Pontifícia Universidade Católica
do Rio de Janeiro

Pedro da Silveira Carvalho Ripper

`QARBoM.jl`: a framework for Classical and Quantum-Assisted Training of Restricted Boltzmann Machines

Dissertação de Mestrado

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica, do Departamento de Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica.

Advisor : Prof. Guilherme Penello Temporão
Co-advisor: Dr. Gustavo Castro do Amaral
Co-advisor: Prof. David Esteban Bernal Neira

Rio de Janeiro
August 2025

Pontifícia Universidade Católica
do Rio de Janeiro

Pedro da Silveira Carvalho Ripper

`QARBoM.jl`: a framework for Classical and Quantum-Assisted Training of Restricted Boltzmann Machines

Thesis presented to the Programa de Pós–graduação em Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica. Approved by the Examination Committee:

Prof. Guilherme Penello Temporão
Advisor
Departamento de Engenharia Elétrica – PUC-Rio

Dr. Gustavo Castro do Amaral
TNO

Prof. David Esteban Bernal Neira
PU

Prof. Can Li
PU

Dr. Esteban Aguilera
TNO

Rio de Janeiro, August the 21st, 2025

Pedro da Silveira Carvalho Ripper

Graduated in Computer Engineering at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Currently a Research and Development (R&D) Analyst at PSR, he first joined the company as an intern in 2022. In his role, he develops software for R&D projects and conducts research in Quantum Optimization. A co-founder of Brazil Quantum, one of the country's first quantum computing initiatives, he is also an IBM Qiskit Advocate since 2021.

Bibliographic data

Ripper, Pedro da Silveira Carvalho

`QARBoM.jl`: a framework for Classical and Quantum-Assisted Training of Restricted Boltzmann Machines / Pedro da Silveira Carvalho Ripper; advisor: Guilherme Penello Temporão; co-advisores: Gustavo Castro do Amaral, David Esteban Bernal Neira. – 2025. 128 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica, 2025. Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Máquina Restrita de Boltzmann. 3. Modelos Baseados em Energia. 4. Computação Quântica. 5. Aprendizado de Máquina assistido por quântica. 6. QUBO . I. Penello Temporão, Guilherme. II. Castro do Amaral, Gustavo. III. Bernal Neira, David Esteban. IV. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. V. Título.

CDD: 621.3

Acknowledgments

Working with my advisors from my home university, Guilherme Temporão and Gustavo Amaral, since 2020 has not only shaped my career by introducing me to the field of Quantum Technologies, but it has also impacted my personal life, as I was presented with a friendship that I will always cherish. I would like to thank them both for teaching me so much and for their support over these last few years. I am also grateful to my advisor, David Bernal, whom I first met while I was still an intern at PSR, and who has been a great influence on the topic of this thesis.

Beyond my official advisors, I was fortunate to have Joaquim Dias Garcia as a mentor and a friend at PSR. He was essential in the early and final steps of this thesis, helping me to outline the functionalities of `QARBoM.jl` and giving very valuable insights in the experimental section.

I would like to thank my whole family, especially my parents, Gláucia and Carlos, my sister, Júlia, and my grandparents, Angela and Roberto, for being forever on my side and for giving me love and emotional support during this period. I am also very grateful for my girlfriend, Fernanda, who has always been there for me, giving me so much love and encouragement.

I would also like to express my gratitude to everyone at PSR, especially my teammates Bianca Lacê, Carolina Monteiro, Fellipe Pessanha, Gabriel Vidigal, Guilherme Bodin, Joaquim Dias Garcia, Luiz Cláudio, Rafael Benchimol, Raphael Sampaio, and Vinicius Justen. I am also grateful for Mario Veiga, who shares my interest in Energy-Based Models and also had a great influence on the topic of this work.

I would also like to thank my lab mates from NITeQ, Alexandre Wanick, Anderson Marinho, Bruno Wolf, Christiano Nascimento, João Neto and Luana Göbel, who participated in discussions that led to the development and improvement of `QARBoM.jl`.

Finally, I would like to thank everyone from the SECQUOIA research group for helping me run `QARBoM.jl` with real Quantum Annealers. Special thanks to Pedro Maciel Xavier for the years of collaboration on working with anything related to QUBOs.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Ripper, Pedro da Silveira Carvalho;Penello Temporão, Guilherme (Advisor);Castro do Amaral, Gustavo (Co-Advisor);Bernal Neira, David Esteban (Co-Advisor). **`QARBoM.jl`: a framework for Classical and Quantum-Assisted Training of Restricted Boltzmann Machines**. Rio de Janeiro,2025. 128p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

This thesis presents `QARBoM.jl`, a platform for benchmarking quantum-assisted against classical training of Restricted Boltzmann Machines (RBMs). Recent works have been testing the training of RBMs using quantum sampling techniques, such as Quantum Annealing, and comparing their results against classical methods. However, these projects are mainly limited to a specific dataset and only one RBM classical training procedure to compare. With that said, `QARBoM.jl` establishes an agnostic benchmarking framework where, with minor code adjustments, one can select over different training algorithms (classical or quantum-assisted) and parameters to model integer or real-valued datasets, expediting the research endeavor on the applications of Quantum Computing for RBMs.

Keywords

Restricted Boltzmann Machine; Energy Based Model; Quantum Computing; Quantum-Assisted Machine Learning; QUBO.

Resumo

Ripper, Pedro da Silveira Carvalho; Penello Temporão, Guilherme; Castro do Amaral, Gustavo; Bernal Neira, David Esteban. **`QARBoM.jl`: um framework para Treinamento Clássico e Quântico-Assistido de Restricted Boltzmann Machines**. Rio de Janeiro, 2025. 128p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Essa dissertação apresenta o `QARBoM.jl`, uma plataforma para comparar técnicas de treinamento assistido por quântica e clássicos de Máquinas Restritas de Boltzmann (MRBs). Trabalhos recentes vêm testando o treinamento de MRBs usando métodos de sampleamento quântico, como Recozimento Quântico, e comparando seus resultados com algoritmos clássicos. Todavia, essas pesquisas são limitadas à uma base de dados específica e geralmente apenas uma técnica de treinamento clássica. Diante desse panorama, o `QARBoM.jl` estabelece uma metodologia agnóstica onde, com pequenos ajustes, seus usuários podem selecionar entre diferentes algoritmos de treinamento (quânticos assistidos ou clássicos) e parâmetros para modelar base de dados com valores inteiros ou contínuos.

Palavras-chave

Máquina Restrita de Boltzmann; Modelos Baseados em Energia; Computação Quântica; Aprendizado de Máquina assistido por quântica; QUBO .

Table of contents

List of figures

List of tables

'Tá na marca.'

Vovô Roberto.

1

Introduction

The United Nations has proclaimed 2025 as the Year of Quantum Science and Technology[1], in order to increase awareness for quantum technologies, such as Quantum Sensing, Communication, and Computing. Several countries have been investing in this area, as portrayed by the latest national quantum initiatives survey by Qureca[2] in Figure 1.1.

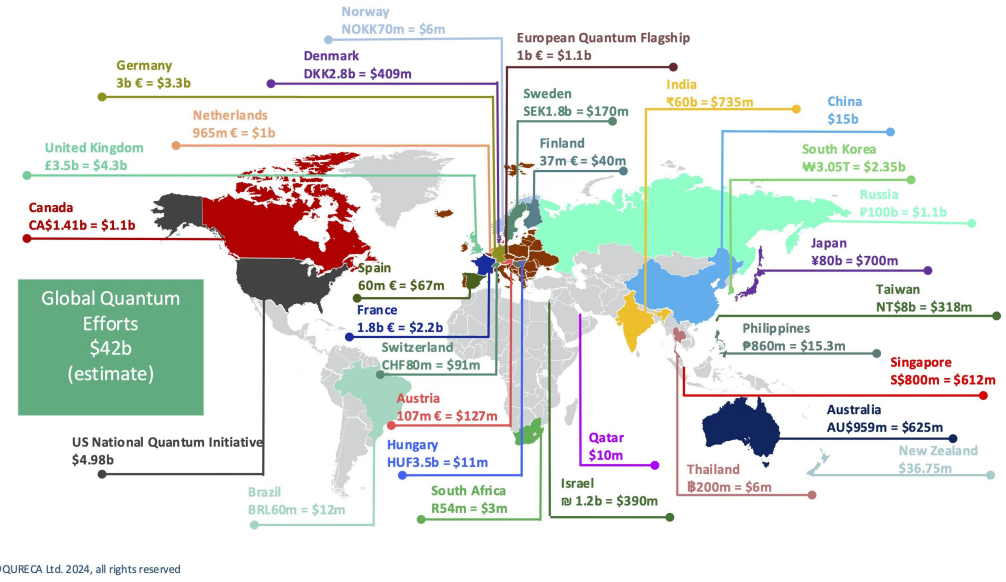

Figure 1.1: National Quantum Initiatives. Source: Qureca

This interest comes from the expected benefits that these technologies can bring in the future. More specifically, quantum computers are predicted to provide computational speedup over non-quantum computers (classical computers), for some specific problems in areas such as in Optimization [2], Chemistry [3] and Machine Learning [4]. This speedup is often referred as *Quantum Advantage* or *Quantum Supremacy*.

[1] Year of Quantum Science and Technology: (link).
[2] Quantum Initiatives Worldwide 2024

Quantum Computing (QC), however, is yet to present an experiment for a real-world problem where a quantum device outperforms a classical computer, as QC is in a preliminary stage known as the Noisy Intermediate Scale Quantum (NISQ) Era [5]. In the NISQ Era, quantum hardware is susceptible to noisy operations and decoherence, and is unfit to run larger problems due to scalability challenges.

That being said, researchers have been experimenting on applications where quantum and classical computers can be used cooperatively, sparing the quantum devices of heavy operations that can be performed on traditional computers. A famous example for that is the class of hybrid algorithms known as Variational Quantum Algorithms (VQA) [6], although their usefulness has been recently questioned [7]. These algorithms work as optimizers searching for optimal parameters for the quantum computer execution. Moreover, there is a specific quantum computer architecture known as Quantum Annealer, which also performs an optimization task.

These two techniques are well fit to solve an Optimization problem in the QUBO (Quadratic Unconstrained Binary Optimization) format, that is, a model comprised only of binary variables, unconstrained and with its polynomial at most quadratic. Its representation follows in the equation below.

$$\min_{\mathbf{x}\in\mathbb{B}^n} \mathbf{x}'\vec{Q}\,\mathbf{x} = \min_{\mathbf{x}\in\mathbb{B}^n} 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} q_{i,j}\,x_i\,x_j + \sum_{i=1}^{n} q_{i,i}\,x_i \tag{1-1}$$

This formulation maps to a model from Statistical Mechanics known as the Ising Spin Model, depicted in Equation 1-2, and it represents ferromagnetic interactions of spin particles in a lattice.

$$H(\sigma) = -\sum_{ij} J_{ij}\sigma_i\sigma_j - \sum_{i} h_i\sigma_i, \text{ where } \sigma_k \in \{\pm 1\} \tag{1-2}$$

Statistical Mechanics has also inspired a class of neural networks called Energy Based Models (EBM) [8], which was developed by John Hopfield and Geoffrey Hinton. The fundamental importance of this work, which bridged concepts from Physics and Machine Learning, was recognized when Hopfield and Hinton were awarded the Nobel Prize in Physics in 2024[3].

[3]Nobel Prize in Physics 2024 : www.nobelprize.org/prizes/physics/2024/summary/

Being neural networks, EBMs have the objective of modeling and predicting patterns in data. A neural network is comprised of two or more layers of interconnected nodes that, using optimally trained parameters, can represent latent features in the data being studied. Moreover, their training is usually conducted by minimizing a *cost function* that depicts how well the neural network has modeled the data.

For the case of EBMs, the cost function is based on the probability of sampling a state $p(\mathbf{v})$, as we will see in the following chapter. EBMs have their probabilities linked to an energy function similar to the Ising Model. That being said, during the training of EBMs, one is interested in sampling low-energy states for this energy function.

The intersection between the EBMs and the optimization problems that the aforementioned quantum algorithms address opened a new branch of research: quantum-assisted training of EBMs. More specifically, different articles [1; 9; 10] have presented results for using QC to support the training of a specific type of EBM called Restricted Boltzmann Machine (RBM) [11]. Leveraging quantum methods that are well-fit to solve QUBOs/Ising models, one can sample low energy states for the RBM's energy function while maintaining the actual training of the neural network in a classical computer.

Although there are different ways of training an RBM, considering classical methods and quantum algorithms (e.g. Variational Quantum Algorithms, Quantum Annealing) methods, these works focused on benchmarking with only one classical technique which known as Contrastive Divergence. Moreover, their quantum-assisted training framework was restricted to Quantum Annealing.

Therefore, regarding the contrasting classical and quantum techniques for training RBMs, there is an absence of an framework to perform a comprehensive benchmark.

To address this issue, we introduce `QARBoM.jl` [12], an open-source Julia package for comparing RBM training over classical and quantum methods. This software is intended to become a simplified framework for benchmarking the different training procedures for RBMs that exist today and might exist in the future.

The work is presented as follows. First, the main concepts of RBMs that were used in the development of `QARBoM.jl` are outlined in Chapter 2. Following, Chapter 3 defines fundamental notions of QC that allow us to understand the quantum algorithms that target QUBO and Ising Models,

which will be explained in Chapter 4. Then, `QARBoM.jl` is finally introduced in depth in Chapter 5, where its benchmarking interface is exemplified with code snippets. After our framework is presented, Chapter 6 contains a use case of `QARBoM.jl`, with a well-known open-source dataset called the Tennessee Eastman process. Finally, Chapter 7 concludes this work, evaluating the possible contributions that `QARBoM.jl` can offer, followed by future works that can be conducted to improve our tool.

Additionally, it is worth mentioning that appendices sections are provided for the Chapters 2 and 3 and can be found at the end of this thesis.

2

Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) are an Energy-Based model with latent variables (usually binary), used to learn underlying data distributions [13]. EBM is a class of Neural Network that stems from the concept of Markov Random Field, which in turn draws notions from Statistical Mechanics [14]. These neural networks are comprised of visible and hidden nodes, connected in a setting that makes RBMs be perceived as complete bipartite graphs, as presented in Figure 2.1.

Visible and hidden nodes are represented by the v_i and h_j vertices, respectively. Moreover, every $\{v_i, h_j\}$ pair has an associated w_{ij} weight, while each visible and hidden node has a bias weight a_i and h_j respectively.

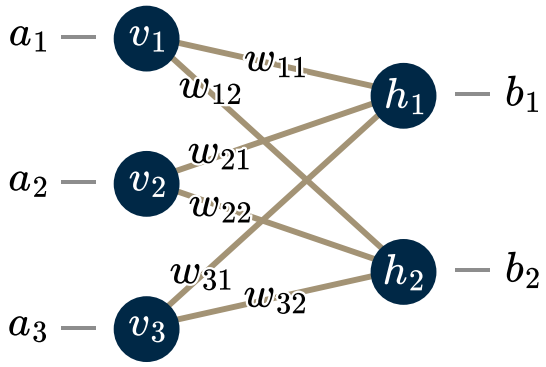

Figure 2.1: Restricted Boltzmann Machine as a complete bipartite graph. All visible nodes are connected to all hidden nodes and vice versa. Each (v_i, h_j) edge has a weight w_{ij}

2.1

Markov Random Fields

Markov Random Fields (MRFs) or Markov Networks are undirected Graphical Models comprised of nodes that represent random variables and edges that contain associated weights. One can find the join distribution over the variables $X = (X_1, \cdots, X_n)$ from a MRF as follows:

$$p(X = \mathbf{x}) = \frac{1}{Z} \prod_{C} f_C(\mathbf{x_C}) \tag{2-1}$$

In Equation 2-1, Z is a normalization constant, meaning that it corresponds to the sum of all possible values for the variables. It is also referred as the partition function. Moreover, f_C represents a function over the C-th maximal clique of the graph, while $\mathbf{x_C}$ is the set of variables in that clique, where $\mathbf{x_C} \subset \mathbf{x}$.

In a graph $G = (E, V)$, with a set of edges E and vertices V, a clique is a subset C of vertices in which every vertex in C is adjacent to every other vertex also in C. A maximal clique is a subgraph that would no longer be a clique if another vertex were added. For example, consider Figure 2.2. Subgraph 1 represents a clique of two vertices, as every vertex in subgraph 1 is adjacent to all the other vertices in the clique. On the other hand, subgraph 2 is a maximal clique because if we were to add the remaining vertex, not all vertices would be connected to each other, thus losing its clique property.

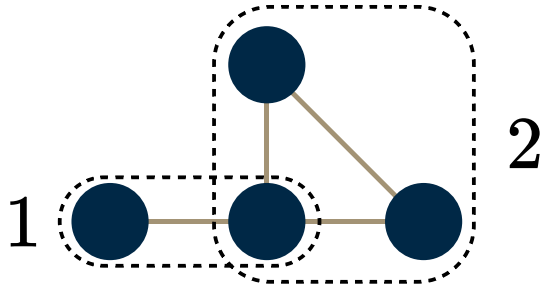

Figure 2.2: Clique (1) and Maximal Clique (2)

Now, returning to Equation 2-1, f_C can be expressed as an arbitrary exponential, as depicted in Equation 2-2, where $E(\mathbf{x}_C)$ is called the Energy Function. It is worth mentioning that, for being arbitrary, one can also multiply f_C by subsets of the maximal clique. This exponential representation is inherited from Statistical Mechanics and it is called the Boltzmann distribution [14]. It is worth noticing that smaller values for the energy function are related to greater probabilities, considering they are followed by a minus sign inside the exponential.

$$f_C(\mathbf{x}_C) = \exp(-E(\mathbf{x}_C)) \tag{2-2}$$

2.2

From Markov Random Fields to Energy Based Models

As stated, Energy-Based Models are Neural Networks that leverages concepts from MRFs. In this section, we will define the distribution function for an RBM, considering what was just presented about MRFs.

Being represented by a bipartite graph (see Figure 2.1) with M visible nodes and N hidden nodes, it is possible to observe that there are $M \times N$ maximal cliques in an RBM graph, one for each visible-hidden nodes edges. With that said, the product component in Equation 2-1 would consider every (v_i, h_j) pair. For each of them, we can define an energy equals to the weight of the edge - w_{ij} - connecting these two nodes. Moreover, as the f_C functions are arbitrary, we can add a biasing term for every visible node, which we will associate with a a_i value for the ith visible node and b_j for the jth hidden node. Finally, once again borrowing concepts from Statistical Mechanics, we want to assign lower energies to higher probabilities. Therefore, each of these terms will be preceded by a minus sign. At the end, we have a product of exponentials, which we can merge into a single exponential.

From this formulation, we have modeled the generalized probability function from Equation 2-1 into the following RBM distribution in Equation 2-3, where $E(\mathbf{v}, \mathbf{h})$ depicts the merged exponentials. Its expanded form is presented in Equation 2-4.

$$\mathrm{p}(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{2-3}$$

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{ij} w_{ij} v_i h_j \tag{2-4}$$

Furthermore, the partition function Z from Equation 2-1 is depicted in Equation 2-5. As mentioned, this function guarantees that the sum of all probabilities will yield 1. Additionally, in order to estimate it, one needs to find the value for $E(\mathbf{v}, \mathbf{h})$ for all possible configurations of visible and hidden nodes. An RBM with M visible and N hidden binary nodes has $2^{M \times N}$ different configurations. Thus, calculating the value for the partition function can be

computationally cumbersome, as the number of visible and hidden nodes increases, finding $p(\mathbf{v}, \mathbf{h})$ becomes exponentially harder.

$$Z = \sum_{\mathbf{v}', \mathbf{h}'} \exp(-E(\mathbf{v}'_{\mathbf{m}}, \mathbf{h}'_{\mathbf{n}})) \tag{2-5}$$

Moreover, the marginal probabilities can be represented as

$$\mathrm{p}(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}'} \exp(-E(\mathbf{v}, \mathbf{h}'_{\mathbf{n}})) \tag{2-6}$$

$$\mathrm{p}(\mathbf{h}) = \frac{1}{Z} \sum_{\mathbf{v}'} \exp(-E(\mathbf{v}'_{\mathbf{m}}, \mathbf{h})) \tag{2-7}$$

Regarding the joint distribution and the marginal distribution formulas, it is noticeable that working with these equation brings an exponential computational cost, as all possible values for the visible and hidden nodes must be computed. With that said, in the next section, we will introduce another concept from Graphical Models: conditional independence.

2.3

Conditional independence properties

In Graphical Models, two variables X, Y are said to be conditionally independent given a third variable Z if every path connecting the nodes X and Y passes through the Z node. The same holds for sets of variables. For instance, in Figure 2.3, we can see that if the nodes C and B were removed, there would be no path from node A to D. Therefore, we say that A and D are conditionally independent given C and B.

Taking this property into account, let us return to the RBM bipartite graph in Figure 2.1. If the hidden nodes h_1 and h_2 were removed, there would be no path between visible nodes, while the path between hidden nodes rely on the visible nodes v_1, v_2 and v_3. Therefore, the visible nodes are conditionally independent given the hidden nodes and vice versa.

Such a fact is quite useful as, in contrast with the joint and marginal distributions, the conditional probabilities can be easily estimated. Although

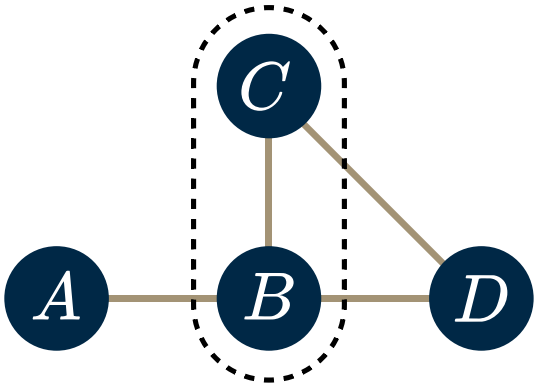

Figure 2.3: Variables A and D are conditionally independent, given the variables C and B ($A \perp\!\!\!\perp D|B, C$)

seeming quite complex at first glance, Equation 2-8 can be greatly simplified after some mathematical manipulations, which can be found in Appendix A.

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{v},\mathbf{h})}{p(\mathbf{v})} = \frac{\exp(-E(\mathbf{v},\mathbf{h}))/Z}{\sum_{\mathbf{h}'}\exp(-E(\mathbf{v},\mathbf{h}'))/Z} \tag{2-8}$$

These refinements yield the following probabilities for each visible and hidden node:

$$\begin{aligned} &p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i w_{ij}v_i) \\ &p(v_i = 1|\mathbf{h}) = \sigma(a_i + \sum_j w_{ij}h_j), \\ &\text{where } \sigma(x) = \frac{1}{1+e^{-x}} \end{aligned} \tag{2-9}$$

Notice that both hidden and visible nodes conditional probabilities in Equation 2-9 can be estimated in polynomial time. For instance, considering $p(h_j = 1|\mathbf{v})$, one needs to sum over the values of the vector $\mathbf{v}$, which has a length $\dim(\mathbf{v})$. Therefore, conditional probabilities are a practical tool for working with RBMs.

2.4

Marginal distributions via Gibbs Sampling

Even though marginal distributions are computationally hard, in the next section it will be disclosed that estimating them is required during the RBM training process. Regarding this issue, there is an algorithm for approximating marginal probabilities called Gibbs Sampling.

This method consists of instantiating a Markov Chain where, at each step, the values for each variable are updated according to their conditional independence given other variables [15].

To illustrate this algorithm, we use the RBM graphical model. First, initialize the visible nodes $\mathbf{v}$ to an arbitrary state $\mathbf{v^0}$. Then, sample the value for each hidden node via conditional distribution, given this arbitrary value $\mathbf{v^0}$. Next, sample values for the visible nodes from the conditional probability again, but now using the sampled values for the hidden nodes. Repeating this procedure for K steps yields an approximation of the marginal distribution $p(\mathbf{v})$ At the end, we would reach the marginal distribution for each visible variable. For better understanding, this process is outlined below in Algorithm 1, followed by an illustration in Figure 2.4.

Algorithm 1 Gibbs Sampling pseudocode

```
function GibbsSampling(rbm, k, v^0)
    v ← v^0
    for k ∈ 1 ··· K do
        h^{k−1} ← p(h|v^{k−1})
        v ← p(v|h^{k−1})
    end for
    return v
```

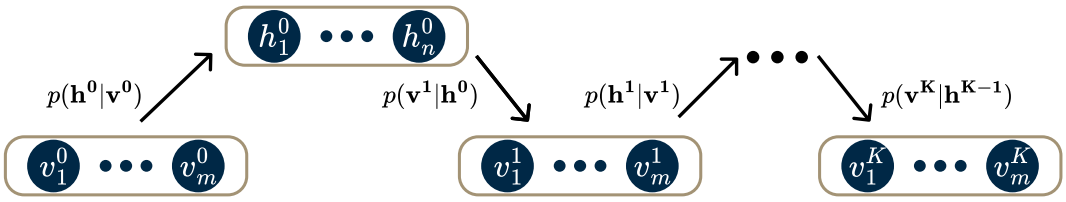

Figure 2.4: Gibbs Sampling algorithm illustration

It is worth mentioning that as the number of iterations K increases, the marginal probability approximation becomes more accurate. However, as we will see in the following sections, it also becomes more computationally expensive.

2.5

Training RBMs

RBMs are optimized by maximizing the log-likelihood of the observed training data ($\log p(\mathbf{v})$) over its parameters $\Theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$, which are, respectively, the matrices and vectors that contain the weights w_{ij} and biases a_i and b_j.

$$\max_{\Theta} \log p(\mathbf{v}) \tag{2-10}$$

Consider that we have started training a three visible node RBM with arbitrary values for Θ. At first, it could present random probabilities for each possible value of $\mathbf{v}$. However, when training with a dataset, this RBM will start increasing the probabilities of the $\mathbf{v}$ values that appear more in the dataset. This behavior is illustrated in Figure 2.5, where we can observe that the dataset for this example had more entries for $\mathbf{v} = [010]$.

