University of Oslo
FYS4411
Computational physics II: Quantum mechanical systems

Project 2: The Restricted Boltzmannn Machine Applied to the Quantum Many Body Problem

Bendik Samseth

May 16, 2018

Abstract

Abstract here, please. All material referenced in this report is available at `https://github.com/bsamseth/FYS4411`.

Contents

1 Introduction

A physicist faced with a quantum many body system is quickly forced to abandon any hope of solving the Schrödinger equation (or one of its even worse relativistic siblings) exactly. Instead he must resort to one of two general workarounds: 1) make simplifying assumptions until the equation is solvable, or 2) make an educated guess for the solution. In either case, the desired quantity which solving the Schrödinger equation would yield is the quantum mechanical wavefunction. If we want to stay away from potentially faulty, or limiting assumptions, we must therefore somehow produce a guess for this wavefunction, without actually solving the equation.

The last project utilized the Variational Monte Carlo (VMC) approach, in which a specific, parametrized form is assumed for the wavefunction, with the subsequent optimization of the parameters.

The main limitation of VMC is that a specific form for the wavefunction is assumed. If this form contains the true optimum that works fine, but if not we are inherently limited in how accurate the final results can be.

Aiming to be overcome this issue, we may observe that the task at hand falls into the general field of function approximation. We wish to approximate the function that takes as input the configuration of the system, and outputs a probability amplitude for the given state. Function approximation is a task which lends itself to techniques from machine learning (ML).

In this project we will attempt to represent the wavefunction with a specific type of ML-model, the Restricted Boltzmann Machine (RBM). The use of RBMs for such applications was presented recently by Carleo & Troyer [1], with encouraging results. We shall therefore attempt to apply the same techniques to a different type of system, namely that of two interacting quantum particles confined in a harmonic oscillator trap.

The choice of using the RBM as our model is somewhat arbitrary, and other model architectures could also be of interest. The general ML-approach which we employ, regardless of the specifics of the underlying model, is as follows:

1. Define the model to be trained.
2. Formulate the objective, or *loss function.*
3. Using some optimization scheme, optimize the model wrt. to the loss function.

2 Theory

2.1 The System

We consider a system of electrons confined in a pure two-dimensional isotropic harmonic oscillator potential, with an idealized total Hamiltonian given by:

$$\begin{aligned}\hat{H} &= \sum_{i=1}^{P}\left(-\frac{1}{2}\nabla_i^2 + V_{ext}(\boldsymbol{r}_i)\right) + \sum_{i<j} V_{int}(\boldsymbol{r}_i, \boldsymbol{r}_j)\\ &= \sum_{i=1}^{P}\left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2\right) + \sum_{i<j}\frac{1}{r_{ij}},\end{aligned} \tag{1}$$

where natural units ($\hbar = c = m_e = 1$) are used with energies in atomic units (a.u.), P denotes the number of particles in the system, and ω is the oscillator frequency of the trap. Further, $\boldsymbol{r}_i$ denotes the position vector of particle i, with $r_i \equiv \|\boldsymbol{r}\|$ and $r_{ij} \equiv \|\boldsymbol{r}_i - \boldsymbol{r}_j\|$ defined for notational brevity.

In this project we limit ourselves to the case of $N = 2$ interacting electrons in a trap with a frequency such that $\hbar\omega = 1$. We do this because for this case we have exact, analytical solutions for the ground state energy. With the interaction term included, the ground state energy is $E_0 = 3\,\text{a.u.}$ [2]. This limitation is purely one of convenience, as having exact benchmarks makes for better verification of results. The methods discussed in the project should extend to other systems.

2.2 Simple Non-Interacting Case

If we omit the interacting terms in Equation 1 we have the standard harmonic oscillator Hamiltonian:

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

This Hamiltonian lends it self to analytical solutions, and the stationary states are:

$$\phi_{n_x,n_y}(x,y) = A H_{n_x}(\sqrt{\omega}x) H_{n_y}(\sqrt{\omega}y) e^{-\frac{\omega}{2}\left(x^2+y^2\right)}, \tag{3}$$

for quantum numbers $n_x, n_y = 0, 1, \dots,$ and the Hermite polynomials H_n. The ground state, $n_x = n_y = 0$ is simply

$$\phi_{00}(x,y) = \sqrt{\frac{\omega}{\pi}} e^{-\frac{\omega}{2}\left(x^2+y^2\right)}. \tag{4}$$

Using this wavefunction we can calculate the ground state energy[1],

$$\epsilon_{00} = \frac{\langle\phi_{00}|\hat{H}_0|\phi_{00}\rangle}{\langle\phi_{00}|\phi_{00}\rangle} = \omega = 1\,\text{a.u.} \tag{5}$$

The ground state wavefunction for the (unperturbed) two-electron case is simply the product of the one-electron wavefunctions,

$$\begin{aligned}\Phi(\boldsymbol{r}_1, \boldsymbol{r}_2) &= \phi_{00}(\boldsymbol{r}_1)\phi_{00}(\boldsymbol{r}_2)\\ &= \frac{\omega}{\pi} e^{-\frac{\omega}{2}\left(r_1^2+r_2^2\right)}.\end{aligned} \tag{6}$$

The ground state energy can once again be evaluated analytically[1] and yields

$$E_0 = \frac{\langle\Phi|\hat{H}_0|\Phi\rangle}{\langle\Phi|\Phi\rangle} = 2\omega = 2\,\text{a.u.} \tag{7}$$

This result is not surprising, as adding one more particle, without any interactions, should simply double the energy. Another way to look at it is that the simple harmonic oscillator solution gives $\omega/2$ per degree of freedom, so adding another two yields and extra ω.

