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

In this thesis a Variational Quantum Boltzmann Machine(VarQBM) employing the use of Variational Quantum Imaginary Time Evolution(VarQITE) to prepare approximated Gibbs states was implemented. The VarQBM in addition to classical Restricted Boltzmann machines (RBMs) were used to find underlying trends in training data. A method of encoding feature values into a VarQBM is proposed utilizing a neural network feature engineering scheme, compressing multiple features into relatively few qubits of choice, which was chosen to be named NN-VarQBM.

Using the VarQBM to generate probability distributions were tested. The VarQBM proved to generate the Bell state greatly, which argued in favor of the VarQBM being able to confidently generate target distributions. The VarQBM, RBMs, and standard classifiers were applied to classical datasets. The NN-VarQBM achieved the best results in all cases except when applied to a 28**x**28 pixel digit dataset, due to too much loss of information when compressing 784 features into a two qubit Hamiltonian. VarQBM proved to be preferred when dealing with complex features containing complex underlying trends which are hard to classify utilizing classical machine learning methods as long as the number of features is trivial. NN-VarQBM and Bias-VarQBM(having features encoded as an appended bias) applied to the transaction dataset yielded the highest F_1 scores of the classifiers with 0.61 and 0.60 respectively, while the RBM classified all samples as fraud. When classifying the 8**x**8 pixel handwritten dataset, the NN-VarQBM achieved an accuracy of 0.99. The Bias-VarQBM achieved an accuracy of 0.56 showing that the NN-encoding is the preferred encoding scheme. The RBM_4 and RBM_{30} achieved accuracy scores of 0.91 and 0.97, utilizing 4 and 30 hidden nodes. When assessing the same approach on a 28**x**28 pixel handwritten dataset, RBM_4 and RBM_{30} achieved accuracies of 0.84 and 0.92. The NN-VarQBM achieved 0.83 showing that there is a limit of features for the NN-encoding scheme applied to a few qubit Hamiltonians. For all datasets, the NN-VarQBM achieved better results than the Bias-VarQBM. The RBM and VarQITE were utilized to find the ground state energy of the Hydrogen molecule. VarQITE outperformed the RBM. VarQITE managed to compute a ground state energy of -1.09 Hartree, with an error of 4.21% compared to the ground state. And using the RBM achieved a score of 0.96 Hartree with a relatively high dissimilarity of 18.71% compared to the experimental target, mainly due to a Gaussian trial function not being viable.

Acknowledgements

First and foremost I would like to thank my supervisor Morten Hjort-Jensen. I still recall visiting your office looking for thesis ideas, which lead to you inspiring me to dive into the astonishing world of quantum computing, to which I am truly grateful for. I also want to thank you for all the guidance and always giving me the opportunity to explore the directions I found exciting.

I would also like to thank my co-supervisor Stian Bilek which always had the door open for questions, to which I truly appreciate. In addition you always provided me with the help needed when I happened to get stuck either on theory, coding or anything else. In addition you always provided great ideas to investigate in my thesis, just wished there were more time to explore them all.

I would like to thank my supervisors and Linus Ekstrøm, for all the valuable discussions and help provided through our friday meetings. After each meeting I always found myself more motivated.

My family, grandparents, and friends also deserve a huge thanks for always being there providing support and caring about my well-being, you have my endless gratitude.

Finally I would like to thank the computational physics group for always being ready for a social talk or a table tennis match in addition to providing valuable academic discussions.

Abbreviations

VarQBM	Variational Quantum Boltzmann Machine
VarQITE	Variational Quantum Imaginary Time Evolution
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
PQC	Parameterized quantum circuit

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

It feels quite intuitive to start this thesis by addressing the famous Richard Feynman talk [1] that took place in 1981[2], resulting in an idea widely known as the birth of quantum computing. The background of the talk was the idea of simulating quantum systems using so-called quantum computers due to classical computers not having the abilities and resources to simulate quantum systems efficiently. Computational resources needed to simulate quantum systems scale exponentially with the size of the quantum system, using classical computers. When utilizing quantum computers, the resources scale polynomially. The renowned talk kick-started a new field of study, which is still a highly attractive branch of science for researchers, companies, and self-taught individuals all around the world that are eager to learn.

Machine learning on the other hand is also a very popular field of study, maybe even more popular than quantum computing currently. Machine learning is one of the fastest improving technical fields, and is based on the idea of letting the computer improve and learn automatically[3]. It is also widely used for finding hidden relations in data, in addition to generating models which generalize to classify unseen datasets, and much more. Machine learning is a topic that just gets more and more attention within all fields of sciences. Naturally, machine learning has been used to solve problems generally within all disciplines of sciences. In the 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.

Quantum computing is getting increasing recognition all over the world since quantum computers have the possibility of changing the world of mathematical computations within almost all branches of sciences 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 of study within classical computing, namely machine learning. Which resulted in a sub-field of quantum computing from around 2013, called quantum machine learning[2].

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 just using classical computing?

Boltzmann machines which were proposed by Geoffrey Hinton and Terry Sejnowski in 1985 and sparked a large interest within the machine learning community[4], will be a quite essential part of the thesis, especially Quantum Boltzmann machines. Boltzmann machines are unsupervised machine learning networks using undirected graph structures to process information [5]. Boltzmann machines consist of visible- and hidden nodes, and is used to learn probability distributions over a set of inputs[6]. But how does a classical Boltzmann machine translate to quantum machine learning? And how well does a quantum version of the Boltzmann machine perform when applied to classical problems classifying datasets like the transaction- and handwritten digit dataset[7][8][9], and how well does variational quantum imaginary time evolution(VarQITE) which is a central part of the quantum Boltzmann machine perform when applied to quantum mechanical problems like finding the ground state of the H_2 molecule? The quantum Boltzmann machine studied will be a version called a Variational quantum Boltzmann machine(VarQBM)[5], which uses the method of VarQITE[10] to prepare approximate Gibbs states based on the implicit trainable Hamiltonian. The VarQBM utilizes both visible- and hidden qubits as a parallel to the classical version of visible and hidden nodes. In addition, a neural network approach of feature engineering will be proposed to deal with labeled data, compressing multiple features into few qubits, keeping the number of qubits in the VarQBM low even though utilizing data containing multiple features.

All source code is located in the following github repository github.com/philipkarim/master_thesis

1.1 Outline

The thesis is organized 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 important aspects 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 are 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 and Chapter 8 reveal the results in addition to a discussion on applying the Boltzmann machines to classical data and quantum mechanical systems respectively. Here the results and discussions of the former are presented pairwise.

Chapter 9 concludes the findings of the thesis.

Appendix A reveals some derivations regarding classical machine learning.

Appendix B reveals some derivations in connection with quantum mechanics and the restricted Boltzmann machine used to solve the quantum mechanical system.

Appendix C is the third and last appendix containing some of the results and figures not essential for the analysis.

PART I

Machine Learning

CHAPTER 2

Principles of Machine Learning

There is a lot of publicity and promotion regarding machine learning, however 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 about machine learning, but how does machine learning actually work and what are the wonders of it?

The main principles of machine learning are quite straightforward 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 generalization 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 defining some loss function to assess the prediction.

Machine learning in a nutshell consists of the following aspects: the dataset if data is needed both labeled or unlabeled. An optimization algorithm is 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.

2.1 Supervised Learning Algorithms

2.1.1 Linear regression

The most common supervised learning method is called linear regression, where the name hints at 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:

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

where ϵ is the statistical noise in the dataset, which can be described by random irregularities in the data. Noise in machine learning models is crucial to ensure that the model does not overfit. This 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 no longer generalize well to new unseen data, in other words, the model would no longer perform as well as possible when presented with data that was not part of the training scheme, due to the model being so rigorously fitted to the training data, it is easier to grasp the concept of overfitting and underfitting with the visual representation in Figure 2.1. 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 that are optimized when training a model. The true value of a single sample can then be expressed by the following expression:

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

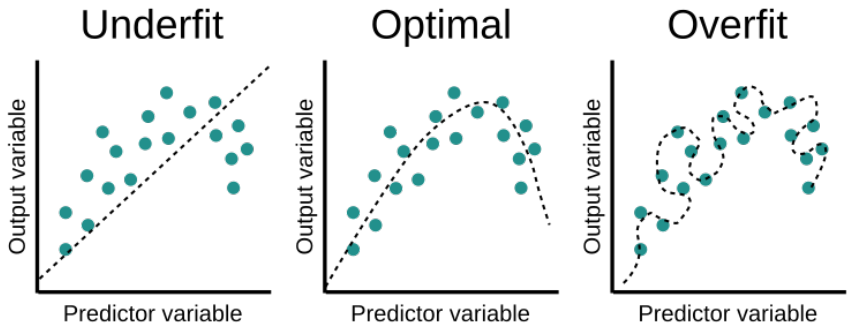


Figure 2.1: A visual representation of underfitting, optimal fitting ,and overfitting from left to right respectively. The underfitted plot uses too brief complexity of the model revealing that the model does not pick up the important trend in the data. The overfitted plot adjusts too much to the data, which makes it harder to generalize to new data. The optimal plot shows how the model is adjusted to the trend of the data without picking up unnecessary details in the data. Figure from [11].

In search of the best generalization, making the finished model able to be applied to 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 are 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:

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

Some loss functions will be presented in section 2.3.

The design matrix

When utilizing different regression methods within most machine learning methods, it is important to sort the data into a well-designed system for easy access and calculations. This is done by sorting the data into a so-called design matrix $\mathbf{X}$. A design matrix can for instance be left looking the following way, where each row represents 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 in the following way:

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

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 last class. The review in this section is based on Hastie et al. [12, 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

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

where G defines the class which will have its probability computed and class C defines the class which will be used as the denominator when computing the probabilities of all other 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:

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

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 dataset $\mathcal{D}(\boldsymbol{x}, \boldsymbol{y})$, 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 [12, p. 120] of a specific output y_i:

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

2.1.3 Nearest neighbors

Nearest neighbors, most known as k-nearest neighbors is a powerful and intuitive method which are still widely used despite its simplicity. As the name describes,

k-nearest neighbors is based on classifying data points based on their nearest neighbors. The k in the name is the number of neighbors that are going to help predict the category of a data point.

The method is best understood intuitively when imagining a two-dimensional graph with data points spread around. Each data point belongs to one of two categories also called classes, where the category of each data point 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 data point that is to be classified within one of the two categories, the k-nearest neighbors method is applied. For example, by having k = 5, the five nearest neighbors will decide the category of the unknown data point. The category that most of the five nearest neighbors belong to, will be the resulting category of the unknown data point.

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

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

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 data point within category y, which is an arbitrary category label from the dataset, can be computed by the following:

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

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

Application of nearest neighbors

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

The example is based on the example found in [13, ch. 1]. The dataset that is going to be used is the titanic dataset, published at the well-known dataset publisher kaggle [14]. The dataset 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 clearly too few to be reliable in real implementations. Only two features from the dataset will be used, the price of the cabin and the room number. The survival of two of the samples is already known, while the last sample is going to be classified as survived or not survived. The small sample dataset from [13, 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 normalization(BN), and the same information after normalization(AN) assuming the maximum ticket price and room number were 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	?(1)

By plotting the data of the travelers the classification is quite easy with such few samples. The plot can be seen in figure 2.2. Due to the unknown sample being closer to the surviving sample, the classifier will label the unknown sample as a survivor, which is correct.

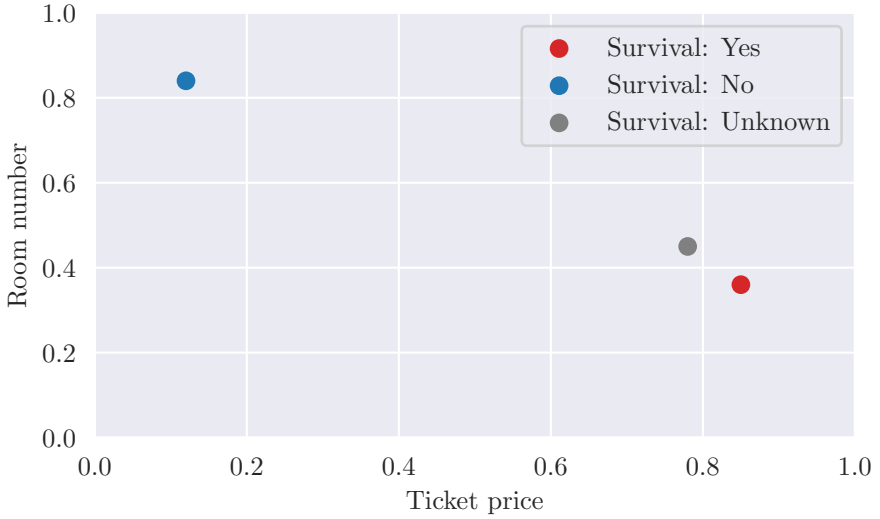


Figure 2.2: Plot of the ticket price and room number for the three travelers.

2.1.4 Dense neural networks

A neural network is a machine learning method that 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 is connected to each node in the layer in front and behind. A visual explanation of a neural network can be seen in Figure 2.3.

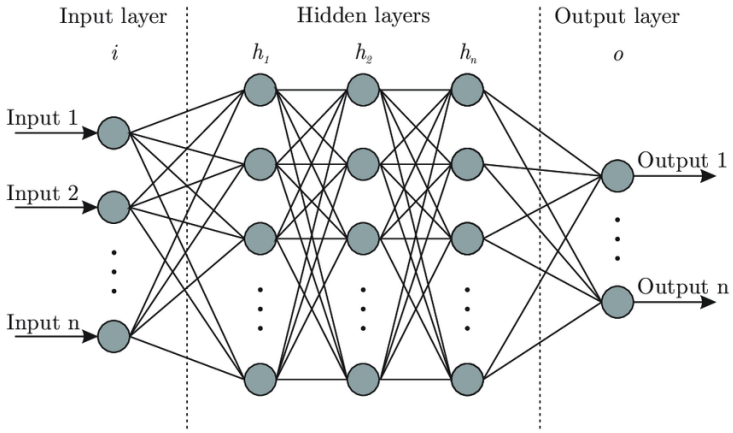


Figure 2.3: Schematic diagram of a dense neural network consisting of up to n hidden layers and nodes. Note that all layers are fully connected. Figure from [15]

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

Feed-forward neural networks which were one of the first and most simplistic types of artificial neural networks [16] are defined by having the information in the neural network only go in one direction without creating cycles of information 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 activations are computed, then the neurons in each layer are computed all the way to the activation(s) of 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 are to go from linear layers to non-linear layers. The activation of layer l is left looking as follows:

$$\boldsymbol{a}^l = f^l(W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l), \tag{2.4}$$

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 [17]. The activation function is often selected based on the issue the neural network is going to overcome. An overview of some popular activation functions can be seen in Table 2.2

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	$f(x) = \max(0.01x, x)$

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

Backpropagation

The widely used training method for a lot of variations of neural networks is 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 is 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 backward 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 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 2.4 to express the activation in the prior layers [18, ch. 7] [19], 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.

2.2 The Boltzmann Machine

The Boltzmann machine was proposed by Geoffrey Hinton and Terry Sejnowski in 1985 and led to a massive interest in Boltzmann machines at the time being[4]. Boltzmann machines are unsupervised machine learning networks using an undirected graph structure to process information [5]. The network is often referred to as a stochastic recurrent neural network, which is able to learn probability distributions over a set of inputs[6]. The reason that Boltzmann machines are stochastic, is because the decision on a node being on or off is made stochastically. Boltzmann machines consist 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 2.4. All nodes are 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[5]. The connections have varying strengths which means that the training of Boltzmann machines is based on optimizing these same weights until the given input results in the desired output with high probability.

Having a look at the stochastic dynamics of Boltzmann machines presented in [6], 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 $\mathbf{z} = \{\mathbf{v}, \mathbf{h}\}$ are 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:

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

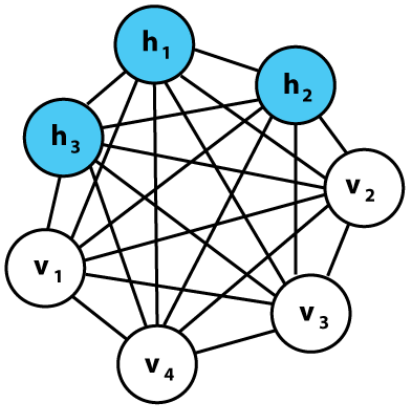


Figure 2.4: Architecture of a Boltzmann machine, having every node connected to each other. This Boltzmann machine consists of four visible nodes and three hidden nodes. Figure from [20]

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 and T is the temperature of the system. k_bT is quite 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 quite well-known feature of Boltzmann machines is that the learning algorithm has the potential to be quite slow due to a hard time converging towards a final state[6]. 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 are connected to one another in the restricted one, while in a regular Boltzmann machine nodes usually are connected to all other nodes[21]. The structure of a restricted Boltzmann machine can be seen in Figure 2.5.

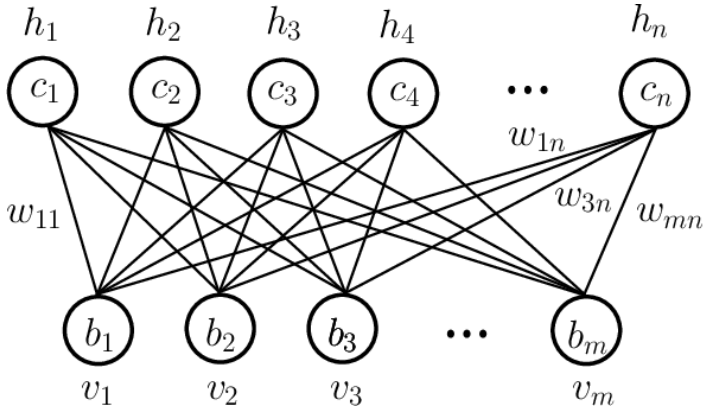


Figure 2.5: Architecture of a restricted Boltzmann machine, having the visible nodes connected to every hidden node and the visible nodes connected to every visible node. The W represents the weight matrix which decides how strong the connection between each visible and hidden node is. Figure from [22].

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:

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i,j}^{V,H} W_{ij} v_i h_j - \sum_{i}^{V} b_i v_i - \sum_{j}^{H} a_j h_j, \tag{2.5}$$

with V and H being the number of visible and hidden nodes respectively, $\theta = \{W, \mathbf{b}, \mathbf{a}\} \in \mathbb{R}$ represent 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[23]:

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

This gives the following by inserting the expression of Equation 2.5 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 $\{0, 1\}$ as values, the last summation can be written out completely which then gives the final marginal probability:

$$p(\mathbf{v}) = \frac{1}{Z} \exp\left(\mathbf{b}^\top \mathbf{v}\right) \prod_{j}^{H} \left(1 + \exp\left(a_j + \sum_{i=1}^{V} W_{ij} v_i\right)\right). \tag{2.6}$$

To determine and update 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:

$$p\left(h_j = 1 \mid \mathbf{v}\right) = \delta\left(\sum_{i}^{V} W_{ij} v_i + a_j\right), \tag{2.7}$$

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

$$p\left(v_i = 1 \mid \mathbf{h}\right) = \delta\left(\sum_{j}^{H} W_{ij} h_j + b_i\right). \tag{2.8}$$

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 quite 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 [23]:

$$E(\mathbf{v}, \mathbf{h}; \theta) = \sum_{i}^{V} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i}^{V} \sum_{j}^{H} W_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{j}^{H} a_j h_j, \tag{2.9}$$

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 2.6, 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 2.9 which gives the following:

$$p\left(v_i = x \mid \mathbf{h}\right) = \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{\left(x - b_i - \sigma_i\sum_j^H h_j W_{ij}\right)^2}{2\sigma_i^2}\right), \tag{2.10}$$

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

$$p\left(h_j = 1 \mid \mathbf{v}\right) = \delta\left(b_j + \sum_i^V W_{ij}\frac{v_i}{\sigma_i}\right), \tag{2.11}$$

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 that is based on modeling a discrete sequence of random variables in a system, where the next state of the system is only dependent on the current state[24]. A Monte Carlo method on the other hand is used to estimate statistical quantities by drawing samples from a distribution, assuming 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 2.7, 2.8, 2.10 and 2.11.