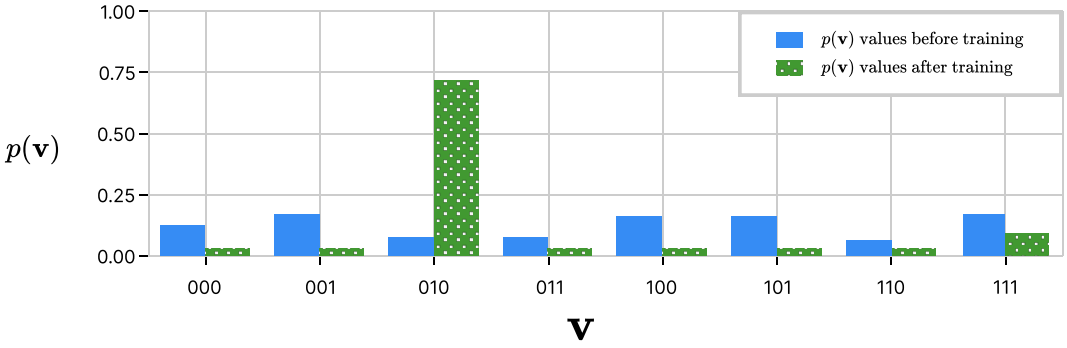

Figure 2.5: Example for the probabilities of the visible nodes of an RBM before and after training.

Observing Equation 2-6, it is possible to see that, in order to maximize $\log p(\mathbf{v})$, we need to increase the value for the exponent that contains the energy function, as the latter contains all the hyperparameters Θ. That being said, as the energy function is preceded by a minus sign, it actually needs to be minimized.

The values for these hyperparameters are iterated via gradient ascent. At first glance, taking the partial derivative for each weight might seem cumbersome. However, these equations can be significantly simplified after some manipulations, which can be found in Appendix B.

In previous works, the gradient for each hyperparameter are usually written as presented in Equations 2-11, 2-12 and 2-13, where $\langle \cdot \rangle$ represents the expected value for *data* and *model*.

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \tag{2-11}$$

$$\frac{\partial \log p(\mathbf{v})}{\partial a_i} = \langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{model}} \tag{2-12}$$

$$\frac{\partial \log p(\mathbf{v})}{\partial b_j} = \langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{model}} \tag{2-13}$$

Taking a closer look, according to Appendix B, the first term represents a conditional probability of h_j given the ith visible node from the training data sample $\mathbf{v}$, as follows in the equations below.

$$\langle v_i h_j \rangle_{data} \rightarrow p(h_j = 1|\mathbf{v}) v_i \tag{2-14}$$

On the other hand, when expanded, the *model* component takes the form depicted in Equation 2-15, where $\mathbf{v'^m}$ represents the mth possible value that the vector $\mathbf{v} = [v_i \cdots v_V]$ can take ($V = dim(\mathbf{v})$). As $\mathbf{v} \in \mathbb{B}^V$, notice that this formula is computationally expensive to be calculated, as one would need to go through an exponential amount of 2^V possible configurations for $\mathbf{v}$.

$$\langle v_i h_j \rangle_{model} \rightarrow \sum_{\mathbf{m}} p(\mathbf{v'^m}) p(h_j = 1|\mathbf{v'^m}) v_i \tag{2-15}$$

For now, we have seen that for each $\mathbf{v}$ training data, in order to maximize $\log p(\mathbf{v})$ via gradient one needs to estimate:

- The $\langle \cdot \rangle_{\text{data}}$ component: This task can be performed in polynomial time, as from Equation 2-9, we have seen that it only takes going through the values for the given training data $\mathbf{v}$.
- The $\langle \cdot \rangle_{\text{model}}$ component: Conversely, this term is more computationally expensive as, although we have seen an approximation for calculating the marginal distribution in Section 2.4, according to Equation 2-15, we are still summing an exponential amount of possible values for $\mathbf{v}$.

That being said, additional approximations are needed in the interest of having a computationally feasible RBM training process. In the following two subsections, we will explain the two main methods for training RBMs

that can meet these requirements: the Contrastive Divergence and Persistent Contrastive Divergence.

2.5.1

Contrastive Divergence

At the beginning of this section, we have introduced that the training of an RBM consists of maximizing the log-likelihood of $p(\mathbf{v})$ over the training set. According to Hinton, this is the same as minimizing the Kullback-Leibler divergence (KL divergence), also known as Relative Entropy, between the distribution of the visible units in the training dataset and their distributions in the starting RBM model [16].

Using this concept, Hinton outlined an algorithm for training RBMs called Contrastive Divergence (CD). From lengthy calculations that are disclosed in Appendix C, CD updates the hyperparameters of an RBM according to the following equation, where α is the learning rate.

$$\begin{aligned} w_{ij} &= w_{ij} + \alpha\Big(p(h_j = 1|\mathbf{v})v_i - p(h_j = 1|\mathbf{v^k})v_i^k\Big) \\ a_i &= a_i + \alpha\Big(v_i - v_i^k\Big) \\ b_j &= b_j + \alpha\Big(p(h_j = 1|\mathbf{v}) - p(h_j = 1|\mathbf{v^k}\Big) \end{aligned} \tag{2-16}$$

Now, we are ready to lay out a pseudo-code for training RBMs with the CD method in Algorithm 2, where we have the following parameters:

- rbm: An object containing the hyperparameters $\{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$
- data: the dataset, containing the $\mathbf{v}$ samples
- k: the number of Gibbs Sampling steps
- α: the learning rate

It is worth mentioning that the training process of neural networks is often repeated in epochs, which means that the function ContrastiveDivergence defined in Algorithm 2 would be repeated for the number of epochs chosen.

Moreover, although CD already introduces significant performance improvements for RBM learning, there are still some areas for improvement. In the following subsection, we will present the Persistent Contrastive Divergence, which, in sum, reduces the number of GibbsSampling calls.

Algorithm 2 Contrastive Divergence (CD) pseudocode

function ContrastiveDivergence(rbm, data, k, α)
  for $\mathbf{v} \in$ `data` **do**
    $\mathbf{h} \leftarrow p(\mathbf{h}|\mathbf{v})$
    $\mathbf{v^k} \leftarrow$ GibbsSampling(rbm, $k, \mathbf{v}$)
    $\mathbf{h^k} \leftarrow p(\mathbf{h}|\mathbf{v^k})$
    rbm.$w_{ij} \leftarrow w_{ij} + \alpha(\mathbf{v}[i] * \mathbf{h}[j] - \mathbf{v^k}[i] * \mathbf{h^k}[j]), \forall i, j$
    rbm.$a_i \leftarrow a_i + \alpha(\mathbf{v}[i] - \mathbf{v^k}[i]), \forall i$
    rbm.$b_j \leftarrow b_j + \alpha(\mathbf{h}[j] - \mathbf{h^k}[j]), \forall j$
  end for

2.5.2

Persistent Contrastive Divergence

From Algorithm 2, we have seen that the CD procedure is performed several times within the training of an RBM. Considering the sole cost of running Gibbs Sampling, researchers usually choose only one step, exchanging better gradient estimations for better performance. With that in mind, Tieleman proposed a variation for the CD algorithm, known as Persistent Contrastive Divergence (PCD) [17].

According to Tieleman, at each hyperparameter update, the RBM model is only slightest changed, influenced by the chosen learning rate. In this regard, the sampled variables from Gibbs Sampling from one dataset sample to another are not be very different. That being said, they suggested to dividing the training dataset in mini-batches and, after each mini-batch being used to train an RBM, the Gibbs Sampling would be performed to update what they named *fantasy* points.

The *fantasy* points are an array that is storing visible and hidden nodes' sampled values from Gibbs Sampling. There are some specifications about *fantasy* points, where for a dataset divided into mini-batches, it is advised to have a number of *fantasy* arrays equal to the size of a mini-batch.

For easier understanding, the PCD method is outlined in Algorithm 3 , with two extra parameters (in comparison with Algorithm 2):

- miniBatches: the dataset partitioned in small sets of $\mathbf{v}$ samples
- fantasyPoints: an array of *fantasy* points arrays, that carry the sampled values from Gibbs Sampling

It is worth noting that, as we do not reset the *fantasy* points after each run of PCD, we are persisting with the Gibbs Sampling's Markov Chain. This feature

Algorithm 3 Persistent Contrastive Divergence (PCD) pseudocode

```
function PCD(rbm, miniBatches, fantasyPoints, k, α)
    for miniBatch ∈ miniBatches do
        for fantasyPoint ∈ fantasyPoints do
            fantasyPoint ← GibbsSampling(rbm, k, fantasyPoint)
        end for
        for v ∈ miniBatch do
            h ← p(h|v)
            v^k ← fantasyPoints[miniBatch].v
            h^k ← fantasyPoints[miniBatch].h
            rbm.w_ij ← w_ij + α(v[i]h[j] − v^k[i]h^k[j]), ∀i, j
            rbm.a_i ← a_i + α(v[i] − v^k[i]), ∀i
            rbm.b_j ← b_j + α(h[j] − h^k[j]), ∀j
        end for
    end for
end function=0
```

is what inspired the name Persistent Contrastive Divergence.

Moreover, as we perform fewer Gibbs Sampling iterations, the algorithm provides better performance than standard CD.

2.6

RBM Variants

Until now, we have seen RBMs in their standard form: comprised solely of binary units, they are used to model a dataset and learn the underlying distributions of the visible and hidden variables. In this section, we will present two RBM variants used for this work: Gaussian-Bernoulli RBMs and Discriminative RBMs.

2.6.1

Gaussian-Bernoulli RBM

Considering that an RBM is made of binary nodes in its original form, its variables follow a Bernoulli distribution. This configuration, however, limits the RBM of learning from continuous-variable datasets.

That being said, there is a special RBM variation that addresses real-valued data for the visible nodes, known as Gaussian-Bernoulli Restricted Boltzmann Machine (GRBM) [11; 18]. There are different proposals for GRBMs; however, in this work, we will refer to the one presented by Hinton [11], where its energy function is defined as:

$$E(\mathbf{v}, \mathbf{h}) = \sum_i \frac{(v_i - a_i)^2}{2\sigma^2} - \sum_j b_j h_j - \sum_{ij} w_{ij} \frac{v_i}{\sigma_i} h_j \tag{2-17}$$

where σ_i is the variance for the ith visible variable.

For easier calculations, it is common to normalize each component of the training data to have zero mean and unit variance. This simplifies the energy function to the following equations for conditional probability [19] presented in Equation 2-18. It is very important to highlight that the function σ in this equation does not relate to the variance σ_i for a visible node, which from now on will always be equal to 1. Additionally, $\mathcal{N}$ is a normal distribution with mean $a_i + \sum_j h_j w_{ij}$ and variance 1.

$$\begin{aligned} p(h_j = 1|\mathbf{v}) &= \sigma(b_j + \sum_i v_i w_{ij}) \\ p(v_i|\mathbf{h}) &= \mathcal{N}(a_i + \sum_j h_j w_{ij}, 1) \end{aligned} \tag{2-18}$$

2.6.2

Discriminative RBM

RBMs were initially presented using generative training examples, such as image reconstruction using the MNIST dataset[1] [17; 18], however, they can also be used for classification tasks [11; 15; 20].

As we are already familiar with, an RBM is comprised of a visible layer, which takes samples from the dataset, and a hidden layer, which is responsible for modeling underlying features from the dataset. In generative training, we want our RBM to model, for instance, the digits from the MNIST dataset. After training, an RBM is expected to be able to reconstruct corrupted images from this model or even generate new ones. These digits can be translated into vectors of zeros and ones, which are then fed to the visible nodes of the RBM, as presented in Figure 2.6.

On the other hand, when considering RBMs for discriminative training, along with the vector representing the digit we would like to model, a label, in the form of a vector is also provided to the RBM visible nodes, as seen in Figure 2.7.

[1]MNIST Dataset: https://en.wikipedia.org/wiki/MNIST_database

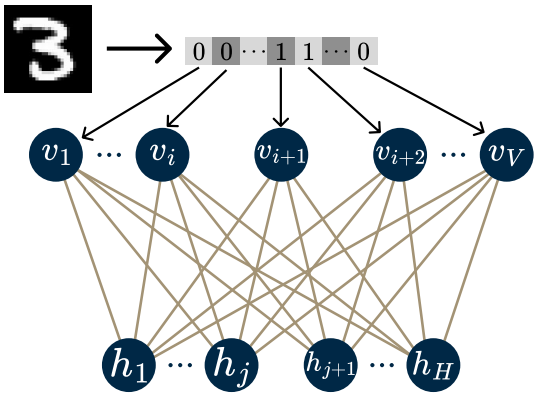

Figure 2.6: Generative training of an RBM

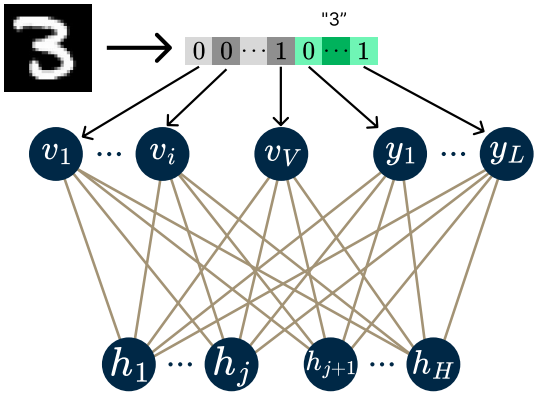

Figure 2.7: Discriminative training of an RBM

From Figure 2.7, we can see that nothing has actually changed in the structure of the RBM, besides the addition of some extra visible nodes (and their connections with hidden nodes). However, we can separate the visible nodes from the generative training (the ones that make up the digit 3 in the example) from the label nodes, yielding the following equation for the energy function [20].

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in\text{visible}} a_i v_i - \sum_{l\in\text{label}} c_l y_l - \sum_{j\in\text{hidden}} b_j h_j - \sum_{ij} w_{ij} v_i h_j - \sum_{lj} u_{lj} y_l h_j \quad (2\text{-}19)$$

Notice that we have added a new weight matrix $\mathbf{U}$, besides the bias $\mathbf{c}$ for the label nodes. At the end of the day, this matrix is actually part of the original $\mathbf{W}$ matrix for the standard RBM.

$$W_{\text{original}} \equiv \begin{bmatrix} W_{\text{generative}} & U_{\text{label}} \end{bmatrix} \quad (2\text{-}20)$$

When presenting this RBM variant, Larochelle [20] outlined two possible training techniques: one that treats the label nodes as if they were regular visible nodes and another that is a hybrid of training with and without labels. The latter was justified by the amount of labeled training data that is usually available. In this work we will not cover the second training method, which can be addressed in a future project.

2.7

Deep Belief Networks

Although this study does not delve into Deep Belief Networks (DBNs), they are an important milestone in the development of deep neural networks and therefore will be briefly presented.

Introduced by Hinton *et al* [21], DBNs have the advantage of modeling more complex data, with non-linear features [21; 22]. In sum, DBNs are comprised of stacked RBMs, where the hidden nodes for the i-th RBM are the visible nodes for the *(i+1)-th* RBM. Figure 2.8 presents a DBN in its usual representation, with the visible nodes at the bottom.

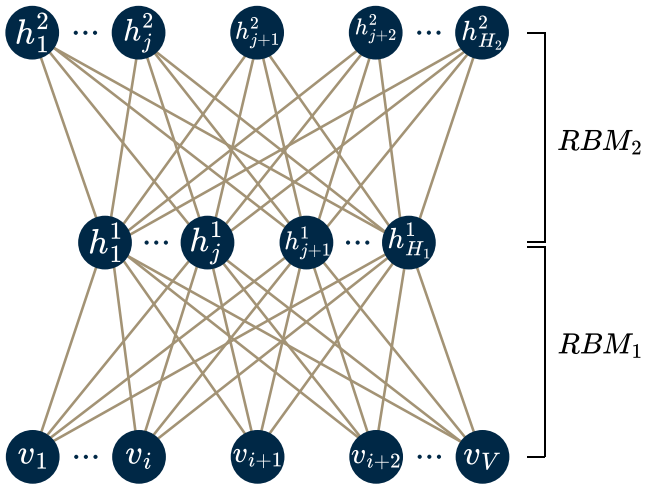

Figure 2.8: Illustration of a Deep Belief Network (DBN), with two hidden layers

There are two stages for training a DBN, as proposed by Hinton *et al* [21; 23]. First it is performed a greedy layer-wise training, where each RBM that makes the DBN is trained in order. The first RBM, the one at the bottom with the visible nodes in Figure 2.8 takes the samples from the dataset and has its weights and biases trained. Then, the following RBM, comprised by the nodes $\mathbf{h^1}$ and $\mathbf{h^2}$ takes as input the samples from the dataset transformed from the previous RBM. For example, considering a sample $\mathbf{s}$ from the dataset, the second RBM would receive $p(\mathbf{h^1}|\mathbf{v} = \mathbf{s})$ as input. Then, a fine-tuning step

introduces a classification layer that is training via back propagation that also adjusts the weights of the original DBN.

2.8

Quantum Computing and RBMs

In Section 2.5 we have seen that training an RBM can be interpreted as minimizing the KL divergence between the model of the training dataset and the current model of this RBM. During this learning process, for both algorithms that were presented, we had to sample a state from the RBM distribution using Gibbs Sampling.

As it is retrieved from an approximation to the RBM distribution, this sampled state is an RBM configuration for the visible and hidden nodes that has a greater possibility of occurrence. That being said, considering Equation 2-3, this sample has a lower energy value $E(\mathbf{v}, \mathbf{h})$.

Moreover, as mentioned at the beginning of this chapter, EBMs inherits concepts from MRFs, which in turn uses notions from Statistical Mechanics. In this regard, the energy function of the RBM derives from the Ising Model from Statistical Mechanics, depicted in Equation 2-21. This formulation has an $n \times n$ matrix $\mathbf{J}$ and a vector $\mathbf{h}$ of size n.

$$H(\sigma) = -\sum_{ij} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i, \text{ where } \sigma_k \in \{\pm 1\} \tag{2-21}$$

The energy function from an RBM can be easily mapped into the Ising Model with a few adjustments. First of all, we need to convert the visible and hidden variables, which are binary, into $\sigma_i \in \{\pm 1\}$ variables. This conversion is trivial as presented in Equation 2-22.

$$\sigma_i = 2x - 1, \text{ where } x \text{ is either a visible or a hidden variable} \tag{2-22}$$

Then, as $\mathbf{J}$ is an $N \times N$ matrix and $\mathbf{W}$ is a $V \times H$ matrix (where V and H are the number of visible and hidden nodes respectively), when performing this reformulation, the resulting $\mathbf{J}$ matrix must have dimensions $(V+H) \times (V+H)$. Now, considering that we would be summing $w_{ij}\sigma_i\sigma_j$ two times in the Ising model, we need to divide the weights by two. Finally, the $\mathbf{h}$ vector is a

concatenation of the **a** and **b** vectors. These manipulations are presented in Figure 2.9.

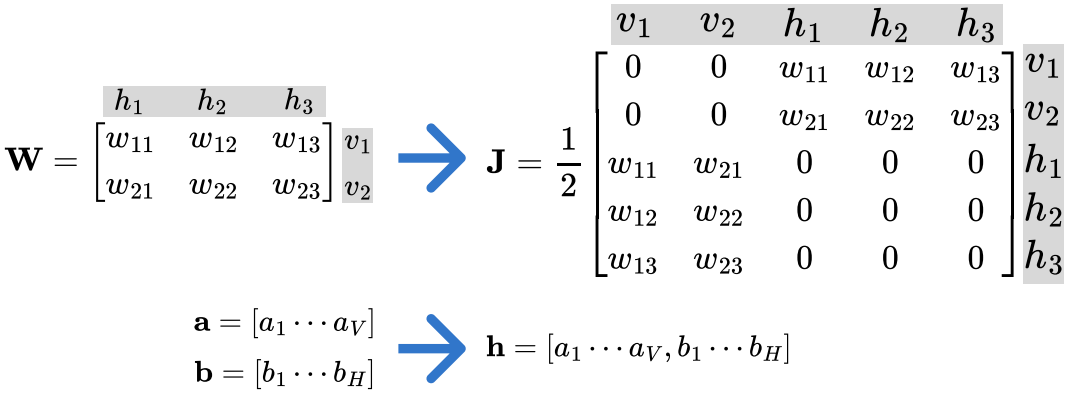

Figure 2.9: RBM weights conversion into the Ising Model coefficients

The Ising Spin Model represents the energy assigned to a spin lattice, where the possible values for the spin particles are their dipole moments 1 and -1. Therefore, when performing Gibbs Sampling, we are, in fact, looking for a value for σ that minimizes $H(\sigma)$.

That being said, as we will see in Chapter 4, there are Quantum Computing methods specialized in finding low energy states for the Ising Model and a binary representation of this formula known as the Quadratic Unconstrained Binary Optimization (QUBO) model. Therefore, ongoing research projects are analysing the usage of quantum computers to sample these low-energy states, instead of the Gibbs Sampling method [1; 9; 10; 24].

These quantum sampling methods are substitutes for Gibbs Sampling, so they can be applied either during CD or PCD training. Algorithm 4 provides an example of PCD using quantum sampling.

Algorithm 4 PCD via Quantum Sampling pseudocode

```
function QuantumSampling(rbm, miniBatches, α)
    for miniBatch ∈ miniBatches do
        QUBO ← generateQUBO(rbm)
        v_model, h_model ← QuantumSample(QUBO)
        for v ∈ miniBatch do
            h ← p(h|v)
            rbm.w_ij ← w_ij + α(v[i]h[j] − v_model[i]h_model[j]), ∀i, j
            rbm.a_i ← a_i + α(v[i] − v_model[i]), ∀i
            rbm.b_j ← b_j + α(h[j] − h_model[j]), ∀j
        end for
    end for
end function
```

In the following chapters, we introduce some basic Quantum Computing concepts and present these quantum sampling techniques. However, it is important to highlight that quantum-assisted training consists of replacing a subroutine of RBM training, usually performed by Gibbs Sampling, with a quantum algorithm capable of finding low-energy states for the RBM energy function.

3

Quantum Computing Concepts

In Chapter 4, we will introduce quantum algorithms capable of sampling low-energy states for functions that are compatible with the energy function of an RBM. That said, coming from a background outside the field of quantum technologies might make understanding these quantum methods more difficult. This chapter presents an overview of the necessary Quantum Computing topics to understand the aforementioned chapter.

3.1

Qubits

A qubit is analogous to the 'bit' from classical computation, where it is the 'smallest' unit of information. According to the first postulate of Quantum Mechanics [25], it is represented by a vector in a two-dimensional complex vector space ($\mathbb{C}^2$) called the Hilbert Space ($\mathcal{H}^2$).

Qubits are usually depicted in the Dirac Notation [25], where a column vector is represented with what we call a 'ket' $|\cdot\rangle$ and a row vector is portrayed with a 'bra' $\langle\cdot|$, which makes this notation be recognized as the Bra-ket Notation. It is worth mentioning that for a state $|\psi\rangle$, its 'row vector representation' is its transpose conjugate ($|\psi\rangle = \langle\psi|^\dagger$). Borrowing from classical computation, which uses bits with values 0 or 1, we have the $|0\rangle$ and $|1\rangle$ states that are the ground and excited states, respectively. Their portrayal in the Bra-ket notation is as follows.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \text{ while, } \langle 0| = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \langle 1| = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{3-1}$$

Moreover, an arbitrary qubit $|\psi\rangle$ can be represented by a linear combination of the $|0\rangle$ and $|1\rangle$ states, as presented in the equation below.

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle, \quad \text{where } \left|\cos\frac{\theta}{2}\right|^2 + \left|e^{i\phi}\sin\frac{\theta}{2}\right|^2 = 1 \tag{3-2}$$

As we will see in Section 3.4, when observing a qubit in the computational basis, their state collapses to either $|0\rangle$ or $|1\rangle$, where the coefficients that indicate the linear combination of $|0\rangle$ and $|1\rangle$ can be used to known the probabilities of $|\psi\rangle$ falling into either state.

Following Equation 3-2, we can graphically represent a qubit $|\psi\rangle \in \mathbb{C}^2$ using a Bloch Sphere. Bloch Sphere is a depiction of a qubit as a unitary vector, parameterized by the values for θ and ϕ, which were introduced in Equation 3-2.

As its name suggests, the Bloch Sphere is a spherical representation, which allows us to represent a qubit in $\mathbb{C}^2$ in a three-dimensional space. The unitary vector for a qubit $|\psi\rangle$ can be estimated as follows in Equation 3-3, while the Bloch Sphere is displayed in Figure 3.1.

$$\begin{aligned}
&\vec{\psi} = (x, y, z), \text{ where} \\
&x = \cos\phi\sin\theta \\
&y = \sin\phi\sin\theta \\
&z = \cos\theta
\end{aligned} \tag{3-3}$$

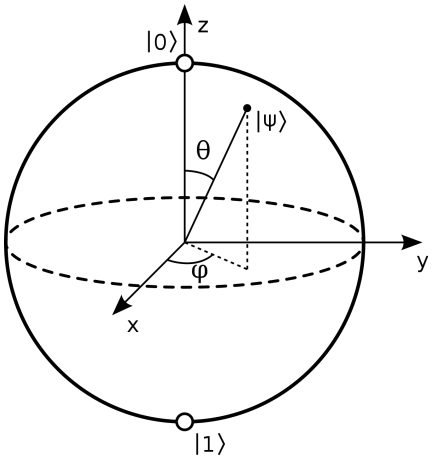

Figure 3.1: Bloch Sphere
Source:Wikipédia

3.2

Dirac notation

In the previous section, we have introduced the concept of a qubit and how its vector representation is denoted in the Dirac notation [25], which is also known as the Bra-ket notation. Besides shortening the writing of vectors, this syntax provides a simple way for expressing vector operations, as outlined in Table 3.1.

Operation	Representation
Inner Product	$\langle y\|x\rangle$
Outer Product	$\|x\rangle\langle y\|$
Tensor Product	$\|x\rangle \otimes \|y\rangle$ or $\|x\rangle \|y\rangle$ or $\|xy\rangle$
Matrix Multiplication	$A\|x\rangle$, $\langle x\|A$

Table 3.1: Dirac notation for vector operations

In this table, it is worth highlighting that the tensor product is used to represent multiple qubit states, where the $\otimes$ symbol in Table 3.1 is the Kronecker Product. For instance, if we had a two-qubit state $|\psi_1\psi_2\rangle$, it would have the following vector notation.

$$|\psi_1\psi_2\rangle = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \otimes \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1a_2 \\ a_1b_2 \\ b_1a_2 \\ b_1b_2 \end{bmatrix}, \text{ where } a_k = \cos\frac{\theta_k}{2} \text{ and } b_k = e^{i\phi_k}\sin\frac{\theta_k}{2} \quad (3\text{-}4)$$

3.3

Operators

Because qubits are vectors in the Hilbert space, operations on them can be represented as matrices. The second postulate of Quantum Mechanics states that the evolution of a closed quantum system (without external influences) can be described by an unitary transformation [25]. Equation 3-5 represents a state $|\psi\rangle$ going to the state $|\psi'\rangle$ after having an unitary operator (matrix) U applied to it.

$$|\psi'\rangle = U|\psi\rangle = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} \cos\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} \end{bmatrix} \quad (3\text{-}5)$$

For context, an unitary operator is an operator that satisfies the following property.

$$UU^\dagger = U^\dagger U = I \tag{3-6}$$

There are some well known operators, such as the Pauli matrices, the Identity and the Hadamard operator (H). Considering qubit interactions, there are operators that act on more than one qubit, such as the CNOT operator. For easier understanding, Table 3.2 presents these operators, with their respective matrices and outcomes when applied to an arbitrary state $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$.

