Quantum Boltzmann Machines and Auto-encoders

Henrik Modahl Breitenstein

April 25, 2024

Abstract

In this thesis we build a restricted Boltzmann machine to approximate the ground state energy of the Lipkin-Meshkow-Glick model, the Ising model, the Heisenberg model and the Pairing model with both Gibbs sampling and with a state acceptance criteria. We found that for the Lipkin model our implementation of the restricted Boltzmann machine ADD. For the Ising model it ADD. Furthermore the Heisenberg model's ground state energy was predicted with an accuracy ADD. Lastly the restricted Boltzmann machine managed to predict the ground state energy of the Pairing model with ADD. The Gibbs sampling method was found to be ADD than sampling with a acceptance criteria. We found our implementation compute the ground state energy for the Lipkin model at ADD, the Ising model at ADD, the Heisenberg model at ADD and for the Pairing at ADD. Compared to standard diagonalization methods we found that our implementation of the restricted Boltzmann machine ADD.

Contents

Chapter 1

Introduction

Machine learning is becoming more and more prevalent in everyday life. As hardware has become stronger and learning models has been refined, machine learning has become a powerful tool to solve a wide variety of problems. Physics is no exception to this, as the last few years has shown a number of fields in physics furthered by machine learning.
Neural networks has been used to solve partial differential equations in the light of fluid dynamics[1]. For easier imaging of single atoms by noise reduction[2]. For finding phase transitions[3], and much more.
Quantum physics has often avenues for direct application of machine learning, as one wants to fit a wavefunction within the bounds of the Hamiltonian. Neural networks can learn through real measurement data or through the information contained in the Hamiltonian itself. The neural network provides a guess for either the amplitude of a given state, or it can be set up to guess the whole wavefunction as a neural net quantum state as introduced by G. Carlo and M. Troyer in 2017[4].
Restricted Boltzmann machines is a type of neural network where the network constitutes a probability distribution. It was first proposed by P. Smolensky in 1986 [5] and brought into the spotlight by G. E. Hinton in 2006 [6]. For development of quantum technologies it is essential to be able to make analysis of quantum systems of increasing complexity. The RBM is capable of learning a probability distribution, something that makes it perfect to use for learning the wavefunction of a quantum system, which is a probability distribution over system states. The use of for machine learning to solve quantum systems has wide application [7][8], and in recent years the accessibility has grown through projects such as the python library NetKet[9][10][11].
However, the high flexibility of NetKet and similar projects can often be a hindrance to deeper understanding of the inner workings of the machine learning methods and algorithms. In this thesis we will look into the use of the restricted Boltzmann machine to find the ground state energy of four different quantum mechanical systems: the Lipkin-Meshkow-Glick model, the Ising model, the Heisenberg model and the Pairing model. For the ease of explaining, and for the proof of concept, the restricted Boltzmann machines and accompanying parts will be built from the ground up.
For a overview the thesis will consist of four parts. In the theory part of the thesis we will go over the relevant theory to understand the implementation,

which is described in the implementation part. We will then test and compare our implemented restricted Boltzmann machine with the previously mentioned quantum systems in mind. Lastly we will summarize and bring it together in the conclusion.

Part I

Theory

Chapter 2

Machine Learning

Machine learning is a machine adapting the way it processes some information by looking at its own output from said processing. A neural network is such a machine.

2.1 Basic Principles

2.1.1 Nodes and Layers

The fundamental building block of neural networks are nodes, which often is analogously seen as an neuron from in a brain, hence the name neural network. In a node we have the bias and the activation function and a weight for each connected node.

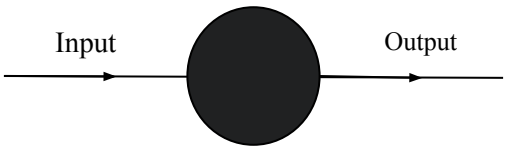


Figure 2.1: A singular node. Has an input which is affected by the weight, bias and activation function within the node.

In the figure above the node takes in a input from the left, x. The weight, w, and bias, b, create the intermediate output of the node accordingly:

$$n = wx + b\,. \tag{2.1}$$

Then n is passed through an activation function, σ and we get the output of the node

$$z = \sigma\left(n\right)\ . \tag{2.2}$$

Combining several nodes which takes the same input we get a layer. Layers are then connected where the previous layer's output is sent as input to the next via

these connections. This is the simplest network, called a feedforward network, as we pass the input through layer by layer without any inter-layer connections.

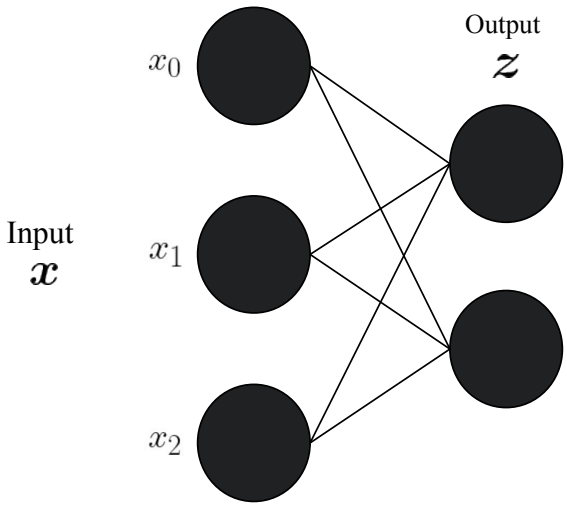


Figure 2.2: A simple feedforward neural network where we have an input layer, individual connections to the input, and a output layer. The output layer's nodes are all connected to each of the input layer's nodes.

Here the first layer is called an input layer as it has a one to one correspondence with a input data point. The input and output layer is fully connected however, where the output of each of the input layer's nodes are sent to each of the output layer's, where each connection has its own weight. Using vectors to represent the layers we have

$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{2.3}$$

as input, which is then inserted in the input layer's nodes as shown in 2.2. The weights now has an extra dimension as each nodes is connected to each node in the next layer, so we can represent it with a matrix:

$$W = \begin{bmatrix} w_{00} & w_{10} & w_{20} \\ w_{01} & w_{11} & w_{21} \end{bmatrix} \tag{2.4}$$

where the weight w_{kj} is the weight of the connection between node k of the input layer and node j of the output layer. Feeding the input forward we first get the intermediate output

$$\boldsymbol{n} = W\boldsymbol{x} + \boldsymbol{b} \tag{2.5}$$

where we use simple matrix multiplication. The output layer often has a unique activation function to make the output more applicable to a certain context. For example, in classification problems the softmax function

$$\sigma_s(\boldsymbol{z}')_i = \frac{e^{z'_i}}{\sum_{j=1}^{N} e^{z'_j}} \ , \tag{2.6}$$

which normalizes the output to a probability distribution, is often used. Using the softmax activation function, we have the final output

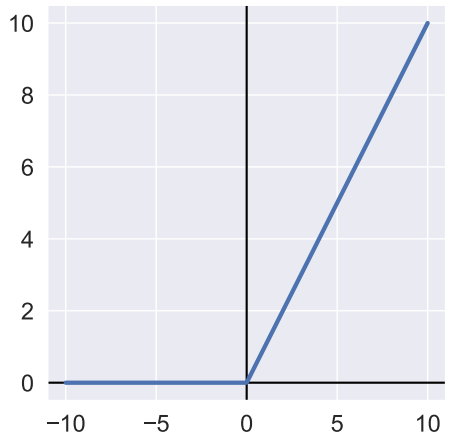
$$\boldsymbol{z} = \sigma_s\left(\boldsymbol{n}\right) \ . \tag{2.7}$$

The network model is easily expandable to more layers as one just sends the output of a layer into the next by following equation 2.5.

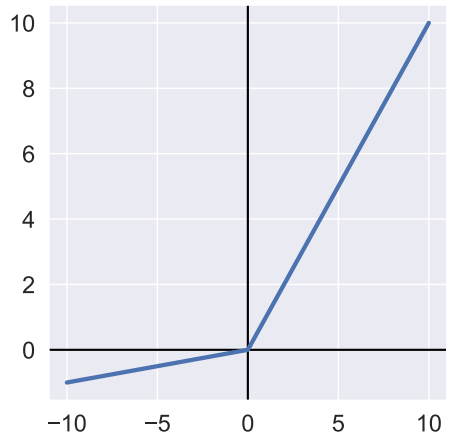
2.1.2 Activation function

The term activation function is inspired from the way biological neurons 'activate' after a certain signal threshold is met. The activation function chosen for a given network can have large impact on the finished trained model. Some common activation functions are

ReLU
$max\left(0, x\right)$



The Rectified Linear Unit activation function is one of the most basic activation functions. Since it does not have an upper bound, the activation function can have problems with values becoming too large. ReLU is however one of the most popular activation function because it has an derivative easy to calculate and avoids the vanishing gradient problem during learning.

Leaky ReLU $max\,(ax, x)\ \ a \in \langle 0, 1\rangle$	

Leaky ReLU adds the possibility for negative values to impact the output of the node.

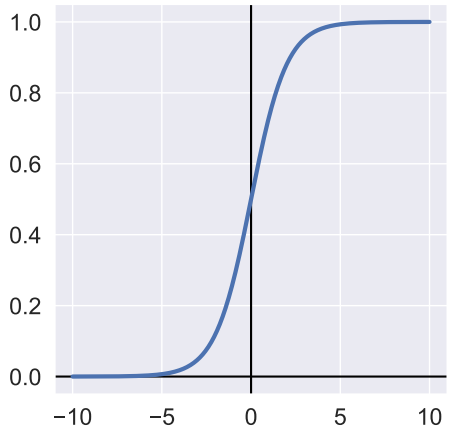
Sigmoid $\sigma\,(x) = \frac{1}{1+e^{-x}}$	

Table 2.1

The Sigmoid activation function has both a upper and lower bound, preventing the values from escalating out of control, but has an more computationally expensive derivative and can struggle with the vanishing gradient problem during learning.

2.1.3 Training a neural network

Training a neural network is done by changing the weights and biases such that the output comes closer to a desired goal. This goal is to minimize a cost function C, which can be any function. As an example, an common cost function for regression is the squared error

$$C = \sum_k (t_k - z_k)^2 \tag{2.8}$$

where t is a target value and z is the model output. We want to change the weights and biases in such a way that the cost function decreases. A natural way would be to use the gradient of the cost function in terms of the weights and biases, and decrease the variables by a proportional amount, such that we approach the minima of the cost function. Finding the gradient of the cost function in terms of the weights and biases is done by backpropagation, a method introduced by S. Linnainmaa in 1960 [12], completed by F. Rosenblatt in 1976 [13] and popularized by D. E. Rumelhart in 1982 [14]. The method is derived by application of the chain rule starting from the output and going layer by layer to get to the input.
We expand our simple input-output model 2.2 and define a subscript for the different layers and nodes as shown here:

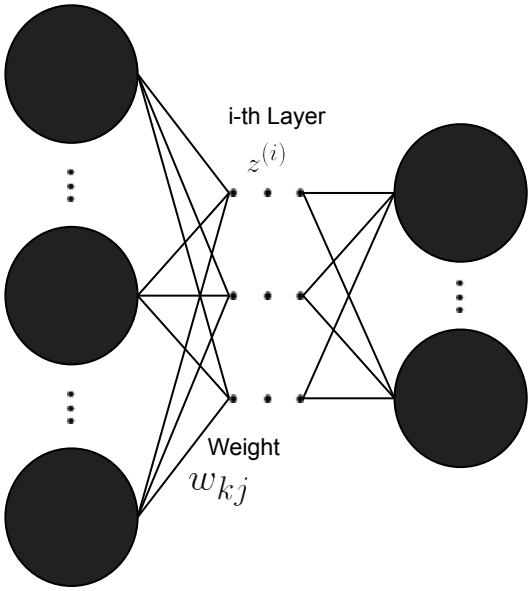


Figure 2.3: A more general network where we w_{kj} is the weight matrix

We have that:

$\boldsymbol{z}^{(i)}$ is the i-th layer vector output. (i) is the layer subscript, going from the input layer $z^{(0)}$ to the output layer $z^{(L)}$.

$\boldsymbol{n}$ is the intermediate output of a layer before it is sent through an activation function.

σ is an activation function.

$W^{(i)}$ is a weight matrix with weights w_{kj}, which is the weight from k-th node of the i-th layer to node j of layer $i+1$.

Starting from the output we want to find

$$\Delta W^{(L-1)} = -\eta \frac{\partial C}{\partial W^{(L-1)}} \, , \tag{2.9}$$

where η is a chosen learning rate. The change in a single weight

$$\Delta w_{kj}^{(L-1)} = -\eta \frac{\partial C}{\partial w_{kj}^{(L-1)}} \ , \tag{2.10}$$

where we can add the dependency on the activation function and intermediate output

$$\Delta w_{kj}^{(L-1)} = -\eta \frac{\partial C}{\partial z_k^{(L)}} \frac{\partial z_k^{(L)}}{\partial n_k^{(L)}} \frac{\partial n_k^{(L)}}{\partial w_{kj}^{(L-1)}} \tag{2.11}$$

Where we have that

$$\frac{\partial C}{\partial z_k^{(L)}} = -2\left(t_k - z_k^{(L)}\right) \tag{2.12}$$

and using the sigmoid activation function, 2.1, we have

$$\frac{\partial z_k^{(L)}}{\partial n_k^{(L)}} = \frac{\partial\left(\frac{1}{1=e^{-n_k}}\right)}{\partial n_k} = \frac{e^{-n_k}}{\left(1+e^{-n_k}\right)^2} = n_k(1-n_k) \tag{2.13}$$

and then lastly the intermediate output

$$\frac{\partial n_k^{(L)}}{\partial w_{kj}^{(L-1)}} = \frac{\partial\left(w_{kj}^{(L-1)} z_j^{(L-1)} + b_j^{(L-1)}\right)}{\partial w_{kj}^{(L-1)}} = z_j^{(L-1)} \tag{2.14}$$

So we can write the change of weight as

$$\Delta w_{kj}^{(L-1)} = \eta\left(t_k - z_k^{(L)}\right) z_k^{(L)}\left(1 - z_k^{(L)}\right) z_j^{(L-1)} \ . \tag{2.15}$$

We simplify by defining

$$\delta_k^{(L)} = \left(t_k - z_k^{(L)}\right) z_k^{(L)}\left(1 - z_k^{(L)}\right) \tag{2.16}$$

and get

$$\Delta w_{kj}^{(L-1)} = \eta \delta_k^{(L)} z_j^{(L-1)} \ , \tag{2.17}$$

where we have shortened the -2 factor into η as well. For the next layer, $(L-2)$, we have that a individual node is connected to each of the previous layer's nodes. We then have the change

$$\Delta w_{jm}^{(L-2)} = \eta\left[\sum_k \frac{\partial C}{\partial z_k^{(L)}} \frac{\partial z_k^{(L)}}{\partial n_k^{(L)}} \frac{\partial n_k^{(L)}}{\partial z_j^{(L-1)}}\right] \frac{\partial z_j^{(L-1)}}{\partial n_j^{(L-1)}} \frac{\partial n_j^{(L-1)}}{\partial w_{jm}^{(L-2)}} \ , \tag{2.18}$$

where we have calculated the expression within the sum already

$$\frac{\partial C}{\partial z_k^{(L)}} \frac{\partial z_k^{(L)}}{\partial n_k^{(L)}} \frac{\partial n_k^{(L)}}{\partial z_j^{(L-1)}} = \left(t_k - z_k^{(L)}\right) z_k^{(L)}\left(1 - z_k^{(L)}\right) w_{kj}^{(L-1)} = \delta_k^{(L)} w_{kj}^{(L-1)} \ . \tag{2.19}$$

The expression outside the sum is calculated as in 2.13 and 2.14

$$\frac{\partial z_j^{(L-1)}}{\partial n_j^{(L-1)}} \frac{\partial n_j^{(L-1)}}{\partial w_{jm}^{(L-2)}} = z_j^{(L-1)} \left(1 - z_j^{(L-1)}\right) z_m^{(L-2)} \,. \tag{2.20}$$

We can then write the change in weight as

$$\Delta w_{jm}^{(L-2)} = \eta \left[\sum_k \delta_k^{(L)} w_{kj}^{(L-1)}\right] z_j^{(L-1)} \left(1 - z_j^{(L-1)}\right) z_m^{(L-2)} \,, \tag{2.21}$$

which we can further simplify by defining

$$\delta_j^{(L-1)} = \left[\sum_k \delta_k^{(L)} w_{kj}^{(L-1)}\right] z_j^{(L-1)} \left(1 - z_j^{(L-1)}\right) \tag{2.22}$$

and we get

$$\Delta w_{jm}^{(L-2)} = \eta \delta_j^{(L-1)} z_m^{(L-2)} \,. \tag{2.23}$$

This process can then be repeated easily with the fact that

$$\delta_k^{i-1} = \left[\sum_k \delta_k^{(i)} w_{kj}^{(i-1)}\right] z_j^{(i-1)} \left(1 - z_j^{(i-1)}\right) \tag{2.24}$$

for any layers further in.

2.1.4 Adaptive learning rate

The learning rate has a large impact on the resulting trained machine. A constant learning rate will make the training susceptible to small local minima of the cost function, and a way to combat this is to introduce an adaptive learning rate. We will use momentum:

$$\begin{aligned} v_t &= v_{t-1}\gamma + \eta \nabla C\left(\theta_{t-1}\right) \\ \theta_t &= \theta_{t-1} - v_t \end{aligned} \tag{2.25}$$

where γ is a hyperparameter and η is the learning rate. The v_t is the "momentum" of the gradient decent of parameter θ of the current timestep, and v_{t-1} is that of the previous timestep. It accumulates as the gradient continues in the same direction, and stumbling upon a small local minima will have to overcome the momentum first, which hopefully prevents the gradient decent from getting stuck.

2.2 Boltzmann Machine

In this thesis we are going to use a slightly different type of neural network, called a Boltzmann machine. First major contribution to Boltzmann machines comes from G. E Hinton and T.J. Sejnowski in 1983 [15]. The neural net is a generative model which learns by matching the probability distribution of its inputs.

2.2.1 Structure and training

A Boltzmann machine has interconnected nodes within a layer.

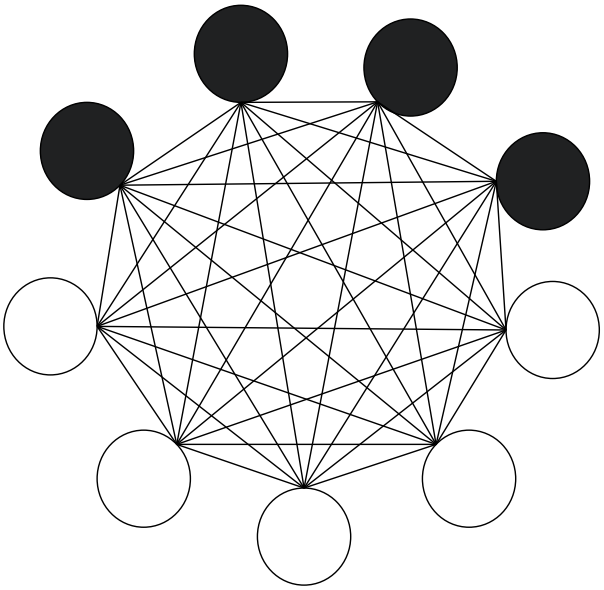


Figure 2.4: A unrestricted Boltzmann machine where every node is connected. Here the black nodes are the visible layer while the white nodes constitute the hidden layer.

