UiO : Department of Physics
University of Oslo

Quantum Computing

Machine Learning with Emphasis on Quantum Boltzmann Machines

Philip K. Sørli Niane

Master's Thesis 2022

Abstract

Greetings young adventurer! Welcome to the abstract-section also known as the final boss. To challenge the final boss known as the abstract you need to fulfill the following quest: **Write a thesis!**. To see the details of the quest, please travel down to the next paragraph:

> *Quest: **Write a thesis!** Write a thesis containing a theory-, implementation- and analysis part.*
>
> *Reward: A graduation diploma and a supper in absence of spaghetti and ketchup.*

If **Write a thesis** is complete: Allow the adventurer to challenge the final boss- the abstract!

Acknowledgements

Insert acknowledgements here, 2 paragraphs?

I would like to thank coffee, red bull and the founder of chocolate-Willy Wonka

Abbreviations

VarQBM	Variational Quantum Boltzmann Machine
DNN	Dense Neural Network
OLS	Ordinary Least Squares
RSS	Residual Sum of Squares
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
EMA	Exponential Moving Average
NQS	Neural-network Quantum State
VarITE	Variational imaginary time evolution
MCC	Merchant Category Code

Checklist of things to do while writititing: - Write transaction data instead of fraud data
- Go through the thesis and fix it so only equations that are referred to are labeled with a number
- Include complexity analysis of the circuits either in the appendix or theory, or discusson not sure
- The Franke function or Franke's function??
-Go through citations: –Be consistent –Remove months, only keep year –Only first page, remove pp xx-xx – example (23, Taut) –remove bad articles –Phys. Rev is forkortet, be consisten all over the other citations
-Define the used ansatzes and Hamiltonians in the theory - Add theory on the idea of using a neural network as H coefficients - Add theory on bias and sample vector in varQBM in theory and give the section the following label: sec:bias_dot_product
-Add a list of activation functions?
- Plot the different activation functions used? A plot consisting of all 4 or 4 subfigures, maybe include the derivative also in the plots? -Do I need to include all abbreviations or just the important ones?
- Fix figures to look nice, rbm from https://www.annualreviews.org/doi/pdf/10.1146/annurev-statistics-010814-020120 is quiet nice. Maybe make them myself? https://www.mathcha.io/ is pretty nice.
- Go through the todo's in the code and in the latex document
-Add quantum information theory? afterwords in book - Remove space under and over equations? Be consistent atleast
-Github: Write Readme and make public - Comment code - Remove results

to the appendix, and make a table instead of the found parameters, als replot some of the plots using a log space. - Remove citations using toward datascience and wikipedia and such - An RBM, An NN, an nn- algorithm is the noun not NN when googling -, or . after equations

Todays tasks:
- Expectation values and Hamiltonian measurement - Fill write sections in chapter 4 - Read chapter 4 Monday:
- Go through chapter 5 and write the missing sections - Read through chapter 5 - Write about qiskit - Write about the results of classical data - Write method about classical rbm, last touch of implementation? Tuesday:
- Plot results of classical data - Insert it and discuss everything with the classical data Wednesday:
- Plot classic RBM - Insert results and discuss them - Do some missing coding and fix plots and such

Thursday:
- Go through the list above - Write conclusion, introduction and abstract

Friday: - Fix code, make public and add comments - Write readme file

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

sec:
intro

Quantum computing is getting an increasing recognition all over the world, due to the fact that quantum computers have the possibility of changing the world of mathematical computations within almost all fields of science as we know. Due to the increasing acknowledgment of quantum computers, it is only natural to try to combine the field with one of the most popular fields within classical computing, machine learning. Combining classical- and quantum computing making a hybrid machine learning model sounds like a good idea at first thinking one could achieve the best of both worlds, but can this hybrid approach challenge the spectacular advancements in machine learning that have been constructed with just classical computing?

Write about the varQBM and why it is good compared to a regular BM, divergence problem and such.

Write about history of quantum machine learning and such. Have a look at pp.91 in Schuld

In this article parameterized quantum circuits containing arbitrary amounts of variational parameters will be used to classify the iris dataset[1]. Two different ansatzes will be investigated as the building blocks of the circuits, and the well known optimization method gradient descent will be used to optimize the variational parameters. The article is built up by starting off with some basics of quantum computing including having a look at some quantum gates and explanation of quantum circuits. The article then follows by having a look at parameterized quantum circuits, and how these works. In addition a couple of ansatzes will be proposed. The way to optimize the parameters of the PQCs will then be explained. The datasets will be discussed which is later followed by the details of the imple- mentation. Then comes the analysis part unveiling the results and discussion of them, naturally followed by a conclusion summing up the article.

Machine learning is a field that just gets more and more attention within all fields of science. Naturally machine learning has been used to solve problems within all these fields of science. In recent past machine learning has also been finding its way into the world of quantum mechanical system simulations, by finding ground state energies, particle positions and other quantum mechanical specifications in various systems.

Moved this from the introduction?

1.1 Outline

The thesis is organised as follows:

Chapter 2 revolves around the fundamentals of machine learning providing some examples of machine learning methods, optimization techniques and some ways to assert the machine learning models.

Chapter 3 asserts some important properties of quantum mechanics regarding topics like the basics of quantum mechanics containing notation and evolution of quantum systems. Measurements and entanglements which is an important aspect of quantum computing.

Chapter 4 informs about some important topics of many-body theory which is especially common in connection with quantum computing.

Chapter 5 is honored the part of quantum machine learning. In this chapter some basics of quantum computing is introduced in addition to a review of the variational Quantum Boltzmann machine.

Chapter 6 goes through the methodology of the thesis, explaining how the thesis is implemented.

Chapter 7 Here the results and discussions of the former are presented pairwise

Chapter 9 Rounds up the findings with a conclusion

Appendix A The first appendix

Appendix B The second appendix

PART I

Machine Learning

CHAPTER 2

Principles of Machine Learning

sec:
second

Hot topic ml

There is a lot of publicity and promotion regarding machine learning, nevertheless the recognition is well deserved. People working within multiple fields of sciences like computer science, mathematics, statistics and others, most likely know a bit or a lot of machine learning, but how does machine machine learning actually work and what are the wonders of it?

The main principles of machine learning are quiet straight forward and intuitive. There are multiple forms of machine learning, supervised learning is one of them. The goal of supervised learning is to create models that aim to learn trends from data that can be generalized to predict other unseen data. In search of the perfect generalisation of data, machine learning models need to adjust to training data by optimizing the model, decreasing some loss computed from prediction estimates and true labels. Another form of machine learning is reinforcement learning looking to improve a model based on prior calculations and observations by rewarding and punishing the model based on correct and wrong estimates. For instance one could use reinforcement learning to train an AI agent to master a game like chess or checkers by simulating multiple games, generating skills each game simulated. There is also unsupervised learning which can be recognized as supervised learning without labeled data. Since the data is not labeled, the goal of unsupervised learning is primarily to decide unknown patterns in data. Even though the three presented directions within machine learning are based on different methods, there are multiple common features, for instance they all need to optimize the models in addition to define some loss function to assess the prediction.

In general, the key elements of creating a machine learning model is the dataset if data is needed, labeled or unlabeled. An optimization algorithm used to optimize the machine learning parameters. And of course a chosen machine learning algorithm. In this chapter the key elements of machine learning will be presented along with some machine learning algorithms.

sec: supervised_learning

2.1 Supervised Learning Algorithms

2.1.1 Linear regression

The most common supervised learning method is called linear regression, where the name hints about its purpose: finding linear relationships between input data and target values. A more mathematical representation of linear regression is to express the true value $\mathbf{y}$ as follows:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon \tag{2.1}$$

{eq: linear regression}

Where ϵ is the statistical noise in the dataset, which can be described by random irregularities in the data. Noise in machine learning models are crucial to ensure that the model does not overfit. Which takes us to another important definition in machine learning- overfitting. Overfitting is when a model is fitted so well to training data that it does not longer generalize well on new unseen data, this prevents the model from predicting as good as possible on new unseen data. $\mathbf{X}$ is the design matrix of shape $p \times n$ with p being the number of features and n the number of samples. β is the coefficients which is optimized when training a model. The true value of a single sample can then be expressed by the following expression:

$$y = \beta_0 + x_1\beta_1 + x_2\beta_2 ... + x_p\beta_p + \epsilon$$

In search of the best generalization making the finished model able to be applied on new unseen data predicting the value, one needs to train the model. Training the model is done by optimizing the coefficients $\boldsymbol{\beta} = [\beta_0, \beta_1, .., \beta_p]^T$ where β_0 is the intercept often referred to as the bias. The coefficients is computed as follows by minimizing some loss function $L(\beta)$:

$$\hat{\beta} = \arg\min_{\beta} \ L(\beta), \tag{2.2}$$

Which defines the final prediction of a single sample as:

$$\hat{y} = \hat{\beta}_0 + x_1\hat{\beta}_1 + x_2\hat{\beta}_2 ... + x_p\hat{\beta}_p \tag{2.3}$$

{eq: linear regression 2}

Some loss functions will be presented in section 2.3.

The design matrix

When utilizing different regression methods not only when performing linear regression, but within lots of supervised learning methods, it is important to sort the data into a well designed system for easy access and easier calculations. This is done by sorting the data into a so called design matrix $\mathbf{X}$. A design matrix can be left looking the following way, where each row represent one sample:

$$\mathbf{X} = \begin{pmatrix} 1 & x_0^{(0)} & x_1^{(0)} & \dots & x_p^{(0)} \\ 1 & x_0^{(1)} & x_1^{(1)} & \dots & x_p^{(1)} \\ & & & \vdots & \\ 1 & x_0^{(n)} & x_1^{(n)} & \dots & x_p^{(n)} \end{pmatrix}$$

By having different coefficients often referred to as weights β in front of each term of the design matrix as in Equation 2.3, it is possible to calculate how much each element of the design matrix should affect the computations in order to make the best approximations. Following Equation 2.1 the predictions can be expressed as a matrix the following way:

.

$$\hat{\mathbf{y}} = \begin{pmatrix} \epsilon_0 & \beta_0 & \beta_1 x_1^{(0)} & \beta_2 x_2^{(0)} & \dots & \beta_p x_p^{(0)} \\ \epsilon_1 & \beta_0 & \beta_1 x_1^{(1)} & \beta_2 x_2^{(2)} & \dots & \beta_p x_p^{(1)} \\ & & & & \vdots & \\ \epsilon_n & \beta_0 & \beta_1 x_1^{(n)} & \beta_2 x_2^{(n)} & \dots & \beta_p x_p^{(n)} \end{pmatrix}.$$

sec: logreg

2.1.2 Logistic regression

Logistic regression is a method for classifying a set of input variables $\boldsymbol{x}$ to an output or class $y_i, i = 1, 2, \dots, K$ where K is the number of classes. The review in this section is based on Hastie et al. [1, ch. 4], and the reader is referred to this book for a more detailed explanation of the topic.

When classifying data, one distinguishes between hard and soft classifications, the former places the input variable into a class deterministically while the latter is a probability that a given variable belongs to a certain class. The logistic regression model is given on the form

$$\begin{aligned} \log \frac{p(G = 1|X = x)}{p(C = K|X = x)} &= \beta_{10} + \beta_1^T x \\ \log \frac{p(G = 2|X = x)}{p(C = K|X = x)} &= \beta_{20} + \beta_2^T x \\ &\vdots \\ \log \frac{p(G = K-1|X = x)}{p(C = K|X = x)} &= \beta_{(K-1)0} + \beta_{K-1}^T x \end{aligned}$$

With K possible classes.

Considering a binary, two-class case with $y_i \in [0, 1]$. The probability that a given input variable x_i belongs to class y_i is given by the Sigmoid-function (also called logistic function):

$$p(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$$

A set of predictors $\boldsymbol{\beta}$, which is to be estimated from data then gives the probabilities:

$$\begin{aligned} p(y_i = 1|x_i, \boldsymbol{\beta}) &= \frac{e^{\boldsymbol{\beta}^T x_i}}{1+e^{\boldsymbol{\beta}^T x_i}} = \frac{1}{1+e^{-\boldsymbol{\beta}^T x_i}} \\ p(y_i = 0|x_i, \boldsymbol{\beta}) &= 1 - p(y_i = 1|x_i, \boldsymbol{\beta}) \end{aligned}$$

Firstly defining the set of all possible outputs in a data set $\mathcal{D}(\boldsymbol{x}, \boldsymbol{y})$, and assuming that all samples $\mathcal{D}(x_i, y_i)$ are independent and identically distributed. The total likelihood can be approximated for all possible outputs of $\mathcal{D}$ by the product of the individual probabilities [1, p. 120] of a specific output y_i:

$$P(\mathcal{D}|\beta) = \prod_{i=1}^{n} [p(y_i = 1|x_i \boldsymbol{\beta})]^{y_i} [1 - p(y_i = 1|x_i, \boldsymbol{\beta})]^{1-y_i} \tag{2.4}$$

{eq: likelihood}

sec:knn

2.1.3 Nearest Neighbours

This Nearest Neighbours section is just taken from the special curriculum. Remove or keep section?

Nearest neighbours, most known as k-nearest neighbours is a powerful and intuitive method which are still widely used even though its simplicity. As the name describes, k-nearest neighbours is based on classifying datapoints based on their nearest neighbours. The k in the name is the amount of neighbours that are going to help predict the category of a datapoint.

The method is best understood intuitively when imagining a two dimensional graph with datapoints spread around. Each datapoint belongs to one of two categories also called classes, where the category of each datapoint is known. The method is just as useful having more than two categories, but for simplicity only two categories are considered. By inserting an unknown datapoint which is to be classified within one of the two categories, k-nearest neighbours method is applied. For example by having $k = 5$, the five nearest neighbours will decide the category of the unknown datapoint. The category that most of the five nearest neighbours belongs to, will be the resulting category of the unknown datapoint.

The method can also be used by having all samples contribute to their category. This way the samples receives weights depending on how close the samples are to the new unknown sample by computing the squared distance. The method is often called nearest neighbours when all samples affect the outcome of a single sample used, since then k equals the entire dataset. The weights are given by:

$$\omega_i = 1 - \frac{1}{c}|\hat{x} - x_i|^2 \tag{2.5}$$

{eq:
knnweight}

Where $\hat{x}$ is the unknown sample, x_i is the i'th sample and c is a constant. By computing the weights of all the samples, the probability of classifying the unknown datapoint within category y, which is an arbitrary category label from the dataset, can be computed by the following:

$$p_{\hat{x}}(\hat{y} = y) = \frac{1}{\chi}\frac{1}{M_y}\sum_{i|y_i=y}\omega_i \tag{2.6}$$

{eq:
knnprob}

Where M_y is all the training inputs labeled withing category y and $\frac{1}{\chi}$ is a normalisation constant. The category with the highest probability naturally labels the unknown sample.

KNNimp

Application of nearest neighbours

A demonstration of the nearest neighbours method utilized in a classical case will be shown, in addition to a quantum nearest neighbours method later i the thesis, both applied to the same dataset.

The example is based on the example found in [2, ch. 1]. The dataset that is going to be used are some few samples from the titanic dataset. The dataset can be found at the well known dataset publisher kaggle [3]. The dataset is known as the titanic dataset, which is based on the survival of the passengers on the titanic. Info like age, cabin number, sex, number of siblings and such is published. The goal of the following example is to place the passengers within two categories, survivor or not a survivor.

To focus on the beauty of the algorithm only three samples will be used. Three samples are quiet obviously too few to be reliable in real implementations. Only two properties from the dataset will be used, the price of the cabin and the ticket number. The survival of two of the samples are already known, while the last sample are going to be classified as survied or not survived. The small sample dataset from [2, ch. 1] can be seen in table 2.1.

Table 2.1: Table showing some samples used for creating a small dataset. The dataset is showing the ticket price the travelers paid and the room number before normalisation(BN), and the same information after normalisation(AN) assuming the maximum ticket price and room number was 10000 and 2500 respectively. The survival column shows the survival of the travelers, 1 means survival while 0 means no survival.

Traveler	Price (BN)	Room number (BN)	Price (AN)	Room number (AN)	Survival
1	8500	0910	0.85	0.36	1
2	1200	2105	0.12	0.84	0
3	7800	1121	0.78	0.45	

tab:
dataset_
tit

By plotting the data of the travelers the classification is quiet easy with so a few samples. The plot can be seen in figure 2.1, which easily shows that the unknown traveler will be classified as a survivor.

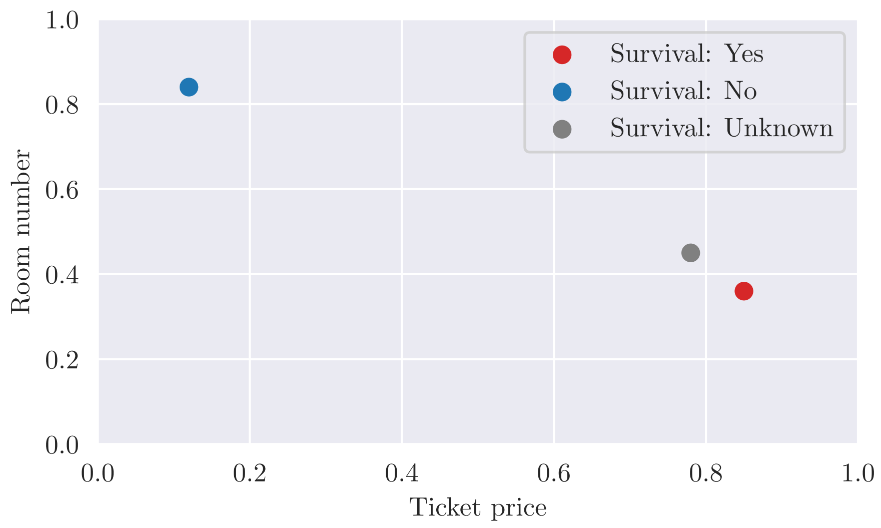


fig: classicknn

Figure 2.1: Plot of the ticket price and room number for the three travelers.

Even though we know the classification, we still want to compute the probability of classifying the unknown sample within the two categories to have a closer look at the comparison with the quantum nearest neighbours later in the thesis. The probability is computed by using equation 2.5 and 2.6. The outcome of the computations results in a probability of classifying the unknown sample as a survivor to be 0.522, while the probability of classifying the same sample as a traveler that did not survive is 0.478.

2.1.4 Dense neural networks

Neural networks is a machine learning method which is roughly made up of the perception of how the real brain works. Neural networks are created by having an input layer, output layer and one or more hidden layers between. Each layer consists of one or more nodes, often referred to as neurons. In a dense neural network, each of the nodes are connected to each node in the layer in front and behind. A visual explanation of a neural network can be seen in Figure 2.2.

Roughly explained each node in the neural network has a corresponding weight which is tunable just like the coefficients β in the earlier sections. The goal of a neural network is to adjust these weights during training of the network to make a certain prediction based on the input data.

Feed forward neural networks which was one of the first and most simplistic types of artificial neural networks [5] is defined by having the information in the neural network only go in one direction without creating cycles of information-

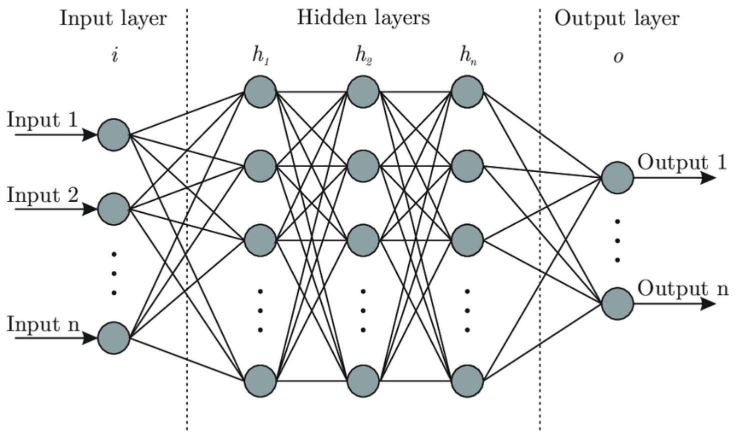


Figure 2.2: Schematic diagram of a dense neural network consisting of up to n hidden layers and nodes. Note that all layers are fully connected. Figure from [4]

fig: nnimg

flow within the network, therefore the information either goes forward during prediction or backward during weight updates.

The feed forward part of the network is done by inserting input values in the first layer. The values of the neurons in the layer in front, also called activation's are computed, then the neurons in each layer are computed all the way to the activaton(s) in the output layer which serves as the final prediction of the network. It is most common to wrap a non linear activation function around each node in addition to adding a potential bias to the node. By using a non linear activation function around the nodes, the network layers have the possibility to go from linear layers to non linear layers. The activation of layer l are left looking as follows:

$$\boldsymbol{a}^l = f^l(W^l\boldsymbol{a}^{l-1} + \boldsymbol{b}^l) \tag{2.7}$$

{eq: activation}

Where W is the weight matrix, b is the bias and $f(x)$ is the activation function used, making the neural network non-linear. There are several popular activation functions, with sigmoid-, ReLU- and tanh function being some of the most common ones [6]. The activation function is often selected based on the issue the neural network are going to overcome, in addition it is important to keep the derivative of the activation functions in mind, due to the possibility of having exploding and vanishing gradients. An overview of some popular activation functions can be seen in Table 2.2

By adding more layers and making the neural network deeper, the computational complexity naturally also increases. Due to the increase in computational complexity, there is a lot of variations within neural networks to ease the computations without loosing too much of the power of dense neural networks, e.g. convolutional neural networks, recurrent neural networks and many more.

Table 2.2: A variety of activation functions, commonly used in neural networks

Name	Activation Function
Identity	$f(x) = x$
Sigmoid	$f(x) = \frac{1}{(1+e^{-x})}$
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Leaky Relu	$f(x) = \max(0.01x, x)$

tab: activaton_ functions2

Backpropagation

The widely used training method for a lot of variations of neural networks are called backpropagation. Backpropagation is used to update the weights in a neural network, by differentiating backward in a neural network. When training models with machine learning, a loss- or a cost function is defined as a measurement of how good the network is performing.

To calculate new and more optimized weights, the loss function needs to be minimized, which naturally opens up the need for a gradient. The gradient of the loss function with respect to the weights and bias are calculated by using the aforementioned backpropagation utilizing the chain rule. This method revolves around calculating the gradient one layer at a time, starting with the last layer's activation(s) and moving backwards layer by layer, therefore the name of the algorithm. The gradients are continued to be computed through the neural network structure to find the final gradient.

The training regime goes as follows, starting of with some data fed forward into the dense neural network which then predicts the output a^{last} in the last layer. Which then is used to calculate the loss $L(a^{last}, y)$, where y is the target value. The prediction will certainly have some error defined by the loss function. Which then are optimized by using some optimization technique, which will be presented in section Section 2.3.

The gradient is computed by using the chain rule and Equation 2.7 to express the activation's in the prior layers [7, ch. 7] [8], and recursively computing the gradients backward into the net. The derivative of some node with respect to some weight and some bias can be written as follows respectively, the derivation can be seen in the appendix:

$$\frac{L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial W_{ij}^{l}} = z_i^l \frac{\partial a_i^l}{\partial W_{ij}^l}$$
$$\frac{L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial b_{i}^{l}} = z_i^l \frac{\partial a_i^l}{\partial b_i^l}$$

Where z_i^l is given as the following:

$$z_i^l = \frac{\partial L(a^{last}, y)}{\partial a_i^l} = \sum_n z_n^{l+1} \frac{\partial a_n^{l+1}}{\partial a_i^l}$$

Which shows how the gradients of activation of nodes in the same layers are summed up and being used in the calculations of the gradients of the layers closer to the input layer in dense neural networks.

sec:BM

2.2 The Boltzmann Machine

The Boltzmann machine was proposed by Geoffrey Hinton and Terry Sejnowski in 1985 and led on to a massive interest in Boltzmann machines at the time being[9]. Boltzmann machines are unsupervised machine learning networks using an undirected graph structure to process information [10]. The network is often referred to as a stochastic recurrent neural network, which is able to learn probability distributions over a set of inputs[11]. The reason that Boltzmann machines are stochastic, is because the decision on a node being on or off is made stochastic. Boltzmann machines consists of visible and hidden nodes, and often biases. The Boltzmann machine is often easier to grasp for the reader after having a look at the schematic structure first, which can be seen in Figure 2.3. All nodes is connected to each other, where the connections are the essence of the network. The connections between the nodes are also called weights, and are used to store prior learned knowledge[10]. The connections have varying strengths which means that the training of Boltzmann machines are based on optimizing these same weights, until the given input results in the desired output with high probability.

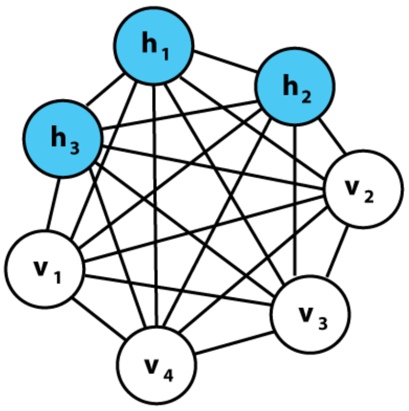


Figure 2.3: Architecture of a Boltzmann machine, having every node connected to each other. This Boltzmann machine consists of four visible nodes and three hidden nodes. Figure from [12]

fig:BM structure

Having a look at the stochastic dynamics of Boltzmann machines presented in [11], some mathematical formulas need to be presented. First of all a unit z_i can be turned on or off, representing binary states usually $z_i \in \{-1, 1\}$ or $z_i \in \{0, 1\}$. The visible and hidden nodes at a certain configuration $\mathbf{z} = \{\mathbf{v}, \mathbf{h}\}$ is then used to compute the energy of the system in a certain configuration by using the following formula, summing through each hidden- and visible node:

$$E(\mathbf{z};\theta) = -\sum_{i}^{N} \tilde{\theta}_i z_i - \sum_{i,j}^{N} W_{ij} z_i z_j$$

Where N is the number of nodes in the Boltzmann machine, $\theta = \{\tilde{\theta}, W\} \in \mathbb{R}$ represents the trainable parameters. W is the interaction matrix defining the strength between nodes and $\tilde{\theta}$ is the bias appended to the nodes.

Updating the nodes consistently until the computations converge, will sooner or later end up with a so called Boltzmann distribution, often called equilibrium distribution. After this the probability of observing a certain configuration of the visible nodes v sampled from the Boltzmann distribution is given by the following formula:

$$p(\mathbf{z};\theta) = \frac{e^{-E(\mathbf{z};\theta)/(\mathrm{k_B T})}}{Z}$$

With k_b being the Boltzmann constant, T is the temperature of the system. $k_b T$ is quiet often set to be equal to 1 which will be done in this thesis also, therefore the factor will be left out in future energy equations in the thesis. Z is the so called canonical partition function given as follows:

$$Z(\mathbf{z};\theta) = \sum_{v,h}^{N} e^{-E(\mathbf{z};\theta)}$$

2.2.1 The restricted Boltzmann machine

A quiet well known feature about Boltzmann machines is that the learning algorithm has the potential to be quiet slow due to a hard time converging towards a final state[11]. A way to deal with this problem is to implement a so called restricted Boltzmann machine instead.