Label	Matrix	Result
I	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$I\,\|\psi\rangle = \|\psi\rangle$
X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$X\,\|\psi\rangle = a\,\|1\rangle + b\,\|0\rangle$
Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$Y\,\|\psi\rangle = a \cdot i\,\|1\rangle - b \cdot i\,\|0\rangle$
Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$Z\,\|\psi\rangle = a\,\|0\rangle - b\,\|1\rangle$
H	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$H\,\|\psi\rangle = a\,\|+\rangle + b\,\|-\rangle$
CNOT	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$CNOT\,\|\psi_1\psi_2\rangle = a_1a_2\,\|00\rangle + a_1b_2\,\|01\rangle + b_1a_2\,\|11\rangle + b_1b_2\,\|10\rangle$

Table 3.2: Qubit operators and their respective results

It is worth noting that for the Hadamard operator we have introduced the $|+\rangle$ and $|-\rangle$ terms, which have the following representations in Equation 3-7. These states denote a superposition between the states $|0\rangle$ and $|1\rangle$, as $a^2 = b^2 = 1/2$, which indicates that they have an equal probability of being measured. In the next section we will have a better understanding what these probabilities are.

$$|+\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \quad , \quad |-\rangle = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle \tag{3-7}$$

Moreover, the CNOT matrix works as a 'conditional operator', as if the first qubit $|\psi_1\rangle$ is $|1\rangle$, the state of the next qubit is flipped.

3.4

Measurements

From the third postulate of Quantum Mechanics [25], quantum measurements are comprised of a set $\{M_m\}$ of measurement operators that act on the state space of the system being measured. The subscript m represents a possible measurement outcome where, if a system is in a state $|\psi\rangle$, the probability of the estimation yielding the eigenstate of M_m is:

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle \tag{3-8}$$

This concept of measurement from the third postulate comes with a few requirements. First, the sum of the operators M_m must be in terms of the *completeness equation* that guarantees that the sum of all probabilities is equal to one, as described in the following equations.

$$\sum_m M_m^\dagger M_m = I \tag{3-9}$$

$$\sum_m p(m) = \sum_m \langle\psi| M_m^\dagger M_m |\psi\rangle = \langle\psi| \left(\sum_m M_m^\dagger M_m \right) |\psi\rangle = \langle\psi| I |\psi\rangle = \langle\psi|\psi\rangle = 1 \tag{3-10}$$

Additionally, there is a specific type of measurement, called Projective Measurement [25]. In this class of measurement, the set of M_m operators, which we will now label as P_m, are projectors into the eigen-space of an observable M, that is an Hermitian matrix that has the following spectral decomposition described in Equation 3-11, where λ_m is the eigenvalue correspondent to the projector P_m. Besides being projectors to an eigen-space, Projective Measurements have the particularity of their operators being orthogonal.

$$M = \sum_m \lambda_m P_m \tag{3-11}$$

For instance, let us consider that we are performing a projective measurement where the observable is the Pauli Z, where

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = 1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} - 1 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = 1P_0 - 1P_1 \tag{3-12}$$

Then, following Equation 3-8, the outcome of measuring the $p(1)$ for a qubit $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$ would be:

$$p(1) = \langle\psi|\, P_1^\dagger P_1\, |\psi\rangle \tag{3-13}$$

As the operators P_m are Hermitian projectors, we can use the fact that $P_m^\dagger P_m = P_m$

$$\begin{aligned} p(1) = \langle\psi|\, P_1^\dagger P_1\, |\psi\rangle &= \langle\psi| \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} |\psi\rangle = \langle\psi|\,|1\rangle\,\langle 1|\,|\psi\rangle \\ &= (a\,\langle 0| + b\,\langle 1|)\,|1\rangle\,\langle 1|\,(a\,|0\rangle + b\,|1\rangle) \\ &= (a\,\langle 0|1\rangle\, b\,\langle 1|1\rangle)(a\,\langle 1|0\rangle + b\,\langle 1|1\rangle) = b^2 \end{aligned} \tag{3-14}$$

Projective Measurements also provide the concept of expected value (the average value) of an observable M. Thus, for a state $|\psi\rangle$, we say that the average value of M is:

$$\begin{aligned} \langle M\rangle_\psi &= \sum_m \lambda_m\, p(m) = \sum_m \lambda_m\, \langle\psi|\, P_m\, |\psi\rangle \\ &= \langle\psi|\, \Big(\sum_m \lambda_m P_m\Big)\, |\psi\rangle = \langle\psi|\, M\, |\psi\rangle \end{aligned} \tag{3-15}$$

Thus, considering our arbitrary state $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$, $\langle M\rangle_\psi$ would be:

$$\langle M\rangle_\psi = \lambda_0\, a^2 + \lambda_1\, b^2 = a^2 - b^2 \tag{3-16}$$

3.5

The Quantum Circuit Model

A sequence of quantum operations on qubits is usually represented in the quantum circuit model. It has some similarities with the notation of logic circuits, where we have 'tracks' of bits that go through gates, that now are 'tracks' of qubits, usually initiated in the $|0\rangle$ state, that go through what we call 'quantum gates' (which are the operators from Section 3.3).

Using the example of a quantum circuit displayed in Figure 3.2, we will explain some quantum circuit notations and how this representation translates into actual linear algebra operations.

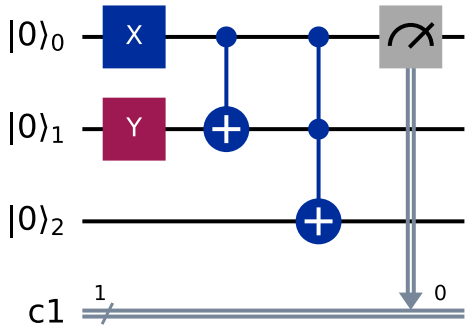

Figure 3.2: Example of a quantum circuit with 3 qubits

According to DiVincenzo's criteria [26], qubits are usually initialized in a fixed, predetermined fiducial state in a quantum circuit, which is usually set to the ground state $|0\rangle$. Thus, for this example, we have three qubits $|0\rangle_0$, $|0\rangle_1$ and $|0\rangle_2$, each with their corresponding 'track'. There is an additional track, with the `c1` symbol, which represents a classical bit that will store the measurement of the qubit $|0\rangle_0$, as we will see later in this explanation.

These three qubits are operated with a total of four quantum gates. Although some of them were already presented in Section 3.3, here we give a brief overview of all gates in this circuit.

- X gate: a one-qubit gate that flips the qubit state (from $|0\rangle$ to $|1\rangle$ and vice-versa)
- Y gate: also a one-qubit gate that flips the qubit state, although adding a i coefficient as seen in the previous section

- $CNOT$ gate: a two-qubit gate, known as the controlled-not gate. It takes a *control* and *target* qubits. If the *control* qubit is in the $|1\rangle$ state, the *target* qubit's state is flipped, as if it were being applied an X gate
- $CCNOT$ gate: a three-qubit gate, known as the Toffoli or controlled-controlled-not gate. It is similar to the $CNOT$ gate, but it takes two *control* qubits and one *target*. Following the $CNOT$ logic, if both *control* qubits are in the $|1\rangle$ state, the *target* qubit state is flipped

The operations depicted in a quantum circuit are applied in order of appearance, from left to right. Thus, the circuit in Figure 3.2 represents the following operations.

$$\begin{aligned} & CCNOT\Big(CNOT(X\,|0\rangle_0 \otimes Y\,|0\rangle_1) \otimes |0\rangle_2\Big) \\ = {} & CCNOT\Big(CNOT(|1\rangle_0 \otimes i\,|1\rangle_1) \otimes |0\rangle_2\Big) \end{aligned} \tag{3-17}$$

We can disregard the imaginary coefficient of $|1\rangle_1$ as it will not affect the final result for this case. Next, as we have a CNOT applied to $|11\rangle$, we flip the second qubit.

$$\begin{aligned} & CCNOT\Big(CNOT(|1\rangle_0 \otimes |1\rangle_1) \otimes |0\rangle_2\Big) \\ = {} & CCNOT\Big(\,|1\rangle_0 \otimes |0\rangle_1 \otimes |0\rangle_2\Big) \end{aligned} \tag{3-18}$$

As the qubit 1 is in the $|0\rangle_1$ state, we do not flip the qubit 2.

$$CCNOT\Big(\,|1\rangle_0 \otimes |0\rangle_1 \otimes |0\rangle_2\Big) = |1\rangle_0 \otimes |0\rangle_1 \otimes |0\rangle_2 \tag{3-19}$$

At the end, we perform a measurement on qubit 0. As there is no specification for the measurement in Figure 3.2, we consider that it is a measurement on the computational basis. As this qubit is in the $|1\rangle_0$ state, we have a probability of 100% for measuring 1 (according to Equation 3-8). This value is stored in the classical bit $c1$.

3.6

Density Matrices and Mixed States

Now that we have seen how a qubit is operated and measured, there is a more general way of representing the state of a qubit (or more): via the Density Matrix (also known as the Density Operator). For a qubit $|\psi\rangle$, one can easily find its Density Matrix ρ estimating the following outer product:

$$\rho = |\psi\rangle \langle\psi| \tag{3-20}$$

We can substitute the Dirac notation for qubits by density matrices in all scenarios in which we have represented qubits in this chapter. First, one can represent a qubit in state ρ in the Bloch sphere retrieving the values for (x, y, z) in the following equation:

$$\rho = \frac{1}{2}\begin{bmatrix} 1+z & x-iy \\ x+iy & 1-z \end{bmatrix} \tag{3-21}$$

Moreover, operators can be applied to density matrices with no extra effort. In Section 3.3, we have seen that when applying an operator U to a qubit $|\psi\rangle$, its state evolves as

$$U|\psi\rangle = |\psi'\rangle \tag{3-22}$$

Now, using the density operator notation, we would have:

$$U|\psi\rangle\langle\psi|U^\dagger = U\rho U^\dagger = \rho' = |\psi'\rangle\langle\psi'| \tag{3-23}$$

The same follows for measuring a qubit. For a set of measurement operators M_m, the probability of the estimation resulting the eigenstate of M_m is:

$$p(m) = \mathrm{tr}(M_m^\dagger M_m \rho) \tag{3-24}$$

More importantly, the main advantage of the density matrix representation is that, as a more general way of representation, it can depict mixed states. Instead of being just the outer product of $|\psi\rangle$ by its transpose conjugate, for a mixed state ρ represents the linear combination in Equation 3-25, where ρ is a pure state if $p_i = 1$ for some i.

$$\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i| \tag{3-25}$$

A mixed state cannot be represented in the Dirac notation, as one is not able to estimate the $|0\rangle$ and $|1\rangle$ coefficients from this mixture. For instance, let us consider the fully entangled Bell State $|\phi+\rangle$:

$$|\phi+\rangle = \frac{1}{\sqrt{2}} \, |00\rangle + \frac{1}{\sqrt{2}} \, |11\rangle \tag{3-26}$$

We know that this composite state is comprised of two qubits that we might consider that can be represented as:

$$|\psi_1\rangle = a_1 \, |0\rangle_1 + b_1 \, |1\rangle_1 \quad \text{and} \quad |\psi_2\rangle = a_2 \, |0\rangle_2 + b_2 \, |1\rangle_2 \tag{3-27}$$

And, by taking a look at Equation 3-26, we can rewrite it as:

$$\begin{gathered} |\phi+\rangle = a_1 a_2 \, |00\rangle + b_1 b_2 \, |11\rangle + a_1 b_2 \, |01\rangle + b_1 a_2 \, |10\rangle \, , \\ \text{where } a_1 a_2 = b_1 b_2 = {}^1\!/\!\sqrt{2}, \quad a_1 b_2 = b_1 a_2 = 0 \end{gathered} \tag{3-28}$$

For now, we are making a representation of this composite state $|\phi+\rangle$ in $\mathcal{H}^4$. But what happens if we try to 'separate' the states? First, we need to find values for $\{a_1, b_1\}$ and $\{a_2, b_2\}$, with the following constraints:

$$a_1 b_2 = 0 \text{ and } a_2 b_1 = 0 \tag{3-29}$$

Thus, either a_i or b_j coefficients in each restriction in Equation 3-29 must be 0. However, there is no assignment that makes $a_1 a_2 = b_1 b_2 = {}^1\!/\!\sqrt{2}$ possible. This

is because the $|\psi_1\rangle$ and $|\psi_2\rangle$ qubits are found in a mixed state when we try to represent them separately.

Now, let us represent these states as density matrices. First, we will re-write the $|\phi+\rangle$ state as a density operator.

$$\rho_{\phi+} = |\phi+\rangle\langle\phi+| = |\psi_1\rangle\langle\psi_1| \otimes |\psi_2\rangle\langle\psi_2| = \begin{bmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.5 \end{bmatrix} \tag{3-30}$$

In order to find the state for ρ_1, the rules of Quantum Mechanics tell us we need to perform an operation called the partial trace. In this method, one of the subsystems is discarded (traced over), according to the following formula:

$$\begin{aligned} &\rho_1 \equiv \mathrm{tr}_2(\rho), \text{ where} \\ &\mathrm{tr}_2(\rho) = \mathrm{tr}_2(|\psi_1\rangle\langle\psi_1| \otimes |\psi_2\rangle\langle\psi_2|) = |\psi_1\rangle\langle\psi_1| \cdot \mathrm{tr}(|\psi_2\rangle\langle\psi_2|) \end{aligned} \tag{3-31}$$

Using this method, we can find the state of ρ_1 as follows:

$$\begin{aligned} \rho_1 &= \mathrm{tr}_2(\rho_{\phi+}) = \mathrm{tr}_2(|\phi+\rangle\langle\phi+|) \\ &= \mathrm{tr}_2\Big(\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \cdot \frac{1}{\sqrt{2}}(\langle 00| + \langle 11|)\Big) \\ &= \mathrm{tr}_2\Big(\frac{|00\rangle\langle 00| + |00\rangle\langle 11| + |11\rangle\langle 00| + |11\rangle\langle 11|}{2}\Big) \\ &= \frac{\mathrm{tr}_2(|00\rangle\langle 00|) + \mathrm{tr}_2(|00\rangle\langle 11|) + \mathrm{tr}_2(|11\rangle\langle 00|) + \mathrm{tr}_2(|11\rangle\langle 11|)}{2} \\ &= \frac{|0\rangle\langle 0|_1 \,\mathrm{tr}(|0\rangle\langle 0|_2) + \cdots + |1\rangle\langle 1|_1 \,\mathrm{tr}(|1\rangle\langle 1|_2)}{2} \\ &= \frac{|0\rangle\,\langle 0|_1 \cdot 1 + |0\rangle\,\langle 1|_1 \cdot 0 + |1\rangle\,\langle 0|_1 \cdot 0 + |1\rangle\,\langle 1|_1 \cdot 1}{2} \\ &= \frac{1}{2}\left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \right) = \frac{1}{2}\mathbb{I} \end{aligned} \tag{3-32}$$

Thus, density matrices allow us to represent the state of any qubit, whether it is a pure state or a mixed one.

Finally, density operators have the following properties:

1. $Tr(\rho) = 1$
2. $\langle\phi|\,\rho\,|\phi\rangle \geq 0, \forall\,|\phi\rangle \in \mathcal{H}$
3. $Tr(\rho^2) = 1 \iff \rho$ is a pure state
4. $Tr(\rho^2) < 1 \iff \rho$ is a mixed state
5. $\langle O\rangle_\rho = \mathrm{tr}(\rho O)$ (this is the density matrix counterpart for expected value, from Equation 3-15)

3.7

Adiabatic Quantum Computing

Until now, we have demonstrated concepts and applications for the standard branch of QC, comprised of gate-based operations. However, there is a second branch of QC that has been shown to be polynomially equivalent to the gate model [27], known as Adiabatic Quantum Computation (AQC) [28]. In this section, we will give a brief introduction to this paradigm.

AQC was idealized for finding the ground state of a Hamiltonian, starting from another Hamiltonian with a known ground state, following the concepts from the Adiabatic Theorem. According to this theorem, if a system is found in the ground state and then suffers a slow enough transformation, it should continue in the ground state.

That being said, the idea behind AQC algorithms is to define an initial Hamiltonian ($\mathcal{H}_0$), with a known ground state, and the Hamiltonian corresponding to the problem you want to find solutions for ($\mathcal{H}_f$). After setting the quantum system to the ground state of $\mathcal{H}_0$, an adiabatic evolution $s(t)$ transitions the quantum system into $\mathcal{H}_f$, adhering to the following time-varying Hamiltonian in Equation 3-33, where t varies from $0 \to t_f$.

$$\begin{aligned} &\mathcal{H}(t) = s(t)\mathcal{H}_0 + (1 - s(t))\mathcal{H}_f, \text{ where} \\ &\quad t : 0 \to t_f, \\ &\quad s(t = 0) = 1 \text{ and } s(t = t_f) = 1 \end{aligned} \tag{3-33}$$

The transition from $\mathcal{H}_0$ to $\mathcal{H}_f$ is required to be slow enough because during the procedure, the quantum system can jump from the ground state to the first excited state, thus preventing the algorithm to find the optimal solution.

According to Equation 3-34 [28], this 'changing rate', $\tau(s)$, is dictated by the minimum spectral gap (δ_m) between the first excited state (λ_1) and the ground state (λ_{GS}), where $\tilde{\mathcal{H}}(s)$ is the equivalent Hamiltonian for $\mathcal{H}(t)$, but in the time scale $s : 0 \to 1$ that evolves at a rate set by t_f, from Equation 3-33. The minimum spectral gap is illustrated in Figure 3.3.

$$\tau(s) \gg \frac{||\frac{\partial}{\partial s}\tilde{\mathcal{H}}(s)||}{(\delta_m)^2} \tag{3-34}$$

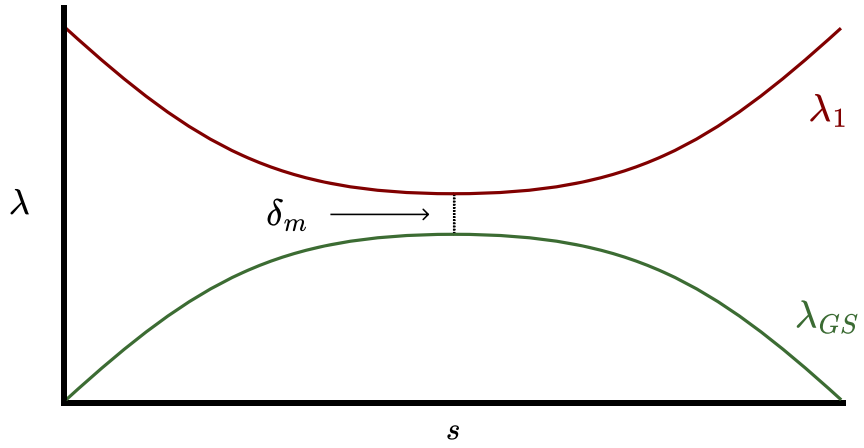

Figure 3.3: Illustration of the minimum spectral gap δ_m between the ground state λ_{GS} and the first excited state λ_1 for the Adiabatic Theorem

4

Quantum Sampling Techniques

4.1

Ising and QUBO models

As mentioned at the end of Chapter 2, quantum computers can participate in the classical training process of RBMs by sampling low-energy states that will be used in the calculation of the learning gradient. This is possible because the energy function of an RBM is based on the Ising Model, which, when translated to its quantum version, is compatible with quantum sampling techniques. The optimization problem for finding a low-energy state in the Ising Model is depicted in the equation below.

$$\min_{s\in\{\pm 1\}} \mathbf{s}'\mathbf{J}\mathbf{s} + \mathbf{h}'\mathbf{s} = \min_{s\in\{\pm 1\}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} J_{ij} s_i s_j + \sum_{i=1}^{n} h_i s_i \tag{4-1}$$

where $\mathbf{J} \in \mathbb{R}^{n\times n}$ is upper triangular and $\mathbf{h} \in \mathbb{R}^{\mathbf{n}}$

There is also another optimization problem formulation which can be converted into the Ising Model known as a Quadratic Unconstrained Binary Optimization (QUBO) problem, depicted in Equation 4-2.

$$\min_{\mathbf{x}\in\mathbb{B}^n} \mathbf{x}'\mathbf{Q}\,\mathbf{x} = \min_{\mathbf{x}\in\mathbb{B}^n} \sum_{i=1}^{n} \sum_{j=i+1}^{n} Q_{ij}\, x_i\, x_j + \sum_{i=1}^{n} Q_{ii} x_i \tag{4-2}$$

That being said, Quantum Sampling techniques can be perceived as Quantum Optimization problems where we want to find values that minimize $\mathbf{s}'\mathbf{J}\mathbf{s}$ for the case of Ising Models and $\mathbf{x}'\mathbf{Q}\mathbf{x}$ for QUBOs.

From their acronym, QUBOs are optimization problems with a quadratic (at most) objective function, without constraints, and comprised solely of binary variables. On the other hand, one can notice that the Ising Model in Equation

4-1 contains variables $s \in \{\pm 1\}$, thus making it a non-binary optimization problem. Fortunately, there is a simple mapping between the two models.

- From Ising to QUBO: convert each variable s to $x = \frac{1}{2}(s+1)$
- From QUBO to Ising: convert each variable x to $s = 2x - 1$

Furthermore, as we have seen in Subsection 2.6.1, one might need to work with continuous datasets to model their RBM. This might impose a challenge, as QUBOs are supposed to have only binary variables (and Ising models only $s \in \{\pm 1\}$ variables). That said, there are some variable encoding techniques that one can harness to represent continuous data in binary variables. Considering that there are lengthy calculations involved and that the QUBO reformulation process is not straightforward, we have dedicated Appendix D to explain how variable encodings work and to present an example of QUBO reformulation from a well-known problem in the energy sector.

For the rest of this chapter, we will present Quantum Sampling techniques for sampling low-energy states from the Ising/QUBO models.

4.2

Variational Quantum Algorithms

The current stage of QC, known as the NISQ Era, is characterized by devices that are limited to a small number of operations due to hardware constraints. Amidst this scenario, a class of quantum algorithms has been developed to address the search for quantum advantage. This family of algorithms, coined as Variational Quantum Algorithms (VQA) [6], is comprised of hybrid methods, where a classical computer aids the quantum hardware in performing a specific task of the protocol.

The main goal of a VQA is to find the optimal parameters $\boldsymbol{\theta}^*$ that minimize a cost function $C(\theta)$, frequently referred to as the loss function. In its generic format, a VQA can be portrayed as presented in Figure 4.1, where a quantum circuit estimates the value of the cost function for a given set of parameters $\boldsymbol{\theta}$ and feed it to a classical optimizer, that will be responsible for selecting new parameters $\boldsymbol{\theta_{i+1}}$.

In the next sections, some essential elements of VQAs will be discussed, followed by the current challenges that this framework presents. The topics that will be covered require some previous knowledge on Quantum Computing concepts, which can be found in Chapter 3.

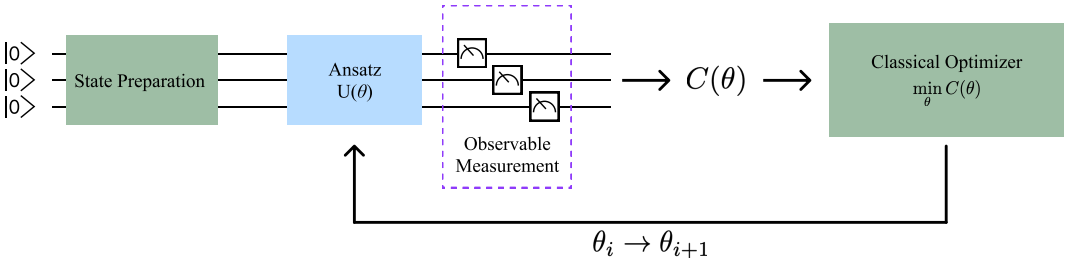

Figure 4.1: Variational Quantum Algorithm (VQA) schematics

4.2.1

Cost Function

The cost function, or loss, as previously mentioned, is responsible for mapping the $\boldsymbol{\theta}$ parameters to real numbers that can be later assessed by the classical optimizer.

A more comprehensive representation of the cost function is presented in Equation 4-3, where ρ represents the initial state, O the observable, and $U(\boldsymbol{\theta})$ a parameterized operator, representing the so-called ansatz. In sum, the cost function is the expected value of an observable O for a state $\rho_\theta \equiv U(\theta)\rho U^\dagger(\theta)$.

$$C_{\rho,O}(\boldsymbol{\theta}) = \mathrm{tr}(OU(\boldsymbol{\theta})\rho U^\dagger(\boldsymbol{\theta})) \tag{4-3}$$

Moreover, when working with a QUBO or Ising Model, this observable O is, actually, a derivation of the $\mathbf{Q}$ or $\mathbf{J}$ matrices (with a few adjustments).

When outlining a cost function, one needs to adhere to the following desirable criteria, presented by Cerezo *et al* [6].

i. A cost function should be consistent with the problem being tackled so that its minimum value corresponds to the solution of the problem;

ii. Besides mapping the global optimum to the solution of the problem, smaller values should indicate better results;

iii. The cost function, or its gradient, should be intractable or at least sufficiently hard to be simulated on classical hardware while being simple to be estimated on a quantum computer. This requisite is important to attain quantum advantage;

iv. The cost function should be *trainable*, i.e., the $\boldsymbol{\theta}$ parameters should be easy to be optimized.

