FYS4411 - Computational Physics II

Project 2

Dorthea Gjestvang
Even Marius Nordhagen

April 27, 2018

- Github repository containing programs and results:
  `https://github.com/evenmn/FYS4411/tree/master/Project%202`

Abstract

Abstract

Contents

1 Introduction

2 Theory

2.1 Presentation of potential

In this project, we simulate a system of P electrons trapped in a harmonic oscillator potential, with a Hamiltonian given by

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

where ω is the harmonic oscillator potential and $r_i = \sqrt{x_i^2 + y_i^2}$ is the position of electron i. The term $\frac{1}{r_{ij}}$ is the interacting term, where $r_{ij} = |r_i - r_j|$ is the distance between a given pair of interacting electrons. Natural units have been used, such that $\hbar = c = m_e = e = 1$.

Since electrons are fermions, we need an antisymmetric wavefunction under exchange of two coordinates, and we need to take the Pauli principle into account. A Slater determinant is therefore needed for multiple fermions to ensure that the total wavefunction is antisymmetric. In this project we will study particles in the ground state only, and according to the Pauli principle we can in this case study a maximum of two particles with spin $s = \pm 1/2$. The slater determinant for two particles read

$$\Psi_T = \begin{vmatrix} \Phi_1(\boldsymbol{r}_1) & \Phi_2(\boldsymbol{r}_1) \\ \Phi_1(\boldsymbol{r}_2) & \Phi_2(\boldsymbol{r}_2) \end{vmatrix} = \Phi_1(\boldsymbol{r}_1)\Phi_2(\boldsymbol{r}_2) - \Phi_2(\boldsymbol{r}_1)\Phi_1(\boldsymbol{r}_2) \tag{2}$$

where $\Phi_i(\boldsymbol{r})$ is the single particle wave function (SPF) of state i. This contains a spatial part and a spin part, and we assume that it can be splitted up such that the spin part takes the antisymmetry property and does not affect the energy. Therefore we only need a symmetric spatial part to calculate the energies.

2.2 Solving this with machine learning

When solving a system of particles as the one described in the previous system with Variational Monte Carlo, as described in [?], we would need an anzats for the wave function, where we use our physical intuition to create the form of a wave function with different variational parameters, and then let it be up to the computer to find the optimal parameters through a minimization method. However, this method is only as good as the physical intuition; if the form of the wave function is unrealistic, the results will be the same

This challenge can be mended by using machine learning. There are several different types of machine learning systems, and the one we will present and utilize in this project has the ability to learn and sample from a probability distribution. This is perfect for quantum mechanical problems, as we know from quantum mechanics the wave function Ψ is nothing more than a probability denisty, giving that Ψ^2 is a probability distribution that says something about where a given particle most probabliy can be found. As we are solely interested in the energy of the two-fermion system, and not the exact wave function, the fact that the machine learning program does not explicitly give the wave function is therefore of no consequence. We still have to give some guidelines for the form of the probability distribution, but the machine learning program can sample from a larger variety of probability distributions compared to the form used in VMC.

2.2.1 Machine learning

With the goal of solving the quantum mechanical system presented in section **??** in mind, we should start by explaining what machine learning is. Machine learning is the idea that a computer can be trained to learn to yield certaint outputs, without directly being told exactly what to give. Examples on this is pattern recognizion, where the computer first is shown for example pictures of wolves and huskies. After training the computer on pictures where the computer sees huskies and wolves and is told the correct answer, it should after a sufficiently long training period, be able to recognize huskies and wolves by itself.

The example described above is what we call supervised learning, where the correct output answer is known during the training program. A machine learning program could also be unsupervised, where the correct answer is unknown, or based on reinforcement learning, where the the program learns by conducting trial-and-error experiments.

2.2.2 Neural network

This sound amazing, and maybe even impossible. Therefore the question now is: how to program computers to learn, just like humans? The answer is, fittingly, that we should make the program run like the the human brain by implementing what is called a neural network. Inspired by neurons in the human brain, a neural network is a programmed network of variables, called nodes, that comminucate in a given manner. Each node preforms a simple process: based on the input it receives, and how that input is weighted, it decides wheter or not to fire. The mathematical model of an example of a neural node was presented by McCulloch and Pitts in 1943 [?], shown in figure ??, where the input is mared x_i, the weights deciding how much the input should count is W_i, and the output from the node based on x_i and W_i is called h.

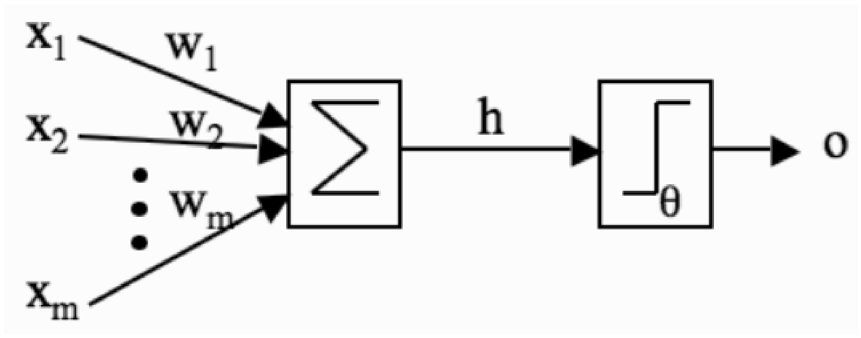

Figure 1: McCullochs and Pitts' model of neutrons visualized. Image reproduced from [?]