The difference between a regular Boltzmann machine and a restricted Boltzmann machine is that no nodes in the same layer is connected to one another in the restricted one, while in a regular Boltzmann machine nodes usually are connected to all other nodes[13]. The structure of a restricted Boltzmann machine can be seen in Figure 2.4.

Should switch out the figure with a nicer one

Binary Restricted Boltzmann machine

RBMs are originally based on binary stochastic visible and hidden variables, with a given energy function of a joint state of visible and hidden nodes looking as follows:

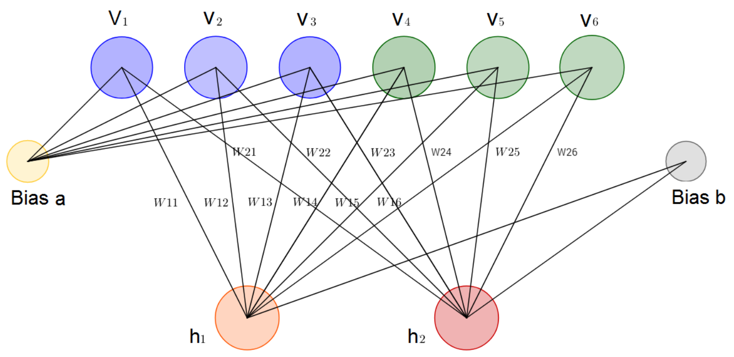


Figure 2.4: Architecture of a restricted Boltzmann machine, having the visible nodes connected to every hidden node and the visible bias, while the hidden nodes are connected to all the visible nodes and the hidden bias. The W's in the image is values of the weight matrix which decides how strong the connection between each visible and hidden node is. Figure from [14].

fig:RBM_structure

$$E(\mathbf{v},\mathbf{h};\theta) = -\sum_{i,j}^{V,H} W_{ij}v_i h_j - \sum_{i}^{V} b_i v_i - \sum_{j}^{H} a_j h_j \tag{2.8}$$

{eq:energfunkeruu}

With V and H being the number of visible and hidden nodes respectively, $\theta = \{W, \mathbf{b}, \mathbf{a}\} \in \mathbb{R}$ represents the trainable parameters; W being the interaction matrix, defining the strength between visible and hidden nodes, b and a represents the visible and hidden bias respectively.

One of the wonders of the structure of RBMs is that due to the hidden nodes and visible nodes being bipartite, the joint probability can be written as a marginal probability p_v as a sum over hidden nodes as follows[15]:

$$p(\mathbf{v}) = \frac{1}{Z}\sum_{\mathbf{h}}^{H} \exp(E(\mathbf{v},\mathbf{h};\theta))$$

Which gives the following by inserting the expression of Equation 2.8 and factorizing the terms not including any nodes within the exponential outside the sum:

$$p(\mathbf{v}) = \frac{1}{Z}\exp\left(\mathbf{b}^\top\mathbf{v}\right)\prod_{j}^{H}\sum_{h_j\in\{0,1\}}\exp\left(a_j h_j + \sum_{i=1}^{V} W_{ij}v_i h_j\right)$$

Since h_j only takes takes $\{0, 1\}$ as values, the last summation can be written out completely which then gives the final marginal probability:

$$p(\mathbf{v}) = \frac{1}{Z} \exp\left(\mathbf{b}^\top \mathbf{v}\right) \prod_{j}^{H} \left(1 + \exp\left(a_j + \sum_{i=1}^{V} W_{ij} v_i\right)\right) \tag{2.9}$$

{eq:marg_prob}

To determine and updating the state of a node, a value is chosen stochastically from a conditional probability. The probability of having a hidden node 'on' taking the binary value of 1 is given by the following equation:

$$p\left(h_j = 1 \mid \mathbf{v}\right) = \delta\left(\sum_{i}^{V} W_{ij} v_i + a_j\right) \tag{2.10}$$

{eq:CP_BH}

Where δ is a sigmoid function. The same conditional probability of having a visible node 'on' is given by:

$$p\left(v_i = 1 \mid \mathbf{h}\right) = \delta\left(\sum_{j}^{H} W_{ij} h_j + b_i\right) \tag{2.11}$$

{eq:CP_BV}

Even though binary RBMs are useful to such a large degree, sometimes real-valued inputs are preferred over binary inputs. Luckily binary RBMs are easily extended to accept continuous input values.

Gaussian-Binary Restricted Boltzmann machine

From time to time binary inputs of RBMs are not always the instinctive input of physical problems, depending on the data. Fortunately RBMs are extended to accept real valued visible inputs quiet straightforwardly. When using real valued visible vectors and binary hidden nodes taking values of $\{0, 1\}$ the model is called a Gaussian-Binary Restricted Boltzmann machine due to the visible units embracing a Gaussian distribution, and the hidden nodes being binary. Sometimes this type of RBM is referred to as a Gaussian-Bernoulli RBM. The energy function is given as the following [15]:

$$E(\mathbf{v}, \mathbf{h}; \theta) = \sum_{i}^{V} \frac{\left(v_i - b_i\right)^2}{2\sigma_i^2} - \sum_{i}^{V} \sum_{j}^{H} W_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{j}^{H} a_j h_j \tag{2.12}$$

{eq:energyfunctiongauss}

With $\theta = \{W, \mathbf{a}, \mathbf{b}, \sigma\}$ being the trainable parameters. σ is the standard deviation. The standard deviation is often predetermined prior to the training.

The marginal probability is then derived the same way as for Equation 2.9, which gives the following marginal distribution:

$$P(\mathbf{v}; \theta) = \sum_{\mathbf{h}} \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{\int_{\mathbf{v}'} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta)) d\mathbf{v}'}$$

With the denominator being the integral over all possible visible states.

The conditional probability of the visible vector taking a value x given a set hidden vector $\mathbf{h}$ is derived from the energy function in Equation 2.12 which gives the following:

$$p\left(v_i = x \mid \mathbf{h}\right) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{\left(x - b_i - \sigma_i \sum_j^H h_j W_{ij}\right)^2}{2\sigma_i^2}\right) \tag{2.13}$$

{eq: CP_CV}

With the conditional probability of a certain hidden node being 'on' given an observed visible vector $\mathbf{v}$ is given as follows:

$$p\left(h_j = 1 \mid \mathbf{v}\right) = \delta\left(b_j + \sum_i^V W_{ij} \frac{v_i}{\sigma_i}\right) \tag{2.14}$$

{eq: CP_CH}

With δ being the sigmoid function.

2.2.2 Gibbs sampling

Due to the objective of training a Boltzmann machine being to represent an intrinsic distribution of provided data, the reconstruction of data is still needed to be done. This is done by sampling from the probability distribution learned by the Boltzmann machine. There are multiple ways of sampling from a learned distribution (e.g. the Metropolis algorithm and the Metropolis-Hastings algorithm), however Gibbs sampling will be utilized in this thesis.

Gibbs sampling is an algorithm based on Markov chain Monte Carlo methods(MCMC). Due to drawing exact quantities from a distribution made by a probabilistic model not always being a walk in the park, MCMC algorithms are therefore used to estimate such quantities.

A Markov chain is essentially a stochastic process which is based on modelling a discrete sequence of random variables in a system, where the next state of the system is only dependent of the current state[16]. A Monte Carlo method on the other hand is used to estimate statistical quantities from drawing samples from a distribution, assumed that samples easily can be drawn from the distribution of relevance. MCMC combines these two methods to draw samples from strenuous-sampled distributions.

Gibbs sampling is an excellent choice of sampling method when dealing with multivariate probability functions in addition to having good access to the conditional distributions. Gibbs sampling is a subcategory of the Metropolis-Hastings algorithm. The essence of Gibbs sampling is to use the conditional distributions to refresh nodes by the probabilities computed, and thereby construct Markov chains following accordingly. Gibbs samples in Boltzmann machines are produced by picking a random node in the network, and updating the node based on the state of the other nodes and the conditional probabilities.

The conditional probabilities for binary- and continuous-binary RBMs are given in Equation 2.10, 2.11, 2.13 and 2.14.

sec: optimization

2.3 Optimizing the Supervised Learning Algorithms

So how is it possible to optimize machine learning models to achieve desired results. As explained in section 2.1, the essence of machine learning depends on optimizing models by finding parameters that give the best predictions applied to unseen data. The process of optimizing parameters towards a complete machine learning model is usually done by finding the parameters giving the lowest loss computed by some loss function L. The loss function evaluates how accurate a prediction is. Then by using an optimization technique it is possible to change the parameters in a way that minimizes the loss fitting the model to the data in other words.

2.3.1 Some loss functions

Mean absolute error

There are several possible loss functions which determines how good or bad a prediction is. Mean absolute error(MAE) is one of them. The MAE formula goes as follows:

$$L(x_i; \boldsymbol{\beta}) = \frac{1}{n} \sum_{i=0}^{n} |y_i - f(x_i; \boldsymbol{\beta})|$$

Where n is the number of samples, y_i is the true label of the i'th sample and $f(x_i; \boldsymbol{\beta})$ is the prediction of the sample. MAE is often used when utilizing linear regression.

Mean squared error

Mean squared error(MSE) is one of the most popular loss functions and is normally used when dealing with linear regression, . The formula for MSE goes as follows:

$$L(x_i; \boldsymbol{\beta}) = \frac{1}{n} \sum_{i=0}^{n} (y_i - f(x_i; \boldsymbol{\beta}))^2$$

Using MSE penalizes so called outliers in the data to a larger degree compared to MAE due to the difference between the true- and predicted value being squared.

Cross entropy loss

The cross entropy loss function is a frequently used loss function within machine learning. Cross entropy is based on computing the total entropy between two probability distributions, and is especially common regarding classification problems. The formula for cross entropy goes as follows:

$$L(x_i;\boldsymbol{\beta}) = -\sum_{i=0}^{n} y_i log(f(x_i;\boldsymbol{\beta})))$$

Where n is the number of probability events.

Cross entropy is often transformed to a binary cross entropy function when dealing with classification between two classes:

$$L(x_i;\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=0}^{n} y_i log(f(x_i;\boldsymbol{\beta})) + (1-y_i)log(1-f(x_i;\boldsymbol{\beta}))$$

Which naturally strips away the first or second part of the formula based on the fact that the true label y_i is either zero or one. In addition, by taking the logarithm of the prediction error, the more confident a prediction is, the larger the loss of the sample will be if the prediction is untrue.

2.3.2 Finding coefficients in linear regression

sec:ols

Ordinary least squares regression

Ordinary least squares (OLS) regression is one of the most common regression model. To fit a linear model the most popular method is using least squares method, which is based on using the residual sum of squares(RSS) as a cost funtion [1, ch. 2]. It is worth mentioning that cost- and loss functions often is used interchangeably in some literature, while in reality cost function is some mean of a loss function applied to multiple samples. The RSS function goes as follow:

$$\text{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - f(x_i;\boldsymbol{\beta}))^2. \tag{2.15}$$

{eq:rss}

Given n datapoints, sample x and prediction coefficients $\boldsymbol{\beta}$. Due to Equation (2.15) being quadratic, the minimum of the function is guaranteed to exist.

The cost function we use for OLS regression is the residual sum of squares (RSS) function:

Finding the optimal parameters $\boldsymbol{\beta}$ by minimizing the RSS is done the following way using matrix notation:

$$\text{RSS}(\boldsymbol{\beta}) = (\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^2 = (\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})$$

Which further on is differentiated with respect to $\boldsymbol{\beta}$:

$$\frac{\partial \mathrm{RSS}}{\partial \boldsymbol{\beta}} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}).$$

Where the derivative naturally is set to 0 and used to find the final coefficients:

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = 0$$

If the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is non singular hence invertible, the final coefficients can be written as follows:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.16}$$

{eq: rss_ols}

OLS models are quiet simple compared to other predictive models, hence OLS models has quiet high bias but low variance at the same time. This makes OLS regression at risk of overfitting. Luckily there are other models which ensures higher variance, less bias and less overfitting.

Lasso regression

Lasso regression is a shrinking method, which penalizes the prediction coefficients based on the sizes of them by a factor λ.

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}. \tag{2.17}$$

{eq: L1_algo}

It is easy to notice that the first part of Equation 2.17 is the RSS function, while the rest of the formula is the attached regularization. By reducing the coefficients, the model ensures less bias and higher variance making overfitting less likely to happen.

rid

Ridge regression

Ridge regression is also a shrinking method, which penalizes the prediction coefficients based on the sizes of them. The penalty is proportional with the squared size of the coefficients, which secures even more variance and less bias in the model compared to Lasso. The optimal coefficients is left looking as follows:

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\},$$

Which gives the final parameters by following the same procedure as for Equation 2.16

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

2.3.3 Batch gradient descent

Almost all machine learning tasks can be defined as a task of optimizing some function $f(\theta)$ [17]. A clever way of finding the best parameters θ which minimizes the loss, is the well known numerical minimization method named batch gradient descent often referred to as just gradient descent.

By formulating a minimization problem as follows:

$$\theta^* = \arg\min_{\theta} f(\theta)$$

Where θ^* are the optimal parameter which minimizes the function $f(\theta)$. The gradient descent is applied by moving towards the negative gradient, closing in towards the minimum more and more every step taken with the gradient descent.

The gradient decent method is quiet straight forward. An initial parameter θ, also known as a guess, is needed to start of the method and compute the next θ. The gradient decent formula goes as follows:

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta J(\theta_n)$$

Where $J(\theta)$ is an parameterized objective function, where the parameters is given by the model's parameters $\theta \in \mathbb{R}^d$ and η is the step size taken each step.

Batch gradient descent updates the parameters after the gradient is computed for the whole dataset before updating the parameters θ [18]. Due to the computation being ran for multiple samples before updating θ once, the road towards the minimum might get rough time- and memorywise.

2.3.4 Stochastic Gradient Decent

The optimization of parameters can be quiet expensive computational wise when using vanilla gradient descent mentioned in the last section. Therefore stochastic gradient descent(SGD) is often used alternatively for a more stable and faster minimization process. The SGD have quiet a bit of similarities to the normal gradient descent method, but instead of using gradients computed from the whole dataset, the parameters θ are updated for each training sample instead, lifting a lot of weight of the memory when using the SGD in addition to usually much faster convergence towards the optimal parameters. A step using SGD step can then be computed by the following formula.

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta J(\theta_n; x^i; y^i)$$

Where x is the $i'th$ training sample, and y the corresponding label.

2.3.5 Adaptive moment estimation optimizer (ADAM)

Adaptive moment estimation also called Adam is an optimized gradient descent technique which is quiet popular within deep learning. The optimizer uses adaptive learning rates meaning the learning rate varies depending on the gradient. The Adam optimizer uses momentum or exponential moving average(EMA), which accelerates the method in the relevant direction and can be explained by visualizing a ball rolling down a hill building up momentum on the way down, or loosing momentum on the way up a hill.

Having a look at the formulas used in the Adam optimizer, $g_n = \nabla_{\theta_n} J(\theta_n)$ for clarity. The first part to be computed is the EMA for the mean m_n and the EMA for the variance v_n.

$$\begin{aligned} m_n &= \beta_1 m_{n-1} + (1-\beta_1)\, g_n \\ v_n &= \beta_2 v_{n-1} + (1-\beta_2)\, g_n^2 \end{aligned} \tag{2.18}$$

{eq:m_v_andv_v}

Where β_1 and β_2 are decaying rates, often set to 0.9 and 0.999 respectively.

The terms are then bias-corrected, ensuring true weighted averaged. The reason for doing this is because m_n and v_n are initialized with vectors of 0's which have a tendency of having terms biased toward 0 [18].

$$\begin{aligned} \hat{m}_n &= \frac{m_n}{1-\beta_1^n} \\ \hat{v}_n &= \frac{v_n}{1-\beta_2^n} \end{aligned}$$

Which then are used in the final update of the parameters θ:

$$\theta_{n+1} = \theta_n - \frac{\eta}{\sqrt{\hat{v}_n} + \epsilon} \hat{m}_n$$

Where ϵ is a small tolerance often with an order in the realm of 10^{-8}.

2.3.6 AMSGrad

AMSGrad is quiet similar to ADAM, and was originally a solution to deal with problems where ADAM failed to converge in some settings. The proposed solution was the optimization method AMSGrad. One issue of the non-convergence problem using the ADAM optimizer were proved to be due to the use of exponential moving averages[19]. This was simply solved by switching out the squared gradient term in ADAM with a maximum of prior squared gradients.

In addition bias corrections of the EMAs were also removed. AMSGrad looks as follows:

$$\hat{v}_n = max(\hat{v}_{n-1}, v_n)$$

Which gives the following rule of parameter update:

$$\theta_{n+1} = \theta_n - \frac{\eta}{\sqrt{\hat{v}_n} + \epsilon} m_n$$

With v_n and m_n being defined as in Equation 2.18

2.3.7 RMSProp

The funny thing about RMSProp is that it is quiet well known, even though it was never officially published, but rather proposed in a lecture by Geoff Hinton [18]. RMSProp is an adaptive learning rate optimizer which uses the squared gradients from present and past iterations to tune the learning rate during training. The squared gradient term are computed the following way:

$$E\left[g^2\right]_n = 0.9E\left[g^2\right]_{n-1} + 0.1g_n^2$$

Where the new parameters are computed by dividing the learning rate by the squared gradient term as follows:

$$\theta_{n+1} = \theta_n - \frac{\eta}{\sqrt{E\left[g^2\right]_n + \epsilon}} g_n$$

2.4 Scaling of Data

Scaling of data is an important part of data preprocessing in machine learning. Scaling the data before letting loose of the machine learning models brings some well known practical apsects with it. A lot of loss functions defines how good a model performs based on the Euclidean distance between the predicted value and the target value and updates the model accordingly during training. When outliers are present in the dataset these will affect the model more than the rest due to their loss being a lot higher. Therefore by scaling the data, outliers and the rest of the datapoints are pushed closer together, and thereby affecting the model approximately the same, without outliers having extra grasp of how the model updates.

Another reason to scale the data is that models using regularization like Ridge or Lasso for instance, should have consistent penalties without outliers. In addition by scaling features, convergence of training is easier achieved when using gradient descent[20]

2.4.1 Standardization

Standardization also called Z-score normalisation is a very popular scaling method within machine learning. The scaling is based on subtracting the mean ensuring that the values have zero mean, in addition to division by the standard deviation which ensures unit-variance. The formula for standardization goes as follows:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Where x' is the new value of the datapoint, $\bar{x}$ is the average of the feature values and σ is the standard deviation of the same feature vector.

2.4.2 Min-max normalisation

Min-max normalisation, also called rescaling is a scaling method used to scale feature values into a certain target range of values in the range of $[a, b]$. The formula goes as follows, min-max scaling is usually scaled in the range of $[0, 1]$. The formula is quiet simple and goes as follows:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

Where $\max(\cdot)$ and $\min(\cdot)$ is the max and min of the given feature vector.

PART II

Quantum Mechanics and Many-Body Methods

CHAPTER 3

Quantum mechanics

sec: third

3.1 Basics of Quantum Mechanics

To understand quantum computing and naturally many-body methods as well, one should know a bit of quantum mechanics. Therefore it is natural to have a short summary of some basic principles of quantum mechanics for an easier grasp of the exciting topics unveiled in the thesis. The summary of the basics of quantum mechanics is quiet shallow and more focused on the mathematical essence. For a more in depth explanation please have a look at [21] and [22] which will be the sources representing the overview in this section.

3.1.1 Bra-ket notation and wave functions

Before going into further details of quantum mechanics, it is worth mentioning a couple of words about the notation that will be used. A wave function Ψ is a function that describes a quantum mechanical system at all times. To describe an isolated physical system a more mathematical way, bra-ket notation is used. Since Ψ is an abstract vector describing a quantum state, this is written as a ket $|\Psi\rangle$. All wave functions are represented in the complex vector space called Hilbert space $\mathcal{H}$, which not necessarily consists of a finite number of dimensions [23, ch. 6]. All wave functions can be written as a sum of basis vectors the following way:

$$|\Psi\rangle = \sum_i c_i |\psi_i\rangle \tag{3.1}$$

{eq: statevector_PSI}

Where ψ_i is a basis vector and c_i is some complex number.

To find the probability of finding a quantum system in state $|\psi_i\rangle$, the inner product between the two are squared:

$$|\langle\psi_i|\Psi\rangle|^2 = |c_i|^2$$

Which is the square product of the expansion coefficient form equation Equation 3.1

It is assumed that the unit norm of Ψ equals 1, due to the fact that $|c_i|^2$ translates to the probability of finding the system in state ψ_i and the system must take place in one of the possible quantum states ψ.

In the dual space of a ket vector $|\Psi\rangle$, lays a bra vector $\langle\Psi|$. Bra vectors is mapped to the dual space the following way:

$$|\Psi\rangle \rightarrow \langle\Psi| = \sum_i c_i^* \langle\psi_i|$$

Where the inner product between two basis vectors are given as follows:

$$\langle\psi_i|\psi_j\rangle = \delta_{ij}$$

Where δ_{ij} is the Kronecker delta product:

Knowing that inner products between normalized vectors from the same orthogonal basis set, either equals 0 or 1 gives the so called Kronecker delta product:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Another property of wave functions living in the abstract vector space $\mathcal{H}$ worth mentioning is that the Hilbert space is complete, which means that any Ψ in the Hilbert space can be constructed as a linear combination of the basis vectors $\{\psi_1, \psi_2 \ ... \ \psi_n\}$. Writing the wave function as follows:

$$|\Psi\rangle = \sum_i \langle\psi_i \mid \Psi\rangle |\psi_i\rangle = \sum_i |\psi_i\rangle \langle\psi_i \mid \Psi\rangle$$

Where the coefficient of ψ_i is given by:

$$c_i = \langle\psi_i \mid \Psi\rangle$$

Which means that the identity matrix I must be the following for a complete set of basis vectors:

$$\sum_k |\psi_k\rangle \langle\psi_k| = I$$

This is also called the completeness relation.

3.1.2 Quantum operators and evolution of quantum systems

A quantum operator $\hat{A}$ being applied to a vector in Hilbert space is a mapping onto itself [23, ch. 6.3]. The mapping and relation between a ket- and bra vector can be written as follows:

$$\hat{A}|\Psi\rangle = |\Psi'\rangle \qquad\qquad \langle\Psi'| = \langle\Psi|\hat{A}^\dagger$$

Where $|\Psi'\rangle$ is the new quantum state. To the right the same quantum state $|\Psi'\rangle$ is written as a bra vector by applying the adjoint operator $\hat{A}^\dagger$.

Sometimes operators commute, which is quiet usefull in quantum mechanics. Comuttation of operators can be written as follows:

$$[\hat{A}, \hat{B}] = 0$$

Which means that the order the operators are applied to some eigenstate is irrelevant the following way:

$$\hat{A}\hat{B}|\Psi\rangle = \hat{B}\hat{A}|\Psi\rangle$$

It is also worth mentioning that the Schrödinger equation is written as follows:

$$\hat{H}\,|\Psi\rangle = E\,|\Psi\rangle$$

Where $\hat{H}$ is the Hamiltonian operator and E is the eigenvalue of Ψ also known as the total energy.

Because measuring the energy E gives real physical values, this is called an observable. In quantum mechanical systems there are a lot of variables and specification which can be measured. These measurable quantities are called observables, and could for instance be the energy, momentum, position or spin of a quantum system [24, ch. 13]. For an operator to represent observable information from a quantum system, the operator needs to equal its own conjugate transpose as follows:

$$\hat{A} = \hat{A}^\dagger$$

So how does a quantum system evolve? Or expressed differently, how is evolution in quantum systems expressed mathematically? One of the so called quantum mechanical postulates is stated as follows in [22, ch. 2.2]:

> The evolution of a closed quantum system is described by a unitary transformation. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t1 and t_2,
>
> $$|\psi'\rangle = U|\psi\rangle$$

Basically what this means is that the new state of a quantum system can be described by applying some unitary operator on the current wave function, which redeems a wave function describing the new quantum system after the evolution has taken place.

3.2 Measurement in Quantum Mechanics

Talking about measurements in quantum mechanics, a good place to start is with a quantum mechanical postulate also stated in [22, ch. 2.2]:

> Quantum measurements are described by a collection $\{M_m\}$ of measurement operators. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by
>
> $$p(m) = \left\langle \psi \left| M_m^\dagger M_m \right| \psi \right\rangle$$
>
> and the state of the system after the measurement is
>
> $$\frac{M_m|\psi\rangle}{\sqrt{\left\langle \psi \left| M_m^\dagger M_m \right| \psi \right\rangle}}$$
>
> The measurement operators satisfy the completeness equation,
>
> $$\sum_m M_m^\dagger M_m = I$$
>
> The completeness equation expresses the fact that probabilities sum to one:
>
> $$1 = \sum_m p(m) = \sum_m \left\langle \psi \left| M_m^\dagger M_m \right| \psi \right\rangle$$

Which is more of a postulate which describes how an experimentalist affects a quantum mechanical system during a measurement, due to the fact that a closed system is no longer closed when it interacts with the 'world of the experimentalist' giving away information when it is measured. This explains a quiet general way of measuring quantum states, but there is also a more special case related way of measuring quantum states when measuring observables called projective measurements.

3.2.1 Projective measurements

An observable M being a Hermitian operator, which is observed on the state space of the system. The observable can be represented as a spectral decomposition:

$$M = \sum_m m\hat{P_m}$$

With M being the eigenspace with m eigenvalues and $\hat{P_m}$ being a projector which projects into eigenspace M. The probability of measuring the observable m when measuring $|\psi\rangle$ is then given by the following:

$$p(m) = \langle \psi | P_m | \psi \rangle .$$

Which gives the following state of the quantum state immediately after the measurement:

$$\frac{P_m|\psi\rangle}{\sqrt{p(m)}}$$

3.2.2 Global and relative phases

It is worth mentioning global and relative phases in quantum mechanics. A global phase factor of $e^{i\theta}$ where θ is a real number, applied to a quantum state $|\psi\rangle$ the following way: $e^{i\theta}|\psi\rangle$ is equal to $|\psi\rangle$ up to the global phase factor $e^{i\theta}$.

Carrying out a measurement as done in the postulate, shows that the probability of measuring eigenvalue m is the same with and without the global phase factor.

$$p(m) = \left\langle \psi \left| e^{-i\theta} M_m^\dagger M_m e^{i\theta} \right| \psi \right\rangle = \left\langle \psi \left| M_m^\dagger M_m \right| \psi \right\rangle$$

Which shows that global phase factors can be ignored from an observers standpoint.