The nodes are binary, being able to take the value 0 or 1, have a weights w_{ij} for connection strength between node v_i and h_j and biases a_i for the visible layer and b_j for the hidden layer. For a system of N_h hidden neurons and N_v visible neurons we have the probability distribution of nodes taking the value 1 defined as

$$P(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \ , \tag{2.26}$$

where the energy of the model is given by

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i}^{N_v} a_i v_i - \sum_{j}^{N_h} b_j h_j - \sum_{i}^{N_v} \sum_{j}^{N_h} v_i w_{ij} h_j \tag{2.27}$$

and the normalization factor

$$Z = \sum_{\boldsymbol{v}, \boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \ , \tag{2.28}$$

where we sum over all possible states of the model, which increases exponentially as $2^{(N_v + N_h)}$. The marginal distribution over the visual layer can be written as

$$P(\boldsymbol{v}) = \sum_{\boldsymbol{h}} \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \tag{2.29}$$

To train a Boltzmann machine we need to have a cost function that compares the predicted distribution of the model and the actual distribution of the data set. As such we will use the Kullback-Leibler divergence as a cost function:

$$KL(\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}) = \sum_{(\boldsymbol{v},\boldsymbol{h})} R(\boldsymbol{v}) \log \frac{R(\boldsymbol{v})}{P(\boldsymbol{v})} \,, \tag{2.30}$$

where $R(\boldsymbol{v})$ is the distribution we want to approximate and $P(\boldsymbol{v})$ is the distribution of the neural network model. Following the derivations of A.L. Yuille [16] we have that

$$\frac{\partial KL(\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b})}{\partial w_{ij}} = -\sum_{(\boldsymbol{v},\boldsymbol{h})} \frac{R(\boldsymbol{v})}{P(\boldsymbol{v})} \frac{\partial P(\boldsymbol{v})}{\partial w_{ij}} \,, \tag{2.31}$$

where we further have

$$\frac{\partial P(\boldsymbol{v})}{\partial w_{ij}} = \frac{1}{Z} \frac{\partial}{\partial w_{ij}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} - \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \frac{\partial \log Z}{\partial w_{ij}} \tag{2.32}$$

which we can express as

$$\frac{\partial P(\boldsymbol{v})}{\partial w_{ij}} = -\sum_{\boldsymbol{h}} v_i h_j P(\boldsymbol{v}, \boldsymbol{h}) + \sum_{\boldsymbol{h}} \left[P(\boldsymbol{v}, \boldsymbol{h}) \sum_{\boldsymbol{v},\boldsymbol{h}} v_i h_j P(\boldsymbol{v}, \boldsymbol{h}) \right] \,. \tag{2.33}$$

Then

$$\frac{\partial P(\boldsymbol{v})}{\partial w_{ij}} = -\sum_{\boldsymbol{h}} v_i h_j P(\boldsymbol{v}, \boldsymbol{h}) + P(\boldsymbol{v}, \boldsymbol{h}) \sum_{\boldsymbol{v},\boldsymbol{h}} v_i h_j P(\boldsymbol{v}, \boldsymbol{h}) \,. \tag{2.34}$$

Using the result of equation 2.34 in equation 2.31 we get that

$$\frac{\partial KL(\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b})}{\partial w_{ij}} = \sum_{\boldsymbol{v},\boldsymbol{h}} v_i h_j \frac{P(\boldsymbol{v}, \boldsymbol{h})}{P(\boldsymbol{v})} R(\boldsymbol{v}) - \left[\sum_{\boldsymbol{v},\boldsymbol{h}} R(\boldsymbol{v}) \right] \sum_{\boldsymbol{v},\boldsymbol{h}} v_i h_j P(\boldsymbol{v}, \boldsymbol{h}) \,, \tag{2.35}$$

which we can simplify

$$\frac{\partial KL(\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b})}{\partial w_{ij}} = \sum_{\boldsymbol{v},\boldsymbol{h}} v_i h_j P(\boldsymbol{h}|\boldsymbol{v}) R(\boldsymbol{v}) - \sum_{\boldsymbol{v},\boldsymbol{h}} v_i h_j P(\boldsymbol{v}, \boldsymbol{h}) \,, \tag{2.36}$$

where we require that

$$\frac{\partial \log Z}{\partial w_{ij}} = \sum_{\boldsymbol{v},\boldsymbol{h}} v_i h_j P(\boldsymbol{v}, \boldsymbol{h}) \,. \tag{2.37}$$

We then define the expectation, or correlation, values

$$\langle v_i h_j \rangle_{\text{data}} = P(\boldsymbol{h}|\boldsymbol{v}) R(\boldsymbol{v}) \tag{2.38}$$

and

$$\langle v_i h_j \rangle_{\text{model}} = P(\boldsymbol{v}, \boldsymbol{h}) \,. \tag{2.39}$$

This gives us the update rule

$$\Delta w_{ij} = -\eta \left(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \right) \,. \tag{2.40}$$

2.2.2 Restricted Boltzmann machine

Estimating $\langle v_i h_j \rangle_{\text{data}}$ and $\langle v_i h_j \rangle_{model}$ is done by Gibbs sampling, which is explained in a later chapter, but can be inefficient and take a long time to converge for complex models. Removing the weights between nodes within the same layer we can alleviate much of the computational cost of training. This type of Boltzmann machine is called a restricted Boltzmann machine, or RBM:

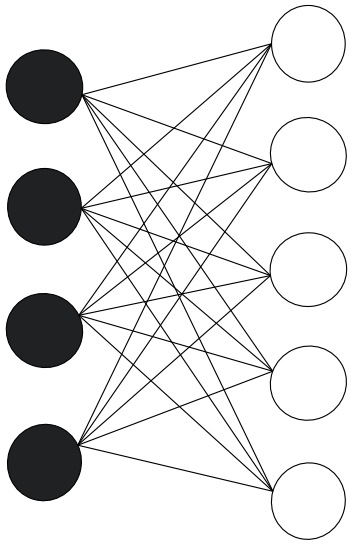


Figure 2.5: A restricted Boltzmann machine where there are no connections between nodes within the same layer. The white nodes are hidden while the black ones are the visible nodes.

Making the nodes independent of nodes in the same layer means we can write the conditional distributions as

$$P(\boldsymbol{v}|\boldsymbol{h}) = \prod_{i \in \boldsymbol{v}} P(v_i|\boldsymbol{h}) \, , \tag{2.41}$$

and

$$P(\boldsymbol{h}|\boldsymbol{v}) = \prod_{j \in \boldsymbol{h}} P(h_j|\boldsymbol{v}) \, . \tag{2.42}$$

In a RBM we first have a forward pass where we insert the input data into the visual layer, then we sample the hidden layer by the distribution:

$$p(h_j^{(0)} = 1|\boldsymbol{z}_v^{(0)}) = \sigma\left(\boldsymbol{z}_v^{(0)} \otimes \boldsymbol{W} + \boldsymbol{b}\right) \, , \tag{2.43}$$

where our activation function σ is the Sigmoid function 2.1. The index (0) indicate that it is the first pass-through the neural network. As the nodes are binary we then take a sample from $p(h_j^{(0)} = 1|\boldsymbol{z}_v^{(0)})$ as a Bernoulli distribution, which means each $z_{h,j}^{(0)}$ takes the value 1 with probability $h_j^{(0)}$. After the forward pass we have a backward pass where we sample from the hidden layer

$$p(v_j^{(1)} = 1|\boldsymbol{z}_h^{(0)}) = \sigma\left(\boldsymbol{z}_h^{(0)} \otimes \boldsymbol{W} + \boldsymbol{a}\right) \ , \tag{2.44}$$

where we then convert it to binary values as well. For a continuous valued output it is optional to let the last visual output to remain as a probability distribution. Estimating $\langle v_i h_j \rangle_{\text{data}}$ is done by sampling from $P(\boldsymbol{h}|\boldsymbol{v})$

$$\langle \boldsymbol{vh} \rangle_{\text{data}} = \boldsymbol{z}_v^{(0)} \otimes p(\boldsymbol{h}^{(0)}|\boldsymbol{z}_v^{(0)}) \tag{2.45}$$

,

while estimating $\langle v_i h_j \rangle_{\text{model}}$ requires that we let the model sufficiently affect the output. To do this we do Gibbs sampling through k iterations of the forward and backward passes. Then we have

$$\langle \boldsymbol{vh} \rangle_{\text{model}} = \boldsymbol{z}_v^{(k)} \otimes p(\boldsymbol{h}^{(k)}|\boldsymbol{z}_v^{(k)}) \tag{2.46}$$

.

And from 2.40 the change in weight becomes

$$\Delta \boldsymbol{W} = \eta \left[\boldsymbol{z}_v^{(0)} \otimes p(\boldsymbol{h}^{(0)}|\boldsymbol{z}_v^{(0)}) - \boldsymbol{z}_v^{(k)} \otimes p(\boldsymbol{h}^{(k)}|\boldsymbol{z}_v^{(k)})\right] \ . \tag{2.47}$$

2.2.3 Neural net quantum state

To apply the restricted Boltzmann machine on a quantum mechanical system, a way to represent the system's state is needed. As the visual layer the output of the machine, and can here be viewed as the collapsed state of the system after measurement, we use the marginal distribution of the visual layer, the probability distribution of the visual layer states given the state of the hidden layer, to represent the wavefunction of the system.

$$\Psi_{rbm}(\boldsymbol{v}) = \frac{1}{Z} \sum_{\boldsymbol{h}} \exp\{-E(\boldsymbol{v}, \boldsymbol{h})\} \tag{2.48}$$

$$= \frac{1}{Z} e^{-\sum_i^{N_v} \frac{(x_i - a_i)^2}{2}} \prod_j^{N_h} (1 + e^{b_j + \sum_i^{N_v} x_i w_{ij}}) \ , \tag{2.49}$$

This is most often referred to as a neural net quantum state, or abbreviated NQS.

2.2.4 Minimizing local energy

We want a restricted boltzmann machine to match the distribution of the ground state of a quantum mechanical systems. This poses a problem with the use of the change in weight, 2.47, as we do not have any data to train the model on. Instead we will use the variational principle with the fact that

$$E\left[\Psi_{rbm}\right] \geq E_0 \ , \tag{2.50}$$

where Ψ_{rbm} is the machine state and E_0 is the ground state of the quantum system we want the machine state to match. Because of 2.50 we can variationly minimize the expected energy and be certain that it approaches the ground

state. This poses another problem, though, because the machine state is not directly accessible, but one can take samples from the machine's distribution. So instead of using the machine state directly, one approximates it by a sufficient amount of samples, essentially taking repeated measurements of the system and constructing the wavefunction from the result. For N total number of samples, and M_k the number of samples that are the basis state $|b_k\rangle \in \boldsymbol{B}$, then we have the approximate machine state:

$$\Psi_{rbm} \approx \sqrt{\frac{M_0}{N}}\,|b_0\rangle + \sqrt{\frac{M_1}{N}}\,|b_1\rangle + \cdots + \sqrt{\frac{M_N}{N}}\,|b_N\rangle \ , \tag{2.51}$$

where we assume the wavefunction is real and positive definite. We use the samples to then approximate the energy by finding the expectation value of their local energy. The local energy is defined as:

$$E_L(s) = \frac{\langle s|\, H\, |\Psi_{rbm}\rangle}{\langle s|\Psi_{rbm}\rangle} \ , \tag{2.52}$$

where $\langle s|$ is a sample. The system energy can then be approximated

$$\langle E\rangle \approx \langle E_L\rangle = \frac{1}{N}\sum_{k=0}^{N} E_L(s_k) \ . \tag{2.53}$$

So minimizing the energy of the machine state can be done through minimizing the local energy. To derive the cost function, our gradient for the gradient decent calculations, we will take inspiration from this derivation [17] of unknown author. Starting of with the expected value of the hamiltonian

$$\langle H\rangle = \frac{\int \mathrm{d}\Psi(X)^* H\Psi(X)}{\int \mathrm{d}X\Psi^*(X)\Psi(X)} \ , \tag{2.54}$$

we then have the derivative by using the chain rule

$$\frac{\partial}{\partial\alpha}\langle H\rangle = \frac{\int \mathrm{d}X\Psi_\alpha^* H\Psi + \Psi^* H\Psi_\alpha}{\int \mathrm{d}X\Psi^*\Psi} - \frac{\left(\int \mathrm{d}X\Psi^* H\Psi\right)\left(\int \mathrm{d}X\Psi_\alpha\Psi + \Psi^*\Psi_\alpha\right)}{\left(\int \mathrm{d}X\Psi^*\Psi\right)^2} \ , \tag{2.55}$$

where Ψ_α is used to denote $\frac{\partial\Psi}{\partial\alpha}$. Now we expand the integrals of the numerator of the second term by $\Psi^*\Psi$ and we get

$$\frac{\partial}{\partial\alpha}\langle H\rangle = \frac{\int \mathrm{d}X\ \Psi_\alpha^* H\Psi + \Psi^* H\Psi_\alpha}{\int \mathrm{d}X\ \Psi^*\Psi} - \langle E_L\rangle\left\langle \frac{\partial}{\partial\alpha}\ln|\Psi|^2\right\rangle \ . \tag{2.56}$$

We then use the fact that the hamiltonian is Hermitian:

$$\int \mathrm{d}X\ \Psi^* H\Psi_\alpha = \int \mathrm{d}X\ \Psi_\alpha (H\Psi)^* \ , \tag{2.57}$$

and we get

$$\frac{\partial}{\partial\alpha}\langle H\rangle = \frac{\int \mathrm{d}X\ \Psi_\alpha^* H\Psi + \Psi_\alpha (H\Psi)^*}{\int \mathrm{d}X\ \Psi^*\Psi} - \langle E_L\rangle\left\langle \frac{\partial}{\partial\alpha}\ln|\Psi|^2\right\rangle . \tag{2.58}$$

then expanding the first terms numerator integrals by $\Psi^*\Psi$ we end up with

$$\frac{\partial}{\partial\alpha}\langle H\rangle = \left\langle \frac{\Psi_\alpha^*}{\Psi^*}\frac{H\Psi}{\Psi} + \frac{\Psi_\alpha}{\Psi}\left(\frac{H\Psi}{\Psi}\right)^* \right\rangle - \langle E_L\rangle \left\langle \frac{\partial}{\partial\alpha}\ln|\Psi|^2 \right\rangle \tag{2.59}$$

$$= \left\langle \frac{\Psi_\alpha^*}{\Psi^*}E_L + \frac{\Psi_\alpha}{\Psi}E_L^* \right\rangle - \langle E_L\rangle \left\langle \frac{\partial}{\partial\alpha}\ln|\Psi|^2 \right\rangle \ . \tag{2.60}$$

If we now assume that Ψ is real, and then E_L would also be real, we can shorten the gradient

$$\frac{\partial}{\partial\alpha}\langle H\rangle = 2\left(\langle E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}\rangle - \langle E_L\rangle\langle\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha}\rangle\right) \ , \tag{2.61}$$

where α then is our collection of weights and biases

2.2.5 Change in weights and biases

For practical use of the cost function 2.61 defined above, we need to derive the change in weights and biases. With the fact that $\frac{1}{\Psi}\frac{\partial\ln\Psi}{\partial\alpha} = \frac{\partial\ln\Psi}{\partial\alpha}$ together with

$$\ln\Psi(\boldsymbol{v}) = -\ln Z - \sum_i^{N_v}\frac{(v_i - a_i)}{2} + \sum_k^{N_h}\ln\left(1 + \exp\left\{b_k + \sum_j^{N_v} v_j w_{jk}\right\}\right) \ , \tag{2.62}$$

we can then find that

$$\frac{\partial}{\partial a_i}\ln\Psi = v_i - a_i \tag{2.63}$$

$$\frac{\partial}{\partial b_j}\ln\Psi = \left(\exp\left\{-b_j - \sum_k^{N_v} v_k w_{kj}\right\} + 1\right)^{-1} \tag{2.64}$$

$$\frac{\partial}{\partial w_{ij}}\ln\Psi = v_i\left(\exp\left\{-b_j - \sum_k^{N_v} v_k w_{kj}\right\} + 1\right)^{-1} \ . \tag{2.65}$$

2.3 Monte Carlo Methods

Monte Carlo methods is a way to gain insight into a probability distribution by using random samples from said distribution. As an example problem one is to calculate the integral

$$I(h, P) = \int h(x)P(x)\,dx,$$

where $h(x)$ is an arbitrary function and $P(x)$ is a probability distribution. Then by taking random samples from $P(x) \to x_s$, one could estimate the integral by

$$I(h, P) \approx \frac{1}{N}\sum_{s=1}^{N} h(x_s).$$

Which is simple enough when one can sample from the target distribution, here $P(x)$, something that is not always the case.

2.3.1 Importance sampling

When the distribution $P(x)$ is unknown we can instead draw our sample from a proposed distribution $Q(x)$ and then use importance weights to correct it. Then continuing with the example above we have

$$I(h, P) = \int \frac{h(x)P(x)}{Q(x)} Q(x)\, dx.$$

And our approximate becomes

$$I(h, P) \approx \frac{1}{N} \sum_{s=1}^{N} \frac{h(x_s)P(x_s)}{Q(x_s)}.$$

This requires a that $Q(x)$ is somewhat close to that of $P(x)$, which is not necessarily easy to construct. It is therefore better to use a adaptive proposal distribution instead.

2.3.2 Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo method and introduces feature adaptive proposal distributions. For a desired distribution $P(x)$, a Markov chain describes the probability of a series of events where we have the transition probability $P(x'|x)$ of transitioning from state x to x'. Adapting the our proposed distribution $Q(x)$ by using Markov chains we move towards a stationary distribution where we then are at a equilibrium:

$$P(x'|x)P(x) = P(x|x')P(x') \,, \tag{2.66}$$

where the transition from x to x' is equally likely both ways. We can rewrite this as

$$\frac{P(x'|x)}{P(x|x')} = \frac{P(x')}{P(x)} \,. \tag{2.67}$$

The Metropolis-Hastings method is to set up a proposal $Q(x'|x)$ and then use a acceptance distribution $A(x', x)$ to either accept or reject samples from the proposed distribution. We can then rewrite the transition probability as

$$P(x'|x) = Q(x'|x)A(x', x) \,. \tag{2.68}$$

And then equilibrium equation 2.67 can then be written as

$$\frac{A(x', x)}{A(x|x')} = \frac{P(x')Q(x|x')}{P(x)Q(x'|x)} \,, \tag{2.69}$$

where we define the acceptance ratio

$$A(x', x) = \min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right) \,. \tag{2.70}$$

For each iteration i we have the state x' taken at random from $Q(x'|x_i)$ which then is either accepted by $x_{i+1} = x'$ or rejected by $x_{i+1} = x_i$. The distribution of $\{x_i\}$ will then approach our desired distribution $P(x)$ after a sufficient number of iterations.

2.3.3 Gibbs Sampling

Gibbs sampling is a special use case where we accept every suggested state. It is then necessary to make the proposed distribution as close to the actual distribution as possible. This is accomplished by sampling back and forth between the visual and hidden layer.

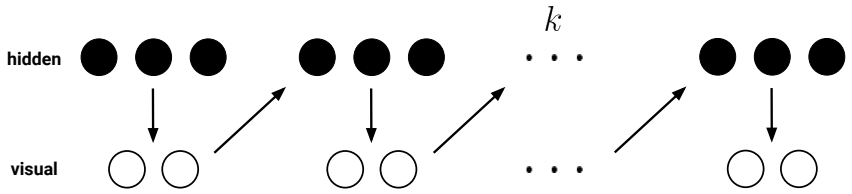


Figure 2.6: Gibbs sampling done k times. The first hidden layer is generated randomly by a uniform distribution.

Sampling of a layer is done by the conditional distributions:

$$P(\boldsymbol{v}|\boldsymbol{h}) = \prod_{i \in \boldsymbol{v}} P(v_i|\boldsymbol{h}) \tag{2.71}$$

$$P(\boldsymbol{h}|\boldsymbol{v}) = \prod_{j \in \boldsymbol{h}} P(h_j|\boldsymbol{v}) \ . \tag{2.72}$$

Repeated sampling converges, as shown by Christian P. Robert and George Casella in "Monte Carlo Statistical Methods", [18] (chapter 10.2.1 page 378).

2.4 Controls and Errors

Seeing how accurate a model is can be use full both during training, to see how well it optimizes, and for a trained model. The simplest way to check the accuracy is to look at the difference from the true value of whatever the model was supposed to find. If we define the model's predicted answer as z_m and the true answer as z_t we have that the answer is:

$$Error = |z_t - z_m| \ , \tag{2.73}$$

where we take the absolute value as the sign of the error is not too interesting. For a set of model predictions $\boldsymbol{z_m}$ and the equivalent set of the true values $\boldsymbol{z_t}$, we might want to reduce the error set,

$$\textbf{Error} = \boldsymbol{z}_t - \boldsymbol{z}_m \ ,$$

into one number to make it easier to compare models. A intuitive solution is to take the mean error, but often it is desirable to emphasize larger differences and for this there are several ways of approach. One approach is to take the mean of the squared error:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{z}_{t,i} - \boldsymbol{z}_{m,i})^2 \ . \tag{2.74}$$

But these error can only be calculated if we have the target solution $\boldsymbol{z_t}$. To compare models without the true solution we can look at the accuracy of our Monte Carlo samples. The ground state energy of a hamiltonian is supposed to have zero total variance in the local energy of the samples. For a discrete set of values $\boldsymbol{X} = \{x_1, x_2, \ldots, x_n\}$ with probabilities $\boldsymbol{P} = \{p_1, p_2, \ldots, p_n\}$, the variance is defined as:

$$Var[\boldsymbol{X}] = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \ , \tag{2.75}$$

where μ is the mean:

$$\mu[\boldsymbol{X}] = \frac{1}{n}\sum_{i=1}^{n} x_i \ . \tag{2.76}$$

The local energy of a sample $|s\rangle \in \boldsymbol{S}$ is:

$$E_L(s) = \frac{\langle s| H |\psi_{rbm}\rangle}{\langle s|\psi_{rbm}\rangle} \ , \tag{2.77}$$

where $|\psi_{rbm}\rangle$ is the machine state. We can then insert the set of all the local energies,

$$\boldsymbol{E}_L = E_L(s_1), E_L(s_2), \ldots, E_L(s_n) \ , \tag{2.78}$$

into the definition of variance, 2.75.

$$Var[\boldsymbol{E}_L] = \frac{1}{n}\sum_{i=1}^{n}(E_L(s_i) - \mu)^2 \ . \tag{2.79}$$

And we get

$$Var[\boldsymbol{E}_L] = \frac{1}{n}\sum_{i=1}^{n}(E_L(s_i) - \langle E_L\rangle)^2 \ , \tag{2.80}$$

which can be rewritten as

$$Var[\boldsymbol{E}_L] = \frac{1}{n}\sum_{i=1}^{n}\left(E_L(s_i)^2 - 2\langle E_L\rangle E_L(s_i) + \langle E_L\rangle^2\right) \ . \tag{2.81}$$

And we get

$$Var[\boldsymbol{E}_L] = \frac{1}{n}\sum_{i=1}^{n}[E_L(s_i)^2] - 2\langle E_L\rangle\frac{1}{n}\sum_{i=1}^{n}[E_L(s_i)] + \langle E_L\rangle^2 \tag{2.82}$$

$$Var[\boldsymbol{E}_L] = \langle E_L^2\rangle - \langle E_L\rangle^2 \tag{2.83}$$

If we take a look at the expectation values $\langle E_L\rangle$ and $\langle E_L^2\rangle$. We estimate the true expectation value of the energy as:

$$\langle E \rangle \approx \langle E_L \rangle = \frac{1}{n} \sum_{k=1}^{n} E_L(s_k) = \frac{1}{n} \sum_{k=1}^{n} \frac{\langle s_k | \, H \, | \psi_{rbm} \rangle}{\langle s_k | \psi_{rbm} \rangle} \, , \tag{2.84}$$

which for a correct trial wavefunction ψ_{rbm} we would have

$$\langle E_L \rangle = \frac{1}{n} \sum_{k=1}^{n} E \frac{\langle s_k | \psi_{rbm} \rangle}{\langle s_k | \psi_{rbm} \rangle} = \langle E \rangle \, . \tag{2.85}$$

And for $\langle E_L^2 \rangle$ we get

$$\langle E_L^2 \rangle = \frac{1}{n} \sum_{k=1}^{n} \frac{\langle s_k | \, H^2 \, | \psi_{rbm} \rangle}{\langle s_k | \psi_{rbm} \rangle} = \frac{1}{n} \sum_{k=1}^{n} E^2 \frac{\langle s_k | \psi_{rbm} \rangle}{\langle s_k | \psi_{rbm} \rangle} = \langle E \rangle^2 \, . \tag{2.86}$$

Inserting this into 2.83 we get

$$Var[\boldsymbol{E}_L] = \langle E \rangle^2 - \langle E \rangle^2 = 0 \, . \tag{2.87}$$

So for a machine state that is close to the true wavefunction we would get a low variance, and if it is perfectly aligned with the true wavefunction we would only need one sample to determine the energy.