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 of fitting the model to the data in other words.

2.3.1 Some loss functions

Mean absolute error

There are several possible loss functions that determine 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

Ordinary least squares regression

Ordinary least squares (OLS) regression is one of the most common regression models. To fit a linear model the most popular method is using the least squares method, which is based on using the residual sum of squares(RSS) as a cost function [12, ch. 2]. It is worth mentioning that cost- and loss functions often are 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 follows:

$$\text{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - f(x_i; \boldsymbol{\beta}))^2, \tag{2.12}$$

given n data points, sample x, and prediction coefficients $\boldsymbol{\beta}$. Due to Equation 2.12 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:

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

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

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

The derivative is quite straight forward 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:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.13}$$

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

Lasso regression

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

$$\hat{\boldsymbol{\beta}}^{\mathrm{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}. \tag{2.14}$$

It is easy to notice that the first part of Equation 2.14 is the RSS function, while the rest of the formula is the attached regularization. By reducing the coefficients, the model ensures higher bias and lower variance, making overfitting less likely to happen.

Ridge regression

Ridge regression is also a shrinking method, which penalizes the prediction coefficients based on their sizes. The penalty is proportional to the squared size of the coefficients, which secures even higher bias and lower variance in the model compared to Lasso. The optimal coefficients are 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 2.13

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

2.3.3 Batch gradient descent

Almost all machine learning tasks can be defined as a task of optimizing some function $f(\theta)$ [25]. 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 that minimizes the function $f(\theta)$. The gradient descent is applied by moving toward the negative gradient, closing in towards the minimum more and more with every step taken with the gradient descent.

The gradient descent method is quite straightforward. An initial parameter θ, also known as a guess, is needed to start the method and compute the next θ. The gradient descent formula goes as follows:

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

where $J(\theta)$ is a parameterized objective function, where the parameters are 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 [26]. Due to the computation being run for multiple samples before updating θ once, the road towards the minimum might get rough time- and memory-wise.

2.3.4 Stochastic Gradient Descent

The optimization of parameters can be quite 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 has quite a bit of similarity 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 off the memory when using the SGD in addition to usually much faster convergence towards the optimal parameters. A step using SGD 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 is 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 quite 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.

$$\begin{aligned} m_n &= \beta_1 m_{n-1} + (1 - \beta_1) g_n, \\ v_n &= \beta_2 v_{n-1} + (1 - \beta_2) g_n^2, \end{aligned} \tag{2.15}$$

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 a true weighted average. 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 [26].

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

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 quite 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 was proved to be due to the use of exponential moving averages[27]. 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 goes 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 2.15

2.3.7 RMSProp

The funny thing about RMSProp is that it is quite well known, even though it was never officially published, but rather proposed in a lecture by Geoff Hinton [26]. RMSProp is an adaptive learning rate optimizer that uses the squared gradients from present and past iterations to tune the learning rate during training. The squared gradient term is computed the following way:

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

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 inserting it into the machine learning models brings some well-known practical aspects with it. A lot of loss functions define how

well 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 data points are pushed closer together, thereby affecting the model approximately the same, without outliers having an 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, the convergence of training is easier achieved when using gradient descent[28]

2.4.1 Standardization

Standardization also called Z-score normalization 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 data point, $\bar{x}$ is the average of the feature values and σ is the standard deviation of the same feature vector.

2.4.2 Min-max normalization

Min-max normalization, 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 quite simple and goes as follows:

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

where $\max(\cdot)$ and $\min(\cdot)$ are the max and min of the given feature vector.

2.5 Evaluation Metrics for Classification Models

2.5.1 Accuracy

Accuracy is a well-known evaluation metric to measure the performance of classification models within machine learning. The accuracy is defined as the number of correctly labeled samples as a ratio of the total number of samples evaluated. The formula goes as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN},$$

where TP and TN stand for true positive and true negative, and FP and FN stand for false positive and false negative. Using accuracy as a metric of how well a classification model performs is a quite good way of assessing a machine learning model when using balanced datasets. Balanced datasets consist of an approximately equal number of samples from each class.

2.5.2 Precision, Recall and F_1 score

The precision metric is based on assessing how well the model predicts true positives compared to all samples that is being labeled as positives[29].

The formula for precision goes as follows:

$$Precision = \frac{TP}{TP + FP}.$$

Recall is a metric based on the amount of positive samples that are labeled correct, as a ratio of all positives in the dataset.

The formula for recall goes as follows:

$$Recall = \frac{TP}{TP + FN}.$$

The F_1 score is a metric combining both precision and recall. The F_1 metric is often used as an assessment method of machine learning methods due to the combination of precision and recall. And is often used instead of accuracy when dealing with imbalanced datasets.

The formula for the F_1 score goes as follows:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}.$$

PART II

Quantum Mechanics and Many-Body Methods

CHAPTER 3

Quantum mechanics

3.1 Basics of Quantum Mechanics

To understand quantum computing and naturally many-body methods as well, one should know a bit about quantum mechanics. Therefore it is natural to have a brief 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 quite shallow and more focused on the mathematical essence. For a more in-depth explanation please have a look at [30] and [31] 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 in 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 consist of a finite number of dimensions [32, ch. 6]. All wave functions can be written as a sum of basis vectors in the following way:

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

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 is squared:

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

which is the square product of the expansion coefficient from 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 are 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 is 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:

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

This 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 [32, 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 quite useful in quantum mechanics. Commutation of operators can be written as follows:

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

This means that the order the operators are applied to some eigenstate is irrelevant in 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 specifications 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 [33, 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 the evolution in quantum systems expressed mathematically? One of the so-called quantum mechanical postulates is stated as follows in [31, 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 t_1 and t_2,

$$|\psi'\rangle = U|\psi\rangle.$$

Briefly 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 [31, 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

$$p(m) = \langle \psi \left| M_m^\dagger M_m \right| \psi \rangle ,$$

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:

$$1 = \sum_m p(m) = \sum_m \langle \psi \left| M_m^\dagger M_m \right| \psi \rangle .$$

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 quite 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 in 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 that 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. Having a constant global phase factor of $e^{i\theta}$ where θ is a real number, multiplied with a quantum state $|\psi\rangle$ the following way: $e^{i\theta}|\psi\rangle$ is equal to $|\psi\rangle$ up to a 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.

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

This shows that global phase factors can be ignored from an observer's standpoint.

Relative phases on the other hand are dependent on the basis. Introducing two arbitrary quantum states with amplitudes α and β. The two quantum states differ by a relative phase of the basis if there is a real number θ which satisfy the following relation $\alpha = \exp(i\theta)\beta$. 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 show 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 called a pure state when the exact quantum state is known, and a mixed state else. But what if a quantum state depends on the state of both quantum systems? Such systems are called composite systems and bring a quantum property called entanglement[13, ch. 3].

Entanglement of quantum systems is 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 suffices to observe that a quantum state expressed as a tensor product of the sub-states on the following form is separable, hence not entangled [13, ch. 3]:

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

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:

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

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[34, ch. 3]:

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

Which gives the following:

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

There are 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 was 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 Bob's qubit also has 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 Alice's qubit will be the same as Bob's, 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

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 quite 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 [30, ch. 8, p. 327]

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

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 quite similar wave functions. That means that if one should be fortunate enough to choose the correct wave function, the exact ground state would be yielded, but this is often not the case. The trial wave function is parameterized, containing trainable parameters. The parameters is varied to minimize the energy, knowing that the energy is just an upper bound of the ground state energy [30].

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 quite popular quantum system to compute the ground 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 is based on handling nuclei as fixed point charges. A general 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 [35], 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 the 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 can be treated as fixed-point charges. The kinetic energy of the nuclei is therefore set equal to 0. In addition, the interaction energy between the nuclei is constant, because the distance between them is set, hence constant. The Hamiltonian without its constant terms is then left looking as follows[10, S.M.][1]:

$$\hat{H} = -\sum_i \frac{\nabla_i^2}{2} - \sum_{i,I} \frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \frac{1}{2}\sum_{i \neq j} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}, \tag{4.2}$$

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.

4.2.2 RBM: Neural network quantum state as the wave function

As presented by Carleo and Troyer[36], 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 standard supervised machine learning methods do, the NQS method will facilitate the fact that minimizing the energy by optimizing the weights and biases gives the best solution.

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

[1] S.M.:Supplementary material

$$\Psi = \sqrt{F_{rbm}(\mathbf{X}, \mathbf{H})} = \sqrt{\frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{X}, \mathbf{H})}}, \tag{4.3}$$

where $\boldsymbol{X}$ is a vector of visible nodes representing the positions of the particles and dimensions, $\boldsymbol{H}$ 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 have continuous values, while the hidden nodes are restricted to be either 0 or 1. T_0 is set to 1 and Z is the partition function often referred to as a normalization factor 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 as:

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

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.

The marginal probability is written as follows:

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

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 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 system's local energy

The local energy of the system can be computed by combining Equation 4.1, the Hamiltonian in Equation 4.2, and the wave function from Equation 4.3 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.

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

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

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

Using Slater determinants is a well-known concept of many-body methods. Slater determinants are used to express a fermionic wave function assuming that the wave function of a quantum system can be written as a product of one-particle states[37].

A two particle Slater determinant $\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2)$ can be written as follows:

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{1}{\sqrt{2}} \begin{vmatrix} \phi_2(\boldsymbol{x}_1) & \phi_2(\boldsymbol{x}_2) \\ \phi_1(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_2) \end{vmatrix} = |1,2\rangle .$$

This easily expands to n particles in the many-body system:

$$\Psi(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \frac{1}{\sqrt{n!}} \begin{vmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_n(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_n) & \phi_2(\mathbf{x}_n) & \cdots & \phi_n(\mathbf{x}_n) \end{vmatrix} = |12...n\rangle , \tag{4.4}$$

which fulfills the anti-symmetric necessity when having a system of fermions, due to the sign of the wave function switching when switching two rows of the determinant hence switching two particles:

$$\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_i, \cdots, \boldsymbol{x}_j, \cdots \boldsymbol{x}_n) = -\Psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_j, \cdots, \boldsymbol{x}_i, \cdots \boldsymbol{x}_n) .$$

4.4 Second Quantization

Second quantization formalism is a practice of handling many-body systems by introducing creation and annihilation operators. Even though second quantization is not the main core of the thesis, rewriting Hamiltonians using second quantization formalism is essential when transforming Hamiltonians into an expression suitable for being simulated using quantum computers.

Before revealing the creation and annihilation operators it is worth mentioning how a wave function of slater determinant products is expressed using occupation number formalism. By having a wave function of slater determinants as in Equation 4.4, the wave function can be expressed as $|\alpha_1, \alpha_2, ...\alpha_n\rangle$ with α_x representing the number of particles occupying each orbital. This translates to $\alpha_x \in \{0, 1\}$ when having a system of fermions due to no fermions being able to occupy the same state at the same time.

4.4.1 Creation and annihilation operators

Now let us present a creation operator $a_\alpha^\dagger$ which creates particles in one-particle states, and an annihilation operator annihilating particles in the same states a_α. Now the creation operator can be used to fill the normalized vacuum state $|0\rangle$ containing no orbitals at all, in the following way:

$$a_\alpha^\dagger |0\rangle \equiv |\alpha\rangle,$$

and removed by the annihilation operator:

$$a_\alpha|\alpha\rangle \equiv |0\rangle.$$

It is worth mentioning that all wave functions in this section are anti-symmetric unless else is stated. If a Slater determinant does not contain the particle acted upon by its annihilation operator or does already contain a particle acted upon by its corresponding creation operator, the Slater determinant is terminated, the following way:

$$a_\alpha^\dagger|\alpha\rangle \equiv 0, \quad a_\alpha|0\rangle \equiv 0.$$

Applying the creation operator on the vacuum state multiple times, each creation operator corresponding to its' particle will then result in a system of multiple particles, hence a product of the creation operators applied to the vacuum state can be used to define an anti-symmetric state the following way [38]:

$$a_{\alpha_1}^\dagger a_{\alpha_2}^\dagger \dots a_{\alpha_n}^\dagger|0\rangle = |\alpha_1 \dots \alpha_n\rangle\,.$$

The fermionic creation and annihilation operators have the following anti-commutation rules which ensures anti-symmetry:

$$\left\{a_\alpha^\dagger, a_\beta^\dagger\right\} = \{a_\alpha, a_\beta\} = 0 \quad \left\{a_\alpha^\dagger, a_\beta\right\} = \delta_{\alpha\beta},$$

where the derivation can be seen in [38]

4.4.2 Second quantized Hamiltonian

From section 4.2 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 has to be written as a second quantized representation. The electronic Hamiltonian consisting of terms regarding electrons is given by the following expression from [10, S.M.]:

$$H = \sum_{\alpha,\beta} h_{\alpha\beta} a_\alpha^\dagger a_\beta + \frac{1}{2} \sum_{\alpha,\beta,\gamma,\delta} h_{\alpha\beta\gamma\delta} a_\alpha^\dagger a_\beta^\dagger a_\gamma a_\delta, \tag{4.5}$$

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

$$h_{\alpha\beta} = \int \mathrm{d}\mathbf{x} \phi_\alpha^*(\mathbf{x}) \left(-\frac{\nabla_i^2}{2} - \sum_I \frac{Z_I}{|\gamma - \mathbf{R}_I|} \right) \phi_\beta(\mathbf{x}).$$

And $h_{\alpha\beta\gamma\delta}$ is the two-body integral given as follows:

$$h_{\alpha\beta\gamma\delta} = \int \mathrm{d}\mathbf{x}_1 \, \mathrm{d}\mathbf{x}_2 \frac{\phi_\alpha^* \left(\mathbf{x}_1\right) \phi_\beta^* \left(\mathbf{x}_2\right) \phi_\delta \left(\mathbf{x}_1\right) \phi_\gamma \left(\mathbf{x}_2\right)}{|\gamma_1 - \gamma_2|},$$

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

PART III

Quantum Computing

CHAPTER 5

Quantum Computing: Machine Learning

5.1 Introduction to Quantum Computing

Quantum computing uses the occurrences of quantum mechanics to process and manipulate information. Quantum computing has endless possibilities to speed up computations when solving certain problems. But to understand the beauty of machine learning using quantum computing, one needs to know the basics of quantum computing first, here comes a brief summary with the most important key elements to grasp the thesis. For a deeper dive into the theory of quantum computing, a good place to start would be the following book [31].

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 take values of 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 is only natural. The two states a qubit can have when being observed is $|0\rangle = [1, 0]^T$ and $|1\rangle = [0, 1]^T$. Where $\langle 1|0\rangle = 0$ and $\langle 0|0\rangle = 1$ due to the 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:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

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

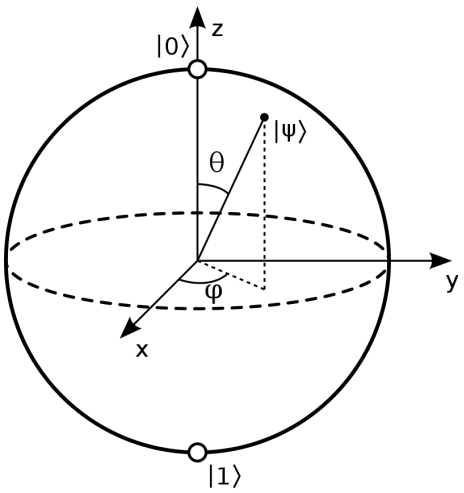


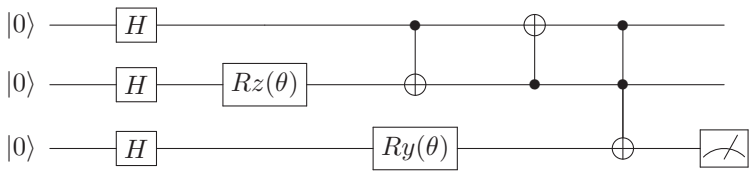
Figure 5.1: The Bloch sphere shows a geometrical representation of the quantum states of a qubit. A qubit can take a position anywhere on the Bloch sphere of the following form: $|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle$, while classical bits are prohibited from taking positions other than the top and bottom. Figure from [39]

A qubit system can be expressed as tensor products of the qubit states $1, 2, ...n$, the following way:

$$|\psi_1\rangle \otimes |\psi_2\rangle ... \otimes |\psi_n\rangle = |\psi_1\psi_2...\psi_n\rangle.$$

5.1.2 Quantum circuits

Quantum circuits consist 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:



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 that 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 encounter 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 show the presence of controlled gates between the two connected wires. Some quantum gates will be discussed in further detail in the next subsection. The last part of the third qubit in the drawn circuit, reveals that a measurement of the qubit will happen here. Since measuring a qubit forces the quantum state to collapse into a distinctive state of $|0\rangle$ or $|1\rangle$, circuits are simulated multiple times. Then by tracking how many times each state is measured, the probability of having the state in either state $|0\rangle$ or $|1\rangle$ is achieved [40].

Quantum Logical Gates

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 unitary matrices. There are infinitely many possible ways of rotating the statevector around the Bloch sphere, resulting in an infinite of ways of manipulating quantum states. Some quantum states are more popular than others, let us have a look at some of the most frequently used ones. The following section is roughly based on [41].

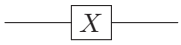
The **Pauli gates** are often used within quantum circuits. The Pauli X-gate or NOT gate which it is also named can be thought of as a rotation around the x axis π radians. The NOT gate is represented by the following matrix and bracket:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0|, \tag{5.1}$$

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

$$\begin{aligned} X|0\rangle &= |1\rangle, \\ X|1\rangle &= |0\rangle. \end{aligned}$$

The Pauli X-gate is usually drawn as follows in a circuit diagram:



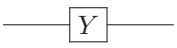
The Pauli Y-gate can be thought of as a rotation around the y axis π radians. The Y-gate is represented by the following matrix and bracket:

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

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

$$\begin{aligned} Y|0\rangle =& i|1\rangle \\ Y|1\rangle =& -i|0\rangle. \end{aligned}$$

The Pauli Y-gate is drawn as follows in a circuit diagram:



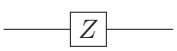
The Pauli Z-gate works the same way as the other Pauli gates. The Z-gate is represented as a rotation around the z axis π radians. The Z-gate is represented by the following matrix and bracket:

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

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

$$\begin{aligned} Z|0\rangle =& |0\rangle, \\ Z|1\rangle =& -|1\rangle. \end{aligned}$$

The Pauli Z-gate is usually drawn as follows in a circuit diagram:



Since there exist operators that rotate quantum states around a certain axis of the Blochs sphere by π radians, it is also utilitarian to rotate qubits an arbitrary amount of radians along the x, y, or z axis. In fact the **rotational operator gates** do this exactly. The rotational operators are given by the following matrices:

$$\begin{aligned} R_x(\theta) =& \exp(-iX\theta/2) = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}, \\ R_y(\theta) =& \exp(-iY\theta/2) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}, \\ R_z(\theta) =& \exp(-iZ\theta/2) = \begin{bmatrix} \exp(-i\theta/2) & 0 \\ 0 & \exp(i\theta/2) \end{bmatrix}, \end{aligned}$$

where the parameter θ decides how many radians the state will be rotated. The subscript indicates which axis the rotation is happening along.

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

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

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:

$$\begin{aligned} H|0\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}},\\ H|1\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \end{aligned}$$

Controlled quantum gates are quantum gates that act 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 qubit, while the qubit which potentially goes through the operation of the gate is called a target qubit. An example of a popular controlled gate is the well-known CNOT which is a controlled NOT gate from Equation 5.1. The CNOT on two qubits controls if the first qubit is in state $|1\rangle$. If the controlled qubit possesses the state, the NOT gate will act on the second qubit mapping $|0\rangle$ to $|1\rangle$ and the 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 is the target which potentially acts. In case of having a qubit in superposition, the controlled gate will act on each term of the superposition individually.