Relative phases on the other hand are dependent of the basis. Having two quantum states with amplitudes α and β differ by a relative phase in the basis if there is a real number θ which is able to make the following relation true $\alpha = \exp(i\theta)\beta$. Even though two states differ by a relative phase, the observable quantities can be different due to the fact that they only differ by some phase in that specific basis, compared to quantum states that differ by global phase factors which shows the same observables.

3.3 Entangled States

As mentioned earlier wave functions of quantum systems live in the complex vector space called a Hilbert space. But what if we have two separate quantum systems hence two separate Hilbert spaces. Normally one would describe one of the systems, without having the necessity of referring to the other system, this is called a pure state when the exact quantum state is known and a mixed state when holding a superposition. But what if a quantum state depends on the state of both quantum systems? Such systems are called composite systems and brings a quantum property called entanglement[2, ch. 3].

Entanglement of quantum systems are the cornerstone of why quantum computing embodies such quantities of being more efficient than classical computers when solving certain problems. The reason for this is that quantum states (also called qubits in quantum computers) can entangle with each other, making the state of a quantum sub-system affect the quantum state of another sub-system. A quantum state $|\psi\rangle$ consisting of sub-states $|\psi_A\rangle$ and $|\psi_B\rangle$ with $|\psi_A\rangle \in \mathcal{H}_A$ and $|\psi_B\rangle \in \mathcal{H}_B$ having the Hilbert space being expressed as a tensor product of the Hilbert spaces of the sub-systems:

$$\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$$

With the following states in the composed Hilbert space:

$$\sum_{i=1}^{N_A} \sum_{j=1}^{N_B} c_{ij} \left| e_i^A \right\rangle \otimes \left| e_j^B \right\rangle$$

With e_k^x being the orthonormal basis vectors of $\mathcal{H}_x$ in dimension k, N_A and N_B represent the number of dimensions in $\mathcal{H}_A$ and $\mathcal{H}_B$ respectively. With the wave functions being normalized, hence the amplitudes of the complex coefficients c_{ij} sum up to 1. To conclude if a quantum state $|\psi\rangle$ is separable or entangled, it suffice to observe that a quantum state expressed as a tensor product of the sub-states on the following form is separable, hence not entangled [2, ch. 3]:

$$|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle \tag{3.2}$$

{eq: separable}

Now imagine two individuals, Alice and Bob holding each their quantum state called a qubit which can take two distinctive states, $|0\rangle$ and $|1\rangle$(qubits will be explained in more detail in section 5.1). The following state is separable, hence unentangled:

$$|0\rangle_A |0\rangle_B$$

Where the first qubit held by Alice is in state 0 and the same with Bob. Now imagine the following quantum state:

$$|\Phi\rangle_{AB} = \frac{1}{\sqrt{2}} \left(|0\rangle_A |0\rangle_B + |1\rangle_A |1\rangle_B \right) \tag{3.3}$$

{eq:Bell_ allice_ bob}

Which can be shown to be entangled due to not being able to write the state on the form of Equation 3.2 using each component separately[25, ch. 3]:

$$(a_1|0\rangle_A + b_1|1\rangle_B) \otimes (a_2|0\rangle_A + b_2|1\rangle_B) = \frac{1}{\sqrt{2}} \left(|0\rangle_A |0\rangle_B + |1\rangle_A |1\rangle_B \right) \tag{3.4}$$

{eq:bell_ separabel}

Which gives the following:

$$\begin{aligned}(a_1|0\rangle_A + b_1|1\rangle_B) \otimes (a_2|0\rangle_A + b_2|1\rangle_B) =& a_1a_2(|0\rangle_A|0\rangle_B) + a_1b_2(|0\rangle_A|1\rangle_B) \\ &+ b_1a_2(|1\rangle_A|0\rangle_B) + b_1b_2(|1\rangle_A|1\rangle_B)\end{aligned}$$

There is no values for a_1, a_2, b_1 and b_2 that can make Equation 3.4 come true, since having $a_1b_2 = 0$ or $a_2b_1 = 0$ would make a_1a_2 or b_1b_2 also equal 0, hence the state is not able to be written as separable terms and is therefore entangled. Nevertheless, this specific state were chosen on purpose, the reason being that this state is one of the most popular states representing entanglement in quantum mechanics, also called a Bell state.

Now back to the Bell state in Equation 3.3. Imagine Alice measures her qubit, which by random chance measures it to be in state 0, now since the wave function of Alice and Bobs qubits only gives room for measuring $|0\rangle_A|0\rangle_B$ or $|1\rangle_A|1\rangle_B$ means that Bobs qubit also have to be in state 0. This is because they are entangled. If Bob measures his qubit before Alice it would be 0 or 1 by a 50% chance, but the state of Alices qubit will be the same as Bobs, by a 100%. Which shows some of the impressive characteristics of quantum mechanics, and that it is possible to know the properties of a system by only measuring its entangled sub-system.

CHAPTER 4

Many-Body Methods

sec: fourth

section: TVM

4.1 The Variational Method

Most many-body systems in quantum mechanics have the property of being highly advanced and complex. The more complex the system is, the less is the probability of having an analytical solution present. The road from a simple quantum mechanical system with an analytical solution present, to a system too complex for an analytical solution is very small. For example a hydrogen atom has one electron and can be computed analytically quiet easily, but when adding an electron and trying to compute the energy of a helium atom it becomes impossible to calculate an analytical solution due to the complexness of the system. This is why the variational method often is used to find a highly accurate estimate.

The variational method is based on the fact that the expectation value of the Hamiltonian H is an upper bound for the ground state energy E_{gs}, for any wave function chosen. The proof can be seen in [21, ch. 8, p. 327]

$$E_{gs} \leq \langle\Psi|H|\Psi\rangle \equiv \langle H\rangle \tag{4.1}$$

{eq:Egs}

Where Ψ is a normalized trial wave function chosen. The trial wave function can be any wave function but it is usually chosen according to the quantum mechanical system, due to similar systems usually having quiet similar wave functions. To ensure lowest possible energy, the trial wave function contains some parameter(s) that is determined by deciding which value of the parameter(s) is giving the lowest energy, knowing that this energy is just an upper bound for the ground state energy [21].

4.2 Quantum System: Hydrogen Molecule

The quantum system that will be investigated in the thesis is the Hydrogen molecule H_2. The hydrogen molecule is a quiet popular quantum system to find the grounds state energy of when assessing new many-body algorithms. The H_2 molecule consists of two nuclei forming a covalent bond with two electrons.

4.2.1 Hydrogen Hamiltonian

The Hamiltonian used in this thesis is the Born-Oppenheimer approximation of the hydrogen molecule. The Born-Oppenheimer approximation states that the Hamiltonian of a molecular Hamiltonian is given by the sum of the kinetic energy operator of the nuclei and electrons and the interaction operators between nuclei and electrons [26], with the Hamiltonian being put together by the following terms:

$$\hat{H} = \hat{T}_n + \hat{H}_e + V_{nn}$$

With $\hat{T}$ representing the kinetic energy operator, and V representing being Coulomb interactions. The subscript n denotes that the operator acts on the nuclei, while the e denotes that the operator acts on the electron. The electron Hamiltonian operator $\hat{H}_e$ is given by the following:

$$\hat{H}_e = \hat{T}_e + V_{ee} + V_{ne}$$

Since the nuclei compared to the electrons contains so much more mass, it is within reason to assume that the electron compared to the nucleus speed will be so mismatched that the nucleus speed will be insignificant. Therefore the nuclei be can be treated as fixed point charges. The kinetic energy of the nuclei is therefore set equal to 0, in addition to setting the interaction energy between the nuclei to 0 because this distance between them are set, hence constant. The Hamiltonian is then left looking as follows[27, S.M.][1]:

$$-\sum_i \frac{\nabla_i^2}{2} - \sum_{i,I} \frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2}\sum_{i\neq j} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

With Z_I and R_I being the charge and position respectively of nucleus I, and r_i being the position of electron i. The unit of the expression is atomic units.

Should the nucleii distance be included? Because how else is the distance that important?? Try including the distance, 0.5 with and without the term

sec: hydrogenhamil

4.2.2 The Hamiltonian

The Hamiltonian operator of the system is given by the following:

$$\hat{H} = \hat{H}_0 + \hat{H}_1 \tag{4.2}$$

Where $\hat{H}_0$ is the unpertubed Hamiltonian, and $\hat{H}_1$ is the pertubed Hamiltonian.

The unpertubed Hamiltonian includes a standard harmonic oscillator part:

[1]S.M.:Supplementary material

$$\hat{H}_0 = \sum_{i}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2 \right) \tag{4.3}$$

{eq: Hamilton}

Where N is the number of particles in the system, and ω is the trap frequency. Natural units are used. The pertubed repulsive part of the Hamiltonian is given as the following:

$$\hat{H}_1 = \sum_{i<j}^{N} \frac{1}{r_{ij}} \tag{4.4}$$

Where r_{ij} is the distance between the particles. r_{ij} is given by $r_{ij} = |r_i - r_j|$, with r_p given as $r_p = \sqrt{r_{px}^2 - r_{py}^2}$.

Which let the Hamiltonian of the whole system be written as follows:

$$\hat{H} = \sum_{i}^{N} \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2 \right) + \sum_{i<j}^{N} \frac{1}{r_{ij}} \tag{4.5}$$

{eq: hamilhamil}

4.2.3 RBM: Neural network quantum state as the wave function

Presented in an article by Carleo and Troyer[28], it is possible to use artificial neural networks to represent the wave function and solve many-body problems. When the wave function is represented this way, the wave function is called a neural-network quantum state(NQS).

Instead of working with training data as most machine learning methods do, the machine learning method will use the fact that minimizing the energy by optimizing the weights and biases of the NQS is giving the best solution, often referred to as reinforcement learning.

Generally the wave function represented by the probability distribution which will be modelled goes as follows for a quantum mechanical system:

$$\Psi = \sqrt{F_{rbm}(\mathbf{X}, \mathbf{H})} = \sqrt{\frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{X}, \mathbf{H})}} \tag{4.6}$$

{eq: wavef}

Where X is a vector of visible nodes representing the positions of the particles and dimensions, H is a vector of the hidden nodes. In this thesis the RBM that will be used is a Gaussian-Binary type, meaning that the visible nodes has continuous values, while the hidden nodes are restricted to be either 0 or 1. T_0 is set to 1, Z is the partition function/normalization constant which is given as follows:

$$Z = \int\int \frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})} d\mathbf{x} d\mathbf{h}$$

$E(\boldsymbol{X}, \boldsymbol{H})$ is the joint probability distribution defined:

$$E(\mathbf{X}, \mathbf{H}) = \frac{||\mathbf{X} - \mathbf{a}||^2}{2\sigma^2} - \mathbf{b}^T \mathbf{H} - \frac{\mathbf{X}^T \mathbf{W} \mathbf{H}}{\sigma^2}$$

Where $\boldsymbol{a}$ is the biases connected to the visible nodes with the same length as $\boldsymbol{X}$. $\boldsymbol{b}$ is the biases connected to the hidden nodes with the same length as $\boldsymbol{H}$. $\boldsymbol{W}$ is a matrix containing the weights deciding how strong the connections between the hidden and visible nodes are.

Writing the marginal probability as

$$F_{rbm}(\mathbf{X}) = \sum_{\mathbf{h}} F_{rbm}(\mathbf{X}, \mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{X}, \mathbf{h})}$$

The sampling method used will be Gibbs sampling, with the j'th hidden node being updated by the conditional probabilities:

$$P\left(h_J = 1 \mid \boldsymbol{X}\right) = \delta\left(\delta^{input}\right)$$

and

$$P\left(h_J = 0 \mid \boldsymbol{X}\right) = \delta\left(-\delta^{input}\right)$$

with $\delta(\cdot)$ being the sigmoid function and δ^{input} being defined as

$$\delta_j^{input} = b_j + \sum_{i=1}^{M} \frac{X_i w_{ij}}{\sigma^2}$$

The visible nodes are then updated according to the hidden nodes by the following condition:

$$P\left(X_i \mid \mathbf{h}\right) = \mathcal{N}\left(X_i; a_i + \mathbf{w}_{i+}\mathbf{h}, \sigma^2\right)$$

The systems local energy

The local energy of the system can be computed by combining Equation 4.1, the Hamiltonian in Equation 4.5 and the wave function from Equation 4.6 which gives the following expression when defining the local energy E_L as follows:

$$E_L = \frac{1}{\Psi} H\Psi = -\sum_{i}^{N} \frac{1}{2\Psi} \nabla_i^2 \Psi - \sum_{i,I}^{N,N_I} \frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2} \sum_{i \neq j}^{N} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

With N being the number of electrons and N_I being the number of nuclei.

Add nuclei interaction also?

After inserting the wave function the final analytical expression for the local energy can be written as follows:

$$E_{L,Gibbs} = -\frac{1}{2}\sum_{i=1}^{M}\Bigg[\frac{1}{4\sigma}\left(a_i - X_i + \sum_{j=1}^{N} w_{ij}\delta(\delta_j^{input})\right)^2 \\ -\frac{1}{2\sigma^2} + \sum_{j=1}^{N}\frac{w_{ij}^2}{2\sigma^4}\delta(\delta_j^{input})\delta(-\delta_j^{input}) - \sum_{i,I}\frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|}\Bigg] + \sum_{i<j}\frac{1}{r_{ij}}$$

Where M is the number of visible nodes, and N is the number of hidden nodes. $\delta(\cdot)$ is a sigmoid function, and δ_j^{input} is defined as follows:

$$\delta_j^{input} = b_j + \sum_{i=1}^{M}\frac{X_i w_{ij}}{\sigma^2}$$

Where the derivation can be seen in the appendix.

4.3 Slater Determinants

4.4 Second Quantization

4.4.1 Particle Holde formalism

4.4.2 Second quantized Hydrogen Hamiltonian which gets Jordan Wigner transformed in chapter 5

From subsection 4.2.2 the Hamiltonian of the hydrogen molecule was written using a Born-Oppenheimer approximation with fixed point charge nuclei. Now to further rewrite the Hamiltonian into an expression appropriate for a quantum computer, the Hamiltonian first have to be written as a second quantised representation. The electronic Hamiltonian consisting of terms regarding electrons is given by the following expression[27, S.M.]:

$$H = \sum_{p,q} h_{pq}a_p^\dagger a_q + \frac{1}{2}\sum_{p,q,r,s} h_{pqrs}a_p^\dagger a_q^\dagger a_r a_s$$

With ϕ being the electronic wave function, and $a^\dagger$ and a being creation and annihilation operators respectively. h_{pq} being the one-body integral given as follows:

$$h_{pq} = \int \mathrm{d}\mathbf{x} \phi_p^*(\mathbf{x}) \left(-\frac{\nabla_i^2}{2} - \sum_I \frac{Z_I}{|\mathbf{r} - \mathbf{R}_I|} \right) \phi_q(\mathbf{x})$$

And h_{pqrs} is the two-body integral given as follows:

$$h_{pqrs} = \int \mathrm{d}\mathbf{x}_1 \, \mathrm{d}\mathbf{x}_2 \frac{\phi_p^* (\mathbf{x}_1) \, \phi_q^* (\mathbf{x}_2) \, \phi_s (\mathbf{x}_1) \, \phi_r (\mathbf{x}_2)}{|\mathbf{r}_1 - \mathbf{r}_2|}$$

With $\boldsymbol{x}$ representing both the position and spin.

Done? There is more in the supplement but maybe wait until the JW trasformation or bravie kitaev transformation?

4.5 Configuration Interaction Theory

4.6 Pairing Hamiltonian

PART III

Quantum Computing:Machine Learning

CHAPTER 5

Quantum Computing: Machine Learning

sec: fifth

sec: basic_qc

5.1 Basics of Quantum Computing

Quantum computing uses the occurrences of quantum mechanics to process and manipulate information. Quantum computing has endless of possibilities to speed up computations when solving certain problems. But to understand the beauty of machine learning within quantum computing, one needs to know the basics of quantum computing first, here comes a short summary with the most important key elements to grasp the article. For a deeper dive into the theory on quantum computing, it would be worth starting here [22].

The key element when comparing classical computers with quantum computers is the comparison between classical bits and qubits. A classical computer uses bits to handle and manipulate information, where each bit can be either 0 or 1. A quantum computer on the other hand uses qubits. Qubits have the possibility of taking the form of superpositions between the original bits of 0 and 1, often explained by using the Blocks sphere in Figure 5.1. A qubit can be a superposition of two states, $\alpha|0\rangle + \beta|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$ which opens up the possibility of having multiple states at once.

5.1.1 Basis

All operations on qubits can be expressed mathematically. Therefore defining a basis of two possible qubit states as orthogonal vectors are only natural. The two states a qubit can have when being observed are $|0\rangle = [1, 0]^T$ and $|1\rangle = [0, 1]^T$. Where $\langle 1|0\rangle = 0$ and $\langle 0|0\rangle = 1$ due to basis states being orthogonal. The basis states expands into $|00\rangle = [1, 0, 0, 0]^T$, $|01\rangle = [0, 1, 0, 0]^T$, $|10\rangle = [0, 0, 1, 0]^T$ and $|11\rangle = [0, 0, 0, 1]^T$ when having two qubits, which naturally increases the vector size the same pattern when adding more qubits. Linear combinations of the different states are also possible, making superpositions as follows:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{5.1}$$

Where $|\alpha|^2 + |\beta|^2 = 1$.

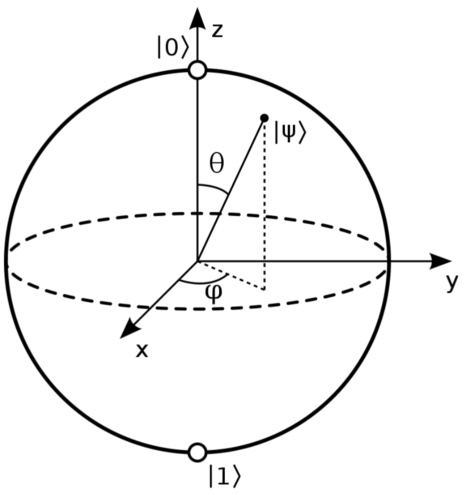


Figure 5.1: The Bloch sphere showing a geometrical representation of the quantum states of a qubit. A qubit can take a position anywhere on the Bloch sphere of the following form: $|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle$, while classical bits are prohibited from taking positions other than the top and bottom. Figure from [29]

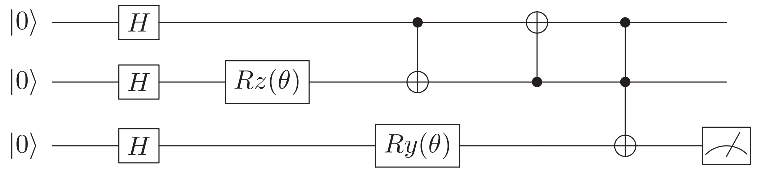
fig: blochsphere

Multiple qubits can be expressed as tensor products of the qubits, the following way:

$$|\psi_1\rangle \otimes |\psi_2\rangle ... \otimes |\psi_n\rangle = |\psi_1\psi_2...\psi_n\rangle \tag{5.2}$$

5.1.2 Quantum circuits

Quantum circuits consists of quantum wires transferring information. The information is then manipulated by letting some quantum logical gate also called a quantum gate, act on the current state. A quantum circuit could for instance look like the following example:



Each horizontal line is called a quantum wire, and each of these represents time evolving from left to right. Each quantum wire represents a qubit which can be altered and manipulated. The start point $|0\rangle$ represents the initial state of the qubit, while the end of the qubit represents the quantum state after the quantum gates have acted on the quantum states. When going from left to right, the quantum states encounters some quantum gates represented by boxes containing a letter or just some circles and dots, which unveils the type of gate it is. These quantum gates are the operators which alter the quantum states. Vertical lines between quantum wires shows the presence of controlled gates between the two connected wires. Some quantum gates will be discussed in further details in the next subsection. The last part of the third qubit in the drawn circuit, reveals that the measurement of a qubit will happen here. The results from the measurement then reveals which state the last qubit is in. By measuring the same qubit multiple times gives the probability of having the state in either state $|0\rangle$ or $|1\rangle$ [30].

sec:not

Quantum Logical Gates [31]

Quantum logical gates or quantum gates which they also are called are different ways of manipulating the information in quantum circuits. Quantum gates are expressed mathematically by matrices. There are infinite many matrices, which makes it possible to manipulate quantum states infinite different ways in theory. Some quantum states are more popular than others, lets have a look at some of the most frequently used ones.

The **Pauli gates** are often used within quantum circuits. The Pauli X-gate or NOT gate which it is also named can be thought of as a rotation around the x axis π radians. The NOT gate are represented by the following matrix and bracket:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0| \tag{5.3}$$

Which interchanges the labels between the two qubit states the following way:

$$X|0\rangle = |1\rangle \tag{5.4}$$

The Pauli Y-gate can be thought of as a rotation around the y axis π radians. The Y-gate are represented by the following matrix and bracket:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = -i|0\rangle\langle 1| + i|1\rangle\langle 0| \tag{5.5}$$

Which maps $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$. the following way:

$$Y|0\rangle = i|1\rangle \tag{5.6}$$
$$Y|1\rangle = -i|0\rangle \tag{5.7}$$

The Pauli Z-gate works the same way as the other Pauli gates. The Z-gate are represented as a rotation around the z axis π radians. The Z-gate are represented by the following matrix and bracket:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1| \tag{5.8}$$

Which maps $|1\rangle$ to $-|1\rangle$ and works as an identity operator on $|0\rangle$ states the following way:

$$Z|0\rangle = |0\rangle \tag{5.9}$$
$$Z|1\rangle = -|1\rangle \tag{5.10}$$

But since there exists operators that rotates the quantum states around certain axis by π radians, there must also exist rotational operators that rotate the qubits an arbitrary amount of radians. In fact the **rotational operator gates** does this exactly. By inserting a parameter θ one can rotate the quantum states around the x, y, or z axis by that many radians. The rotational operators are given by the following matrices:

$$R_x(\theta) = \exp(-iX\theta/2) = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \tag{5.11}$$

$$R_y(\theta) = \exp(-iY\theta/2) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \tag{5.12}$$

$$R_z(\theta) = \exp(-iZ\theta/2) = \begin{bmatrix} \exp(-i\theta/2) & 0 \\ 0 & \exp(i\theta/2) \end{bmatrix} \tag{5.13}$$

Where the subscript indicates which axis the rotation is happening along.

The **Hadamard gate** also called a H-gate are one of the most common quantum gates used. The H-gate are represented by the following matrix and bracket:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \langle 0| + \frac{|0\rangle - |1\rangle}{\sqrt{2}} \langle 1| \tag{5.14}$$

Which maps $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. The following way:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{5.15}$$

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{5.16}$$

Controlled quantum gates are quantum gates that acts on two or more qubits when the criteria of another quantum state are satisfied. The quantum state that needs to satisfy the condition is the controlled gate. An example of a popular controlled gate are the well known CNOT which is a controlled NOT gate. The CNOT on two qubits controls if the first qubit is in state $|1\rangle$. If the controlled qubit possess the state, the NOT gate will act on the second qubit mapping $|0\rangle$ to $|1\rangle$ and opposite. If the controlled qubit does not contain the criteria state, the CNOT will not act on the state, and instead act as an identity operator. Controlled gates are expressed in quantum circuit diagrams as vertical lines between two or more qubits as in figure Figure 5.2, where the dark dots are the control qubits and the other are the gate that potentially acts on the state.

Quantum circuit depth

The more quantum gates a circuit has, the deeper it is. Quantum circuit depth is the longest path in the cirquit along the quantum wires. Because a quantum state is evolving through time as quantum gates is manipulating the information, time complexity is also used to reefer to the depth or the time steps needed to execute the cirquit.

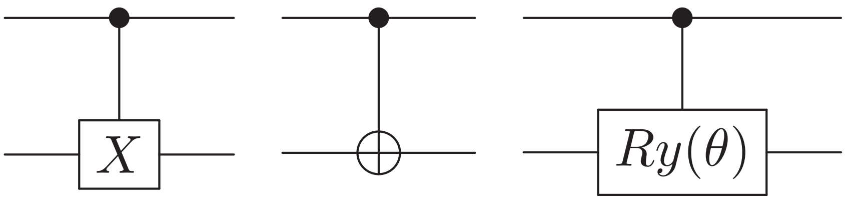


Figure 5.2: Some controlled quantum gates. a) A controlled NOT gate, called a CNOT. b) Also a CNOT but a slightly different visual appearence but does the same as a controlled NOT gate. c) A controlled Ry gate.

fig:cnot

The depth of quantum cirquits is based on the fact that keeping quantum states stable for longer periods of time is quiet hard, hence circuits need to be short enough to stay stable during the process. The circuit needs to be complex enough to give satisfactory results, but still be simple enough to keep the circuit depth within sensible restrictions, this is way time complexity of a circuit often is the bottleneck when performing quantum computations.

sec:PQC

5.1.3 Parameterized Quantum Circuits

Parameterized quantum circuits also called PQCs are quantum circuits consisting of one or more quantum gates that are dependent of some variational parameter. Most often multiple quantum gates that depends of some parameter(s) is put together into a circuit which is used as an ansatz. The ansatzes used in this article will be revealed in section 5.1.3.

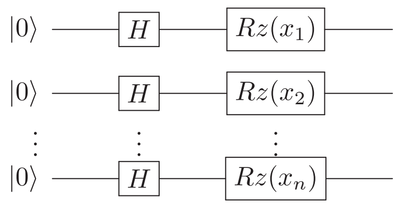
So how is PQCs connected to supervised learning? Shortly summarised, supervised learning aims to learn trends from data that can be generalized to predict other unseen data. In search of the perfect generalisation of data, the machine learning models need to adjust to training data by optimizing the parameters decreasing the loss which is computed from the prediction estimates and the true labels.

Using this same approach by having the PQC which is dependent on some parameters, predict an output and optimize the parameters used in the circuit gives a machine learning approach to PQCs. To read more about machine learning and PQCs feel free to have a look here [32].

Encoders

Encoding the information that is going to be manipulated in quantum circuits is the first part of the quantum circuits and therefore crucial. There are multiple ways of encoding the data e.g. basis encoding, amplitude encoding, qsample encoding etc. Different encoding methods can be studied in [2, ch. 5].