Chapter 3

Quantum Mechanics

Quantum mechanics is the physics and mathematics describing how things behave at the smallest of scales. Certainty becomes no more and everything devolves into probabilistic happenstance. The systems we plan to solve are of quantum mechanical nature, so we will start with a small introduction to the field.

3.1 Wavefunction and superposition

A quantum state is described with the help of a wavefunction, which for one dimension we write as

$$\psi(x) : \mathbb{R} \to \mathbb{C} \, .$$

In and of itself the wavefunction does not have a good physical interpretation, but the squared absolute value becomes a probability distribution, such that

$$P(x) = |\psi(x)|^2 \, , \tag{3.1}$$

meaning that $|\psi(x)|^2$ is the probability of finding the particle at position x. The wavefunction therefor needs to be normalized, such that

$$\int_{-\inf}^{\inf} \mathrm{d}x \psi^* \psi = 1 \, . \tag{3.2}$$

It then also means the particle is in a undetermined position before measurement. The quantum state is a combination of all the possible positions it can be in, weighted by $\psi(x)$, in intervals for a continuum of eigenstates $|\phi\rangle$

$$|\psi\rangle = \int \mathrm{d}x \psi(x) \, |\phi\rangle \ . \tag{3.3}$$

And for a discrete set of eigenstates

$$|\psi\rangle = \sum_i \psi(x_i) \, |\phi\rangle \ . \tag{3.4}$$

Such a state $|\psi\rangle$ is called a superposition.

3.2 Operators and the Schrödinger equation

Affecting the wavefunction is done mathematically through operators. An operator maps a state to another, transforming the wavefunction. For a generalized operator $\hat{O}$ and the state $|\psi\rangle$ we have

$$\hat{O}\,|\psi\rangle = |\psi'\rangle \ , \tag{3.5}$$

where $|\psi'\rangle$ is a new state. When an operator acts on a eigenstate:

$$\hat{O}\,|\phi\rangle = \varepsilon\,|\phi\rangle \ , \tag{3.6}$$

the eigenvalue ε is a quantity of what $\hat{O}$ represents. For this quantity to be something physically measurable, the operator needs to be hermitian

$$\hat{O} = \hat{O}^\dagger \ , \tag{3.7}$$

such that $\varepsilon \in \mathbb{R}$. An important observable operator is the hamiltonian, representing the energy of the system. The hamiltonian maps a state to its energy distribution, indicating its time evolution which is dictated by the Schrödinger equation:

$$i\hbar\frac{\partial}{\partial t}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle \ , \tag{3.8}$$

where t is time. Eigenstates of the hamiltonian are the stable energy states of the system

$$H\,|\psi_n\rangle = E_n\,|\psi_n\rangle \ , \tag{3.9}$$

where E_n is the energy of the state.

3.3 Global and local energy

The global energy is the energy of the whole system, the possible are values of energy that can be measured. This is the energy eigenvalues of the hamiltonian:

$$E_n = \frac{\langle\Psi|\,H\,|\Psi\rangle}{\langle\Psi|\Psi\rangle} \ . \tag{3.10}$$

For a state:

$$|\Psi\rangle = \alpha_1\,|\psi_1\rangle + \alpha_2\,|\psi_2\rangle \cdots + \alpha_n\,|\psi_n\rangle \ .$$

For any basis state

$$|\phi\rangle \in \mathbf{B} = \{|\psi_1, |\psi_2\rangle \ldots |\psi_n\rangle\rangle \ ,$$

we have that the local energy is

$$E_{\text{local}}(\phi) = \frac{\langle\phi|\,H\,|\Psi\rangle}{\langle\phi|\Psi\rangle} \ . \tag{3.11}$$

For a Hamiltonian matrix

3.4 Pauli Exclusion Principle

The Pauli Exclusion Principle says that, within a quantum system, identical fermions are limited to one per quantum state. This is expressed mathematically with a anti-symmetric wavefunction:

$$\Psi(\dots, x_i, \dots, x_j, \dots) = -\Psi(\dots, x_j, \dots, x_i, \dots) \,, \tag{3.12}$$

where x_i and x_j is two particles being switched. To show why this is we look at the simplest cast of only two fermions x_1 and x_2 with the respective position $\mathbf{r}_1$ and $\mathbf{r}_2$. We have the two states

$$\psi_{12} = \psi_{x_1}(\mathbf{r}_1)\psi_{x_2}(\mathbf{r}_2)$$
$$\psi_{21} = \psi_{x_1}(\mathbf{r}_2)\psi_{x_2}(\mathbf{r}_1)$$

The wavefunction will be a superposition of these two states and we have the symmetric case:

$$\Psi_s = \sqrt{\frac{1}{2}} \left[\psi_{x_1}(\mathbf{r}_1)\psi_{x_2}(\mathbf{r}_2) + \psi_{x_1}(\mathbf{r}_2)\psi_{x_2}(\mathbf{r}_1)\right]$$

and the anti-symmetric case:

$$\Psi_a = \sqrt{\frac{1}{2}} \left[\psi_{x_1}(\mathbf{r}_1)\psi_{x_2}(\mathbf{r}_2) - \psi_{x_1}(\mathbf{r}_2)\psi_{x_2}(\mathbf{r}_1)\right] \,.$$

To express the Pauli exclusion principle we want to negate the probability of the two fermions being in the same position state, and when we set $\mathbf{r}_1 = \mathbf{r}_2$ we get that:

$$\Psi_s = \sqrt{\frac{1}{2}} \left[\psi_{x_1}(\mathbf{r}_1)\psi_{x_2}(\mathbf{r}_1) + \psi_{x_1}(\mathbf{r}_1)\psi_{x_2}(\mathbf{r}_1)\right]$$

$$\Psi_a = 0 \,,$$

we see that the anti-symmetric wavefunction requirement does exactly that.

3.5 The bra-ket notation

For this thesis the bra-ket notation is used to represent states of the vector form. The bra, $\langle|$, represents a $1 \times N$ matrix:

$$\langle v| = \begin{bmatrix} v_1 & v_2 & \dots & v_N \end{bmatrix} \,.$$

The ket, $|u\rangle$, represents it as a $N \times 1$ matrix:

$$|u\rangle = \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_N \end{bmatrix} \,.$$

With this we have the basic vector-vector multiplication. The inner product is written as:

$$\mathbf{u}^T\mathbf{v} = \langle u|v\rangle = \sum_i u_i v_i \ ,$$

and the outer product is written as:

$$|v\rangle\langle u| = \begin{bmatrix} u_1v_1 & u_1v_2 & \dots & u_1v_N \\ u_2v_1 & u_2v_2 & \dots & u_2v_N \\ \vdots & \vdots & & \vdots \\ u_Nv_1 & u_Nv_2 & \dots & u_Nv_N \end{bmatrix} .$$

We want to use this notation to describe a quantum state $|\psi\rangle$. We start with the standard basis, where a system is in a 0 or 1 state, often called a qubit:

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

This means that the system can only be measured to be in state $|0\rangle$ or $|1\rangle$. Before measurement, however, $|\psi\rangle$ can be a superposition of the computational basis states. Which can be described as a linear combination:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \ ,$$

where both $\alpha, \beta \in \mathbb{C}$ and with the normalization condition $|\alpha| + |\beta| = 1$. It is usefull to visualize $|\psi\rangle$ as a sphere of possible states it can be in, called the Bloch sphere.

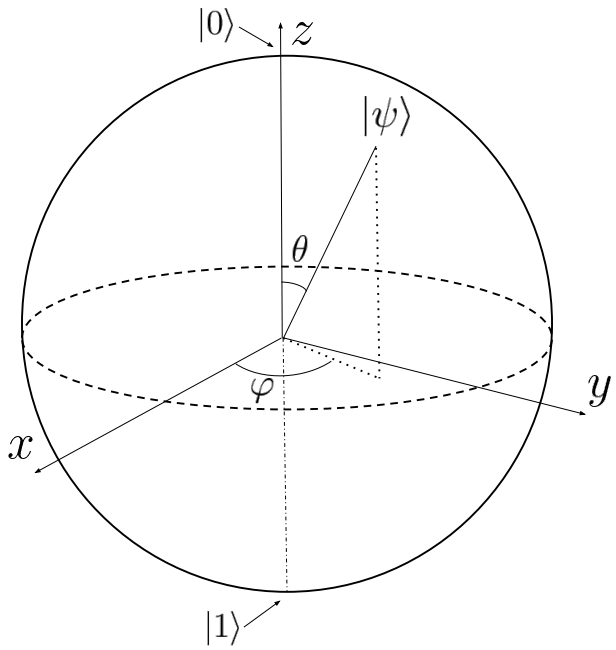


Figure 3.1: The possible states of a system of a single qubit represented by a sphere with the apexes being the computational basis states.

Since we have the restriction $|\alpha| + |\beta| = 1$, the two complex values α and β only contribute to two degrees of freedom, resulting in the surface of the Bloch sphere, 3.1, as the space of possible states $|psi\rangle$.

3.5.1 Multi-Qubit States

Bringing in another qubit will give us more possible measured outcomes. Therefor our computational basis needs to be expanded, encompassing every possible combination. This is done by tensor product, or Kronecker product, of the possible single qubit measurements. For a two-qubit system we then have

$$\begin{aligned}
|0\rangle \otimes |0\rangle &= \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \\
|1\rangle \otimes |0\rangle &= \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \\
|0\rangle \otimes |1\rangle &= \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \\
|1\rangle \otimes |1\rangle &= \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \ ,
\end{aligned} \tag{3.13}$$

which then is a orthonormal basis for our new two-qubit system. To simplify writing, the tensor product between states are often written as

$$\begin{aligned}
|0\rangle \otimes |0\rangle &= |00\rangle \\
|1\rangle \otimes |0\rangle &= |10\rangle \\
|0\rangle \otimes |1\rangle &= |01\rangle \\
|1\rangle \otimes |1\rangle &= |11\rangle \ .
\end{aligned} \tag{3.14}$$

Adding more qubits is simple. For each additional qubit one uses the tensor product of the current basis with the single-qubit basis, resulting in a doubling of possible states. For a N qubit system we then get 2^N possible measured states.

3.6 Measurement in Quantum Mechanics

Measurement in quantum mechanics is different from classical measurement in the way that the act of measuring a state affects the state itself. When a quantum state is measured the output will be a real, classical, value. As such, if we have a wavefunction $|\Psi\rangle$ of a particle in the superposition

$$|\Psi\rangle = \alpha_{\downarrow}\psi_{\downarrow} + \alpha_{\uparrow}\psi_{\uparrow} \,, \tag{3.15}$$

where $\psi_{\downarrow}$ is the spin down state and $\psi_{\uparrow}$ is the spin up state of the particle. A measurement, here indicated with an M, can yield either

$$M(|\Psi\rangle) = \psi_{\downarrow} \text{ or } M(|\Psi\rangle) = \psi_{\uparrow} \,.$$

But as the measurement is done the wavefunction 'collapses' into the measured state

$$M(|\Psi\rangle) = \psi_{\downarrow|\uparrow} \rightarrow |\Psi\rangle = \psi_{\downarrow|\uparrow}$$

Most often we are interested in the original superposition wavefunction and a measurement result does not tell us anything meaningful about it. To get a look at the wavefunction we need to take multiple measurements, where the system is put back into its original state, and estimate $\alpha_{\downarrow}$ and $\alpha_{\uparrow}$ with the averages

$$\alpha_{\downarrow} \approx \sqrt{\frac{N_{\downarrow}}{N}} \tag{3.16}$$

$$\alpha_{\uparrow} \approx \sqrt{\frac{N_{\uparrow}}{N}} \,, \tag{3.17}$$

where N is the total number of measurements and $N_{\downarrow|\uparrow}$ is the number of times the measurement yields the respective state.

Chapter 4

Many-Body Methods

4.1 Basic Principles

4.1.1 Slater Determinants

Given a system with 2 fermions. The first particle particle has quantum numbers i and spin value α, while the second particle has quantum numbers k and spin value β. Since both particles are indistinguishable form each other, we have no way of knowing which particle is where. As a result the wavefunction needs to reflect this by being indifferent to particle permutation. A naive product wavefunction of the two fermions would look like

$$|\psi(x_1, x_2)\rangle = \varphi_{i\alpha}(x_1)\varphi_{k\beta}(x_2) \ . \tag{4.1}$$

If we try to switch the positions of x_1 and x_2 we get that

$$|\psi(x_2, x_1)\rangle = \varphi_{i\alpha}(x_2)\varphi_{k\beta}(x_1) \ , \tag{4.2}$$

which fails the anti-symmetric requirement for fermionic wavefunctions, which makes them distinguishable from each other. Instead we would want a wave function accounts for both scenarios in the first place

$$|\psi(\mathbf{x}_1, \mathbf{x}_2)\rangle = \frac{1}{\sqrt{2}}\left(\varphi_{i\alpha}(x_1)\varphi_{k\beta}(x_2) - \varphi_{i\alpha}(x_2)\varphi_{k\beta}(x_1)\right) \ , \tag{4.3}$$

where the wavefunction is a normalized combination of the two product wavefunctions. Here we have the antisymmetry

$$|\psi(\mathbf{x}_1, \mathbf{x}_2)\rangle = -|\psi(x_2, x_1)\rangle \ . \tag{4.4}$$

Which can be further generalized for a system with N fermions.

$$\psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \frac{1}{\sqrt{N!}}\begin{vmatrix} \varphi_1(\mathbf{x}_1) & \varphi_2(\mathbf{x}_1) & \cdots & \varphi_N(\mathbf{x}_1) \\ \varphi_1(\mathbf{x}_2) & \varphi_2(\mathbf{x}_2) & \cdots & \varphi_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(\mathbf{x}_N) & \varphi_2(\mathbf{x}_N) & \cdots & \varphi_N(\mathbf{x}_N) \end{vmatrix} \ . \tag{4.5}$$

where it gets its name from.

4.2 Second quantization

4.2.1 Creation and annihilation operators

Starting with the vacuum state $|0\rangle$. The creation operator creates a single-particle state α_i:

$$\hat{a}^\dagger_{\alpha_1} |0\rangle = |\alpha_1\rangle \ , \tag{4.6}$$

which can be extended for more particles

$$\hat{a}^\dagger_{\alpha_1} \hat{a}^\dagger_{\alpha_2} \dots \hat{a}^\dagger_{\alpha_n} |0\rangle = |\alpha_1 \alpha_2 \dots \alpha_n\rangle \ . \tag{4.7}$$

The annihilation operator destroys the particle

$$\hat{a}_{\alpha_1} |\alpha_1\rangle = |0\rangle \ . \tag{4.8}$$

They are hermitian conjugate, which means

$$\hat{a}_{\alpha_i} = \left(\hat{a}^\dagger_{\alpha_i}\right)^\dagger \ . \tag{4.9}$$

Furthermore, the Pauli principle says we cannot have two particles in the same state, which gives us

$$\hat{a}^\dagger_{\alpha_i} \hat{a}^\dagger_{\alpha_i} = 0 \ . \tag{4.10}$$

Similarly we cannot annihilate a particle that does not exist in the first place

$$\alpha \neq \{\alpha_i\}$$

$$\hat{a}_\alpha |\alpha_1 \alpha_2 \dots \alpha_n\rangle = 0 \ . \tag{4.11}$$

For fermions the state wavefunction is anti symmetric, therefor switching position of two particles yields a factor of -1, and we get that

$$\hat{a}^\dagger_{\alpha_i} \hat{a}^\dagger_{\alpha_k} = -\hat{a}^\dagger_{\alpha_k} \hat{a}^\dagger_{\alpha_i} \ .$$

From this we have the commutation relations

$$\left\{\hat{a}^\dagger_\alpha \hat{a}^\dagger_\beta\right\} = 0 \tag{4.12}$$

$$\left\{\hat{a}_\alpha \hat{a}_\beta\right\} = 0 \tag{4.13}$$

$$\left\{\hat{a}^\dagger_\alpha \hat{a}_\beta\right\} = \delta_{\alpha\beta} \tag{4.14}$$

4.2.2 Operators in second quantization

For a one-body operator in coordinate space we have

$$\hat{H}_0 = \sum_i \hat{h}_0(x_i) \ , \tag{4.15}$$

and using the anti-symmetric Slater determinant from 4.5, which we can write as:

$$\Phi(x_1, x_2, \ldots, x_n, \alpha_1, \alpha_2, \ldots, \alpha_n) = \frac{1}{\sqrt{n!}} \sum_p (-1)^p \hat{P} \psi_{\alpha_1}(x_1) \psi_{\alpha_2}(x_2) \ldots \psi_{\alpha_n}(x_n) \ , \tag{4.16}$$

we can define

$$\hat{h}_0(x_i)\psi_{\alpha_i}(x_i) = \sum_{\alpha'_k} \psi_{\alpha'_k}(x_i) \langle \alpha'_k | \hat{h}_0 | \alpha_k \rangle \ . \tag{4.17}$$

So for each one-particle function in the Slater determinant we gain a contribution to $\hat{H}_0 |\Phi\rangle$. Our Slater determinant can be written in second quantization as simply

$$|\Phi\rangle = |\alpha_1, \alpha_2, \ldots, \alpha_n\rangle \ . \tag{4.18}$$

With this we have

$$\begin{aligned} \hat{H}_0 |\alpha_1, \alpha_2, \ldots, \alpha_n\rangle = & \sum_{\alpha'_1} \langle \alpha'_1 | \hat{h}_0 | \alpha_1 \rangle | \alpha'_1 \alpha_2 \ldots \alpha_n \rangle \\ & + \sum_{\alpha'_2} \langle \alpha'_2 | \hat{h}_0 | \alpha_2 \rangle | \alpha_1 \alpha'_2 \ldots \alpha_n \rangle \\ & + \ldots \qquad (4.19) \\ & + \sum_{\alpha'_n} \langle \alpha'_n | \hat{h}_0 | \alpha_n \rangle | \alpha_1 \alpha_2 \ldots \alpha'_n \rangle \ , \qquad (4.20) \end{aligned}$$

where we go over each possible permutation of each particle. Furthermore, if we use the fact that we can write

$$|\alpha_1 \alpha_2 \ldots \alpha'_k \ldots \alpha_n\rangle = a^\dagger_{\alpha'_k} a_{\alpha_k} |\alpha_1 \alpha_2 \ldots \alpha_k \ldots \alpha_n\rangle \ , \tag{4.21}$$

we can then shorten the expression to

$$\hat{H}_0 = \sum_{\alpha\beta} \langle \alpha | \hat{h}_0 | \beta \rangle a^\dagger_\alpha a_\beta \ . \tag{4.22}$$

As a generalized one-body operator, that preserves the number of particles, in second quantization. A two-body operator in coordinate space can be written as

$$\hat{H}_I = \sum_{i<j} V(x_i, x_j) \ , \tag{4.23}$$

where V is some interaction force between two particles. Using our Slater determinant again we have that

$$V(x_i, x_j)\psi_{\alpha_k}(x_i)\psi_{\alpha_l}(x_j) = \sum_{\alpha'_k \alpha'_l} \psi'_{\alpha_k}(x_i)\psi'_{\alpha_l}(x_j) \langle \alpha'_k \alpha'_l | \hat{v} | \alpha_k \alpha_l \rangle \tag{4.24}$$

Once again summing over all possible permutations of each combination of two-particle pairs, we get

$$\begin{aligned} H_I|\alpha_1\alpha_2\ldots\alpha_n\rangle = &\sum_{\alpha_1',\alpha_2'}\langle\alpha_1'\alpha_2'|\hat{v}|\alpha_1\alpha_2\rangle|\alpha_1'\alpha_2'\ldots\alpha_n\rangle \\ &+\ldots \\ &+\sum_{\alpha_1',\alpha_n'}\langle\alpha_1'\alpha_n'|\hat{v}|\alpha_1\alpha_n\rangle|\alpha_1'\alpha_2\ldots\alpha_n'\rangle \\ &+\ldots \\ &+\sum_{\alpha_2',\alpha_n'}\langle\alpha_2'\alpha_n'|\hat{v}|\alpha_2\alpha_n\rangle|\alpha_1\alpha_2'\ldots\alpha_n'\rangle \\ &+\ldots \\ &+\sum_{\alpha_{n-1}',\alpha_n'}\langle\alpha_{n-1}'\alpha_n'|\hat{v}|\alpha_{n-1}\alpha_n\rangle|\alpha_1\alpha_{n-1}'\ldots\alpha_n'\rangle\ . \end{aligned} \tag{4.25}$$

Then, using the fact that

$$a^\dagger_{\alpha_k'}a^\dagger_{\alpha_l'}a_{\alpha_l}a_{\alpha_k}|\alpha_1\alpha_2\ldots\alpha_k\ldots\alpha_l\ldots\alpha_n\rangle == |\alpha_1\alpha_2\ldots\alpha_k'\ldots\alpha_l'\ldots\alpha_n\rangle\ , \tag{4.26}$$

we end up with

$$\begin{aligned} H_I|\alpha_1\alpha_2\ldots\alpha_n\rangle &= \sum_{\alpha_1',\alpha_2'}\langle\alpha_1'\alpha_2'|\hat{v}|\alpha_1\alpha_2\rangle a^\dagger_{\alpha_1'}a^\dagger_{\alpha_2'}a_{\alpha_2}a_{\alpha_1}|\alpha_1\alpha_2\ldots\alpha_n\rangle \\ &= {\sum_{\alpha,\beta,\gamma,\delta}}'\langle\alpha\beta|\hat{v}|\gamma\delta\rangle a^\dagger_\alpha a^\dagger_\beta a_\delta a_\gamma|\alpha_1\alpha_2\ldots\alpha_n\rangle\ , \end{aligned} \tag{4.27}$$

where $\alpha,\ \beta$ are single-particle states while $\gamma,\ \delta$ are pairs of single-particle states. We can remove this distinction with the fact that

$$\langle\alpha\beta|\hat{v}|\gamma\delta\rangle = \langle\beta\alpha|\hat{v}|\delta\gamma\rangle\ . \tag{4.28}$$

And we end up with

$$\hat{H}_I = \frac{1}{2}\sum_{\alpha\beta\gamma\delta}\langle\alpha\beta|\hat{v}|\gamma\delta\rangle a^\dagger_\alpha a^\dagger_\beta a_\delta a_\gamma\ , \tag{4.29}$$

where all indices are summed over single-particle states only.