When the two particles are electrons, we may say something about their total spin. As electrons are fermions, their total wavefunction must be anti-symmetric upon interchanging the labels 1 and 2. Equation 6 is obviously symmetric, and so the spin-wavefunction must necessarily be anti-symmetric. For the combination of two spin-1/2 particles, there is only one candidate, namely the spin-0 singlet:

$$\chi_0 = \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle). \tag{8}$$

[1]See `projects/python/Sympy.ipynb` in the Github repository for an explicit calculation of the ground state energy

3 Representing the Wavefunction with an RBM

Our machine learning model of choice is the Restricted Boltzmann Machine. It consists of of two densely interconnected layers, the *visible* layer and the *hidden* layer. It is called restricted because we only allow connections between nodes in different layers - no visible-to-visible or hidden-to-hidden connections are included.

The RBM is a generative model, meaning it learns a probability distribution for its inputs. This means that a trained RBM can produce outputs which when viewed as a distribution, would be similar to the distribution of the inputs. For our case, this means learning the probability distribution for the space configurations of the electrons in our system. We may interpret this distribution as the wavefunction, as the wavefunction serves this same purpose.

In this case we do not have a training set of positions for any of the systems under consideration. This means that the most desired training regime, *supervised training*, is not relevant for our problem. Instead we shall use *reinforcement training*, were updates of the model are based on the variational principle: The true ground state wavefunction is the wavefunction for which the lowest ground state energy is obtained. We can treat the energy that a certain wavefunction (model configuration) produces as the penalty, and let the model adapt as to minimize the penalty it receives.

This approach should work in general, but there is one potential issue with this. In order to evaluate how good (or bad) a proposed model work, we need to evaluate the ground state energy. In order to do this we, as in project 1, use the proposed wavefunction to sample positions to be used in the evaluation of the energy. This will only work well if the model is somewhat decent at modeling the true wavefunction. If the proposed model is very far off, the resulting energy evaluations will be very unstable, and we could end up with large errors in the computed gradients. This could in turn lead to an even worse model, and this could go on until something crashes. This issue is similar to the *exploding gradient* problem which often occurs in neural network models, if not addressed properly. We will not address this directly in this project, and instead lean on the fact that we probably have rather decent starting guesses, as we have exact solutions to the ideal case. It is, however, a potential issue if this approach should be applied to systems for which we have a very poor sense of the underlying wavefunction form.

3.1 The Math

In the following, $\boldsymbol{x}$ denotes the values of the visible layer (our position coordinates), and $\boldsymbol{h}$ denotes the values of the hidden layer.

The joint probability distribution over $\boldsymbol{x}$ and $\boldsymbol{h}$ is:

$$F_{RBM}(\boldsymbol{x},\boldsymbol{h}) = \frac{1}{Z} e^{-\frac{1}{T_0}E(\boldsymbol{x},\boldsymbol{h})}, \tag{9}$$

where Z is the partition function ensuring that F_{RBM} is normalized. In accordance with common norm, we set $T_0 = 1$. The function $E(\boldsymbol{x},\boldsymbol{h})$ is known as the energy of a configuration of the nodes, not to be confused with the energy of the quantum system. It encodes the probability of a given configuration - high energy configurations are less likely.

3.1.1 Gaussian-Binary RBM

The type of energy function we will use is called Gaussian-Binary, meaning our inputs (the position coordinates) are Gaussian (continuous), while the hidden nodes take binary values $h_j \in \{0,1\}$. It looks as follows:

$$E(\boldsymbol{x},\boldsymbol{h}) = \frac{\|\boldsymbol{x}-\boldsymbol{a}\|}{2\sigma^2} - \boldsymbol{b}^T\boldsymbol{h} - \frac{\boldsymbol{x}^T\boldsymbol{w}\boldsymbol{h}}{\sigma^2}, \tag{10}$$

where $\boldsymbol{a} \in \mathbf{R}^M, \boldsymbol{b} \in \mathbf{R}^N$ are bias vectors for the visible and hidden layers respectively, and $\boldsymbol{w} \in \mathbf{R}^{M\times N}$ is a matrix encoding the weights for every connection between the two layers. In our case, $M = PD$ for P particles in D dimensions, while N will be chosen freely by us.

3.1.2 The Wavefunction

The wavefunction shall be the probability amplitude of any given system configuration. We obtain this by tracing out the hidden values:

$$\begin{aligned}\Psi(\boldsymbol{X}) = F_{RBM}(\boldsymbol{x}) &= \sum_{\boldsymbol{h}} F_{RBM}(\boldsymbol{x},\boldsymbol{h}) \\ &= \frac{1}{Z} e^{-\sum_i^M \frac{(X_i-a_i)^2}{2\sigma^2}} \prod_j^N \left(1+e^{b_j+\sum_i^M \frac{X_i w_{ij}}{\sigma^2}}\right) \\ &\equiv \frac{1}{Z} e^{-\sum_i^M u_i} \prod_j^N (1+e^{v_j}) \end{aligned} \tag{11}$$

where u_i and v_j are defined for convenience. We will treat a_i, b_i and w_{ij} as tunable parameters, while σ^2 will be taken as some constant, specifically the value we would have for the ideal wavefunction for the non-interacting case.