4.2.2

Ansatz

The *Ansatz* is a parameterized quantum circuit $U(\boldsymbol{\theta})$ that takes the $\boldsymbol{\theta}$ parameters to dictate operations, as presented in Equation 4-4, where W_k is a fixed unitary operation. Figure 4.2 presents an example of ansatz.

$$U(\boldsymbol{\theta}) = \prod_{k=1}^{K} U_k(\boldsymbol{\theta}_k) W_k \tag{4-4}$$

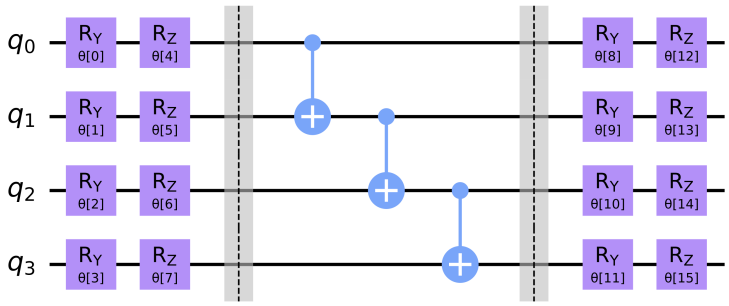

Figure 4.2: Example of an ansatz

The combination of possible parameters for the ansatz compose the accessible space of solutions $\mathbb{U}$, depicted in Equation 4-5.

$$\mathbb{U} = \{U(\boldsymbol{\theta}_1), U(\boldsymbol{\theta}_2), \ldots, U(\boldsymbol{\theta}_n)\} \tag{4-5}$$

There are two initial important concepts for ansätze. First, "*Expressibility*" corresponds to the size of the accessible space of solutions $\mathbb{U}$. Second, considering a problem A that a VQA is trying to solve, its ansatz is said to be "*complete*" if $\mathbb{U}_A \cap \mathbb{U} \neq \emptyset$ [29]. That being said, the following cases are possible:

a. an ansatz may be expressive and contain a solution to A;

b. an ansatz may be expressive but not contain a solution to A;

c. an ansatz may be inexpressive and contain a solution to A;

All three scenarios are illustrated in Figure 4.3.

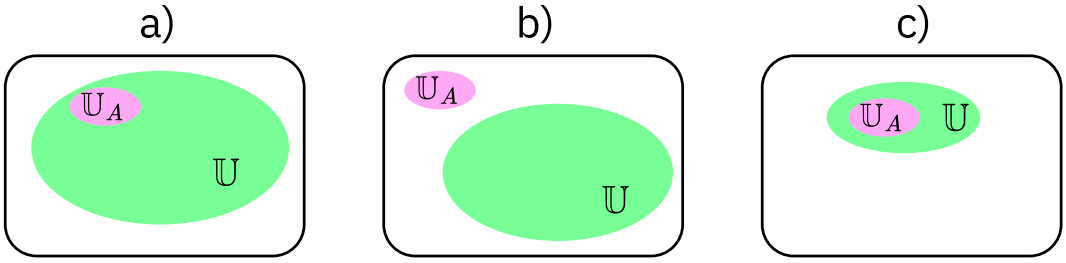

Figure 4.3: Three possible scenarios for an ansatze: *a)* expressive and complete; *b)* expressive but not complete; *c)* inexpressive and complete

4.2.3

Current challenges with VQAs

The field of VQAs is a vibrant and ongoing topic. Researchers have been trying to understand and tackle the current challenges that limit this framework. As previously mentioned, these hybrid quantum-classical parameterized algorithms were presented as candidates for short or medium-term quantum advantage, as other entirely quantum methods require fault-tolerant hardware. However, although being designed in the midst of the NISQ era, VQAs still suffer from noise and other factors that restrict or hamper the search for the optimal $\boldsymbol{\theta}$ parameters in the cost function.

The main agent hindering VQAs is the Barren Plateau (BP) phenomenon, in which the cost function gradients vanish exponentially with the number of qubits. The VQA framework uses gradient-based methods to navigate the cost landscape. As gradients become increasingly small, the cost function becomes untrainable.

There are different causes for the BP phenomenon, and they have been intensely investigated over the last few years [29; 30]. However, more recent results have shown that the proposed methods to escape BPs actually yield classically simulable circuits [7].

That being said, the usefulness of VQAs in near-term devices is still an open question, for there are still some special cases where these algorithms might be useful [7; 31], especially for problem-inspired Ansätze that are not limited by the current hardware restrictions.

4.2.4

Examples of VQAs

Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE), was designed to find the eigenvalue λ of a Hamiltonian for a given eigenstate $|\psi\rangle$, as follows.

$$H|\psi\rangle = \lambda|\psi\rangle \tag{4-6}$$

When it was first unveiled, Peruzzo *et al* [32] presented an application for the VQE on molecule ground state estimation. That being said, the cost function for this VQA derives from Equation 4-6, where, by adding the $\boldsymbol{\theta}$ parameters, yields Equation 4-7. Iterating on the values for $\boldsymbol{\theta}$, one can find the minimum eigenvalue λ_{min} for the Hamiltonian, i.e., the ground state energy. This process is depicted in Figure 4.4.

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle \tag{4-7}$$

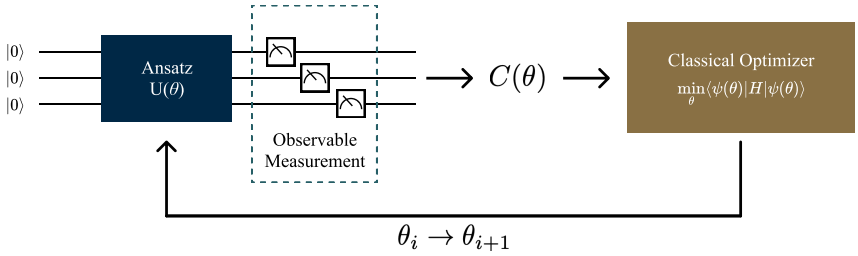

Figure 4.4: Variational Quantum Eigensolver (VQE) diagram

Considering the current state of quantum computers, the challenges of embedding the molecule Hamiltonian in the loss function must be considered. First, the number of electrons and nuclei increases the complexity of the Hamiltonian. In order to circumvent this issue, some approximations are necessary, such as the Born-Oppenheimer (BOA) approximation [33], where the nuclei are treated as stationary particles.

The resulting Hamiltonian from the BOA approximation is configured in the Fock space. Therefore, considering that quantum computer operations usually lie within the Hilbert space, one still needs to encode the fermionic system into qubits. One of the most used mappings is the Jordan-Wigner Transformation [34], where the fermionic Hamiltonian is rewritten as a sum of Pauli tensors (e.g. $XXYZ$ for a four-qubit system).

Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) [35] is another variant of VQA which uses concepts from AQC (see section 3.7). Analogous to AQC, the QAOA algorithm also uses an initial Hamiltonian, that is known as the mixing Hamiltonian ($\mathcal{H}_M$) and a final Hamiltonian, labeled as the cost Hamiltonian ($\mathcal{H}_C$).

Considering that VQAs are applied to gate-based quantum computers, these Hamiltonians are actually encoded into the mixing and cost operators $U_M(\beta)$ and $U_C(\gamma)$, presented below.

$$\begin{aligned} U_C(\gamma) &= e^{-i\gamma\mathcal{H}_C} \\ U_M(\beta) &= e^{-i\beta\mathcal{H}_M} \end{aligned} \tag{4-8}$$

From the time-varying Hamiltonian presented in Equation 3-33 from the AQC introduction, the time variable t is substituted by a factor p, which represents how many layers of $U_C(\gamma)$ and $U_M(\beta)$ operators the quantum circuit will have.

Thus, for a starting state $|\psi\rangle$, the following operator is applied.

$$U(\boldsymbol{\gamma}, \boldsymbol{\beta}) = e^{-i\beta_1\mathcal{H}_M} e^{-i\gamma_1\mathcal{H}_C} e^{-i\beta_2\mathcal{H}_M} e^{-i\gamma_2\mathcal{H}_C} \cdots e^{-i\beta_p\mathcal{H}_M} e^{-i\gamma_p\mathcal{H}_C} \tag{4-9}$$

Then, for each iteration of the QAOA algorithm, the quantum circuit is measured and a classical optimizer updates the values for the $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ parameters. This process is illustrated in Figure 4.5.

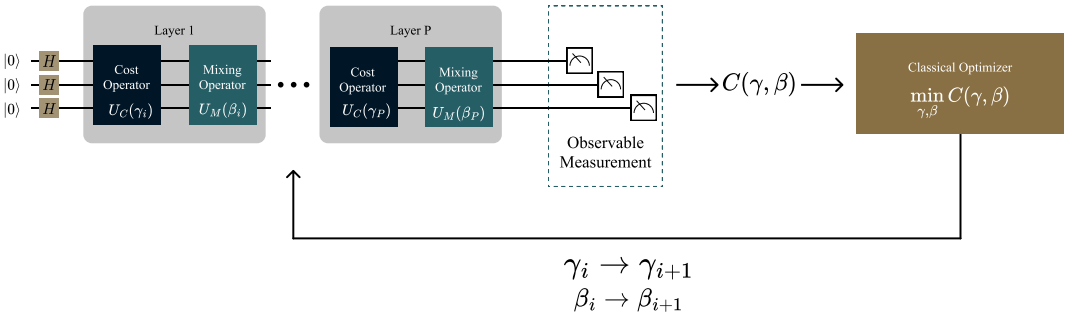

Figure 4.5: Quantum Alternating Operator Ansatz (QAOA) diagram

It is worth mentioning that the QAOA acronym is also used for an adaptation for the Quantum Approximate Optimization Algorithm, called the Quantum Alternating Operator Ansatz [36]. This variant uses different types of mixing operators that can improve the optimization process.

4.3

Quantum Annealing

Quantum Annealing (QA) is a heuristic quantum algorithm that has a lot of similarities with Adiabatic Quantum Computing (AQC), from Chapter 3. It was designed to sample low-energy solutions for the quantum version of the Ising Model, also known as the Ising Spin Glass or Transverse Field Ising Model. Before going over its main characteristics, it is essential to introduce a classical algorithm known as Simulated Annealing, from which we can find parallels to QA.

4.3.1

Simulated Annealing

Kirkpatrick *et al* [37] introduced a classical probabilistic algorithm to find the global optimal value of a given function known as Simulated Annealing (SA). In contrast with standard methods such as gradient descent that may stop at a local optima, SA proposes an acceptance probability of moving towards a worse solution that might lead to a global optima at the end of the search, as illustrated in Figure 4.6.

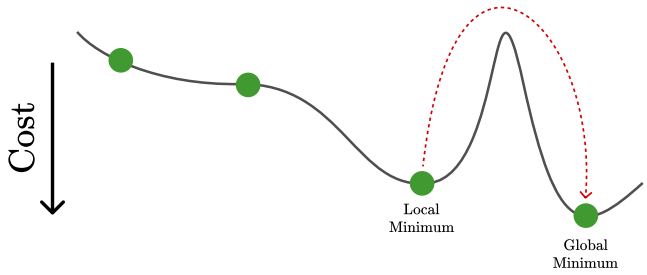

Figure 4.6: Example of a Simulated Annealing search, where the global optimal value is found by going over a 'worse solution' than a local minima

The acceptance probability of this algorithm is dictated by the cost difference between the current value and the new candidate (Δ_{cost}) and the current "temperature" T, as presented in Equation 4-10. The temperature is decremented over each iteration according to a cooling schedule, implementing in-silico the

annealing process in metallurgy[1], a resemblance that was used to name this algorithm. A pseudo-code for this procedure is outlined in Algorithm 5.

$$P_{accept}(T, \Delta_{cost}) = \min(1, \exp(^{-\Delta}/_{T}) \tag{4-10}$$

Algorithm 5 Simulated Annealing (SA) pseudocode

```
function SimulatedAnnealing(x = initialState, cost = f(x))
    i = 1
    while true do
        T = coolingSchedule(i++)
        xNew = nextCandidate(x)
        costNew = f(xNew)
        if costNew < cost then
            x = xNew; cost = costNew
        else if pAccept(T, costNew - cost) then
            x = xNew; cost = costNew
        else
            break
        end if
    end while
```

4.3.2

Quantum Annealing

QA was developed around solving optimization problems in the form of an Ising Spin Glass, presented in Equation , which is the quantum counterpart for the previously presented Ising Model. According to Catherine McGeoch [28], this framework was proposed independently by different research groups, such as in [38; 39; 40; 41].

This algorithm is very similar to the SA procedure, as one can also say that it has an "acceptance probability" when reaching a possible local minimum. However, instead of being parameterized by the current temperature, QA has the transverse field coefficient Γ, also known as the tunneling coefficient [28].

Looking back at the time-varying Hamiltonian for AQC in Equation 3-33, QA has a very similar equation, as presented below. That being said, at first, $\Gamma(t)$

[1] Annealing (Wikipedia)

is initialized with a high value that is gradually decreased as time (t) passes. At $t = 0$, the Hamiltonian takes the form of the *disordering Hamiltonian* $\mathcal{H}_D$, that ensures that the initial superposition of states is equiprobable. Thus, instead of "jumping over" hills with higher cost, as seen in SA, in QA we escape local minima by "tunneling through" these landscapes, as illustrated in Figure 4.7.

$$\mathcal{H}(t) = \mathcal{H}_f + \Gamma(t)\mathcal{H}_D \tag{4-11}$$

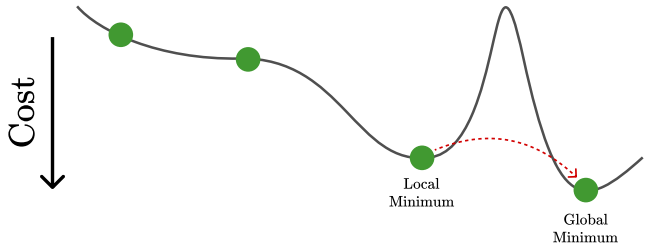

Figure 4.7: Example of a Quantum Annealing, where the global optimal value is found by "tunneling through" a hill with a higher cost than a local minima

Although SA works for more general optimization tasks, both algorithms work well for sampling low-energy solutions for the Ising and QUBO models.

5

QARBoM.jl

In the introductory chapters of this work we have presented the concept of RBMs and how one could use QC to aid the training of these neural networks using quantum sampling algorithms. Considering the possible variants of an RBM, the different training methods and the available quantum sampling techniques, we have designed a framework to benchmark the classical and quantum-aided training procedures.

Although being an agnostic platform, with respect to the training procedure, this software was named `QARBoM.jl` (Quantum-Assisted Restricted Boltzmann Machine) [12]. In this chapter we will present said framework, explaining its submodules and dependencies.

5.1

Introduction

`QARBoM.jl` is an open-source Julia package that has the main purpose of providing a smooth benchmarking experience between different RBM training methods, both classical and quantum-assisted. The choice for the programming language relies mainly on the existence of the `QUBO.jl` package [42], which is a software stack dedicated to reformulate general optimization problems into the QUBO or Ising models and interface with different quantum sampling devices and algorithms.

Our framework offers three modes of training: Contrastive Divergence, Persistent Contrastive Divergence and Quantum Sampling. That being said, the main advantage that `QARBoM.jl` offers is the ability of easily interchanging classical and quantum-assisted training.

In Chapter 2, we have seen that quantum sampling can be used to draw a low energy sample from the RBM's model which is usually estimated using Gibbs Sampling. Thus, as `QARBoM.jl` strives for maintaining the comparison between training methods as fair as possible, Figure 5.1 illustrates an overview of how our package works.

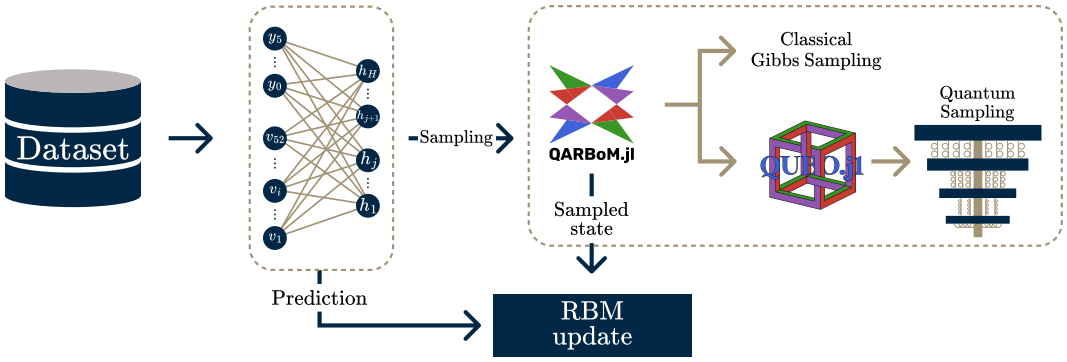

Figure 5.1: Simple diagram of `QARBoM.jl`'s purpose: providing a fair and simple way of benchmarking quantum-assisted and classical training of RBMs

5.2

`QUBO.jl` overview

`QUBO.jl` [42] is an ecosystem for working with Quantum Optimization, where one can send problems in the QUBO format to different quantum computing platforms and other quantum inspired methods. It works as an interface between the `JuMP.jl` mathematical programming package [43] and quantum sampling techniques.

Choosing `QUBO.jl` as the basis for `QARBoM.jl`'s Quantum Sampling training was pivotal for two main reasons. First, `QUBO.jl` provides connection with different quantum devices, such as IBM and D-Wave's, and quantum-inspired classical algorithms. Secondly, this package contain an automatic reformulator that converts general optimization problems into the QUBO or Ising Formats. This includes problems that contain continuous variables, which facilitates the use of Gaussian-Bernoulli RBMs.

In Chapter 6, only `QUBO.jl`'s extension for D-Wave will be used for the experiments, a choice that will be justified in that chapter. However, this ecosystem provides an interface for other platforms and quantum-inspired classical algorithms, such as `QiskitOpt.jl` [44], which connects with IBM's devices and `PySA.jl` [45], that is an interface for a Simulated Annealing algorithm developed by NASA. More information about `QUBO.jl`'s interface coverage can be found in its paper [42].

5.3

QARBoM.jl interface basics

Being developed in the Julia language, `QARBoM.jl` is able to provide a simplified interface for different training techniques and RBM types.

In Chapter 2 we have introduced two RBM variants: the Gaussian-Bernoulli RBM (GRBM) and the RBM for classification. With that said, `QARBoM.jl` contains four types of RBMs: `RBM`, `GRBM`, `RBMClassifier` and `GRBMClassifier`. Using these different structures allows us to treat each RBM type with a specific function dispatch. Additionally, for the cases where there is no necessary distinction between RBM types, we use an abstract type `AbstractRBM`.

5.3.1

Training

With an instantiated RBM, one can train the neural network with CD, PCD or Quantum Sampling. The interface that `QARBoM.jl` presents for each method is quite similar as portrayed in Table 5.1, where the main differences are actually expected according to the training procedure:

- No batch size for CD training (see Algorithm 2)
- No Gibbs Sampling steps for Q. Sampling, as it does not use this sampling technique
- No Quantum Solver attributes for CD and PCD, as they do not use any quantum sampling methods

Attribute	CD	PCD	Q. Sampling
Epochs	✓	✓	✓
Batch Size		✓	✓
Gibbs Sampling Steps	✓	✓	
Learning Rate	✓	✓	✓
Test Dataset	✓	✓	✓
Metrics	✓	✓	✓
Early Stopping	✓	✓	✓
Saves best iteration	✓	✓	✓
Saves results	✓	✓	✓
Q. Solver Attributes			✓

Table 5.1: `QARBoM.jl`'s attributes according to each training method

That being said, Listings 5.1, 5.2 and 5.3 present a simple example of usage for the `QARBoM.jl` package for an arbitrary dataset. After instantiating the

training and test samples, the user can define and RBM that will be then trained with `QARBoM.train!()`.

It is worth mentioning that for the case of Quantum Sampling, we need to define a QUBO/Ising solver to receive the RBM's energy function. This solver needs to be one of the packages in the `QUBO.jl` environment and for Listing 5.3, we have used `DWave.jl` [46] that has a simulated quantum annealing method. Moreover, as these quantum samplers have their own customizable parameters for execution, `QARBoM.jl` allows its users to define a "setup function" where these attributes are set. This tailoring step requires a more advanced knowledge of the `JuMP` and `QUBO.jl` packages, but can be disregarded as it would use default configurations.

Listing 5.1: `QARBoM.jl` code for training an RBM with Contrastive Divergence

```
using QARBoM

train_data = TRAIN_DATA
test_data = TEST_DATA

N_EPOCHS = 100
rbm = RBM(10,5)

QARBoM.train!(
    rbm,
    x_train,
    CD;
    n_epochs = N_EPOCHS,
    gibbs_steps = 5,
    learning_rate = [0.001/(j^0.6) for j in 1:N_EPOCHS],
    x_test_dataset = x_test
)
```

Listing 5.2: `QARBoM.jl` code for training an RBM with Persistent Contrastive Divergence

```
using QARBoM

train_data = TRAIN_DATA
test_data = TEST_DATA

N_EPOCHS = 100
BATCH_SIZE = 10
rbm = RBM(10,5)

QARBoM.train!(
    rbm,
    x_train,
```

```
    PCD;
    n_epochs = N_EPOCHS,
    batch_size = BATCH_SIZE,
    learning_rate = [0.001/(j^0.6) for j in 1:N_EPOCHS],
    x_test_dataset = x_test
)
```

Listing 5.3: `QARBoM.jl` code for training an RBM using Quantum Sampling via classical Simulated Annealing by D-Wave

```
using QARBoM, DWave

train_data = TRAIN_DATA
test_data = TEST_DATA

MOI = QARBoM.ToQUBO.MOI
MOI.supports(::DWave.Neal.Optimizer, ::MOI.ObjectiveSense) = true

setup = function setup_dwave(model, sampler)
    MOI.set(model, MOI.RawOptimizerAttribute("num_reads"), num_reads)
    MOI.set(model, MOI.RawOptimizerAttribute("num_sweeps"), num_sweeps)
end

N_EPOCHS = 100
BATCH_SIZE = 10
rbm = RBM(10,5)

QARBoM.train!(
    rbm,
    x_train,
    QSampling;
    n_epochs = N_EPOCHS,
    batch_size = BATCH_SIZE,
    learning_rate = [0.001/(j^0.6) for j in 1:N_EPOCHS],
    x_test_dataset = x_test,
    model_setup=setup,
    sampler=DWave.Neal.Optimizer
)
```

5.3.2

Additional features

Besides the standard training interface, there are some optional configurations that can be set by the user. This section will give a brief overview of these features.

As `QARBoM.jl` was envisioned as a benchmarking framework, it is important for all RBMs to start with the same weights to keep a fair comparison. For that, the user can pass the weight matrix when creating a new RBM, as presented

in Code 5.4.

Listing 5.4: `QARBoM.jl` code for setting a starting weight matrix **W**

```
using QARBoM

W = randn(N_VISIBLE, N_HIDDEN)

rbm = RBM(N_VISIBLE, N_HIDDEN, W)
```

Additionally, considering the different metrics that one might need to evaluate during a benchmarking procedure, `QARBoM.jl` contributors can add their own metrics to the package that, currently covers Mean Squared Error for generative training and Accuracy for discriminative training.

At the end of execution, `QARBoM.jl` stores the outcomes for the evaluated metrics in each epoch in a CSV file, that can be renamed with the parameter `file_path` in the `QARBoM.train!()` function.

5.4

Quantum Sampling for GRBMs

Although we have presented GRBMs as a possible RBM variant in Chapter 2, using quantum-assisted training with this kind of neural network poses a challenge, as we need to handle non-binary data. This introduces another layer of complexity for mapping the energy function from a GRBM into an Ising Model, as one would need to perform binary expansion on the visible variables - see Section 4.1 and Appendix D. Such challenge was mentioned in Ajagekar and You [1], where they had to use a Deep Belief Network (DBN) with a GRBM and a regular RBM layer. The former would be trained classically, while the latter would receive the binary values from the first layer and then be trained with quantum sampling.

That being said, `QARBoM.jl` is able to circumvent the need for a DBN by outsourcing the task of "binary reformulation" to `QUBO.jl`, that encodes the variables as binary automatically. The only drawback is that the user is required to inform the maximum and minimum values for each visible variables, as two vectors, to the `QARBoM.train!()` function, as presented in Code 5.5.

Listing 5.5: `QARBoM.jl` code for training a Gaussian-Bernoulli RBM (GRBM) using Quantum Sampling via classical Simulated Annealing by D-Wave

```
using QARBoM, DWave

train_data = TRAIN_DATA
test_data = TEST_DATA

max_visible = MAX_VALUES_FOR_VISIBLE_VARIABLES
min_visible = MIN_VALUES_FOR_VISIBLE_VARIABLES

MOI = QARBoM.ToQUBO.MOI
MOI.supports(::DWave.Neal.Optimizer, ::MOI.ObjectiveSense) = true

setup = function setup_dwave(model, sampler)
    MOI.set(model, MOI.RawOptimizerAttribute("num_reads"), num_reads)
    MOI.set(model, MOI.RawOptimizerAttribute("num_sweeps"), num_sweeps)
end

N_EPOCHS = 100
BATCH_SIZE = 10
rbm = GRBM(10,5)

QARBoM.train!(
    rbm,
    x_train,
    QSampling;
    n_epochs = N_EPOCHS,
    batch_size = BATCH_SIZE,
    learning_rate = [0.001/(j^0.6) for j in 1:N_EPOCHS],
    x_test_dataset = x_test,
    model_setup=setup,
    sampler=DWave.Neal.Optimizer,
    max_visible = max_visible,
    min_visible = min_visible,
)
```

For the case of PCD and CD training, the user only needs to instantiate a `GRBM` and `QARBoM.jl` will use the new condional probabilities defined in Equation 2-18.

5.5

Discriminative training

`QARBoM.jl` users can also perform discriminative training, following the RBM classifier variant defined by Larochelle and Bengio [20]. First, it is necessary to instantiate an `RBMClassifier` or a `GRBMClassifier`, informing the number of visible, hidden and label nodes respectively. Then, the user can call `QARBoM.train!()` as usual, but with the possibility of adding a specific learing rate schedule for the labels. These specifications are presented in Code 5.6. Additionally, it is worth mentioning that our framework was developed to handle binary labels in the one-hot format [1] and we plan to expand for other

[1] One-hot encoding: Wikipedia

formulations in a future work.

Listing 5.6: `QARBoM.jl` code for Discriminative Training with Persistent Contrastive Divergence

```
using QARBoM

train_data, label_train_data = TRAIN_DATA
test_data, label_test_data = TEST_DATA

N_EPOCHS = 100
BATCH_SIZE = 10
rbm = RBMClassifier(10,5,3)

QARBoM.train!(
    rbm,
    x_train,
    label_train_data,
    PCD;
    n_epochs = N_EPOCHS,
    batch_size = BATCH_SIZE,
    learning_rate = [0.001/(j^0.6) for j in 1:N_EPOCHS],
    label_learning_rate = [0.001/(j^0.6) for j in 1:N_EPOCHS],
    x_test_dataset = x_test,
    y_test_dataset = label_test_data
)
```

5.6

DBN training

The development of DBN training in `QARBoM.jl` is still being planned. However, making use of the multiple-dispatch paradigm from the Julia language, it is envisioned for `QARBoM.jl` users to be able to instantiate different types of RBM layers (with continuous or binary variables), each trained with a chosen technique (classical or quantum assisted).