The neutrons are arranged in layers in the neural network, one visible layer that receives the input, and up to several hidden layers. The layers are arranged such that the output values from the visible nodes is the input values of the visible nodes. The nodes can also have bias values, that shift the output value h with a ceirtan number, and the nodes in different layers can be connected in different ways. An example of a neural network with two layers is shown in figure ??.

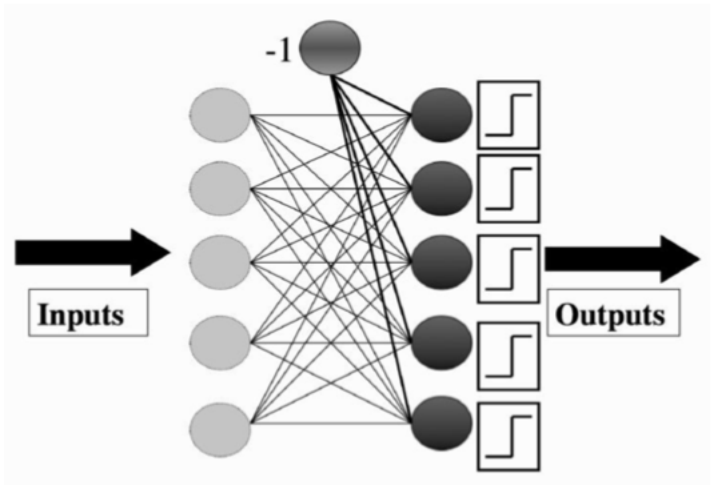

Figure 2: An example of a neural network with one visible and one hidden layer of nodes, and with bias -1 on the hidden nodes. Image reproduced from [**?**]

The idea behind machine learning is that the weights W_i, that decides how much a node puts emphazis on a given input, can be changed, and thus change the system's response to the same input. We will explain with the example from earlier, with huskies and the wolves: first, the program are shown pictures of wolves and huskies, and it is told the correct answer. The weights W_i are then updated such that when a husky is shown, the output "husky" is generated, and the same for wolves. After a sufficiently long training period, the program's weights are optimalized for recognizing wolves and huskies. When shown a picture, the program should then by itself be able to determine wheter it is a wolf or a husky that it sees.

2.2.3 Restricted Boltzmann Machines

There are plenty of ways to put together a neural network, as one can modify the number of nodes and layers, the bias, and also which nodes that are allowed to communicated. In this project, we will be using the so-called Restricted Boltzmann Machine (RBM). It is a two-layer network. The reason why it is named "restrictive" is that there are no connections between nodes in the same layer, but every node in the previous layer is connected to all the nodes in the next layer. The RBM can learn to draw samples from a probability distribution, which is just what we want in our project. In addition, we want to use a Gaussian-Binary RBM, where the hidden nodes have binary values, while the positions of the particles can take continous values, as they are, in fact, positions.

The joint probability distribution is known from statistical mechanics,

$$F(\boldsymbol{X},\boldsymbol{H}) = \frac{1}{Z}e^{-\beta E(\boldsymbol{X},\boldsymbol{H})} \tag{3}$$

where we set $\beta = 1/kT = 1$ and Z is the partition function, which can be ignored since it will vanish anyway.

The system energy of a Gaussian-Binary RBM is given by

$$E(\boldsymbol{X},\boldsymbol{H}) = \sum_{i=1}^{M}\frac{(X_i-a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{N} b_j H_j - \sum_{i,j=1}^{M,N}\frac{X_i W_{ij} H_j}{\sigma_i^2} \tag{4}$$

(Hinton 2010).

By setting $\Psi = F(\boldsymbol{X},\boldsymbol{H})$, we know have a general form of the probability distribution. In equation **??** the form of the wave function is shown, when we have used that we have M visible nodes and N hidden nodes, and that the hidden nodes should take binary values.

$$\Psi = \frac{1}{Z}e^{\sum_i^M \frac{(X_i-a_i)^2}{2\sigma^2}} \prod_j^N (1 + e^{b_j + \sum_i M \frac{X_i W_{ij}}{\sigma^2}}) \tag{5}$$