4.3 Methods and Algorithms

4.3.1 Full Configuration Interaction Theory

We want to show that diagonalizing the Hamiltonian matrix yields the energies of the different levels. Starting of with the time-independent Schrodinger equation

$$H\,|\psi\rangle = E\,|\psi\rangle\ . \tag{4.30}$$

Assuming we can express $|\psi\rangle$ in terms of an orthonormal basis $\{|n\rangle\}$:

$$|\psi\rangle = \sum_n c_n |n\rangle \ , \tag{4.31}$$

we can then insert this into the time-independent Schrodinger equation and get

$$\sum_n c_n H |n\rangle = E \sum_n c_n |n\rangle \ . \tag{4.32}$$

Since the basis $\{|n\rangle\}$ is orthonormal we that the identity matrix can be expressed as

$$I = \sum_k |k\rangle \langle k| \ ,$$

where $|k\rangle \in \{|n\rangle\}$. Inserting this into equation 4.32 we get that

$$\sum_k \sum_n \langle k| H |n\rangle c_n |k\rangle = E \sum_n c_n |n\rangle \ . \tag{4.33}$$

Multiplying from the left side by $|m\rangle \in \{|n\rangle\}$ we get

$$\sum_n \langle k| H |n\rangle c_n \langle m|k\rangle = E \sum_n c_n \langle m|n\rangle \ , \tag{4.34}$$

where

$$\langle m|k\rangle = \delta_{mk} \langle m|n\rangle \qquad\qquad = \delta_{mn} \ ,$$

such that

$$\sum_n \langle m| H |n\rangle c_n = E c_m \ . \tag{4.35}$$

$\langle m| H |n\rangle$ is the element m, n of the Hamiltonian matrix, so 4.35 can be expressed as

$$\sum_n H_{mn} c_n = E c_m \ . \tag{4.36}$$

And in matrix form this becomes

$$HC = EC \ , \tag{4.37}$$

where

$$C = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} \ .$$

Diagonalizing H will then give us the eigenvalues E as the energy of the different eigenstates of the system.

4.4 Neural network quantum states

To use a neural network to solve a quantum mechanical system we need a way to represent the quantum state with the neural network. The solution is using neural network quantum states, abbreviated NQS, which was introduced by G. Carlo and M. Troyer in 2017 [4]. A quantum state $|\Psi\rangle$ can be expressed by a neural network as

$$\langle s_1 \dots s_N | \Psi; \boldsymbol{W}, \boldsymbol{B} \rangle = F(s_1 \dots s_N; \boldsymbol{W}, \boldsymbol{B}) \, , \tag{4.38}$$

where we have N input variables, with accordance to the number of degrees of freedom the state $|\Psi\rangle$ has, as well as weights $\boldsymbol{W}$ and biases $\boldsymbol{B}$ of the neural network F.

This means our visual layer in our RBM will be of a basis state of the system. If we imagine a two-bit system we have the basis states:

$$\begin{aligned} |0\rangle \otimes |0\rangle &= |00\rangle \\ |1\rangle \otimes |0\rangle &= |10\rangle \\ |0\rangle \otimes |1\rangle &= |01\rangle \\ |1\rangle \otimes |1\rangle &= |11\rangle \ . \end{aligned} \tag{4.39}$$

Our visual layer will then be a binary matrix of size two where it corresponds to a basis state:

$$\mathbf{v} = \big[1, 0\big] \rightarrow |1, 0\rangle \ .$$

The machines probability distribution over the possible outputs of the visual layer then works as a wavefunction, and generated outputs would be measurements of that wavefunction.

Chapter 5

Models

5.1 The Lipkin-Meshkow-Glick model

One of problem that many-body physics encounter in complex, real world, system is the fact that the many-body Schrödinger equation isn't exactly solvable. This may require either to approximate the Schrödinger equation or limit the number of particles in a system. Therefor it is more common to test many-body methods on simplified models where the exact solution is available without approximation. One of these models is the Lipkin-Meshkow-Glick model, abbreviated LMG or Lipkin in this thesis, first introduced in the 1960's [19].

5.1.1 The model system

The idea behind the LMG model is to have two energy levels separated by an energy-value ε, one just below the Fermi level and one just above. In our case we fill up the base energy level with any number of particles, which then can be excited up to the second energy level. 5.1 shows a visualized example with two particles:

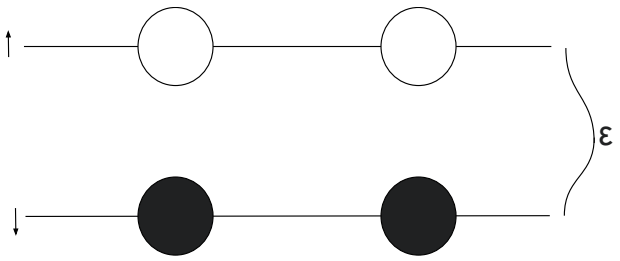


Figure 5.1: The LMG model with two particles and two holes where the particles can move between the two layers. The two levels are separated by a constant ε.

For a N-fermion system the levels are N-fold degenerate, represented by the different positions the particles can be in in 5.1, with two characteristic quantum numbers associated with each particle. The σ quantum number is assumed to be -1 in the lower level and $+1$ in the higher level, which can be seen as particle spin. We will use p to denote the degenerate state in which a particle resides in.

So σ denotes which level the particle is on and p denotes where in that level it resides. The Hamiltonian proposed by Lipkin, Glick and Meshkow is as follows:

$$H = \sum_{p\sigma}\left(\frac{1}{2}\sigma\varepsilon\right)\hat{a}^{\dagger}_{p\sigma}\hat{a}_{p\sigma} + \frac{V}{2}\sum_{pp'\sigma}\hat{a}^{\dagger}_{p\sigma}\hat{a}^{\dagger}_{p'\sigma}\hat{a}_{p'-\sigma}\hat{a}_{p-\sigma} + \frac{W}{2}\sum_{pp'\sigma}\hat{a}^{\dagger}_{p\sigma}\hat{a}^{\dagger}_{p'-\sigma}\hat{a}_{p'\sigma}\hat{a}_{p-\sigma}\ , \tag{5.1}$$

where the operators $\hat{a}^{\dagger}_{p\sigma}$ creates and $\hat{a}_{p\sigma}$ destroys a particle in the energy level associated with σ and in the p position within that level. As an example, if we start out with two particles in the lower level:

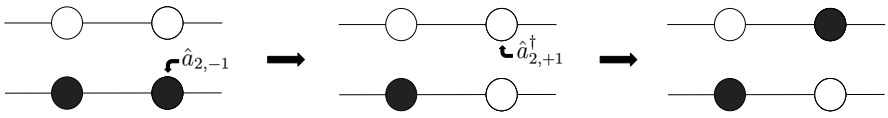


Figure 5.2: Two particles in the $\sigma = -1$ level filling out the $p = 1$ and $p = 2$ state of the LMG model. The particle at $p = 2$ and $\sigma = -1$ is destroyed and then a particle is created at $p = 2$, $\sigma = +1$.

Here we excite one of the particles in the lower layer up to the $\sigma = +1$ layer by destroying and then creating a particle. The Hamiltonian has three parts. Firstly we have the single-particle energy of each particle in the system. Then there is a term that moves pairs of particle from one level to another, with an interaction strength proportional to V. Lastly we have a term that scatters a pair of particles, with an interaction strength proportional to W, exciting one up and de-exciting another.

5.1.2 Rewriting the Hamiltonian

An advantage of the LMG model is the fact that the two-body interaction does not change the value of p. Together with the two-valued σ, each particle can only exist in two possible states. This suggests to use the quasi-spin operators to rewrite the Hamiltonian. These quasi-spin operators are defined as follows:

$$J_{\pm} = \sum_{p} a^{\dagger}_{p\pm}a_{p\mp},\ , \tag{5.2}$$

$$J_z = \frac{1}{2}\sum_{p,\sigma}\sigma a^{\dagger}_{p\sigma}a_{p\sigma},\ , \tag{5.3}$$

$$J^2 = J_+J_- + J_z^2 - J_z,\ , \tag{5.4}$$

where the $+,-$ indicates the quantum number $\sigma = \{+1, -1\}$ as associated with spin up and spin down respectively. The $\hat{J}_+$ accounts for the energy of a particle being excited up a level, the $\hat{J}_-$ accounting for a particle falling down a level. While the $\hat{J}_z$ takes into account the difference in single-particle energy of the two energy levels. All for each degenerate states p. Following the calculations from the lecture notes by M.H Hjensen[20] we show that these obey the commutation relations for angular momentum.

$$
\begin{aligned}
[J_z, J_\pm] &= J_z J_\pm - J_\pm J_z \\
&= \left(\frac{1}{2}\sum_{p,\sigma} \sigma a^\dagger_{p\sigma} a_{p\sigma}\right)\left(\sum_{p'} a^\dagger_{p'\pm} a_{p'\mp}\right) - \left(\sum_{p'} a^\dagger_{p'\pm} a_{p'\mp}\right)\left(\frac{1}{2}\sum_{p,\sigma} \sigma a^\dagger_{p\sigma} a_{p\sigma}\right) \\
&= \frac{1}{2}\sum_{p,p',\sigma} \sigma\left(a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{p'\pm} a_{p'\mp} - a^\dagger_{p'\pm} a_{p'\mp} a^\dagger_{p\sigma} a_{p\sigma}\right).
\end{aligned}
$$

And with the relations

$$\{a_l, a_k\} = 0, \ , \tag{5.5}$$

$$\{a^\dagger_l, a^\dagger_k\} = 0, \ , \tag{5.6}$$

$$\{a^\dagger_l, a_k\} = \delta_{lk}, \ , \tag{5.7}$$

we can move the operators to match in order of the products:

$$
\begin{aligned}
[J_z, J_\pm] &= \frac{1}{2}\sum_{p,p',\sigma} \sigma\left(a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{p'\pm} a_{p'\mp} - a^\dagger_{p'\pm}\left(\delta_{p'p}\delta_{\mp\sigma} - a^\dagger_{p\sigma} a_{p'\mp}\right) a_{p\sigma}\right) \\
&= \frac{1}{2}\sum_{p,p',\sigma} \sigma\left(a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{p'\pm} a_{p'\mp} - a^\dagger_{p'\pm}\delta_{p'p}\delta_{\mp\sigma} a_{p\sigma} + a^\dagger_{p'\pm} a^\dagger_{p\sigma} a_{p'\mp} a_{p\sigma}\right),
\end{aligned}
$$

which gives us

$$
\begin{aligned}
[J_z, J_\pm] &= \frac{1}{2}\sum_{p,p',\sigma} \sigma\left(a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{p'\pm} a_{p'\mp} - a^\dagger_{p'\pm}\delta_{pp'}\delta_{\mp\sigma} a_{p\sigma} + a^\dagger_{p\sigma} a^\dagger_{p'\pm} a_{p\sigma} a_{p'\mp}\right) \\
&= \frac{1}{2}\sum_{p,p',\sigma} \sigma\left(a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{p'\pm} a_{p'\mp} - a^\dagger_{p'\pm}\delta_{pp'}\delta_{\mp\sigma} a_{p\sigma} + a^\dagger_{p\sigma}\left(\delta_{pp'}\delta_{\pm\sigma} - a_{p\sigma} a^\dagger_{p'\pm}\right) a_{p'\mp}\right) \\
&= \frac{1}{2}\sum_{p,p',\sigma} \sigma\left(a^\dagger_{p\sigma}\delta_{pp'}\delta_{\pm\sigma} a_{p'\mp} - a^\dagger_{p'\pm}\delta_{pp'}\delta_{\mp\sigma} a_{p\sigma}\right).
\end{aligned}
$$

And we can shorten it further by comparing it to 5.2.

$$
\begin{aligned}
[J_z, J_\pm] &= \frac{1}{2}\sum_p \left((\pm 1) a^\dagger_{p\pm} a_{p\mp} - (\mp 1) a^\dagger_{p\pm} a_{p\mp}\right) = \pm\frac{1}{2}\sum_p \left(a^\dagger_{p\pm} a_{p\mp} + (\pm 1) a^\dagger_{p\pm} a_{p\mp}\right) \\
&= \pm\sum_p a^\dagger_{p\pm} a_{p\mp} = \pm J_\pm,
\end{aligned}
$$

Further we can use 5.2 to compute the next relation

$$
\begin{aligned}
[J_+, J_-] &= J_+J_- - J_-J_+ \\
&= \left(\sum_p a^\dagger_{p'+}a_{p-}\right)\left(\sum_{p'} a^\dagger_{p'-}a_{p'+}\right) - \left(\sum_{p'} a^\dagger_{p'-}a_{p'+}\right)\left(\sum_p a^\dagger_{p+}a_{p-}\right) \\
&= \sum_{p,p'}\left(a^\dagger_{p'+}a_{p-}a^\dagger_{p'-}a_{p'+} - a^\dagger_{p'-}a_{p'+}a^\dagger_{p+}a_{p-}\right) \\
&= \sum_{p,p'}\left(a^\dagger_{p'+}a_{p-}a^\dagger_{p'-}a_{p'+} - a^\dagger_{p'-}\left(\delta_{++}\delta_{pp'} - a^\dagger_{p+}a_{p'+}\right)a_{p-}\right) \\
&= \sum_{p,p'}\left(a^\dagger_{p'+}a_{p-}a^\dagger_{p'-}a_{p'+} - a^\dagger_{p'-}\delta_{pp'}a_{p-} + a^\dagger_{p'-}a^\dagger_{p+}a_{p'+}a_{p-}\right) \\
&= \sum_{p,p'}\left(a^\dagger_{p'+}a_{p-}a^\dagger_{p'-}a_{p'+} - a^\dagger_{p'-}\delta_{pp'}a_{p-} + a^\dagger_{p+}a^\dagger_{p'-}a_{p-}a_{p'+}\right) \\
&= \sum_{p,p'}\left(a^\dagger_{p'+}a_{p-}a^\dagger_{p'-}a_{p'+} - a^\dagger_{p'-}\delta_{pp'}a_{p-} + a^\dagger_{p+}\left(\delta_{--}\delta_{pp'} - a_{p-}a^\dagger_{p'-}\right)a_{p'+}\right) \\
&= \sum_{p,p'}\left(a^\dagger_{p+}\delta_{pp'}a_{p'+} - a^\dagger_{p'-}\delta_{pp'}a_{p-}\right),
\end{aligned}
$$

which gives us

$$
[J_+, J_-] = \sum_p \left(a^\dagger_{p+}a_{p+} - a^\dagger_{p-}a_{p-}\right) = 2J_z,
$$

and we have that

$$
[J^2, J_\pm] = [J_+J_- + J_z^2 - J_z, J_\pm] = [J_+J_-, J_\pm] + [J_z^2, J_\pm] - [J_z, J_\pm].
$$

Together with the relations

$$
[AB, C] = A[B, C] + [A, C]B, \tag{5.8}
$$

$$
[A, BC] = [A, B]C + B[A, C], \ , \tag{5.9}
$$

we get

$$
[J^2, J_\pm] = J_+[J_-, J_\pm] + [J_+, J_\pm]J_- + J_z[J_z, J_\pm] + [J_z, J_\pm]J_z - [J_z, J_\pm].
$$

This can further be used

$$
\begin{aligned}
[J^2, J_+] &= -2J_+J_z + J_z[J_z, J_+] + [J_z, J_+]J_z - [J_z, J_+] \\
&= -2J_+J_z + J_zJ_+ + J_+J_z - J_+ \\
&= -2J_+J_z + J_+ + J_+J_z + J_+J_z - J_+ = 0,
\end{aligned}
$$

and that

$$
\begin{aligned}
[J^2, J_-] &= 2J_zJ_- + J_z[J_z, J_-] + [J_z, J_-]J_z - [J_z, J_-] \\
&= 2J_zJ_- - J_zJ_- - J_-J_z + J_- \\
&= J_zJ_- - (J_zJ_- + J_-) + J_- = 0.
\end{aligned}
$$

Lastly we have

$$
\begin{aligned}
[J^2, J_z] &= [J_+J_- + J_z^2 - J_z, J_z] \\
&= [J_+J_-, J_z] + [J_z^2, J_z] - [J_z, J_z] \\
&= J_+[J_-, J_z] + [J_+, J_z]J_- \\
&= J_+J_- - J_+J_- = 0 \ ,
\end{aligned}
$$

which completes the set of relations:

$$
[J_z, J_\pm] = \pm J_\pm, \tag{5.10}
$$

$$
[J_+, J_-] = 2J_z, \tag{5.11}
$$

$$
[J^2, J_\pm] = 0, \tag{5.12}
$$

$$
[J^2, J_z] = 0, \ . \tag{5.13}
$$

Together with the number operator:

$$
N = \sum_{p,\sigma} a^\dagger_{p\sigma} a_{p\sigma} . \ , \tag{5.14}
$$

we can go through each term of the Hamiltonian separately. The first part, H_0, is simple to see can be written as:

$$
H_0 = \varepsilon J_z . \ . \tag{5.15}
$$

The H_1 part of the Hamiltonian can be rewritten using the relations 5.5,5.6 and 5.7:

$$
\begin{aligned}
H_1 &= \frac{1}{2} V \sum_{p,p',\sigma} a^\dagger_{p\sigma} a^\dagger_{p'\sigma} a_{p'-\sigma} a_{p-\sigma} \\
&= \frac{1}{2} V \sum_{p,p',\sigma} -a^\dagger_{p\sigma} a^\dagger_{p'\sigma} a_{p-\sigma} a_{p'-\sigma} \\
&= \frac{1}{2} V \sum_{p,p',\sigma} -a^\dagger_{p\sigma} \left(\delta_{pp'} \delta_{\sigma-\sigma} - a_{p-\sigma} a^\dagger_{p'\sigma} \right) a_{p'-\sigma} \\
&= \frac{1}{2} V \sum_{p,p',\sigma} a^\dagger_{p\sigma} a_{p-\sigma} a^\dagger_{p'\sigma} a_{p'-\sigma}
\end{aligned}
$$

And we get

$$
\begin{aligned}
H_1 &= \frac{1}{2}V\sum_{p,p'} a^\dagger_{p+}a_{p-}a^\dagger_{p'+}a_{p'-} + a^\dagger_{p-}a_{p+}a^\dagger_{p'-}a_{p'+} \\
&= \frac{1}{2}V\left[\sum_p \left(a^\dagger_{p+}a_{p-}\right)\sum_{p'}\left(a^\dagger_{p'+}a_{p'-}\right) + \sum_p \left(a^\dagger_{p-}a_{p+}\right)\sum_{p'}\left(a^\dagger_{p'-}a_{p'+}\right)\right] \\
&= \frac{1}{2}V\left[J_+J_+ + J_-J_-\right] = \frac{1}{2}V\left[J_+^2 + J_-^2\right],
\end{aligned}
$$

For the final part, H_2, we have

$$
\begin{aligned}
H_2 &= \frac{1}{2}W\sum_{p,p',\sigma} a^\dagger_{p\sigma}a^\dagger_{p'-\sigma}a_{p'\sigma}a_{p-\sigma} \\
&= \frac{1}{2}W\sum_{p,p',\sigma} -a^\dagger_{p\sigma}a^\dagger_{p'-\sigma}a_{p-\sigma}a_{p'\sigma} \\
&= \frac{1}{2}W\sum_{p,p',\sigma} -a^\dagger_{p\sigma}\left(\delta_{pp'}\delta_{-\sigma-\sigma} - a_{p-\sigma}a^\dagger_{p'-\sigma}\right)a_{p'\sigma} \\
&= \frac{1}{2}W\sum_{p,p',\sigma} -a^\dagger_{p\sigma}\delta_{pp'}a_{p'\sigma} + a^\dagger_{p\sigma}a_{p-\sigma}a^\dagger_{p'-\sigma}a_{p'\sigma} \\
&= \frac{1}{2}W\left(-\sum_{p,\sigma} a^\dagger_{p\sigma}a_{p\sigma} + \sum_{p,p',\sigma} a^\dagger_{p\sigma}a_{p-\sigma}a^\dagger_{p'-\sigma}a_{p'\sigma}\right)
\end{aligned}
$$

and with the number operator it becomes

$$
\begin{aligned}
\sum_{p,p',\sigma} a^\dagger_{p\sigma}a_{p-\sigma}a^\dagger_{p'-\sigma}a_{p'\sigma} &= \sum_{p,p'} a^\dagger_{p+}a_{p-}a^\dagger_{p'-}a_{p'+} + a^\dagger_{p-}a_{p+}a^\dagger_{p'+}a_{p'-} \\
&= \sum_p \left(a^\dagger_{p+}a_{p-}\right)\sum_{p'}\left(a^\dagger_{p'-}a_{p'+}\right) + \sum_p \left(a^\dagger_{p-}a_{p+}\right)\sum_{p'}\left(a^\dagger_{p'+}a_{p'-}\right) \\
&= J_+J_- + J_-J_+,
\end{aligned}
$$

Summing the parts up we have that, using these quasi-spin operators, we can rewrite the Hamiltonian as:

$$
H = \varepsilon\hat{J}_z + \frac{V}{2}\left(\hat{J}_+\hat{J}_+ + \hat{J}_-\hat{J}_-\right) + \frac{W}{2}\left(-N + \hat{J}_+\hat{J}_- + \hat{J}_-\hat{J}_+\right) \tag{5.16}
$$

5.1.3 Analytical Solution

For an exact solution of the LMG model one can use the full configuration interaction theory described in section 4.3.1. However, for a given total spin J the spin of a state can overlap with systems with fewer particles. For example, a system with $J = 2$ and another with $J = 1$ can both have the same spin projection $J_z = -1$ as seen below.