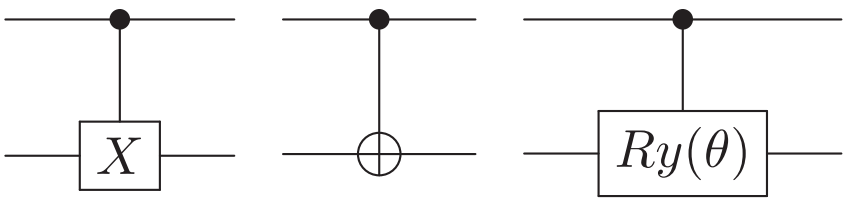


Figure 5.2: Some controlled quantum gates. a) A controlled NOT gate, called a CNOT. b) Also a CNOT but a slightly different visual appearance but does the same as a controlled NOT gate. c) A controlled Ry gate.

Quantum circuit depth

Quantum circuit depth is a term used for the time complexity of running quantum circuits. The longest path in a circuit along the quantum wires can be used as a pointer for the circuit depth. Because a quantum state is evolving through time as quantum gates are manipulating the information, the time

complexity is used to refer to the depth or the time steps needed to execute the circuit. The circuit depth can be thought of as the number of time steps needed to run the whole circuit, with qubits being able to run through parallel gates at the same time step.

Keeping quantum states stable for longer periods of time is quite 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. The circuit depth is often the bottleneck when performing quantum computations, which is the more reason that keeping track of the circuit depth is quiet important in some cases.

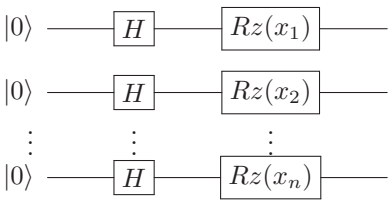
5.1.3 Parameterized Quantum Circuits

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 depend on some parameter(s) are put together into a circuit that is used as an ansatz.

PQCs and supervised learning fit like a glove. The reason for this is that supervised learning aims to learn trends from data that can be generalized to unseen data. In search of the perfect generalization 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. This is easily transferable to PQCs by having the rotational gates of a circuit act as the trainable parameters. To read more about machine learning and PQCs feel free to have a look here [42].

Encoders

The first part of quantum supervised learning is to encode information into the circuit. There are multiple ways of encoding the data e.g. basis encoding and amplitude encoding[13, ch. 5]. A quite regular and intuitive way of encoding data into quantum circuits is by having every qubit start with a Hadamard gate splitting the quantum state into a superposition of the measurement basis, which is then acted on by an $Rz(x)$ gate. The clue on encoding classical data into quantum states lies here because the input x would then be the classical feature data. The encoding part of the circuit would then look as follows:

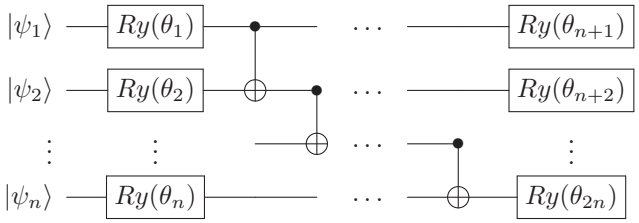


where n is the number of qubits. Rewriting each feature into a $Rz(x)$ quantum gate gives each feature its own qubit. Therefore the number of qubits would be equal to the number of features, excluding possible qubits used for measuring.

Which would have a hard time being applied to real-life quantum hardware when analyzing large datasets of many features.

Ansatzes

Ansatzes used in quantum computations are parts of quantum circuits which is assumed to mimic the behavior of some function. Making the ansatz more complex opens up the possibility of reaching larger parts of the Hilbert space where all quantum states reside. Making really large and complex quantum circuits is difficult due to quantum states being quite unstable over time, which is why ansatzes must avoid being too complex. 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 θ_x is the variational parameters that need to be optimized and n is the number of qubits. Notice that the input states of the circuit are $|\psi_x\rangle$ due to the input of the ansatz-part of the circuit is the already altered quantum state from the encoder.

Entanglement of the qubits

The last part of a PQC is the entangling part which entangles qubits together, making it possible for the qubits to affect each other's 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 often intertwined in the ansatz or appended as an extra part of controlled gates to the PQC. For instance, all qubits could entangle onto a measurement qubit, which yields a 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 $|1\rangle$.

5.2 Quantum Nearest Neighbors

Introducing quantum nearest neighbors(QNN) as an example of how classical machine learning methods are applied to quantum computers. QNN is based on the same strategy as in subsection 2.1.3, but with some smaller implementation differences to make the method suitable for quantum computing. This section is closely based on the example provided in [13, ch. 1].

Then it is time to have a look at the same example as in subsection 2.1.3 if implemented on a quantum computer, classifying the unknown sample in the small dataset in Table 2.1.

To use the nearest neighbors method with quantum computers, the first step is naturally to do a little preprocessing of the data. The data consisting of a vector of the ticket price and room number is first and foremost normalized to unit length. 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 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 normalization(BN), and the same information after normalizing the data to unit length(AN) assuming the maximum ticket price and room number were 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	?(1)

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

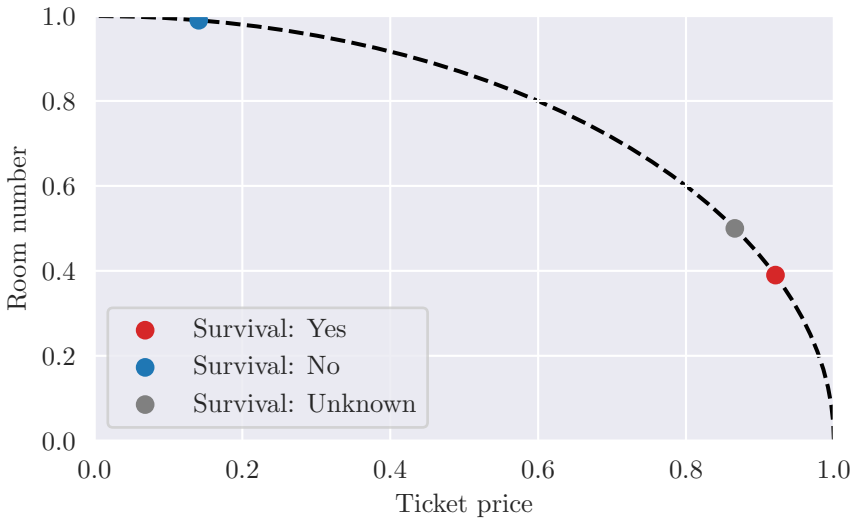


Figure 5.3: Plot of the ticket price and room number for the three travelers, after normalizing the data to unit length.

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 normalizing the vector. Next step follows and the features of the data are appended to the same vector, with the unknown sample appended twice, giving the following vector:

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

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

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

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

which can be written in the form of a quantum state as follows:

$$\begin{aligned} |\psi_0\rangle = \frac{1}{2}(&0.921|0001\rangle + 0.390|0011\rangle + 0.141|0100\rangle + 0.990|0110\rangle \\ &+0.866|1001\rangle + 0.500|1011\rangle + 0.866|1100\rangle + 0.500|1110\rangle). \end{aligned} \tag{5.2}$$

Now that the preprocessing and encoding are done, it is time for the next step, which is to apply the quantum operations to the quantum state in Equation 5.2. When performing the nearest neighbors method, the Hadamard gate is a central quantum operator to be used. The Hadamard operator creates new amplitude vectors when applied to some quantum state. Since the Hadamard operator maps the states into superpositions, the new amplitude vector contains a mix of the different samples.

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:

$$\begin{aligned} |\psi_1\rangle = H|\psi_0\rangle = \frac{1}{2}(&(0.921 + 0.866)|0001\rangle + (0.390 + 0.500)|0011\rangle \\ &+(0.141 + 0.866)|0100\rangle + (0.990 + 0.500)|0110\rangle \\ &+(0.921 - 0.866)|1001\rangle + (0.390 - 0.500)|1011\rangle \\ &+(0.141 - 0.866)|1100\rangle + (0.990 - 0.500)|1110\rangle). \end{aligned} \tag{5.3}$$

Written in a more compact way equals:

$$|\psi_1\rangle = H|\psi_0\rangle = 0.8935|0001\rangle + 0.445|0011\rangle + 0.5035|0100\rangle + 0.745|0110\rangle + 0.0275|1001\rangle - 0.055|1011\rangle - 0.3625|1100\rangle + 0.245|1110\rangle.$$

Now the next step is to reject half of the states due to the fact that half of the states has the unknown sample data encoded into its state, which is more clear having a look at Equation 5.3. There is only a 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 5.3 that the states that need to be rejected are the states which have the first qubit in state $|1\rangle$, or else the amplitude is set to 0. The resulting states named ψ_2 go as follows:

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

With χ being a normalization constant, which equals the following after normalization.

$$|\psi_2\rangle = 0.665|0001\rangle + 0.331|0011\rangle + 0.375|0100\rangle + 0.555|0110\rangle).$$

As mentioned at the start of the subsection, the fourth qubit was added to be used as a label indicator. This means that states with equal fourth qubits represent 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 categories by adding the squared amplitudes within each class as follows:

$$p(|xxx0\rangle) = |0.375|^2 + |0.555|^2 \approx 0.448.$$

This 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:

$$p(|xxx1\rangle) = 1 - p(|xxx0\rangle) = |0.665|^2 + |0.331|^2 \approx 0.552.$$

This means that the unknown passenger is classified as a survivor, which is the same as in the classical case of nearest neighbors.

QNN vs. classical NN

The results of the calculations done in section 2.1.3, showed that computing the nearest neighbor method for both the classical and quantum computing case managed to classify the unknown sample correctly.

Comparing the complexity of both the methods reveals that the quantum implementation of nearest neighbors is more efficient the more samples the

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

The bad thing in the classical case is that the length to its neighbors needs to be computed for each sample, which can be quite computationally expensive when dealing with a large number of samples, or if the amount of contributing neighbors is high bringing a lot of contribution to take into account for 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 has a possibility of being samples that lay within the same classes getting pushed away from each other when forcing each sample along the same unit line. Observing the example with quantum nearest neighbors showed that in this case loss of information was not a problem.

5.3 Variational Quantum Boltzmann Machine

This section is closely related to and based on the article [5]. 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 model in search of the optimal 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 determine 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:

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

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

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

As mentioned earlier, the purpose of a QBM is to compute probability distributions such that sampling ρ^{Gibbs} corresponds to sampling from a classical target distribution with as much overlap between the probability distributions as possible. The loss normally used and will be utilized in this thesis is cross entropy loss:

$$L = -\sum p_v^{\text{data}} \log p_v^{\text{QBM}}, \tag{5.4}$$

with p_v^{data} being the probability of having configuration 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 parameter. The difference between a regular QBM and the VarQBM which will be utilized in this thesis is that in VarQBM the gradient of the loss function is computed analytically. The steps of a VarQBM are visualized in figure 5.4.

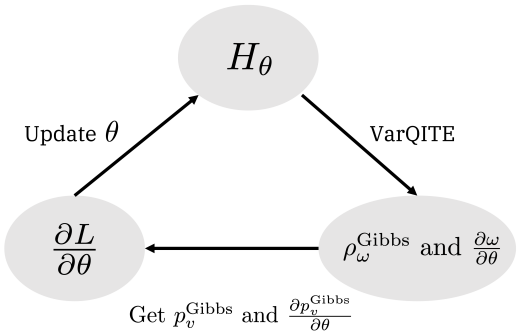


Figure 5.4: Steps of VarQBM, first step is to approximate the Gibbs states using VarQITE. The next step is to find the gradient of the loss function which is then used to update the variational parameters according to some classical optimization algorithm. Figure from [5]

The goal of the method is to be able to reproduce probability distributions by optimizing some variational parameter θ such that sampling from a generated distribution using Gibbs states prepared with VarQITE approximates the training distribution. The variational parameters are optimized by minimizing Equation 5.4 the following way:

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

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

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

which is computed by using the chain rule. The expression for the gradients of the configurations with respect to θ_i is then left looking as:

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

where $\omega(\tau)$ is some time evolved rotational parameter of a trial circuit $V(\tau) = U_q(\omega_q)\dots U_1(\omega_1)$ and U_x being a quantum gate dependent on ω_x. The gradient of configuration p_v^{QBM} with respect to ω is given as:

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

which can be computed using the parameter-shift rule [43], which is based on shifting the parameters aside and approximating the gradient of the shift. $\frac{\partial \omega_k(\tau)}{\partial \theta_i}$ is attained during Variational Quantum Imaginary Time Evolution(VarQITE) which will be revealed soon. At last, the optimizer is utilized to optimize the variational parameters, completing one Boltzmann cycle.

5.3.1 Variational quantum imaginary time evolution

When facilitating a VarQBM, Gibbs states that need to be prepared during training. A way to do this is to utilize a method called Variational 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 quite usual when looking for ground state energies and studying quantum systems using classical computers [10]. Following the explanation in [5], ITE is based on starting 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 for instance the following time-independent Hamiltonian H which will be used in the VarQBM:

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

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

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

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

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

The evolution of the quantum state is represented by the Wick-rotated Schrödinger equation. Wick rotations are used to solve mathematical problems by substituting real variables with imaginary variables, done here by introducing imaginary time [44]. The Wick-rotated Schrödinger equation is represented by the following differential equation:

$$\frac{d\left|\psi_\tau\right\rangle}{d\tau} = -\left(H - E_\tau\right)\left|\psi_\tau\right\rangle, \tag{5.5}$$

with E_τ given as the energy when H acts on ψ_τ. According to Zoufal, Lucchi, and Woerner [5], the terms in the Hamiltonian representing the largest eigenvalue decay 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 the ground state of the system when the imaginary time goes towards infinity[5]. Further on one can prepare the Gibbs states by having a finite limit $\tau = 1/2\,(\mathrm{k_B T})$ on the time evolution.

Now that the main details of ITE are unveiled, it is time to jump into varQITE. The idea of varQITE is to minimize the energy using McLachlan's variational principle [45] 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 $V(\tau) = U_q(\omega_q)\ldots U_1(\omega_1)$. An initial quantum state is also defined as $|\psi_{in}\rangle$ which is the input state. The trial state is then written as follows:

$$\left|\psi_\omega\right\rangle := V(\omega)\left|\psi_{\mathrm{in}}\right\rangle.$$

Rewriting Equation 5.5 by throwing all terms to the left hand side and factorizing the evolved state out gives rise to writing the equation as follows:

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

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:

$$\delta\| \left(\frac{d}{d\tau} + H - E_\tau\right)\left|\psi_\omega\right\rangle \| = 0. \tag{5.6}$$

Now McLachlan's variational principle is a central part of determining the parameters $\omega(\tau)$. Then solving Equation 5.6 leads to the following system of linear equations:

$$A\dot{\omega} = C, \tag{5.7}$$

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

$$
\begin{aligned}
A_{pq}(\tau) &= \mathrm{Re}\left(\mathrm{Tr}\left[\frac{\partial V^{\dagger}(\omega(\tau))}{\partial \omega(\tau)_p}\frac{\partial V(\omega(\tau))}{\partial \omega(\tau)_q}\rho_{\mathrm{in}}\right]\right), \\
C_p(\tau) &= -\sum_i \theta_i \,\mathrm{Re}\left(\mathrm{Tr}\left[\frac{\partial V^{\dagger}(\omega(\tau))}{\partial \omega(\tau)_p} h_i V(\omega(\tau))\rho_{\mathrm{in}}\right]\right),
\end{aligned}
\tag{5.8}
$$

with $Re(\cdot)$ being the real part of the expression and $\rho_{\mathrm{in}} = |\psi_{\mathrm{in}}\rangle\langle\psi_{\mathrm{in}}|$. To evaluate A and C in varQITE, some expectation values need to be computed simulating multiple quantum circuits, more on the evaluation and form of the quantum circuits will be unveiled promptly.

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 [5]:

$$
M(\omega) = \sum_p m_p(\omega) h_p,
$$

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:

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

,

where k is the k'th gate in the trial circuit V.

Now that the steps of VarQITE are 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:

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

As mentioned earlier $\frac{\partial \omega_k(\tau)}{\partial \theta_i}$ is attained during VarQITE. This is done by differentiating Equation 5.7 with respect to the variational parameters, giving the derivative as follows:

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

which can be 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 the explicit Euler method, the following expression is given:

$$\frac{\partial \omega_k(\tau)}{\partial_{\theta_i}} = \frac{\partial \omega_k(0)}{\partial_{\theta_i}} + \sum_{j=1}^{\tau/\delta\tau} \frac{\partial \dot{\omega}_k(j\delta\tau)}{\partial_{\theta_i}} \delta\tau,$$

quite similar to the way $\omega(\tau)$ is found.

Preparation of Gibbs states with VarQITE

Let us have a look at how it is possible to prepare the Gibbs state ρ^{Gibbs} using VarQITE. The Gibbs state expresses a probability density operator. The density operator defines the configuration space of a system in thermal equilibrium [46] [5]. So how is ρ^{Gibbs} prepared? There are multiple different ways proposed which consist of strategies like Quantum Phase Estimation [47–49], evaluation of quantum thermal averages [50–52], and more. As said earlier VarQITE will be used for this task, due to the fact that this scheme is compatible with near-term quantum computers in addition to having a clear knowledge of the needed ancilla qubits.

So let us walk through the steps of preparing the Gibbs state ρ^{Gibbs} , also here a lot of the walkthrough will follow [5]. The 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:

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

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

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

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

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

with the superscript denoting which sub-system the operator acts upon. Since sub-system b will be a maximally mixed state [5], it will act as an ancillary qubit system, where tracing the sub-system gives the following:

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

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

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

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

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

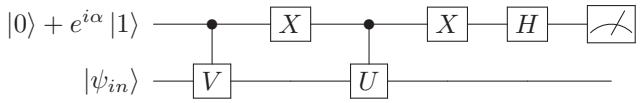
and we arrive at the Gibbs state approximation using VarQITE.

Evaluation of expression- A and C and their gradients

The expressions A and C from Equation 5.8 used in varQITE are a couple of expressions which is composed of the following form:

$$\text{Re}\left(e^{i\alpha}\,\text{Tr}\left[U^\dagger V \rho_{\text{in}}\right]\right). \tag{5.9}$$

It is possible to evaluate the two expressions in Equation 5.8 by simulating the following circuit, according to [5] and [10][53][54]:



To make the phase factor $e^{i\alpha}$ go along with the expressions in Equation 5.8, the first qubit is initialized a bit differently for the two expressions. Expression A is initialized with an H gate, while C is initialized 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:

$$U(\omega) = R_{\sigma_l}(\omega),$$

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

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

Further on the terms of the derivatives of expression A and C can be written on the same form as Equation 5.9. The derivative of expression A can be expressed as follows:

$$\partial_{\theta_i} A_{p,q}(\tau) = \sum_s \frac{\partial \omega_s(\tau)}{\partial_{\theta_i}} \operatorname{Re}\left(\operatorname{Tr}\left[\left(\frac{\partial^2 V^\dagger(\omega(\tau))}{\partial\omega_p(\tau)\partial\omega_s(\tau)}\frac{\partial V(\omega(\tau))}{\partial\omega(\tau)_q} + \frac{\partial V^\dagger(\omega(\tau))}{\partial\omega(\tau)_p}\frac{\partial^2 V(\omega(\tau))}{\partial\omega_q(\tau)\partial\omega_s(\tau)}\right)\rho_{\text{in}}\right]\right). \tag{5.10}$$

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

$$\begin{aligned}\partial_{\theta_j} C_p(\tau) = &- \operatorname{Re}\left(\operatorname{Tr}\left[\frac{\partial V^\dagger(\omega(\tau))}{\partial\omega(\tau)_p} h_j V(\omega(\tau))\rho_{\text{in}}\right]\right) \\ &- \sum_{i,s} \theta_i \frac{\partial\omega_s(\tau)}{\partial_{\theta_j}} \operatorname{Re}\left(\operatorname{Tr}\left[\left(\frac{\partial V^\dagger(\omega(\tau))}{\partial\omega_p(\tau)} h_i \frac{\partial V(\omega(\tau))}{\partial\omega_s(\tau)} \right.\right.\right. \\ &\left.\left.\left. + \frac{\partial^2 V^\dagger(\omega(\tau))}{\partial\omega_p(\tau)\partial\omega_s(\tau)} h_i V(\omega(\tau))\right)\rho_{\text{in}}\right]\right).\end{aligned} \tag{5.11}$$

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

5.3.2 Encoding feature values into the VarQBM

When utilizing the VarQBM for problems regarding supervised learning the features of the dataset are required to get encoded into the Boltzmann machine. In this thesis two ways will be utilized and investigated, one way called bias encoding-, and another called neural network encoding of features.