A way of encoding the data which will also be done in this thesis, is by having every qubit start off with a Hadamard gate splitting the quantum state into a superposition which is then acted on by a $Rz(x)$ gate. The clue on encoding classical data into quantum states lays here, because the input x will be the classical feature data. The encoding part of the ansatzes used be as follows:



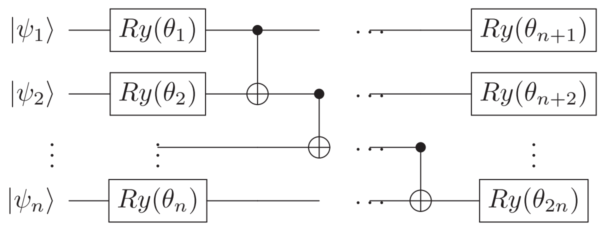
Where n is the number of qubits. By rewriting each feature into a $Rz(x)$ quantum gate which is dealt its own qubit, the needed qubits is therefore equal to the number of features, excluding possible qubits used for measuring.

sec_ansa

Ansatzes

Ansatzes used in quantum computations are parts of quantum circuits which is assumed to mimic the behaviour of some function. Making the ansatz more complex opens up the possibility of reaching more parts of the Hilbert space where all quantum states resides. Making really large and complex quantum circuits is difficult due to quantum states being quiet unstable over time, which is why ansatzes must avoid being too complex. In this article, two ansatzes will evaluated.

An ansatz could for instance consist of rotational operators $Ry(\theta)$ along each qubit, which is then followed by some entangling between the qubits using controlled not gates between every pairwise qubit as follows:



Where $\boldsymbol{\theta}$ is the variational parameters that needs to be optimized and n is the number of qubits. When more variational parameters are used, the ansatz adds more $Ry(\theta)$ gates following the same pattern.

Notice that the input states of the circuit is $|\psi_x\rangle$ since the input of the ansatz-part of the circuit is the already altered quantum states from the encoder.

Entanglement of the qubits

The last part of a PQCs is the entangling part which entagles all qubits together, making it possible for the qubits to affect eachothers quantum state. The remarkable thing about quantum entanglement is that affecting a qubit might make rise to a change in another qubit, which is one of the main reasons quantum computing is such a powerful tool.

The entangling part is an extra part of the quantum circuit which is appended onto the ansatz. A way of entangling the example ansatz could for instance be a controlled not gate controlling all other qubits except the target qubit which is the last qubit. A measurement could then be performed on the last qubit, giving the probability $|c_1|^2$ of having the qubit in state $|1\rangle$, which can be used as a classification prediction between class $|0\rangle$ and class $|1\rangle$.

sec:qnn

5.2 Quantum Nearest Neighbours

This section is from earlier report, rewrite it to match the thesis, especially the discussion part

Quantum Nearest neighbours is based on the same strategy as in section 2.1.3, but with some smaller implementation differences to make the method suitable for quantum computing. This section is based on the example provided in [2, ch. 1]

To use the nearest neighbours method with quantum computers, the first step is naturally to do a little preprocessing of the data. The data is first and foremost normalised to one. This way the data is only described by the angles according to the origin since all samples have the same unit length from the origin.

The next step is to encode the data. A safe and secure way to encode the data is to use amplitude encoding. The amplitude vector is made by concatenating the features of the data, and the normalised data. After the amplitude vector is made, the number of qubits needed to perform the algorithm are decided.

The next step is to apply the quantum operations. When performing the nearest neighbours method the Hadamard is the quantum operator to use. The Hadamard operator acts on the amplitude vector, which creates a new amplitude vector. Since the Hadamard operator maps the states into superpositions, the new amplitude vector contains a lot of different states and values which will be rejected or accepted when the measurement starts.

Then the rejection sampling starts by measuring the first qubit, rejecting and setting the value to zero if the state are $|1\rangle$, hence only keeping the accepted states $|0\rangle$ or opposite, which then are normalised.

The ancilla qubit is measured, and the amplitudes in similar states are squared and added together, which results in the final probabilities to find the unknown datapoint within the given category. A theoretical example of the method will be explored in section 2.1.3.

5.2.1 Quantum computing case

Then it is time to have a look at the same example if implemented on a quantum computer. Classifying the unknown sample in the small dataset from table 2.1 by using nearest neigbours with quantum computing is done by following the recipe in section 5.2. The first step is to preprocess the data by having the vector of ticket price and room number normalized. This way all the samples will have the same length from the center, making the angles the only relevant descripting of the sample placements. The dataset normalized to unit length can be seen in table 5.1.

Table 5.1: Table showing some samples used for creating a small dataset. The dataset is showing the ticket price the travelers paid and the room number before normalisation(BN), and the same information after normalising the data to unit length(AN) assuming the maximum ticket price and room number was 10000 and 2500 respectively. The survival column shows the survival of the travelers, 1 means survival while 0 means no survival.

Traveler	Price (BN)	Room number (BN)	Price (AN)	Room number (AN)	Survival
1	8500	0910	0.921	0.390	1
2	1200	2105	0.141	0.990	0
3	7800	1121	0.866	0.500	

tab: dataset_tit2

The new values can be plotted the same way as in the classical case, giving the plot in figure 5.3

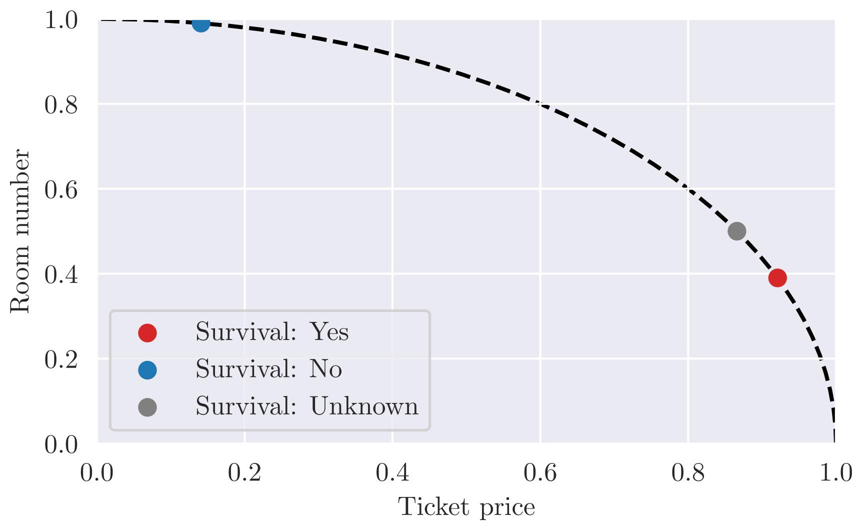


Figure 5.3: Plot of the ticket price and room number for the three travelers, after normalizing the data to unit length.

fig: knn_qua

Next step follows and the data is encoded into amplitude vectors. The features of the data are appended to the same vector, and the unknown sample are appended twice, which makes the unknown sample data added and subtracted to both the known samples, which gives the following vector:

$$\alpha = \frac{1}{2}(0.921, 0.39, 0.141, 0.99, 0.866, 0.5, 0.866, 0.5)^T \tag{5.17}$$

Where the factor $\frac{1}{2}$ at the start is a normalisation factor, to ensure that the sum of the squared amplitudes equals 1.

Since three qubits have the possibilliy of having $2^3 = 8$ states, one more qubit is added representing the two categories, which results in the following amplitude vector having $2^4 = 16$ states. The states added from the fourth qubit are currently only zeros, leaving the following amplitude vector:

$$\alpha = \frac{1}{2}(0, 0.921, 0, 0.39, 0.141, 0, 0.99, 0, 0, 0.866, 0, 0.5, 0.866, 0, 0.5, 0)^T \tag{5.18}$$

Which can be written in the form of quantum states as follows:

$$\begin{aligned}|\psi_0\rangle = \frac{1}{2}(&0.921|0001\rangle + 0.390|0011\rangle + 0.141|0100\rangle + 0.990|0110\rangle \\ &+0.866|1001\rangle + 0.500|1011\rangle + 0.866|1100\rangle + 0.500|1110\rangle)\end{aligned} \tag{5.19}$$

Now that the preprocessing and encoding is done the Hadamard transformation can begin. The Hadamard operator acts on the current state, which adds and subtracts the last four amplitudes to the first four amplitudes which gives the following results:

$$\begin{aligned}|\psi_1\rangle = H|\psi_0\rangle = \frac{1}{2}(&(0.921 + 0.866)|0001\rangle + (0.390 + 0.500)|0011\rangle \\ &+(0.141 + 0.866)|0100\rangle + (0.990 + 0.500)|0110\rangle \\ &+(0.921 - 0.866)|1001\rangle + (0.390 - 0.500)|1011\rangle \\ &+(0.141 - 0.866)|1100\rangle + (0.990 - 0.500)|1110\rangle)\end{aligned} \tag{5.20}$$

{eq:sub2}

Which written in a more compact way equals:

$$\begin{aligned}|\psi_1\rangle = H|\psi_0\rangle =&0.8935|0001\rangle + 0.445|0011\rangle + 0.5035|0100\rangle + 0.745|0110\rangle \\ &+0.0275|1001\rangle - 0.055|1011\rangle - 0.3625|1100\rangle + 0.245|1110\rangle\end{aligned} \tag{5.21}$$

{eq:sub}

Now the next step is to reject half of the states due to the fact that half of the samples has the unknown sample data summed to their amplitude, while the rest has the same data subtracted, which is more clear having a look at equation (5.20). There is only need for one linear combination containing the unknown sample data, which is why half of the data is rejected. It is easy to see from equation (5.21) that the states that needs to be rejected are the states which has the first qubit in state $|1\rangle$, else the amplitude is set to 0. Then the resulting states named ψ_2 are as follows:

$$|\psi_2\rangle = \frac{1}{\chi}(0.8935|0001\rangle + 0.4450|0011\rangle + 0.5035|0100\rangle + 0.7450|0110\rangle)) \tag{5.22}$$

Which equals the following after normalisation.

$$|\psi_2\rangle = 0.665|0001\rangle + 0.331|0011\rangle + 0.375|0100\rangle + 0.555|0110\rangle)) \tag{5.23}$$

As mentioned at the start of the subsection, the fourth qubit was added to be used as a label indicator. Which means that states with equal fourth qubits represents the same category. Then the amplitudes of the states in the same category can be used to compute the probability of finding the unknown sample in each of the category by adding the squared amplitudes within each class as follows:

$$p(|xxx0\rangle) = |0.375|^2 + |0.555|^2 \approx 0.448 \tag{5.24}$$

Which means that the probability of finding the unknown sample as a traveler that did not survive equals 0.448. While the probability of finding the traveler as a survivor equals:

$$p(|xxx1\rangle) = 1 - p(|xxx0\rangle) = |0.665|^2 + |0.331|^2 \approx 0.552 \tag{5.25}$$

Which means that the unknown passenger is classified as a survivor, which is the same as for the classical case of nearest neighbours.

QNN vs. classical NN

The results of the calculations done in section 2.1.3, showed that computing a nearest neighbour method for both a classical and quantum computing case managed to classify the unknown samples correct. In addition the probability of classifying the unknown case within each category were also the same when using equation (2.5) and (2.6) according to [2, ch. 1]. Which makes sense due to the Hadamard operator being an universal classical gate, the proof can be seen here **cite:hadamardproof**.

Comparing the complexity of both the methods reveals that the quantum implementation of nearest neighbours is more efficient the more samples the dataset contains because the quantum states possible are proportional to 2^n using n qubits. The quantum implementation of the nearest neighbours used one Hadamard transformation and two measurements, which is quite few operations remembering the fact that with just these few manipulations it is possible to classify unknown samples.

The bad thing in the classical case is that the length to its neighbours need to be computed for each sample, which can be quiet computationally expensive when dealing with a large number of samples, and even more expensive if k is high and a lot of the neighbours are contributing to the classification of each unknown sample.

In the quantum computing case, there might be some loss of information when normalizing the data to unit length. The loss of information could be that samples within the same classes might get pushed away from each other when forcing each sample along the same unit circle. Observing the example with quantum nearest neighbours showed that in this case loss of information was not a problem.

5.3 Variational Quantum Boltzmann Machine

Maybe use this as the section headline, then ite under. Add a subsection proposing a neural network finding the coefficients.

sec:
VarITE

5.4 Variational Quantum Imaginary Time Evolution

In a variational quantum Boltzmann machine(VarQBM) Gibbs states need to be prepared beforehand, a way to do this is to use a method called variatioanal quantum imaginary time evolution(VarQITE), the details on VarQBM and how to prepare the Gibbs states using varQITE will be revealed later in the theory section, but firstly the details on classical imaginary time evolution(ITE) and varQITE need to be laid out first.

ITE is quiet usual when looking for ground state energies and studying quantum systems using classical computers [27]. Following the explanation in [10] ITE is based on starting of with an initial quantum state $|\psi_0\rangle$. ITE is then used to evolve $|\psi_0\rangle$ into quantum state $|\psi_\tau\rangle$ at time τ using the following time-independent Hamiltonian H:

$$H = \sum_{i=0}^{p-1} \theta_i h_i \tag{5.26}$$

Where θ_i is some real coefficients and h_i represent some Pauli matrix. The evolved quantum state can then be denoted as the following:

$$|\psi_\tau\rangle = C(\tau) e^{-H\tau} |\psi_0\rangle \tag{5.27}$$

Where $C(\tau)$ is the normalization factor given as the following:

$$C(\tau) = 1/\sqrt{\mathrm{Tr}\left[e^{-2H\tau} |\psi_0\rangle \langle\psi_0|\right]} \tag{5.28}$$

The evolution of the quantum state is represented by the Wick-rotated Schrödinger equation. Wick rotations is used to solve mathematical problems by substituting real variables with imaginary variables, done here by introducing imaginary time [33]. The Wick-rotated Schrödinger equation is represented by the following differential equation:

$$\frac{d|\psi_\tau\rangle}{d\tau} = -\left(H - E_\tau\right)|\psi_\tau\rangle \tag{5.29}$$

{eq:
wick_SE}

With $E_\tau = \langle\psi_\tau|H|\psi_\tau\rangle$. According to Zoufal, Lucchi and Woerner [10], the terms in the Hamiltonian representing the largest eigenvalue decays faster than the terms representing the small eigenvalues. Due to this, as long as there is some overlap between the eigenvalue and the ground state, the eigenvalue found is in fact the ground state of the system when $\tau \to \infty$ [10]. Further on one can prepare the Gibbs states by having a finite limit $\tau = 1/2\,(\mathrm{k_B T})$ on the time evolution.

Why is that? The 'accordning to..' part

Now that the main details of ITE is unveiled, it is time to jump into varQITE. The idea of varQITE is to minimize the energy using McLachlan's variational principle [34] by introducing a parameterized trial state $|\psi_\omega\rangle$ in addition to representing the time evolution by some parameters $\omega(\tau)$.

Since varQITE is intended to be run on quantum computers, a quantum circuit is defined as $V(\tau) = U_q(\omega_q) \dots U_1(\omega_1)$. An initial quantum state is also defined as $|\psi_{in}\rangle$ which is the input state. The trial state is the written as follows:

$$|\psi_\omega\rangle := V(\omega) |\psi_{\text{in}}\rangle \tag{5.30}$$

Rewriting Equation 5.29 by throwing all terms to the left hand side and factorising the evolved state the equation can be written as follows

$$\left(\frac{d}{d\tau} + H - E_\tau\right) |\psi_\tau\rangle = 0 \tag{5.31}$$

Then by switching the evolved state $|\psi_\tau\rangle$ with the trial state $|\psi_\omega\rangle$ and multiplying with a factor δ to account for the error that arises when $|\psi_\tau\rangle \neq |\psi_\omega\rangle$, one arrives at McLachlan's variational principle:

$$\delta \left\| \left(\frac{d}{d\tau} + H - E_\tau\right) |\psi_\omega\rangle \right\| = 0 \tag{5.32}$$

{eq:mcvar}

Now McLachlan's variational principle is a central part in determining the parameters $\omega(\tau)$. Then solving Equation 5.32 leads to the following system of linear equations:

$$A\dot{\omega} = C \tag{5.33}$$

{eq:AwC}

Where $\dot{\omega} = \frac{d\omega}{d\tau}$ and A and C is given as the following:

$$\begin{aligned} A_{pq}(\tau) &= \text{Re}\left(\text{Tr}\left[\frac{\partial V^\dagger(\omega(\tau))}{\partial\omega(\tau)_p}\frac{\partial V(\omega(\tau))}{\partial\omega(\tau)_q}\rho_{\text{in}}\right]\right) \\ C_p(\tau) &= -\sum_i \theta_i \,\text{Re}\left(\text{Tr}\left[\frac{\partial V^\dagger(\omega(\tau))}{\partial\omega(\tau)_p} h_i V(\omega(\tau))\rho_{\text{in}}\right]\right) \end{aligned} \tag{5.34}$$

{eq:A_and_C}

With $Re(\cdot)$ being the real part of the expression and $\rho_{\text{in}} = |\psi_{\text{in}}\rangle\langle\psi_{\text{in}}|$. To evaluate A and C in varQITE, some expectation values need to be computed by using some quantum circuit composed following some rules, more on the evaluation and form of the quantum circuit will be unveiled in the next section.

Demanding that the matrices $U(\omega)$ within $V(\omega)$ are arbitrary unitary matrices $U(\omega)$ it is possible to evaluate the expressions A and C. The reason for this is that all unitary matrices can be written as $U(\omega) = e^{iM(\omega)}$ where $M(\omega)$ is a parameterized Hermitian matrix. The Hermitian matrices can then be rewritten as weighted sums of pauli terms as explained in [10]:

$$M(\omega) = \sum_p m_p(\omega) h_p \tag{5.35}$$

Where $m_p(\omega) \in \mathbb{R}$ and h_p is given as $h_p = \bigotimes_{j=0}^{n-1} \sigma_{j,p}$, j being which qubit the operation is performed on and $\sigma_{j,p} \in \{I, X, Y, Z\}$. The gradient of $U_k(\omega_k)$ is then defined as follows:

$$\frac{\partial U_k(\omega_k)}{\partial \omega_k} = \sum_p i \frac{\partial m_{k,p}(\omega_k)}{\partial \omega_k} U_k(\omega_k) h_{k_p} \tag{5.36}$$

.

No that the steps of varQITE is defined, the last step remaining is letting the parameters evolve with time, which can be done multiple ways, for instance with an explicit Euler the following way:

$$\omega(\tau) = \omega(0) + \sum_{j=1}^{\tau/\delta\tau} \dot{\omega}(\tau)\delta\tau \tag{5.37}$$

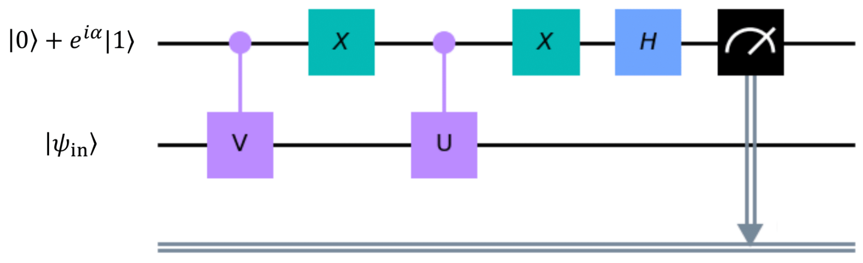
5.4.1 Evaluation of expression- A and C and their gradients

The expressions A and C from Equation 5.34 used in varQITE is a couple of expressions which is composed on the following form:

$$\mathrm{Re}\left(e^{i\alpha}\,\mathrm{Tr}\left[U^\dagger V \rho_{\mathrm{in}}\right]\right) \tag{5.38}$$

{eq:
trial_
state_
basic_
form}

By computing the expectation value of an observable Z it is possible to evaluate the two expressions in Equation 5.34 by sampling from Z with respect to the circuit in (5.39) according to [10] and [27][35][36].



(5.39)

{circ_
fig:
appendix}

Might draw the same figure with qcircuit. Should a circuit be an equation or a figure?

To make the phase factor $e^{i\alpha}$ go along with the expressions in Equation 5.34, the first qubit is initialized a bit differently for the two expressions. Expression A is initialised with an H gate, while C is initialised with an H- followed by an S gate.

It is also important to remember that since the trial states are constructed via Pauli rotations the matrix $U(\omega)$ can be written as follows:

$$U(\omega) = R_{\sigma_l}(\omega) \tag{5.40}$$

Where $\sigma_l \in \{X, Y, Z\}$. Which gives the following derivative:

$$\frac{\partial U(\omega)}{\partial \omega} = -\frac{i}{2}\sigma_l R_{\sigma_l}(\omega) \tag{5.41}$$

Further on the terms of the derivatives of expression A and C can be written on the same form as Equation 5.38. The derivative of expression A can be expressed as follows:

$$\begin{aligned}\partial_{\theta_i} A_{p,q}(\tau) = \sum_s \frac{\partial \omega_s(\tau)}{\partial_{\theta_i}} \operatorname{Re}\Bigg(\operatorname{Tr}\Bigg[\Bigg(&\frac{\partial^2 V^\dagger(\omega(\tau))}{\partial \omega_p(\tau)\partial \omega_s(\tau)} \frac{\partial V(\omega(\tau))}{\partial \omega(\tau)_q} \\ &+ \frac{\partial V^\dagger(\omega(\tau))}{\partial \omega(\tau)_p} \frac{\partial^2 V(\omega(\tau))}{\partial \omega_q(\tau)\partial \omega_s(\tau)}\Bigg)\rho_{\text{in}}\Bigg]\Bigg)\end{aligned} \tag{5.42}$$

And the derivative of expression C can be expressed as follows:

$$\begin{aligned}\partial_{\theta_j} C_p(\tau) = &- \operatorname{Re}\left(\operatorname{Tr}\left[\frac{\partial V^\dagger(\omega(\tau))}{\partial \omega(\tau)_p} h_j V(\omega(\tau))\rho_{\text{in}}\right]\right) \\ &- \sum_{i,s} \theta_i \frac{\partial \omega_s(\tau)}{\partial_{\theta_j}} \operatorname{Re}\Bigg(\operatorname{Tr}\Bigg[\Bigg(\frac{\partial V^\dagger(\omega(\tau))}{\partial \omega_p(\tau)} h_i \frac{\partial V(\omega(\tau))}{\partial \omega_s(\tau)} \\ &+ \frac{\partial^2 V^\dagger(\omega(\tau))}{\partial \omega_p(\tau)\partial \omega_s(\tau)} h_i V(\omega(\tau))\Bigg)\rho_{\text{in}}\Bigg]\Bigg)\end{aligned} \tag{5.43}$$

Where α is set equal to $\frac{\pi}{2}$ and 0 in the expressions corresponding to A and C respectively.

5.5 Quantum Boltzmann Machine

Before jumping into VarQBM, regular QBM which lays the basis of VarQBM needs to be explained.

As mentioned in section 2.2 a BM consists of hidden and visible units. A QBM on the other hand consists of visible and hidden qubits. A QBM can be defined as a parameterized Hamiltonian $H_\theta = \sum_{i=0}^{p-1} \theta_i h_i$ where $\theta \in \mathbb{R}^p$ and $h_i = \bigotimes_{j=0}^{n-1} \sigma_{j,i}$ with $\sigma_{j,i} \in \{I, X, Y, Z\}$ acting on qubit j.

The visible qubits determines the output of the QBM, which makes it natural to define the probability of measuring a configuration v with respect to the projective measurement $\Lambda_v = |v\rangle\langle v| \otimes I$ on the Gibbs state given by:

$$\rho^{\text{Gibbs}} = \frac{e^{-H_\theta/(\mathrm{k_B T})}}{Z} \tag{5.44}$$

Where $Z = \mathrm{Tr}\left[e^{-H_\theta/(\mathrm{k_B T})}\right]$ which gives the probability of measuring $|v\rangle$ as follows:

$$p_v^{\text{QBM}} = \mathrm{Tr}\left[\Lambda_v \rho^{\text{Gibbs}}\right] \tag{5.45}$$

As mentioned before, the purpose of a QBM is to compute probability distributions such that sampling ρ^{Gibbs} corresponds to sampling from the classical target distribution with as much overlap between the probabillity distributions possible. The loss used in QBM is the same as in classical Boltzmann machines:

$$L = -\sum p_v^{\text{data}} \log p_v^{\text{QBM}} \tag{5.46}$$

{eq: Loss_QBM}

With p_v^{data} being the probability of having v taking place in the training data. The loss function naturally depends on the variational parameters θ, such that the gradients are computed with respect to the same parameters. The difference between a regular QBM and a VarQBM which will be implemented in this thesis is that in VarQBM the gradient of the loss function is computed analytically.

That was a bit on regular QBMs, feel free to have a look at the following article for a more in depth explanation on QBMs [37].

Even though there are multiple similarities between QBMs and VarQBMs, there is still some key differences. Now that general QBMs are covered it is time to move further into the details of VarQBM. The steps of a VarQBM is visualized in figure 5.4.

5.5.1 Preparation of Gibbs states with VarQITE

Lets have a look at how it is possible to prepare the Gibbs state ρ^{Gibbs} using Variational Quantum Imaginary Time Evolution. The Gibbs state expresses a probability density operator. The density operator defines the configuration space of a system in thermal equilibrium [38] [10]. So how is ρ^{Gibbs} prepared? There are multiple different ways proposed which consists of strategies like Quantum Phase Estimation [39]–[41], evaluation of quantum thermal averages [42]–[44] and more. As said earlier VarQITE will be used for this task, due to

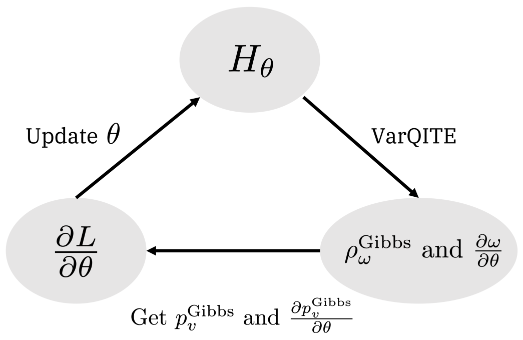

H_θ
Update θ
VarQITE
$\frac{\partial L}{\partial \theta}$
$\rho_\omega^{\text{Gibbs}}$ and $\frac{\partial \omega}{\partial \theta}$
Get p_v^{Gibbs} and $\frac{\partial p_v^{\text{Gibbs}}}{\partial \theta}$

Figure 5.4: Steps of VarQBM, first step is to approximate the Gibbs states using VarQITE. Next step is to find the gradient of the loss function which is then used to update the variational parameters according to some classical optimization algorithm. Figure from [10]

fig: varQBMsteps

the fact that this scheme is compatible with near-term quantum computers in addition to having a clear knowing of the needed ancilla qubits.