(a) $J = 1$, $J_z = -1$ (b) $J = 2$, $J_z = -1$

Figure 5.3: Two systems with different total spin but both are in a state where $J_z = -1$.

Therefor a more complete Hamiltonian would differentiate between these states. For a four-particle system we would then have

$$H_4 = \begin{bmatrix} H_{J=2} & & 0 \\ & H_{J=1} & \\ 0 & & H_{J=0} \end{bmatrix} . \tag{5.17}$$

Since the Hamiltonian commutes with J^2,

$$\left[H, J^2\right] = 0 \, , \tag{5.18}$$

J is a good quantum number and all other elements in the H_4 Hamiltonian becomes zero. As a diagonal block matrix we can then instead diagonalize the J-specific Hamiltonians separately. For $J = 2$ we construct the hamiltonian matrix with:

$$J_{2,z} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$J_{2,+} = \begin{bmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{6} & 0 & 0 \\ 0 & 0 & 0 & \sqrt{6} & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$J_{2,-} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{6} & 0 & 0 & 0 \\ 0 & 0 & \sqrt{6} & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix} ,$$

and we get

$$H_{J=2} = \begin{bmatrix} -2\varepsilon & 0 & \sqrt{6}V & 0 & 0 \\ 0 & -\varepsilon + 3W & 0 & 3V & 0 \\ \sqrt{6}V & 0 & 4W & 0 & \sqrt{6}V \\ 0 & 3V & 0 & \varepsilon + 3W & 0 \\ 0 & 0 & \sqrt{6}V & 0 & 2\varepsilon \end{bmatrix} , \tag{5.19}$$

If we the diagonalize this matrix we get the following eigenvalues for $\varepsilon = 1$ and $W = 0$.

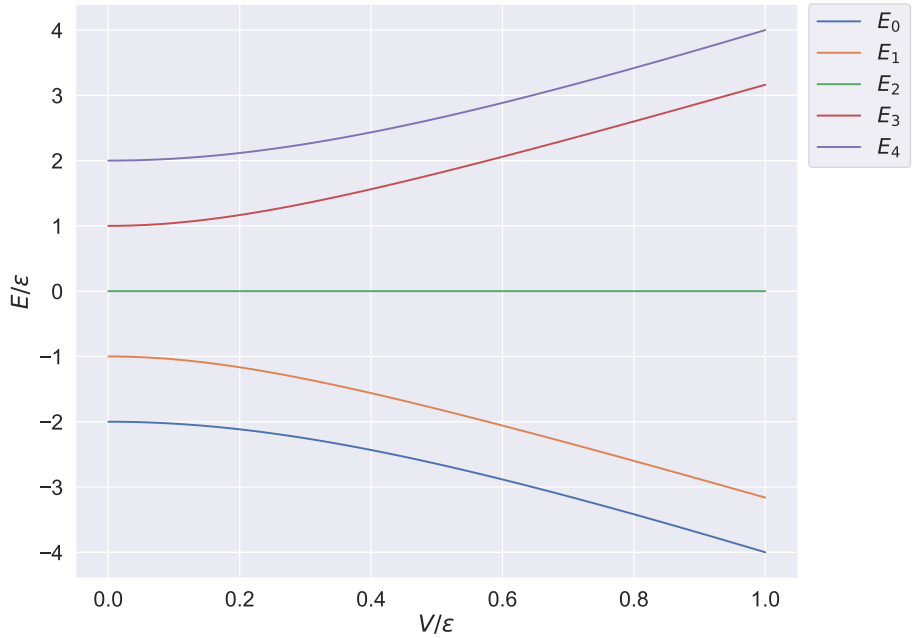


Figure 5.4: The analytical solution for the LMG model with total spin $J = 2$, $\varepsilon = 1$ and $W = 0$.

And for $J = 1$:

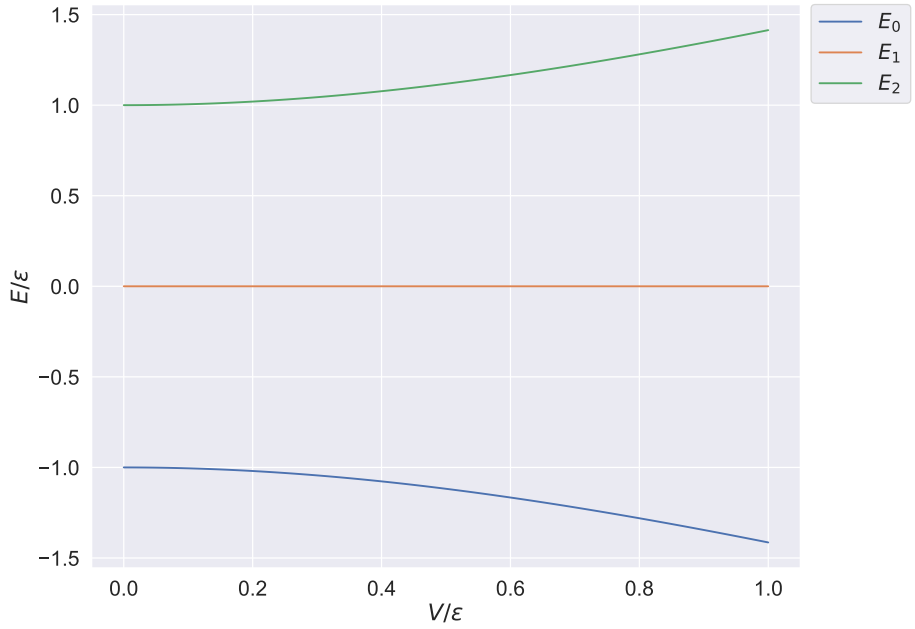


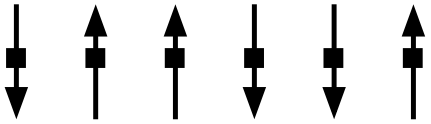
Figure 5.5: The analytical solution for the LMG model with total spin $J = 1$, $\varepsilon = 1$ and $W = 0$.

5.2 The Ising model

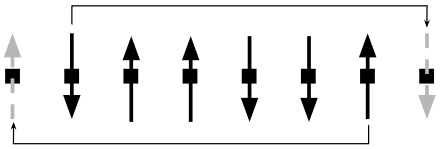
The Ising model consist of a lattice $\boldsymbol{\sigma}$ where each cite $\sigma \in \boldsymbol{\sigma}$, a point on the lattice, can have binary spin value $\sigma \in \{+1, -1\}$. The Hamiltonian function is given by

$$H(\boldsymbol{\sigma}) = J \sum_{\langle i,j \rangle} \sigma_i^z \sigma_j^z + L \sum_j \sigma_j^x \ , \tag{5.20}$$

where i and j are nearest neighbours on the lattice. The constant J is the interaction, or coupling, strength while the constant L is referred to and understood as the strength of a external magnetic field. The Hamiltonian function outputs the energy of a given configuration $\boldsymbol{\sigma}$.
For a one-dimensional Ising model we have a line of particles, here $N = 6$.



and we would have interaction only on in one direction. An obvious problem occurs at the endpoints where the particles are missing a neighbour. There are two main solutions, to either place the boundary conditions $\sigma_0 = \sigma_2$ together with $\sigma_{N+1} = \sigma_{N-1}$ or the condition $\sigma_{N+1} = \sigma_1$ and $\sigma_{-1} = \sigma_N$, where we will use the latter. This results in a continuous-like lattice:



To find the exact solution we want to diagonalize the Hamiltonian matrix, and from the Hamiltonian function we can interpret the spin operators σ_i^x as the as a flip of the particle's spin at lattice point i. In vector form we can represent a lattice point by its spin:

$$\downarrow = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \uparrow = |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ . \tag{5.21}$$

A N-particle configuration we would represented as

$$|\boldsymbol{\sigma}\rangle = |\sigma_1\rangle \otimes |\sigma_2\rangle \otimes \cdots \otimes |\sigma_N\rangle \ .$$

Changes to the configuration, or state, $|\boldsymbol{\sigma}\rangle$ through operators can be understood in matrix form. No change is done by the identity matrix $\mathbb{I}_{2^N}$, constructed by affecting each lattice point by $\mathbb{I}_2$:

$$\mathbb{I}_{2^N} = \mathbb{I}_{2,1} \otimes \mathbb{I}_{2,1} ... \otimes \mathbb{I}_{2,N} \ .$$

A spin flip by the external field part, with the Pauli matrices:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$
$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \ ,$$

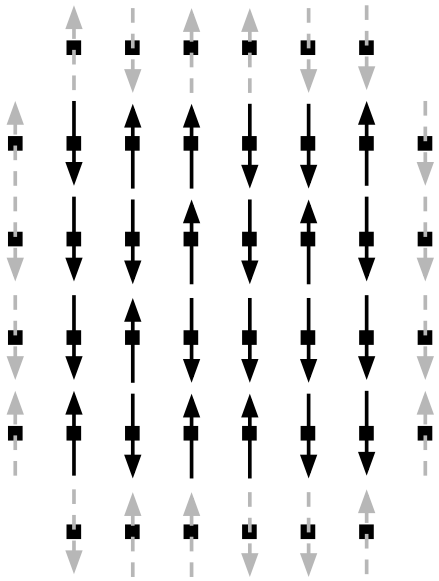
is then done by replacing the $\mathbb{I}_{2,i}$ with the spin matrix σ_x:

$$\sigma_i^x = \mathbb{I}_{2,1} \otimes \cdots \otimes \sigma_{x,i} \otimes \ldots \mathbb{I}_{2,N} \ . \tag{5.22}$$

The coupling interaction is interpreted as

$$\sigma_i^z \sigma_{i+1}^z = \mathbb{I}_{2,1} \otimes \cdots \otimes \sigma_{z,i} \otimes \sigma_{z,i+1} \otimes \cdots \otimes \mathbb{I}_{2,N} \ . \tag{5.23}$$

The Hamiltonian matrix can then be constructed by constructing and adding together each of the addend matrices. Further we have a grid of particles for the two-dimensional Ising model, here a 4×6 grid:



The Hamiltonian function uses $\langle i, j \rangle$ to represent closest neighbouring lattice points, but for two dimensions specifically we can rewrite the Hamiltonian function as

$$H_{2D}(\boldsymbol{\sigma}) = J \sum_{i=1}^{M} \sum_{j=1}^{N} \left(\sigma_{i,j}^z \sigma_{i,j+1}^z + \sigma_{i,j}^z \sigma_{i+1,j}^z \right) + L \sum_{i=1}^{M} \sum_{j=1}^{N} \sigma_{i,j}^z \ . \tag{5.24}$$

where i is the i-th row and j is the j-th particle in that row. The configuration is here represented in our basis by

$$|\boldsymbol{\sigma}\rangle_{2D} = |\sigma_{1,1}\rangle \otimes |\sigma_{1,2}\rangle \otimes \cdots \otimes |\sigma_{1,N}\rangle \otimes |\sigma_{2,1}\rangle \otimes \cdots \otimes |\sigma_{M,N}\rangle$$

Where we can then construct the Hamiltonian matrix the same way as explained for one dimension by following 5.24 together with 5.22 and 5.23.

5.2.1 Solution by diagonalization

If we take the transverse-field one-dimensional Ising model with two particles we have the Hamiltonian matrix constructed by:

$$H = L\left(\sigma_1^x \otimes \mathbb{I} + \mathbb{I} \otimes \sigma_2^x\right) + J\left(\sigma_1^z \otimes \sigma_2^z + \sigma_1^z \otimes \sigma_2^z\right) \ ,$$

and we calculate the tensor products:

$$\sigma^x \otimes \mathbb{I} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbb{I} \otimes \sigma^x = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\sigma^z \otimes \sigma^z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ .$$

We then get

$$H_{J=1} = \begin{bmatrix} J & L & L & 0 \\ L & -J & 0 & L \\ L & 0 & -J & L \\ 0 & L & L & J \end{bmatrix} \ , \tag{5.25}$$

and if with $L = -1$ and $J \in -1, 1$ we diagonalize and plot the eigenvalues:

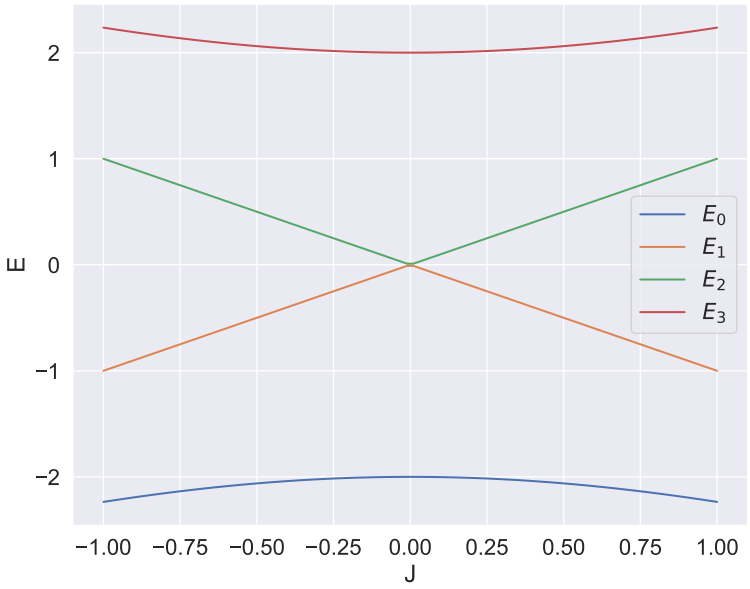


Figure 5.6: The energy derived from diagonalization of the $H_{J=1}$ Ising Hamiltonian with $L = 1$.

For a two-dimensional version we use four particles in a 2×2 grid, and with the same field strength we get:

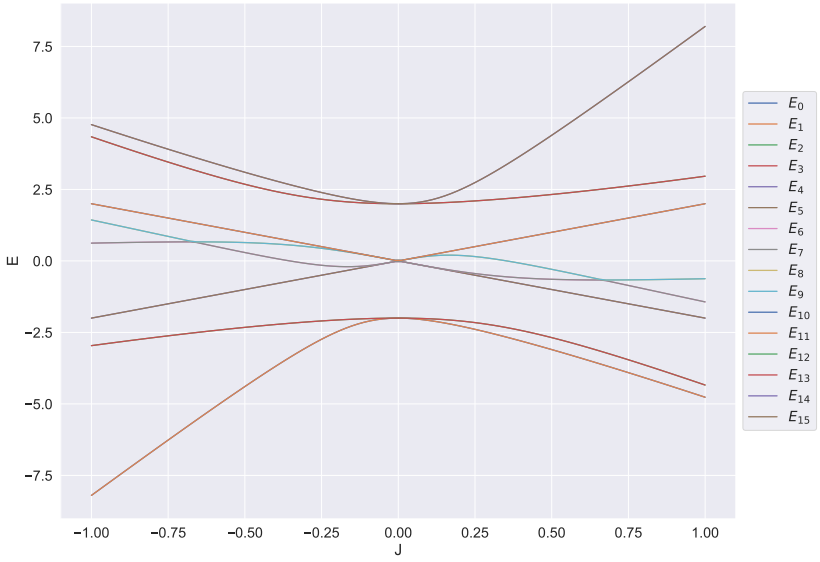


Figure 5.7: The energy derived from diagonalization of the $H_{J=2}$ two-dimensional Ising Hamiltonian with $L = 1$.

5.3 The Heisenberg model

The Heisenberg model is similar to the Ising model. We have lattice points on a line or a grid with a spin value, affected by a external field. In contrast with the Ising model, however, the Heisenberg model introduces coupling in all directions: x, y and z. We will look at the so called XXX model, where the coupling strength is the same in each direction. So we expand our Ising Hamiltonian accordingly:

$$H(\boldsymbol{\sigma}) = J \sum_{\langle i,j \rangle} \left(\sigma_i^x \sigma_j^x + \sigma_i^y \sigma_j^y + \sigma_i^z \sigma_j^z \right) + L \sum_j \sigma_j^z \,, \tag{5.26}$$

The idea behind construction of the Hamiltonian matrix is the same as for Ising, where we only need to extend 5.23 by replacing the Pauli-z spin matrix with Pauli-x and Pauli-y accordingly.

5.3.1 Eigenvalues through diagonalization

Takeing the two-particle one-dimensional heisenberg model we set the external field strength to $L = 1$ and vary the couplng strength from $J = -1$ to $J = 1$. The energy levels then evolve as:

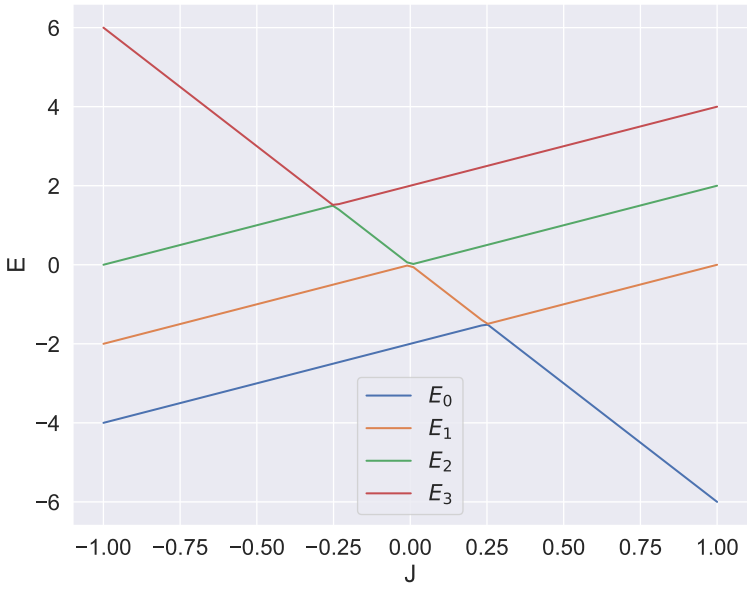


Figure 5.8: The eigenvalues of the two-particle Heisenberg Hamiltonian matrix with $L = 1$.

For a two-dimensional Heisenberg model we up the particle count to four to be able to make a 2×2 lattice. We then get:

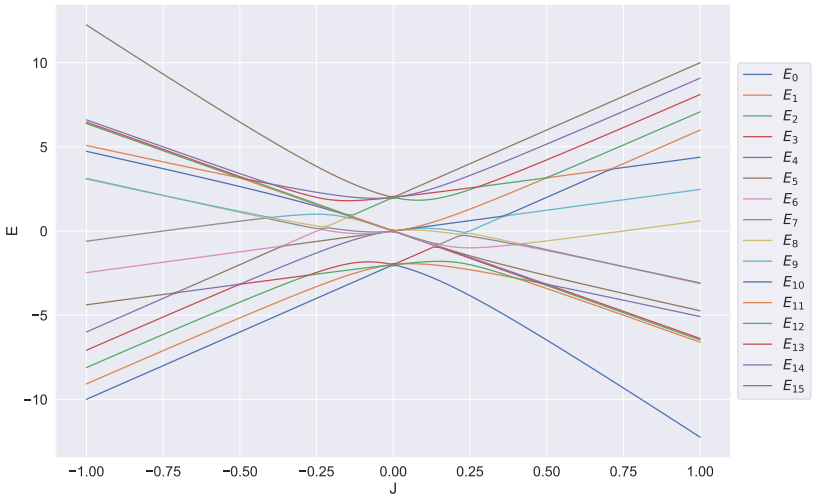


Figure 5.9: The eigenvalues of the four-particle two-dimensional Heisenberg Hamiltonian matrix with $L = 1$.