6

Experiments

For the experimantal part of this work, we will showcase some of `QARBoM.jl`'s features that were presented in the previous chapter and how it can expedite the benchmarking of RBM training between quantum-assisted and classical methods. With that said, we have chosen a well known open-source dataset from the field of Process Systems Engineering, which has been extensively used for fault detection and diagnosis benchmarks.

This dataset is called Tennessee Eastman Process (TEP) [47] and it is comprised of artificial data from a simulation of a chemical plant. It contains 53 columns that indicate properties of the chemical plant and has 21 operating modes (classes): 1 normal and 20 different types of faults that can happen in this industrial plant. More details on each of these operating modes can be found in [48], while the dataset used for this work is hosted in the Harvard Dataverse repository [49].

The complexity of the TEP dataset has been reported in several papers over the past two decades [1; 48; 50; 51; 52; 53], where different techniques were employed in order to improve the fault detection framework being studied.

Zhang and Zhao [51], for instance, proposed the use of the One-Class-One-Network (OCON) technique, where they segmented the training for each class into a specific neural network which, for this case, was a DBN.

Moreover, Xie and Bai [50] used a hierarchical deep neural network (DNN) that had a similar purpose as the two previous works that were mentioned. This hierarchical DNN was comprised of 6 DNN, one being a supervisory agent that was able to read a data sample and address it to another DNN that was specialized for a specific group of classes. These networks worked on four groups, disclosed below, where the faults 3, 9 and 15 were not considered for being too hard to diagnose. The clustering procedure worked by assembling faults that, when classified by the same DNN, the misclassification rate did not decrease by 10%. It is worth mentioning that the normal class was also disregarded, but with no further explanation.

- Group 1: Fault 2
- Group 2: Faults 12 and 17
- Group 3: Faults 1, 6, 8, 13, 18
- Group 4: Faults 4, 5, 7, 10, 11, 14, 16, 19, 20

Besides Xie and Bai [50], other works mentioned that the faults 3, 9 and 15 had a low classification rate [1; 48; 51], with Zhang and Zhao [51] stating that faults 5 and 16 were also hard and were often misclassified as a normal state. This find can actually be noticed in the confusion matrices at the end of the first experimental section, where faults 3, 5 and the normal state were the classes with the highest errors (see Figures 6.12, 6.13 and 6.14).

This dataset has also already been tested with quantum-assisted training of an RBM in a work presented by Ajagekar and You [1] where, similar to Zhang and Zhao [51], they used a specialized network for each fault class. However, their learning framework, presented in Figure 6.1, was mainly consisted of other classicaly-trained neural networks layers besides two quantum-assisted RBMs. In sum, they have used:

1. Two DBNs, containing a GRBM in the first layer and a regular (Bernoulli) RBM in the second layer. The first DBN (DBN-N) was trained using normal data and the second (DBN-F) was modeled after only one of all possible faults in the dataset. Both DBNs were trained generatively and only the second layer of each DBN, the that was a Bernoulli RBM, was trained with the help of a quantum algorithm.

2. Two neural network layers that were not DBNs or RBMs, and that were classically trained via supervised discriminative learning.

Thus, at the end of the framework presented in Figure 6.1, the output was either *Normal* or *Fault X*, where X was the fault used during training. Additionally, this framework had to be repeated for each of the 20 faults in the dataset. This greatly simplified the modeling process, as they had a different DBN being trained for each fault mode, avoiding any correlation issues between fault classes that are well known in this dataset.

Moreover, still in their paper on industrial processes [1], they have only compared quantum-assisted training against the Contrastive Divergence (CD) algorithm, leaving out an important and more modern classical variant: the Persistent Contrastive Divergence (PCD). Having only CD to compare, Quantum Advantage was claimed for quantum-assisted RBMs, although the majority of the training of their full neural network (Figure 6.1) was classical.

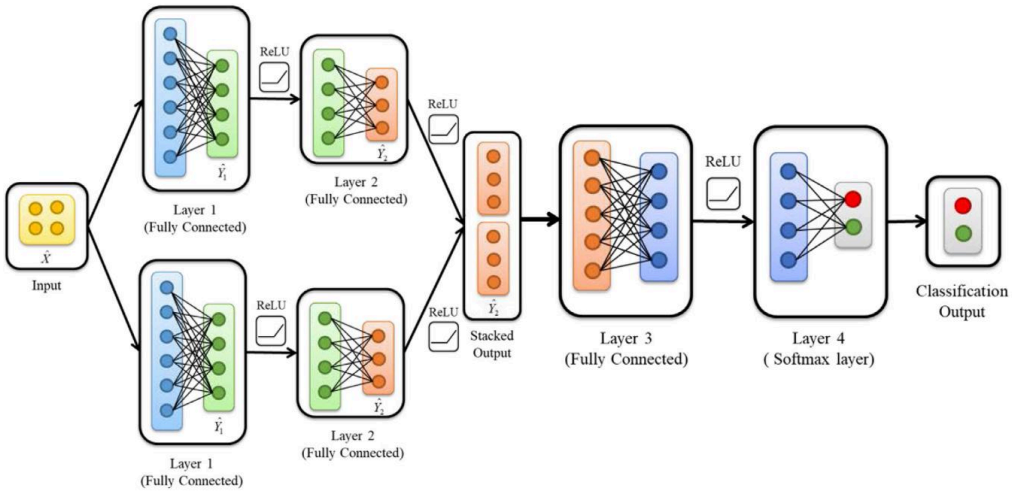

Figure 6.1: Training framework presented in [1] with one quantum-assisted layer

The experimental sections are divided as follows. First, we select the normal operating mode and the first 5 fault classes and try to train a GRBM with a classification layer, comparing the results for CD, PCD and Simulated Annealing (SA), which is provided as a "classical simulator" for D-Wave's quantum annealers. Then, in the second experiment we go over some techniques that were employed in previous works to improve the results that we have presented in the first experiment. Finally, for the last analysis, we experiment on a real Quantum Annealer from D-Wave and compare its results against the previous experimental iterations. It will be possible to notice that the choice of

Before delving into the experiments, it is important to explain the reason for selecting D-Wave as the quantum platform to be studied, considering that `QARBoM.jl` could also interface with IBM's devices using `QUBO.jl`. The motive mainly lies in the long waiting lines for submitting a job to IBM's platform. This would impact negatively in the time comparison between classical and quantum algorithms, besides the fact that it could make the whole training of an RBM, that requires several quantum sampling procedures, impractical given the timeline of this project. On the other hand, with an academic credit program, D-Wave does not have a perceptible waiting queue, making it the best choice for this experimental section.

6.1

Initial experiments on the TEP dataset

In this section we perform a comprehensive analysis of several learning configurations for the TEP dataset, using the PCD, CD and SA (from the `D-Wave.jl` package). With the intent of presenting results that are easy to interpret, each training parameter group will be studied separately.

Moreover, as will be seen in a future section, access to Quantum Annealing hardware is limited, considering that it is offered as a paid cloud service, with a few trial credits. This justifies the choice for the SA alternative for this section, where several RBMs will be trained, which would be infeasible to perform in real devices.

Additionally, for this experimental section, the dataset was simplified. Instead of the 21 classes, only 6 were considered (faults 1 to 5 and the normal operating mode). This simplification allowed us to investigate the impact of each training parameter more thoroughly, but still presented some interesting aspects of the original data. Also, only a single GRBM with a classifier label was used, as illustrated in Figure 6.2.

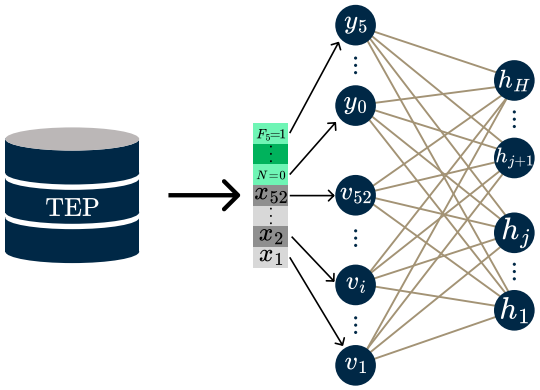

Figure 6.2: Framework for working with the TEP dataset, using 5 faults and the normal operating mode. The classes are represented using one-hot encoded, where only a single label vector value is one (for this example is the fault 5). Besides the label nodes, there are 52 visible nodes $x_k \in \mathbb{R}$.

Finally, the dataset was splitted into training, validation and test. The learning curves for the training process were obtained by estimating the accuracy of the RBM over the validation dataset. At the end of this section, the best RBMs for each training mode will be selected to classify the test dataset.

Starting parameters

First, we will consider the starting configuration for the RBM. For this test, only the weight matrices **W** and **U** were considered, while the biases **a**, **b** and **c** were always instantiated as zero vectors.

We have created 25 possible combinations of weight matrices, selecting 5 starting variants for **W** and **U**, where the first one of each matrix is a zero matrix and the remaining four are random matrices. Then, the following hyperparameters disclosed in Table 6.1 were selected, where the learning rate α_0 is actually the starting learing rate, for a learning rate schedule of $\alpha_x = {}^{\alpha_0}/_{x^{0.6}}$, where x is the epoch. Additionally, the training for all methods had an early stop condition, in case the accuracy decreases, with a patience of 10 epochs.

Parameter	**CD**	**PCD**	**SA**
Learning rate(α_0) [1]	0.001	0.001	0.001
Batch size	-	300	300
Gibbs Sampling Steps	1	1	-
Number of Sweeps	-	-	100
Patience	10	10	10

[1] A decaying learning rate $\alpha_x = {}^{\alpha_0}/_{x^{0.6}}$, where x is the epoch

Table 6.1: Fixed learning parameters for iterating over different starting weights

From the results reached for each method are presented in Figures 6.3, 6.4 and 6.5, it is possible to notice that starting from a zero matrix **U** and a random **W** yields better and more stable results, for both quantum-assisted (SA) and classical modes. Moreover, the results for SA do not display the iteration for a zero **U** and zero **W** matrices because it would yield a "QUBO" with only zero coefficients at first and this introduces an error for the `D-Wave.jl` solver.

It is worth highlighting that the profiles for the CD method displayed that the training quickly diverges, independent of the starting matrices. This behavior will be analyzed and addressed as we study other parameters.

Number of hidden units

Having investigated the impact of an RBM's starting weights, another aspect that is worth analyzing is the number of hidden nodes. The connections

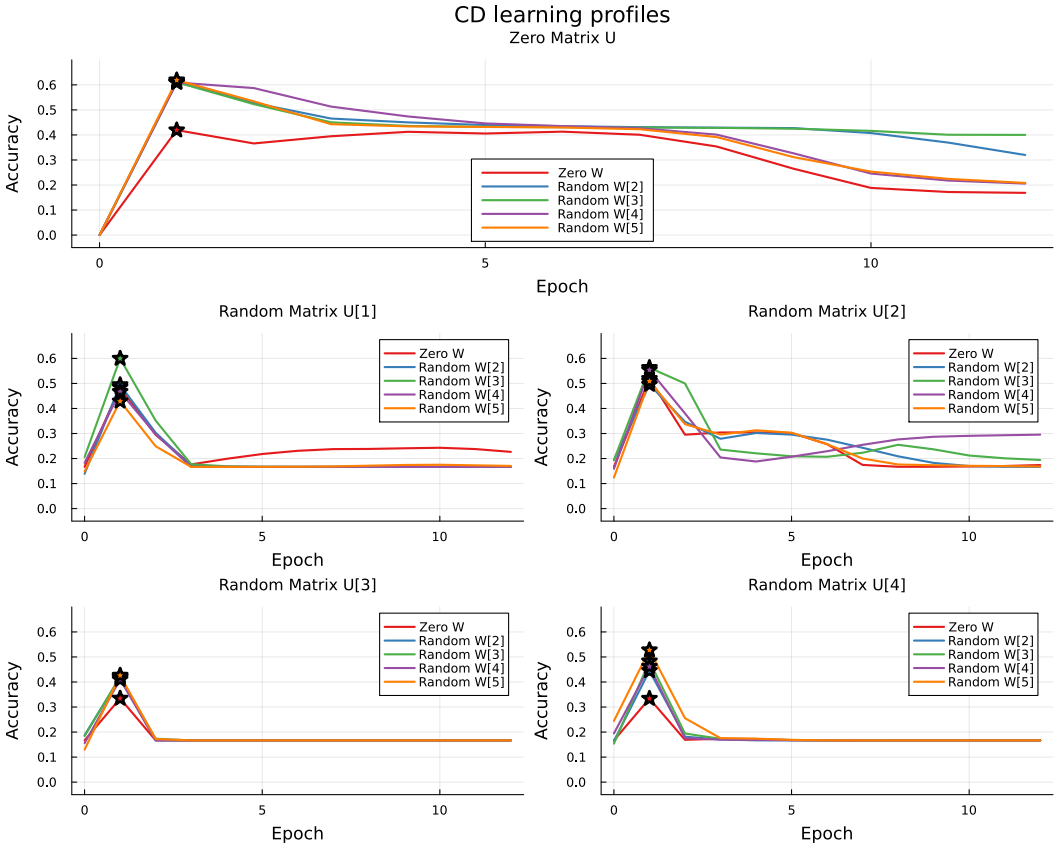

Figure 6.3: CD Learning curve for different starting matrices $\mathbf{W}$ and $\mathbf{U}$.

between the visible and hidden nodes of an RBM are responsible for modeling the latent features from a dataset. That being said, as the number of hidden nodes increases, at first it is expected that the classification accuracy follows the same trend. However, one can find themself in a scenario where the RBM model overfits, making it hard to classify out-of-sample data.

The number of hidden nodes also comes with computational costs, as the size of the weight matrices and the hidden nodes' bias vector also increase. Furthermore, under the perspective of SA, more specifically real Quantum Sampling, there is an extra toll: the number of qubits required to map the QUBO model. Considering the current state of quantum hardware where the number of qubits is a constraint that limits the size of quantum algorithms, there is a limit for the number of hidden nodes that one can choose to perform quantum-assisted training. That being said, although SA is still a classical algorithm, it simulates a quantum one and inherits the costs of a growing number of qubits.

For this experiment, we have maintained the same parameters presented in Table 6.1, with the exception of the number of hidden nodes, which we will

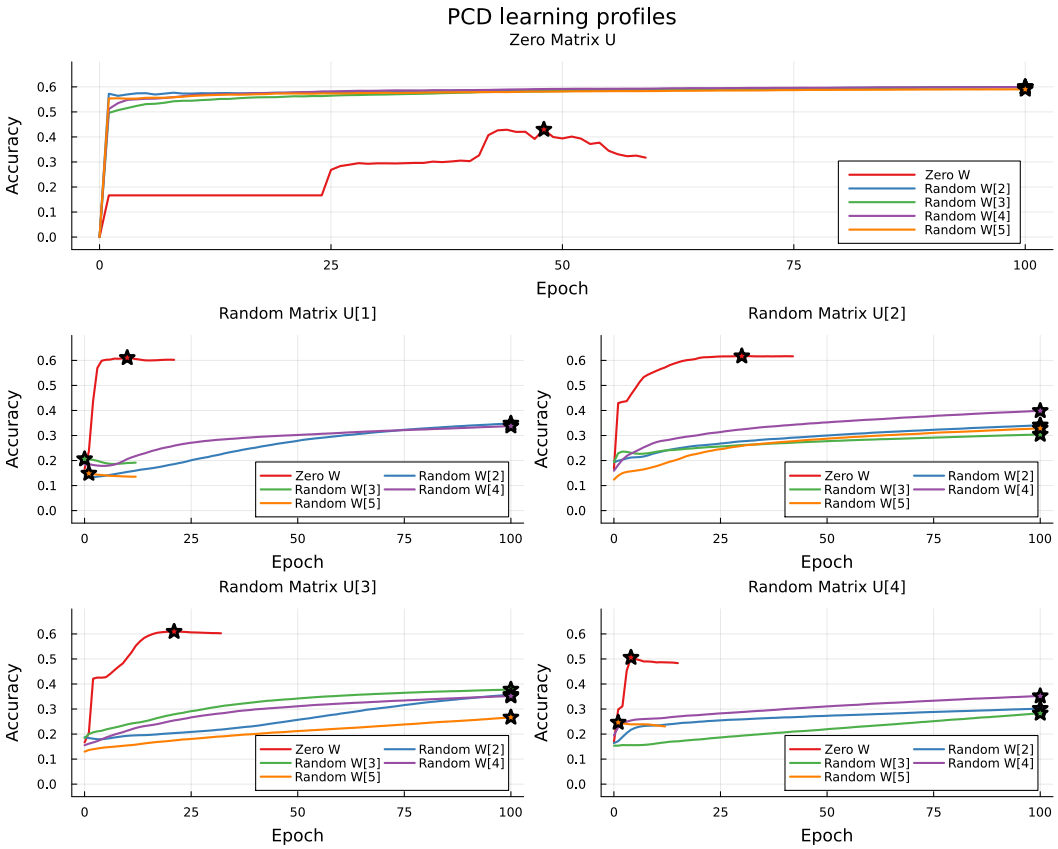

Figure 6.4: PCD Learning curve for different starting matrices **W** and **U**

vary between 50, 100 and 200. Additionally, as the size of the matrices vary, the value for the **W** matrix were selected randomly, while the **U** was initiated as a zero matrix.

The results presented in Figure 6.6 need to be analyzed in parts. First, the plot for CD shows that, again, the learning curves diverge quickly, not being able to maintain the same accuracy as the other methods. The number of Gibbs Sampling steps (1), combined with learning rate ($\alpha = 10^{-4}$) are probably the main contributors for this occurrence. A low number of Gibbs steps introduces high variance to the gradient approximation and the the large step might be giving too much weight to this gradient value.

Now, considering the outcomes for PCD, the number of Gibbs Sampling steps and the learning rate do not seem to have made a negative impact in the learning curve. The primary reason for that is the fact that the Gibbs Sampling chains persist over the epochs with the fantasy points (see Subsection 2.5.2). Moreover, using a batch size of 300 should have damped the effect of the learning step. Furthermore, the zoomed section of the plot is is very useful to highlight the influence of the number of hidden nodes for the learning profile,

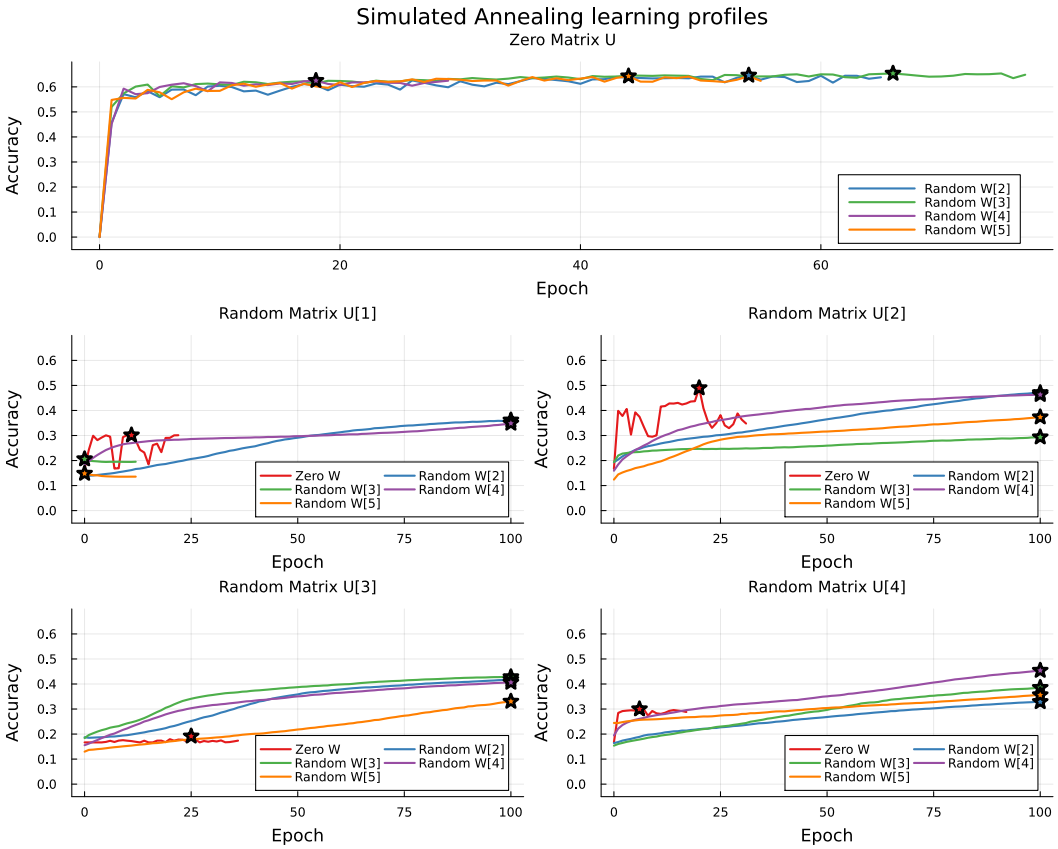

Figure 6.5: Quantum Sampling Learning curve for different starting matrices **W** and **U**

as the three curves for the 200-hidden matrices were the ones with the highest accuracy.

Finally, the learning curves for the SA method do not show a clear difference between accuracies with 100 and 200 hidden units as PCD results did. The three 200-hidden nodes variants are among the top 4 best profiles, but there is a 100-hidden nodes curve that has the third best accuracy at its best epoch. However, the 50-hidden variables lines are all below 64% accuracy, making it clear that they are the worst ones.

Number of Gibbs Sampling steps

For this subsection, the number of Gibbs Sampling steps, which is exclusive for the classical algorithms is studied. In the previous analysis we have seen that the CD technique falls short in every iteration against its classical and quantum-assisted counterparts, where one of the reasons for that were the number of Gibbs Sampling steps, which was actually 1. That being said, we

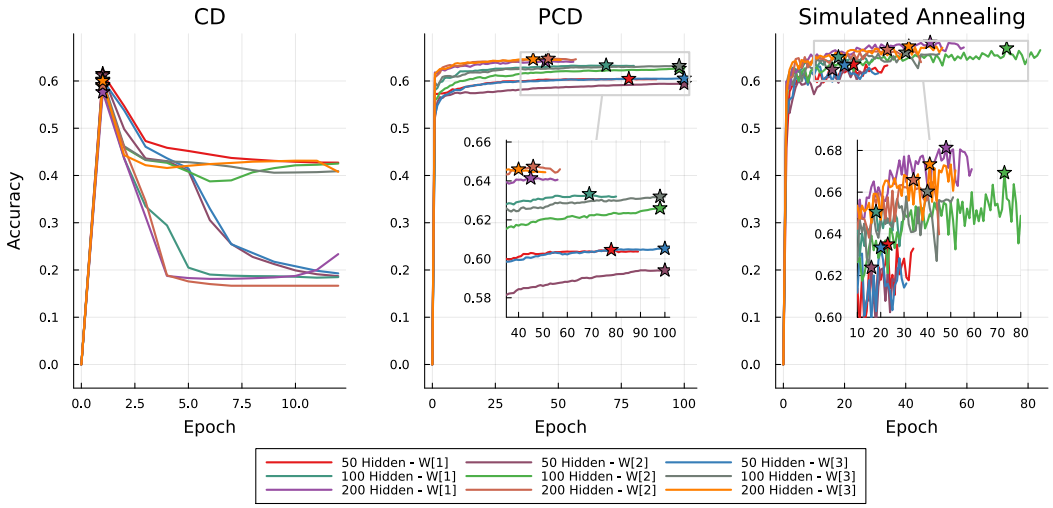

Figure 6.6: Learning curves for different numbers of hidden nodes

have analyzed the influence of this parameter for the values 1, 10 and 20, but fixing the number of hidden nodes to 200 (the best iteration in the previous experiment) and keeping the starting learning rate and batch size to 0.001 and 300 respectively.

The outcomes presented in Figure 6.7 shows with clarity that using a higher number of Gibbs Sampling steps helps with the CD's learning curves. The only lines that diverge instantly after the first epoch are the ones with 1 step, while the others persisted for a small number of epochs. It is worth highlighting that one of the iterations with 20 steps reached the highest accuracy value and was able to last for more epochs until stopping.

For the case of PCD, the difference between each iteration was very subtle, as most learning profiles had their best accuracy around 64%.

Number of sweeps

Having investigated the impact of the number of Gibbs Sampling steps, we now move to a parameter that is specific to the SA technique: the number of sweeps. This argument is related to the number of times that each spin qubit is updated in the simulation process. Varying it between 10, 50 and 100, we can see that, as shown in Figure 6.8, a higher number of sweeps is related to better accuracy results.