Bias encoding

The first approach suggested by [5] is done by appending feature values of classical samples as a bias to the Hamiltonian coefficients, in other words, each Hamiltonian coefficient θ will be a function of variational parameters and sample features as follows:

$$\theta = f(\boldsymbol{\theta}, \boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x},$$

with $\boldsymbol{\theta}$ being a vector of trainable parameters and $\boldsymbol{x}$ being a vector containing the features of a given sample. The given method will be referred to as a Bias-VarQBM.

Neural network encoding

The second approach is by having each coefficient in the Hamiltonian of the VarQBM represented as the output value of some neural network. In other words, the feature values of the samples are utilized as input values of a feed-forward neural network. The output values is then used as Hamiltonian coefficients in the VarQBM. The given method will be referred to as a NN-VarQBM.

5.4 Jordan-Wigner Transformation

Taking a base of the second quantized electronic Hamiltonian from Equation 4.5, it is often useful to rewrite a Hamiltonian making it suitable to be simulated on quantum computers. This is done by doing a mapping of creation and annihilation operators to quantum gates, by utilizing n qubits to represent n spin-orbitals[55]. Since each qubit represents a spin orbital, it is quite intuitive to have $|0\rangle$ representing an empty orbital, while $|1\rangle$ represents a filled orbital. The creation operator σ^+ and annihilation operator σ^- is defined as follows:

$$\sigma^+ = \frac{1}{2}(X - iY) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix},$$
$$\sigma^- = \frac{1}{2}(X + iY) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

Which gives the following results when the creation operator acts on the basis states:

$$\sigma^+|0\rangle = |1\rangle, \quad \sigma^+|1\rangle = 0,$$

and the following when the annihilation operator acts on the same basis states:

$$\sigma^-|1\rangle = |0\rangle, \quad \sigma^-|0\rangle = 0.$$

Now, these raising- and lowering operators are not anti-symmetric as is necessary due to dealing with fermions. This is simply solved by extending the current operators by a factor of Pauli-Z gates, yielding the following creation and annihilation operators for the Jordan-Wigner mapping:

$$a_i^\dagger = \sigma_i^+ \bigotimes_{j<i} Z_j,$$
$$a_i = \sigma_i^- \bigotimes_{j<i} Z_j,$$

with the Pauli-Z gate acting on all qubits leading up to qubit i which is operated on.

5.5 Expectation Values of Hamiltonian Quantum Circuits

A trial circuit represents some ansatz which can be optimized, by searching for a minimized expectation energy. The ansatz needs to be simulated together with the Hamiltonian, to compute the expectation value of the energy. So how is expectation energies computed with the use of quantum circuits, when measuring qubits only provides sampling probabilities between 0 and 1?

In the context of Boltzmann machines, all Hamiltonians compatible with Boltzmann distributions can be used and written as a sum of Pauli products[5]. Computing the expectation energy of a Hamiltonian given on the following form is 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 a Hamiltonian act on a quantum state $|\psi\rangle$ represented as a quantum circuit is used to find the expectation energy the following way:

$$\langle E \rangle = \langle \psi | H | \psi \rangle = \sum_{i=0}^{p-1} \theta_i \langle \psi | h_i | \psi \rangle . \tag{5.12}$$

Which shows how the different terms of a Hamiltonian are 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 that looks as follows:

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

When measuring quantum circuits, the measured quantum states are forced out of superposition and into some eigenstate, therefore a quantum circuit is 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 5.12 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 $Z_0 Z_1$ meaning the first and second qubit consists of a Z gate each, there are four possible quantum states when measuring each of the qubits, since each qubit can take values of 0 or 1. The eigenvalue of each qubit is multiplied by each other 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 the same way as for Pauli-Z gates.

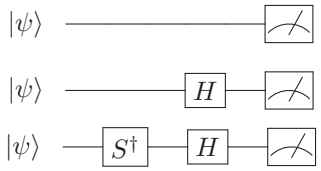
The easiest way to explain this process is probably by showing an example of circuits used. Let us assume some Hamiltonian is given by the following:

$$H = Z + X + Y,$$

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.

PART IV

Implementation

CHAPTER 6

Method

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 in this thesis is qiskit[56] which is an open-source python library made by IBM, making it possible to simulate a quantum computer running quantum circuits on a classical computer, with and without simulated sampling probabilities.

In this thesis, the circuits will be simulated using qiskit's statevector-simulator, which gives the exact sampling probabilities of simulated quantum circuits, in addition to giving a speedup of the circuit simulations due to using a statevector simulator removing the need for sampling states, hence quantum circuits only needs to be simulated once. Even though real quantum hardware contains noise, and the most realistic simulations would be to run the simulations using real quantum hardware or simulators mimicking real life quantum hardware, the algorithm is a more important aspect of the implementations which is why the statevector simulator is used.

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 into 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 [7]. 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 [7], which will serve as a transaction dataset in this thesis. The transaction dataset is published

here[57].

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 transaction 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 transaction dataset using one hidden and one visible qubit 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 [5]. The ZIP codes are discretized and relabeled into 0,1 or 2 corresponding to if the ZIP code belongs to the east, central, or west of the US accordingly. In addition, the time of the transactions and amount of purchase were also classified within the same bins. The way they are discretized can be seen in Table 6.1. The remaining discretization was applied to the MCCs. Studying MCCs with values less than 10.000 opens up the door to split the MCC feature into 10 more features containing 0's and a 1, according to the thousandth of the MCC.

Table 6.1: The time of purchase was discretized in groups of transactions completed between certain time frames, in addition to transaction amounts corresponding to different ranges as well. The ZIP codes were also part of the preprocessing, all of the features were sorted into bins corresponding to values of 0, 1, and 2.

Feature	Condition	Value
Time	0 AM - 11 AM	0
	11 AM - 6 PM	1
	6 PM - 0 AM	2
Transaction Amount	$< \$ 50$	0
	\$ 50 - \$ 150	1
	$> \$ 150$	2
ZIP code	East	0
	Central	1
	West	2

The dataset used consists of two sets, a training set consisting of 400 samples, and a validation set that will work as a test set consisting of 250 samples. The test will be used to see how well the final model performs using an optimal amount of training epochs. Both of the sets contain 20% fraudulent credit card transactions, which is more in terms of 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 [8] 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[9] 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 is 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 needed to be simulated low, in addition to running 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. Staying within the same range of data samples as for the transaction data, the training sets for the handwritten datasets will contain 400-500 samples. The digit dataset will use the rest of the available data as a validation set, while the MNIST dataset will use a validation set of 250 samples.

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 quite typical to scale images using min-max normalization to ensure feature values between in the range of $[0, 1]$, which will be done on both handwritten datasets.

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:

$$f(x,y) = \frac{3}{4}\exp\left\{\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right)\right\} + \frac{3}{4}\exp\left\{\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right)\right\} + \frac{1}{2}\exp\left\{\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right)\right\} - \frac{1}{5}\exp\{(-(9x-4)^2 - (9y-7)^2)\}.$$

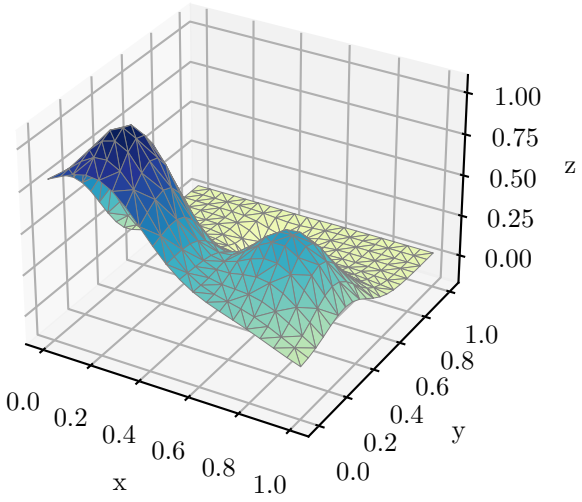


Figure 6.1: Shape of Franke's function which will be used as a regression goal.

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. Franke's data is 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 in the following way:

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

With d being the degree of the polynomial. Making the degree higher leads to more complicated equations which naturally give a more precise fit to the training data and therefore increase the possibility of overfitting. It is worth mentioning that 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 $\epsilon \sim \mathcal{N}(0, \sigma)$ is also added to Franke's function in 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 quite 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 normalization is applied to Franke's data. This is crucial to have all target variables lay within the span of possible sampling probabilities. The data used consists of $20x20 = 400$ samples which are divided into a train- and validation/test with a ratio 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

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

After the Hamiltonian and the trial circuit are ready to go, the circuits are sent into the VarQITE 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 Initialization of Hamiltonian coefficients

When doing generative learning, and generating target probability distributions, the Hamiltonian coefficients are drawn from a uniform distribution from the interval of $[-1, 1]$ in accordance with [5].

When doing discriminative learning, the input of samples must be encoded into the VarQBM. This is done using two approaches, first approach is done by following the bias dot product method explained in section 5.3.2, the Hamiltonian vector which is multiplied by feature values are drawn from a uniform distribution the same way as for generative learning.

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 initialize the Hamiltonian

When initiating a DNN multiple parameters affect the initialization and thereby the output. Firstly the weights and bias of the network need to be initialized,

there are lots of different ways to do this, but in this thesis Xavier normal- and uniform[58] and He normal- and uniform[59] 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 6.2

Table 6.2: 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	$f(x) = \max(0.01x, x)$

Another thing that is one of the most important things to define in a neural network is the number of layers and hidden nodes within each layer. 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 [60, ch. 10] to test. The pyramid rule is a quite shallow direction of choosing the size of 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 is 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 quite similar approach by defining a variable $r = \sqrt[3]{\frac{n}{m}}$, which gives the following hidden layer sizes:

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

Even though 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 quantum imaginary time evolution

The preparation of Gibbs states through VarQITE 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. The reason that all circuits can be created before the training starts, is because the terms in Equation 5.8, Equation 5.10 and Equation 5.11 containing circuit evaluations, reuses the structure of the circuits each VarQITE step. Even

though the structure of the circuits are reused each time step, the rotational parameters still need to be updated since the parameters are evolving through VarQITE. All circuits can therefore be initialized before the time evolution starts, and bind new parameters to the gates after each timestep.

The initial trial circuit $V(\omega(0))$ is sent into the VarQITE 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, matrix $A(t)$ and vector $C(t)$ from Equation 5.8, are computed according to subsection 5.3.1 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 the current time step. Due to matrix $A(t)$ having a possibility of being singular, hence having infinitely many solutions, regularization methods will be used. Dealing with ill-conditioned systems of equations is 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 VarQITE process. After each time step ω and $\frac{\partial\omega}{\partial\theta}$ are updated using explicit Euler methods. An outline of the algorithm from [5] can be seen in 1

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 $\omega(\tau)$, $\partial\omega(\tau)/\partial\theta$

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

After finding the Gibbs states from the VarQITE process, tracing out the ancillary system, the Boltzmann distribution is 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 is quite easily done. When doing multiclass labeling, the Boltzmann distribution is used as a one hot encoded vector, meaning that each index of the distribution represents one label, giving the index of largest probability the final label. In multilabel classification the number of classes the algorithm is able to represent 2^n, scales exponential with the number of visible qubits 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 quite 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.

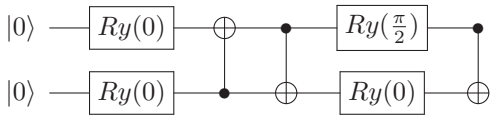
After a prediction is done within the VarQBM, the gradients of the probabilities corresponding to each configuration are computed using the parameter shift rule[43]. The derivatives are taken further to compute the gradients of the loss with respect to each Hamiltonian parameter. Now all the necessary 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 VarQITE, some form of SGD optimization is executed

6.3.4 VarQBM ansatzes

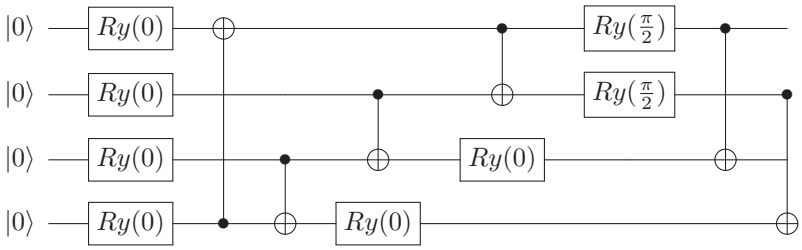
The VarQBM consists of two ansatzes: The Hamiltonian ansatz which will contain the learnable parameters and the trial circuit. Primarily in this thesis, mainly two sets of ansatzes will be used unless else is stated: a 1-qubit Hamiltonian paired with a 2-qubit trial circuit, and a 2-qubit Hamiltonian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

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

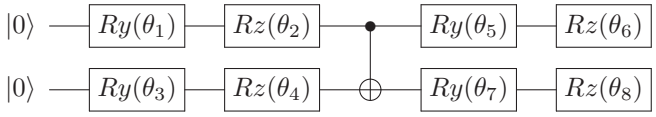
with the first and second Pauli gate referring to the first and second qubit respectively. The trial circuits mainly used in this thesis looks as follows, the first trial circuit used in combination with H_1:



and the trial circuit used in combination with H_2:



When utilizing VarQITE to find the ground state energy, another ansatz from [10] was used. The ansatz goes as follows:



6.3.5 Jordan-Wigner transformed Hamiltonian

The Jordan-Wigner transformed Hamiltonian will be found using OpenFermion[61] which is an open-source software library enabling simulations of quantum chemistry of both bosonic and fermionic systems. In addition, the open-source package psi4[62] which supplies a variety of quantum chemical methods, will be used to find the exact ground state energy of the hydrogen system, to be used as a benchmark when finding the ground state using approaches of simulating quantum computers. The Jordan-Wigner transformed hydrogen Hamiltonian is given as follows:

$$\begin{aligned} H_{H_2} = h_1 &+ h_2 ZZII + h_2 IIZZ + h_3 ZIZI + h_3 IZIZ + h_4 IZZI \\ &+ h_5 ZIIZ + h_6 XXXX + h_6 XXYY + h_6 YYXX \\ &+ h_6 YYYY + h_7 ZIII + h_7 IIIZ + h_8 IZII + h_8 IIZI, \end{aligned}$$

with the coefficients h_i dependent on the bond length between the atoms and the order of Pauli gates corresponding to the order of the qubits. The Hamiltonian can further be reduced to a 2-qubit Hamiltonian according to the following article [63] which states that the Hamiltonian of the Hydrogen molecule can be written as follows:

$$H_{H_2} = g_1 I + g_2 XX + g_3 ZZ + g_4 ZI + g_5 IZ,$$

with the coefficients given as follows:

$$g_1 = h_1 - 2h_2, \quad g_2 = 4h_6, \quad g_3 = -2h_3 + h_4 + h_5,$$
$$g_4 = h_7 - h_8, \quad g_5 = h_7 - h_8.$$

6.4 Restricted Boltzmann Machine

When referring to a classical Boltzmann machine or an RBM in this thesis, a classical computational approach is taken without the use of quantum algorithms. Due to problems of convergence well known when using regular Boltzmann machines, the classical approaches of Boltzmann machines in this thesis are 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[64] 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. To ensure that the visible states lays within the range of $[0, 1]$, the data is scaled using min-max normalization. The Scikit learn class automatically initializes the variance as 0.01 prior to training.

6.4.2 Gaussian-Binary RBM

When utilizing a classical RBM to solve a quantum mechanical problem, a Gaussian-Binary RBM using the NQS as a wave function in the Monte Carlo simulation is implemented using C++. The flow of the program of the restricted Boltzmann machine goes as follows. The implementation starts by determining the desired parameters like bond length, learning rate, and the number of hidden nodes. The restricted Boltzmann machine simulation then starts off by initializing the values of the nodes, biases, and weight matrix, where the visible nodes represent the positions of the electrons. The electrons are distributed using a normal distribution, with a deviation σ chosen beforehand.

After everything is initialized, the simulation starts and takes several Gibbs steps moving the electrons around. After all the steps are done, the bias and weights are updated according to SGD. This equals one cycle of the RBM. The RBM cycles carry on minimizing the energy. Then one final simulation is done with the optimized values, computing the final energy of the system.

6.4.3 Implementation of the Stochastic Gradient Descent algorithm

Implementation of gradient descent is quite straightforward. Using subsection 2.3.4 results in the new values for the visible biases, hidden biases, and the weights. The length of the simulation can be long or short depending on

how hard it is for the gradient descent to converge towards optimal parameters. The optimization method runs the simulation until one of two options is ticked. Either an amount of RBM cycles are reached, or the derivative of the parameters is less than a tol of 10^{-5}.

6.4.4 The Blocking Method

The blocking method is utilized to estimate the deviation of the sampled energies accumulated during the simulation. The blocking method is based on applying so-called blocking transformations on samples of data, rooting out correlations until the deviation is estimated of approximately uncorrelated samples. The statistical handling of the blocking method will be computed using the blocking method script written by Marius Jonsson[65], where a more in-depth explanation of the same method also can be seen in the same citation.

6.4.5 Modelling the Hydrogen system

The hydrogen system in this thesis consists of two hydrogen atoms at fixed positions with a distance R between them. Each of the hydrogen atoms brings one electron each to the quantum mechanical system modeled. The electrons are free to flow around the system, being affected by the forces of each nucleus in addition to the force of the other electron, these forces naturally depend on the distances between the particles in the system. A schematic sketch of the model can be seen in Figure 6.2.

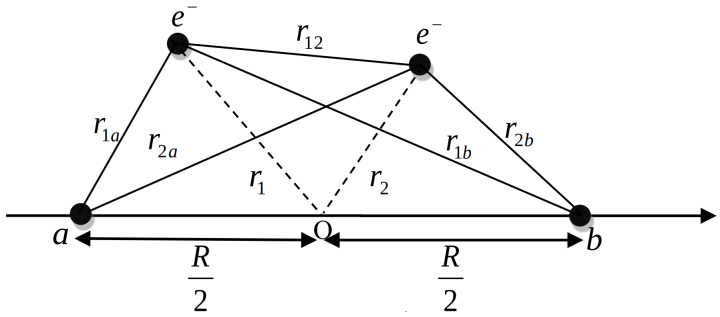


Figure 6.2: Schematic sketch of the hydrogen system consisting of two hydrogen atoms and two electrons. Figure from [66]

PART V

Results and Discussion

CHAPTER 7

Classical data

7.1 Preparation of Gibbs States

The fidelity between the analytical Gibbs states and the approximated Gibbs states through VarQITE were computed to assess the Gibbs state preparations. The Hamiltonians used in this section regarding the fidelity will be the following:

$$H_{F1} = 1.0Z,$$
$$H_{F2} = 1.0ZZ - 0.2ZI - 0.2IZ + 0.3XI + 0.3IX.$$

The fidelity was computed as a function of the regularization parameter λ of the least square methods Ridge and Lasso when finding $\dot{\omega}$ using different Hamiltonians. The results can be seen in Figure 7.1. Ridge regression was chosen as the regularization method in the further Gibbs state preparations. The regularization value was 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 final fidelity after 10 steps using a dynamical λ resulted in fidelities of 0.98 and 1.0 using H_{F1} and H_{F2} respectively.