5.4 The Pairing model

The Pairing model hamiltonian, when limited to up to two-body interactions, is defined as:

$$\hat{H} = \sum_{\alpha\beta} \langle\alpha| h_0 |\beta\rangle \hat{a}^\dagger_\alpha \hat{a}_\beta + \frac{1}{4} \sum_{\alpha\beta\gamma\delta} \langle\alpha\beta| \hat{V} |\gamma\delta\rangle \hat{a}^\dagger_\alpha \hat{a}^\dagger_\beta \hat{a}_\delta \hat{a}_\gamma \tag{5.27}$$

The hamiltonian represents fermions in a
With the assumption that the single particle energies grows linearly with the single particle levels $(p-1)$ and that the two-body interaction strength is constant, we define

$$\begin{aligned} \hat{H}_0 &= \epsilon \sum_{p\sigma} (p-1)\, \hat{a}^\dagger_{p\sigma} \hat{a}_{p\sigma} \\ \hat{V} &= -\frac{1}{2} g \sum_{pq} \hat{a}^\dagger_{p+} \hat{a}^\dagger_{p-} \hat{a}_{q-} \hat{a}_{q+} \ , \end{aligned} \tag{5.28}$$

where ϵ defines the energy gap between levels, and g represents the two-body pairing contribution strength. We will focus on the case of $S = 0$, which means the system contains no broken pairs. If we then define the creation and annihilation operators for a pair

$$\begin{aligned} \hat{P}^+_p &= \hat{a}^\dagger_{p+} \hat{a}^\dagger_{p-} \\ \hat{P}^-_p &= \hat{a}_{p-} \hat{a}_{p+} \ , \end{aligned}$$

we can rewrite the hamiltonian as

$$\hat{H} = \epsilon \sum_{p\sigma} (p-1)\, \hat{a}^{\dagger}_{p\sigma} \hat{a}_{p\sigma} - \frac{1}{2} g \sum_{pq} \hat{P}^{+}_{p} \hat{P}^{-}_{q} \ . \tag{5.29}$$

If we further think of pairs of particles as a particle in and of itself instead, we can rewrite it again:

$$\hat{H} = 2\varepsilon \sum_{p} \hat{a}^{\dagger}_{p} \hat{a}_{p} - \frac{1}{2} g \sum_{pq} \hat{a}^{\dagger}_{p} \hat{a}_{q} \ .$$

With this we can take a look at a state

$$|\psi\rangle = |\alpha_1 \alpha_2 \dots \alpha_P\rangle \ ,$$

where P is the number of included energy levels in our model. In the light of constructing the Hamiltonian matrix we can view the set $\{\alpha_1, \alpha_2, \dots, \alpha_P\}$ representing that a pair of particles is occupying energy level i if $\alpha_i = 1$ and that it is empty if $\alpha_i = 0$. If we then apply the Hamiltonian to the state:

$$\hat{H}\,|\psi\rangle = \hat{H}_0\,|\psi\rangle + \hat{H}_1\,|\psi\rangle \ ,$$

where we first look at the single particle contributions. The single particle energy only comes into account when $\hat{a}^{\dagger}_{p}\hat{a}_{p}$ encounters a pair at a energy level, and then it contributes with a factor of $(p-1)$. In our state we have already encoded the layer count into each α, which means that for the case where $\alpha_i = 1$ we will see a contribution of $(i-1)$. We can then add together each i where $\alpha_i = 1$ and then subtract the number of pairs we have in our model to get the whole single particle contribution.
When it comes to the pair excitation part, H_1, we only see a contribution where a state differs from $|\psi\rangle$ by two quantum numbers α. As an example we will look at the two-layered Pairing model with one pair.

$$\mathbf{H} = \begin{bmatrix} \langle 00|\, H\, |00\rangle & \langle 00|\, H\, |10\rangle & \langle 00|\, H\, |01\rangle & \langle 00|\, H\, |11\rangle \\ \langle 10|\, H\, |00\rangle & \langle 10|\, H\, |10\rangle & \langle 10|\, H\, |01\rangle & \langle 10|\, H\, |11\rangle \\ \langle 01|\, H\, |00\rangle & \langle 01|\, H\, |10\rangle & \langle 01|\, H\, |01\rangle & \langle 01|\, H\, |11\rangle \\ \langle 11|\, H\, |00\rangle & \langle 11|\, H\, |10\rangle & \langle 11|\, H\, |01\rangle & \langle 11|\, H\, |11\rangle \end{bmatrix} \ ,$$

The states $|00\rangle$ and $|11\rangle$ cannot be created through our Hamiltonian 5.29 because we can only move pairs, not create or destroy them, so we can safely say that

$$H\,|00\rangle = H\,|11\rangle = 0$$

We then have only the block of four in the middle, and starting with $\langle 10|\, H\, |10\rangle$. For the single particle part we have:

$$H_0\,|10\rangle = 2\varepsilon \left[(1-1)\hat{a}^{\dagger}_{1}\hat{a}_{1}\,|10\rangle + (2-1)\hat{a}^{\dagger}_{2}\hat{a}_{2}\,|10\rangle \right] = 0 \ ,$$

and

$$H_0\,|01\rangle = 2\varepsilon \left[(1-1)\hat{a}^{\dagger}_{1}\hat{a}_{1}\,|01\rangle + (2-1)\hat{a}^{\dagger}_{2}\hat{a}_{2}\,|01\rangle \right] = 2\varepsilon\,|01\rangle \ ,$$

For the H_1 we have the matrix elements we have the contributions

$$
\begin{aligned}
H_1 |10\rangle &= -\frac{1}{2} g \left(\hat{a}_1^\dagger \hat{a}_2 |10\rangle + \hat{a}_2^\dagger \hat{a}_1 |10\rangle \right) \\
&= -\frac{1}{2} g |01\rangle \ .
\end{aligned}
$$

For $|01\rangle$ we then have

$$
\begin{aligned}
H_1 |01\rangle &= -\frac{1}{2} g \left(\hat{a}_1^\dagger \hat{a}_2 |01\rangle + \hat{a}_2^\dagger \hat{a}_1 |01\rangle \right) \\
&= -\frac{1}{2} g |10\rangle \ .
\end{aligned}
$$

And summarized we have that

$$
\begin{aligned}
\hat{H} |10\rangle &= -\frac{1}{2} g |01\rangle \\
\hat{H} |01\rangle &= 2\varepsilon |01\rangle - \frac{1}{2} g |10\rangle
\end{aligned}
$$

This give us then the matrix elements:

$$
\begin{aligned}
\langle 10| \hat{H} |10\rangle &= 0 \\
\langle 10| \hat{H} |01\rangle &= -\frac{1}{2} g \\
\langle 01| \hat{H} |01\rangle &= 2\varepsilon \\
\langle 01| \hat{H} |10\rangle &= -\frac{1}{2} g
\end{aligned}
$$

and we can get the the whole Hamiltonian matrix:

$$
\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} g & 0 \\ 0 & -\frac{1}{2} g & 2\varepsilon & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} ,
$$

5.4.1 Eigenvalues by diagonalization

Diagonalizing the above matrix with a $\varepsilon = 0.5$ and varying $g = -1 \rightarrow g = 1$ we get:

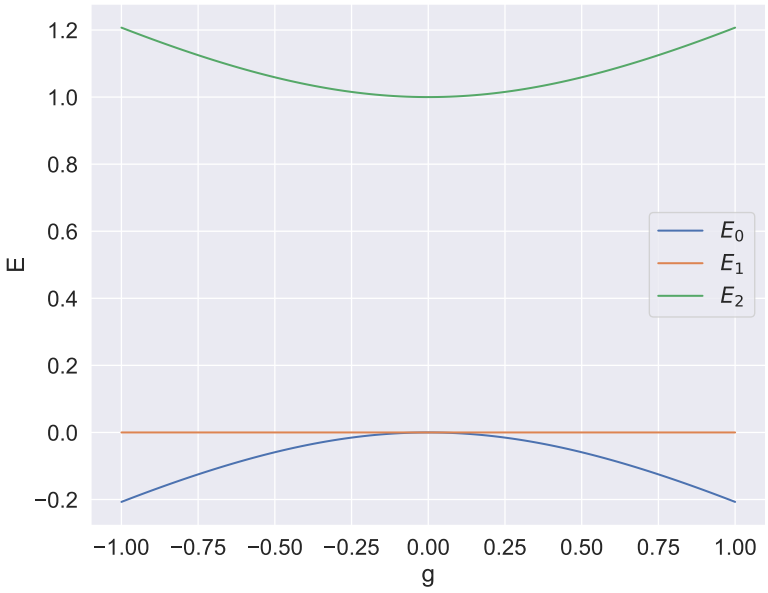


Figure 5.10: The Pairing model with $P = 2$ and $n = 1$ where we have set $\varepsilon = 0.5$. Eigenvalues of the Hamiltnian matrix computed through diagonalization.

For a system where we include three energy levels, continuing with one pair, and changing we get:

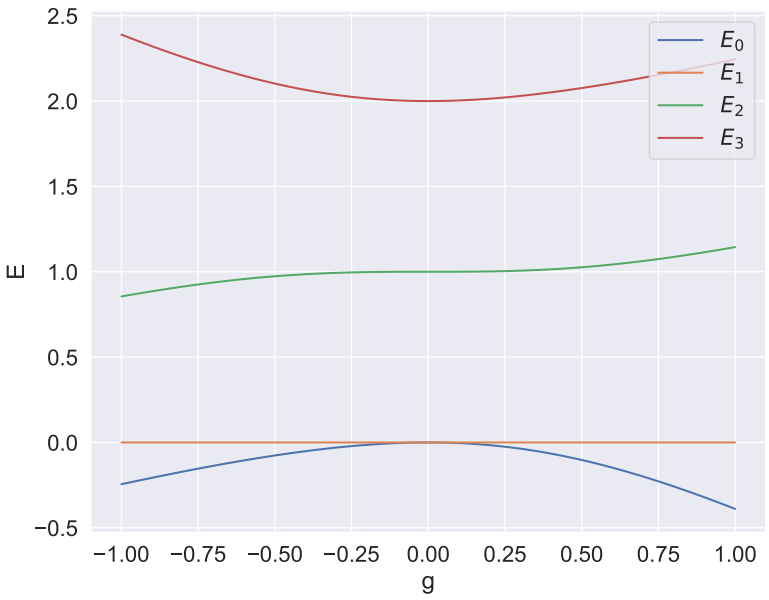


Figure 5.11: The Pairing model with $P = 3$ and $n = 1$ where we have set $\varepsilon = 0.5$. Eigenvalues of the Hamiltnian matrix computed through diagonalization.

Part II

Implementation

Chapter 6

Libraries and hardware

6.1 CUDA and torch

When we have large amounts of data and do the same unconditional operations on them, we can reduce the time it takes for the RBM to find the ground state by parallelizing parts of the calculations. The GPU, graphical processing unit, is designed for such tasks. To use the GPU we need a interface that can communicate with it. There are several options for GPU interfaces, but for NVIDIA[21] GPU's there is the CUDA[22] interface. For simplicity we will use a python library: PyTorch[23]. PyTorch is a machine learning library which has CUDA integration possibilities. To enable the use of the GPU we first initialize the GPU as a device:

```
import torch

device = torch.device('cuda')
```

Then when we create a tensor we do so with the GPU as its device:

```
tensor = torch.tensor(elements, dtype=torch.float64, device=device)
```

where `elements` is the whole object we want on the GPU. Operations on `tensor` will then be run on the processors in the GPU if possible.

Chapter 7

Restricted Boltzmann Machine

7.1 The base structure

We have implemented the restricted Boltzmann machine in separated parts. The parts are connected to each other through the main file, `RBMmodules\\main.py`. The main `run` function takes in the options for that specific run, the model (the quantum system) in question and the machine setup as arguments. These arguments are sent to the different parts of the RBM: the initializer, the local energy function, the adaptive learning rate function and finally the solver, which outputs the predicted ground state energy. In this section we will go over the machine model initialization briefly, while going more in depth into the local energy function and the solver, as these are less straight forward.

7.2 Model initialization

The model initialization is just the how the RBM's structure is desired, but there are some caveats. The RBM really only consists of a few arrays: the visua layer bias, the hidden layer bias and the weights. Defining the size of these is for the most part all that is needed to 'create' an RBM. But, for the use of CUDA through PyTorch we have to define where these arrays are stored, and we would want flexibility as well, so that the code is useable with and without a CUDA graphics card. We do this as such:

```
def set_up_model(visual_n, hidden_n, precision, device):
    visual_bias = torch.zeros(visual_n, dtype=precision, device=device)
    hidden_bias = torch.zeros(hidden_n, dtype=precision, device=device)
    W = W_scale*torch.rand(visual_n, hidden_n, dtype=precision, device=device)

    model = create_model_dataclass(precision, device)

    init_model = model(
        visual_bias = visual_bias,
```

```
        hidden_bias = hidden_bias,
        weights = W,
        device = device,
        precision = precision)

    return init_model
```

The supporting functions have been omitted to save space. The `create_model_dataclass()` is only for the possibility of changing precision with ease, even though everything in the thesis is done in 64-bit floating point precision. At the top of the function is the true creation of the RBM and the rest is for packaging it into one unit together with their data types.

7.3 The solver

Solving a system here means training the RBM some length of time or until a certain threshold is met, and get the machines last guess at the system's ground state energy. To train the machine we take first need to implement the sampling of each layer.

```
def sample_visual(hidden, visual_bias, W):
    given = S(hidden@W.T + visual_bias)
    binary = torch.bernoulli(given)
    return given, binary
```

This is done in accordance to the what has been derived mathematically, where the @ symbol indicates matrix multiplication. The `S(...)` function is the sigmoid function. The given probability distribution of the layer is then sent through a bernoulli function, which returns a binary array based on the given array's distribution. Both versions are returned, so it is only minor changes that is required to go from continuous to binary visual layer. The equivalent `sample_hidden` has only the two bias arrays switched in the `given` definition. The next step is to find the cost function derivative based on the biases and weights. A estimation of the machine state is needed, and this is here done with two different methods. First of we have the Metropolis-Hastings algorithm which accept states conditionally, then secondly we have Gibbs sampling, which accepts all states. The use of a condition makes Metropolis-Hastings unable to be vectorized, and therefor severely slower than Gibbs sampling. Both methods are combined into the same function, here we have left some variable definitions out for ease of reading or lack of relevance.

```
def MonteCarlo(cycles, H, masking_func, gibbs_k, model):
    hidden = torch.bernoulli(torch.rand(
        cycles,
        hidden_n,
        dtype=model.precision,
        device=model.device
    ))
    _, samples = p_visual(hidden)
```

```
    hidden, dist_s = p_sampler(samples)

    dPsidvb = (dist_s - vb)
    dPsidhb = 1/(torch.exp(-hb-dist_s@W)+ 1)
    dPsidW = dist_s[:, :, None]*dPsidhb[:, None, :]
    amplitudes = masking_func(dist_s)
    E_local = hamiltonian.local_energy(H, amplitudes)
    E_mean = torch.mean(E_local)
    E_diff = E_local - E_mean
    DeltaVB = torch.mean(E_diff[:, None]*dPsidvb, axis=0)
    DeltaHB = torch.mean(E_diff[:, None]*dPsidhb, axis=0)
    DeltaW = torch.mean(E_diff[:, None, None]*dPsidW, axis=0)
    dE = torch.mean(E_local - E_mean)
    return DeltaVB, DeltaHB, DeltaW
```

The functions prefixed with a "p" are partial versions, to not needing to pass already defined argument. In the `p_sampler(...)` function we split between either of the two sampling methods. Looking at Metropolis-Hastings first:

```
def metropolis_hastings(input, visual_bias, hidden_bias, W):

    size = input.size(dim=0)
    given_h, hidden = sample_hidden(input, hidden_bias, W)
    rand_nums = torch.rand(size)
    previous = input[0]
    E_prev = net_Energy(previous, visual_bias, hidden, hidden_bias, W)
    for i in range(input.size(dim=0)):
        E_current = net_Energy(input[i], visual_bias, hidden, hidden_bias, W)
        acc_ratio = E_current/E_prev
        if acc_ratio <= rand_nums[i]:
            input[i] = previous
        previous = input[i]
        E_prev = E_current

    return input
```

We have a simple loop that generates a random number and checks it against the acceptance rate by the machine energy ratio. The Gibbs sampling is done by back and forth sampling of the layers.

```
def gibbs_update(input, visual_bias, hidden_bias, W, k):
    given_h, hidden = sample_hidden(input, hidden_bias, W)
    given_v, visual_k = sample_visual(hidden, visual_bias, W)
    for _ in range(k):
        given_h, hidden_k = sample_hidden(given_v, hidden_bias, W)
        given_v, visual_k = sample_visual(given_h, visual_bias, W)

    return hidden_k, visual_k
```

Where the number of cycles `k`, often referred to as `gibbs_k` in the rest of the codebase, is predefined by the user. The two first lines does one Gibbs cycle, which means the total number of cycles is $k + 1$.
Then we have taken to use of all the derived equations for the derivative of the wave function and cost function based on the model variables. Something to note here is the use of automatic casting functionality PyTorch has, done by expanding the correct dimension in the arrays one does arithmetic with. Because of its avid use and the confusing syntax, we will explain a few of the above lines thoroughly.

```
dPsidW = dist_s[:, :, None]*dPsidhb[:, None, :]
```

Both `dist_s` and `dPsidhb` are two-dimensional arrays. Inside the square brackets the `:` sign means that all elements of that dimension is to be included. The `None` creates an empty dimension to the array in the specified depth. For example `dist_s` has `None` in the lowest depth, at the scalar level. This replaces the scalar element with a 1×1 array with the scalar element in it. Like so:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \rightarrow \begin{bmatrix} [a_{11}] & [a_{12}] \\ [a_{21}] & [a_{22}] \end{bmatrix}$$

for a imagined 2×2 `dist_s` with dummy elements. With `dPsidhb[:, None, :]` we have the rows being wrapped up in the same way:

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \rightarrow \begin{bmatrix} [b_{11} & b_{12}] \\ [b_{21} & b_{22}] \end{bmatrix}$$

though as a extra layer above the vector of the row of elements stored in memory. The broadcasting happens when a missing or empty dimension comes up against another missing or empty dimension. PyTorch then makes duplicates of one dimension to fill the missing pieces, and this can be used to get the end result we want. A better example for understanding how broadcasting works is this line here:

```
DeltaVB = torch.mean(E_diff[:, None]*dPsidvb, axis=0)
```

The array `E_diff` is one-dimensional and is stored in memory as a $1 \times n$ matrix, called row major storage for reference, and by adding a wrapper around every element, they themself are stored as a row of only one element. This is then the same as transposing the vector:

$$E_{diff} = [a_1, a_2, \ldots, a_n] \rightarrow \begin{bmatrix} [a_1] \\ [a_2] \\ \vdots \\ [a_n] \end{bmatrix}$$

The transposed array, though now with an extra superficial dimension, is then multiplied element-wise with a one-dimensional array. But since the elements of `E_diff` is now arrays in their own right, `dPsidvb` is copied for each row of `E_diff`. Result in

$$\begin{bmatrix}[a_1]\\[a_2]\\\vdots\\[a_n]\end{bmatrix} * [b_1, b_2, \ldots, b_n] \rightarrow \begin{bmatrix}a_1 \cdot [b_1, b_2, \ldots, b_n]\\a_2 \cdot [b_1, b_2, \ldots, b_n]\\\vdots\\a_n \cdot [b_1, b_2, \ldots, b_n]\end{bmatrix}$$

Where * is element-wise multiplication and $\cdot$ is the scalar product. Going back to the first example we then have

$$\begin{bmatrix}[a_{11}] & [a_{12}]\\[a_{21}] & [a_{22}]\end{bmatrix} * \begin{bmatrix}[b_{11} & b_{12}]\\[b_{21} & b_{22}]\end{bmatrix} \rightarrow \begin{bmatrix}[a_{11}] \cdot [b_{11} & b_{12}] & [a_{12}] \cdot [b_{11} & b_{12}]\\[a_{21}] \cdot [b_{21} & b_{22}] & [a_{22}] \cdot [b_{21} & b_{22}]\end{bmatrix} \rightarrow \begin{bmatrix}a_{11}b_{11} + a_{11}b_{12} & a_{12}b_{11} + a_{12}b_{12}\\a_{21}b_{21} + a_{21}b_{22} & a_{22}b_{21} + a_{22}b_{22}\end{bmatrix}$$

Where it is important to notice that we have refrained from using the = sign as this is all meant as a way to visualize the process as apposed to being mathematically correct.
We then come to the update part of the solver, which has been severly shortened here as there is much of it that is used for logging different run statistics.

```
def find_min_energy(...):

  ...

  for n in range(epochs):
    DeltaVB, DeltaHB, DeltaW = MonteCarlo(...)

    adapt_lr = adapt_func(...)

    model.visual_bias -= adapt_lr*DeltaVB
    model.hidden_bias -= adapt_lr*DeltaHB
    model.weights -= adapt_lr*DeltaW


  ...

return stats
```

For each epoch it calls the `MonteCarlo(...)` function for the derivatives of the cost function based on the different variables we want to optimize. Then there is an adaptive learning rate function, which is defined by the user at the main given˙v, visual˙k = sample˙visual(hidden, visual˙bias, W) **for** ˙ **in** range(k): given˙h, hidden˙k = sample˙hidden(given˙v, hidden˙bias, W) given˙v, visual˙k = sample˙visual(given˙h, visual˙bias, W)
return hidden˙k, visual˙k call, and then we update the machine variables.