4 Learning the Wavefunction

4.1 The Cost Function - Energy

The cost function we use to train the network shall be the expectation value of the Hamiltonian, under the wavefunction modeled by the network. Minimizing the energy will yield the best possible wavefunction obtainable within the model. The energy is expressed as

$$Q = E[H] = \langle H \rangle = \frac{\int \mathrm{d}\boldsymbol{x}\, \Psi^* \hat{H} \Psi}{\int \Psi^* \Psi}. \tag{12}$$

where $\boldsymbol{x}$ is the vector containing all the positions of the particles in the system, $\boldsymbol{x} = [x_1, y_1, \ldots, x_n, y_n]$. In order to numerically evaluate this integral we first manipulate it a bit. The probability density at position $\boldsymbol{x}$, under the trial wave function, is

$$P(\boldsymbol{x}) = \frac{|\Psi|^2}{\int \mathrm{d}\boldsymbol{x}\, |\Psi|^2}. \tag{13}$$

We finally define a new quantity, called **the local energy**:

$$E_L(\boldsymbol{x}) = \frac{1}{\Psi} \hat{H} \Psi \tag{14}$$

Combining these two definitions we can now rewrite $\langle H \rangle$ as follows:

$$\begin{aligned}\langle H \rangle &= \int \mathrm{d}\boldsymbol{x}\, P(\boldsymbol{x}) E_L(\boldsymbol{x}) \\ &\approx \frac{1}{n} \sum_{i=1}^n E_L(\boldsymbol{x}_i),\end{aligned} \tag{15}$$

where $\boldsymbol{x}_i$ are n randomly drawn positions from the PDF $P(\boldsymbol{x})$. We have therefore that estimating the average value of E_L yields an approximated value for $\langle H \rangle$.

4.2 Optimization

In order to train the model to minimize $\langle \hat{H} \rangle$ we need to know how to adjust the parameters $\boldsymbol{\alpha} = (a_1, \ldots, a_M, b_1, \ldots, b_N, w_{11}, \ldots, w_{MN})$. We do this using some optimization algorithm, which will in turn be based on the partial derivatives of the expectation of the local energy [3]:

$$G_i = \frac{\partial \langle E_L \rangle}{\partial \alpha_i} = 2\left(\left\langle E_L \frac{1}{\Psi} \frac{\partial \Psi}{\partial \alpha_i} \right\rangle - \langle E_L \rangle \left\langle \frac{1}{\Psi} \frac{\partial \Psi}{\partial \alpha_i} \right\rangle \right) \tag{16}$$

These partial derivatives are trivial to compute analytically, and come out as follows:

$$\frac{1}{\Psi} \frac{\partial \Psi}{\partial a_k} = \frac{x_k - a_k}{\sigma^2} \tag{17}$$

$$\frac{1}{\Psi} \frac{\partial \Psi}{\partial b_k} = \frac{1}{1+e^{-v_k}} \tag{18}$$

$$\frac{1}{\Psi} \frac{\partial \Psi}{\partial w_{kl}} = \frac{1}{1+e^{-v_l}} \frac{x_k}{\sigma^2} \tag{19}$$

The expression for the local energy it self is a bit less trivial to work out, but still doable. And as we shall need to compute the local energy often, it will be useful to do the differentiation analytically, so as to speed up the computation compared to doing the second derivative in $\hat{H}$ numerically. The details are laid out in Appendix A.

The final expression we shall use for the local energy is:

$$E_L = \sum_{i=1}^M \frac{1}{2} x_i^2 + \sum_{i<j}^P \frac{1}{r_{ij}} - \frac{1}{2} \sum_{k=1}^M \frac{1}{\Psi} \frac{\partial^2}{\partial x_k^2} \Psi, \tag{20}$$

where,

$$\begin{aligned}\frac{1}{\Psi} \frac{\partial^2}{\partial X_k^2} \Psi = &-\frac{1}{\sigma^2} + \sum_j^N \frac{w_{kj}^2}{\sigma^4} \frac{e^{-v_j}}{(1+e^{-v_j})^2} \\ &+ \frac{1}{\sigma^4} \left(a_k - x_k + \sum_j^N \frac{w_{kj}}{1+e^{-v_j}} \right)^2 \end{aligned} \tag{21}$$

The complexity of E_L is $\mathcal{O}(M + P^2 + MNM) = \mathcal{O}(P^2 + M^2 N)$. This can be optimized slightly by computing all the v_j terms at once (as opposed to on demand within the sums), which brings this down to $\mathcal{O}(P^2 + MN)$. It still scales quadratically with additional particles (and dimensions), and linearly with the number of hidden nodes.

Comparing this with the results from project 1, where the complexity of a local energy evaluation was $\mathcal{O}(P^3)$ when a similar interaction was considered, we see a significant difference. *Assuming* we can obtain good results using an RBM where N is comparable to P in size, this new approach has much better time-complexity and therefore looks much more promising for use with large systems. We will not pursue this large P regime much further in this project, but this is something to note, if the RBM prove to be fruitful.

4.2.1 Optimization Schemes

Now equipped with a gradient, the general approach of optimization goes as follows:

Algorithm 1 General Optimization Routine

```
Require: Cost function $Q$
Require: Update function $O$
Ensure: $\boldsymbol{\alpha}$ minimizes $O$
  Initialize $\boldsymbol{\alpha}$ with random values.
  while $\boldsymbol{\alpha}$ not converged do
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + O(\nabla Q)$
  end while
```

What remains to plugged into Algorithm 1 is some update function which should return a suitable update to apply to our parameters. There are a myriad of versions for how to do this, with the most simple being *Stochastic Gradient Decent* (SGD). The basic algorithm for SGD is shown in Algorithm 2.