So lets walk through the steps of preparing the Gibbs state ρ^{Gibbs} , also here a lot of the walk through will follow [10]. First step is to set up the quantum circuit $V(\omega)$, $\omega \in \mathbb{R}^q$ and initialize the variational parameters to satisfy the following initial state:

$$|\psi_0\rangle = V(\omega(0))|0\rangle^{\otimes 2n} = \left|\phi^+\right\rangle^{\otimes n} \tag{5.47}$$

With $|\phi^+\rangle$ being the Bell state:

$$\left|\phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{5.48}$$

By introducing two n-qubit sub-systems a and b where the first qubit of the Bell states starts of in a and the second qubit in b, an effective Hamiltonian H_{eff} is defined as follows:

$$H_{eff} = H_\theta^a + I^b \tag{5.49}$$

With the superscript denoting which sub-system the operator acts upon. Since sub-system b will be a maximally mixed state [10], it will act as an ancillary qubit system, where tracing the sub-system gives the following:

$$\text{Tr}_b\left[\left|\phi^+\right\rangle^{\otimes n}\right] = \frac{1}{2^n} I \tag{5.50}$$

Next step is to find the approximation which is the goal of VarQITE. This is done by starting the propagation until time $\tau = 1/2(k_bT)$ is reached with respect to the effective Hamiltonian. Doing this gives the following state:

$$|\psi_\omega\rangle = V(\omega(\tau))|0\rangle^{\otimes 2n} \tag{5.51}$$

Which is used to compute $\rho_\omega^{\text{Gibbs}}$ as follows:

$$\rho_\omega^{\text{Gibbs}} = \text{Tr}_b[|\psi(\omega(\tau))\rangle\langle\psi(\omega(\tau))|] \approx \frac{e^{-H_\theta/(\text{k}_\text{B}\text{T})}}{Z} \tag{5.52}$$

And we arrive at the Gibbs state approximation using VarQITE.

5.5.2 Variational Quantum Boltzmann Machine

Finally it is time to unveil the VarQBM algorithm, and at the same time give an overview and connect the different parts together. As mentioned earlier, the goal of the method is to be able to reproduce probability distributions by optimizing some variational parameters θ such that when sampling from the generated distribution using Gibbs states prepared with VarQITE approximates the training data.

The variational parameters are optimized by minimizing Equation 5.46 the following way:

$$\min_\theta L = \min_\theta \left(-\sum_v p_v^{\text{data}} \log p_v^{\text{QBM}} \right) \tag{5.53}$$

Where the gradient is computed with respect to each variational parameter the following way:

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial \left(-\sum_v p_v^{\text{data}} \, \log p_v^{\text{QBM}} \right)}{\partial \theta_i} = -\sum_v p_v^{\text{data}} \frac{\partial p_v^{\text{QBM}}/\partial \theta_i}{p_v^{\text{QBM}}} \tag{5.54}$$

Which is computed by using the chain rule, which gives the following expression for the gradient:

$$\frac{\partial p_v^{\text{QBM}}}{\partial \theta_i} = \frac{\partial p_v^{\text{QBM}}}{\partial \omega(\tau)} \frac{\partial \omega(\tau)}{\partial \theta_i} = \sum_{k=0}^{q-1} \frac{\partial p_v^{\text{QBM}}}{\partial \omega_k(\tau)} \frac{\partial \omega_k(\tau)}{\partial \theta_i} \tag{5.55}$$

With

$$\frac{\partial p_v^{\text{QBM}}}{\partial \omega_k(\tau)} = \frac{\partial \, \text{Tr}\left[\Lambda_v \rho_\omega^{\text{Gibbs}}\right]}{\partial \omega_k(\tau)} \tag{5.56}$$

Which can be found by utilizing gradient methods like the parameter-shift rule [45]. By differentiating Equation 5.33, $\frac{\partial \omega_k(\tau)}{\partial \theta_i}$ can be found. The derivative of Equation 5.33 is given as follows:

$$A\left(\frac{\partial \dot{\omega}(\tau)}{\partial \theta_i}\right) = \frac{\partial C}{\partial \theta_i} - \left(\frac{\partial A}{\partial \theta_i}\right)\dot{\omega}(\tau) \tag{5.57}$$

Which can solved with respect to the wanted expression $\frac{\partial \dot{\omega}(\tau)}{\partial \theta_i}$, by evaluating the expression for every time step used in VarQITE. Further on by utilizing a differentiation method like explicit Euler method for instance, the following expression is given:

$$\begin{aligned}\frac{\partial \omega_k(\tau)}{\partial_{\theta_i}} &= \frac{\partial \omega_k(\tau - \delta\tau)}{\partial_{\theta_i}} + \frac{\partial \dot{\omega}_k(\tau - \delta\tau)}{\partial_{\theta_i}}\delta\tau \\ &= \frac{\partial \omega_k(0)}{\partial_{\theta_i}} + \sum_{j=1}^{\tau/\delta\tau} \frac{\partial \dot{\omega}_k(j\delta\tau)}{\partial_{\theta_i}}\delta\tau\end{aligned} \tag{5.58}$$

The optimizer is then utilized to optimize the variational paramters. In this thesis the Adam optimize will be chosen to optimize the VarQBM.

5.6 Expectation Values of Hamiltonian Quantum Circuits

When optimizing a trial circuit representing some ansatz to minimize some expectation energy, the ansatz naturally needs to be applied to the Hamiltonian to compute the expectation value of the energy, but how is this done when measuring qubits only provides a sampling probability between 0 and 1?

In context of Boltzmann machines all Hamiltonians compatible with Boltzmann distributions can be used and written as a sum of Pauli products[10]. Computing the expectation energy of a Hamiltonian given on the following form is actually fairly straightforward:

$$H = \sum_{i=0}^{p-1} \theta_i h_i,$$

with p being the number of Hamiltonian terms in the Hamiltonian, θ being some factor and h_i being a quantum gate. Now having an Hamiltonian act on a quantum state $|\psi\rangle$ represented as a quantum circuit used to find the expectation energy the following way:

$$\langle E \rangle = \langle\psi| H |\psi\rangle = \sum_{i=0}^{p-1} \theta_i \langle\psi| h_i |\psi\rangle . \tag{5.59}$$

{eq: expect_ energy}

Which shows how the different terms of a Hamiltonian is assessed. But one thing is missing, namely the expectation value of the Hamiltonian terms. First and foremost h_i is a tensor product of pauli gates. Pauli matrices are 2×2 matrices which looks as follows:

$$\sigma_{\mathrm{x}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_{\mathrm{y}} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_{\mathrm{z}} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{5.60}$$

When measuring quantum circuits, the measured quantum states are forced out of superposition and into some eigenstate, therefore a quantum circuit are measured multiple times and averaged over the possible states measured to get a statistical distribution of the possibilities to measure the different states, often referred to as sampling probabilities. The number of times measured is often referred to as shots. The most common basis to measure quantum circuits in is the $Pauli-Zbasis$ with eigenstates $|0\rangle$ and $|1\rangle$, there is no problem using other bases as well. The eigenvalues of eigenstate $|0\rangle$ and $|1\rangle$ equal 1 and -1 respectively. When summing the terms of Equation 5.59 the sampling probability of a given state is then multiplied by its corresponding eigenvalue -1 or 1, before multiplied by the factor θ.

Further on the Hamiltonian terms often consists of tensor products of multiple Pauli gates, which means that there are multiple qubits hence even more possible quantum states to collapse into. When evaluating a Hamiltonian term Z_0Z_1 meaning the first and second qubit consists of a Z gate each, there is four possible quantum states when measuring each of the qubits, since each qubit can take values of 0 or 1. The eigenvalues of each qubit is multiplied by eachother resulting in the addition or subtraction of the sampling probability of the given state. The same approach follows when utilizing Hamiltonians consisting of terms existing of an arbitrary number of qubits.

Even though the Pauli-Z basis is most often used to measure quantum circuits, the Hamiltonian terms might not always consist of Pauli-Z matrices. So how is quantum gates other than Pauli-Z gates measured in the Z basis? This is simply solved by rotating the statevector of the quantum state to point in the direction of the computational basis and measure same way as for Pauli-Z gates.

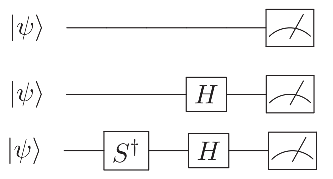
The easiest way to explain this process is probably by showing an example of circuits used. Lets assume some Hamiltonian is given by the following:

$$H = Z + X + Y, \tag{5.61}$$

which gives the following expectation value:

$$\begin{aligned} \langle\psi| H |\psi\rangle &= \langle\psi| Z |\psi\rangle + \langle\psi| X |\psi\rangle + \langle\psi| Y |\psi\rangle \\ &= \langle\psi| Z |\psi\rangle + \langle\psi| H'ZH' |\psi\rangle + \langle\psi| H'SZH'S^\dagger |\psi\rangle \\ &= \langle\psi| Z |\psi\rangle + \langle\psi H'| Z |H'\psi\rangle + \langle\psi H'S| Z \left|H'S^\dagger\psi\right\rangle, \end{aligned}$$

where H' is the Hadamard gate. Which shows that by adding a few gates to the quantum circuit it is possible to measure the contributions of X and Y oriented quantum states. The circuits used to measure the different contributions of Z,X and Y looks as follows:



Where the qubits are used to find $\langle\psi| Z |\psi\rangle$, $\langle\psi| X |\psi\rangle$ and $\langle\psi| Y |\psi\rangle$ respectively.

5.7 Jordan-Wigner Transformation

Write some general theory on JW

Write about how the qubits are sorted based on orbital, spin and spacial orbital

5.7.1 Jordan-Wigner transformation of the pairing Hamiltonian

Rewrite the second quantized pairing Hamiltonian into qubits form using JW

5.8 Theory on Variational Quantum Simulations? Got Some Nice Litterature on the Topic at the Start of the Thesis

PART IV

Method

CHAPTER 6

Method

sec:
sixth

6.1 Qiskit

There are lots of available quantum computing libraries making the implementation and simulation of quantum circuits and systems easier, but the chosen one used i this thesis is qiskit[31] which is a python library made by IBM, making it possible to simulate a q1uantum computer running a quantum circuits on a classical computer, both with and without. Write something more maybe about exact statevector simulation?idk Have a look at qiskit mentions in articles

Write more about qiskit and statevector and such, and include measurements and smaplings

6.1.1 Types of backends?

6.2 Datasets

6.2.1 Synthetic made fraud classification

Hundreds of million transactions are completed each day around the world, which is why fraudulent transactions are so important to sieve out and eliminate, which also is a great fit for the abilities of machine learning. Even though there are lots and lots of bank transactions each day around the world, collecting a dataset in search of a complete dataset to be used to classify fraud could easily turn to a burdensome venture due to multiple reasons. First and foremost due to ethical reasons regarding stringent privacy of financial data such transaction details are not always available, in addition to labeling such data is often an overtiring process [46]. It is also worth mentioning that fraudulent bank transactions are much less likely to happen, which would give a large imbalance of fraud to non fraud categories. Lucky for us, synthetic made fraud transactions are created to mimic real life bank transactions based on [46], which will serve as a transaction dataset in this thesis. The transaction dataset is published here [47].

The dataset is based on transactions done in the US, including the following features: location based on ZIP code, time of the day the purchase was done, amount of dollars the transaction revolved around and Merchant Category

Code(MCC) which is a way credit card companies defines what kind of services and goods are sold at different businesses.

When classifying the fraud dataset using the VarQBM, only one visible qubit is necessary due to transactions being either fraud or not fraud. Therefore the VarQBM will be applied to the fraud dataset using one hidden and one visible nodes in addition to the two ancilla qubits.

Preprocessing the transaction dataset

Since preprocessing of data is such an important aspect of machine learning some preprocessing will naturally be applied to the transaction data as well. Firstly the features are discretized into certain bins in resemblance with [10]. The ZIP codes are discretized and relabeled into 0,1 or 2 regarding if the ZIP code belongs to the east, central or west of the US accordingly. The time of purchase were also discretized in groups of transactions completed between, 0AM and 11AM, 11AM and 6PM and 6PM and 0AM corresponding to labels of 0,1 and 2 accordingly. The same were done for amounts less than \$50, between \$50 and 150, and more than \$150, into bins labeled 0,1 and 2 accordingly. The remaining discretization were applied to the MCCs. By studying MCCs with values less than 10.000, opens up the door of splitting the MCC feature into 10 more features containing 0's and a 1, according to the thousandth of the MCC.

Maybe use a table instead to show the discretization

The datasets used consists of three sets, a training set consisting of 500 samples. A validation set, consisting of 250 samples which will be used to explore the optimal amount of training epochs needed to minimize the validation loss and prevent overfitting and underfitting. And a test set to see how well the final model generalizes. Each of the three sets contains 20% fraudulent credit card transactions, which is more in terms with real world transactions, due to most transactions in real life being non-fraudulent. The data is also scaled by applying standardization to the features to ensure a centered mean and unit-variance.

6.2.2 Handwritten digits

The next datasets used in the thesis were based on classifying handwritten digits based on images or pixel strengths in other words. Both a smaller digit dataset made by [48] containing approximately 180 samples of each digit collected from a total of 43 people with sample sizes of 8 x 8 pixels, the dataset will be referred to as the 'digit dataset' in the thesis. The other dataset based on handwritten digits used is the more advanced and famous MNIST dataset[49] which is one of the most popular datasets within machine learning and a remarkable way of testing new machine learning models. The MNIST dataset consists of more than 60.000 training samples in addition to a 10.000 sized test set. The size of the samples are 28 x 28 pixels. This dataset will be referred to as the 'MNIST dataset' in the thesis.

To keep the computational time and number of circuits in addition to run enough epochs to train a complete model only four classes will be used. Using only samples from four classes makes it possible to classify the four numbers of the digits datasets using only 2 visible qubits and 2 ancilla qubits.

Preprocessing of handwritten digit datasets

Due to the limited number of samples, in the digit dataset, the sets used will only consist of a train set and a validation set-which will behave as a test set. Following the trend from the transaction dataset, the training sets for both handwritten datasets will use training sets containing 500 samples. The digit dataset will use the rest of available data as a validation set, while the MNIST dataset will use a validation- and a test set of 250 samples each.

Scaling the data is always a clever idea, and due to pixel values in images having values in the range of $[0, 255]$, it is quiet typical to scale images using min-max normalisation to ensure feature values between in the range of $[0, 1]$, which will be done on both handwritten datasets. Further more the pixels will

6.2.3 Franke's function

Franke's function is often used as a function to test different regression methods. Franke's function is a three dimensional function which can be seen in figure Figure 6.1. Franke's function has two Gaussian peaks of differing heights, and a smaller dip. The formula of the function goes as follows:

$$\begin{aligned} f(x,y) = \frac{3}{4}\exp\Big\{\Big(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\Big)\Big\} + \frac{3}{4}\exp\Big\{\Big(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\Big)\Big\} \\ + \frac{1}{2}\exp\Big\{\Big(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\Big)\Big\} - \frac{1}{5}\exp\big\{\big(-(9x-4)^2 - (9y-7)^2\big)\big\} \end{aligned}$$

Using the Vandermonde matrix to sort data from Franke's function

Keeping data in a well designed system for easy access and application is important in all sorts of supervised learning tasks. Due to sampling probabilities of qubits in quantum computations being quiet unstable leaving lots of room for deviations, Franke's data are sorted into a so called Vandermonde matrix containing geometrical progression of the data, and used as a design matrix. The design matrix is constructed by creating each sample as a polynomial of some x and y point extracted from Franke's function the following way:

$$\mathbf{X} = \begin{pmatrix} 1 & x_0 & y_0 & x_0^2 & x_0y_0 & y_0^2 & \dots & x_0^d y_0^d \\ 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & \dots & x_1^d y_1^d \\ & & & & & & \vdots & \\ 1 & x_n & y_n & x_n^2 & x_ny_n & y_n^2 & \dots & x_n^d y_n^p \end{pmatrix}$$

With d being the degree of polynomial. Making the degree higher leads to more complicated equations which naturally gives a more precise fit to the training data, and therefore increasing the possibility of overfitting. It is worth

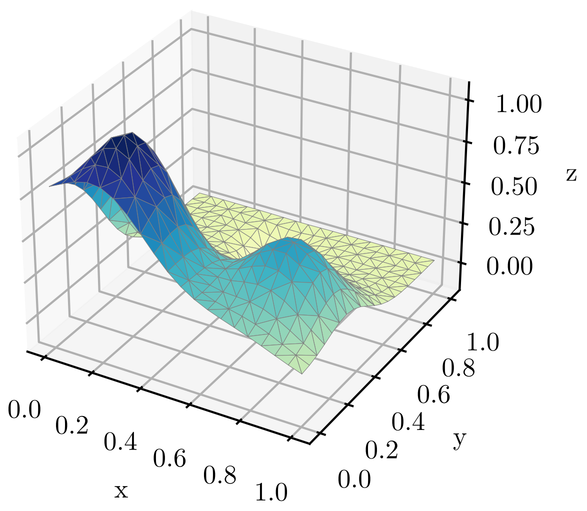


fig: frankesplot

Figure 6.1: Shape of Franke's function which will be used as a regression goal.

mentioning that by using a polynomial Vandermonde matrix will lead to some correlation between the samples, but since the goal of using the mentioned dataset is to see how well the VarQBM are able to manipulate sampling probabilities in regression and study how well this can be generalized, a blind eye will be turned towards the correlating part of the samples.

Preprocessing and generation of the Franke data

When generating the dataset some stochastic noise $\epsilon \sim \mathcal{N}(0, \sigma)$ are also added to Franke's function as an effort to lessen the chance of overfitting. The data is generated by inserting equally spaced points as $x, y \in [0, 1]$ together with the desired degree of polynomial into the aforementioned design matrix which is created and filled with values from Franke's function.

The values are already quiet nicely lined up due to $x, y \in [0, 1]$, but to ensure that the target variable also lays between 0 and 1 min-max normalisation is applied to Franke's data. This is crucial to have all target variables lay in the span of possible sampling probabilities. The data used consists of $20x20 = 400$ samples which is divided into a train- and validation/test with a ration of 0.75 and 0.25 accordingly. The chosen degree of polynomial is 5 to keep the correlations and overfitting within a reasonable level, which gives 21 features.

6.3 Variational Quantum Boltzmann Machine

Explains a general outline on how the VarQBM is initialised, and explain different aspects of it under

The VarQBM primarily consists of three steps. The initialisation and optimization of Hamiltonian coefficients in addition to preparing a trial circuit which yields Bell states when subtracing pairwise qubits of the trial circuit. Generating and preparing Gibbs states, and finally computing the Boltzmann distribution and derivatives which is nedded for the optimization process.

After the Hamiltonian and the trial circuit are ready to go, the circuits are sent into the varITE class, which prepares the Gibbs states of the circuits, and the corresponding derivatives. The Gibbs states are then used to compute the Boltzmann distribution along with the gradient of the Boltzmann distribution with respect to the Hamiltonian parameters, using the chain rule along with the derivatives of the Gibbs states. The gradients are then used to optimize the coefficients of the Hamiltonian according to some optimization technique to reproduce some specific target distribution. This naturally goes on for multiple loops, until the optimal parameters are found.

6.3.1 Initialisation of Hamiltonian coefficients

When doing generative learning, generating target probability distributions, the Hamiltonian coefficients are drawn from an uniform distribution from the interval of $[-1, 1]$ in accordance with [10].

When doing discriminative learning, the input of samples must to be encoded into the VarQBM. This is done using two approaches, first approach is done by following the bias dot product method explained in **??**, the Hamiltonian vector which is multiplied by feature values are drawn from a uniform distribution the same way as for generative learning.

Add the referral above

The second approach is by having feature values of a given sample encoded into the coefficients by inserting them into a DNN, which goes through the network and outputs the Hamiltonian coefficient.

Neural network to initialise the Hamiltonian

When initiating a DNN there are multiple parameters that affects the initialisation and thereby the output. Firstly the weights and bias of the network need to be initialised, there are lots of different ways to do this, but in this thesis Xavier normal- and uniform and He normal- and uniform will be explored.

In addition activation functions are most often favoured to include in a neural network. Some different activation functions are tested to observe the rate of convergence toward some minimum. An overview of the activation functions used in this thesis can be seen in Table 6.1

Another thing is probably is one of the most important things to define in a neural network is the number of layers and hidden nodes within each layers. Therefore multiple sizes of layers will be tested, in addition to putting the very rough general rule of thumb called the pyramid rule provided in [50, ch. 10] to test. The pyramid rule is a quiet shallow direction of choosing the size of

Table 6.1: A variety of activation functions used common in neural networks

Name	Activation Function
Identity	$f(x) = x$
Sigmoid	$f(x) = \frac{1}{(1+e^{-x})}$
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Leaky Relu	$f(x) = \max(0.01x, x)$

tab: activaton_ functions

neural networks consisting of one or two layers. The rule is based on creating a pyramid shaped network, in other words having the input layer stand as the largest layer in the network, followed by a smaller hidden layer, which goes on until the output layer consisting of the fewest nodes are reached.

The geometric progression of hidden nodes in a 1- hidden layer network consists of $\sqrt{mn}$, with n being the number of input neurons and m corresponding to the output neurons.

A 2-hidden layer network would have a quiet similar approach by defining a variable $r = \sqrt[3]{\frac{n}{m}}$, which gives the following hidden layer sizes:

$$\begin{aligned} NHid_1 &= mr^2 \\ NHid_2 &= mr \end{aligned}$$

Eventhough the pyramid rule only should be used as a guideline of the network sizes, this should be sufficient in this particular case due to the network only being used to find Hamiltonian coefficients used in the VarQBM, the heavy lifting is still executed by the VarQBM, which gives room for the neural network not necessarily using the most optimal parameters.

6.3.2 Variational imaginary time evolution

The preparation of Gibbs states through VarITE is one of the main parts of the VarQBM. After inserting the trial circuit and the Hamiltonian into the VarQITE class, all circuits needed are created before the time evolution starts. Since the parameters evolved through varITE are the parameters of the trial circuit, all circuits can be initialized before the time evolution starts, and bind new parameters to the gates after each timestep.

The initial trial circuit $V(\omega(0))$ are sent into the VarITE process with the objective of computing $\omega(\tau)$ and the corresponding derivative $\frac{\partial\omega(\tau)}{\partial\theta}$ with τ being the final time. When the time evolution starts off, matrix $A(t)$ and vector $C(t)$ are computed according to section 5.4 at time step t. Which are then put together as a system of equations $A(t)\dot{\omega} = C(t)$, which results in the derivative of the trial circuit parameters with respect to current time step. Due to matrix $A(t)$ having a possibility of being singular, hence having infinite many solutions, non-invertible matrices are more probable to make an appearance when the trial circuit consists of an excessive amount of symmetry. Dealing with ill-conditioned systems of equations are done by applying Ridge or Lasso regression to the system. The regularizing term should be small enough to ensure that matrix A is invertible or tuned to give the least amount of difference between $A\dot{\omega}$ and $C(t)$.

Further on a loop goes through each of the Hamiltonian parameters, computing the derivative of $\omega(t)$ with respect to the Hamiltonian coefficients. This is done using regularization schemes as done earlier in the VarITE process. After each time step ω and $\frac{\partial \omega}{\partial \theta}$ are updated using explicit Euler methods. An outline of the algorithm from [10] can be seen in 1

algo: VarQITE

Algorithm 1 VarQITE for

```
input
$H_{\text{eff}} = H_\theta^a + I^b$
$\tau = 1/2\,(\mathrm{k_B T})$
$|\psi(\omega(0))\rangle = V(\omega(0))\,|0\rangle^{\otimes 2n} = |\phi^+\rangle^{\otimes n}$
with $|\phi^+\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$
procedure
for $t \in \{\delta\tau, 2\delta\tau, \ldots, \tau\}$ do
    Evaluate $A(t)$ and $C(t)$
    Solve $A\dot{\omega}(t) = C$
    for $i \in \{0, \ldots, p-1\}$ do
        Evaluate $\partial_{\theta_i} C$ and $\partial_{\theta_i} A$
        Solve $A(\partial_{\theta_i}\dot{\omega}(t)) = \partial_{\theta_i} C - (\partial_{\theta_i} A)\,\dot{\omega}(t)$
        Compute $\partial_{\theta_i}\omega(t+\delta\tau) = \partial_{\theta_i}\omega(t) + \partial_{\theta_i}\dot{\omega}(t)\,\delta\tau$
    end for
    Compute $\omega(t+\delta\tau) = \omega(t) + \dot{\omega}(t)\,\delta\tau$
end for
return $\omega(\tau)$, $\partial\omega(\tau)/\partial\theta$
```

item: linEqDer

item: grad_omega_tau

item: computeUpdateDer

Computing- and interpreting the Boltzmann distribution to classification- and regression values

After finding the Gibbs states from the VarITE process, tracing out the ancillary system, the Boltzmann distribution are then found by doing a projective measurement on the quantum states representing the visible qubits for each possible configuration of the visible qubits, which would translate to the diagonal of the matrix corresponding to the visible qubits.

Translating the Boltzmann distribution into classification values are quiet easily done by. When doing multiclass labeling, the Boltzmann distribution is used as a one hot encoded vector, meaning that the each index of the distribution represents one label, giving the index of largest probability the final label. In multilabel classification the needed number of visible qubits n scales exponential with the number of classes 2^n. When doing binary classification, only one visible qubit is needed, interpreting the probability of the qubit being in state 0 or 1 as the probability of the binary label being 0 or 1. Regression problems follow the same lane, using the sampling probability as the regression values, ensuring that the true values are scaled within 0 and 1.

6.3.3 Optimization process

The optimization process of the VarQBM is conducted in a classical fashion without the use of quantum algorithms. The optimization process is quiet usual compared to most optimization problems. The loss function used throughout the thesis is primarily cross entropy, with an exception of MSE used with regression of Franke's function. Due to the sampling probability being a regression target instead of a probability distribution.

After a prediction is done within the VarQBM, the gradients of the probabilities corresponding to each configuration is computed using the parameter shift rule introduced in section **??**. The derivatives are taken further to compute the gradients of the loss with respect to each Hamiltonian parameter. Now all the necessary gradients are computed and are used to optimize the Hamiltonian according to some optimizer. Multiple optimizers will be tested to evaluate how the choice of optimization technique affects the Boltzmann distribution. Due to the relatively high number of circuits simulated when doing an iteration of VarITE, some form of SGD optimization is executed

Add referral to the parameter shift rule, is it even necessary to refer to the theory section?

sec: hamiltonian_ ansatzes

6.3.4 VarQBM ansatzes

The VarQBM consists of two ansatzes: The Hamiltonian ansatz which will contain the learnable parameters and the trial circuit. Primarily in this thesis, mainly two sets of ansatzes will be used unless else is stated: a 1-qubit Hamiltonian paired with a 2-qubit trial circuit, and a 2-qubit Hamiltonian paired with a 4-qubit trial circuit. Half of the qubits in both trial circuits acts as ancilla qubits. The Hamiltonian ansatzes used will be referred to as H_1 and H_2 throughout the thesis unless else is stated:

$$
\begin{aligned}
H_1 =& \theta_0 Z \\
H_2 =& \theta_0 ZZ + \theta_1 IZ + \theta_2 ZI
\end{aligned}
$$

With the first and second Pauli gate referring to the first and second qubit respectively. The trial circuits used can be seen in Figure 6.2

Make these circuits by drawing them?