7.4 Calculation of the local energy

At the sampling part we skipped over an important line.

```
non_zero[:, None]*non_zero_H
```

Where we have to calculate the local energy of the quantum system in question. We know that the local energy of a particular sample state $|s\rangle$ can be calculated as

$$E_{loc} = \frac{\langle s| H |\psi_{rbm}\rangle}{\langle s|\psi_{rbm}\rangle} \ , \tag{7.1}$$

but here we will explain in more detail how this is done for the each of the different systems we are looking at. First of all it is important to remember the structure of the input to our function calculating the local energy. We want to vectorize the calculations as much as possible so the input will be all m samples, taken with the Gibbs or Metropolis-Hastings algorithm, together in one array:

$$\mathbf{S} = [s_0, s_1, \ldots, s_m] \ , \tag{7.2}$$

where $s_0, s_1, \ldots, s_n \in \mathbf{B}$, where $\mathbf{B}$ is the basis of the system. To calculate the local energy the wave function $|\psi_{rbm}\rangle$:

$$|\Psi_{rbm}\rangle = \alpha_0 |\psi_0\rangle + \alpha_1 |\psi_1\rangle + \cdots + \alpha_n |\psi_n\rangle \ , \tag{7.3}$$

but we don't have the exact coefficients $\alpha_0, \alpha_1, \ldots, \alpha_n$, so we will need to approximate it. When we insert a random state into the hidden layer of the RBM and sample the it we can think of it as a measurement of the quantum state. If we denote $|b\rangle$ as the resulting state of such a measurement, the probability of a measurement resulting in a peculiar basis state is given by:

$$P(|b\rangle = |\psi_i\rangle) = \alpha_i \,|^2 \ . \tag{7.4}$$

And we can thus use our sample set $\mathbf{S}$, which is a collection of many measurements of the machine state, to approximate the coefficients $\{\alpha\}$. If we denote N_i as the number of $|\psi_i\rangle$ we have in our sample set, we can approximate the machine state as:

$$|\Psi_{rbm}\rangle \approx \sqrt{\frac{N_0}{m}} |\psi_0\rangle + \sqrt{\frac{N_1}{m}} |\psi_1\rangle + \cdots + \sqrt{\frac{N_n}{m}} |\psi_n\rangle \ . \tag{7.5}$$

If take a look at the different parts of the local energy, 7.1, we have the normalization factor for a basis state $|b\rangle = |\psi_i\rangle \in \mathbf{B}$.

$$\langle b|\Psi_{rbm}\rangle = \alpha_i \ .$$

To calculate the numerator of E_{local} we start with

$$H |\Psi_{rbm}\rangle = \alpha_0 H |\psi_0\rangle + \alpha_1 H |\psi_1\rangle + \cdots + \alpha_n H |\psi_n\rangle \tag{7.6}$$

The hamiltonian transforms the states into a new one

$$H |\psi_i\rangle = \beta_{i,0} |\psi_0\rangle + \cdots + \beta_{i,n} |\psi_n\rangle \tag{7.7}$$

so we get

$$H\left|\Psi_{rbm}\right\rangle = \alpha_0(\beta_{0,0}\left|\psi_0\right\rangle + \cdots + \beta_{0,n}\left|\psi_n\right\rangle) \tag{7.8}$$
$$+\alpha_1(\beta_{1,0}\left|\psi_0\right\rangle + \cdots + \beta_{1,n}\left|\psi_n\right\rangle) \tag{7.9}$$
$$\vdots \tag{7.10}$$
$$+\alpha_n(\beta_{n,0}\left|\psi_0\right\rangle + \cdots + \beta_{n,n}\left|\psi_n\right\rangle) \tag{7.11}$$

We then have that for the example state $|b\rangle$

$$\langle b|\, H\left|\Psi_{rbm}\right\rangle = \sum_{i=0}^{n} \alpha_j \beta_{i,j}\ , \tag{7.12}$$

where j again is the corresponding state $|b\rangle = |\psi_j\rangle$. The local energy of each sample can then be calculated by this code snippet.

```
def local_energy(H, amplitudes):
    weight, non_zero_mask, mask = amplitudes
    non_zero = weight[non_zero_mask]
    non_zero_H = H[non_zero_mask]

    E = torch.sum(non_zero[:,None]*non_zero_H, dim=0)/weight

    return E[mask]
```

Where we have removed the rows of the Hamiltonian matrix whose corresponding basis has a amplitude of zero in our machine state. The 5th line is where the contributions are summed together, where we use a broadcasting functionality of PyTorch.

```
    E = torch.sum(non_zero[:,None]*non_zero_H, dim=0)/weight
```

If we imagine a basis size of four states we would have a `weight` array in the form of:

$$\texttt{weight} = \left[\sqrt{\tfrac{N_0}{m}} \quad \sqrt{\tfrac{N_1}{m}} \quad \sqrt{\tfrac{N_2}{m}} \quad \sqrt{\tfrac{N_3}{m}}\right] .$$

Assuming that each of the approximated coefficients are greater than zero to make visualization easier, we have that `non_zero` = `weight`. The `[:,None]` is to transpose the `non_zero` array:

$$\texttt{non\_zero[:, None]} = \begin{bmatrix} \sqrt{\frac{N_0}{m}} \\ \sqrt{\frac{N_1}{m}} \\ \sqrt{\frac{N_2}{m}} \\ \sqrt{\frac{N_3}{m}} \end{bmatrix} .$$

The $*$ symbol is for element-wise multiplication. The Hamiltonian matrix is in python a two-dimensional array where it is represented as a array of rows. The multiplication as it stands now doesn't match:

$$\texttt{non\_zero[:, None]*non\_zero\_H} = \begin{bmatrix} \sqrt{\frac{N_0}{m}} \\ \sqrt{\frac{N_1}{m}} \\ \sqrt{\frac{N_2}{m}} \\ \sqrt{\frac{N_3}{m}} \end{bmatrix} * \begin{bmatrix} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & \beta_{0,n} \\ \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & \beta_{1,n} \\ \beta_{2,0} & \beta_{2,1} & \beta_{2,2} & \beta_{2,2} \\ \beta_{3,0} & \beta_{3,1} & \beta_{3,2} & \beta_{3,3} \end{bmatrix} .$$

The `non_zero` is therefore expanded in the direction it the axis of too few elements:

$$\texttt{non\_zero[:, None]*non\_zero\_H} = \begin{bmatrix} \sqrt{\frac{N_0}{m}} \\ \sqrt{\frac{N_1}{m}} \\ \sqrt{\frac{N_2}{m}} \\ \sqrt{\frac{N_3}{m}} \end{bmatrix} \rightarrow \begin{bmatrix} \sqrt{\frac{N_0}{m}} & \sqrt{\frac{N_0}{m}} & \sqrt{\frac{N_0}{m}} & \sqrt{\frac{N_0}{m}} \\ \sqrt{\frac{N_1}{m}} & \sqrt{\frac{N_1}{m}} & \sqrt{\frac{N_1}{m}} & \sqrt{\frac{N_1}{m}} \\ \sqrt{\frac{N_2}{m}} & \sqrt{\frac{N_2}{m}} & \sqrt{\frac{N_2}{m}} & \sqrt{\frac{N_2}{m}} \\ \sqrt{\frac{N_3}{m}} & \sqrt{\frac{N_3}{m}} & \sqrt{\frac{N_3}{m}} & \sqrt{\frac{N_3}{m}} \end{bmatrix} .$$

So we get that

$$\texttt{non\_zero[:, None]*non\_zero\_H} = \begin{bmatrix} \sqrt{\frac{N_0}{m}}\beta_{0,0} & \sqrt{\frac{N_0}{m}}\beta_{0,1} & \sqrt{\frac{N_0}{m}}\beta_{0,2} & \sqrt{\frac{N_0}{m}}\beta_{0,n} \\ \sqrt{\frac{N_1}{m}}\beta_{1,0} & \sqrt{\frac{N_1}{m}}\beta_{1,1} & \sqrt{\frac{N_1}{m}}\beta_{1,2} & \sqrt{\frac{N_1}{m}}\beta_{1,n} \\ \sqrt{\frac{N_2}{m}}\beta_{2,0} & \sqrt{\frac{N_2}{m}}\beta_{2,1} & \sqrt{\frac{N_2}{m}}\beta_{2,2} & \sqrt{\frac{N_2}{m}}\beta_{2,2} \\ \sqrt{\frac{N_3}{m}}\beta_{3,0} & \sqrt{\frac{N_3}{m}}\beta_{3,1} & \sqrt{\frac{N_3}{m}}\beta_{3,2} & \sqrt{\frac{N_3}{m}}\beta_{3,3} \end{bmatrix} .$$

We then sum the rows together and divide by the machine state amplitudes. The calculation is only done on the basis states and then we afterwards extract the corresponding energy for each sample through as mask `E[mask]`. To do this calculation if the first place we need to derive the approximated amplitudes. To do so we first we create an array of basis states.

```
def create_basis(n, dtype, device):
    basis = torch.tensor(
        list(itertools.product([0, 1], repeat=n)),
        dtype = dtype,
        device= device
    )
    return basis
```

where the `itertools.product(...)` creates all combinations of 0 and 1 into a array, and the rest is to make into a PyTorch tensor. With the basis in place we can check it against our samples. Then we check each sample which basis state it is:

```
# From https://netket.readthedocs.io/en/v3.11.4/tutorials/gs-ising.html
def ising_hamiltonian(N, M, J, L):
    import netket.hilbert as hb
```

```
    from netket.operator.spin import sigmax,sigmaz
        hi = hb.Spin(s=1 / 2, N=N*M)
    H = sum([L*sigmax(hi,i) for i in range(N)])
    if M == 1:
        H += sum([J*sigmaz(hi,i)*sigmaz(hi,(i+1)%N) for i in range(N)])
    else:
        for _ in range(M):
            for i in range(N):
                H += J*sigmaz(hi,N*i)*sigmaz(hi,(N*i+1)%N)
                H += J*sigmaz(hi,N*i)*sigmaz(hi, (N*i+N)%(N*M))
    return np.array(H.to_dense())
```

Where we once again use the broadcasting functionality to check each sample against each basis state `samples[:, None] == basis`, and sum the resulting matches. We then divide by the number of samples m and take the square root as dictated by $\sqrt{\frac{N_i}{m}}$ approximation of the amplitudes. We then change the amplitudes to the basis states that isn't represented in the sample set as if it had one anyway, because we need to avoid division by zero in the `local_energy(H, samples)` function. The local energy can then be calculated if we have the Hamiltonian matrix, but the construction of these are unique to each model of course.

7.4.1 The Lipkin Model

The Lipkin model hamiltonian is defined as

$$H = \sum_{p\sigma}\left(\frac{1}{2}\sigma\varepsilon\right)\hat{a}^{\dagger}_{p\sigma}\hat{a}_{p\sigma} + \frac{V}{2}\sum_{pp'\sigma}\hat{a}^{\dagger}_{p\sigma}\hat{a}^{\dagger}_{p'\sigma}\hat{a}_{p'-\sigma}\hat{a}_{p-\sigma} + \frac{W}{2}\sum_{pp'\sigma}\hat{a}^{\dagger}_{p\sigma}\hat{a}^{\dagger}_{p'-\sigma}\hat{a}_{p'\sigma}\hat{a}_{p-\sigma}\ , \tag{7.13}$$

and we have rewritten it in terms of the quasi spin operators:

$$H = \varepsilon\hat{J}_z + \frac{V}{2}\left(\hat{J}_+\hat{J}_+ + \hat{J}_-\hat{J}_-\right) + \frac{W}{2}\left(-N + \hat{J}_+\hat{J}_- + \hat{J}_-\hat{J}_+\right)$$

We use this to construct our Hamiltonian matrix. First we construct the quasi-spin matrices with that for $|S,m\rangle$ $m =\in -S,\ldots,S$:

$$\begin{aligned}
\langle m'|\,\hat{S}_z\,|m\rangle &= \delta_{m',m}m \\
\langle m'|\,\hat{S}_+\,|m\rangle &= \delta_{m',m+1}\sqrt{S(S+1)-m'm} \\
\langle m'|\,S_-\,|m\rangle &= \delta_{m'+1,m}\sqrt{S(S+1)-m'm}
\end{aligned}$$

Done by the following code snippet

```
def construct_spin(S):

  size = int(2*S + 1)
  J_pluss = np.zeros((size, size))
```

```
    J_minus = np.zeros((size, size))
    J_z = np.zeros((size, size))
    J_y = np.zeros((size, size))
    J_x = np.zeros((size, size))

    for i in range(size):
        for k in range(size):
            m = i-S
            n = k-S

            pm_factor = np.sqrt(S*(S+1) -m*n)

            J_pluss[i, k] = dd(m, n+1)*pm_factor
            J_minus[i, k] = dd(m+1, n)*pm_factor
            J_z[i, k] = dd(m,n)*m
            J_x[i, k] = (dd(m, n+1)+dd(m+1,n))*pm_factor
            J_y[i, k] = (dd(m, n+1)-dd(m+1,n))*pm_factor


    return J_z, J_pluss, J_minus, 0.5*J_x, (0.5+0.5j)*J_y
```

We then insert the correct spin to get the size that we need and multiply the matrices together according to the Hamiltonian:

```
def lipkin_hamiltonian(n, eps, V, W):
    J_z, J_pluss, J_minus = construct_spin(n/2)[:3]
    N = np.eye(int(n/2)+1)*n
    H = eps*J_z
    H += V/2*(J_pluss@J_pluss + J_minus@J_minus)
    H += W/2*(-N + J_pluss@J_minus+J_minus@J_pluss)
    return H
```

Where `N = np.eye(int(n/2)+1)*n` is the number operator.

7.4.2 The Ising Model

The background for the construction of the Ising model Hamiltonian matrix has been explained in the theory section 5.2, but we will repeat some of it here. The Hamiltonian is defined as

$$H(\boldsymbol{\sigma}) = J \sum_{\langle i,j \rangle} \sigma_i^z \sigma_j^z + L \sum_j \sigma_j^x \ , \tag{7.14}$$

And we interpret the Pauli matrices as acting on the tensor product space of our system:

$$\sigma_i^x = \mathbb{I}_{2,1} \otimes \cdots \otimes \sigma_{x,i} \otimes \ldots \mathbb{I}_{2,N} \ . \tag{7.15}$$

$$\sigma_i^z \sigma_{i+1}^z = \mathbb{I}_{2,1} \otimes \cdots \otimes \sigma_{z,i} \otimes \sigma_{z,i+1} \otimes \cdots \otimes \mathbb{I}_{2,N} \ . \tag{7.16}$$

For two dimensions we need to think of coupling between rows as well as shown in the rewritten Hamiltonian:

$$H_{2D}(\boldsymbol{\sigma}) = J\sum_{i=1}^{M}\sum_{j=1}^{N}\left(\sigma_{i,j}^{z}\sigma_{i,j+1}^{z} + \sigma_{i,j}^{z}\sigma_{i+1,j}^{z}\right) + L\sum_{i=1}^{M}\sum_{j=1}^{N}\sigma_{i,j}^{z}\,. \tag{7.17}$$

To construct the Hamiltonian we will use a python library called NetKet [9][10][11], which is a library for study of many-body quantum systems with machine learning. We use their construction example from their website [24].

```
# From https://netket.readthedocs.io/en/v3.11.4/tutorials/gs-ising.html
def ising_hamiltonian(N, M, J, L):
    import netket.hilbert as hb
    from netket.operator.spin import sigmax,sigmaz
        hi = hb.Spin(s=1 / 2, N=N*M)
    H = sum([L*sigmax(hi,i) for i in range(N)])
    if M == 1:
        H += sum([J*sigmaz(hi,i)*sigmaz(hi,(i+1)%N) for i in range(N)])
    else:
        for _ in range(M):
            for i in range(N):
                H += J*sigmaz(hi,N*i)*sigmaz(hi,(N*i+1)%N)
                H += J*sigmaz(hi,N*i)*sigmaz(hi, (N*i+N)%(N*M))
    return np.array(H.to_dense())
```

Where we have added the part for the second dimension.

7.4.3 The Heisenberg model

The Heisenberg model's difference to the Ising model is the input, we go from binary spin to continuous, so we reuse the Ising model hamiltonians for both one and two dimensions. To change to continuous spin we use the `given_v` instead of the `binary` from the layer sampling functions described in the section about the solver.

7.4.4 The Pairing model

The Pairing model Hamiltonian is given by

$$\hat{H} = \sum_{\alpha\beta}\langle\alpha|\, h_0\,|\beta\rangle\,\hat{a}_{\alpha}^{\dagger}\hat{a}_{\beta} + \frac{1}{4}\sum_{\alpha\beta\gamma\delta}\langle\alpha\beta|\,\hat{V}\,|\gamma\delta\rangle\,\hat{a}_{\alpha}^{\dagger}\hat{a}_{\beta}^{\dagger}\hat{a}_{\delta}\hat{a}_{\gamma} \tag{7.18}$$

7.5 Implementation tests

As we have implemented the restricted Boltzmann machine and the relevant hamiltonians, it would be good to know for certain that it produces meaningful result for each model. To accomplish this we will test the RBM on some small systems of the different models. This section is only meant to verify our implementation, so here we can sacrifice accuracy for less computation time. For all calculations in this section we iterate through 500 epochs with 10000 samples and a learning rate $\eta = 0.5$.

7.5.1 The Lipkin model

For the Lipkin model we first want to see the machine find the correct energy with no interaction between particles, single particle energy only. We make sure the machine handles increasing complexity by looking at systems with increasing size: 2, 4, 8 and 16 particles. With only single particle energy the ground state is the state where all particles occupying the lower level, which is for a N-particle system:

$$E_0 = \frac{1}{2}\varepsilon N \ . \tag{7.19}$$

To keep the output consistent, and make it easier to analyze, we will set

$$\varepsilon = 1$$

all throughout this section. Combining the convergence graphs into one we get:

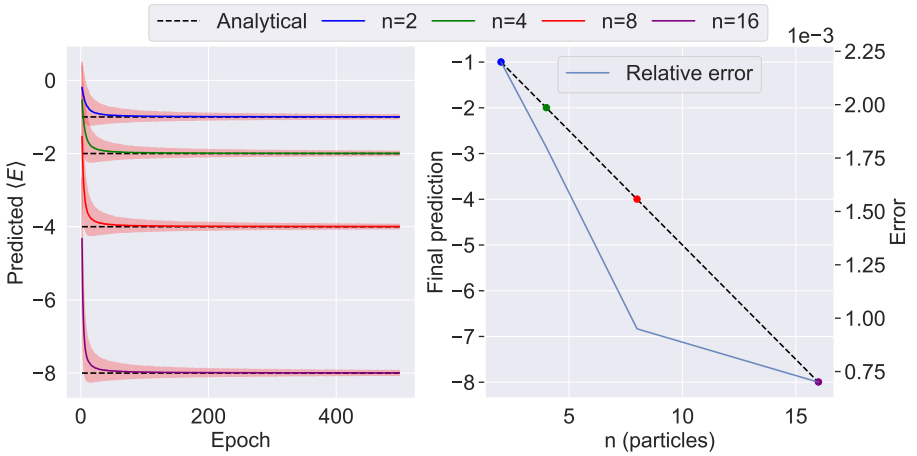


Figure 7.1: The predicted ground state energy for the Lipkin model without interaction and $\varepsilon = 1$. On the left is convergence plotted for each of the different number of particles and the colored area indicates the standard deviation of the local energy of the different samples. The first few data points of each convergence line is removed for readability. On the right the final output of the machine is shown with a dot for n particles 2, 4, 8 and 16 together with the relative error. The dashed lines are the analytical ground state energies.

We see that the RBM manages to find the ground state energy for each system size. We can also see the standard deviation approaching zero. This tells us the RBM wavefunction is close to the true wavefunction because, as explained in 2.4:

$$\begin{gathered} \psi_{rbm} = \psi_{true} \\ \Downarrow \\ Var[E_L] = 0 \ . \end{gathered}$$

The right side plot reiterates that the predicted energies follows the analytical solution, but we also see that the relative error is decreasing, although not with a factor equal to the decrease in ground state energy. Which means the error is increasing somewhat with the size of the system. Looking at the relative standard deviation:

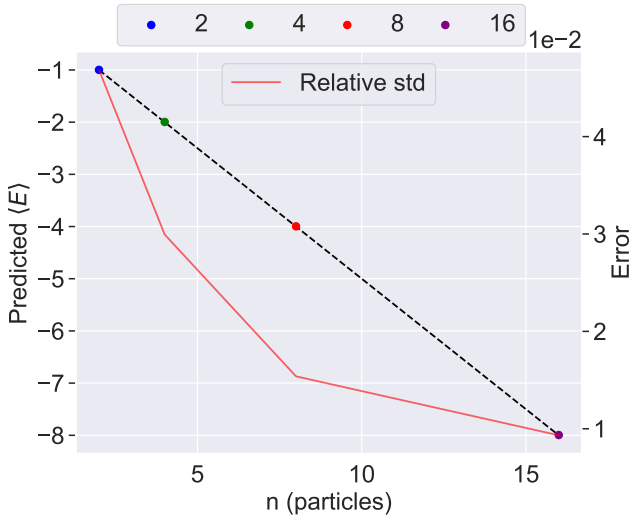


Figure 7.2: The found ground state energy by the restricted Boltzmann machine for the Lipkin model without interaction together with the relative standard deviation error.

we can see that it also decreases with the number of particles. However, the relative standard deviation decreases more, relatively from $n = 2$ to $n = 16$, than the relative error does. This would indicate that the overall increase in error mostly comes from the increase in local energy of the basis states that are wrongly part of the machine state rather than a increase in the machine state's spread.

7.5.2 The Ising model

7.5.3 The Pairing model

7.5.4 The Heisenberg model

Chapter 8

Optimazation of Hyperparameters

8.1 The optimization method

The predefined parameters of a machine has drastic impact on the accuracy of the output. To get the best possible accuracy every possible combination of parameters would be need to be tested and compared, but such an approach becomes unfeasible very quickly. Even with only five parameters and the meager resolution of ten points per parameter, we would have to compute the RBM's prediction 10^5 times. It would take a whole day even if each run is done within a second. To find a useable set of parameters we need to make some decisions of compromise to get within a practical time frame. We will heavily referee to Deep Learning [25], written by Ian Goodfellow, Yoshua Begio and Aaron Courville, who has been in the forefront of the field. In chapter 11 and 12 they cover the best methods of approaching the problem at hand.

For a overview of the approach we have chosen to use, we will give a shorten summary which is explained more in depth in each following section, together with the resulting parameters. To start of we want to find parameters that work best for each quantum system model and for every number of particles. The search time gets quickly out of control with greater system sizes, and as such we want to try to use regression to predict the best parameter for the upper end. As noted by Goodwill et. al. the hyperparameters is the most impactful so we will start by doing a search of optimal learning rate η. The number of hidden nodes is then searched, and for Gibbs sampling we will check the optimal number of sampling cycles.