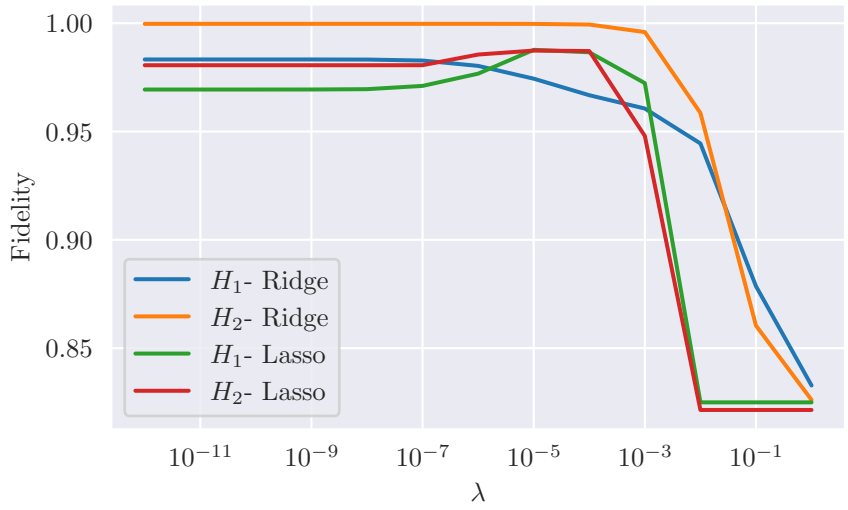


Figure 7.1: Fidelity as a function of lambda for Ridge and Lasso regression using Hamiltonian H_{F1} and H_{F2}. The number of imaginary time steps for the preparation was 10.

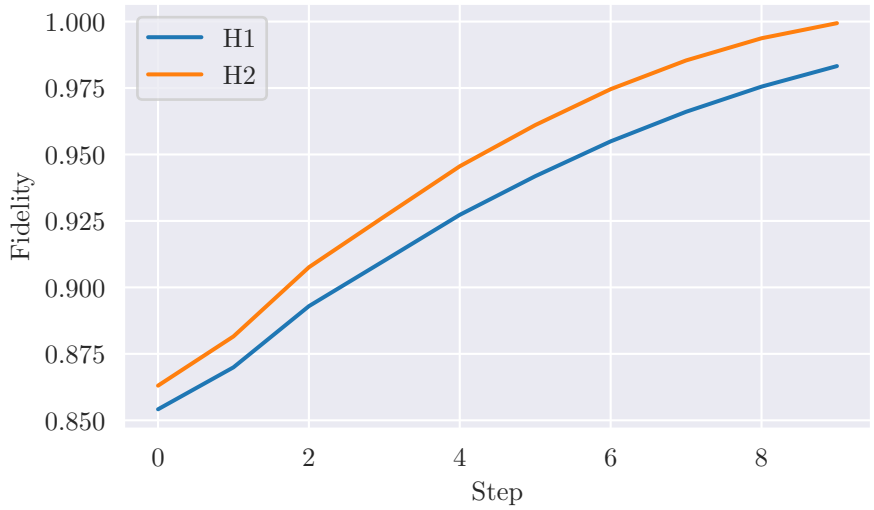


Figure 7.2: Fidelity as a function of imaginary time step using Ridge regression, varying the regularization parameter each step, choosing the parameter giving the numerical $\dot{\omega}$ closest to the analytical $\dot{\omega}$ without getting singular matrices.

7.1.1 Preparation of Gibbs states- Discussion

Having a look at Figure 7.1 shows that the highest fidelity between the analytical and approximated Gibbs state was achieved by H_{F2} using Ridge regression. For the H_{F1} Hamiltonian, on the other hand, the highest fidelity was 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 was 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 true $\dot{\omega}$, 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 desirable.

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_{F1} and H_{F2}. H_{F1} and H_{F2} resulted in fidelities of 0.98 and 1.0 respectively. The reason for H_{F2} 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 with H_{F1}. 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[5], 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 a dynamically changeable λ finds the best regularisation of each step, hence generalizing easier to other systems.

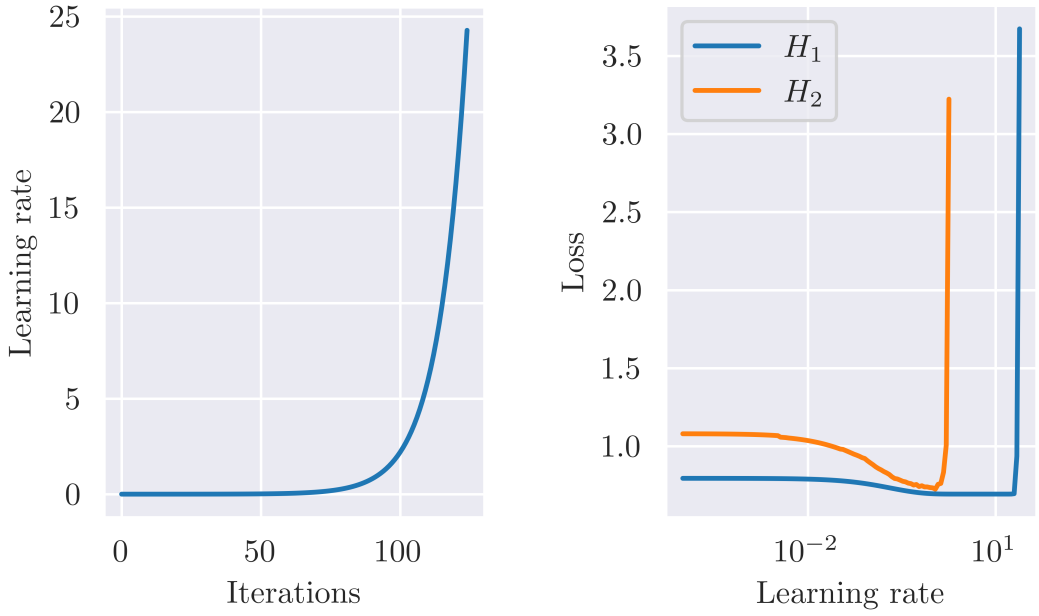
7.2 Generative Learning

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 was chosen. To get a reasonable estimate of the initial learning rate a method motivated by [67] was used. The motive of the referenced method is that the approximated optimal learning rate can be found by searching for the largest decrease in loss while increasing the learning rate. 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 of the learning rate can be seen in 7.3.

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.



(a) (b)

Figure 7.3: Learning rate investigation using SGD as the optimization technique. The learning rate was increased during training after each iteration in search of the learning rate giving the largest decrease in loss. a) Plot showing the increment of learning rate as a function of iteration, while computing the loss showed in Figure b). b) Loss as a function of increased learning rate

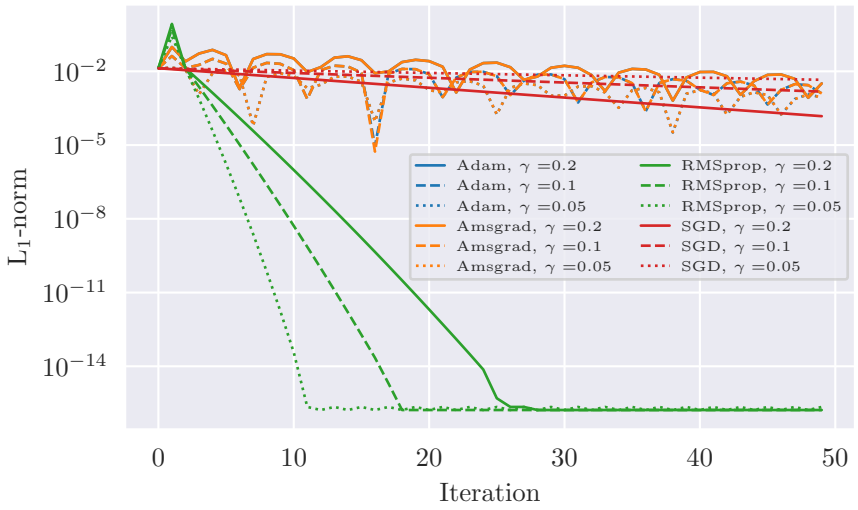


Figure 7.4: Norm as a function of iterations. Generating a probability distribution $[0.5, 0.5]$ using a 1-qubit Hamiltonian and a variety of optimization methods and learning rates. The norm is used instead of loss to perceive the trend easier. The norm is computed between the target distribution and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

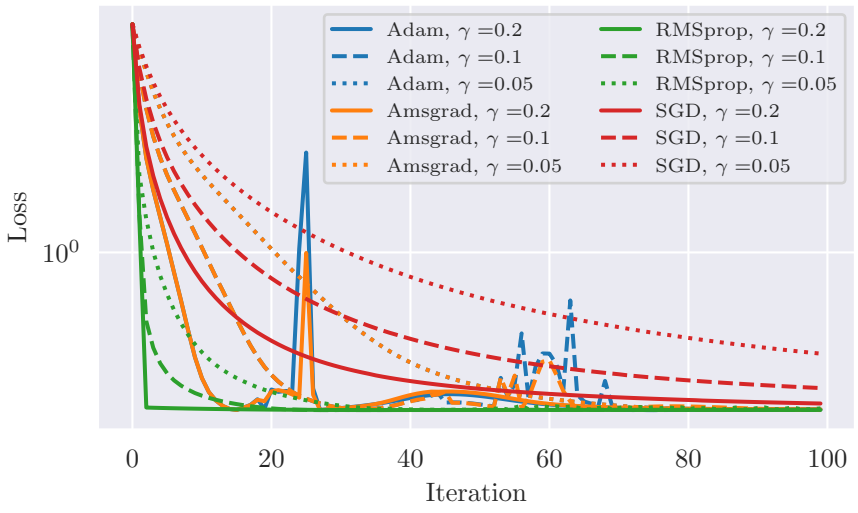


Figure 7.5: Loss as a function of iterations. Recreating the Bell state using a 2-qubit Hamiltonian and a variety of optimization methods and learning rates.

Parameter investigation - Discussion

Figure 7.3 argues that the optimal learning rate is 0.22, but this is taken as a rough 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 too big. At least while the increasing learning rate and primitively decreasing gradient are on par during training, until the blowup of loss due to the enormous learning rate.

The estimation of the learning rate was taken into account and applied to different optimization techniques. Figure 7.4 and Figure 7.5 both argue that the optimal optimization method would be RMSProp due to the fastest converging towards a minimum. The learning rate on the other hand seems to be best being set to 0.05 in the case of a 1-qubit Hamiltonian, while for a 2-qubit Hamiltonian the fastest convergence is achieved while utilizing a learning rate of 0.2, which is why the settled learning rate, in this case is decided to be 0.1. It seems like all RMSprop optimizers find the same convergence value, which argues that the value of the learning rate is not significant. It is worth mentioning that the norm is plotted in figure Figure 7.4 while the loss is plotted in Figure 7.5 due to the fastest convergence being easier to grasp for the second case with the loss function rather than the norm function.

The momentum parameter of the RMSProp optimization technique was 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, which often is used as a default value in programming libraries providing the given optimization method.

7.2.2 Generating probability distributions

Using the chosen parameters from subsection 7.2.1 were used to generate a target probability distribution using H_1 and H_2. The results from the 1-qubit Hamiltonian and 2-qubit Hamiltonian can be seen in Figure 7.6 and Figure 7.7 respectively, both Boltzmann machines are fully visible. The sampling probability of a generated Bell-state can be seen in Figure 7.8

Generating probability distributions - Discussion

The VarQBM's ability to generate probability distributions showed to be in line with theory, being able to reproduce a target distribution. Figure 7.6 showed to converge to the target distribution fairly quickly within approximately 5 iterations for all 10 seeds, where the fast convergence can be justified with the Hamiltonian only containing one optimization parameter, which makes it fairly easy to optimize fast. The 2-qubit case shown in Figure 7.7 showed quite good results as well as being able to get close to the target distribution in all cases. The best and worst generated Bell-state in Figure 7.8 by the VarQBM showed to be acceptable. Concluding that generating target distributions using VarQBMs is acceptable.

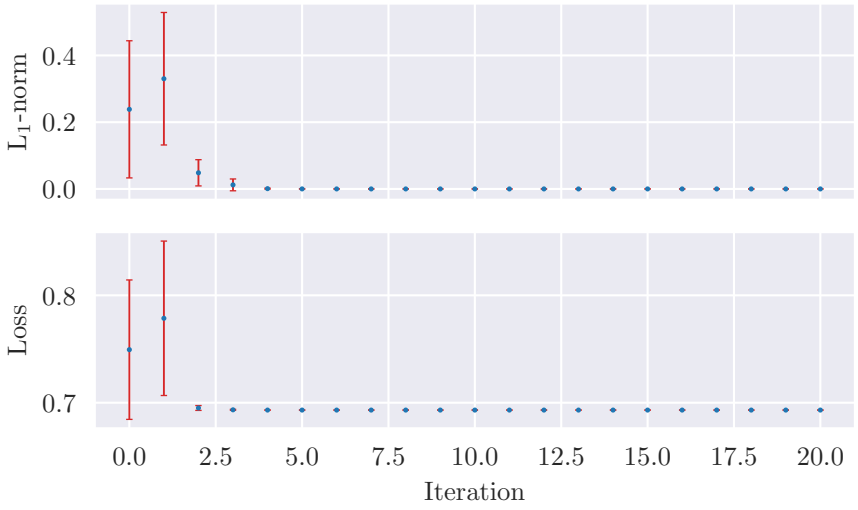


Figure 7.6: Generation of a probability distribution $[0.5, 0.5]$ using a 1-qubit Hamiltonian. The figure shows the loss and norm as a function of iteration steps. The plot represents the mean of 10 random seeds, and the bars represent the standard deviation.

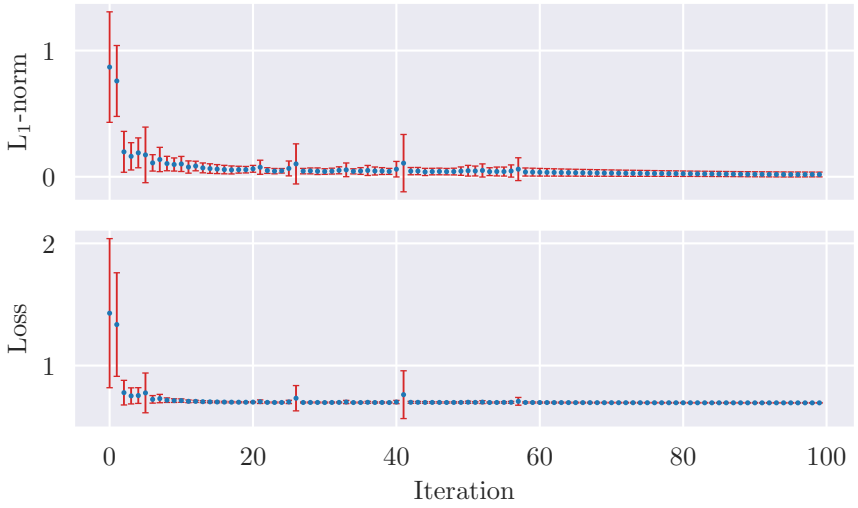


Figure 7.7: Generation of Bell-state using a 2-qubit Hamiltonian. The figure shows the loss and norm as a function of iteration steps. The plot represents the mean of 10 random seeds, and the bars represent the standard deviation.

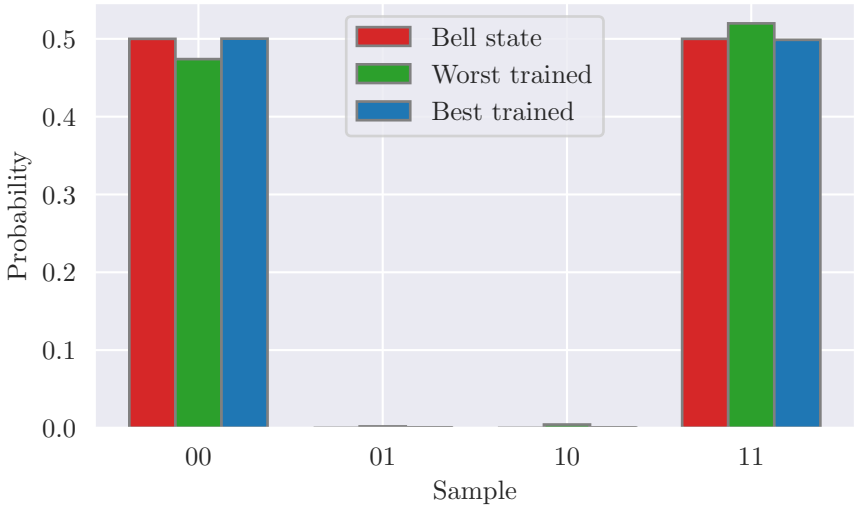


Figure 7.8: The sampling probability of the Bell-state showing the best and worst generated distributions of 10 seeds.

7.3 Transaction Dataset - Discriminative Learning

The following sections will cover the VarQBM's ability to learn relations in classical data and classify results within different classes, in addition to a comparison with classical RBMs consisting of various amounts of hidden nodes and other standard classifiers. All VarQBMs consisted of 2 qubit Hamiltonians utilizing H_2.

The first dataset that will be covered is the transaction dataset, intending to classify samples within a fraud class or outside the fraud class.

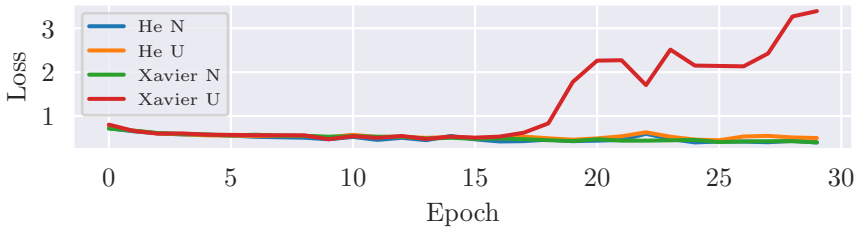
7.3.1 Parameter investigation

Results regarding the parameter investigation. The results mainly consist of loss as a function of epochs, utilizing different parameters to investigate which parameters make the model able to fit best to the data. The plots display training loss unless else is stated.

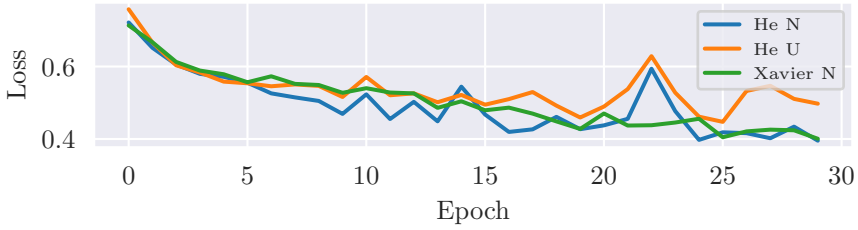
Variational Quantum Boltzmann machine

A parameter investigation was done for some of the most important learning parameters, with emphasis on the parameters in the neural network encoded VarQBM.

The initialization of weights in the neural network used to encode features into the VarQBM was computed with the results being available in Figure 7.9. The learning rate was also tested slightly which can be seen in Figure 7.10. Clamping activation functions around the nodes in the neural network making the network non-linear, was also tested in Figure 7.11.



(a)



(b)

Figure 7.9: Loss as a function of epoch training on 50 samples with a learning rate of 0.01, using different initialization methods of the weights in the neural net connected to a VarQBM. N represents a normal distribution and U represents a uniform distribution. a) 4 different initializations b) A better view of He normal, He uniform, and Xavier normal

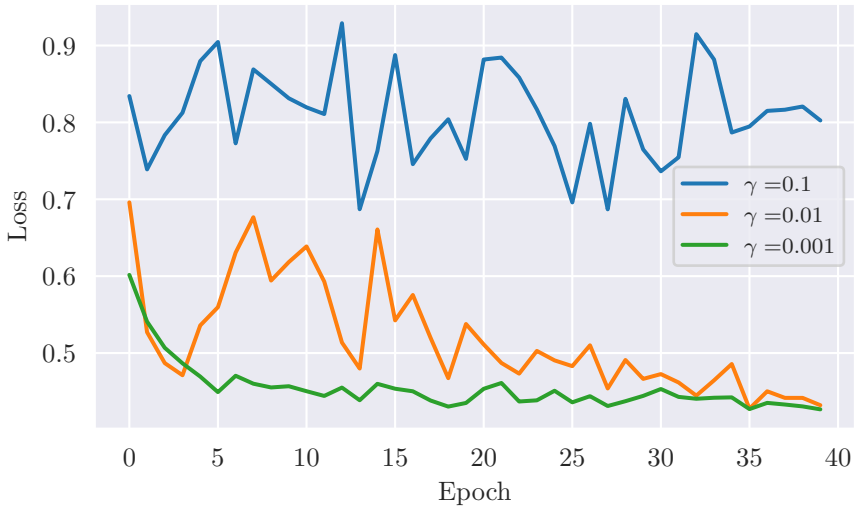


Figure 7.10: Loss as a function of epoch for different learning rates. The model consists of a 2-qubit Hamiltonian with one hidden qubit, using a neural network of two hidden layers with 12 hidden nodes each using 50 samples.