Algorithm 2 The Stochastic Gradient Decent Algorithm

```
Require: Cost function gradient $\nabla Q$
Require: Learning rate $\eta > 0$
  return $-\eta \nabla Q$
```

Many, many additions can be made to SGD in order to improve convergence speed and stability. Examples of such modifications are using a decaying learning rate η, including previous updates in the calculation of new ones (momentum), and many other variations. We will limit this project to implementing two update functions, the aforementioned SGD, and ADAM [4]. The ADAM optimizer is widely used in ML and consistently performs on par or better than any other scheme currently known. The pseudo-code for the algorithm is given in the original paper [4], and the implementation can be found on the Github repository under the filename `src/optimizer.cpp`

4.3 Regularization

We may wish to impose some regularization in the cost function as well. Often times this can help guide the optimization out of local minima, as well as shaping ill-formed cost functions. We can modify the cost function as follows:

$$Q = \langle H \rangle + \gamma \|\boldsymbol{\alpha}\|_d^d, \tag{22}$$

where $\gamma > 0$ is a hyper-parameter controlling the amount of regularization, and $\|\cdot\|_d^d$ is the L_d norm. Two of the most widely used types of regularization are Ridge and Lasso, which use $d = 2$ and $d = 1$, respectively. In order to keep the gradient simple, we will only implement Ridge loss. We obtain then a slightly modified form for G_i:

$$G_i = \frac{\partial \langle E_L \rangle}{\partial \alpha_i} + 2\gamma\alpha_i, \tag{23}$$

5 Sampling Algorithms

We will continue to use the Metropolis and Metropolis-Hastings algorithms presented in project 1 for this project also. For the sake of conciseness, we shall not repeat the derivations or motivations for these algorithms here, as they are easily adapted to work with this new model. However, the RBM model enables us to employ another sampling technique, *Gibbs sampling.*

5.1 Gibbs Sampling

For the particular case of systems like the ones we are currently considering, we know that the true wavefunction is positive definite. This allows us to use (yet) another sampling algorithm: Gibbs sampling. In order for us to make efficient use of Gibbs sampling, we have to change our wavefunction representation a bit,

$$\Psi_G(\boldsymbol{x}) = \sqrt{\Psi(\boldsymbol{x})} = \sqrt{F_{RBM}(\boldsymbol{x})}, \tag{24}$$

such that F_{RBM} models the probability $|\Psi_G|^2$ directly, instead of the probability amplitude Ψ_G.

This is done in order to have efficient, direct expressions to sample from later (more in next section). This transformation is only valid in general when the wavefunction is positive definite, while the original definition Ψ remains valid in general.

5.2 The Algorithm

Gibbs sampling comes up whenever we are dealing with a *joint* PDF from which we cannot easily (or at all) sample values. In our case we are dealing with the PDF from Equation 9, which does not have a standard, direct sampling strategy. In general, given a PDF $P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$, Gibbs relies on the conditionals $P(X_i = x_i | X_j = x_j \forall j \neq i)$. *If* these conditionals are easy to sample from (i.e. can be expressed as one of the known distributions for which direct sampling is possible), then we may obtain approximate samples from the joint PDF by iteratively sampling individual values from the conditionals. Going back to Equation 9, the algorithm becomes:

Algorithm 3 Gibbs sampling of $\boldsymbol{x}$ from $F_{RBM}(\boldsymbol{x}, \boldsymbol{h})$

```
Require: $n$ number of samples to yield.
Ensure: $\boldsymbol{x}, \boldsymbol{h} \overset{\mathrm{d}}{=} P(\boldsymbol{x}, \boldsymbol{h})$.
  Initialize $\boldsymbol{x}$ randomly (e.g. normally).
  for $n$ iterations do
    $\boldsymbol{h} \overset{\mathrm{d}}{\leftarrow} P(\boldsymbol{h}|\boldsymbol{x})$
    $\boldsymbol{x} \overset{\mathrm{d}}{\leftarrow} P(\boldsymbol{x}|\boldsymbol{h})$
    Save $\boldsymbol{x}$ as the i-th sample.
  end for
```

From Algorithm 3 is is clear that we need to be able to sample from both $P(\boldsymbol{x}|\boldsymbol{h})$ and $P(\boldsymbol{h}|\boldsymbol{x})$. These are simply stated here without further derivation, and follow from the form of the RBM:

$$P(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}\Big(\boldsymbol{x}; \boldsymbol{a} + \boldsymbol{w}\boldsymbol{h}, \sigma^2\Big), \tag{25}$$

$$P(\boldsymbol{h}|\boldsymbol{x}) = \prod_j^N \frac{e^{v_j h_j}}{1 + e^{v_j}} \tag{26}$$

The first conditional is just the normal distribution, which there are standard tools to sample from. The second seems a little harder, until we look at it for one single h_i:

$$P(h_j|\boldsymbol{x}) = \frac{e^{v_j h_j}}{1 + e^{v_j}} \Rightarrow P(h_j = 1|\boldsymbol{x}) = \frac{1}{1 + e^{-v_j}}. \tag{27}$$

So to sample h_i we set it to 1 with probability $p = (1 + \exp(-v_j))^{-1}$, and 0 otherwise.

