with two degrees of freedom. There is a reason for this, as the analytical energy value of such a system is known to be 3 a.u.. This in turn makes it easier for us to tune our model and test if it can accurately numerically predict the energy of an interacting case of two fermions.
As for the Hamiltonian from above, given the interaction between the two particles, it gets an additional term:

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

such that the full Hamiltonian becomes:

$$\hat{H} = \hat{H}_0 - \hat{H}_1 = \sum_{i=1}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2 \right) + \sum_{i<j} \frac{1}{r_{ij}} \tag{4}$$

The energy of such a system is described as:

$$E_L = \frac{1}{\Psi}\hat{H}\Psi \tag{5}$$

where Ψ is the wavefunction represented by the RBM. A full derviation and of this expression of energy can be found at Appendix A.

3.2 Restricted Boltzmann Machines

We're going to be working with a Gaussian-binary RBM instead of the original binary-binary RBM in the project. The RBM is a generative neural network model that has many use cases. In this case, we're going to use it to simulate the wavefunction of a quantum mechanical system. An RBM consists of a visible layer and a hidden layer. The visible layer represents the visible quantum states, whilst the hidden layer represents underlying connections.
In our case, we let the visible layers represent the quantum states of the system and the hidden states the underlying quantum mechanical properties of the system. We can then model a joint probability distribution between the visible and hidden states to model the complexity of the wavefunction.
The joint probability distrubtion is given as:

$$F_{rbm}(v,h) = \frac{1}{Z})e^{-\frac{1}{T_0}E(v,h)} \tag{6}$$

where Z is defined as:

$$Z = \iint e^{-\frac{1}{T_0}E(v,h)}$$

Mathematically, we may define the energy of the RBM as:

$$E(v,h) = \sum_i \frac{(v_i - b_{vi})^2}{2\sigma_i^2} - \sum_j b_{hj}h_j - \sum_{i,j} \frac{v_i w_{ij} h_j}{\sigma_i^2} \tag{7}$$

3.2.1 Contrastive Divergence

When training RBMs, one of the most common methods is that of Contrastive Divergence (CD). The CD method approximates the gradient of the log-likelihood function by doing a series of Monte Carlo cycles, starting from the initial visible state and updating the visible and hidden states with each iteration. CD estimates the gradient by using a "phantom" particle from the model distribution and after a few sampling steps, subtracting the expectation given the current model distribution from the expectation of the "phantom" particle. We then use this approximation to update the RBM parameters.
The algorithm looks like this:

1. Sample the hidden states given the visible input using a sampling method.
2. Compute the positive gradient by taking the dot product between the transpose of the visible input and the sampled hidden states.
3. Perform sampling for a specified number of chains starting from the visible input.
4. Compute the negative gradient by taking the dot product between the transpose of the final chain state and the sampled hidden states.
5. Update the weights, visible biases, and hidden biases using the computed positive and negative gradients.
6. Repeat steps 1-5 until convergence or a predefined stopping criterion is met.

This method is very effective in training an RBM, but there is an extension of this method that is even better.
The Persistent Contrastive Divergence (PCD) is an extension of CD. In PCD, instead of initializing a Markov Chain for each iteration, we instead maintain a persistent chain throughout the entire training process. The persistent chain is updated after each iteration using the "phantom" particle.

1. Sample the hidden states given the visible input using a sampling method.
2. Compute the positive gradient by taking the dot product between the transpose of the visible input and the sampled hidden states.
3. Perform sampling for a specified number of chains starting from the visible input.
4. Compute the negative gradient by taking the dot product between the transpose of the final chain state and the sampled hidden states.
5. Update the weights, visible biases, and hidden biases using the computed positive and negative gradients.
6. Sample the hidden states given the persistent chain using sampling.
7. Generate a visible state by sampling from the hidden units of the persistent chain, and then sample the hidden units based on this visible state.
8. Update the weights, visible biases, and hidden biases using the sampled persistent chain.
9. Repeat steps 1-8 until convergence or a predefined stopping criterion is met.

The PCD, naturally, as an extension of CD, is better and faster than its unmodified counterpart.