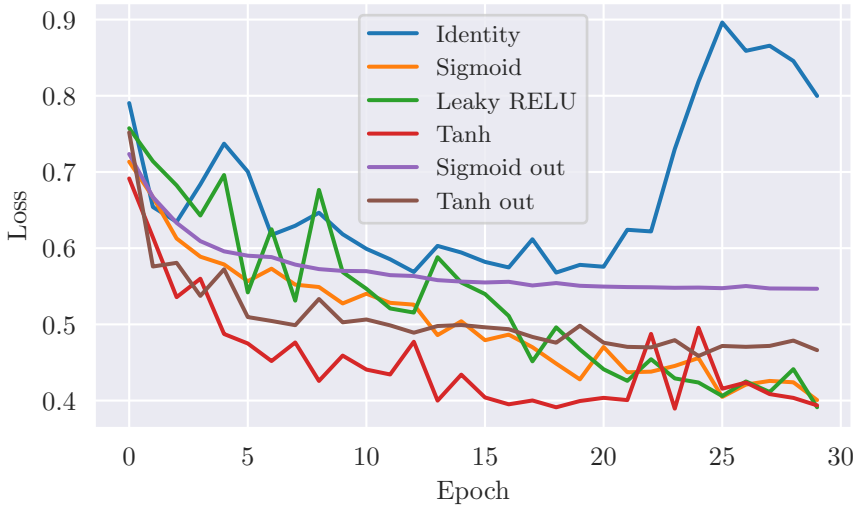


Figure 7.11: 2 qubit Hamiltonian with 1 hidden qubit, using a neural network of 2 hidden layers with 12 hidden nodes and bias, using 50 samples, computed using different activation functions.

Table 7.1: Parameters were chosen for the final assessment of the transaction dataset using a VarQBM with features encoded as a dot product of feature values and weights, and a VarQBM using a NN feature encoding approach. The output is defined as the number of Hamiltonian parameters.

Encoding	Initialization	γ	NN - size	Activation	Bias in network
Dot product	$\mu[-1, 1]$	0.01	-	-	-
NN	Xavier Normal	0.001	$[8, 5, \text{output}]$	Tanh	Yes

Restricted Boltzmann machine

A classical RBM was also used to classify the transaction dataset. This was done by utilizing an RBM with a logistic regression layer on top to take care of the classes.

The number of hidden nodes was set to 4 and 30 throughout the entire calculations of the transaction dataset. To find the optimal parameters, an exhaustive search over specified parameter values was done. The investigated parameters were 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.

The final parameters used after the search was a learning rate of 0.001, batch size of 1, and hyperparameter C of 0.5.

7.3.2 Transaction data scores

The final runs achieved using classical RBMs, the VarQBM with and without neural network feature engineering and various standard classifiers can be seen in Table 7.2. The training data consists of 400 samples, with a validation set of 250 samples, both sets contain approximately 20% fraudulent data.

Table 7.2: Final results of the transaction dataset, using the VarQBM and standard Scikit-learn classifiers for a training period of 50 epochs. The scores are collected from the optimal epoch of a validation set of 250 samples, both the training- and validation sets contained 20% fraudulent data. The dense neural network used a learning rate of 0.001, *tanh* activation within the network, and a sigmoid layer on the outputs. The K-nearest neighbors classified based on the contribution of 3 neighbors. The RBM_x method utilized x hidden nodes and went through a grid search of parameters. The rest of the parameters were default parameters of Scikit-learn. Both RBMs classified all samples as non-fraud, so the recall, precision, and F_1 score was set to 0.

Model	Accuracy	Precision	Recall	F_1 score
Bias-VarQBM	0.69	0.60	0.63	0.60
NN-VarQBM	**0.83**	**0.80**	**0.59**	**0.61**
RBM_4	0.80	0	0	0
RBM_{30}	0.80	0	0	0
DNN	0.81	0.9	0.52	0.48
OLS	0.79	0.53	0.50	0.46
Ridge	0.80	0.40	0.50	0.44
Lasso	0.80	0.40	0.50	0.44
Nearest Neighbors	0.72	0.53	0.52	0.52
Logistic Regression	0.79	0.40	0.49	0.44

The confusion matrices for the three approaches can be seen in Figure 7.12.

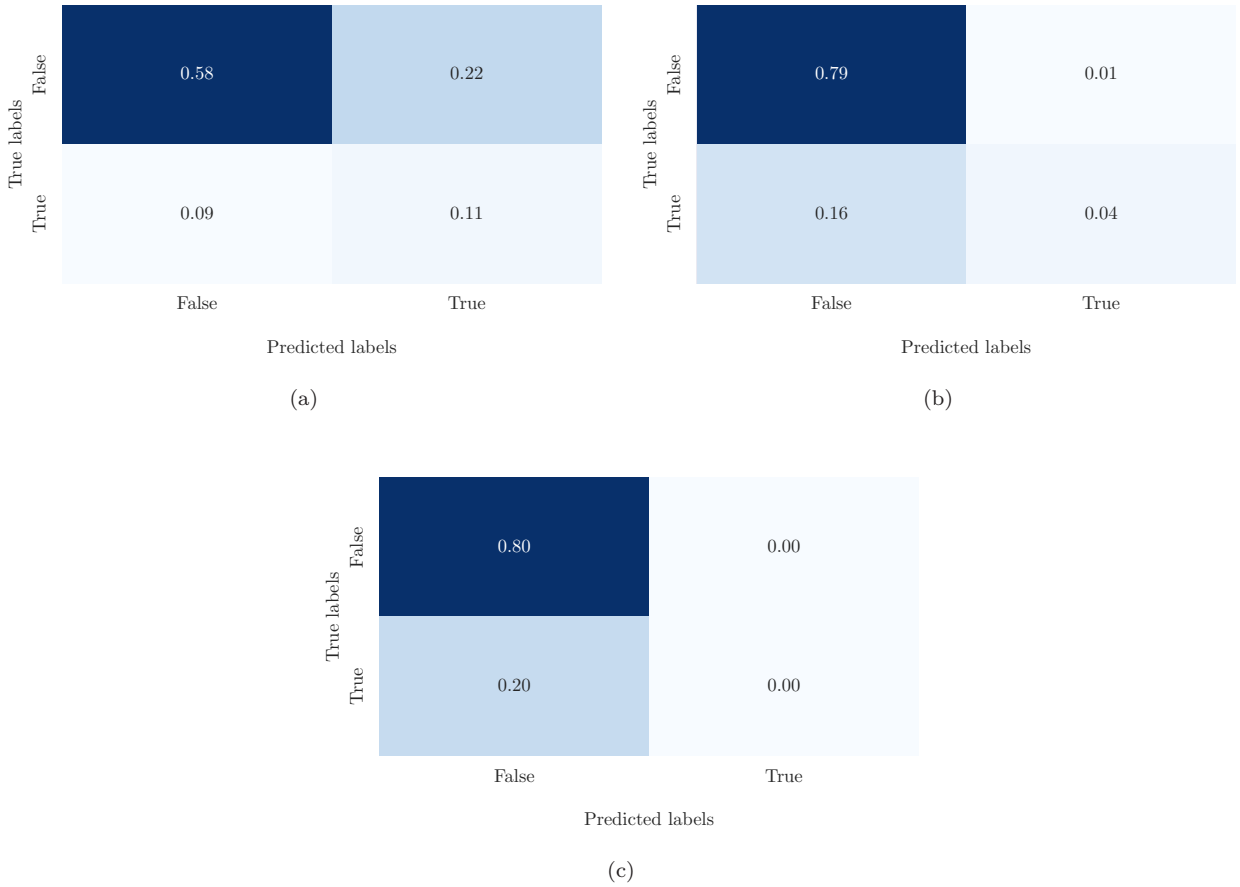


Figure 7.12: Confusion matrices showing how the different Boltzmann machine approaches classified the validation set of the transaction data. (a): VarQBM bias encoding, (b): VarQBM NN encoding, (c):RBM_4.

7.3.3 Transaction dataset - Discussion

Parameter investigation

Choosing the parameters used in the VarQBM was done quite straight forward by testing some different variables and choosing the most sensible. In Figure 7.9(a), He uniform, He normalized and Xavier normalized managed to keep the most stable, while He uniform blew up. The reason for this is probably due to some of the other parameters not assessed yet making the network method unstable, most probably the network was too large having the size of $[12, 12]$ probably making the network suffer from vanishing gradients, which can occur if the values inside the sigmoid activation function get too large or too small. The normalized Xavier initialization was used in the rest of the applications of neural network encoded VarQBM, due to giving the most stable reduction in loss which can be seen in Figure 7.9(b)

Figure 7.10 argues that a learning rate of 0.001 converges faster than 0.1 and

0.01 in addition to having a relatively smooth curve. A learning rate of 0.001 was chosen in the case of neural network encoding due to having a smaller learning rate would make it harder to escape local minima.

Having a look at Figure 7.11, two things were argued, first thing is that using tanh as an activation function in the network is preferred due to the fastest reduction in loss for most of the training which then is adjoined by the sigmoid- and leaky relu activation function at the end. And the second is that wrapping activation functions around the output nodes hinders the VarQBM from performing best as possible. This makes sense due to the Hamiltonian in the VarQBM using the output nodes of the neural network as variational parameters, therefore activation functions should not be used as a constraint directly on the Hamiltonian variational parameters, but rather inside the neural network as a support of the training of the network.

In the case of RBMs, models consisting of more hidden nodes give more complex models and increases the need for computational resources. By trial and error it was observed that all sensible numbers of hidden nodes classified the transaction data as fraud. A number of 30 hidden nodes were chosen to utilize a more complex RBM, in addition to an RBM consisting of 4 hidden nodes to have a more clear comparison between hidden nodes and the 4 hidden qubits utilized in the VarQBM. The rest of the parameters while mapping out the number of hidden nodes were default values of the Scikit-learn library. Further on the rest of the parameters were chosen from a grid search, resulting in sensible parameters. RBMs consisting of 4 and 30 hidden nodes were kept used on all classical datasets.

Final scores

Having a look at the scores in Table 7.2, the VarQBM using NN-encoded features reached an accuracy of 0.83, and performed greater than the VarQBM with bias encoded features which gained an accuracy of 0.69. The NN-VarQBM also outperformed the classical RBM using both 4- and 30 hidden nodes, by 3% which gained an accuracy of 0.80 due to all transaction samples being labeled as non-fraud using the RBMs. The F_1 scores were also compared. The NN-VarQBM got a F_1 score of 0.61 having fairly high precision and a modest recall compared to the other methods. The Bias-VarQBM achieved an F_1 score of 0.60 which is only 0.01 less than the NN-VarQBM. Comparing the runs to the other machine learning methods in the same table, showed that the NN-VarQBM performed best in all cases assessing the accuracy, while the accuracy was worst using the Bias-VarQBM, even though the F_1 score was the second highest after the NN-VarQBM. The reason for this is that the F_1 score takes all classes into account, which makes it a more appropriate assessing tool regarding imbalanced datasets. The method achieving the best F_1 score after the quantum Boltzmann machines were the nearest neighbor method achieving a F_1 score of 0.52, which is 0.09 less than the NN-VarQBM. It is also worth mentioning that attaching the VarQBM to a neural network improved the results of the neural network by an accuracy of 0.01 and the F_1 score by 0.13.

Having a look at how the samples were labeled in the confusion matrices in

Figure 7.12. The easiest confusion matrix to notice is the RBM's in (c) showing that all samples were classified as non-fraudulent, thereby giving an accuracy of 0.80. The NN-VarQBM managed to classify 99% of the non-fraudulent data correct, and 20% of the fraudulent data correct, which certainly is the most difficult class to label. The Bias-VarQBM managed to classify 72.5% of the non-fraudulent data correct and was the method that classified the most fraudulent samples correct with 55.0% correctly labeled fraudulent samples. The VarQBM proved itself to be a fine candidate for labeling fraud compared to a classical RBM, but still has room for improvement.

7.4 Discriminative Learning-Handwritten Image Recognition

In this section the VarQBM and RBM will be used to classify two handwritten datasets, the 8**x**8 pixel digit dataset and the 28**x**28 pixel MNIST dataset.

7.4.1 Parameter investigation

Variational Quantum Boltzmann machine

Some of the VarQBM parameters found were brought along from the transaction runs, this includes the initialization of the NN-VarQBM and Bias-VarQBM, in addition to using tanh as an activation function inside the NN. The pyramid rule was followed for the 8**x**8 pixel samples, while for the 28**x**28 pixel samples a smaller network of two layers consisting of 32 nodes each was used, due to larger networks showing a tendency of becoming a bit unstable during training in our algorithm. Finding the new parameters was done doing quite the same procedure as for the transaction dataset. The results of the exploration can be seen in Appendix C. A summary of the final parameters used for the handwritten datasets' classification can be seen in Table 7.3.

Table 7.3: Parameters were chosen for the final assessment of the handwritten image dataset using a VarQBM with features encoded as a dot product of feature values and weights, and a VarQBM using a NN feature encoding approach. All NN-encoding approaches are using tanh as activation functions within the neural network in addition to bias nodes. The output is defined as the number of Hamiltonian parameters.

Features	Encoding	Initialization	NN - size	γ
8**x**8	Dot product	$\mu[-1, 1]$	-	0.01
8**x**8	NN	Xavier Normal	$[23, 8, \text{output}]$	0.01
28**x**28	NN	Xavier Normal	$[32, 32, \text{output}]$	0.01

Restricted Boltzmann machine

Solving the digit dataset using a restricted Boltzmann machine with a logistic regression layer on top. The number of hidden nodes was chosen to be 4 and 30, to see how the RBMs of various sizes perform. Then an exhaustive search was

done for the transaction data was done searching the same parameter range on the 8**x**8 pixel samples and brought along to the 28**x**28 pixel samples.

The same parameters were used for both image recognition datasets. The final parameters chosen were a learning rate of 0.05, batch size of 1, and hyperparameter C set to 100.

7.4.2 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 7.4, with the confusion matrices in Figure 7.13

Table 7.4: Final validation results from classifying handwritten digits of 4 classes using the VarQBM and standard Scikit-learn classifiers for a training period of 500 samples for 40 epochs for 8**x**8 pixel images and 400 samples for 50 epochs for 28**x**28 pixel images. The scores are collected from the optimal epoch of the validation set. The dense neural network used a learning rate of 0.01 and *tanh* activation functions within the network. The K-nearest neighbors classified based on the contribution of 3 neighbors. The RBM_x method utilized x hidden nodes and went through a grid search of parameters. The rest of the parameters were default parameters of Scikit-learn.

Model	Features	Accuracy	Precision	Recall	F_1 score
Bias-VarQBM	8**x**8	0.56	0.54	0.56	0.53
NN-VarQBM	8x8	**0.99**	**0.99**	**0.99**	**0.99**
RBM_4	8**x**8	0.91	0.91	0.91	0.91
RBM_{30}	8**x**8	0.97	0.97	0.97	0.97
DNN	8**x**8	0.98	0.98	0.98	0.98
OLS	8**x**8	0.97	0.97	0.97	0.97
Ridge	8**x**8	0.97	0.97	0.97	0.97
Lasso	8**x**8	0.98	0.98	0.98	0.98
Nearest Neighbors	8x8	**0.99**	**0.99**	**0.99**	**0.99**
Logistic Regression	8x8	**0.99**	**0.99**	**0.99**	**0.99**
NN-VarQBM	28**x**28	0.83	0.80	0.59	0.61
RBM_4	28**x**28	0.84	0.88	0.85	0.85
RBM_{30}	28**x**28	0.92	0.93	0.92	0.92
DNN	28x28	**0.96**	**0.96**	**0.96**	**0.96**
OLS	28**x**28	0.64	0.66	0.67	0.64
Ridge	28**x**28	0.88	0.87	0.88	0.88
Lasso	28**x**28	0.94	0.94	0.94	0.94
Nearest Neighbors	28**x**28	0.93	0.93	0.93	0.93
Logistic Regression	28**x**28	0.93	0.93	0.93	0.93

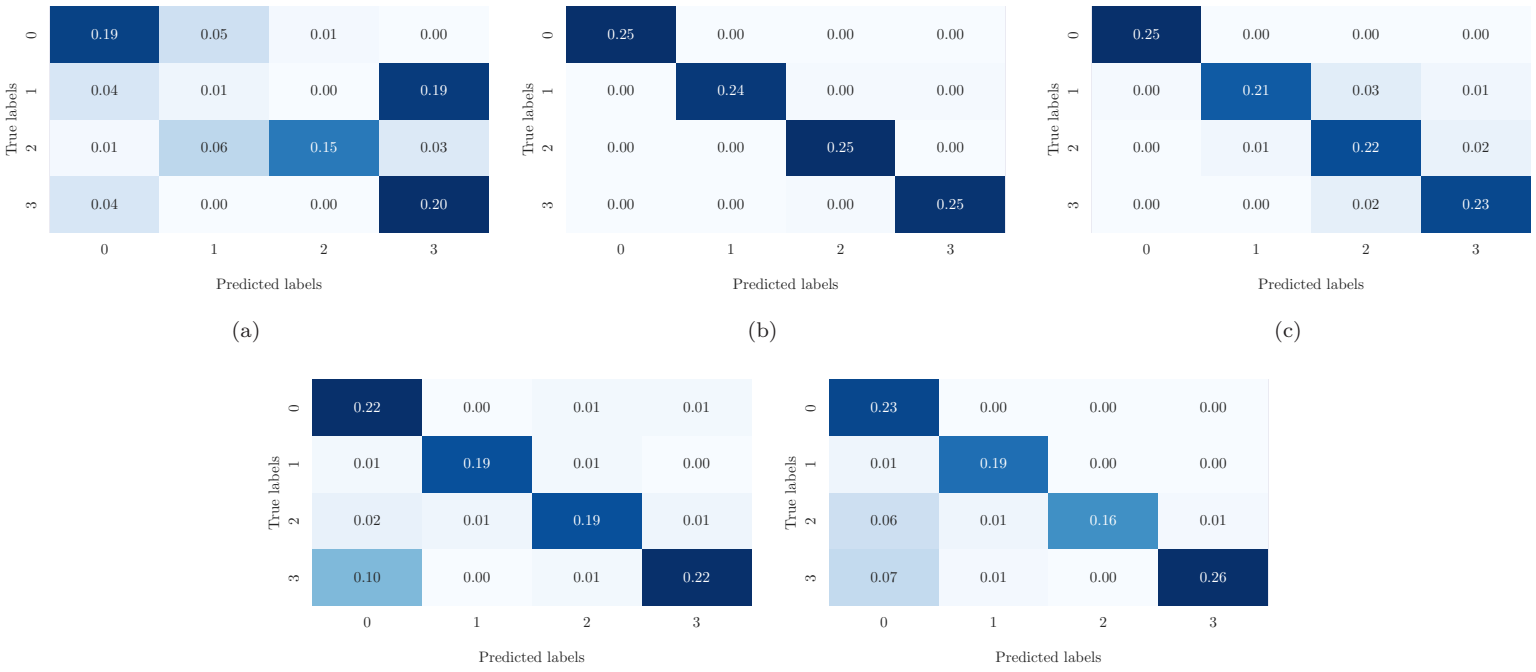


Figure 7.13: Confusion matrices showing how the different Boltzmann machine approaches investigated, and labeled the validation set of the handwritten image datasets. (a): 8**x**8 features - VarQBM bias encoding, (b): 8**x**8 features - VarQBM NN encoding, (c): 8**x**8 features - RBM_4, (d): 28**x**28 features - VarQBM NN encoding, (e): 28**x**28 features - RBM_4

7.4.3 Handwritten image recognition - Discussion

Final scores

Starting with the 8**x**8 featured image dataset in Table 7.4, the VarQBM using NN-encoded features reached an accuracy of 0.99, and performed better than the VarQBM with bias encoded features which gained an accuracy of 0.56. The reason for the low accuracy in the case of using bias-encoding argues that 8**x**8 features are too many features to handle for the mentioned encoding scheme, to transform the features into a compressed representation. The NN-VarQBM achieved higher accuracy than the classical RBM by 8% for the RBM consisting of 4-hidden nodes and 2% compared to the RBM consisting of 30 hidden nodes. Also, in this case the accuracy was improved by attaching an RBM to the neural network, even though the accuracy was only improved by 2%, it argues that the neural network encoding works as an encoding scheme for the VarQBM.