2.3 Energy calculation

2.4 Onebody density

2.5 Scaling

2.6 Error estimation

3 Method

3.1 Variational Monte Carlo

3.2 Metropolis Algorithm

3.2.1 Brute force

3.2.2 Importance sampling

3.2.3 Gibbs sampling

3.3 Gradient descent

We use stochastic gradient descent to update the weights in order to minimize the local energy. The process itself is quite similar to that one typically used in Variational Monte Carlo (VMC) where the weights are our variational parameters, but compared with a normal VMC we have a lot more parameters to vary. The updating algorithm for updating the variational parameter α_i goes as

$$\alpha_i^+ = \alpha_i - \eta d\alpha_i \tag{6}$$

where the gradient of the local energy with respect to parameter α_i is

$$d\alpha_i = \frac{\partial \langle E_L \rangle}{\partial \alpha_i} = 2\bigg(\Big\langle E_L \frac{1}{\Psi}\frac{\partial \Psi}{\partial \alpha_i}\Big\rangle - \Big\langle E_L \Big\rangle \Big\langle \frac{1}{\Psi}\frac{\partial \Psi}{\partial \alpha_i}\Big\rangle\bigg) \tag{7}$$

where η is the learning rate. We need to do this for all the parameters, and for our purposes it is convenient to vectorize the gradient such that $d\boldsymbol{a}$ and $d\boldsymbol{b}$ are vectors and $d\boldsymbol{W}$ is a matrix. The gradients to implement are

$$d\boldsymbol{a} = \frac{1}{\Psi}\frac{\partial \Psi}{\partial \boldsymbol{a}} = \frac{\boldsymbol{X} - \boldsymbol{a}}{\sigma^2} \tag{8}$$

$$d\boldsymbol{b} = \frac{1}{\Psi}\frac{\partial \Psi}{\partial \boldsymbol{b}} = \tag{9}$$

3.3.1 Adaptive stochastic gradient descent

A problem with the standard stochastic gradient descent method is that it will either converge fast, but the precision is poor, or the precision is good, but it converges slowly. To avoid this problem, we want a large learning rate when the energy is far from converging and a small learning rate when the energy converges.

4 Code

4.1 Structure

4.2 Implementation

5 Results

6 Discussion

7 Conclusion

8 Appendix A - Local energy calculations

The energy can be calculated by introducing a local energy, where the acerage local energy goes to the true energy with a sufficient amount of samples. This local energy can be splitted up in a kinetic part, a part from the harmonic oscillator potential and a interacting part,

$$E_L = \sum_{k=1}^{M} (E_{\text{KIN},k} + E_{\text{EXT},k}) + E_{\text{POT}}. \tag{10}$$

As discussed in [**?**], the kinetic part can be expressed as

$$\begin{aligned} E_{\text{KIN},k} &= \frac{1}{\Psi_T} \nabla_k^2 \Psi_T \qquad (11) \\ &= (\nabla_k \ln \Psi_T)^2 + \nabla_k^2 \ln \Psi_T \qquad (12) \end{aligned}$$

where we have used that

$$\frac{1}{\Psi_T} \nabla_k \Psi_T = \nabla_k \ln \Psi_T. \tag{13}$$

$$\frac{\partial}{\partial X_k} \ln \Psi_T = -\frac{X_k - a_k}{\sigma^2} + \sum_{j=1}^{N} \frac{W_{kj}}{\sigma^2} \text{Logistic}(v(j)) \tag{14}$$

$$\frac{\partial^2}{\partial X_k^2} \ln \Psi_T = -\frac{1}{\sigma^2} + \sum_{j=1}^{N} \frac{W_{kj}^2}{\sigma^4} \text{Logistic}^2(v(j)) e^{v(j)} \tag{15}$$

where

$$\text{Logistic}(x) = \frac{1}{1 + e^x} \tag{16}$$

and

$$v(j) = b_j + \sum_{i=1}^{M} \frac{X_i W_{ij}}{\sigma^2}. \tag{17}$$

Thus the local energy can be expressed as

$$E_L = \sum_{i=1}^{N} \frac{\boldsymbol{W}_{*i}^T \boldsymbol{W}_{*i}}{\sigma^4} \text{Logistic}^2\big(v(i)\big) + \tag{18}$$

$$\sum_{i,j=1}^{N,N} \frac{\boldsymbol{W}_{*i}^T \boldsymbol{W}_{*j}}{\sigma^4} \text{Logistic}\big(v(i)\big) \text{Logistic}\big(v(j)\big) - \tag{19}$$

$$2 \sum_{i=1}^{N} \frac{\boldsymbol{W}_{*i}^T (\boldsymbol{X} - \boldsymbol{a})}{\sigma^4} \text{Logistic}(v(i)) + \tag{20}$$

$$\frac{(\boldsymbol{X} - \boldsymbol{a})^T \cdot (\boldsymbol{X} - \boldsymbol{a})}{\sigma^4} - \frac{1}{\sigma^2} + \boldsymbol{X}^T \boldsymbol{X} + E_{\text{INT}} \tag{21}$$

9 References

INCLUDE ONLY THOSE REFERENSES WE USE

[1] Morten Hjorth-Jensen. Computational Physics 2: Variational Monte Carlo methods, Lecture Notes Spring 2018. Department of Physics, University of Oslo, (2018).

[2] S. Marsland, *Machine Learning: An algorithmic Perspective, Second edition* (2015)

[3] D. Gjestvang, E. M. Nordhagen *Computational Physics II: Project 1* (2018)