3.2.2 Gradient Descent

Another very common method of training is the gradient descent method. Again we update the parameters of the RBM with each iteration, aiming to minimize a cost function. The cost function depends on the hidden biases b_h and the visible biases b_v, together with the weights w
We aim to find the optimal values for the biases and weights that minimize the cost function like thus:

1. Sample the hidden states given the visible input using a sampling method.
2. Compute the positive gradient by taking the dot product between the transpose of the visible input and the sampled hidden states.
3. Sample the visible and hidden states again for reconstruction by starting from the visible input.
4. Generate a reconstructed visible state by sampling from the hidden units, and then sample the hidden units based on this reconstructed visible state.
5. Compute the negative gradient by taking the dot product between the transpose of the reconstructed visible state and the sampled hidden states.
6. Update the weights, visible biases, and hidden biases using the computed positive and negative gradients.
7. Repeat steps 1-6 until convergence or a predefined stopping criterion is met.

3.3 Sampling methods

3.3.1 Metropolis sampling

When we say metropolis sampling, we refer to the Metropolis-Hastings algorithm, a Markov chain Monte Carlo method. The algorithm generates a sequence of samples from a target probability distribution. We construct a Markov chain [B] whose distribution is the target distribution. For each step in the chain, a sample is generated and then accepted or rejected based on the acceptance probability. The acceptance probability is chosen by the target distribution ratio at the generated and current state, respectively, along with a new proposed distribution.
In context of an RBM, we use metropolis sampling to sample from the RBMs learned distribution. We assign initial values values to the visible units, and for each iteration we we construct a Markov chain by updating the visible units given the current state. For a sufficient number of iterations, the samples will then approximate the true distribution of the wavefunction. This allows us to better model quantum mechanical systems, and allows the RBM to better model complex correlations and dependencies between the visible and hidden units

3.3.2 Importance sampling

Importance sampling, as opposed to directly sampling from the target distribution, instead sample from an easier to sample distribution. We attempt to choose a distribution that is similar to the target distribution in areas of high importance (hence the name). This in turn often makes Importance sampling a better alternative to the perhaps more brute-force method of Metropolis sampling. In the case of Importance sampling, the very nature of it ensures that the proposed distribution is tailored to closely match the important regions of the target distribution. This leads to low(er) variance in the estimated numerical results.

4 Results

Let us first examine the non-interacting case with Metropolis sampling for all three training methods, CD, PCD and GD:

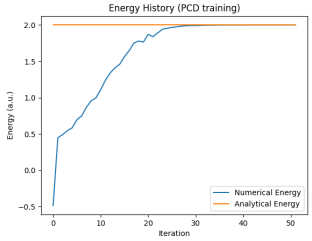

Figure 1: Energy convergence per iteration for Persistent Contrastive Divergence

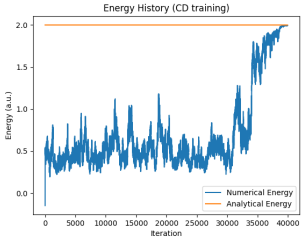

Figure 2: Energy convergence per iteration for Contrastive Divergence

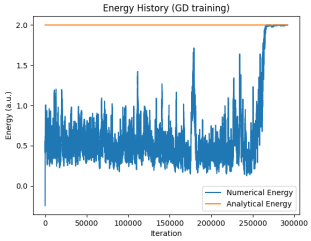

Figure 3: Energy convergence per iteration for Gradient Descent

Here we see how the energy converges over iterations for Metropolis sampling
$\eta = 0.005$, $h.u. = 16$

We see here that PCD performs by far the best, converging much earlier. We should note that this isn't a one-time occurrence. PCD generally needs less than 100 iterations to converge at an energy value. Of course, this depends on other factors and variables, but it is generally much, much faster. In terms of iterations, GD performs worse than CD, but it is worth mentioning that an iteration of GD is much faster than a CD iteration, so much so that GD is actually, despite the iterations, two times faster than that of CD.
On the next page we will examine a table of results and errors for a set of initial variables:

Type	Result [a.u.]	Error [a.u.]
PCD	2.00	2.16×10^{-7}
CD	2.00	5.57×10^{-5}
GD	2.00	4.69×10^{-5}

Table 1: The results given metropolis sampling and a blocking analysis with the following variables: MC Cycles: 4992, Hidden Units: 16, Visible Units: 4, η: 0.005, Sweeps: 100, Chain length: 100

We see here that all methods can accurately predict the numerical energy value of a 2 dimensional system with 2 fermions with varying degrees of error. PCD performs the best whilst being much faster than the other algorithms.