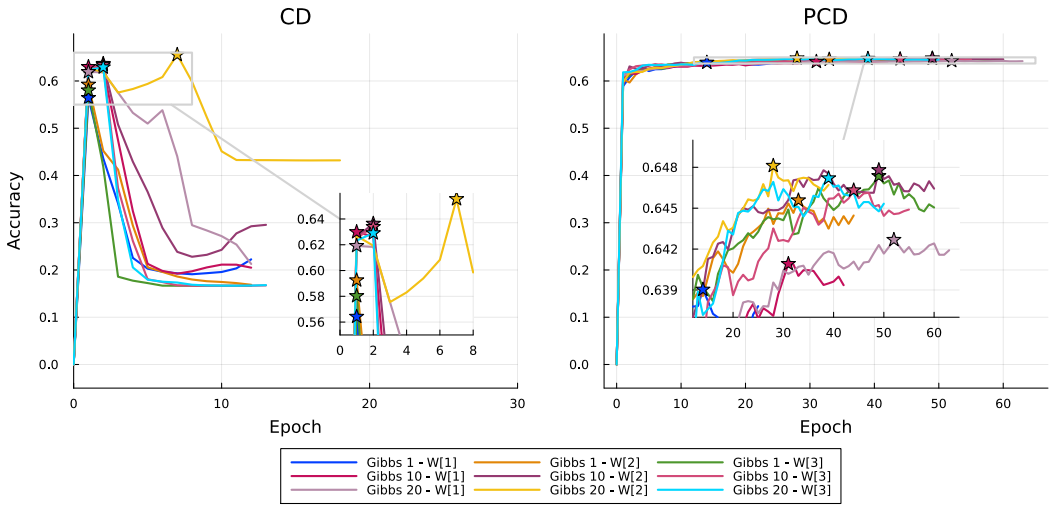

Figure 6.7: Learning curves for different numbers of Gibbs Sampling steps

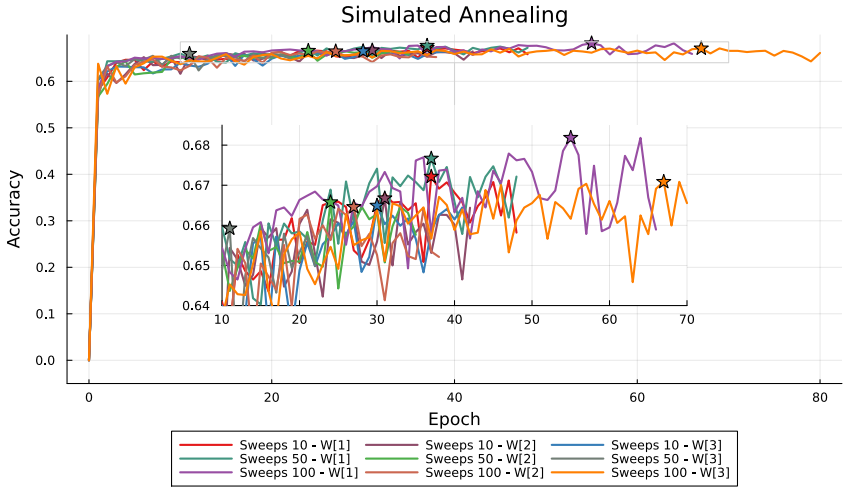

Figure 6.8: Learning curves for different numbers of sweeps

Learning rate and batch size

Finally, there are two last parameters that we need to analyze: the learning rate and batch size. Considering that for the case of PCD and SA, the learning rate and batch size are related, they will be studied together in this experimental section. Moreover, it is worth reminding that only the starting learning rate α_0 was compared, as the following learning rate function schedule was maintained.

$$\alpha_x = \frac{\alpha_0}{x^{0.6}}, \text{ where } x \text{ is the epoch} \tag{6-1}$$

Starting with the results for CD, which are presented in Figure 6.9, it is noticeable that a smaller starting learning rate yields a curve that does not diverge quickly. Until this point, α_0 was fixed at $\alpha_0 = 10^{-4}$ and, from the beginning, the learning profile has always displayed a fast decay, with a small improvement when the number of Gibbs Sampling steps was increased to 20. With that said, using $\alpha_0 = 10^{-5}$ and $\alpha_0 = 10^{-6}$ resulted in accuracies that are more competitive with the results from PCD and SA.

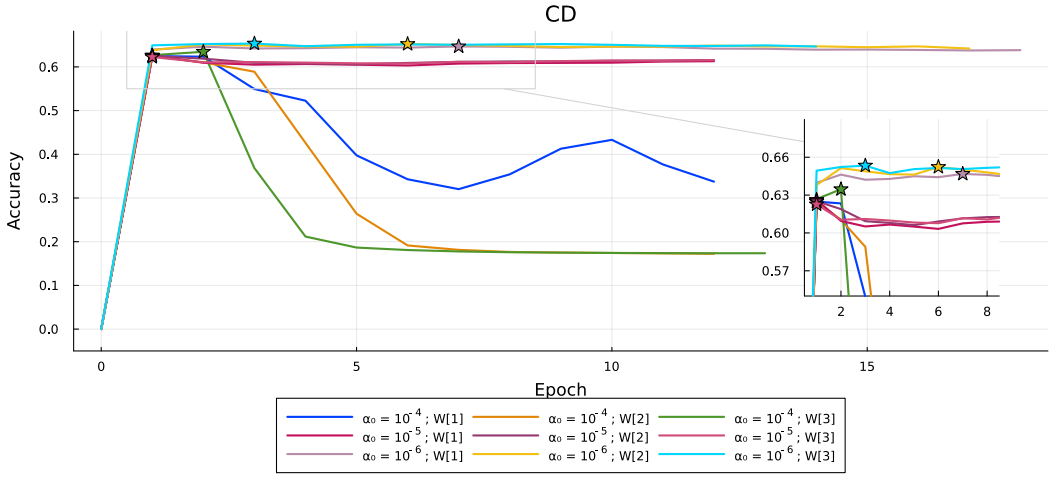

Figure 6.9: Learning curves for different starting learning rates α_0 and initial matrices $\mathbf{W}$

Now, considering the outcomes for PCD, in Figure 6.10, increasing the batch size only made the learning profiles more "stable", although not yielding better results than the top two accuracies with a batch size of 100. Both of these results used the standard starting learning rate $\alpha_0 = 10^{-4}$ and, even though they reached higher accuracies, they diverged quickly right after hitting their best epoch. It is worth mentioning that in order to simplify the plots, the initial epochs were cut out from the images, as they did not add much information to this discussion.

At last, as seen in Figure 6.11, in contrast with PCD, SA provided a better accuracy using a batch size of 300, but still maintaining $\alpha_0 = 10^{-4}$. It is worth noting that a smaller learning rate did not contribute to a more stable curve and even presented a faster decay. Moreover, just as with PCD, the initial epochs were also disregarded for simplicity.

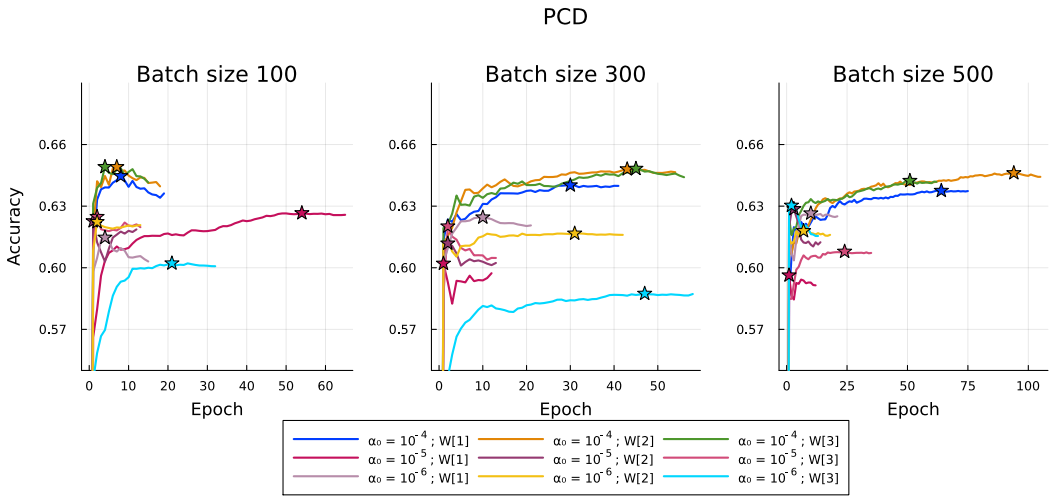

Figure 6.10: Learning curves for different starting learning rates α_0, batch sizes and initial matrices $\mathbf{W}$

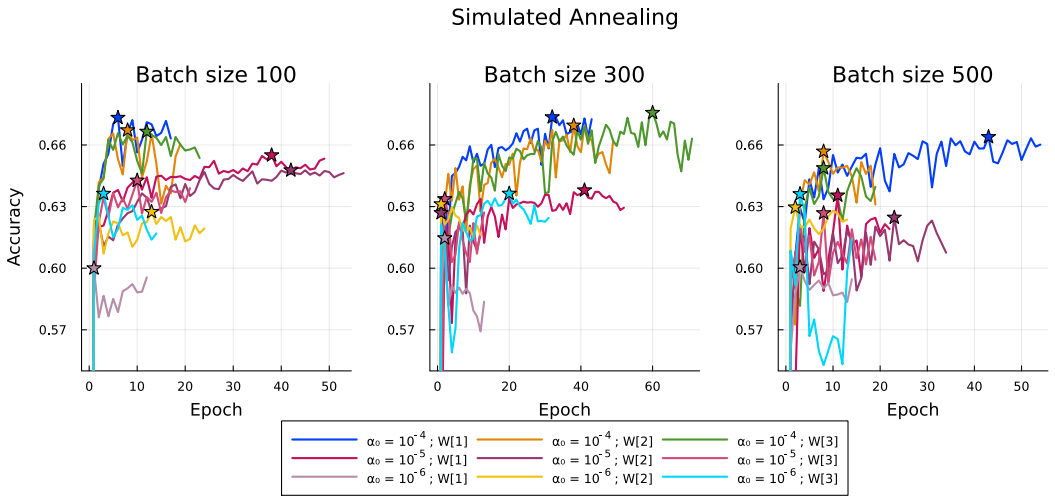

Figure 6.11: Learning curves for different starting learning rates α_0, batch sizes and initial matrices $\mathbf{W}$

Final analysis

In this section, several learning parameters were analyzed, yielding in best RBM configurations for each training mode, presented in Table 6.2.

Using these optimized RBM models, Figures 6.12, 6.13 and 6.14 present the confusion matrices for all three datasets: train, validation and test. These outcomes show that, overlooking the accuracy that is limited to below 70%, the RBM models demonstrate good generalization, as the confusion matrices for

Parameter	CD	PCD	SA
Hidden nodes	200	200	200
Learning rate α_0	10^{-6}	10^{-4}	10^{-4}
Batch size	-	100	300
GB Steps	20	20	-
Sweeps	-	-	100
Matrix	3	3	3

Table 6.2: Best training parameters for each training mode

all three datasets are quite similar. Moreover, it seems that there were some difficulties to analyze the faults 3, 5 and the faultless operating mode ('Normal'). From the preliminary discussions in the beginning of this chapter, this behavior was actually expected and is aligned with outcomes from previous works.

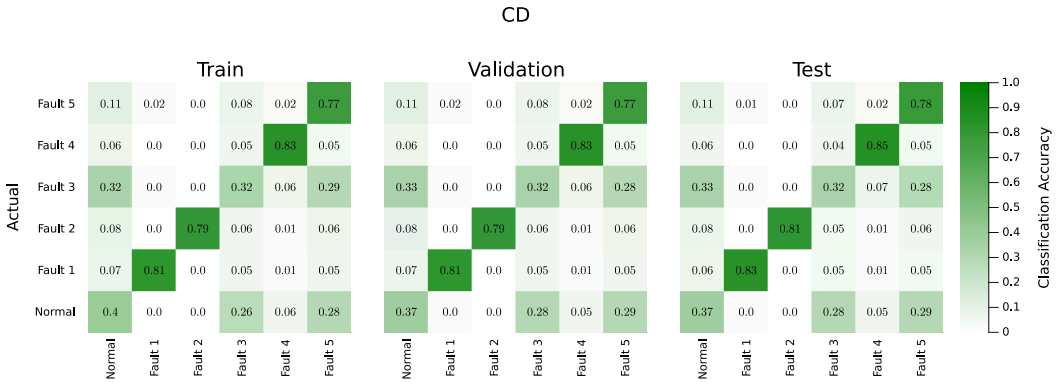

Figure 6.12: Confusion matrix for CD training over train, validation and test datasets

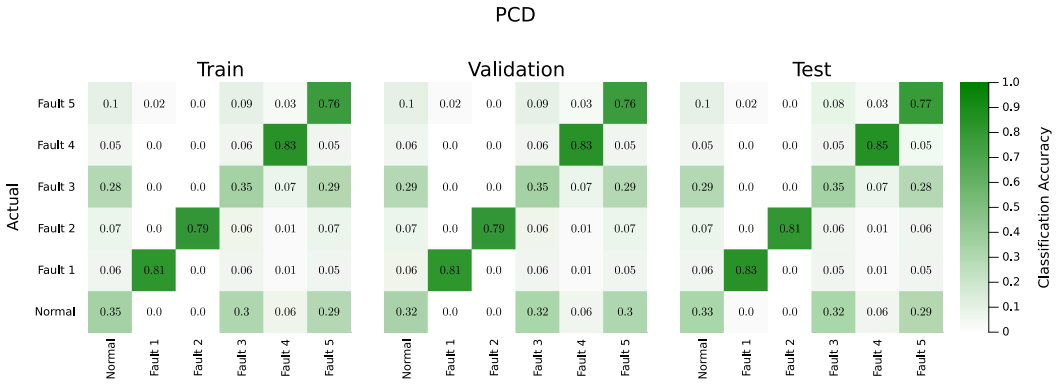

Figure 6.13: Confusion matrix for PCD training over train, validation and test datasets

Furthermore, according to Table 6.3 and the confusion matrices that were presented, SA provided better accuracies in comparison with the classical

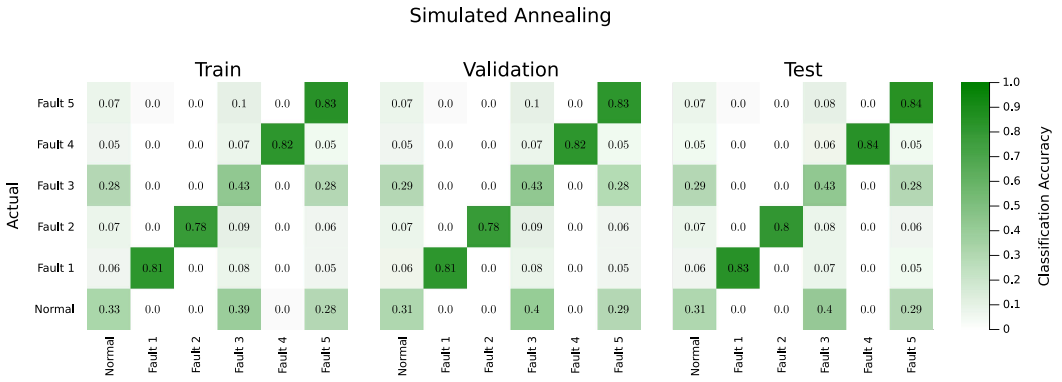

Figure 6.14: Confusion matrix for SA training over train, validation and test datasets

training methods. This gain can be perceived in the confusion matrices for faults 3 and 5.

Dataset	CD	PCD	SA
Train	65.47%	64.96%	66.71%
Validation	64.91%	64.43%	66.36%
Test	66.07%	65.62%	67.56%

Table 6.3: Accuracies for each RBM training over the training, validation and test datasets

However, even though SA reached higher accuracies, it is still hard to justify its use over the other methods. Being a classical simulation of Quantum Annealing, this method is required to mimic the problem that a real quantum annealer is supposed to tackle, which introduces a large time overhead to train the RBM at each epoch. Table 6.4 displays the total training time for each algorithm to achieve the accuracies disclosed in Table 6.3, where it is noticeable that the time that SA spends at each epoch is over 3.5 times higher than CD and PCD.

	CD	PCD	SA
Total time (s)	785.469	801.561	14695.0
Epochs	14	15	71
Avg. time ($\frac{s}{epoch}$)	56.11	53.43	206.97

Table 6.4: Time to train, number of epochs required and average time spent at an epoch for each training method (using the best learning configurations)

This extra duration is mainly related to the fact that SA needs to work with a QUBO/Ising model that was translated from the RBM's energy function. And, as we were working with a dataset that contains continuous variables, during

the reformulation process, a binary expansion technique had to be applied, which increased the actual number of variables for the target QUBO problem.

In Appendix D, we have seen that during the binary expansion from continuous variables, there is a error tolerance constant τ that impacts the number of binary variables required to represent a continuous one. Using `QUBO.jl` in a lower level, it is possible to understand the impact of this reformulation.

As no value for τ was set, `QUBO.jl` uses the default $\tau = 0.25$. This configuration yields the following mapping in Table 6.5, where the number of visible variables goes from 52 continuous to 200 binary, increasing the total number of variables and thus the number of qubits that the SA algorithm is required to represent.

Model	Visible	Label	Hidden	Total
RBM	52	6	200	258
QUBO	200	6	200	406

Table 6.5: Number of variables before and after the reformulation of an RBM's energy function into a QUBO. Notice that the number of visible variables increases because they are continuous and require binary expansion, while the label and hidden variables are already binary.

Increasing the value for τ would reduce the number of binary visible variables, in exchange for a loss in precision. Although this was not studied in this experimental segment, it will become indispensable for Section 6.3, where the number of qubits will be a hard constraint.

Summarizing, this experiment tested several training parameters for CD, PCD and SA. From the results, we have concluded that SA reaches higher accuracies, but in a considerably slower time. For the next experiments we will analyze the TEP dataset in more detail and perform quantum-assisted training on a real quantum computer.

6.2

Experiment 2: A deeper look into the TEP dataset

In the previous experiment, we have trained an RBM for only six classes in the TEP dataset (normal operation and faults 1 to 5), where we concluded that, although only considering a fraction of the dataset labels, the classification accuracy was below 70% for all learning curves. That being said, in this section, we analyze an alternative way to improve the accuracy scores while classifying all 21 classes in the dataset.

Our objective was to develop a framework similar to the one presented by Ajagekar and You [1], where they have paired 20 RBMs trained generatively to a Multilayer Perceptron (MLP) that was responsible for classifying the samples. Although they used specialized DBNs (two connected RBMs), each trained specifically for one of the 20 faults, we have used a single generative RBM that received samples for all classes. Additionally, for this experimental section, we have used Ajagekar and You's dataset version for the TEP problem, which was provided in the supplementary section of their submission. Finally, it is worth disclosing that this dataset variant does not have a distinction between validation and test data.

We start this section by comparing the training of a single RBM for all faults using the PCD and SA modes from `QARBoM.jl`. Then, considering the shortcomings that will be discussed, we propose combining an RBM to a MLP in order to improve the results.

For this study, we have considered a different approach for the choice of training parameters. First, the learning rate schedule was chosen according to a new function, described in Equation 6-2. Using the learning rate curve described by this equation, we can avoid vanishing gradients, as there is a fixed final learning rate that is set with α_f. Additionally, adhering to convention in other ML works, power of two batch sizes (256, 512 and 1024) were tested.

$$\begin{aligned}\alpha(x) =& \alpha_f + (\alpha_0 - \alpha_f) \cdot \exp(-\delta \cdot x), \text{where} \\ & \alpha_0 \text{ and } \alpha_f \text{ are the initial and final learning rates, respectively} \\ & x \text{ is the epoch} \\ & \delta \text{ is the decay rate}\end{aligned} \tag{6-2}$$

The tested parameters for this analysis are disclosed in Table 6.6, where different pairs of initial an final learning rates (α_0, α_f) were set.

From the outcomes presented in Figures 6.15 and 6.16, there is clearly a 50% accuracy barrier that could not be exceeded by any training iteration. Furthermore, it is noticeable that there is a sweet spot for the initial learning rate α_0 during PCD training (Figure 6.15), as for $\alpha_0 \in \{1^{-1}, 1^4, 1^{-5}\}$ the learning curves hit a plateau of 0.332 in accuracy, which is very low compared to the best iterations with $\alpha_0 \in \{1^{-2}, 1^{-3}\}$, that achieved accuracies above 0.40.

Parameter	Value
Hidden nodes	100
(α_0, α_f)	$(1^{-1}, 1^{-3})$;$(1^{-2}, 1^{-3})$;$(1^{-2}, 1^{-4})$;$(1^{-2}, 1^{-5})$; $(1^{-4}, 1^{-5})$;$(1^{-4}, 1^{-6})$;$(1^{-3}, 1^{-4})$;$(1^{-5}, 1^{-6})$
δ	0.1, 0.07
Batch size	256, 512, 1024
Gibbs steps	20
Reads and Sweeps	100

Table 6.6: Training parameters for the discriminative training of an RBM using the TEP dataset. Note that the gibbs steps value is exclusive to the PCD training procedure, while the reads and sweeps values are solely for the SA training.

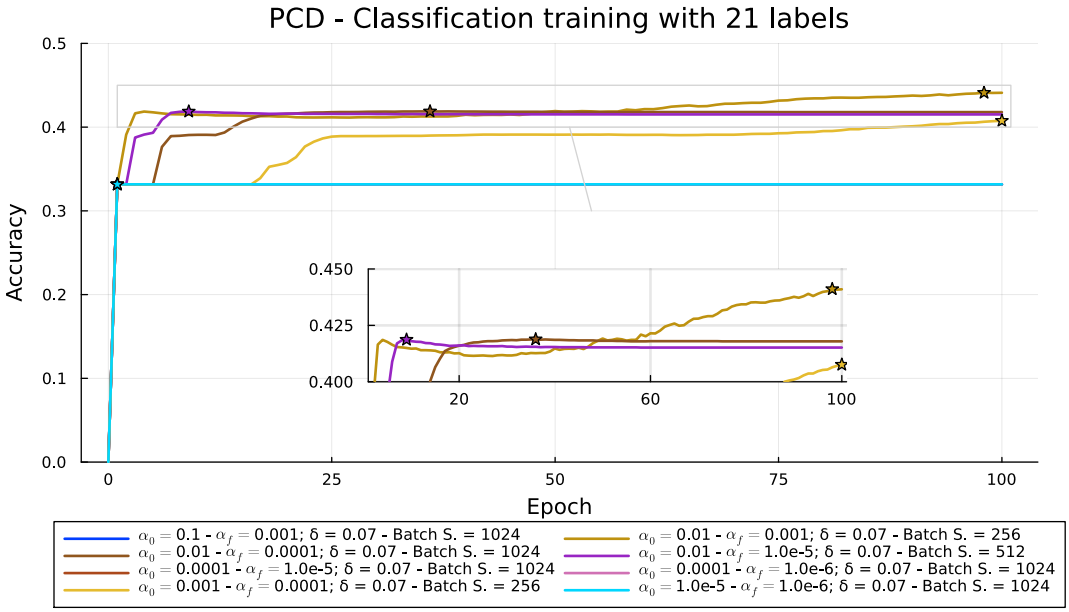

Figure 6.15: Learning curves for RBM classification training using Persistent Constrastive Divergence for 21 labels from the TEP dataset. There are only 5 visible curves in this plot because the learning curves with starting learning rates $\alpha_0 \in \{1^{-1}, 1^{-4}, 1^{-5}\}$ presented overlapping fixed learning curves after the first epoch with an accuracy of 0.332.

In contrast, the SA training presented in Figure 6.16 had all learning iterations with their best accuracy above 0.4, but with a drawback: most of them had a high standard deviation. That being said, the plot was divided in two graphs to facilitate its understanding. Again, it is evident that there is also a learning schedule sweet spot, as the higher values for α_0 $(1^{-1}, 1^{-2}, 1^{-3})$ were very noisy, while the smallest value $\alpha_0 = 1^{-5}$ hit a plateau with an accuracy below 0.425. Additionally, despite the fact that SA reached higher accuracies, besides the noisy and unpredictable curves, its highest value was 0.496.

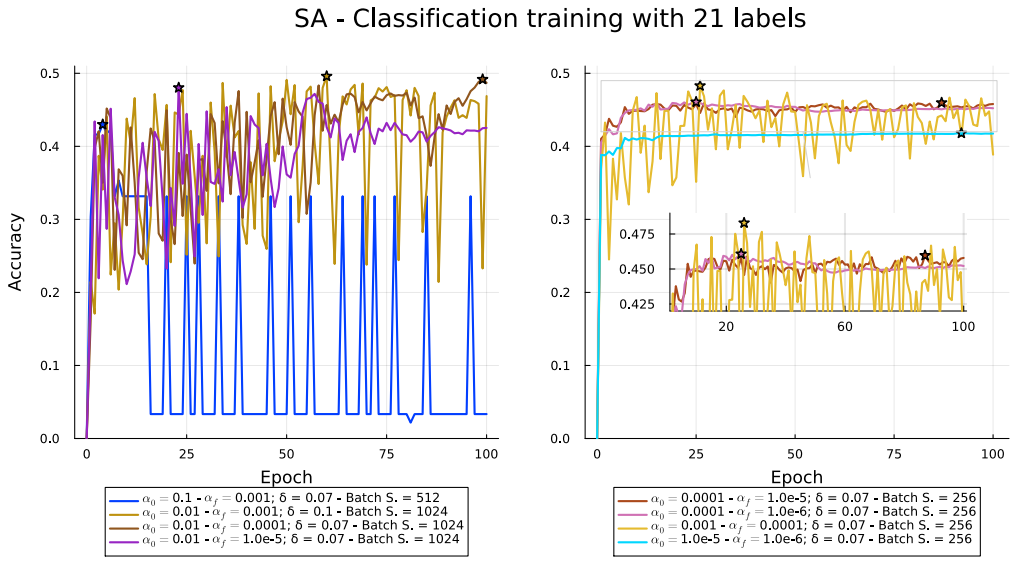

Figure 6.16: Learning curves for RBM classification training using Simulated Annealing for 21 labels from the TEP dataset. Notice that, although having a higher starting learning rate α_0, the best accuracies presented very noisy learning curves.

In conclusion, discriminative training for the 21 classes in the TEP dataset using a single RBM never surpassed the 50% accuracy threshold. At this step, we outlined three possible options to address this shortcoming:

1. Increase the number of hidden units
2. Implement a Deep Belief Network
3. Combine an RBM to another neural network

In the first experimental section, we have seen that increasing the number of hidden units did have an impact in the accuracy, but between 100 and 200 hidden units there was only a small difference (see Figure 6.6). Moreover, implementing a Deep Belief Network requires further development of `QARBoM.jl` that would not fit within the time constraints of this thesis.

Therefore, it was decided to experiment combining an RBM to a MLP, that would be responsible for receiving the hidden node values from an RBM and using them for classification. The RBM would be responsible for modeling the 52 visible features of the TEP dataset generatively, without classifying them. After being trained, the RBM would be used on the dataset again, feeding the hidden node values, along to their corresponding label from the dataset, to a

MLP. Figure 6.19, that will be discussed in more depth later, illustrates this framework.

6.2.1

Generative RBM training

For the generative training, we have trained different RBMs, using the PCD and SA procedures, with the parameters presented in Table 6.7. Additionally, as the TEP dataset values remain continuous, a GRBM was still required for this experiment.

Parameter	Value
Hidden nodes	100
α_0	0.01, 0.001
α_f	0.001, 0.0005
δ	0.1, 0.05
Batch size	256, 512, 1024
Gibbs steps	20
Reads and Sweeps	100

Table 6.7: Training parameters for the generative training of an RBM using the TEP dataset. Note that the gibbs steps value is exclusive to the PCD training procedure, while the reads and sweeps values are solely for the SA training.

The learning progress of these RBMs was evaluated using the Mean Squared Error metric, comparing the test samples with their reconstruction by the RBM (see Figure 6.17 for an illustration on reconstruction).