Samples containing 8**x**8 features in Table 7.4, shows that NN-VarQBM, the nearest neighbor method and logistic regression performed best. This is probably due to the digit dataset having really distinct features, in other words having really clear trends corresponding to each class, being a relatively easy dataset

to classify. Compared to the transaction dataset has quite few features, but still has complex trends connected to each class, making it hard to distinguish between the different classes.

The F_1 scores were also compared. The NN-VarQBM got a F_1 score of 0.99 outperforming the Bias-VarQBM which gained a F_1 score of 0.53 while the RBMs consisting of 4 and 30 hidden nodes achieved scores of 0.91 and 0.97 respectively.

It is worth having a look at how the different methods classified the samples. From Figure 7.13(a)-(c) it is easy to see that the Bias-VarQBM was a bit unsure about the images and classified them a bit over the place. Even though it showed itself fairly strong classifying $3's$ and $0's$ correct, with almost non of the $1's$ labeled correct. The NN-VarQBM showed a quite straightforward confusion matrix showing that almost all samples were labeled correctly. Moving over to the RBM confusion matrix utilizing four hidden nodes shows that the classes were mostly classified correct, with all 0's being labeled correct, and the classes representing the digits 1 - 3 being labeled mostly correct with a 1-3% labeling across the same classes. No classes except the 0-samples were labeled as 0, which shows that the model handled the 0's perfectly.

Moving to the 28**x**28 MNIST dataset, the NN-VarQBM showed less promising results than the classical machine learning methods. The reason for this is most likely due to the same reason the Bias-VarQBM being outperformed in the 8**x**8 pixel case, losing too much information to represent the space of the data, during the compression of data into a 2-qubit Hamiltonian. The method achieving the best result was achieved by utilizing the neural network which achieved an accuracy of 0.96. The RBMs achieved accuracies of 0.84 and 0.92 utilizing 4 and 30 hidden nodes respectively. The NN-VarQBM managed to achieve an accuracy of 0.83. Even though the accuracy still has room for improvement, the VarQBM shows promising results remembering that the VarQBM utilized a 2 qubit Hamiltonian to express 784 compressed features, which opens up a thought of there might being a proportionality between the number of features able to be compressed into a certain number of qubits, which would be a quite natural future topic as well investigating this line of scalability.

Moving over to the confusion matrices in Figure 7.13(d)-(e). The VarQBM showed to be fairly certain in its predictions, except for a bit of 3's being classified as 0's, which can be discussed as being due to a written three having tendencies of having parts of the images looking like parts of a zero. The classical RBM utilizing four hidden nodes classified samples quite consistent along with all classes, with some misclassifications from place to place.

7.5 Franke's Function - Regression Learning

The VarQBM was used to train a model to reconstruct Franke's function, using neural network encoding. The results of the run utilizing two distinct learning rates of 0.01 and 0.001 can be seen in Figure 7.14.

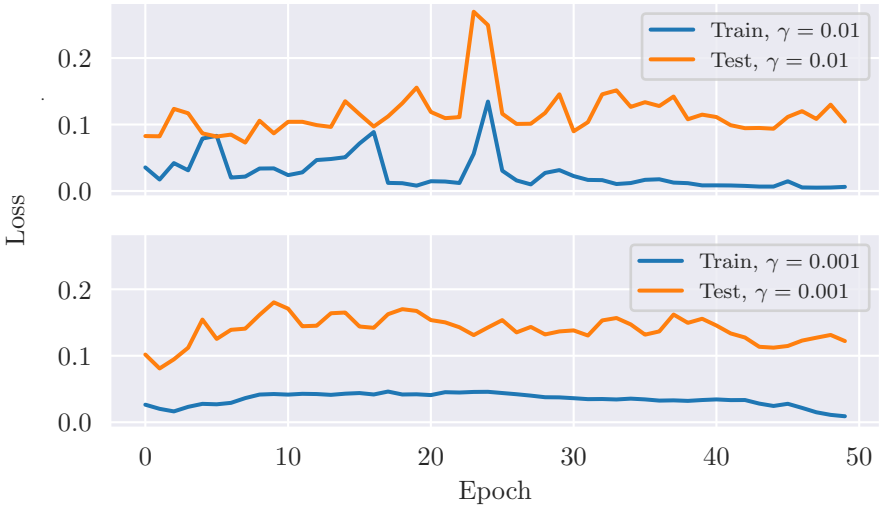


Figure 7.14: MSE as a function of epoch, reconstructing Franke's function using a dataset consisting of 20**x**20 datapoints, split into a training- and test set of 70%, and 30% respectively. The degree of the polynomial sample construction was chosen to be 5. The neural network encoding utilized 2 hidden layers of 11 and 6 neurons with tanh as activation functions within the network. The learning rate was chosen to be 0.01 in the upper plot and 0.001 in the lower plot

7.5.1 Franke's function - discussion

From figure Figure 7.14 it is easy to see that the VarQBM does not generalize well to test data, in addition to being relatively unstable, which seems to be due to the learning rate being too large. To achieve a smoother training curve, the learning rate was reduced to 0.001, which makes the training more stable, but still does not generalize well to test data. Towards the end of both models, the VarQBMs fit better and better to the training data, which shows that the VarQBMs have the power to fit Franke's function relatively well. The bad generalization might be due to 280 data points not being enough training data to learn the trend of the data. In addition the complexity degree of the samples might not be the most appropriate with a polynomial degree of 5. It is also important to remember that in this case, the sampling probabilities are used directly in difference with the classification tasks, where the probabilities had to stay within some threshold, this certainly increases the need for a more complex Hamiltonian, still it seems possible in theory and will be left as a topic to investigate in the future.

CHAPTER 8

Quantum Systems

8.1 Hydrogen Molecule

8.1.1 Finding the ground state energy using VarQITE

The Hydrogen Hamiltonian of the Hydrogen molecule is transformed using the Jordan-Wigner transformation with a bond length of $R = 0.75$Å.

Finding the grounds state energy of a hydrogen molecule using VarQITE and different initializations of the ansatz parameters can be seen in Figure 8.1.

Using a distribution of $[-1, 1]$, the ground state energy was found using different step sizes. The figure can be seen in Figure 8.2

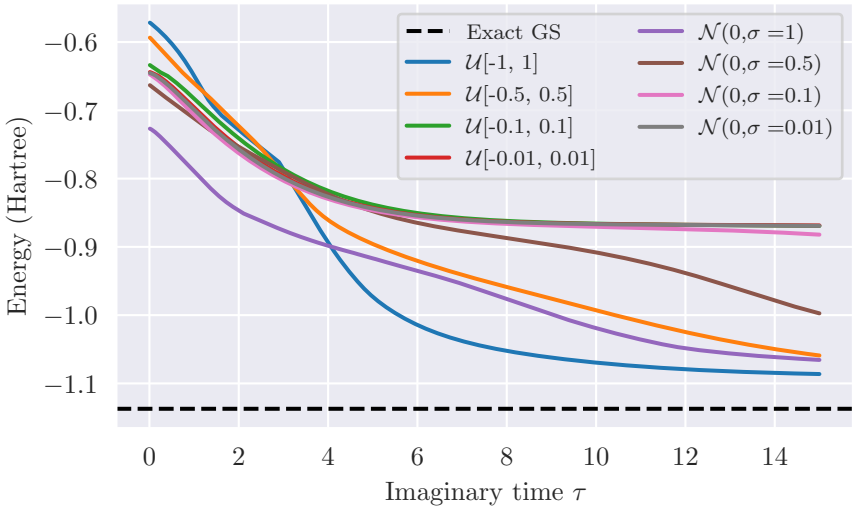


Figure 8.1: Energy as a function of imaginary time using different distributions and a step size of 0.01

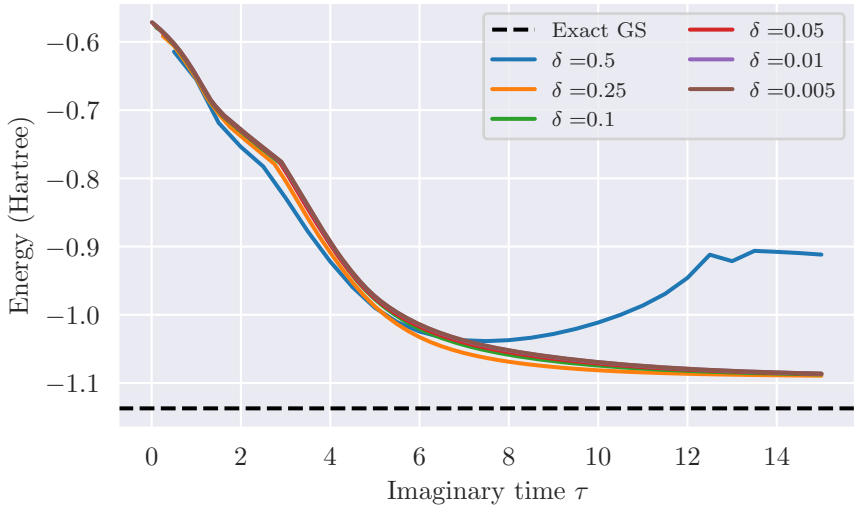


Figure 8.2: Energy as a function of imaginary time for different sizes of time steps, initializing the parameters using $U[-1, 1]$.

8.1.2 Finding the ground state energy using RBM

The Restricted Boltzmann machine using 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 8.3

The chosen value of σ was set to 0.9 for future computations. The learning rate γ was seen to affect the results fairly little with $\gamma \in [10^{-3}, 10^{-1}]$, therefore the learning rate was set to 0.01. The number of hidden nodes was then investigated for a variety of hidden nodes. The final results can be seen in Table 8.1

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 stand for computed and referenced. The reference value represents the experimental ground state energy[68].

Hidden Nodes	$E_C(Hartree)$	μ_C	$E_{ref}(Hartree)$	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

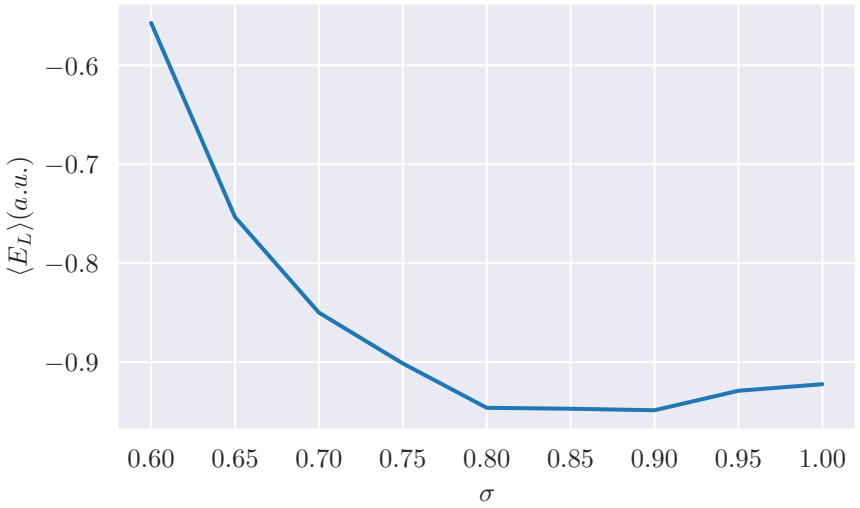


Figure 8.3: Energy as a function of σ when utilizing the Gaussian-Binary RBM computing the energy of H_2

8.1.3 Hydrogen molecule - Discussion

Having a look at figure Figure 8.1 shows that the distributions have some effect on the convergence of the energy. Either through not converging smoothly or converging quite slowly. The optimal distribution was shown to be $U[-1, 1]$, which was used further on.

Figure 8.2 shows that almost all time evaluations varying step sizes converged toward some proposed ground state energy, arguing that the step size is not as important as the distribution of parameters. The optimal step size was seen to be $\delta = 0.25$ with energy converged toward -1.09 Hartree, with an error of 4.21% which have some potential to improve. A way to improve the results might be to test different ansatzes to be able to search for the ground state at a larger part of the Hilbert space.

Regarding the classical RBM, the ground state energy found was worse compared to the VarQITE approach. Utilizing the classical RBM found a ground state energy of -0.96 Hartree which has room for improvement. The reason the ground state energy was found to be this high is most likely due to the trial function used due to a gaussian trial function not being able to express the wave function of the quantum system. Exploring different trial functions would certainly be a first step in the right direction for future prospects.

PART VI

Conclusion and Future Prospects

CHAPTER 9

Conclusion

In this thesis, the Variational quantum Boltzmann machine has been utilized in addition to classical versions of restricted Boltzmann machines. The VarQBM consisted of a hybrid version of classical- and quantum computing, where the optimization routine was done utilizing classical optimization techniques. The trainable parameters were infused into the Hamiltonian, giving rise to some target probability distribution. The quantum proportion of the VarQBM is based on exploiting VarQITE to generate approximated Gibbs states incorporating an intrinsic distribution of the trainable Hamiltonian.

Generating target distributions using a VarQBM was shown to be a success utilizing a 1-qubit Hamiltonian, and a 2-qubit trial circuit as an ansatz circuit for a simple probability distribution consisting of two bins, in addition to a 2-qubit Hamiltonian utilizing a 4-qubit trial circuit to generate the Bell state.

Applying the VarQBM on classical datasets was also a success. When applied to a transaction dataset, the VarQBM outperformed the classical RBMs both using 4- and 30 hidden nodes, by a couple of percents regarding the accuracy, when utilizing a neural network approach to encode the data into two qubits. While having the features encoded into the Hamiltonian as an additional bias did not reach the height of the two other methods regarding accuracy, but achieved the second highest F_1-score right below NN-VarQBM. In addition, a series of other standard classifiers was also outperformed by the VarQBM.

When applied to handwritten digit datasets of varying sizes the NN-VarQBM was among the three methods achieving the highest accuracy, while the Bias-VarQBM were outperformed most likely due to the compact encoding of features. The fairly simple model being a fully visible VarQBM using a 2-qubit Hamiltonian and a 4-qubit trial circuit even showed promising results when applied to images of more than 700 features with a datset consisting of four classes. The VarQBM classified most of the samples correct but was still not able to reach the most of the standard classifiers, which most likely is due to the Hamiltonian used in the VarQBM being too simple to express all features when the many features are compressed into the Hamiltonian, it is worth mentioning that lifting the results of the VarQBM higher seems fairly effortless by introducing more complex Hamiltonians in addition to larger trial circuits, with the price of needing longer preparation times of the Gibbs states.

It is also worth mentioning that the VarQBM applied to Franke's function did not yield any certain conclusion, most likely due to manipulating sampling probabilities being strenuous, in comparison to classification tasks where the sampling probabilities need to be within a certain threshold. It still seems like a possible task, provided enough data points, and a complex enough Hamiltonian. This will be left for future prospects to investigate.

Finding the ground state energy of the H_2 molecule was studied as well by utilizing VarQITE in addition to an RBM utilizing neural quantum state as the wave function. The results were promising for the VarQITE approach outperforming the RBM by utilizing a 2-qubits Hamiltonian as well as a 2-qubit ansatz circuit. Even though there is still room for improvement, most likely with the use of another trial circuit reaching a larger part of the Hilbert space. Here it is also worth mentioning that the classical RBM has large potential to increase its results by investigating different trial functions for a better suitable function for the given problem.

Summing up the conclusion, the VarQBM showed promising results in generating target probability distributions as well as applied to classical datasets. In the case of classical dataset applications, the VarQBM outperformed the classical RBM in most cases, but when the number of features got too big, the classical RBM seemed like a better fit. Even though due to the neural network encoding of features the VarQBM seems to have huge untapped potential. This is mainly due to the number of features in a sample easily being encoded into quantum circuits ready to run the VarQBM approach as long as the Hamiltonian has the power to express the feature space. In addition, the number of classes that the VarQBM is able to differentiate is exponential with the number of visible qubits. Which would make the quantum version of the Boltzmann machine preferable when dealing with a high amount of classes.

9.1 Future Prospects

Regarding the future work of this thesis and especially the VarQBM, a quite intuitive prospect would be to investigate more complex Hamiltonian ansatzes, in addition to larger Hamiltonians consisting of more qubits, as well as more complex trial circuits. In addition, classifying all classes within the image recognition datasets would also be a quite natural approach to future work. In addition, it would be quite natural to investigate the power of appending X and Y gates to the Hamiltonian to see how the complexity changes. The reason that X and Y gated Hamiltonians were not utilized in this thesis was due to not managing to find an appropriate Hamiltonian without increasing the time preparation of the approximated Gibbs states too much.

Another aspect of future work worth studying would be to investigate how the algorithm performs using real quantum hardware, in addition to investigating how error correction would affect the simulations. An idea would also be to see how major the impact of error correction would be during training due to the additional noise plausibly helping the model generalize well.

Regarding quantum mechanical future prospects, an idea from [5] that seems

like an intriguing idea would be to generate Hamiltonians based on experimental data.

Lastly, as a future approach regarding quantum mechanical problem solving, it would be worth investigating how the VarQBM performs finding ground state energies of complex many-body methods. An idea would be to prepare an ansatz utilizing the VarQBM. The proposed idea starts off by preparing a trial circuit which will represent the ansatz, in addition to two Hamiltonians: one Hamiltonian with trainable parameters as utilized in this thesis, in addition to a second fixed Hamiltonian representing the quantum system containing the ground state. The trial circuit evolves its parameter through VarQITE as done in this thesis. After the trial circuit is evolved the ansatz is applied to the fixed Hamiltonian representing the quantum system. The energy then has to be minimized by tuning the parameters of the first Hamiltonian containing the trainable parameters. This would represent one Boltzmann cycle. The optimization of the Hamiltonian ansatz will most likely be quite straightforward due to the gradients of $\frac{d\omega}{d\theta}$ already being computed during VarQITE.

Appendices

APPENDIX A

Machine learning

A.1 Source Code

All the source code is located in github.com/philipkarim/master_thesis.

A.2 Derivation of Gradients Used in Backpropagation

The derivation is done according to [18, 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:

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

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:

$$\begin{aligned}
\frac{\partial L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial W_{ij}^{last-1}} &= \sum_k \frac{\partial L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial a_k^{last}} \frac{\partial a_k^{last}}{\partial W_{ij}^{last-1}}, \\
&= \sum_k \frac{\partial L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial a_k^{last}} \frac{\partial a_k^{last}}{\partial z_k^{last}} \frac{\partial z_k^{last}}{\partial a_i^{last-1}} \frac{\partial a_i^{last-1}}{\partial z_i^{last-1}} \frac{\partial z_i^{last-1}}{\partial W_{ij}^{last-1}}, \\
\frac{\partial L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial b_i^{last-1}} &= \sum_k \frac{\partial L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial a_k^{last}} \frac{\partial a_k^{last}}{\partial b_i^{last-1}}, \\
&= \sum_k \frac{\partial L\left(\boldsymbol{a}^{last}, \boldsymbol{y}\right)}{\partial a_k^{last}} \frac{\partial a_k^{last}}{\partial z_k^{last}} \frac{\partial z_k^{last}}{\partial a_i^{last-1}} \frac{\partial a_i^{last-1}}{\partial z_i^{last-1}} \frac{\partial z_i^{last-1}}{\partial b_i^{last-1}}.
\end{aligned}$$

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:

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

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

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

APPENDIX B

Quantum Mechanics

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:

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

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:

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

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:

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

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:

$$\begin{aligned} \frac{\partial \log \Psi'}{\partial X_i} &= \frac{(a_i - X_i)}{\sigma^2} + \sum_{j=1}^{N} \frac{w_{ij} \exp\left(b_j + \sum_{i=1}^{M} \frac{X_i w_{ij}}{\sigma^2} \right)}{\sigma^2 \left(1 + \exp\left(b_j + \sum_{i=1}^{M} \frac{X_i w_{ij}}{\sigma^2} \right) \right)}, \\ &= \frac{a_i - X_i}{\sigma^2} + \frac{1}{\sigma^2} \sum_{j=1}^{N} w_{ij} \delta(\delta_j^{input}). \end{aligned} \tag{B.3}$$

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

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

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.3 and Equation B.4 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:

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

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:

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

Then by inserting equation Equation B.2 into the equation and differentiate with respect to the different free parameters leaves the following expression:

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

$$\frac{\partial \log \Psi'}{\partial b_k} = \frac{\exp\left(b_k + \sum_{i=1}^{M}\frac{X_i w_{ik}}{\sigma^2}\right)}{1 + \exp\left(b_k + \sum_{i=1}^{M}\frac{X_i w_{ik}}{\sigma^2}\right)} = \delta(\delta^{input}),$$

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

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.