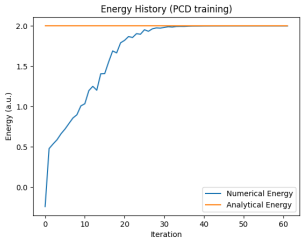

Figure 4: Energy convergence per iteration for Persistent Contrastive Divergence

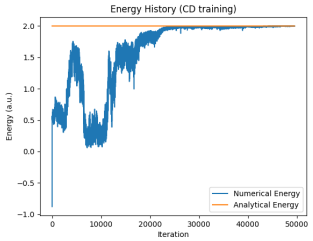

Figure 5: Energy convergence per iteration for Contrastive Divergence

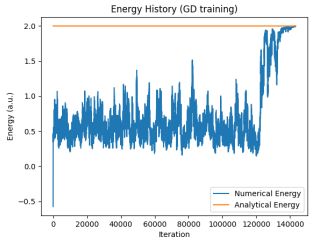

Figure 6: Energy convergence per iteration for Gradient Descent

Here we see how the energy converges over iterations for importance sampling
$\eta = 0.005$, $h.u. = 16$

Again we see much the same results. The only real difference is that CD more consistently trends towards the analytical value rather than plateauing at another value for a while.
We see a bigger difference in the accompanying table:

Type	Result [a.u.]	Error [a.u.]
PCD	2.00	2.51×10^{-9}
CD	2.00	1.50×10^{-7}
GD	2.00	4.38×10^{-7}

Table 2: The results given importance sampling and a blocking analysis with the following variables: MC Cycles: 4992, Hidden Units: 16, Visible Units: 4, η: 0.005, Sweeps: 100, Chain length: 100

Here we see that generally the errors are improved by a factor of 1/100th, whilst the predicted value remains the same. Again we also see the same order of performance in terms of errors when it comes to the types of training, where PCD vastly outperforms the other methods of training.

We now wish to examine the interactive case. Sadly, we weren't able to correctly implement this method. Namely, an issue where the particles converge at a single point, thus leading to values approaching infinity in the case of Eq. [3]. This leads to plots like these:

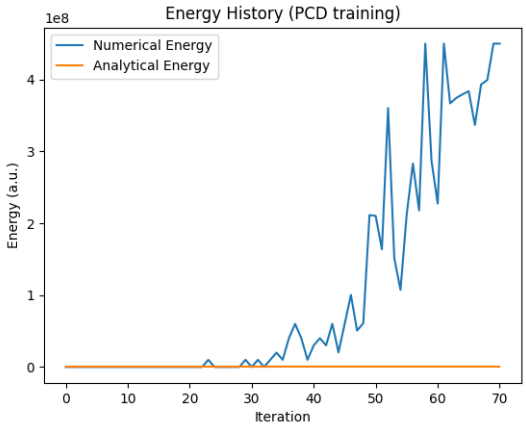

Figure 7: The energy grows exponentially as the particles approach the same location, in this case for PCD with Metropolis sampling, but this is true for every other training method and sampler too.

which sadly tells us next to nothing.

5 Discussion

5.1 Results and errors

Firstly, I want to discuss the results. As seen, the results in the case of both metropolis and importance sampling for all methods yield an energy of 2.0, which is the nominal value. This seems a little too good to be true, and I suspect there's some fishy business with the code. Of course, I ultimately have nothing to base this on, but the code does occasionally produce results in the range of 2.0 ± 0.10, which is a little more comforting but still not enough. Generally the code almost always produces an energy value of 2.0.

5.2 Code and variable exploration

We were asked to play around with variables and examine their effect of the code. Generally, atleast according to theory, as we increase most variables we'll see an increase in numerical accuracy (cycles, sweep, chain length, etc.). While many of these variables are straight forward, one that is interesting and unique to the RBM in particular are the hidden units. Typically, in the case of RBMs, the hidden units model the underlying complex relationships and features of the model (in our case the wavefunction). In my case, 16 hidden units was what I used the most, and I found it to be sufficent for numerical accuracy. Another quirk with the hidden units is that, as they increase, the convergence time logarithmically decreases.

Another quirk of the code I feel like is worth pointing out is the variable `N_SCALE`. While you'd think letting the state of the system be of shape $(1, 4)$ would be sufficent, this in my case did not work. Introducing then a variable `N_SCALE` such that the state is of shape $(\texttt{N\_SCALE}, 4)$ and then scaling the energy (which would be `N_SCALE`x higher than nominal value) yielded very good results. The reason for this I could not deduce, I tried letting `N_SCALE` be 1, 10, 50 and 100 respectively and it produced correct results for the latter three.