5.3 New Expressions for Ψ_G

Using Ψ_G as our wavefunction will slightly change all the expressions for the derivatives. Luckily, not much work is needed to redo them all using the following observations:

$$\ln \Psi_G = \ln \sqrt{\Psi} = \frac{1}{2} \ln \Psi, \tag{28}$$

and

$$\frac{1}{\Psi}\frac{\partial \Psi}{\partial x_k} = \frac{\partial \ln \Psi}{\partial \Psi}\frac{\partial \Psi}{\partial x_k} = \frac{\partial \ln \Psi}{\partial x_k} \tag{29}$$

$$\Rightarrow \frac{1}{\Psi_G}\frac{\partial \Psi_G}{\partial x_k} = \frac{1}{2}\frac{1}{\Psi}\frac{\partial \Psi}{\partial x_k}. \tag{30}$$

Propagating this factor of a half through all the expressions quite easily yields:

$$\frac{1}{\Psi_G}\frac{\partial \Psi_G}{\partial a_k} = \frac{x_k - a_k}{2\sigma^2} \tag{31}$$

$$\frac{1}{\Psi_G}\frac{\partial \Psi_G}{\partial b_k} = \frac{1}{2(1 + e^{-v_k})} \tag{32}$$

$$\frac{1}{\Psi_G}\frac{\partial \Psi_G}{\partial w_{kl}} = \frac{1}{1 + e^{-v_l}}\frac{x_k}{2\sigma^2} \tag{33}$$

$$\frac{1}{\Psi_G}\frac{\partial^2}{\partial X_k^2}\Psi_G = -\frac{1}{2\sigma^2} + \sum_j^N \frac{w_{kj}^2}{2\sigma^4}\frac{e^{-v_j}}{(1 + e^{-v_j})^2} + \frac{1}{4\sigma^4}\left(a_k - x_k + \sum_j^N \frac{w_{kj}}{1 + e^{-v_j}}\right)^2 \tag{34}$$

6 Results

6.1 One Particle in One Dimension

As a first test of the RBM, we apply it to the very simple simple case of a single particle in one dimension. We will use the standard metropolis sampling algorithm to begin with. Figure 1 shows the learned wavefunction after $2 \cdot 10^4$ iterations, using a learning rate of 0.9, and no regularization. The energy produced by this wavefunction comes out as $0.49999985 \pm 2 \cdot 10^{-6}$a.u., compared to the exact value of 0.5 a.u.. The error given here is the standard error of the mean local energy, as calculated with the Blocking method discussed in project 1, using 2^{20} samples. If we allow for more training, the energy is lower somewhat more, although the gains diminish as E_L approaches its ideal value. In any case, this is precision more than acceptable.

Applying regularization in this case, for almost any reasonable choice for γ (e.g. 0.1), the RBM converges *very* quickly to the exact wavefunction, by setting all parameters as small as possible. The effect of regularization is artificially magnified in this situation, as it so happens that the minimizing choice of parameters coincide with the minimizing choice for the regularization term, i.e the trivial $\boldsymbol{\alpha} = \mathbf{0}$ solution. We cannot expect this to work as well for less trivial tests, but it serves as a nice sanity check for the developed codes.

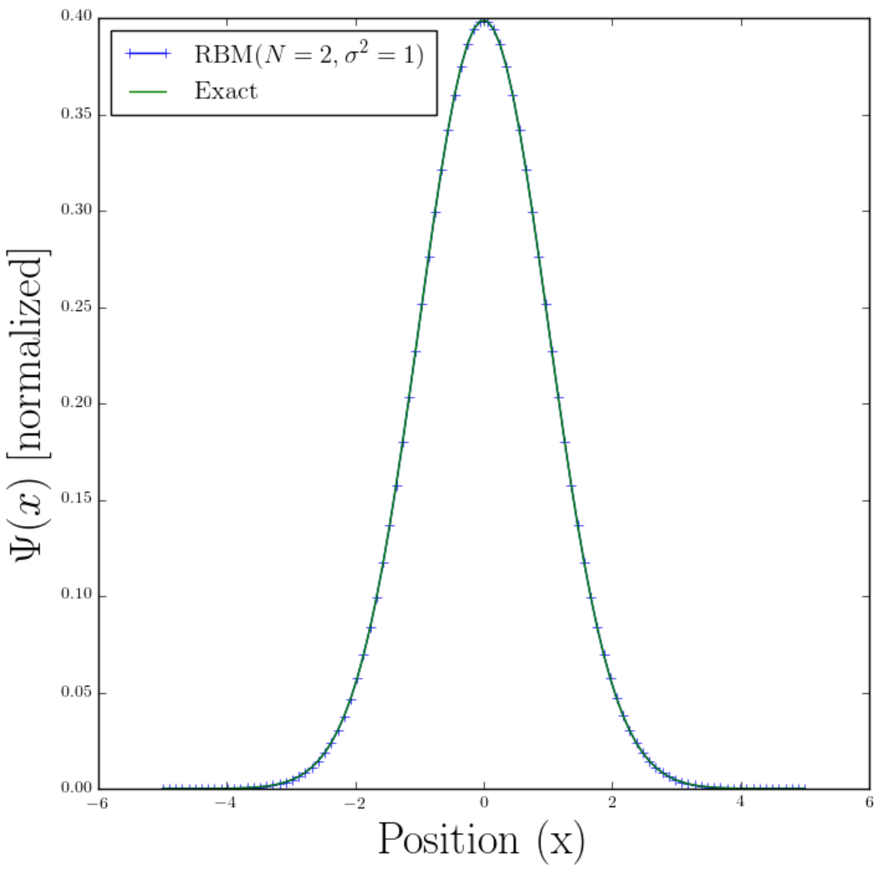

Figure 1: Ideal and learned wavefunction for one particle in one dimension.