APPENDIX C

Excessive Results

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

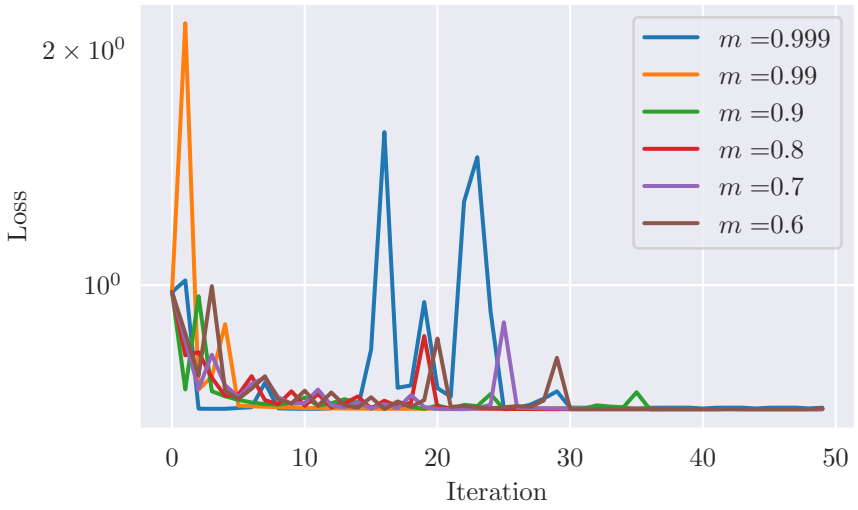


Figure C.1: Loss as a function of iteration using RMSprop with different initial momentum and learning rate of 0.1 The target distribution was the Bell state.

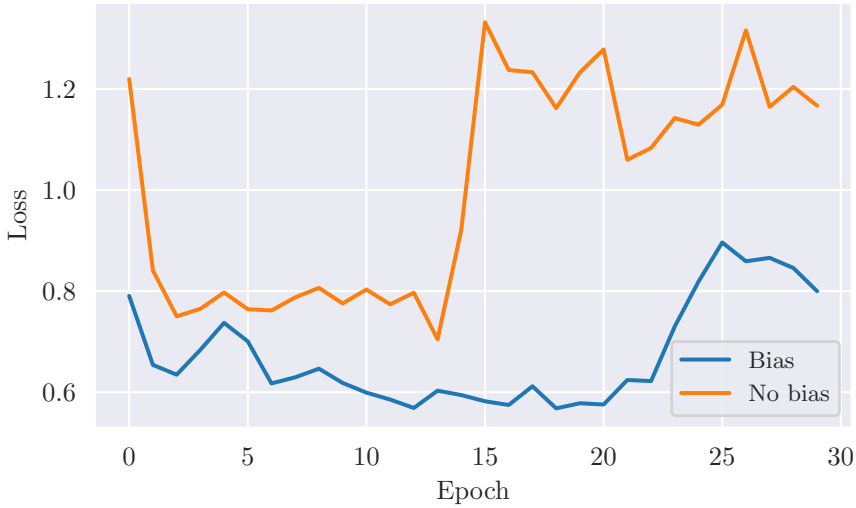


Figure C.2: 2 hidden layers with 12 hidden nodes each and sigmoid activation functions within the hidden layers, with and without bias initialized with a learning rate of 0.01 using 50 samples. The neural net is initialized using Xavier normal initialisation.

C.2 Discriminative Learning

C.2.1 Parameter investigation

Results regarding the parameter investigation. The results mainly consists of loss as functions of epochs using different parameters. The plots display training loss unless else is stated. The reason for this is that the parameter investigation is done in search of parameters making the network fit to the data best.

Transaction dataset

Investigating how including bias neurons in the neural network encoded VarQBM affects the training can be seen in Figure C.2. The reason for the instability of the training is probably due to the neural network being too large in addition to a bit few samples and a possibly high learning rate, but since the network containing a bias lays lower in loss than the network without a bias during the whole training, the bias will be kept. In addition having a bias in a neural network would most often be a natural thing to include. The reason for this is that the bias lets itself tune during the training, so if the case of having no bias would be preferred the weight of the bias should tune itself toward 0 during training.

A bit of different neural network structures was tested, which can be seen in Figure C.3. The figure argues that the size of the network have somewhat affect on the network when the network becomes too large, or is too small. The figure suggests a network size of 2 layers, it is also worth noticing that using

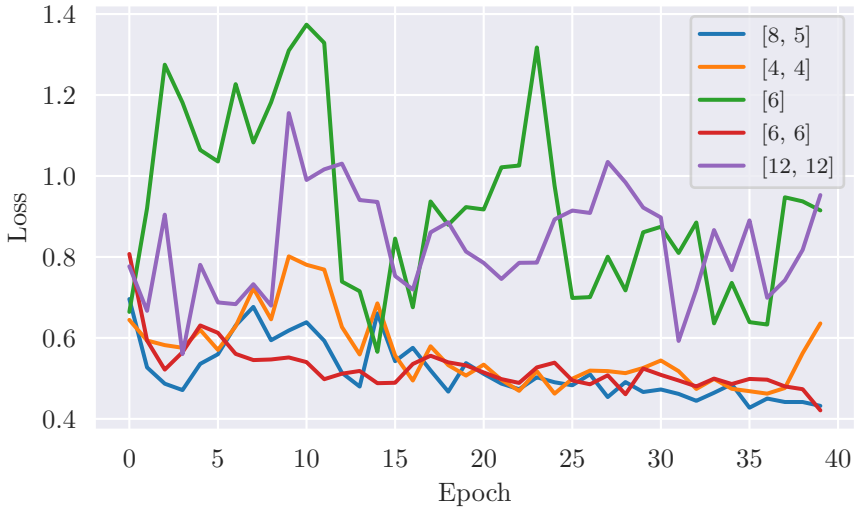


Figure C.3: Loss as function of epoch for network structures connected to the VarQBM using a learning rate of 0.01.

the pyramid rule to assemble a $[8, 5]$ structure managed to train well compared to the rest of the layer structures.

Handwritten image recognition

A bit of different neural network structures was tested, which can be seen in Figure C.4. The figure argues that as long as the network is small enough or follows the pyramid rule for neural network structures, the loss converges without blowing up the loss, it would most probably be possible to reduce the learning rate for a smoother learning curve in some of the cases presented.

The learning rate were also investigated, with the results given in Figure C.5, in this case, the learning rate or activation in the activation .

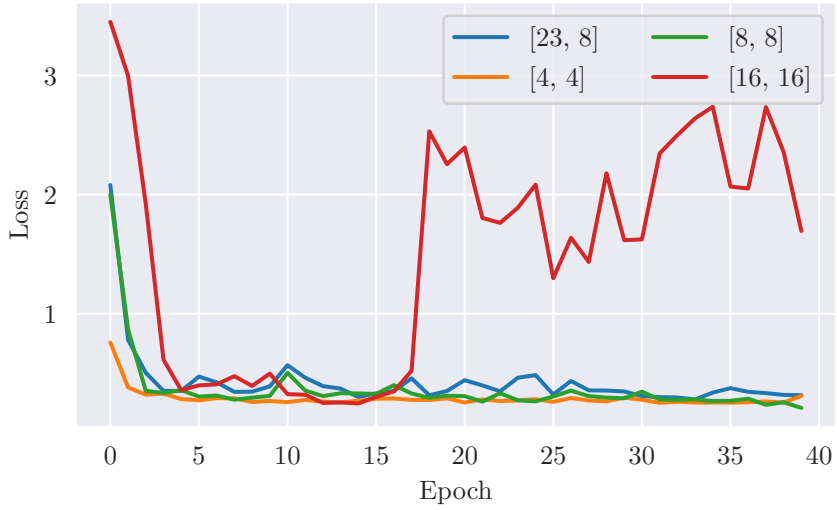


Figure C.4: Loss as a function of epoch for different layer sizes, training on the digit dataset using 50 samples.

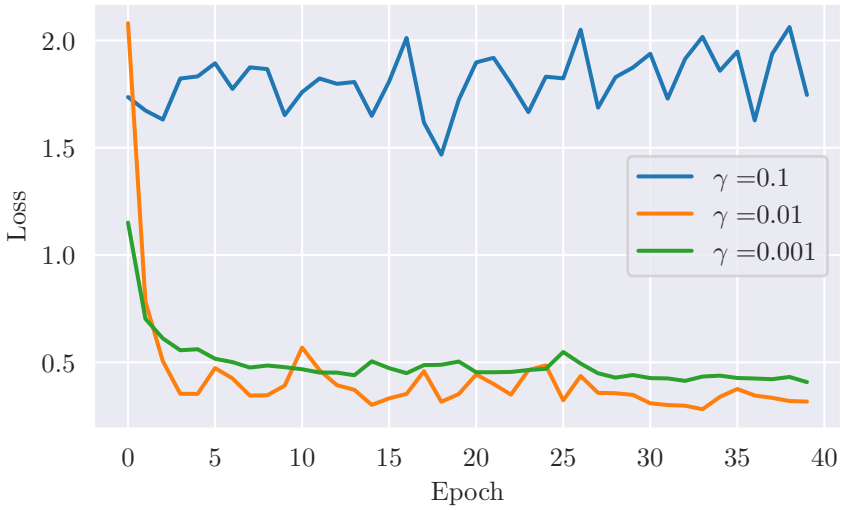


Figure C.5: Loss as a function of epoch for different learning rates, training on the digit dataset using 50 samples.

Bibliography

1. Feynman, R. P. Simulating physics with computers. *International journal of theoretical physics* vol. 21, 467 (1982).
2. Preskill, J. Quantum computing 40 years later. arXiv: 2106.10522 (2021).
3. Jordan, M. I. & Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science* vol. 349, 255 (2015).
4. Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. A learning algorithm for boltzmann machines. *Cognitive Science* vol. 9, 147 (1985).
5. Zoufal, C., Lucchi, A. & Woerner, S. Variational quantum Boltzmann machines. *Quantum Machine Intelligence* vol. 3 (2021).
6. Hinton, G. E. Boltzmann machine. *Scholarpedia* vol. 2. revision #91076, 1668 (2007).
7. Altman, E. R. Synthesizing Credit Card Transactions. arXiv: 1910.03033 (2019).
8. Dua, D. & Graff, C. *UCI Machine Learning Repository, Optical Recognition of Handwritten Digits Data Set* https://archive.ics.uci.edu/ml/datasets/optical+recognition+of+handwritten+digits. [Online; accessed Feb-2022]. (2017).
9. Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* vol. 29, 141 (2012).
10. McArdle, S. *et al.* Variational ansatz-based quantum simulation of imaginary time evolution. *npj Quantum Information* vol. 5 (2019).
11. Educative Answers Team. *Overfitting and underfitting* https://www.educative.io/answers/overfitting-and-underfitting. [Online; accessed June-2022].
12. Hastie, T., Tibshirani, R. & Friedman, J. *Elements of Statistical Learning: Data Mining, Inference, and Prediction* eng (Springer, 2009).
13. Schuld, M. & Petruccione, F. *Supervised Learning with Quantum Computers* 1st (Springer, 2018).
14. Kaggle Contributors. *Titanic - Machine Learning from Disaster* https://www.kaggle.com/c/titanic/overview. [Online; accessed Apr-2021].

15. Bre, F., Gimenez, J. M. & Fachinotti, V. D. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings* vol. 158, 1429 (2018).

16. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks* vol. 61, 85 (2015).

17. Nwankpa, C., Ijomah, W., Gachagan, A. & Marshall, S. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. arXiv: 1811.03378 (2018).

18. Rojas, R. *Neural Networks: A Systematic Introduction* (Springer, 1996).

19. Silva, S. D. *The Maths behind Back Propagation* https://towardsdatascience.com/the-maths-behind-back-propagation-cf6714736abf. [Online; accessed June-2021]. (2020).

20. Commons, W. *File:Boltzmannexamplev1.png — Wikimedia Commons, the free media repository* https://commons.wikimedia.org/w/index.php?title=File:Boltzmannexamplev1.png&oldid=560934009. [Online; accessed Jul-2021]. (2012).

21. Hjorth-Jensen, M. Advanced Topics in Computational Physics: Computational Quantum Mechanics. [Online; accessed May-2021] ((2021)).

22. Vuuren, V., Bosch, L. & Niesler, T. Unconstrained Speech Segmentation using Deep Neural Networks. *ICPRAM 2015 - 4th International Conference on Pattern Recognition Applications and Methods, Proceedings* vol. 1, 248 (2015).

23. Salakhutdinov, R. Learning Deep Generative Models. *Annual Review of Statistics and Its Application* vol. 2, 361 (2015).

24. Fischer, A. & Igel, C. Training restricted Boltzmann machines: An introduction. *Pattern Recognition* vol. 47, 25 (2014).

25. Andrychowicz, M. *et al.* Learning to learn by gradient descent by gradient descent. arXiv: 1606.04474 (2016).

26. Ruder, S. An overview of gradient descent optimization algorithms. arXiv: 1609.04747 (2016).

27. Reddi, S. J., Kale, S. & Kumar, S. On the Convergence of Adam and Beyond. arXiv: 1904.09237 (2019).

28. Ioffe, S. & Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML'15* 448 (2015).

29. Hossin, M. & M.N, S. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining Knowledge Management Process* vol. 5, 01 (2015).

30. Griffiths, D. J. & Schroeter, D. F. *Introduction to Quantum Mechanics* 3rd ed. (Cambridge University Press, 2018).

31. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information: 10th Anniversary Edition* 10th (Cambridge University Press, 2011).

32. Hemmer, P. *Kvantemekanikk: P.C. Hemmer* (Tapir akademisk forlag, 2005).

33. Cresser, J. *Quantum Physics Notes* (Macquarie University 2009, 2009).
34. Rieffel, E. & Polak, W. *Quantum Computing: A Gentle Introduction* 1st (The MIT Press, 2011).
35. Fernández, F. *The Born-Oppenheimer approximation* 2019.
36. Carleo, G. & Troyer, M. Solving the quantum many-bodyproblem with artificialneural networks. *Science* vol. 355, 602 (2017).
37. Pathak, S. K. Introduction to Second Quantization. *Modern Condensed Matter Physics* (2019).
38. Hjorth-Jensen, M. Many-body Hamiltonians, basic linear algebra and Second Quantization. Lecture Notes in Computational Physics, University of Oslo. [Online; accessed Jun-2021] ((2018)).
39. Wikipedia contributors. *Bloch sphere — Wikipedia, The Free Encyclopedia* https://en.wikipedia.org/w/index.php?title=Bloch_sphere&oldid=1008849646. [Online; accessed Mar-2021]. (2021).
40. Marciano, D. *Quantum Computing for Everyone - Part III: Quantum Circuits and OpenQASM, Code Project* https://www.codeproject.com/Articles/1182208/Quantum-Computing-for-Everyone-Part-III-Quantum-Ci. [Online; accessed Mar-2021]. (2018).
41. ANIS, M. S. *et al. Qiskit Textbook: An Open-source Framework for Quantum Computing* (2021).
42. Benedetti, M., Lloyd, E., Sack, S. & Fiorentini, M. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology* vol. 4 (2019).
43. Crooks, G. E. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. arXiv: 1905.13311 (2019).
44. Wick, G. C. Properties of Bethe-Salpeter Wave Functions. *Physical Review* vol. 96, 1124 (1954).
45. McLachlan, A. A variational solution of the time-dependent Schrodinger equation. *Molecular Physics* vol. 8, 39 (1964).
46. Gibbs, J. W. *Elementary Principles in Statistical Mechanics: Developed with Especial Reference to the Rational Foundation of Thermodynamics* (Cambridge University Press, 2010).
47. Temme, K., Osborne, T. J., Vollbrecht, K. G., Poulin, D. & Verstraete, F. Quantum Metropolis sampling. *Nature* vol. 471, 87 (2011).
48. Yung, M.-H. & Aspuru-Guzik, A. A quantum-quantum Metropolis algorithm. *Proceedings of the National Academy of Sciences* vol. 109, 754 (2012).
49. Poulin, D. & Wocjan, P. Sampling from the Thermal Quantum Gibbs State and Evaluating Partition Functions with a Quantum Computer. *Physical Review Letters* vol. 103 (2009).
50. Motta, M. *et al.* Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution. *Nature Physics* vol. 16, 205 (2019).

51. Brandao, F. G. S. L. & Kastoryano, M. J. Finite correlation length implies efficient preparation of quantum thermal states. arXiv: 1609.07877 (2019).

52. Kastoryano, M. J. & Brandao, F. G. S. L. Quantum Gibbs Samplers: the commuting case. arXiv: 1409.3435 (2016).

53. Yuan, X., Endo, S., Zhao, Q., Li, Y. & Benjamin, S. C. Theory of variational quantum simulation. *Quantum* vol. 3, 191 (2019).

54. Somma, R., Ortiz, G., Gubernatis, J. E., Knill, E. & Laflamme, R. Simulating physical phenomena by quantum networks. *Physical Review A* vol. 65 (2002).

55. Tranter, A., Love, P. J., Mintert, F. & Coveney, P. V. A Comparison of the Bravyi–Kitaev and Jordan–Wigner Transformations for the Quantum Simulation of Quantum Chemistry. *Journal of Chemical Theory and Computation* vol. 14, 5617 (2018).

56. Aleksandrowicz, G. *et al. Qiskit: An Open-source Framework for Quantum Computing* version 0.7.2. 2019.

57. Padhi, I. *et al. Tabular transformers for modeling multivariate time series* in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2021), 3565.

58. Glorot, X. & Bengio, Y. *Understanding the difficulty of training deep feedforward neural networks.* in *AISTATS* **9** (2010), 249.

59. He, K., Zhang, X., Ren, S. & Sun, J. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification* in *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), 1026.

60. Masters, T. *Practical Neural Network Recipes in C++* (Academic Press Professional, Inc., 1993).

61. McClean, J. R. *et al.* OpenFermion: the electronic structure package for quantum computers. *Quantum Science and Technology* vol. 5, 34014 (2020).

62. Turney, J. *et al.* PSI4: an open-source ab initio electronic structure program. *Wiley interdisciplinary reviews: Computational Molecular Science.* vol. 2, 556 (2012).

63. Moll, N., Fuhrer, A., Staar, P. & Tavernelli, I. Optimizing qubit resources for quantum chemistry simulations in second quantization on a quantum computer. *Journal of Physics A: Mathematical and Theoretical* vol. 49, 295301 (2016).

64. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* vol. 12, 2825 (2011).

65. Jonsson, M. Standard error estimation by an automated blocking method. *Physical Review E* vol. 98, 43304 (2018).

66. Doma, S. B., Abu-Shady, M., El-Gammal, F. N. & 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, 1787 (2016).

67. Smith, L. N. Cyclical Learning Rates for Training Neural Networks. arXiv: 1506.01186 (2015).

68. Moskowitz, J. W. & Kalos, M. H. New look at correlations in atomic and molecular systems. I. Application of fermion Monte Carlo variational method. *Int. J. Quant. Chem.; (United States)* vol. 20 (1981).