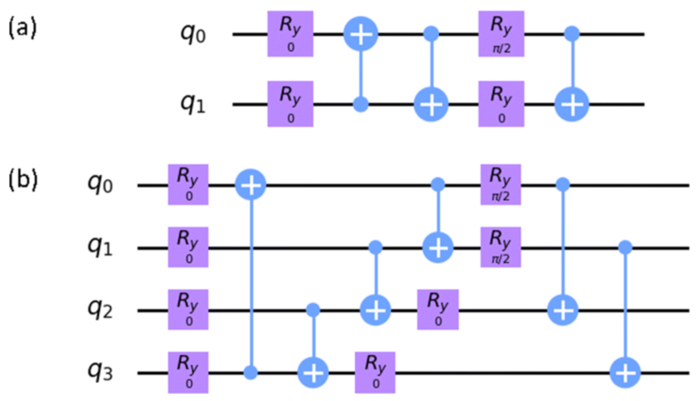


Figure 6.2: The trial circuits mainly used in the thesis. (a): Ansats used in combination with H_1. (b): Ansats used in combination with H_2. Figure from [10]

fig: ansatzes_ trialcirc

6.4 Restricted Boltzmann Machine

When referring to a classical Boltzmann machine or a RBM in this thesis, a classical computational approach is taken without the use of quantum algorithm. Due to problems of convergence well known when using regular Boltzmann machines, the classical approaches of Boltzmann machines in this thesis is therefore done utilizing RBMs. Two classical RBMs are utilized in this thesis, a Bernouilli-RBM and a Gaussian-binary RBM.

6.4.1 Scikit learn's Bernoulli-RBM

A Bernoulli-RBM provided by the widely used python library Scikit Learn[51] is used when doing classification- and regression tasks on classical datasets. A Bernoulli-RBM originally only accepts binary nodes for both visible and hidden nodes, but by inserting real valued visible values in the range of $[0, 1]$ instead, the visible states are treated as visible probabilities instead of binary states, the scikit learn class automatically initialises the variance as 0.01 prior to the training. Therefore data applied to accepting real valued input, therefore the data are scaled in the

6.4.2 Gaussian-Binary RBM

When utilizing a classical RBM to solve a quantum mechanical problem, a self written Gaussian-Binary RBM using a Variational Monte Carlo approach is implemented using C++.

6.4.3 Modelling the Hydrogen system

The hydrogen system consists of two hydrogen atoms at fixed positions with distance R between them. The modell can be seen in Figure 6.3

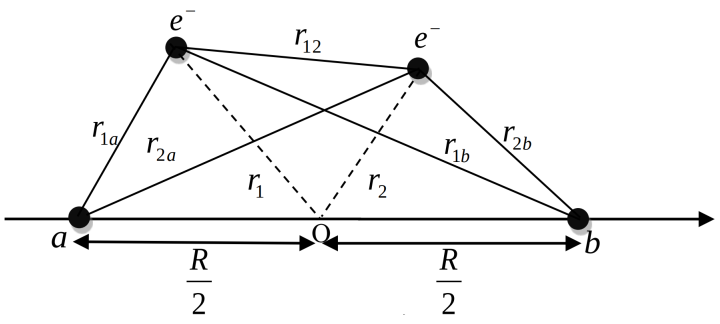


Figure 6.3: H2 model. Figure from [52]

fig: h2model

Implementation of the Gaussian RBM

Method of RBM from fys4411, write after it is used

PART V

Results and Discussion

CHAPTER 7

Classical data

sec: seventh

7.1 Preparation of Gibbs States

The fidelity between the analytical Gibbs state and the approximated Gibbs state through VarITE were computed to assess the Gibbs state preparations. The Hamiltonians used in this section regarding the fidelity will be the following, before going back to referring to H_1 and H_2 as the Hamiltonian ansatzes from subsection 6.3.4

$$
\begin{aligned}
H_1 &= 1.0Z, \\
H_2 &= 1.0ZZ - 0.2ZI - 0.2IZ + 0.3XI + 0.3IX.
\end{aligned}
$$

The fidelity was computed as a function of the regularization parameter λ of the least square methods Ridge and Lasso, finding $\dot{\omega}$ using different Hamiltonians. The results can be seen in Figure 7.1. Ridge regression were chosen as the regularization method in the further Gibbs state preparations. The regularization value were chosen by performing Ridge regression using logarithmic increments $\lambda \in \log_{10}[-10, -2]$, computing the loss using multiple λ and settling on the one giving the least loss each time step can be seen in Figure 7.2.

The Gibbs state results is summarized in **??**.

The final fidelity after 10 steps using a dynamical λ resulted in fidelities of 0.98 and 1.0 using H_1 and H_2 respectively.

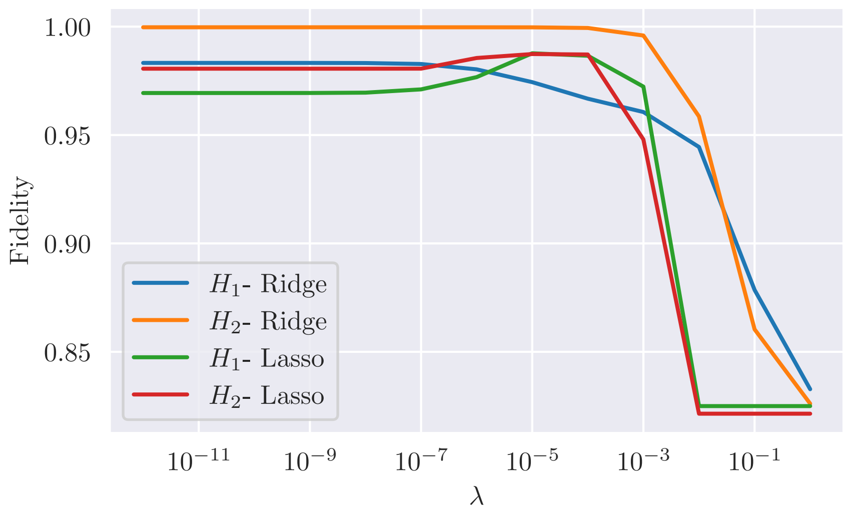


Figure 7.1: Fidelity as function of lambda for Ridge and Lasso regression using Hamiltonian H_1 and H_2. The number of imaginary time steps for the preparation were 10.

fig: fidelity_ vs_lmb

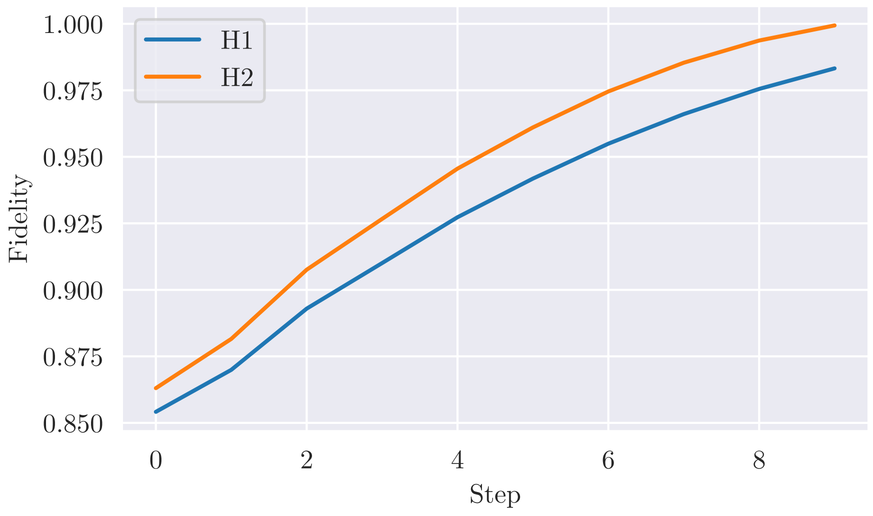


Figure 7.2: Fidelity as a function of imaginary time step using Ridge regression, varying the regularization parameter each step, choosing the parameter giving the numerical $\dot{\omega}$ closest to the analytical $\dot{\omega}$ without getting singular matrices.

fig: fidelity_ lambda_ log_ increament

7.1.1 Preparation of Gibbs states- Discussion

Having a look at Figure 7.1 it shows that the highest fidelity between the analytical and approximated Gibbs state were received by H_2 using Ridge regression. For the H_1 Hamiltonian on the other hand, the highest fidelity were achieved by using Lasso regression for a small portion of $\lambda \in \log_{10}[-6, -3]$.

Since it seems that Ridge outperforms Lasso for small values of λ, Ridge where chosen as the regression method for the rest of the calculations. Another reason for choosing Ridge regression due to small values of λ is that the smaller regularization, the closer the numerical $\dot{\omega}$ will be to the $\dot{\omega}$, as long as singular values are avoided. This is a different approach to how one normally would use Ridge regression in a supervised learning approach, where the regularisation value is used to prevent overfitting, while in this case, overfitting is desireable.

Figure 7.2 shows that having a dynamically changeable λ during the evolution of Gibbs states provides approximated Gibbs states corresponding to fidelities close to 1 for both H_1 and H_2. H_1 and H_2 resulted in fidelities of 0.98 and 1.0 respectively. The reason for H_2 being able to evolve into a more accurate Gibbs state is probably due to the four-qubit trial circuit being a better fit compared to the two-qubit trial circuit used in accordance to H_1. Fortunately due to the VarQBM being used for machine learning purposes the approximated Gibbs states do not require a 100% match with the analytical states[10], and will therefore work as noise in the dataset when the states are deviated by a small amount.

Using a set λ seems to evolve the states approximately into the same Gibbs states as for the dynamically changeable λ, as long as λ lays within a reasonable range. The dynamically changeable λ will be used for further computations. The reason for this is that due to a dynamically changeable λ finds the best regularisation each step, hence generalize easier to other systems.

7.2 Generative Learning

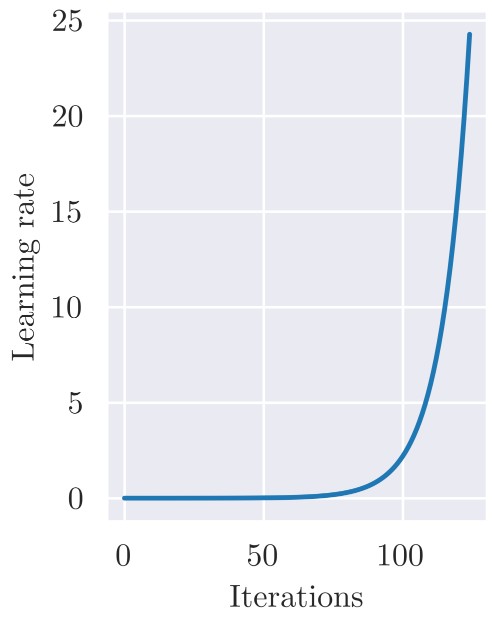
The VarQBM was used to generate probability distributions. The chosen learning parameters and final results will be revealed in this section.

7.2.1 Parameter investigation

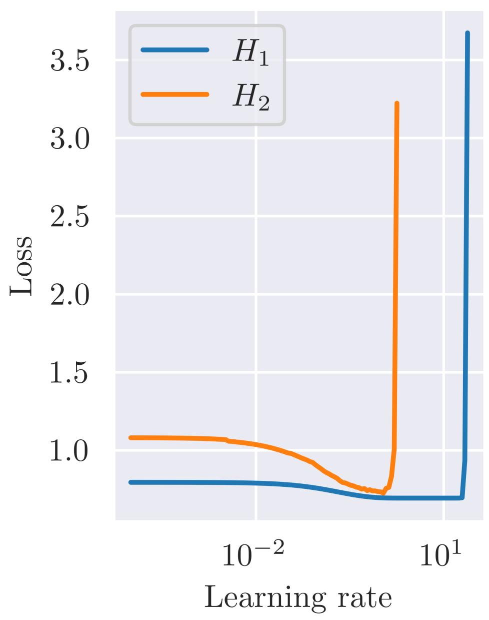
Before the distributions were computed, some learning parameters had to be chosen. Firstly the learning rate were chosen. To get a reasonable estimate of the initial learning rate a method motivated by [53] were used. The learning rate was increased exponentially after each epoch, resulting in an estimated learning rate of approximately 0.2 for both Hamiltonians H_1 and H_2. The plot of the loss in addition to the increment learning rate can be seen in 7.10.

Figure 7.4 and Figure 7.5 shows how the learning rates and optimization technique applies to a 1-qubit- and 2-qubit Hamiltonian respectively.

Plot again using a log scale



(a)



(b)

Figure 7.3: Learning rate investigation using SGD as the optimization technique. a) Increment of the learning rate as function of iterations applied to figure b). b) Loss as a function of increased learning rate

fig: lr_exp

Plot again using a log scale

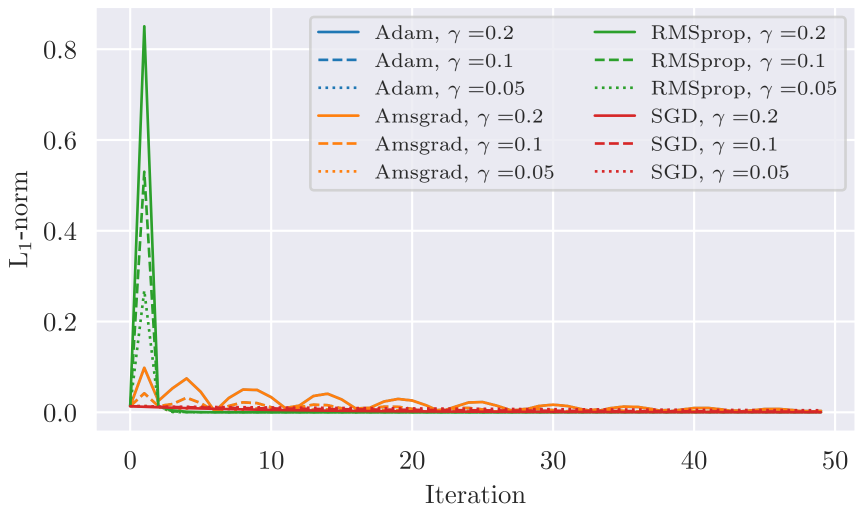


fig:H1_loss_es

Figure 7.4: Norm as function of iterations. Generating a probability distribution [0.5, 0.5] using a 1-qubit Hamiltonian and a variety of optimization methods and learning rates. The norm is used instead of loss to perceive the trend easier. The norm is computed between the target distribution and the generated distribution. The ADAM optimizer overlaps with the AMSGrad optimizer.

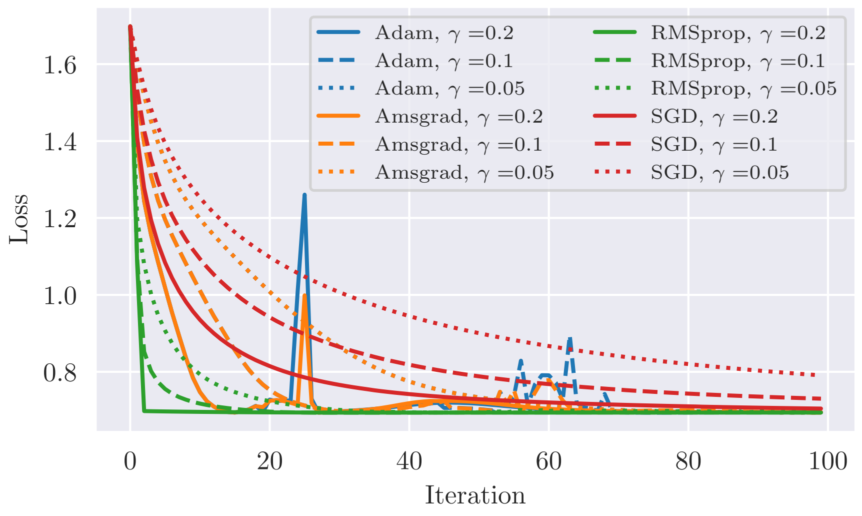


fig:loss_vs_it_vs_lr

Figure 7.5: Loss as function of iterations. Recreating the Bell state using a 2-qubit Hamiltonian and a variety of optimization methods and learning rates.

Parameter investigation - Discussion

Figure 7.10 argues that the optimal learning rate is 0.22, but this is taken as a roughly estimate, mainly due to the fact that the target distribution needless to say is constant during the whole training, in theory making the generated distribution a bit more identical to the target after each iteration, hence making it more difficult for the later iterations to have a large jump in decrement of loss before the learning rate gets to big. Even though this might even out due to the learning rate increasing and the gradient decreasing during training until the blowup of loss due to the enormous learning rate.

The estimation of the learning rate were taken into account and applied to different optimization techniques. Figure 7.4 and Figure 7.5 both argues that the optimal optimization method would be RMSProp due to the fastest converging towards a minimum. The learning rate on the other hand seem to be best being set to 0.2 for the 2-qubit Hamiltonian, and quiet good for a 1-qubit Hamiltonian, but due to a large blowup at the start of the training of the 1-qubit Hamiltonian, a learning rate of 0.1 where chosen.

The momentum parameter of the RMSProp optimization technique were investigated slightly showing that all momentum terms $m \in [0.6, 0.999]$ converged during training. The figure can be seen in Appendix C. The momentum was set to 0.99 for further computations.

7.2.2 Generating probability distributions

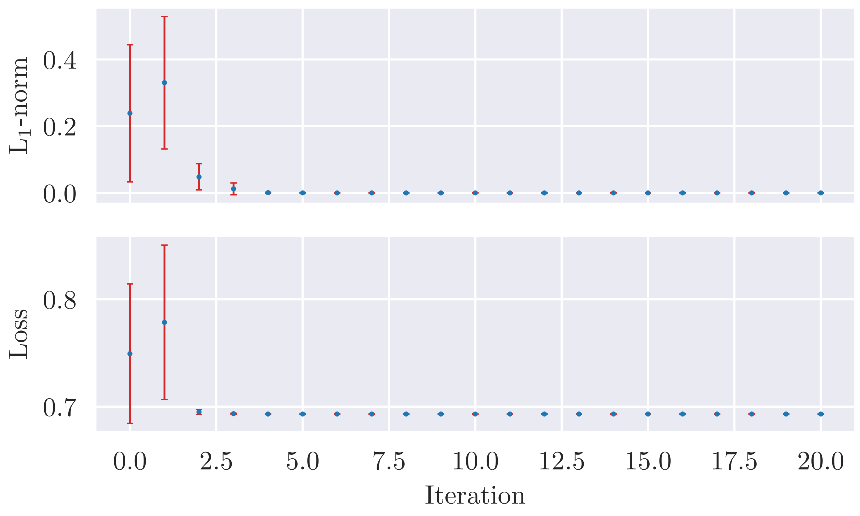


Figure 7.6: Generation of a probability distribution [0.5, 0.5] using a 1 qubit Hamiltonian. The quantum Boltzmann machine is fully visible. Loss and norm as a function og iteration steps. The plot represents the mean of 10 random seeds, and the bars represent the standard deviation.

fig: nolabel

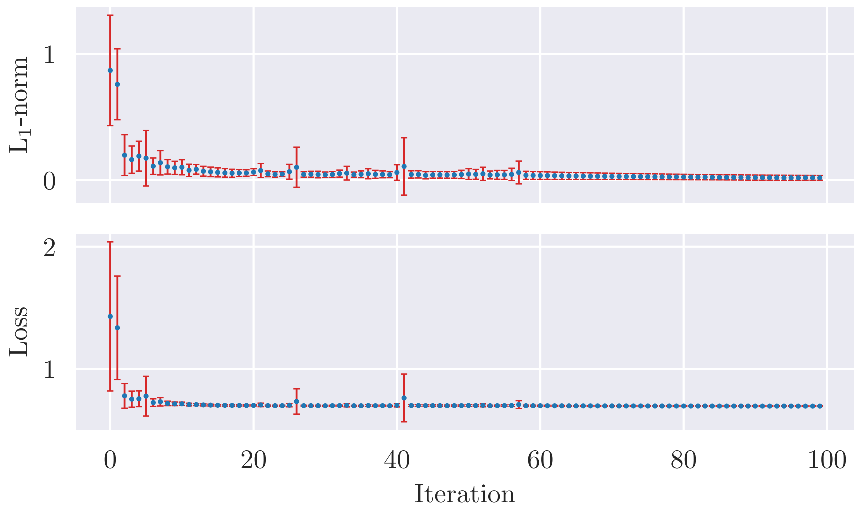


Figure 7.7: Too hard to see? Generation of a 2 qubit Hamiltonian. The quantum Boltzmann machine is fully visible. Loss and norm as a function of iteration steps. The plot represents the mean of 10 random seeds, and the bars represent the standard deviation.

fig: nolabel

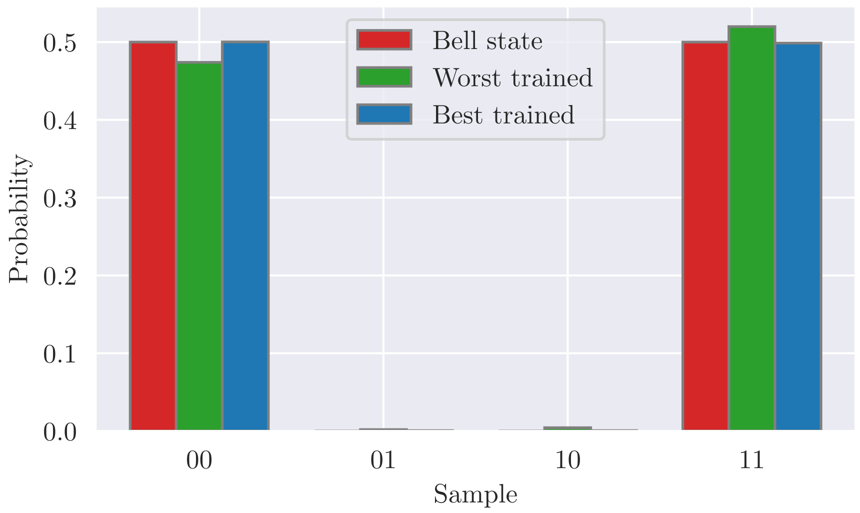


Figure 7.8: Seems good

fig: nolabel

7.3 Discriminative Learning-Transaction Dataset

This section contains results regarding discriminative learning. Where The fraud dataset and 8x8 MNIST dataset were used.

7.3.1 VarQBM: Input as bias

In this section the input of the VarQBM were inserted according to

Insert referal to the bias input section when the section is done

7.3.2 VarQBM: Neural Network feature engineering

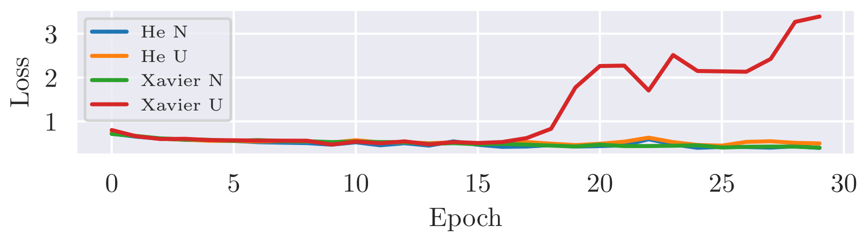
In this section the input of the VarQBM were inserted according to

Insert referal to the neural network input section when the section is done

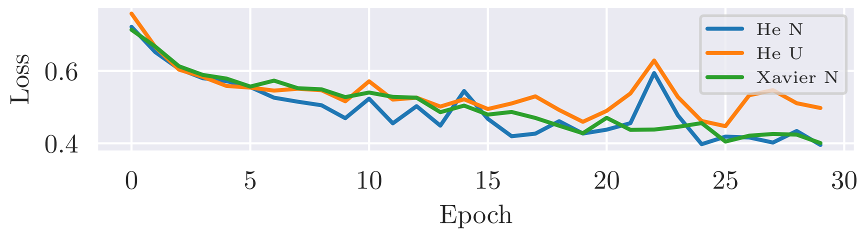
Starts with initialization of weights in ??.

7.3.3 Restricted Boltzmann machine

Solving the transaction data using a restricted Boltzmann machine with logistic regression layer on top.