Figure 2 shows the error we make as a function of training iterations for the three different samplers discussed. We can observe a couple of interesting points from this plot. First, brute force Metropolis sampling has, as expected, a significantly higher variance in energy compared with Metropolis-Hastings. Interestingly that does not appear to affect the learning much, as both algorithms follow roughly the same curve. However, we would prefer to use Metropolis-Hastings here due to the stability it provides.

Gibbs sampling affects the training in a noticeably different way. We see it starting out with slower convergence compared the others, but as we continue to train, the Gibbs sampling curve holds a steeper decline, and surpassing the results obtained by the Metropolis variants. The effect is small though, keeping in mind the logarithmic y-axis which acts to amplify such differences. In fact, only the slightly slower start is always reproduced, as we sometimes get stuck in a local minimum before Gibbs can catch up fully.

One thing to note is that the constant value assumed for σ^2 here was $\sigma^2 = 1$ for the Metropolis variants, and $\sigma^2 = 1/2$ for Gibbs. Both of these were chosen so that setting $\boldsymbol{\alpha} = \mathbf{0}$ would correspond to the ideal wavefunction. Using $\sigma^2 = 1$ with Gibbs makes it significantly worse. While this could potentially be helped by adding more parameters to the model (i.e. increasing N), this illustrates the potential issue with this general approach, namely the dependence on somewhat decent starting points.

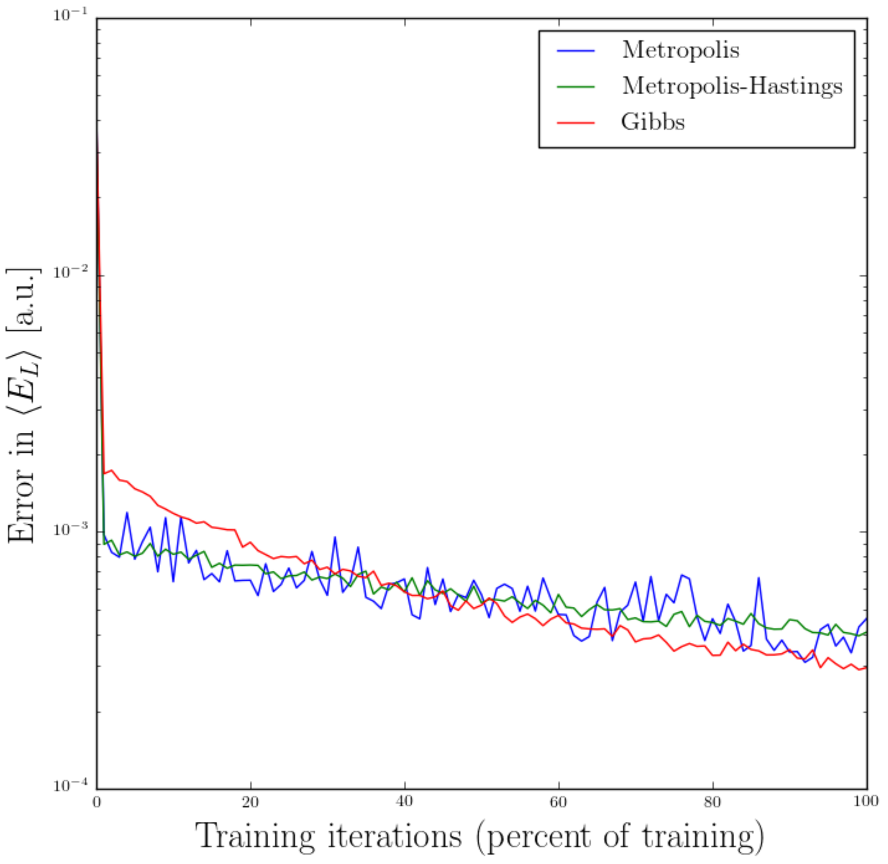

Figure 2: Learning progression, comparing the three different sampling algorithms. We see a slight difference, especially in the jaggedness of the curve, indicating a difference in variance.

Figure 3 shows a similar comparison but for various optimizers. We can see that choosing a smaller learning rate for SGD results in slower convergence. Ideally, given sufficient training iterations we expect the smaller η to result in slightly better accuracy, as it allows for more fine tuned adjustments. However, this is only guaranteed for ideally convex cost functions. Here we observe the small-η optimizer getting stuck at some local minimum, unable to get out due to the small learning rate. This goes to show the importance of choosing a suitable learning rate when using vanilla SGD. Although not shown here, choosing to large values can result in catastrophic results.

Better than both other methods is the Adam optmizer, here used with the default parameters given by the original authors [4]. Although the large-η SGD variant is initially somewhat faster, Adam ends up close to an order of magnitude better. Interestingly we appear to reach a point were further training reduces the energy slightly, while the variance remains roughly the same (causing an apparent increase in variance, due to the log-axis). In addition to the improved results, the main appeal of Adam is how well it performs without hyper-parameter tuning. Due to its adaptive nature, it performs well with the defaults in all scenarios attempted.