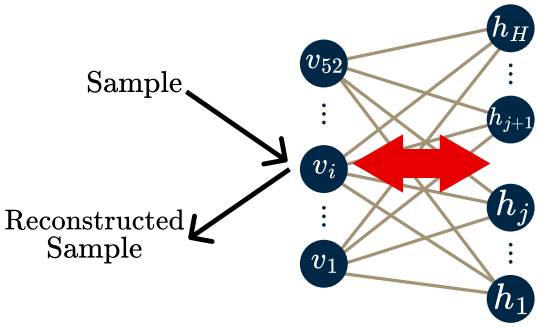

Figure 6.17: RBM reconstruction illustration. A sample is fed to the RBM, which then samples hidden nodes given the input values. After estimating values for the hidden nodes, the RBM samples the values for the visible nodes, given the sampled hidden ones.

After iterating over the parameters described in Table 6.7, the minimum values found for PCD and SA training were 57.82 and 102.03, respectively, using the arguments displayed in Table 6.8. Moreover, the learning curves for these RBM variants are presented in Figure 6.18, where it is possible to notice that the

PCD training outperformed the SA procedure, as it did not reach a plateau in the early epochs.

Parameter	PCD	SA
Hidden nodes	100	100
α_0	1^{-2}	1^{-2}
α_f	1^{-4}	1^{-4}
δ	0.1	0.07
Batch size	512	256
Gibbs steps	20	-
Reads and Sweeps	-	100

Table 6.8: Best parameters found for PCD and SA generative training

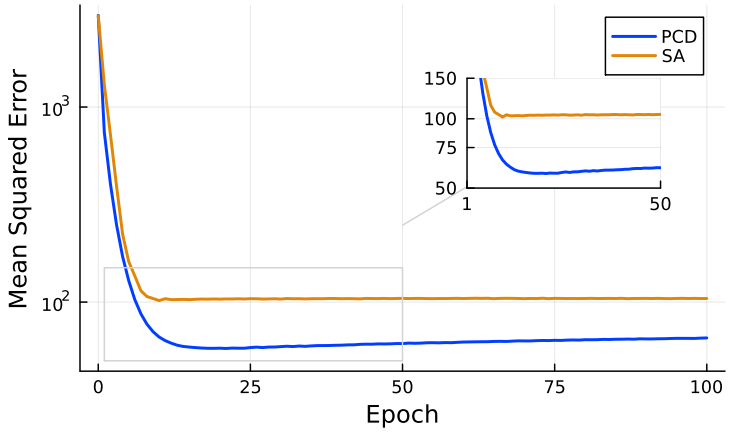

Figure 6.18: Mean Squared Error learning curve for the best RBM variants using PCD and SA training.

6.2.2

RBM-MLP discriminative training

Having the two best RBMs trained generatively with PCD and SA, we were able to attach each of them to a MLP and train it using the data from the hidden nodes from its corresponding RBM. However, the data from the hidden variables is treated in a normalization step before the visible layer of the MLP consuming it. This normalization step uses the mean (μ^{Train}) and standard deviation (σ^{Train}) of the RBM's hidden node values for the training dataset.

Moreover, the chosen MLP network had one hidden and one output layers, of 128 and 21 nodes, where the last one is responsible for classifying the data. Therefore, considering that the trained RBMs were comprised of 100 hidden nodes, the full MLP structure was 100 x 128 x 21, with a ReLU activation function[1] between the visible and hidden layers, and a Softmax function

[1]ReLU activation function: https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

between the hidden and output layers[2]. This configuration is illustrated in Figure 6.19.

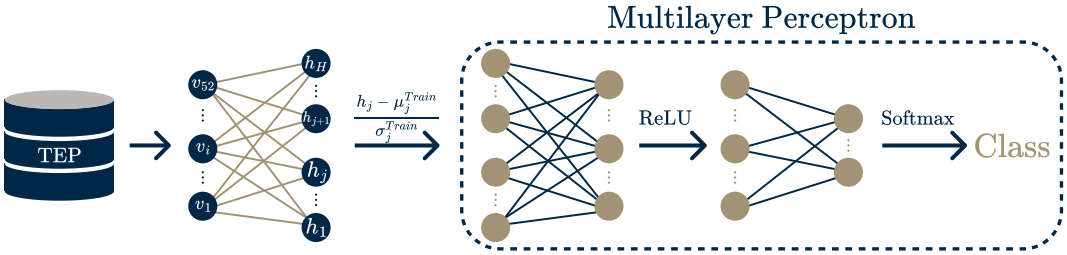

Figure 6.19: RBM-MLP framework for categorizing the 21 classes of the TEP dataset. First, a generative trained RBM (with either PCD or SA methods) receives a sample from the datataset. Then it feeds normalized values from its hidden units. Finally, the MLP network, which is comprised of one hidden and one classification layer labels the input sample with one of the 21 possible classes.

As MLPs are not the focus of this thesis, they were used solely as a device to validate whether it would be possible to improve the results from an RBM. Thus, in this experimental section, the training parameters of the MLPs were not tuned using a grid search, as commonly seen with RBMs for this experimental chapter. In this regard, in order to use a `QARBoM.jl` RBM combined with a MLP, we have used `Flux.jl`'s [54] MLP instance.

`Flux.jl` is a Machine Learning (ML) library written in Julia, that is comprised with several ML models and training procedures. Having two Julia packages allows us to seamlessly integrate `QARBoM.jl` and `Flux.jl` code.

Besides evaluating the improvement in classification accuracy of an RBM-MLP model compared to an RBM, it is important to check the performance of a single MLP. Therefore, for this benchmark, we have also considered a single MLP with the following structure: 52 x 128 x 64 x 21. Notice that the visible layer is not comprised of 100 variables because it is not attached to a 100 hidden node RBM. It receives the data samples from the TEP dataset just as the visible nodes from an RBM do. Additionally, the single MLP has one more hidden layer because it is not connected to any other neural network and we aiming for a more fair comparison.

All MLP instances were trained for 300 epochs, yielding the results with the test dataset presented in Figure 6.20, where it is clear that combining an RBM with a MLP improves the accuracy results.

[2]Softmax activation function: https://en.wikipedia.org/wiki/Softmax_function

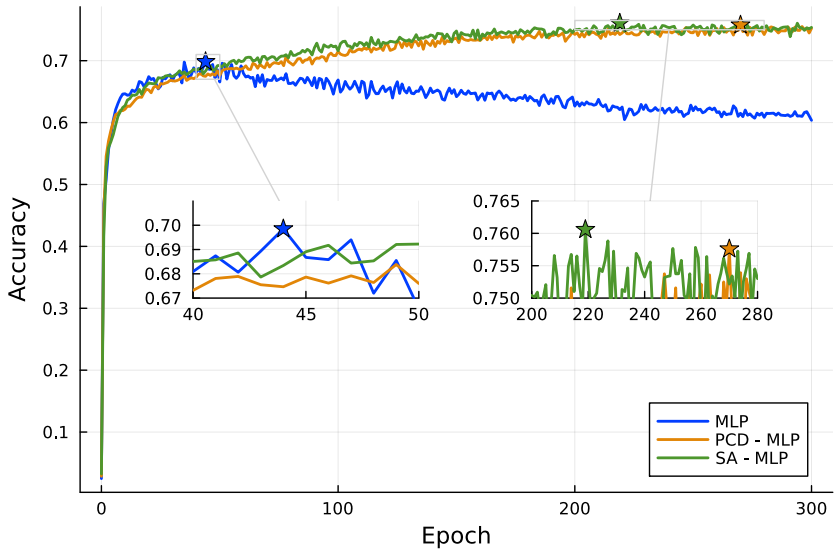

Figure 6.20: Learning curves for three different neural network combinations: (*i*) a single MLP; (*ii*) an RBM trained with PCD attached to a MLP; (*iii*) an RBM trained with SA attached to a MLP. Notice that combining an RBM with a MLP, regardless of the training procedure, yields a higher classification accuracy.

After the training procedure, all three network variants (PCD-MLP, SA-MLP and MLP) were used to generate confusion matrices for all 21 classes in the TEP dataset using test samples. These matrices are presented in Figures 6.21, 6.22 and 6.23, where it is possible to draw some conclusions. First, the SA-MLP and PCD-MLP configurations presented very similar accuracies (0.76 and 0.758 respectively), both having difficulties to classify faults 3, 9, 15 and 19. It is worth recalling the review of previous works on the TEP dataset in the beginning of this chapter, where it was mentioned that other papers [1; 48; 50; 51] had already found challenges for classifying faults 3, 9 and 15.

Moreover, the single MLP setting presented a clear mislabeling issue where it classified multiple classes as fault 20. This network has also presented difficulties for labeling the challenging faults 3, 9 and 15, besides the normal class.

In sum, although a single MLP already provided better results than a single RBM, trained with either PCD or SA, combining both networks yielded a better classification framework. This outcome lays the groundwork for an investigation on Deep Belief Networks, that will be considered as a future work.

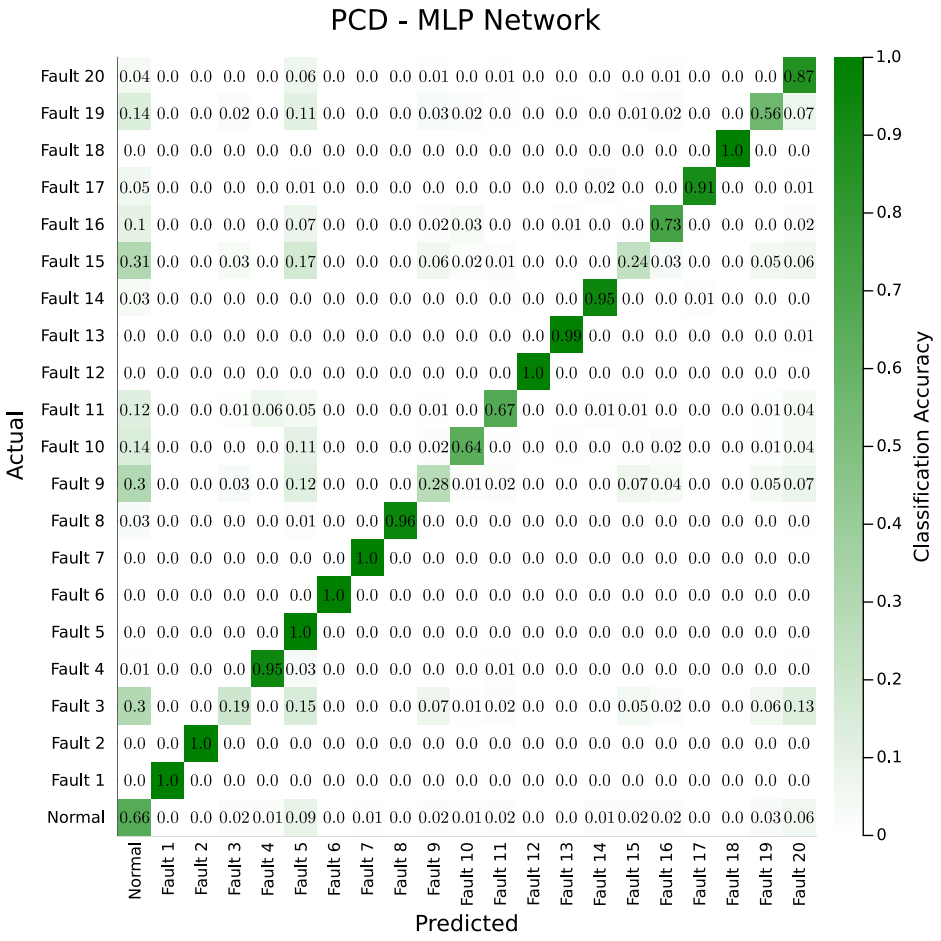

Figure 6.21: Confusion matrix for classifying the 21 classes of the TEP dataset, using an RBM (trained with PCD) paired with a MLP.

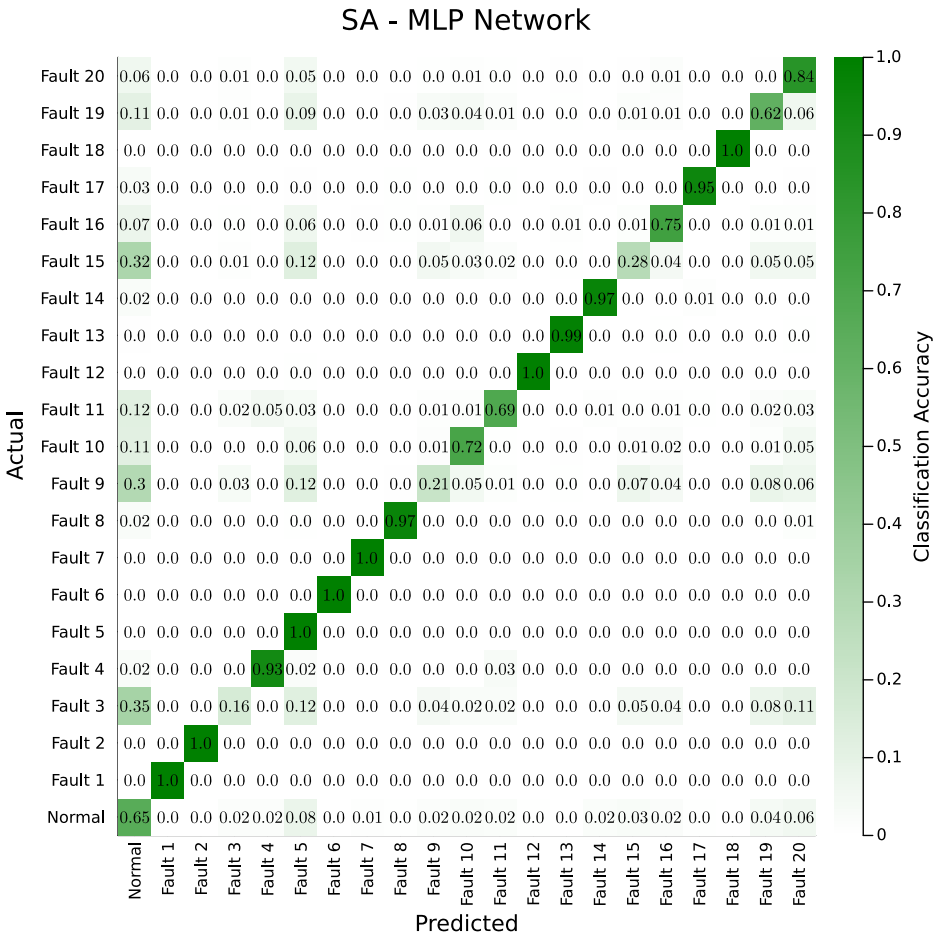

Figure 6.22: Confusion matrix for classifying the 21 classes of the TEP dataset, using an RBM (trained with SA) paired with a MLP.

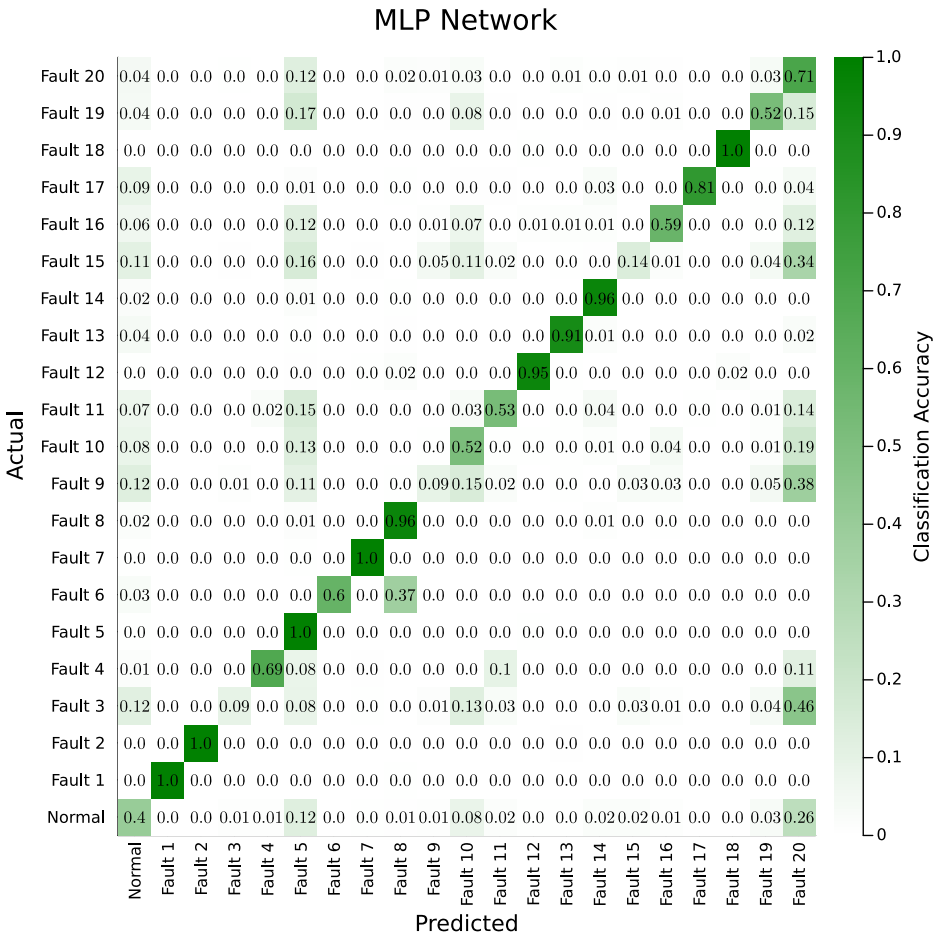

Figure 6.23: Confusion matrix for classifying the 21 classes of the TEP dataset, using a MLP.

6.3

Experiment 3: Quantum Sampling on a real Quantum Annealer

In this experimental section, we finally study the quantum-assisted training of an RBM with a real quantum annealer from D-Wave, using the same dataset from Section 6.1, with 6 classes (1 normal operating mode and 5 faults).

The same package that was used for Simulated Annealing (`DWave.jl`) also provides interface with real quantum annealers, with just a few code adjustments. However, as presented in the following subsection, there are some hardware restrictions that require further adaptations.

6.3.1

Adaptations for D-Wave hardware

In Section 6.1, when using SA, adding more nodes to an RBM had only an impact on the simulation time, as more variables introduced more complexity. However, in real Quantum Annealing, if the number of qubits surpasses the quantum hardware's size, performing any computation becomes impossible. That said, as will be discussed, during the QUBO reformulation, binary expansion precision had to be lowered to a smaller number of variables in the final QUBO model, along with the number of hidden units of the RBM, which, for the SA case, was 200 in its best iteration (see Section 6.1).

Additionally, there is an extra challenge related to the number of variables: the embedding of qubits in the quantum chip. When mapping the QUBO problem into the hardware, the connectivity between qubits need to be taken into account, as the qubit topology of any quantum device is not a fully connected graph. Moreover, in order to make an embedding fit, auxiliary qubits could be added to the problem which, again, might stumble into the qubit amount constraint. This process of sending a reformulated QUBO into D-Wave's platform, which is then embedded and solved by a quantum device, is illustrated in Figure 6.24. For more details about embedding algorithms and D-Wave hardware, please refer to Klymko *et al* [55] and the D-Wave documentation[3].

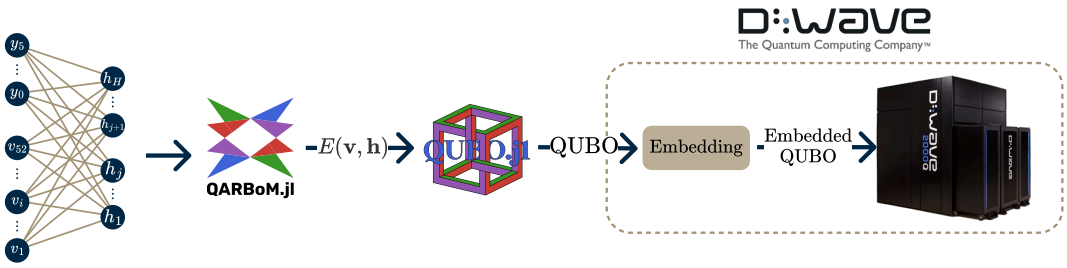

Figure 6.24: Workflow of the RBM's energy function being sent to a D-Wave's device. `QARBoM.jl` uses `QUBO.jl` to reformulate the energy function $E(\mathbf{v}, \mathbf{h})$ into a QUBO, which is then sent to D-Wave's cloud service. After being uploaded, this QUBO model rewritten by an embedding algorithm in order to fit the quantum hardware's topology.

Having these two related issues in mind, it was necessary to alter the binary expansion precision during the QUBO reformulation of the RBM's energy function. As disclosed in Appendix D, when representing a continuous variable with

[3]D-Wave Documentation - Topologies (link)

binary ones, it is important to set a value for representation error τ, which is related to the precision of the binary expansion. Increasing τ reduces the quality of the reformulation in exchange for a smaller number of variables in the final QUBO. This parameter, which in `QUBO.jl` has a default value of 0.25, can be altered inside `QARBoM.jl`'s `variable_encoding_tolerance` argument, as depicted in the code snippet bellow.

Listing 6.1: `QARBoM.jl` code for altering the binary expansion error τ

```
using QARBoM, D-Wave

rbm = GRBM(10,5)

QARBoM.train!(rbm,
    ...
    QSampling;
    model_setup=model_setup,
    sampler=D-Wave.Optimizer,
    variable_encoding_tolerance = 0.30
    ...
)
```

Besides the physical restrictions of quantum hardware, which we can circumvent by changing the encoding error tolerance or the number of nodes in the RBM, the process of embedding a QUBO function, which is a classical algorithm, introduces a time overhead during the training process. Fortunately, in a lower level control of D-Wave's devices, it is possible to save the embedding map in a first run and reuse it for the remaining sampling routines.

Thus, before conducting the actual experiment, different RBM model and QUBO encoding variants were tested to check whether there was an embedding for each configuration and, if it had, this mapping was saved. Unfortunately, ranging the encoding error tolerance τ from 0.25 to 0.40, no encoding was found for 100 or 200 hidden units. The best (lowest) value found was $\tau = 0.30$ for 50 hidden units which, based on SA results from Figure 6.6, at first glance, could impact negatively in the accuracy.

As an illustration of how this representation error impacts on the total number of variables sent to the quantum computer, Figure 6.25 displays the differences for each value of $\tau \in [0.01, 1.0]$ on the reformulation performed by `ToQUBO.jl`, using an RBM with 52 visible and 6 label units. It is worth noting that, for this case, the amount of hidden units in the RBM only adds a constant value to the total number of variables in the resulting QUBO, as only the visible units take continuous values that need to be expanded into binary variables.

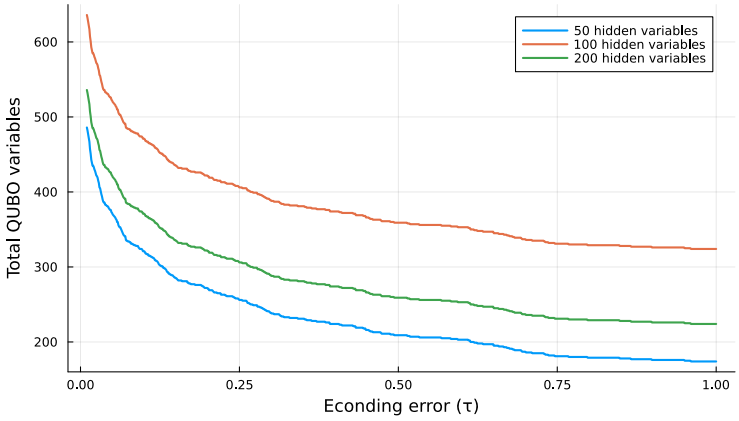

Figure 6.25: Impact of the encoding error $\tau \in [0.01, 1.0]$ on the total number of variables of the reformulated QUBO function from the RBM model, considering 50, 100 and 200 hidden units.

6.3.2

Classical parameter grid search

Performing the same comprehensive grid search of training parameters as in Section 6.1 would require several computing credits on D-Wave cloud. Thus, in order to mitigate this expense, a grid search was performed using SA from D-Wave again, but using a QUBO with the same number of variables that would be sent to a D-Wave device.

As we were more concerned with the outcome of the real quantum experiment, which could not be repeated several times, this evaluation was more detailed with the selection of parameters, using new learning rate schedules that followed the equation below, where different values for α_{final}, α_0 and δ (the decay rate for α) were tested.

$$\alpha_{epoch} = \alpha_{final} + (\alpha_0 - \alpha_{final}) \exp(-\delta * epoch) \tag{6-3}$$

The arguments for the best training version are displayed in Table 6.9. Notice that the learning rates for the visible and label units are actually different, which is a feature that `QARBoM.jl` provides. Moreover, the power of two batch sizes from Section 6.2 were used again, yielding a best batch size of 256. The learning curve for this iteration is presented in Figure 6.26, alongside the real QA procedure, where it is possible to notice that, although we have used an inferior QUBO representation, with higher error tolerance and lower number of hidden units, the best accuracy was as competitive as the ones from

Section 6.1, which used 200 hidden units. A good reason for that is probably the learning rate schedule, which prevented vanishing gradients.

Parameter	Value
Label initial learning rate (α_0^L)	0.01
Label final learning rate (α_{final}^L)	0.001
Label learning rate decay (δ^L)	0.07
Generative initial learning rate (α_0)	0.001
Generative final learning rate (α_{final})	0.0001
Generative learning rate decay (δ)	0.07
Batch size	256
Sweeps	100
Reads	100

Table 6.9: Best Simulated Annealing training parameters for an RBM with 52 continuous visible, 6 binary label and 50 hidden units, using an encoding error $\tau = 0.3$

6.3.3

Quantum-assisted training

Having the training parameters selected from the SA grid search, there were just two parameters left to adjust, that are specifically related to the quantum annealer that `DWave.jl` provides: the annealing time and the number of reads. Trying different values for these arguments, the best iterations had a maximum accuracy of 0.6051 and 0.6011 for 50 and 100 microseconds respectively, both using 100 reads. The learning curves are compared against the best SA iteration in Figure 6.26.