8.2 The Lipkin model

To determine the accuracy the RBM's output we will use the variance of the local energy, and not the error to the true solution. This may seem counter-intuitive, but by using the true ground state energy we will essentially slightly train the machine on data that it is not supposed to have. We have seen that

for a machine state that matches a eigenstate of the Hamiltonian we get that:

$$Var[E_{local}] = 0 \ ,$$

but this introduces another problem. The variance can approach zero even though it is not of an eigenstate of the Hamiltonian: if the state is dominated by a single basis-state. Therefore we will split off from all figures the parameters which causes the machine to fail to converge at all. Furthermore we will plot two variants of the variance, one of the variance of all the samples' local energy $Var[E(\boldsymbol{S})]$, and second the variance of the local energy of the basis states the machine state consists of, $Var[E(\boldsymbol{B})]$. Both are susceptible to skewed machine states, but on opposite ends. The first will drown out high differences between the basis state local energies by being dominated by a single state, while the other is very susceptible to just one copy of a wrong basis state in the samples. We will therefor choose variables of the quantum system where we expect a distribution of all basis states. But if the machine might still find a state with zero variance by mistake as seen here:

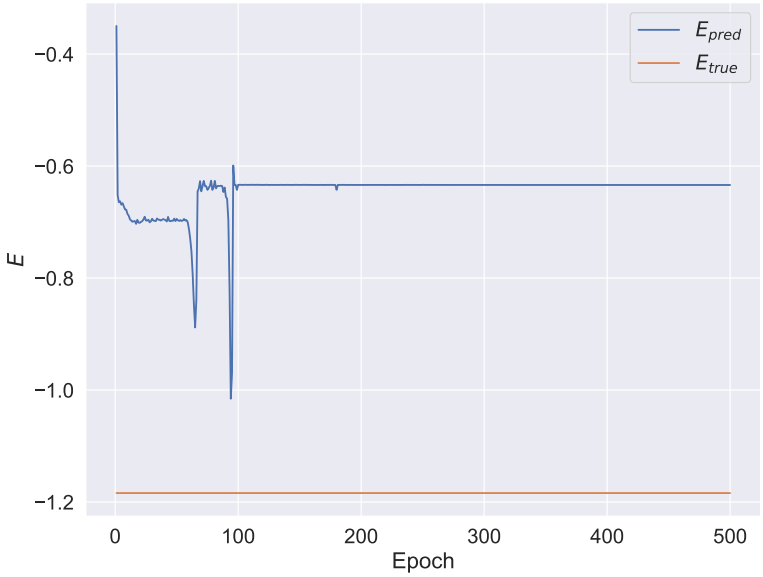


Figure 8.1: The machine in the wrong state, but still achieving close to zero variance of the local energy of each sample. Here the $J = 2$ Lipkin model is used with $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$ and $W = 0$. The number of hidden nodes has here been increased to 20 and the learning rate is at $\eta = 3$.

We will therefore watch the variance of the local energy of the basis states as well, which will increase when the machine state is wrongly dominated by a single basis state. To begin we set up the resulting array first.

```
resolution = 10
```

```
samples_var = np.zeros(resolution)
basis_var = np.zeros(resolution)
```

and we iterate through the two axes:

```
for i in range(resolution):
    for j in range(resolution):

        run_options['adaptive_function'] = adaptives.nop
        run_options['learning_rate'] = search_y[i]

        result = main.run(
            model_options,
            machine_options,
            run_options,
            "",
            log = False,
            verbose = verbose
        )

        samples_var[i] = result['variance'][-1]
        basis_var[i] = result['part_var'][-1]
```

First we attempt to find a suitable range of learning rates to look closer at. Starting of with a wide range of learning rates, $\eta = 1 \rightarrow \eta = 12$, so that we can zoom in on the lower parts afterwards instead. We use 50 data points, but to make sure there are no random outlier we will take the average over five repetitions. We use the Lipkin model with 2 particles and the other parameters and model options are:

- Monte Carlo cycles (number of samples) : 50000
- Epochs : 500
- Number of hidden nodes : 4
- $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$
- $V = -1$
- $W = 0$

We then get:

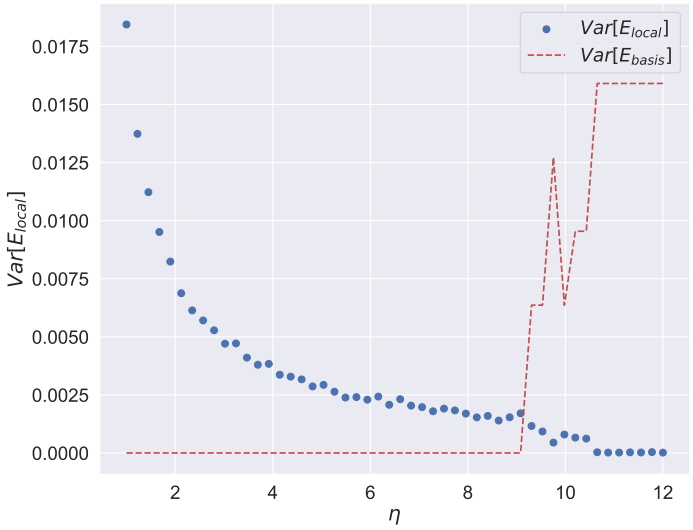


Figure 8.2: First search of optimal learning rate η. The variance of the local energy of each sample together with the variance of the local energy of each basis state. The variance of E_{basis} has been normalized to stay in the same range as E_{local} to make it readable. The Lipkin model is used with $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$ and $W = 0$.

The value of $Var[E_{basis}]$ is not that important, but it is rather the increase and decrease we are interested in, so we have normalized the array to match the same range of values as that of $Var[E_{local}]$. In 8.2 the variance decreases smoothly towards around $\eta = 9$ to $\eta = 10$ where it dips down suddenly. At this point the variance of the basis state's local energy spikes up, indicating that the machine defaulted back to a single basis state wavefunction. We look closer at the range $\eta = 7 \rightarrow \eta = 9$, and we get:

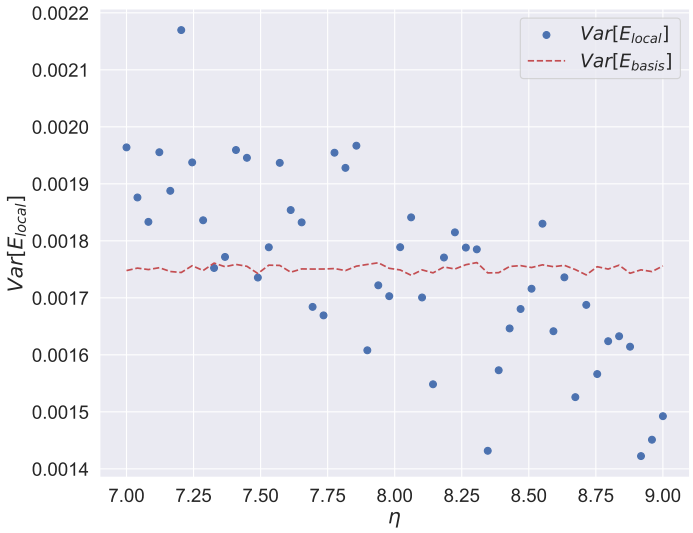


Figure 8.3: The variance of the local energy of each sample together with the variance of the local energy of each basis state. The variance of E_{basis} has been normalized to stay in the same range as E_{local} to make it readable. The Lipkin model is used with $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$ and $W = 0$.

$Var[E_{local}]$ is now a bit more unstable, but it is possible to distinguish a downwards trend towards $\eta = 9$. We will then stop the search here, and since $Var[E_{basis}]$ stays consistent we choose the lowest $Var[E_{local}]$, which is at $\eta = 8.92$. Next step is then to check what number of hidden nodes gives the best accuracy, and we now set the learning rate to the one we just determined to be optimal. For $h_n = 2 \rightarrow h_n = 11$ we get the following:

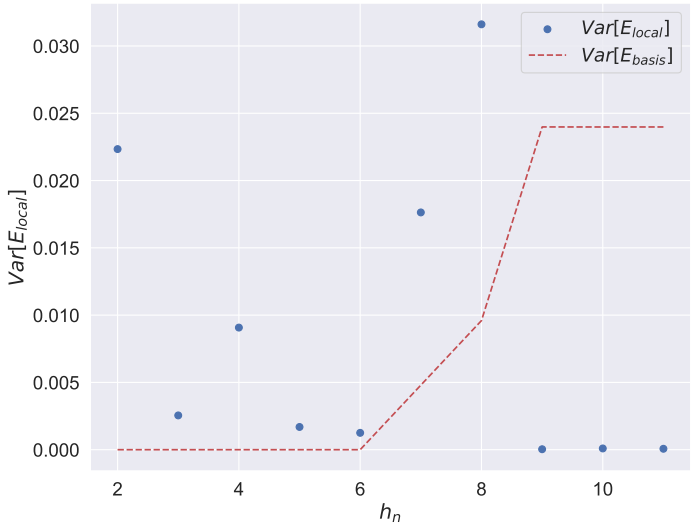


Figure 8.4: Search for optimal number of hidden nodes when $\eta = 8.92$. In the plot is the variance of the local energy of each sample together with the variance of the local energy of each basis state. The variance of E_{basis} has been normalized to stay in the same range as E_{local} to make it readable. The Lipkin model is used with $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$ and $W = 0$.

It is clear that after $h_n = 6$ the machine struggles to find the correct wave-function. For hidden nodes we can't look any closer than integer values, so we choose the lowest variance again, this time at $h_n = 6$. We then do the same thing for the number of Gibbs sampling cycles, with $k = 1 \rightarrow k = 10$,

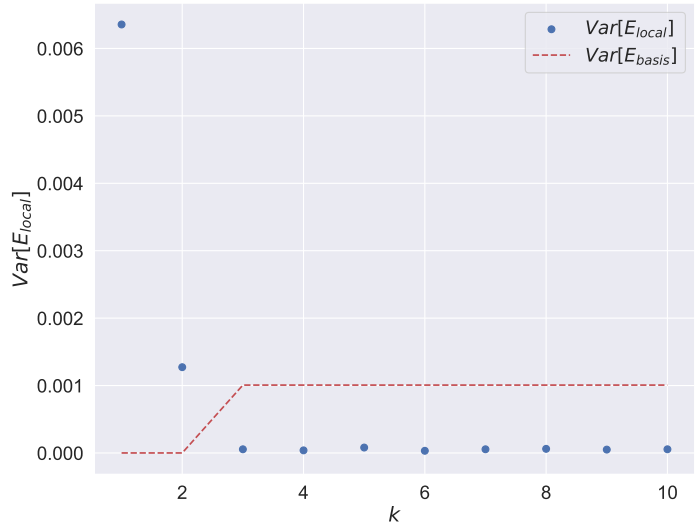


Figure 8.5: Search for optimal number of Gibbs sampling cycles when $\eta = 8.92$. In the plot is the variance of the local energy of each sample together with the variance of the local energy of each basis state. The variance of E_{basis} has been normalized to stay in the same range as E_{local} to make it readable. The Lipkin model is used with $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$ and $W = 0$.

We see a clear dip in $Var[E_{local}]$ from $k = 1$ to $k = 2$, and right after that the $Var[E_{basis}]$ jumps up, so we set $k = 2$ to be optimal for the Lipkin model at 2 particles.

8.2.1 Predicting optimal parameters

For the learning rate η, the number of hidden nodes, and the number of cyces k we want to predict the optimal value as the system size increases. We will use regression and try to fit our data found by hand to a function $f_p(n)$ of parameter p that takes in the size of the system. We will try two different types of ansatz for the learning rate parameter:

Exponential decrease:

$$f_p(n) = ae^{-b*n} + c \tag{8.1}$$

Rational decrease:

$$f_p(n) = \frac{a}{(x+b)} + c \tag{8.2}$$

These two we think is most likely to fit the optimal learning rate data best, but for h_n and k we want to for linear increase:

$$f_p(n) = ax + b \tag{8.3}$$

We use the SciPy[26] python library to fit our data to the curve with the `curve_fit` method:

```
def inverse(x, a, b, c):
    return a/(x+b) + c

def exponential(x, a, b, c):
    return a*np.exp(-b*x) + c

abc = np.array([1, 1, 1])
fitted_inv, pcov_inv    = curve_fit(inverse, x, y, abc)
fitted_exp, pcov_exp    = curve_fit(exponential, x, y, abc)
```

And we choose the one with least variance in the fitted result. With this we get for the Lipkin model that the exponential decrease is the best fit for learning rate:

Figure 8.6: The fitted curve or the learning rate for the Lipkin model ground state energy predictions. The curve type here is exponential decrease, 8.1.

And for γ we get:

Figure 8.7: The fitted curve or the γ momentum parameter for the Lipkin model ground state energy predictions. The curve type here is exponential decrease, 8.1.

We can then summarize the parameters in a table.

8.2.2 Final parameters

Table 8.1: The final parameters chosen for the Lipkin model for different sizes. The colored boxes are derived from regression of the calculated data points.

size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									
size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									

8.3 The Ising model

Table 8.2: The final parameters chosen for the Ising model for different sizes. The colored boxes are derived from regression of the calculated data points.

size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									
size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									

8.4 The Heisenberg model

Table 8.3: The final parameters chosen for the Heisenberg model for different sizes. The colored boxes are derived from regression of the calculated data points.

size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									
size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									

8.5 The Pairing model

Table 8.4: The final parameters chosen for the Pairing model for different sizes. The colored boxes are derived from regression of the calculated data points.

size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									

size	2	3	4	5	6	7	8	9	10
η									
γ									
h_n									

Part III

Results and Discussion

The restricted Boltzmann machine is up and running and the hyperparamters has been optimized for the different system sizes. We will now thoroughly test the machines accuracy over the different system variables. Each system's ground state energy evolves differently, and that may affect the machine's accuracy, as well as the gap between the ground state and first excited state and the distance from the initialized machine state. For each system we will go through each variable in turn and see how the machine accuracy evolves with it. Following we will do a comparison of the computation time between the restricted Boltzmann machine and diagonalization of the Hamiltonian matrix through Numpy[27].

Chapter 9

The Lipkin model

9.1 The Lipkin model

9.1.1 The effect of ε on RBM prediction accuracy

9.1.2 The effect of V on RBM prediction accuracy

9.1.3 The effect of W on RBM prediction accuracy

9.1.4 Comparing computation time with diagonalization

Chapter 10

The Ising model

10.1 The Ising model

10.1.1 The effect of J on RBM prediction accuracy

10.1.2 The effect of L on RBM prediction accuracy

10.1.3 Comparing computation time with diagonalization

Chapter 11

The Heisenberg model

11.1 The Heisenberg model

11.1.1 The effect of J on RBM prediction accuracy

11.1.2 The effect of L on RBM prediction accuracy

11.1.3 Comparing computation time with diagonalization

Chapter 12

The Pairing model

12.1 The Pairing model

12.1.1 The effect of ε on RBM prediction accuracy

12.1.2 The effect of g on RBM prediction accuracy

12.1.3 Comparing computation time with diagonalization

Chapter 13

Comparing Metropolis-Hasting algorithm and the Gibbs sampling method

13.1 Comparing conditional acceptance and the Gibbs sampling method.

The Metropolis-Hastings sampling algorithm uses a conditional acceptance ratio. The Gibbs sampling method, which is what we have mostly used throughout this thesis, removes this condition and accepts every proposed sample state. With Gibbs sampling we can vectorize the process, resulting in a substantial decrease in computation time. Here we would like to see if the acceptance criteria, based on the RBM's internal energy, can improve prediction accuracy by giving a better approximation of the machine state.

For comparison we will use the Lipkin model with $J = 4$ and $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$, and $W = 0$. We first compare the evolution of the methods accuracy as the number of samples, or the number of Monte Carlo cycles, increases. The number of Gibbs cycles are here set to $k = 1$.

Figure 13.1: The error of two RBMs where one uses the Gibbs sampling method, and the other uses a conditional acceptance criteria of the RBM's internal energy. The number of Gibbs sampling cycles is $k = 1$. The quantum system is the Lipkin model with $J = 4$ and $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$, and $W = 0$.

We then see if the number of Gibbs sampling cycles makes a difference:

Figure 13.2: The error of two RBMs where one uses the Gibbs sampling method, and the other uses a conditional acceptance criteria of the RBM's internal energy. The number samples is set to $5 * 10^5$. The quantum system is the Lipkin model with $J = 4$ and $\varepsilon = 1.5 - \frac{\sqrt{3}}{2}$, $V = -1$, and $W = 0$.

Part IV

Conclusion

Chapter 14

Conclusion

We successfully implemented a restricted Boltzmann machine uses the Hamiltonian matrix of a quantum mechanical model to approximate the ground state energy. For the Lipkin model we have seen that the accuracy of our implemented machine ADD. The Ising model has similar effects with ADD. On approximating the ground state energy of the Heisenberg model we see that the accuracy differs with ADD. The Pairing model's ground state energy approximation is better for ADD.
Our implementation has been done with the possibility of changing between the Gibbs sampling method and the Metropolis-Hastings algorithm with an acceptance criteria based in on the machines internal energy. We found that for ADD we saw a increase in accuracy of the ground state energy prediction. The computation time was drastically lowered by the Gibbs sampling method because of its vectorization capabilities. With the Gibbs sampling method our implemented restricted Boltzmann machine's computation time evolved at the rate of ADD for the Lipkin model. For the Ising model we saw a increase in computation time in the form of ADD based on system size. The computation time for the Heisenberg model similarly increased at the rate of ADD. And for the Pairing model we saw that the computation time increased with ADD. Compared to the time taken to diagonalize the Hamiltonian matrix through the Numpy python library, we saw that our restricted Boltzmann machine gaining an advantage at the point of ADD for the Lipkin model, ADD for the Ising model, ADD for the Heisenberg model and ADD for the Pairing model.
For further research it is possible to increase the prediction accuracy through the addition of an adaptive learning rate, as discussed with the comparison with NetKet's accuracy with the one-dimensional Ising model. On a similar note the use of the stochastic reconfiguration method to optimize the hyperparameters would be necessary to compete with cutting edge implementations of the restricted Boltzmann machines. To test ones implementation it would also benefit to look at other quantum models as well, and especially at the higher end of system sizes, which has been limited here by hardware and computation time.

Bibliography

[1] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118307125

[2] G. Ness, A. Vainbaum, C. Shkedrov, Y. Florshaim, and Y. Sagi, "Single-exposure absorption imaging of ultracold atoms using deep learning," *Phys. Rev. Appl.*, vol. 14, p. 014011, Jul 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.14.014011

[3] E. P. L. van Nieuwenburg, Y.-H. Liu, and S. D. Huber, "Learning phase transitions by confusion," *Nature Physics*, vol. 13, no. 5, pp. 435–439, May 2017. [Online]. Available: https://doi.org/10.1038/nphys4037

[4] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," *Science*, vol. 355, no. 6325, pp. 602–606, 2017. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aag2302

[5] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," *Parallel Distributed Process*, vol. 1, pp. 194–281, 01 1986.

[6] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647

[7] R. Xia and S. Kais, "Quantum machine learning for electronic structure calculations," *Nature Communications*, vol. 9, no. 1, p. 4195, Oct 2018. [Online]. Available: https://doi.org/10.1038/s41467-018-06598-z

[8] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, "Fast and accurate modeling of molecular atomization energies with machine learning," *Phys. Rev. Lett.*, vol. 108, p. 058301, Jan 2012. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.108.058301

[9] F. Vicentini, D. Hofmann, A. Szabó, D. Wu, C. Roth, C. Giuliani, G. Pescia, J. Nys, V. Vargas-Calderón, N. Astrakhantsev, and G. Carleo, "NetKet 3: Machine Learning Toolbox for Many-Body Quantum

Systems," *SciPost Phys. Codebases*, p. 7, 2022. [Online]. Available: https://scipost.org/10.21468/SciPostPhysCodeb.7

[10] G. Carleo, K. Choo, D. Hofmann, J. E. Smith, T. Westerhout, F. Alet, E. J. Davis, S. Efthymiou, I. Glasser, S.-H. Lin, M. Mauri, G. Mazzola, C. B. Mendl, E. van Nieuwenburg, O. O'Reilly, H. Théveniaut, G. Torlai, F. Vicentini, and A. Wietek, "Netket: A machine learning toolkit for many-body quantum systems," *SoftwareX*, p. 100311, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352711019300974

[11] D. Häfner and F. Vicentini, "mpi4jax: Zero-copy mpi communication of jax arrays," *Journal of Open Source Software*, vol. 6, no. 65, p. 3419, 2021. [Online]. Available: https://doi.org/10.21105/joss.03419

[12] F. Rosenblatt, *Principles of neurodynamics; perceptrons and the theory of brain mechanisms.* Washington: Spartan Books, 1962, bibliography : p. 609-616. [Online]. Available: http://hdl.handle.net/2027/mdp.39015039846566

[13] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, Jun 1976. [Online]. Available: https://doi.org/10.1007/BF01931367

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[15] G. E. Hinton and T. J. Sejnowski, "Analyzing cooperative computation," *Fifth annual connference of the Cogniitive Science Society*, 1983.

[16] A. Yuille, "Boltzmann machine," *JHU*, 2016.

[17] U. author. (2019) Derivation of gradient of the expectation of local energy. [Online]. Available: https://physics.stackexchange.com/questions/473533/derivation-of-gradient-of-the-expectation-of-local-energy

[18] C. P. Robert and G. Casella, *The Gibbs Sampler.* New York, NY: Springer New York, 1999. [Online]. Available: https://doi.org/10.1007/978-1-4757-3071-5_7

[19] H. Lipkin, N. Meshkov, and A. Glick, "Validity of many-body approximation methods for a solvable model: (i). exact solutions and perturbation theory," *Nuclear Physics*, vol. 62, no. 2, pp. 188–198, 1965. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002955826590862X

[20] M. Hjort-Jensen. (2024) Quantum computing and solving the eigenvalue problem for the lipkin model. [Online]. Available: https://github.com/CompPhysics/QuantumComputingMachineLearning/blob/gh-pages/doc/pub/week8/ipynb/week8.ipynb

[21] NVIDIA, "Nvidia," 2024. [Online]. Available: https://www.nvidia.com/en-us/about-nvidia/#About%20Us

[22] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda compilation tools, release 12.3, v12.3.103," 2023. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[24] F. V. (EPFL-CQSL). (2024) Ground-state: Ising model. [Online]. Available: https://netket.readthedocs.io/en/v3.11.4/tutorials/gs-ising.html

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[26] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. [Online]. Available: https://rdcu.be/b08Wh

[27] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2