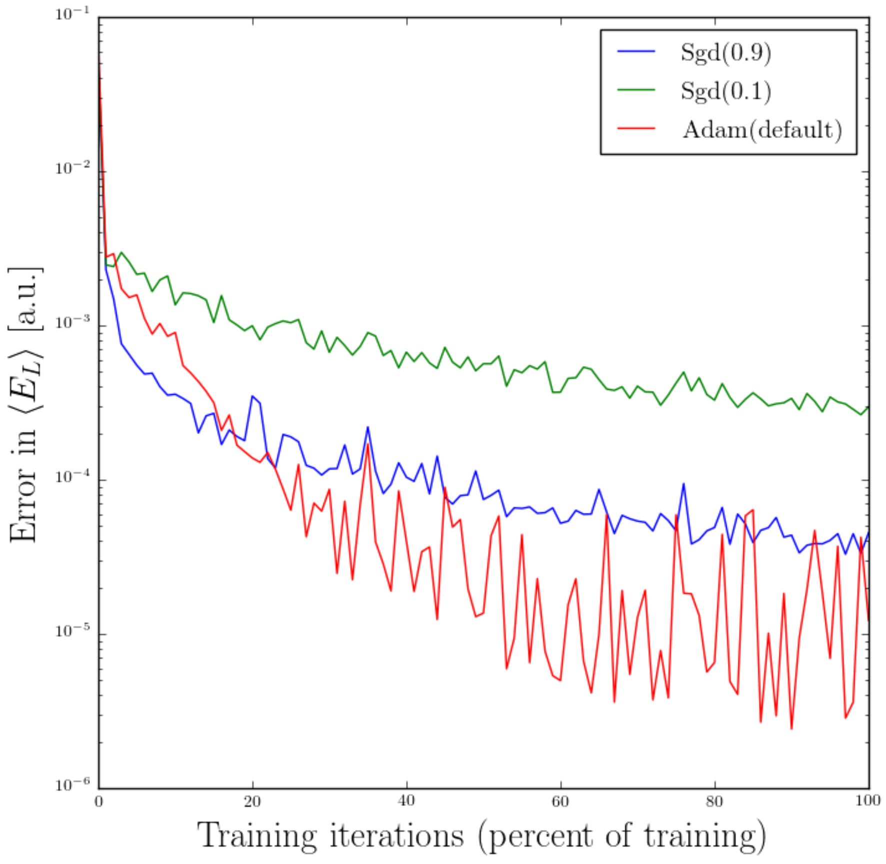

Figure 3: Learning progression, comparing different optimization algorithms. We observe Adam outperforming SGD, with the small learning rate converging very slowly. Training was performed using $4 \cdot 10^4$ iterations with 100 samples per iterations.

Figure 4 shows yet again a similar plot of training progression, this time for various values for N. Here we see that increasing N only served to slow down convergence significantly, as well as increasing the run-time. This results makes sense, considering that the ideal case has an optimum for $\boldsymbol{\alpha} = \mathbf{0}$. Adding more parameters just means it gets harder for the optimizer to locate the minimum, and increases the density of local, non-optimal minima. Increasing N might be more interesting when we consider the non-trivial interacting case.

References

1. 5 family=Carleo, family$_i = C., given = Giuseppe, given_i = G., 47\&5 family = Troyer, family_i = T., given = Matthias, given_i = M., 47$. Solving the quantum many-body problem with artificial neural networks. *Science* **355,** 602–606. ISSN: 0036-8075 (2017).

2. 5 family=Taut, family$_i = T., given = M., given_i = M., 47$. Two electrons in an external oscillator potential: Particular an-

alytic solutions of a Coulomb correlation problem. *Phys. Rev. A* **48,** 3561–3566 (5 Nov. 1993).

3. 5 family=Hjorth-Jensen, family$_i$ = $H.$-$J., given = Morten, given_i = M.$, 4 7. *Computational Physics 2: Variational Monte Carlo methods, Lecture Notes* Spring 2018. <`http://compphysics.github.io/ComputationalPhysics2/doc/web/course`>.

4. 5 family=Kingma, family$_i$ = $K., given = DiederikP., given_i = D.P.$, 47&5 $family = Ba, family_i = B., given = Jimmy, given_i = J.$, 47. Adam: A Method for Stochastic Optimization. *CoRR* **abs/1412.6980.** arXiv: `1412.6980`. <`http://arxiv.org/abs/1412.6980`> (2014).