5.3 Replacing the RBM with a Neural Network

One (admittedly optional) part of the project was to see if we can replace the RBM with a Neural Network. While this was optional (and thus left for the end), I wanted to explore it as it is relevant for my Thesis. However, I had trouble getting it work, or rather, how I was supposed to train the Network effectively. Ultimately, I decided upon a very heavy-handed, probably not correct at all solution where it is simply trained based on whether the particles are interactive or not and the analytical energy values. This yielded some results, namely:

Interact?	Result [a.u.]	Error [a.u.]
No	1.99	1.70×10^{-8}
Yes	3.00	1.5×10^{-8}

Table 3: The energy estimated by the Neural Network based on whether or not the particles are interactive.

The issue I ran into mainly was that it was hard to train the network in the case of interactive fermions. In the case of non-interacting fermions, it would've been easy as the energy of such a system is known numerically through Eq. [1], but this is not the case for the interactive system. Thus it would've been very hard to train the network for anything but the non-interactive case, and ultimately (probably) useless in the case of interaction between the particles.

6 Conclusion

In closing, we've used a Restricted Boltzmann Machine to simulate the wave function of a Quantum Mechanical System of Fermions. We've examine how the RBM can be used to accurately predict the local energy of a two fermion system of two degrees of freedom without interaction for several different training and sampling methods. We've also tried to examine the interactive case to less than stellar results. Lastly, we tried our hand using a deep network instead to model the energy, and found that it too was sufficient, albeit with some caveats.

References

[1] Carreira-Perpinan E. Hinton. On contrastive divergence learning. URL: https://www.cs.toronto.edu/~hinton/absps/cdmiguel.pdf.

[2] Morten Hjorth-Jensen. *FYS4411 Lecture Notes*. 2023. URL: http://compphysics.github.io/ComputationalPhysics2/doc/web/course.

[3] Morten Hjorth-Jensen. Fys5429 lecture notes, 2023. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.

[4] Simen Løken. Studying a bosonic gas using a variational monte carlo method, 2023. URL: https://github.com/simloken/FYS4411/blob/main/Project_1/report/Project_1.pdf.

[5] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. URL: https://www.cs.toronto.edu/~tijmen/pcd/pcd.pdf.

Appendices

A Expression in 1a)

Given the definition of the Hamiltonian in Eq. [4] and the accompanying local energy in Eq. [5] we may express the energy:

$$E_L = \frac{1}{\Psi}\hat{H}\Psi$$

as:

$$E_L = \frac{1}{\Psi}\left[\sum_{i=1}^{N}\left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2\right) + \sum_{i<j}\frac{1}{r_{ij}}\right]\Psi$$

We can rewrite this as:

$$E_L = -\frac{1}{2\Psi}\sum_{i=1}^{N}\sum_{d=1}^{D}\frac{\delta^2\Psi}{\delta x_{id}^2} + \frac{1}{2}\omega^2\sum_{i=1}^{N} r_i^2 + \sum_{i<j}\frac{1}{r_{ij}}$$

Using then that:

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

we can rewrite this as:

$$E_L = \frac{1}{2}\sum_{i=1}^{N}\sum_{d=1}^{D}\left[-\left(\frac{\delta}{\delta x_{id}}\ln\Psi\right)^2 - \frac{\delta^2}{\delta x_{id}^2}\ln\Psi + \omega^2 x_{id}^2\right] + \sum_{i<j}\frac{1}{r_{ij}} \tag{8}$$

where N is the number of particles and D is the degrees of freedom, as above.

B Markov Chains

Markov chains are a mathematical framework used to describe a sequence of states or events, where the next state depends only on the current state. Markov chains are incredibly flexible and employed in many fields, able to both model and analyze dynamic processes with random and state dependency.
Mathematically, we may describe a Markov chain as a series of states $S_1, S_2, \ldots, S_n$ and a transition matrix $\mathcal{P}$ where a given element $\mathcal{P}_{ij}$ represents the probability of S transitioning from S_i to S_j
Additionally, the following must hold true

$$\sum_{i,j}^{N}\mathcal{P}_{ij} = 1$$

The idea is that the steady-state (or convergent state) distribution will remain unchanged when multiplied by the transition matrix $\mathcal{P}$, allowing us to converge.

C Code

Any code can be found at my github