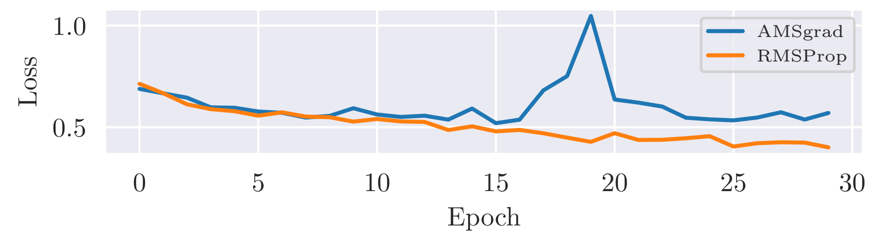
(a)



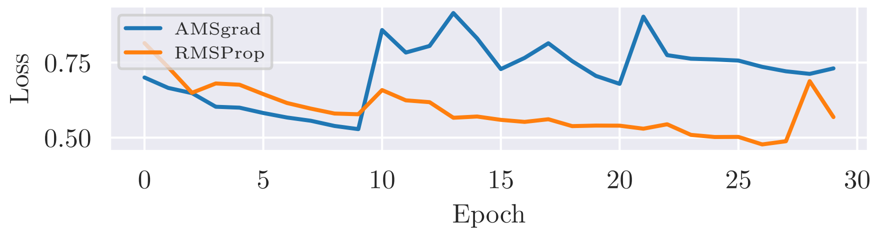
(b)

Figure 7.9: Initialisation of weights. N is normal, U is uniform a) 4 different initialisations b) A better view of He normal, He uniform and Xavier normal

fig: init_X_ and_He



(a)



(b)

Figure 7.10: Tested both H2 and H2.2 with AMSgrad and RMSProp, might remove this, cause it is not that relevant and but we'll what ends up in the thesis eventually.

fig:
lr_exp

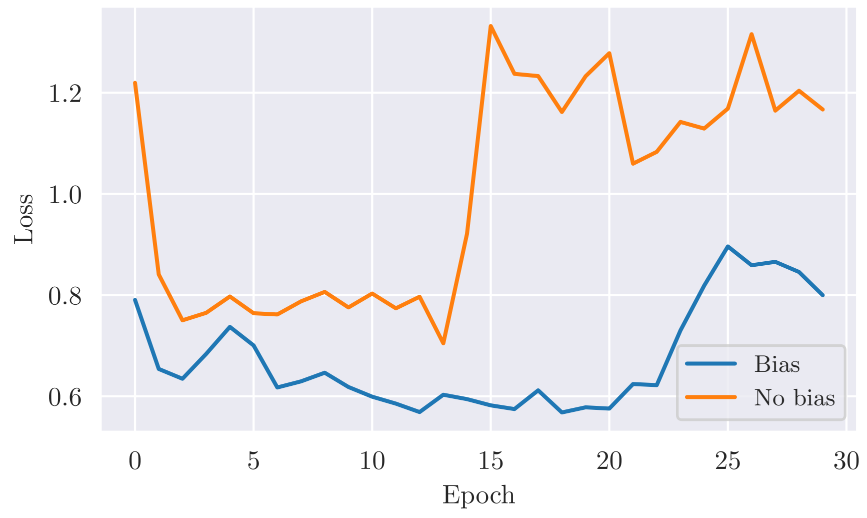


Figure 7.11: 2 hidden layers with 12 hidden nodes each and sigmoid activation functions within the hidden layers, with and without bias initialized as 0.01 using 50 samples. The neural net is initialized using xavier normal initialisation.

fig:4

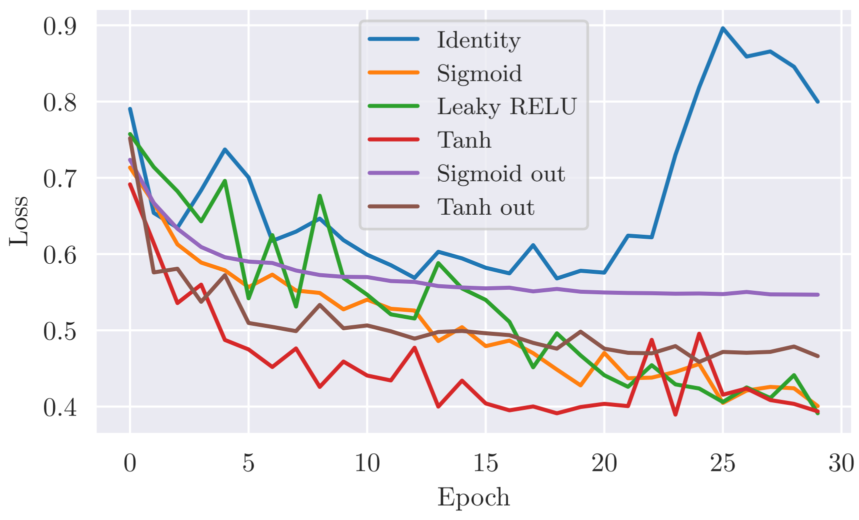


Figure 7.12: 2 qubit Hamiltonian with 1 hidden qubit, using a neural network of 2 hidden layers with 12 hidden nodes and bias, using 50 samples, computed using different activation functions.

fig:3

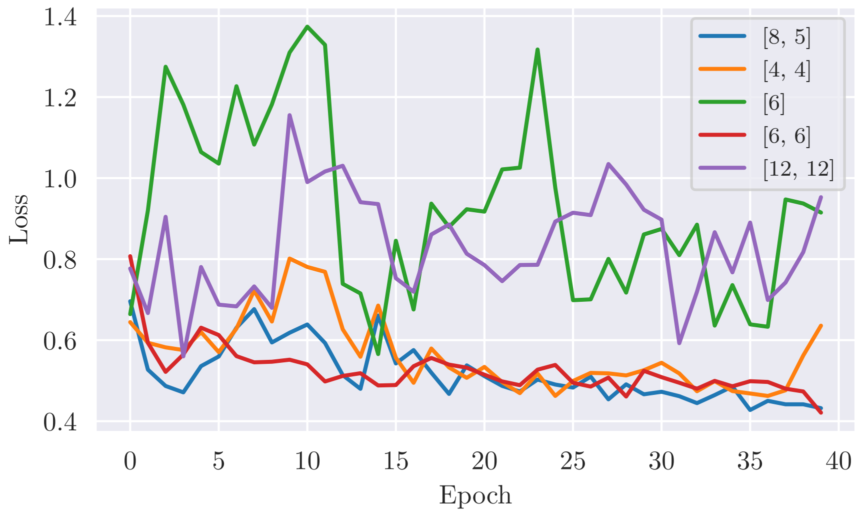


fig:2

Figure 7.13: Layer sizes- Transaction dataset. 100 samples in all plots below

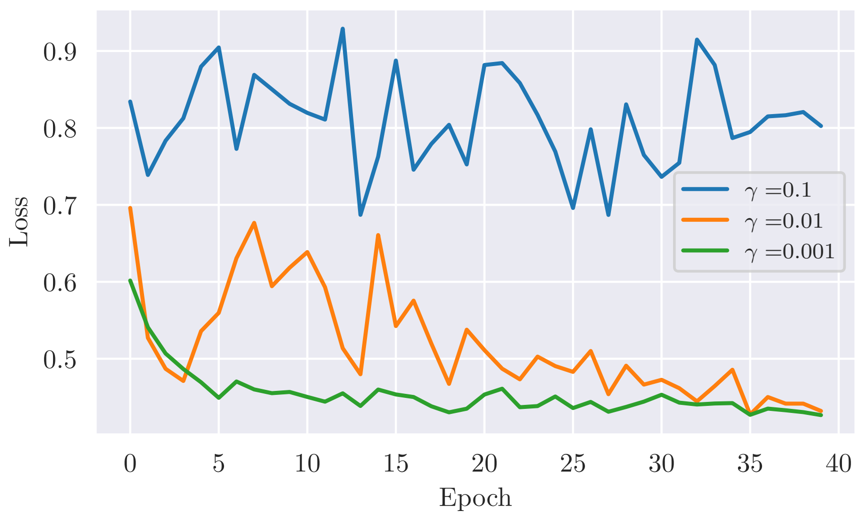


fig:1

Figure 7.14: Learning rate- Transaction dataset

First set the amount of hidden nodes. The accuracy were plotted as a function of hidden nodes using default values of scikit-learn. Here all samples were labeled as non fraud, so 30 hidden nodes were chosen.

Then an exhaustive search over specified parameter values were done searching across the following values:

Learning rate:	0.001, 0.005, 0.01, 0.05, 0.1, 0.5
Batch size	1, 2, 5, 10
Logistic hyperparameter C	0.5, 1, 5, 10, 50, 100, 500

Best parameters found from grid search performed was learning rate of 0.001, batch size of 1 and hyperparameter C 0.5.

The cofusion matrix can be seen in Figure 7.15. Easy to see that the RBM still labels all samples as fraud.

7.3.4 Transaction data scores

The final runs achieved using the classical RBM and the VarQBM with and without neural network feature engineering can be seen in Table 7.1

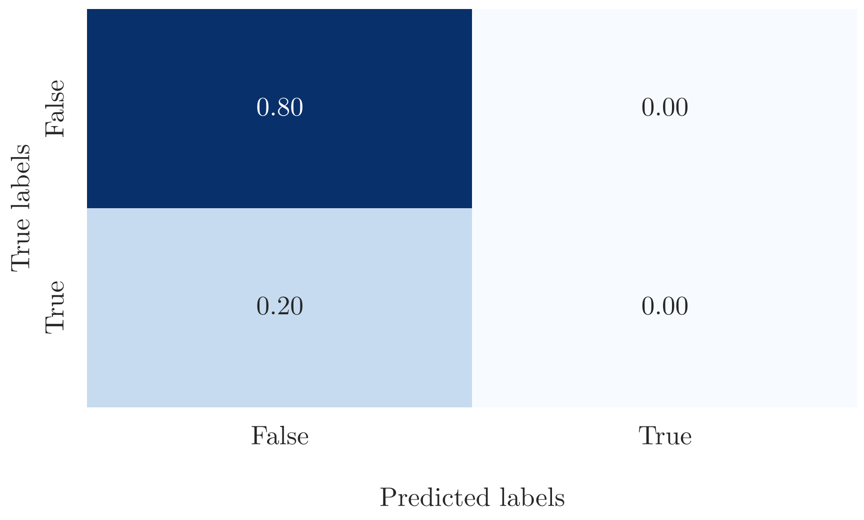


Figure 7.15: RBM- confusion matrix- Transactions. All samples are classified as fraud

fig: transaction_ cm

Model	Accuracy	Precision	Recall	F_1 score
VarQBM bias encoding	0.69	0.76	0.69	0.71
VarQBM NN encoding	**0.82**	**0.81**	**0.82**	**0.78**
RBM	0.80	0.64	0.80	0.71

Table 7.1: Final results, 400 samples, 50 epochs, 0.01

tab: final_ results_ table_ transaction

7.4 Discriminative Learning-Handwritten Image Recognition

Choosing to continue with some of the found parameters from the transaction investigation. Including a bias, if it is better without a bias in the network, it will reduce to 0 during training. In addition the Xavier N initialisation and tanh as activation function within the network are used.

7.4.1 VarQBM: Input as bias

Run computations when input is inserted the normal way

7.4.2 VarQBM: Neural Network feature engineering

7.4.3 Restricted Boltzmann machine

Solving the digit dataset using a restricted Boltzmann machine with logistic regression layer on top.

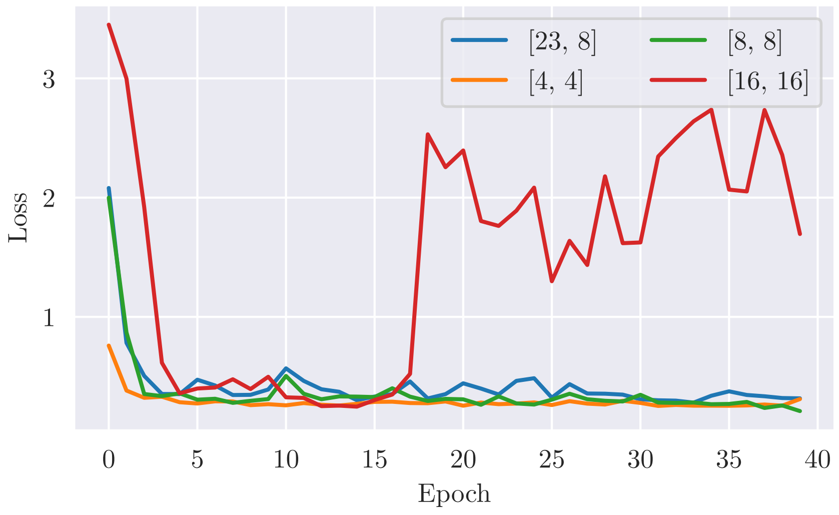


Figure 7.16: Layer sizes- Digit dataset

fig: loss_ epoch_ layers

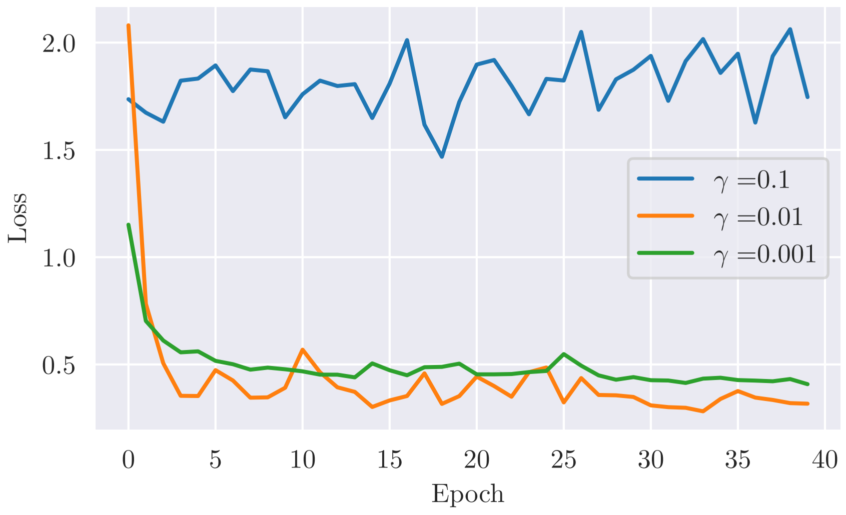


Figure 7.17: Learning rate- Digit dataset

fig: loss_ep_ gamma

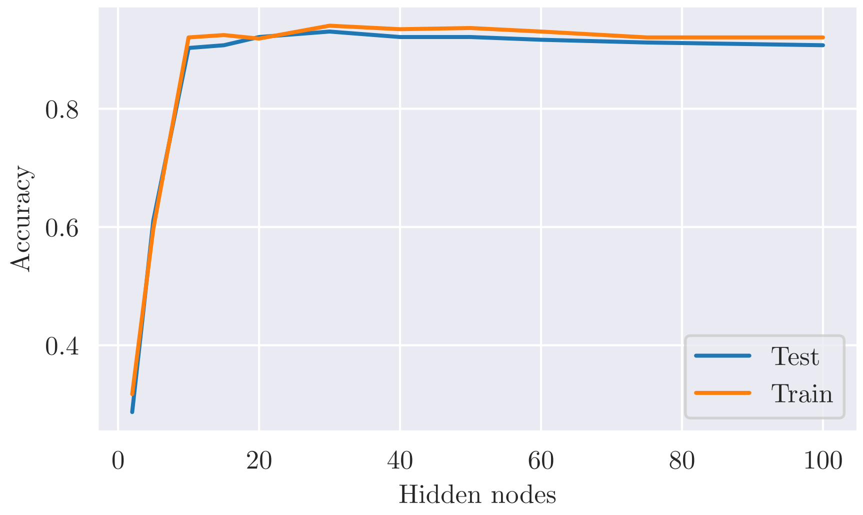


Figure 7.18: RBM - Accuracy as a function of the number of hidden nodes-Transactions

fig:hn_vs_acc

First set the amount of hidden nodes. The accuracy were plotted as a function of hidden nodes using default values of scikit-learn. The results cn be seen in figure Figure 7.18. The choosen amount of hidden nodes was 30.

Then an exhaustive search over specified parameter values were done searching across the same values as for the transaction dataset in

Add referral to the table of gridsearch parameters

Best parameters found from grid search performed was learning rate of 0.05, batch size of 2 and hyperparameter C of 100.

The cofusion matrix can be seen in Figure 7.19.

RBM with 0.99 accuracy on digit dataset(Only four classes.

7.4.4 Handwritten image recognition scores

The final runs achieved using the classical RBM and the VarQBM with and without neural network feature engineering can be seen in Table 7.2

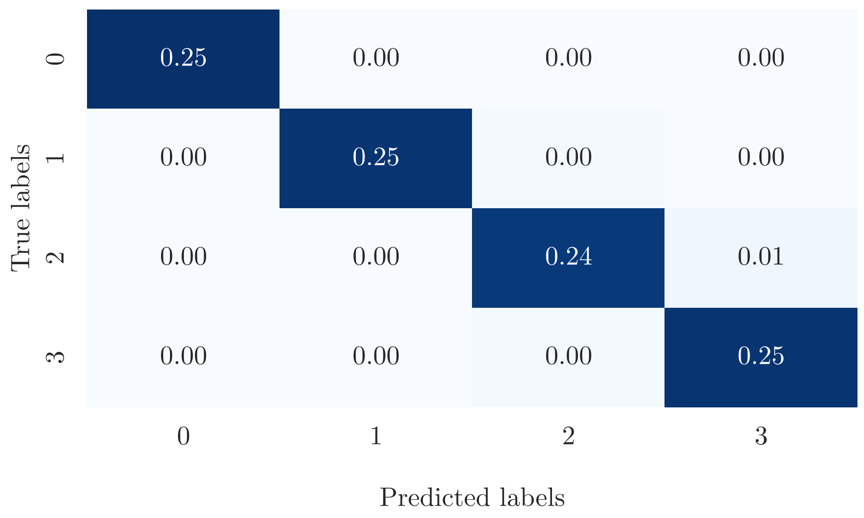


fig: digit_cm

Figure 7.19: RBM- confusion matrix- Digit dataset.

Table 7.2: Final results, 400 samples, 50 epochs, 0.1lr. weighted precision, recall and F1 score, NN layer: [8,5]

Model	Accuracy	Precision	Recall	F_1 score
VarQBM bias encoding	0.56	0.54	0.56	0.53
VarQBM NN encoding	0.73	0.85	0.73	0.65
RBM	**0.98**	**0.98**	**0.98**	**0.98**

tab: final_ results_ table_ digits

7.5 MNIST Scores

The final runs achieved using the classical RBM and the VarQBM with neural network feature engineering can be seen in Table 7.3

Table 7.3: 28x28 pixels, 4 classes. Final results, 400 samples, 50 epochs, 0.01lr. weighted precision, recall and F1 score

Model	NN encoding	Accuracy	Precision	Recall	F_1 score
VarQBM	[32,32]	0.83	0.86	0.83	0.84
VarQBM	[123,19]	0.33	0.11	0.33	0.16
RBM		**0.91**	**0.91**	**0.91**	**0.91**

tab: final_ results_ table_ mnist

CHAPTER 8

Quantum Systems

sec: sevenfive

8.1 Hydrogen Molecule

8.1.1 Finding the ground state energy using VarITE

The Hydrogen Hamiltonian of the Hydrogen molecule transformed using the Bravely-Kitaev transformation and a bond length of $R = 0.75$ [27] goes as follows:

$$H = 0.2252I + 0.3435Z_0 - 0.4347Z_1 + 0.5716Z_0Z_1 + 0.0910Y_0Y_1 + 0.0910X_0X_1$$

The ansatz can be seen in Figure 8.1

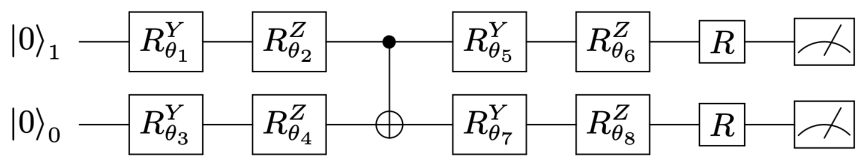


Figure 8.1: Ansatz used to find the ground state energy. The R gates represent the rotational gates depending on the basis of measurement

fig:H2_ ansatz

Finding the grounds state energy of a hydrogen molecule using VarITE and different initialisations of the ansatz parameters can been seen in Figure 8.2.

Using a distribution of $\mathbb{N}(0, \sigma = 0.1)$, the ground state energy were found using different step sizes. The figure can be seen in Figure 8.3

8.1.2 Finding the ground state energy using RBM

The Restricted Boltzmann machine where used with neural-network quantum states was utilized to find the ground state energy of the H_2 molecule with a bond length of 1.40 Bohr radii. Doing a parameter search of the fixed variance σ of the NQS can be seen in figure Figure 8.4

The chosen value of σ was set to 0.9 for future computations. The learning rate γ were seen to affect the results fairly little with $\gamma \in [10^{-3}, 10^{-1}]$, therefore

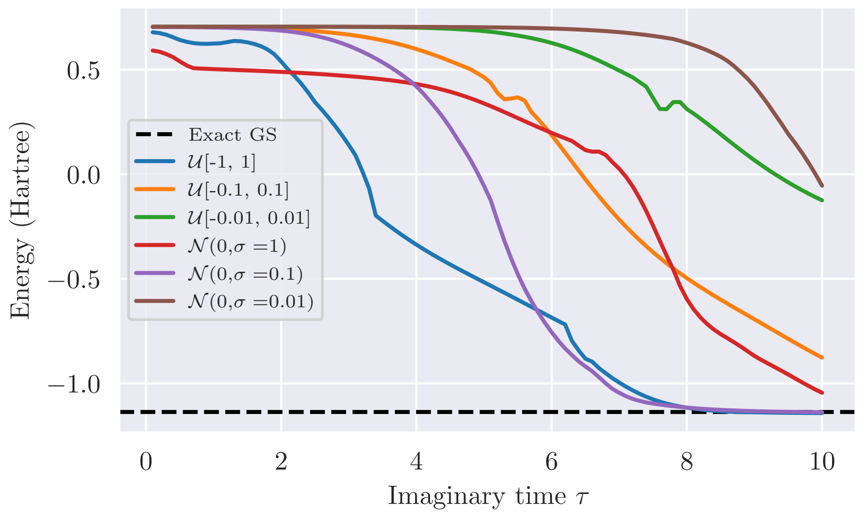


fig:H2_distribution

Figure 8.2: Energy as a function of imaginary time using different distributions

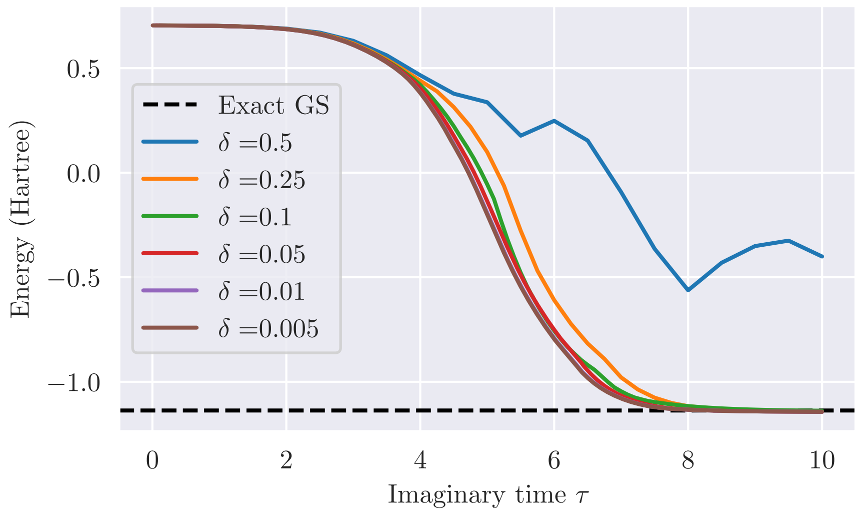


Figure 8.3: Energy as a function of Energy as a function of imaginary time for different sizes of time steps.

fig:H2_step

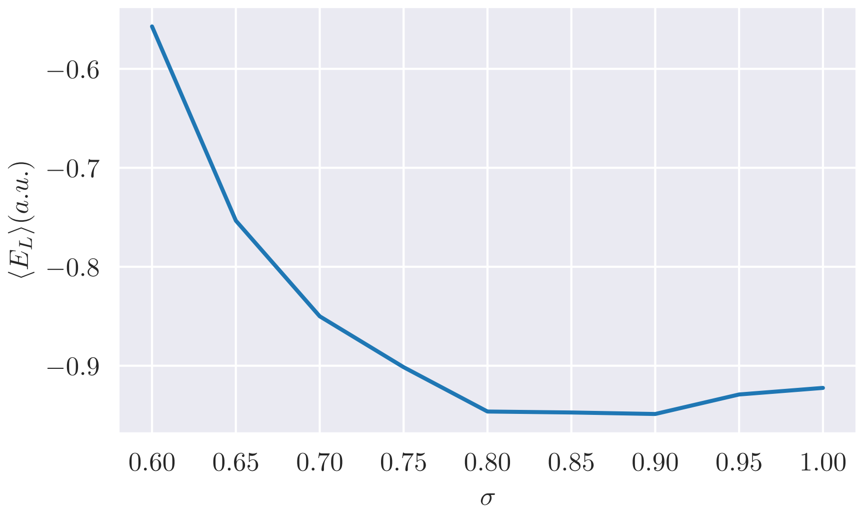


fig: sigmarbm

Figure 8.4: Energy as function of σ

the learning rate was set to 0.01. The number of hidden nodes were then investigated for a variety of hidden nodes. The final results can be seen in Table 8.1

Further on the

Hidden Nodes	E$_C(a.u)$	μ_C	E$_{ref}(a.u)$	Dissimilarity
2	-0.952	$2.9{\cdot}10^{-3}$	1.1746	18.94
5	**-0.955**	$\mathbf{2.9{\cdot}10^{-3}}$	**1.1746**	**18.71**
10	-0.951	$2.9{\cdot}10^{-3}$	1.1746	19.03
20	-0.946	$2.8{\cdot}10^{-3}$	1.1746	19.45

Table 8.1: Local energies of the H_2 molecule with a bond length of 1.40 Bohr radii, utilizing 50 RBM cycles and 2^{18} Gibbs steps. The standard deviation μ was computed using the blocking method. The subscript C and ref stands for computed and referenced. The reference value represents the experimental ground state energy.

tab:rbm_ hid_h2

CHAPTER 9

Conclusion

sec:
eight

Appendices

APPENDIX A

Machine learning

sec: first-app

A.1 Source Code

All the source code is located in this GitHub repository.