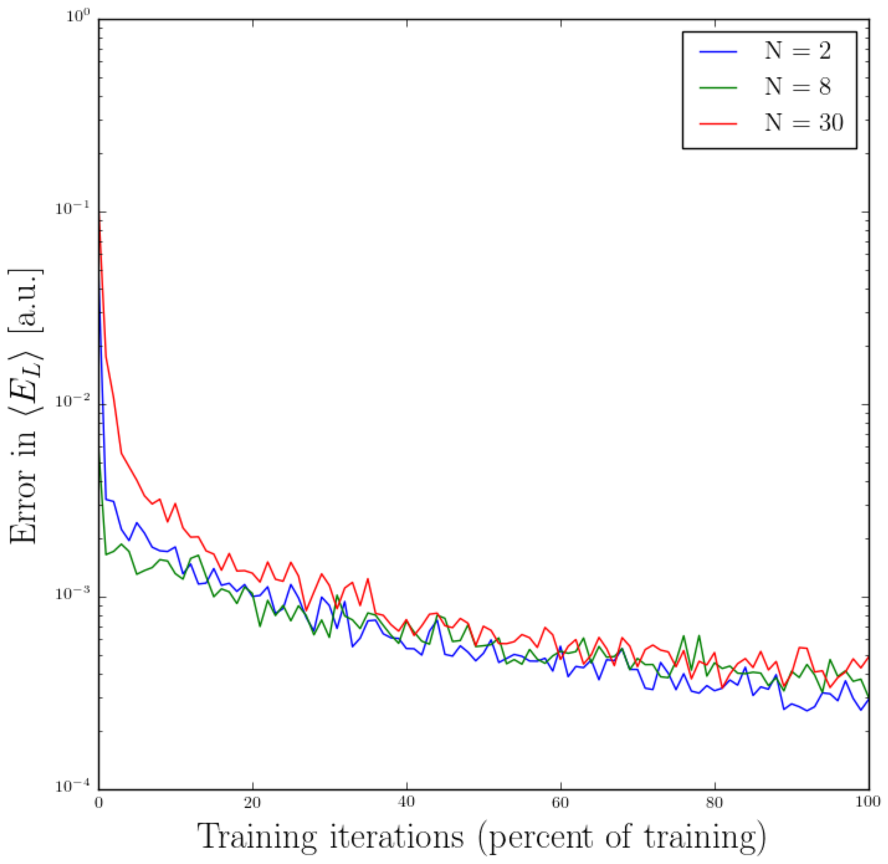

Figure 4: Learning progression, comparing different number of hidden neurons. Training was performed using $4 \cdot 10^4$ iterations with 100 samples per iterations, using Metropolis-Hastings sampling and Adam optimizer.

Appendices

A Analytic Expression for the Local Energy

To get started we will be needing a couple of partial derivatives:

$$\frac{\partial}{\partial X_k}(1+e^{v_j}) = e^{v_j}\frac{\partial}{\partial X_k}v_j = e^{v_j}\frac{w_{kj}}{\sigma^2} \tag{35}$$

$$\begin{aligned}\frac{\partial}{\partial X_k}e^{-\sum_i^M u_i} &= e^{-\sum_i^M u_i}\frac{\partial}{\partial X_k}\left(-\sum_i^M u_i\right)\\ &= e^{-\sum_i^M u_i}\frac{\partial}{\partial X_k}(-u_k)\\ &= -e^{-\sum_i^M u_i}\frac{x_k-a_k}{\sigma^2}\end{aligned} \tag{36}$$

We are now ready to compute the first derivate:

$$\begin{aligned}\frac{\partial}{\partial X_k}\Psi &= \frac{1}{Z}\left[\prod_j^N(1+e^{v_j})\frac{\partial}{\partial X_k}e^{-\sum_i^M u_i}\right.\\ &\quad\left.+e^{-\sum_i^M u_i}\frac{\partial}{\partial X_k}\prod_j^N(1+e^{v_j})\right]\\ &= -\frac{x_k-a_k}{\sigma^2}\Psi+\Psi\sum_j^N\left(\frac{1}{1+e^{v_j}}\frac{\partial}{\partial X_k}(1+e^{v_j})\right)\\ &= -\frac{x_k-a_k}{\sigma^2}\Psi+\Psi\sum_j^N\frac{1}{1+e^{v_j}}\left(\frac{w_{kj}}{\sigma^2}e^{v_j}\right)\\ &= \frac{1}{\sigma^2}\left(a_k-x_k+\sum_j^N\frac{w_{kj}}{1+e^{-v_j}}\right)\Psi.\end{aligned}$$

Before jumping into the second derivative, one more helpful derivative:

$$\begin{aligned}\frac{\partial}{\partial X_k}\left(\frac{w_{kj}}{1+e^{-v_j}}\right) &= -w_{kj}\frac{e^{-v_j}}{(1+e^{-v_j})^2}\frac{\partial(-v_j)}{\partial X_k}\\ &= \frac{w_{kj}^2}{\sigma^2}\frac{e^{-v_j}}{(1+e^{-v_j})^2}\end{aligned} \tag{37}$$

Now, finally the second derivative:

$$\begin{aligned}\frac{1}{\Psi}\frac{\partial^2}{\partial X_k^2}\Psi &= \frac{1}{\Psi}\frac{\partial}{\partial X_k}\left[\frac{1}{\sigma^2}\left(a_k-x_k+\sum_j^N\frac{w_{kj}}{1+e^{-v_j}}\right)\Psi\right]\\ &= \frac{1}{\sigma^2}\left[\frac{\partial}{\partial X_k}\left(a_k-x_k+\sum_j^N\frac{w_{kj}}{1+e^{-v_j}}\right)\right.\\ &\quad\left.+\left(a_k-x_k+\sum_j^N\frac{w_{kj}}{1+e^{-v_j}}\right)\frac{1}{\Psi}\frac{\partial}{\partial X_k}\Psi\right]\\ &= -\frac{1}{\sigma^2}+\sum_j^N\frac{w_{kj}^2}{\sigma^4}\frac{e^{-v_j}}{(1+e^{-v_j})^2}\\ &\quad+\frac{1}{\sigma^4}\left(a_k-x_k+\sum_j^N\frac{w_{kj}}{1+e^{-v_j}}\right)^2\end{aligned} \tag{38}$$

The final expression we shall use then for the local energy is:

$$E_L = \sum_{i=1}^P\frac{1}{2}r_i^2+\sum_{i<j}^P\frac{1}{r_{ij}}-\frac{1}{2}\sum_{k=1}^M\frac{1}{\Psi}\frac{\partial^2}{\partial x_k^2}\Psi, \tag{39}$$

substituting in Equation 38.