A.2 Derivation of Gradients Used in Backpropagation

The derivation is done according to [7, ch. 7]. Starting of with the expression of finding the gradient of the activation in the last layer with respect to some weight and with respect to some bias by applying the chain rule:

$$\frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial W_{ij}^{last}} = \frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial a_i^{last}}\frac{\partial a_i^{last}}{\partial W_{ij}^{last}} = \frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial a_i^{last}}\frac{\partial a_i^{last}}{\partial z_i^{last}}\frac{\partial z_i^{last}}{\partial W_{ij}^{last}} \tag{A.1}$$

$$\frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial b_i^{last}} = \frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial a_i^{last}}\frac{\partial a_i^{last}}{\partial z_i^{last}}\frac{\partial z_i^{last}}{\partial b_i^{last}} \tag{A.2}$$

Where the nodes in the foregoing layers naturally affect the output which means that the derivatives of these foregoing nodes also is needed, which gives the following expressions of the derivatives of these nodes with respect to the weights and bias:

$$\frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial W_{ij}^{last-1}} = \sum_k \frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial a_k^{last}}\frac{\partial a_k^{last}}{\partial W_{ij}^{last-1}} \tag{A.3}$$

$$= \sum_k \frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial a_k^{last}}\frac{\partial a_k^{last}}{\partial z_k^{last}}\frac{\partial z_k^{last}}{\partial a_i^{last-1}}\frac{\partial a_i^{last-1}}{\partial z_i^{last-1}}\frac{\partial z_i^{last-1}}{\partial W_{ij}^{last-1}} \tag{A.4}$$

$$\frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial b_i^{last-1}} = \sum_k \frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial a_k^{last}}\frac{\partial a_k^{last}}{\partial b_i^{last-1}} \tag{A.5}$$

$$= \sum_k \frac{\partial L\left(\boldsymbol{a}^{last},\boldsymbol{y}\right)}{\partial a_k^{last}}\frac{\partial a_k^{last}}{\partial z_k^{last}}\frac{\partial z_k^{last}}{\partial a_i^{last-1}}\frac{\partial a_i^{last-1}}{\partial z_i^{last-1}}\frac{\partial z_i^{last-1}}{\partial b_i^{last-1}} \tag{A.6}$$

Then by expressing the derivative of the output layer, with respect to the activation in some random layer l, as a sum of the derivatives of the nodes in the following layer with respect to the activation of the same node, the following expression can be written as follows:

$$z_i^l = \frac{\partial L(\boldsymbol{a}^{last}, \boldsymbol{y})}{\partial a_i^l} = \sum_n z_n^{l+1} \frac{\partial a_n^{l+1}}{\partial a_i^l} \tag{A.7}$$

Which gives the final expressions for the derivative of the output layer:

$$\frac{L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial W_{ij}^l} = z_i^l \frac{\partial a_i^l}{\partial W_{ij}^l} \tag{A.8}$$

$$\frac{L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial b_i^l} = z_i^l \frac{\partial a_i^l}{\partial b_i^l} \tag{A.9}$$

APPENDIX B

Quantum Mechanics

sec: second-app

B.1 Hydrogen Molecule

B.1.1 Analytical expression of the local energy

Local energy

To derive the analytical expression for the local energy, an intuitive place to start is by the expression for the local energy and inserting the Hamiltonian expression:

The local energy is given by the following:

$$E_L = \frac{1}{\Psi}\hat{H}\Psi = \sum_{i=1}^{M}\left(-\frac{1}{2\Psi}\nabla_i^2\Psi + \frac{1}{2}\omega^2 r_i^2\right) + \sum_{i<j}\frac{1}{r_{ij}}, \tag{B.1}$$

{eq: ap_LE}

where the heavy-lifting of the computations will be deriving the second derivative of the wave function:

$$\frac{1}{\Psi}\nabla_i^2\Psi,$$

which can be rewritten as:

$$\left(\frac{1}{\Psi}\nabla\Psi\right)^2 + \nabla\left(\frac{1}{\Psi}\nabla\Psi\right) = [\nabla\log\Psi]^2 + \nabla^2\log\Psi,$$

which is easier to compute. Now defining a quantity $\Psi' = \Psi^2$ and differentiating Ψ' will make calculations easier to grasp at a later point. The logarithm of Ψ' can be written as the follow:

$$\log\Psi' = -\log Z - \sum_{i=1}^{M}\left(\frac{(X_i - a_i)^2}{2\sigma^2}\right) + \sum_{j=1}^{N}\log\left(1 + \exp\left(b_j + \sum_{i=1}^{M}\frac{X_i w_{ij}}{\sigma^2}\right)\right).$$

Then using the sigmoid function,

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

and defining the following quantity which is used at a later point as input of the sigmoid function:

$$\delta_j^{input} = b_j + \sum_{i=1}^{M} \frac{X_i w_{ij}}{\sigma^2}.$$

When differentiating the logarithm of Ψ' with respect to the visible nodes the result goes as follows:

$$\begin{aligned}\frac{\partial \log \Psi'}{\partial X_i} &= \frac{(a_i - X_i)}{\sigma^2} + \sum_{j=1}^{N} \frac{w_{ij} \exp\left(b_j + \sum_{i=1}^{M} \frac{X_i w_{ij}}{\sigma^2}\right)}{\sigma^2 \left(1 + \exp\left(b_j + \sum_{i=1}^{M} \frac{X_i w_{ij}}{\sigma^2}\right)\right)}, \\ &= \frac{a_i - X_i}{\sigma^2} + \frac{1}{\sigma^2} \sum_{j=1}^{N} w_{ij} \delta(\delta_j^{input}).\end{aligned} \tag{B.2}$$

{eq:first_der_wfrbm}

Now that the first derivative is defined, the next derivative can be written as follows by differentiating one more time:

$$\frac{\partial^2 \log \Psi'}{\partial X_i^2} = -\frac{1}{\sigma^2} + \frac{1}{\sigma^4} \sum_{j=1}^{N} w_{ij}^2 \delta(\delta_j^{input}) \delta(-\delta_j^{input}), \tag{B.3}$$

{eq:sec_der_wfrbm}

Now that the first and second derivative of $\log \Psi'$ is known, the derivative of the real trial function can be found:

$$\log \Psi = \log \sqrt{\Psi'} = \frac{1}{2} \log \Psi'.$$

This shows that the derivatives of Equation B.2 and Equation B.3 only needs to be multiplied by a factor of $\frac{1}{2}$ to find the correct terms. Everything is then inserted into Equation B.1 which gives the the final expression for the local energy of the system:

$$E_{L,Gibbs} = -\frac{1}{2} \sum_{i=1}^{M} \Bigg[\frac{1}{4\sigma^4} \left(a_i - X_i + \sum_{j=1}^{N} w_{ij} \delta(\delta_j^{input}) \right)^2$$

$$-\frac{1}{2\sigma^2}+\sum_{j=1}^{N}\frac{w_{ij}^2}{2\sigma^4}\delta(\delta_j^{input})\delta(-\delta_j^{input})\Big]+\sum_{i,I}\frac{Z_I}{|\mathbf{r}_i-\mathbf{R}_I|}+\sum_{i<j}\frac{1}{r_{ij}}$$

Where M is the number of visible nodes, and N is the number of hidden nodes.

B.1.2 Derivatives of the free parameters

The derivatives used in the stochastic gradient decent method are the derivatives of the wave function with respect to the parameters. The same way the local energy was computed by defining $\Psi' = \Psi^2$ will make the calculations easier. The gradient of the local energy is defined as follow:

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

Which uses the following expression in the first term:

$$\frac{1}{\Psi'}\frac{\partial\Psi'}{\partial\alpha_k}=\frac{\partial}{\partial\alpha_k}\log\Psi' \tag{B.4}$$

Then by inserting equation (B.1.1) into the equation and differentiate with respect to the different free parameters leaves the following expression:

$$\frac{\partial\log\Psi'}{\partial a_k}=\frac{X_k-a_k}{\sigma^2} \tag{B.5}$$

$$\frac{\partial\log\Psi'}{\partial b_k}=\frac{\exp\left(b_k+\sum_{i=1}^{M}\frac{X_iw_{ik}}{\sigma^2}\right)}{1+\exp\left(b_k+\sum_{i=1}^{M}\frac{X_iw_{ik}}{\sigma^2}\right)}=\delta(\delta^{input}) \tag{B.6}$$

$$\frac{\partial\log\Psi'}{\partial w_{kl}}=\frac{X_k\exp\left(b_l+\sum_{i=1}^{M}\frac{X_iw_{il}}{\sigma^2}\right)}{\sigma^2\left(1+\exp\left(b_l+\sum_{i=1}^{M}\frac{X_iw_{il}}{\sigma^2}\right)\right)}=\frac{X_k}{\sigma^2}\delta(\delta^{input}) \tag{B.7}$$

Now it is time to switch Ψ' with the real trial function Ψ. Because of the following relation

$$log\sqrt{\Psi}=\frac{1}{2}log\Psi,$$

the derivatives with respect to the free parameters will follow the same result as for Ψ', but with a factor $\frac{1}{2}$, which gives the following results:

$$\frac{\partial \log \Psi_T}{\partial a_k} = \frac{X_k - a_k}{2\sigma^2},$$

$$\frac{\partial \log \Psi_T}{\partial b_k} = \frac{1}{2}\delta(\delta^{input}),$$

$$\frac{\partial \log \Psi_T}{\partial w_{kl}} = \frac{X_k}{2\sigma^2}\delta(\delta^{input}),$$

Which is the final gradients of the local energy with respect to the free parameters of the trial function.

B.2 Rewriting the Pairing Hamiltonian Using Jordan-Wigner Transformation

To rewrite the pairing Hamiltonian to an Hamiltonian fit for a quantum computer the first step is to:

CONTINUE HERE!

APPENDIX C

Excessive Results

sec: third-app

In the following appendix, results seemingly not being necessary for the reader is being showed.

C.1 Generative Learning

C.1.1 Parameter investigation

Investigating how the momentum term affects the optimization of a 2-qubit Hamiltonian can be seen in Figure C.1

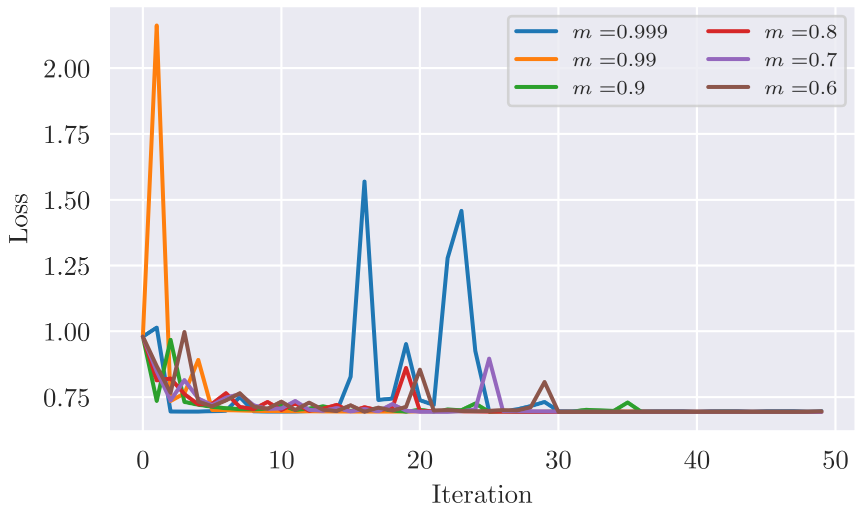


Figure C.1: Loss as a function of iteration using RMSprop with different initial momentum and learning rate of 0.1 The target distribution was the Bell state.

fig: rms_m

Bibliography

HastieTrevor2009EoSL [1] Hastie, T., Tibshirani, R. and Friedman, J., *Elements of Statistical Learning: Data Mining, Inference, and Prediction*, eng, ser. Springer series in statistics. New York: Springer, 2009.

10.5555/3309066 [2] Schuld, M. and Petruccione, F., *Supervised Learning with Quantum Computers*, 1st. Springer Publishing Company, Incorporated, 2018.

cite:kaggletitanic [3] contributors, K., *Titanic - machine learning from disaster*, [Online; accessed 1-April-2021].

cite:nnfig [4] Facundo Bre, *Prediction of wind pressure coefficients on building surfaces using artificial neural networks*, https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051, [Online; accessed 4-April-2021], 2017.

Schmidhuber_2015 [5] Schmidhuber, J., 'Deep learning in neural networks: An overview,' *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

activation_cunctions_citation [6] Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S., *Activation functions: Comparison of trends in practice and research for deep learning*, 2018.

rojasRa:nnsystematic [7] Rojas, R., *Neural Networks: A Systematic Introduction.* Berlin, Heidelberg: Springer-Verlag, 1996.

chain_rule [8] Silva, S. D., *The maths behind back propagation*, [Online; accessed 8-June-2021], 2020.

ACKLEY1985147 [9] Ackley, D. H., Hinton, G. E. and Sejnowski, T. J., 'A learning algorithm for boltzmann machines,' *Cognitive Science*, vol. 9, no. 1, pp. 147–169, 1985.

VQB:litteraturelist [10] Zoufal, C., Lucchi, A. and Woerner, S., 'Variational quantum boltzmann machines,' *Quantum Machine Intelligence*, vol. 3, no. 1, Feb. 2021.

Hinton:2007 [11] Hinton, G. E., 'Boltzmann machine,' *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007, revision #91076.

BMfig [12] Commons, W., *File:boltzmannexamplev1.png — wikimedia commons, the free media repository*, [Online; accessed 31-July-2021], 2021.

BM_book [13] Hjorth-Jensen, M., *Advanced topics in computational physics: Computational quantum mechanics*, https://compphysics.github.io/ComputationalPhysics2/doc/LectureNotes/_build/html/intro.html, [Online; accessed 18-Mai-2021], 2021.

fig_RBM [14] Oppermann, A., *Deep learning meets physics: Restricted boltzmann machines part i*, https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-i-6df5c4918c15, [Online; accessed 18-Mai-2021], 2018.

rbm_article [15] Salakhutdinov, R., 'Learning deep generative models,' *Annual Review of Statistics and Its Application*, vol. 2, no. 1, pp. 361–385, 2015. eprint: https://doi.org/10.1146/annurev-statistics-010814-020120.

article_rbm2 [16] Fischer, A. and Igel, C., 'Training restricted boltzmann machines: An introduction,' *Pattern Recognition*, vol. 47, pp. 25–39, Jan. 2014.

andrychowicz2016learning [17] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B. and Freitas, N. de, *Learning to learn by gradient descent by gradient descent*, 2016. arXiv: 1606.04474 [cs.NE].

ruder2017overview [18] Ruder, S., *An overview of gradient descent optimization algorithms*, 2016.

amsgrad_article [19] Reddi, S. J., Kale, S. and Kumar, S., *On the convergence of adam and beyond*, 2019.

gd_conv_scale [20] Ioffe, S. and Szegedy, C., *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.

griffiths_schroeter_2018 [21] Griffiths, D. J. and Schroeter, D. F., *Introduction to Quantum Mechanics*, 3rd ed. Cambridge University Press, 2018.

10.5555/1972505 [22] Nielsen, M. A. and Chuang, I. L., *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th. USA: Cambridge University Press, 2011.

hemmer2005kvantemekanikk [23] Hemmer, P., *Kvantemekanikk: P.C. Hemmer*. Tapir akademisk forlag, 2005.

lecturenotes:cresser [24] Cresser, J., *Quantum Physics Notes*. Macquarie University 2009, 2009.

gentle_intro_qc [25] Rieffel, E. and Polak, W., *Quantum Computing: A Gentle Introduction*, 1st. The MIT Press, 2011.

boopa [26] Fernández, F., *The born-oppenheimer approximation*, Mar. 2019.

McArdle_2019 [27] McArdle, S., Jones, T., Endo, S., Li, Y., Benjamin, S. C. and Yuan, X., 'Variational ansatz-based quantum simulation of imaginary time evolution,' *npj Quantum Information*, vol. 5, no. 1, Sep. 2019.

solving_manybody_with_ann [28] Carleo, G. and Troyer, M., 'Solving the quantum many-bodyproblem with artificialneural networks,' *Science*, vol. 355, pp. 602–606, 6325 Feb. 2017.

blochsphere_fig [29] Wikipedia contributors, *Bloch sphere — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Bloch_sphere&oldid=1008849646, [Online; accessed 29-March-2021], 2021.

cite_circuit_explained [30] Marciano, D., *Quantum computing for everyone - part iii: Quantum circuits and openqasm, code project*, [Online; accessed 30-March-2021], 2018.

Qiskit_book [31] ANIS, M. S., Abby-Mitchell, Abraham, H., AduOffei, Agarwal, R., Agliardi, G., Aharoni, M., Akhalwaya, I. Y., Aleksandrowicz, G., Alexander, T., Amy, M., Anagolum, S., Arbel, E., Asfaw, A., Athalye, A., Avkhadiev, A., Azaustre, C., BHOLE, P., Banerjee, A., Banerjee, S., Bang, W., Bansal, A., Barkoutsos, P., Barnawal, A., Barron, G., Barron, G. S., Bello, L., Ben-Haim, Y., Bennett, M. C., Bevenius, D., Bhatnagar, D., Bhobe, A., Bianchini, P., Bishop, L. S., Blank, C., Bolos, S., Bopardikar, S., Bosch, S., Brandhofer, S., Brandon, Bravyi, S., Bronn, N., Bryce-Fuller, Bucher, D., Burov, A., Cabrera, F., Calpin, P., Capelluto, L., Carballo, J., Carrascal, G., Carriker, A., Carvalho, I., Chen, A., Chen, C.-F., Chen, E., Chen, J. (, Chen, R., Chevallier, F., Chinda, K., Cholarajan, R., Chow, J. M., Churchill, S., CisterMoke, Claus, C., Clauss, C., Clothier, C., Cocking, R., Cocuzzo, R., Connor, J., Correa, F., Crockett, Z., Cross, A. J., Cross, A. W., Cross, S., Cruz-Benito, J., Culver, C., Córcoles-Gonzales, A. D., D, N., Dague, S., Dandachi, T. E., Dangwal, A. N., Daniel, J., Daniels, M., Dartiailh, M., Davila, A. R., Debouni, F., Dekusar, A., Deshmukh, A., Deshpande, M., Ding, D., Doi, J., Dow, E. M., Drechsler, E., Dumitrescu, E., Dumon, K., Duran, I., EL-Safty, K., Eastman, E., Eberle, G., Ebrahimi, A., Eendebak, P., Egger, D., ElePT, Emilio, Espiricueta, A., Everitt, M., Facoetti, D., Farida, Fernández, P. M., Ferracin, S., Ferrari, D., Ferrera, A. H., Fouilland, R., Frisch, A., Fuhrer, A., Fuller, B., GEORGE, M., Gacon, J., Gago, B. G., Gambella, C., Gambetta, J. M., Gammanpila, A., Garcia, L., Garg, T., Garion, S., Garrison, J. R., Gates, T., Gil, L., Gilliam, A., Giridharan, A., Gomez-Mosquera, J., Gonzalo, Puente González, S. de la, Gorzinski, J., Gould, I., Greenberg, D., Grinko, D., Guan, W., Guijo, D., Gunnels, J. A., Gupta, H., Gupta, N., Günther, J. M., Haglund, M., Haide, I., Hamamura, I., Hamido, O. C., Harkins, F., Hartman, K., Hasan, A., Havlicek, V., Hellmers, J., Herok, Ł., Hillmich, S., Horii, H., Howington, C., Hu, S., Hu, W., Huang, J., Huisman, R., Imai, H., Imamichi, T., Ishizaki, K., Ishwor, Iten, R., Itoko, T., Ivrii, A., Javadi, A., Javadi-Abhari, A., Javed, W., Jianhua, Q., Jivrajani, M., Johns, K., Johnstun, S., Jonathan-Shoemaker, JosDenmark, JoshDumo, Judge, J., Kachmann, T., Kale, A., Kanazawa, N., Kane, J., Kang-Bae, Kapila, A., Karazeev, A., Kassebaum, P., Kelso, J., Kelso, S., Khanderao, V., King, S., Kobayashi, Y., Kovi11Day, Kovyrshin, A., Krishnakumar, R., Krishnan, V., Krsulich, K., Kumkar, P., Kus, G., LaRose, R., Lacal, E., Lambert, R., Landa, H., Lapeyre, J., Latone, J., Lawrence, S., Lee, C., Li, G., Lishman, J., Liu, D., Liu, P., Lolcroc, M, A. K., Madden, L., Maeng, Y., Maheshkar, S., Majmudar, K., Malyshev, A., Mandouh, M. E., Manela, J., Manjula, Marecek, J., Marques, M., Marwaha, K., Maslov, D., Maszota, P., Mathews, D., Matsuo, A., Mazhandu, F., McClure, D., McElaney, M., McGarry, C., McKay, D., McPherson, D., Meesala, S., Meirom, D., Mendell, C., Metcalfe, T., Mevissen, M., Meyer, A., Mezzacapo, A., Midha, R., Miller, D., Minev, Z., Mitchell, A., Moll, N., Montanez, A., Monteiro, G., Mooring, M. D., Morales, R., Moran, N., Morcuende, D., Mostafa, S., Motta, M., Moyard, R., Murali, P., Müggenburg, J., NEMOZ, T., Nadlinger, D., Nakanishi, K., Nannicini, G., Nation, P., Navarro, E., Naveh, Y., Neagle, S. W., Neuweiler, P., Ngoueya, A., Nicander, J., Nick-Singstock, Niroula, P., Norlen, H., NuoWenLei, O'Riordan, L. J., Ogunbayo, O., Ollitrault, P., Onodera, T., Otaolea, R., Oud, S., Padilha, D., Paik, H., Pal, S., Pang, Y., Panigrahi, A., Pascuzzi, V. R., Perriello, S., Peterson, E., Phan, A., Pilch, K., Piro, F., Pistoia, M., Piveteau, C., Plewa, J., Pocreau, P., Pozas-Kerstjens, A., Pracht, R., Prokop, M., Prutyanov, V., Puri, S., Puzzuoli, D., Pérez, J., Quant02, Quintiii, Rahman, R. I., Raja, A., Rajeev, R., Rajput, I., Ramagiri, N., Rao, A., Raymond, R., Reardon-Smith, O., Redondo, R. M.-C., Reuter, M., Rice, J., Riedemann, M., Rietesh, Risinger, D., Rocca, M. L., Rodrıguez, D. M., RohithKarur, Rosand, B., Rossmannek, M., Ryu, M., SAPV, T., Sa, N. R. C., Saha, A., Saki, A. A., Sanand, S., Sandberg, M., Sandesara, H., Sapra, R., Sargsyan, H., Sarkar, A., Sathaye, N., Schmitt, B., Schnabel, C., Schoenfeld, Z., Scholten, T. L., Schoute, E., Schulterbrandt, M., Schwarm, J., Seaward, J., Sergi, Sertage, I. F., Setia, K., Shah, F., Shammah, N., Sharma, R., Shi, Y., Shoemaker, J., Silva, A., Simonetto, A., Singh, D., Singh, D., Singh, P., Singkanipa, P., Siraichi, Y., Siri, Sistos, J., Sitdikov, I., Sivarajah, S., Sletfjerding, M. B., Smolin, J. A., Soeken, M., Sokolov, I. O., Sokolov, I., Soloviev, V. P., SooluThomas, Starfish, Steenken, D., Stypulkoski, M., Suau, A., Sun, S., Sung, K. J., Suwama, M., Słowik, O., Takahashi, H., Takawale, T., Tavernelli, I., Taylor, C., Taylour, P., Thomas, S., Tian, K., Tillet, M., Tod, M., Tomasik, M., Tornow, C., Torre, E. de la, Toural, J. L. S., Trabing, K., Treinish, M., Trenev, D., TrishaPe, Truger, F., Tsilimigkounakis, G., Tulsi, D., Turner, W., Vaknin, Y., Valcarce, C. R., Varchon, F., Vartak, A., Vazquez, A. C., Vijaywargiya, P., Villar, V., Vishnu, B., Vogt-Lee, D., Vuillot, C., Weaver, J., Weidenfeller, J., Wieczorek, R., Wildstrom, J. A., Wilson, J., Winston, E., WinterSoldier, Woehr, J. J., Woerner, S., Woo, R., Wood, C. J., Wood, R., Wood, S., Wootton, J., Wright, M., Xing, L., YU, J., Yang, B., Yang, U., Yao, J., Yeralin, D., Yonekura, R., Yonge-Mallo, D., Yoshida, R., Young, R., Yu, J., Yu, L., Zachow, C., Zdanski, L., Zhang, H., Zidaru, I., Zoufal, C., aeddins-ibm, alexzhang13, b63, bartek-bartlomiej, bcamorrison, brandhsn, charmerDark, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios-ts, gruu, hhorii, hykavitha, itoko, jeppevinkel, jessica-angel7, jezerjojo14, jliu45, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, nico-lgrs, ntgiwsvp, ordmoj, pahwa, sagar, pritamsinha2304, ryancocuzzo, saswati-qiskit, septembrr, sethmerkel, sg495, shaashwat, sternparky, strickroman, tigerjack, tsura-crisaldo, vadebayo49, welien, willhbang, wmurphy-collabstar, yang.luh and Čepulkovskis, M., *Qiskit: An open-source framework for quantum computing*, 2021.

Benedetti_ 2019 [32] Benedetti, M., Lloyd, E., Sack, S. and Fiorentini, M., 'Parameterized quantum circuits as machine learning models,' *Quantum Science and Technology*, vol. 4, no. 4, p. 043 001, Nov. 2019.

PhysRev. 96.1124 [33] Wick, G. C., 'Properties of bethe-salpeter wave functions,' *Phys. Rev.*, vol. 96, pp. 1124–1134, 4 Nov. 1954.

doi: 10.1080/ 00268976400100041 [34] McLachlan, A., 'A variational solution of the time-dependent schrodinger equation,' *Molecular Physics*, vol. 8, no. 1, pp. 39–44, 1964. eprint: https://doi.org/10.1080/00268976400100041.

Yuan_ 2019 [35] Yuan, X., Endo, S., Zhao, Q., Li, Y. and Benjamin, S. C., 'Theory of variational quantum simulation,' *Quantum*, vol. 3, p. 191, Oct. 2019.

Somma_2002 [36] Somma, R., Ortiz, G., Gubernatis, J. E., Knill, E. and Laflamme, R., 'Simulating physical phenomena by quantum networks,' *Physical Review A*, vol. 65, no. 4, Apr. 2002.

Amin_2018 [37] Amin, M. H., Andriyash, E., Rolfe, J., Kulchytskyy, B. and Melko, R., 'Quantum boltzmann machine,' *Physical Review X*, vol. 8, no. 2, May 2018.

GibbsElementaryPI [38] Gibbs, J., 'Elementary principles in statistical mechanics: Developed with especial reference to the rational foundation of thermodynamics.'

Temme_2011 [39] Temme, K., Osborne, T. J., Vollbrecht, K. G., Poulin, D. and Verstraete, F., 'Quantum metropolis sampling,' *Nature*, vol. 471, no. 7336, pp. 87–90, Mar. 2011.

Yung_2012 [40] Yung, M.-H. and Aspuru-Guzik, A., 'A quantum-quantum metropolis algorithm,' *Proceedings of the National Academy of Sciences*, vol. 109, no. 3, pp. 754–759, Jan. 2012.

Poulin_2009 [41] Poulin, D. and Wocjan, P., 'Sampling from the thermal quantum gibbs state and evaluating partition functions with a quantum computer,' *Physical Review Letters*, vol. 103, no. 22, Nov. 2009.

Motta_2019 [42] Motta, M., Sun, C., Tan, A. T. K., O'Rourke, M. J., Ye, E., Minnich, A. J., Brandão, F. G. S. L. and Chan, G. K.-L., 'Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution,' *Nature Physics*, vol. 16, no. 2, pp. 205–210, Nov. 2019.

brandao2019finite [43] Brandao, F. G. S. L. and Kastoryano, M. J., *Finite correlation length implies efficient preparation of quantum thermal states*, 2019. arXiv: 1609.07877 [quant-ph].

kastoryano2016quantum [44] Kastoryano, M. J. and Brandao, F. G. S. L., *Quantum gibbs samplers: The commuting case*, 2016. arXiv: 1409.3435 [quant-ph].

crooks2019gradients [45] Crooks, G. E., *Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition*, 2019. arXiv: 1905.13311 [quant-ph].

atman [46] Altman, E. R., *Synthesizing credit card transactions*, 2019.

padhi2021tabular [47] Padhi, I., Schiff, Y., Melnyk, I., Rigotti, M., Mroueh, Y., Dognin, P., Ross, J., Nair, R. and Altman, E., 'Tabular transformers for modeling multivariate time series,' in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 3565–3569.

UCI_repo2019 [48] Dua, D. and Graff, C., *UCI machine learning repository*, 2017.

deng2012mnist [49] Deng, L., 'The mnist database of handwritten digit images for machine learning research,' *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

Timothy_Masters [50] Masters, T., *Practical Neural Network Recipes in C++*. USA: Academic Press Professional, Inc., 1993.

scikitlearn [51] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 'Scikit-learn: Machine learning in Python,' *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

hydrogen_model_figure [52] Doma, S. B., Abu-Shady, M., El-Gammal, F. N. and Amer, A. A., ‘Ground states of the hydrogen molecule and its molecular ion in the presence of a magnetic field using the variational monte carlo method,’ *Molecular Physics*, vol. 114, no. 11, pp. 1787–1793, 2016. eprint: https://doi.org/10.1080/00268976.2016.1154198.

cyclical_lr [53] Smith, L. N., ‘Cyclical learning rates for training neural networks,’ in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017, pp. 464–472.