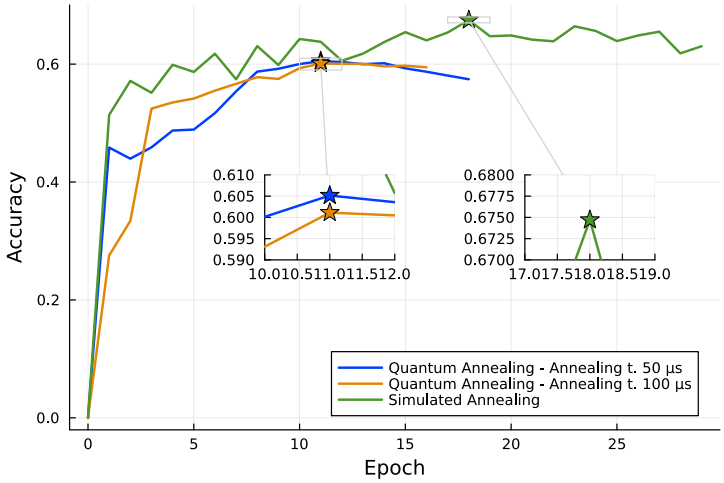

Figure 6.26: Learning curves for real Quantum Annealing (QA) from D-Wave using an annealing time of $50\mu s$ and $100\mu s$, against Simulated Quantum Annealing

Although not achieving the best outcome, it is still unclear whether QA could perform a better training than SA, as with the limited computing time available it was not possible to conduct a comprehensive parameter grid search. Instead, as mentioned, it used the best training parameters for the SA training procedure. However, the best accuracies for QA and SA are not too distant, which validates the importance of benchmarking Quantum-Assisted training of RBMs using `QARBoM.jl`.

7

Conclusions

As discussed in Chapter 1, QC has several potential applications to be explored such as in Chemical Engineering, Machine Learning and Optimization. However, as a proof of Quantum Advantage is yet to be presented for a real-world problem, there is a need for benchmarking tools capable of establishing a fair environment for comparing classical and quantum algorithms.

That said, `QARBoM.jl` is able to fill the gap for benchmarking tools for RBM training, where QC can be used as a replacement for a subroutine that is currently performed by a classical approximation algorithm (Gibbs Sampling). Additionally, as we have seen in chapter 6, besides quantum methods, any method capable of sampling low energy states for a QUBO or Ising Model, such as Simulated Annealing, can be used as a candidate to aid the training of RBMs.

Being an open-source Julia package, `QARBoM.jl` is accessible for any research group interested in evaluating RBM training with different types of algorithms, both classical and quantum. However, using `QARBoM.jl` with real quantum-sampling requires privileged access to quantum hardware, considering queue and computing time constraints.

Future works can contemplate different features for this package. First, as stated in Chapter 5, `QARBoM.jl` still lacks methods for working with DBNs, where each layer should be optimized with any of the training methods developed for regular RBMs. Moreover, `QARBoM.jl` leverages Julia's multiple dispatch paradigm, making it easy for its users to include new training algorithms that could be contemplated in a future benchmark.

Another area for improvement is the expansion of `QABoM.jl` RBM coverage for other Energy Based Models, such as Boltzmann Machines and Hopfield Networks. This might require the creation of a new library that contains `QARBoM.jl`.

Considering the different devices and methods capable of performing Quantum Sampling, `QARBoM.jl` users can just make use of `QUBO.jl`'s extension packages for sampling solutions for QUBOs in quantum computers. As `QUBO.jl`'s environment develops, there will be more quantum-assisted training options for `QARBoM.jl`.

Finally, the experiments presented in Chapter 6 highlighted the current drawbacks of quantum computers, that limit the size of the RBM that can be trained using quantum-assisted algorithms. As we move from the NISQ Era towards a Fault-Tolerant Quantum Computing Era, it is expected that we will be able to perform more comprehensive benchmarks between quantum-assisted and classical training of RBMs. However, there has been a change in the accessibility to real quantum computers over the last few years, as the interest in this technology increases and the hardware improves. Thus, as already discussed in this chapter, even though `QARBoM.jl` is an open-source library, researchers need to acquire privileged access to perform their experiments.

A

Conditional probabilities

In Section 2.3, we have disclosed that estimating the conditional probabilities for each node is as simple as:

$$\begin{aligned} p(h_j = 1|\mathbf{v}) &= \sigma(b_j + \sum_i w_{ij} v_i) \\ p(v_i = 1|\mathbf{h}) &= \sigma(a_i + \sum_j w_{ij} h_j), \text{ where } \sigma(x) = 1/(1+e^{-x}) \end{aligned} \tag{A-1}$$

In this appendix, we will go over some lengthy manipulations that get us to these simple equations. First, let us rewrite Equation 2-8:

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{v},\mathbf{h})}{p(\mathbf{v})} = \frac{\exp(-E(\mathbf{v},\mathbf{h}))/Z}{\sum_{\mathbf{n}} \exp(-E(\mathbf{v},\mathbf{h'^n}))/Z} \tag{A-2}$$

Dividing the numerator and denominator by $\frac{1}{Z}$ and expanding the energy function E:

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp(\sum_i a_i v_i + \sum_j b_j h_j + \sum_{ij} w_{ij} v_i h_j)}{\sum_{\mathbf{n}} \exp(\sum_i a_i v_i + \sum_j b_j \mathbf{h'^n}_j + \sum_{ij} w_{ij} v_i \mathbf{h'^n}_j)} \tag{A-3}$$

Separating the sum over all visible nodes makes it simple to remove them from the fraction

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp(\sum_i a_i v_i)\exp(\sum_j b_j h_j + \sum_{ij} w_{ij} v_i h_j)}{\exp(\sum_i a_i v_i)\sum_{\mathbf{n}} \exp(\sum_j b_j \mathbf{h'^n}_j + \sum_{ij} w_{ij} v_i \mathbf{h'^n}_j)} \tag{A-4}$$

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp(\sum_j b_j h_j + \sum_{ij} w_{ij} v_i h_j)}{\sum_{\mathbf{n}} \exp(\sum_j b_j \mathbf{h'^n}_j + \sum_{ij} w_{ij} v_i \mathbf{h'^n}_j)} \tag{A-5}$$

Simplifying the denominator

For the next step of our manipulations, we need to rearrange the denominator from Equation A-5. First, we know that $\sum_{\mathbf{n}}$ is summing over all possible values that the $\mathbf{h}$ vector can take. Considering that $\mathbf{h} \in \mathbb{B}^H$, where $H = dim(H)$, we can expand the summation as

$$\begin{aligned} &\sum_{\mathbf{n}} \exp(\sum_j b_j \mathbf{h'^n}_j + \sum_{ij} w_{ij} v_i \mathbf{h'^n}_j) \\ =&\exp(\sum_j b_j \mathbf{h'^1}_j + \sum_{ij} w_{ij} v_i \mathbf{h'^1}_j) + \cdots + \exp(\sum_j b_j \mathbf{h'^N}_j + \sum_{ij} w_{ij} v_i \mathbf{h'^N}_j) \end{aligned} \tag{A-6}$$

Considering that each $\mathbf{h'^n}$ is a possible value for the $\mathbf{h}$ vector, if $dim(\mathbf{h}) = 3$, for instance, one could say that $\mathbf{h'^1} = [0, 0, 0]$, $\mathbf{h'^2} = [0, 0, 1]$ and so on. With that in mind, we can drop the vector notation and consider each hidden variable value as $h_j^{=0}$ if the j-th hidden variable is zero and $h_j^{=1}$ in the case that it is equal to one. Using this new notation, it is possible to rewrite Equation A-6 as

$$\begin{aligned} &\exp\Big([b_1 h_1^{=0} + b_2 h_2^{=0} + \cdots + b_H h_H^{=0}] + \sum_i [w_{i1} v_i h_1^{=0} + \cdots + w_{iH} v_i h_H^{=0}]\Big) \\ +&\exp\Big([b_1 h_1^{=0} + b_2 h_2^{=0} + \cdots + b_H h_H^{=1}] + \sum_i [w_{i1} v_i h_1^{=0} + \cdots + w_{iH} v_i h_H^{=1}]\Big) \\ +&\cdots \\ +&\exp\Big([b_1 h_1^{=1} + b_2 h_2^{=1} + \cdots + b_H h_H^{=1}] + \sum_i [w_{i1} v_i h_1^{=1} + \cdots + w_{iH} v_i h_H^{=1}]\Big) \end{aligned} \tag{A-7}$$

After this manipulation, we can factor out each h_j variable:

$$
\begin{aligned}
&\exp\left(h_1^{=0}[b_1 + \sum_i w_{i1}v_i] + \cdots + h_H^{=0}[b_H + \sum_i w_{iH}v_i]\right) \\
&+ \exp\left(h_1^{=0}[b_1 + \sum_i w_{i1}v_i] + \cdots + h_H^{=1}[b_H + \sum_i w_{iH}v_i]\right) \\
&+ \cdots \\
&+ \exp\left(h_1^{=1}[b_1 + \sum_i w_{i1}v_i] + \cdots + h_H^{=1}[b_H + \sum_i w_{iH}v_i]\right)
\end{aligned}
\tag{A-8}
$$

At this step, inside each exp() term, we have a sum of $dim(\mathbf{h}) = H$ terms. Let us rewrite these sums as products of exponentials.

$$
\begin{aligned}
&\exp\left(h_1^{=0}[b_1 + \sum_i w_{i1}v_i]\right) \times \cdots \times \exp\left(h_H^{=0}[b_H + \sum_i w_{iH}v_i]\right) \\
&+ \exp\left(h_1^{=0}[b_1 + \sum_i w_{i1}v_i]\right) \times \cdots \times \exp\left(h_H^{=1}[b_H + \sum_i w_{iH}v_i]\right) \\
&+ \cdots \\
&+ \exp\left(h_1^{=1}[b_1 + \sum_i w_{i1}v_i]\right) \times \cdots \times \exp\left(h_H^{=1}[b_H + \sum_i w_{iH}v_i]\right)
\end{aligned}
\tag{A-9}
$$

Now we can factor each $\exp\left(h_j^{=0}[b_j + \sum_i w_{ij}v_i]\right)$. Starting with $j = 1$:

$$
\begin{aligned}
&\exp\left(h_1^{=0}[b_1 + \sum_i w_{i1}v_i]\right)\Big[\\
&\quad \exp\left(h_2^{=0}[b_2 + \sum_i w_{i2}v_i]\right) \times \cdots \times \exp\left(h_H^{=0}[b_H + \sum_i w_{iH}v_i]\right) \\
&+ \cdots \\
&+ \exp\left(h_2^{=1}[b_2 + \sum_i w_{i2}v_i]\right) \times \cdots \times \exp\left(h_H^{=1}[b_H + \sum_i w_{iH}v_i]\right)\Big] \\
&+ \exp\left(h_1^{=1}[b_1 + \sum_i w_{i1}v_i]\right)\Big[\\
&\quad \exp\left(h_2^{=0}[b_2 + \sum_i w_{i2}v_i]\right) \times \cdots \times \exp\left(h_H^{=0}[b_H + \sum_i w_{iH}v_i]\right) \\
&+ \cdots \\
&+ \exp\left(h_2^{=1}[b_2 + \sum_i w_{i2}v_i]\right) \times \cdots \times \exp\left(h_H^{=1}[b_H + \sum_i w_{iH}v_i]\right)\Big]
\end{aligned}
\tag{A-10}
$$

As we have factor the exponential terms for $h_1 = 0$ and $h_1 = 1$, yielding two terms, we can rewrite it as follows.

$$\sum_{h_1 \in \{0,1\}} \exp\Big(h_1[b_1 + \sum_i w_{i1} v_i]\Big) \Big[\\ \exp\Big(h_2^{=0}[b_2 + \sum_i w_{i2} v_i]\Big) \times \cdots \times \exp\Big(h_H^{=0}[b_H + \sum_i w_{iH} v_i]\Big) \\ + \cdots \\ + \exp\Big(h_2^{=1}[b_2 + \sum_i w_{i2} v_i]\Big) \times \cdots \times \exp\Big(h_H^{=1}[b_H + \sum_i w_{iH} v_i]\Big)\Big] \tag{A-11}$$

Continuing to factor each $\exp\Big(h_j^{=0}[b_j + \sum_i w_{ij} v_i]\Big)$ term and using the $\sum_{h_j \in \{0,1\}}$ maneuver, we would get the following.

$$\sum_{h_1 \in \{0,1\}} \exp\Big(h_1[b_1 + \sum_i w_{i1} v_i]\Big) \Big[\\ \sum_{h_2 \in \{0,1\}} \exp\Big(h_2[b_2 + \sum_i w_{i2} v_i]\Big) \Big[\cdots \Big[\sum_{h_H \in \{0,1\}} \exp\Big(h_H[b_H + \sum_i w_{iH} v_i]\Big) \Big] \Big] \Big] \tag{A-12}$$

Rearranging:

$$\sum_{h_1 \in \{0,1\}} \exp\Big(h_1[b_1 + \sum_i w_{i1} v_i]\Big) \times \cdots \times \sum_{h_H \in \{0,1\}} \exp\Big(h_H[b_H + \sum_i w_{iH} v_i]\Big) \tag{A-13}$$

Observing Equation A-13, each $\exp\Big(h_j[b_j + \sum_i w_{ij} v_i]\Big)$ term can take the following values:

$$\exp\Big(h_j[b_j + \sum_i w_{ij} v_i]\Big) = \begin{cases} 1, & \text{if } h_j = 0 \\ \exp(b_j + \sum_i w_{ij} v_i), & \text{if } h_j = 1 \end{cases} \tag{A-14}$$

Thus, it is possible to rewrite each summation term as

$$\sum_{h_j \in \{0,1\}} \exp\Big(h_j[b_j + \sum_i w_{ij} v_i]\Big) = (1 + \exp(b_j + \sum_i w_{ij} v_i)) \tag{A-15}$$

Finally, as we once had a product of exponentials in Equation A-13, we now have the following simplified denominator.

$$\sum_{h_1 \in \{0,1\}} \exp\left(h_1[b_1 + \sum_i w_{i1} v_i]\right) \times \cdots \times \sum_{h_H \in \{0,1\}} \exp\left(h_H[b_H + \sum_i w_{iH} v_i]\right) = \prod_j (1 + \exp(b_j + \sum_i w_{ij} v_i)) \tag{A-16}$$

Finding the simplified conditional probability

Now that we have a simpler denominator, let us replace it in Equation A-5:

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp(\sum_j b_j h_j + \sum_{ij} w_{ij} v_i h_j)}{\prod_j (1 + \exp(b_j + \sum_i w_{ij} v_i))} \tag{A-17}$$

We can simplify the equation even further working on the numerator.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp(\sum_j b_j h_j + \sum_{ij} w_{ij} v_i h_j)}{\prod_j (1 + \exp(b_j + \sum_i w_{ij} v_i))} = \frac{\exp(\sum_j b_j [h_j + \sum_i w_{ij} v_i])}{\prod_j (1 + \exp(b_j + \sum_i w_{ij} v_i))} \tag{A-18}$$

Manipulating the exponential:

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp(\sum_j b_j [h_j + \sum_i w_{ij} v_i])}{\prod_j (1 + \exp(b_j + \sum_i w_{ij} v_i))} = \frac{\prod_j \exp(b_j [h_j + \sum_i w_{ij} v_i])}{\prod_j (1 + \exp(b_j + \sum_i w_{ij} v_i))} \tag{A-19}$$

Concluding, considering that the hidden variables are independent, given the visible variables, we can write $p(\mathbf{h}|\mathbf{v})$ as

$$p(\mathbf{h}|\mathbf{v}) = \prod_j \frac{\exp(b_j [h_j + \sum_i w_{ij} v_i])}{(1 + \exp(b_j + \sum_i w_{ij} v_i))} = \prod_j p(h_j|\mathbf{v}) \tag{A-20}$$

Finally, from Equation A-20, it is noticeable that $p(\mathbf{h}|\mathbf{v})$ does not require iterating over all possible states of the visible or hidden nodes. Thus, it can be estimated in polynomial time. Moreover, one can estimate the conditional probability for a single node, as follows.

$$p(h_j = 1|\mathbf{v}) = \frac{\exp(b_j + \sum_i w_{ij} v_i)}{1 + \exp(b_j + \sum_i w_{ij} v_i)} \tag{A-21}$$

Which can be rearranged to:

$$\begin{aligned} p(h_j = 1|\mathbf{v}) &= \frac{\exp(b_j + \sum_i w_{ij} v_i)}{1 + \exp(b_j + \sum_i w_{ij} v_i)} \cdot \frac{\exp(-b_j - \sum_i w_{ij} v_i)}{\exp(-b_j - \sum_i w_{ij} v_i)} \\ &= \frac{1}{1 + \exp(-b_j - \sum_i w_{ij} v_i)} \end{aligned} \tag{A-22}$$

This probability can be represented by the sigmoid function $\sigma(x) = 1/1 + \exp(-x)$

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i w_{ij} v_i) \tag{A-23}$$

The same simplifications performed in this appendix can be applied for the visible nodes, yielding:

$$p(v_i = 1|\mathbf{h}) = \sigma(a_i + \sum_j w_{ij} h_j) \tag{A-24}$$

B

Calculating learning gradients

In Section 2.5, we defined the objective function in Equation B-1 for training RBMs. In this optimization problem we are trying to maximize the value of $\log p(\mathbf{v})$ over a set of decision variables $\Theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$.

$$\max_{\Theta} \log p(\mathbf{v}) \tag{B-1}$$

These variables appear inside the probability $p(\mathbf{v})$, which depends on the Energy Function of the RBM and a partition function Z, as defined in Equation B-2, where:

- Upper-case and bold letter $\mathbf{W}$ represent the matrix of weights for the RBM
- Lower-case bold letters $\mathbf{a}, \mathbf{b}$ depict the vectors of bias terms for visible and hidden nodes, while $\mathbf{v}, \mathbf{h}$ represent the vectors of visible and hidden nodes, respectively
- $\mathbf{v}' = \{\mathbf{v'^1}, \cdots, \mathbf{v'^M}\}$ and $\mathbf{h}' = \{\mathbf{h'^1}, \cdots, \mathbf{h'^N}\}$ are sets that represent all the possible values that each visible and hidden vector can take
- $\mathbf{v'^m}$ and $\mathbf{h'^n}$ portray the m-th and n-th possible values for the visible and hidden nodes vectors, respectively.
- $\mathbf{v}$ without a superscript is a training data sample

$$\begin{aligned}
\mathrm{p}(\mathbf{v}) =& \frac{1}{Z}\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^{n}})) \\
=& \frac{\sum_{\mathbf{n}} \exp(\mathbf{v}^T\mathbf{W}\mathbf{h'^{n}} + \mathbf{a}^T\mathbf{v} + \mathbf{b}^T\mathbf{h'^{n}})}{\sum_{\mathbf{m,n}} \exp(-E(\mathbf{v'^{m}}, \mathbf{h'^{n}})} \\
=& \frac{\sum_{\mathbf{n}} \exp(\mathbf{v}^T\mathbf{W}\mathbf{h'^{n}} + \mathbf{a}^T\mathbf{v} + \mathbf{b}^T\mathbf{h'^{n}})}{\sum_{\mathbf{m,n}} \exp(\mathbf{v'^{m}}^T\mathbf{W}\mathbf{h'^{n}} + \mathbf{a}^T\mathbf{v'^{m}} + \mathbf{b}^T\mathbf{h'^{n}})}
\end{aligned} \tag{B-2}$$

The objective function in Equation B-1 is optimized via gradient descent, taking partial derivatives over each decision variable $\mathbf{W}, \mathbf{a}, \mathbf{b}$. However, as mentioned in Section 2.5, estimating the gradient for Equation B-1 can be a daunting task.

Fortunately, there are some algebraic manipulations one can apply in order to simplify the gradient equations. In this appendix, we will go through these simplifications, which have been taken from Fischer and Igel [15] and professor Hugo Larochelle's online course on Neural Networks [1].

First, let us expand $\log p(\mathbf{v})$.

$$\begin{aligned}
\log p(\mathbf{v}) &= \log\left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^{n}}))/Z\right) \\
&= \log\left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^{n}}))\right) - \log(Z) \\
&= \log\left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^{n}}))\right) - \log\left(\sum_{\mathbf{m,n}} \exp(-E(\mathbf{v'^{m}}, \mathbf{h'^{n}}))\right)
\end{aligned} \tag{B-3}$$

Now, let us apply the derivative over Θ.

$$\begin{aligned}
\frac{\partial}{\partial\Theta}\log p(\mathbf{v}) = & \frac{\partial}{\partial\Theta}\log\left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^{n}}))\right) \\
& - \frac{\partial}{\partial\Theta}\log\left(\sum_{\mathbf{m,n}} \exp(-E(\mathbf{v'^{m}}, \mathbf{h'^{n}}))\right)
\end{aligned} \tag{B-4}$$

[1] https://larocheh.github.io/neural_networks/content.html

Simplifying the first term

For easier understanding, we will work on each term separately. Starting with the first one, we can apply the chain rule, yielding the following:

$$\begin{aligned}
&\frac{\partial}{\partial \Theta} \log \left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n})) \right) \\
&= \frac{1}{\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))} \cdot \frac{\partial}{\partial \Theta} \sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n})) \\
&= \frac{1}{\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))} \cdot \sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n})) \frac{\partial}{\partial \Theta} (-E(\mathbf{v}, \mathbf{h'^n})) \\
&= \frac{\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n})) \frac{\partial}{\partial \Theta} (-E(\mathbf{v}, \mathbf{h'^n}))}{\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))}
\end{aligned} \tag{B-5}$$

Now, considering that

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{h}, \mathbf{v})}{p(\mathbf{v})} = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))} \tag{B-6}$$

We can replace it in the equation, giving us

$$\begin{aligned}
&\frac{\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n})) \frac{\partial}{\partial \Theta} (-E(\mathbf{v}, \mathbf{h'^n}))}{\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))} = \sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v}) \cdot \frac{\partial}{\partial \Theta} (-E(\mathbf{v}, \mathbf{h'^n})) \\
&= -\sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v}) \cdot \frac{\partial}{\partial \Theta} (E(\mathbf{v}, \mathbf{h'^n}))
\end{aligned} \tag{B-7}$$

Thus far, we have been working with the total derivative of all hyperparameters. From now on, for this example, we will be concerned with the partial derivative of a weight w_{ij}.

$$
\begin{aligned}
&-\sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v}) \cdot \frac{\partial}{\partial \Theta}(E(\mathbf{v}, \mathbf{h'^n})) = \\
&-\sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v}) \cdot \frac{\partial}{\partial w_{ij}}(E(\mathbf{v}, \mathbf{h'^n})) = \sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v}) v_i h_j
\end{aligned}
\tag{B-8}
$$

In Appendix A, we have proved that:

$$
p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) \tag{B-9}
$$

Therefore we can use this identity and manipulate $\sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v})$ as presented in Equation B-10 below, where $\mathbf{n}_k$ stands for the k-th hidden node value from the n-th possible configuration for all hidden nodes, as:

- $\mathbf{h'} = \{\mathbf{h'^1}, \cdots, \mathbf{h'^N}\}$
- $\mathbf{h'^n} = [\mathbf{h}'^n_1, \cdots, \mathbf{h}'^n_K]$, where $K = dim(\mathbf{h})$

$$
\sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v}) v_i h_j = \sum_{\mathbf{n}} \prod_k p(\mathbf{h'^n}_k|\mathbf{v}) v_i h_j \tag{B-10}
$$

It is very important remember that the energy function had $\mathbf{h'^n}$ as a parameter. Thus, although we have already solved the partial derivative $\frac{\partial}{\partial w_{ij}}(E(\mathbf{v}, \mathbf{h'^n}))$, h_j can be interpreted as presented in Equation B-11.

$$
h_j = \mathbf{h'^n}_k, \text{ where } j = k \tag{B-11}
$$

Now, we will use some extra manipulations that will help us to simplify the current equation even further. First, let us expand the product that was introduced in Equation B-10.

$$
\sum_{\mathbf{n}} \prod_k p(\mathbf{h'^n}_k|\mathbf{v}) v_i h_j = \sum_{\mathbf{n}} \left[p(\mathbf{h'^n}_1|\mathbf{v}) \times \cdots \times p(\mathbf{h'^n}_K|\mathbf{v})\right] v_i h_j \tag{B-12}
$$

Considering that $\mathbf{h} \in \mathbb{B}^K$, we are summing over all possible binary combinations for the hidden nodes. Thus, one could expand this summation in the

following way.

$$\begin{aligned}
&\sum_{\mathbf{n}} \left[p(\mathbf{h'^{n}}_1|\mathbf{v}) \times \cdots \times p(\mathbf{h'^{n}}_K|\mathbf{v})\right] v_i h_j \\
&= p(h_1 = 0|\mathbf{v}) \times \cdots \times p(h_K = 0|\mathbf{v}) v_i (h_j = 0) \\
&+ p(h_1 = 0|\mathbf{v}) \times \cdots \times p(h_K = 1|\mathbf{v}) v_i (h_j = 0) \\
&+ \cdots + p(h_1 = 1|\mathbf{v}) \times \cdots \times p(h_K = 1|\mathbf{v}) v_i (h_j = 1)
\end{aligned} \tag{B-13}$$

Then, we can start excluding the h_1 term as presented below:

$$\begin{aligned}
&\sum_{\mathbf{n}} \left[p(\mathbf{h'^{n}}_1|\mathbf{v}) \times \cdots \times p(\mathbf{h'^{n}}_K|\mathbf{v})\right] v_i h_j \\
&= p(h_1 = 0|\mathbf{v}) \Big[p(h_2 = 0|\mathbf{v}) \times \cdots \times p(h_K = 0|\mathbf{v}) v_i (h_j = 0) + \cdots + \\
&\qquad\qquad p(h_2 = 1|\mathbf{v}) \times \cdots \times p(h_K = 1|\mathbf{v}) v_i (h_j = 1) \Big] \\
&+ p(h_1 = 1|\mathbf{v}) \Big[p(h_2 = 0|\mathbf{v}) \times \cdots \times p(h_K = 0|\mathbf{v}) v_i (h_j = 0) + \cdots + \\
&\qquad\qquad p(h_2 = 1|\mathbf{v}) \times \cdots \times p(h_K = 1|\mathbf{v}) v_i (h_j = 1) \Big] \\
&= \sum_{h_1 \in \{0,1\}} p(h_1|\mathbf{v}) \Big[p(h_2 = 0|\mathbf{v}) \times \cdots \times p(h_K = 0|\mathbf{v}) (h_j = 0) + \cdots + \\
&\qquad\qquad p(h_2 = 1|\mathbf{v}) \times \cdots \times p(h_K = 1|\mathbf{v}) (h_j = 1) \Big] v_i
\end{aligned} \tag{B-14}$$

We can repeat this method to the other h_k terms, yielding the following equation:

$$\begin{aligned}
&\Big[\sum_{h_1 \in \{0,1\}} p(h_1|\mathbf{v}) \times \sum_{h_2 \in \{0,1\}} p(h_2|\mathbf{v}) \times \cdots \times \sum_{h_j \in \{0,1\}} p(h_j|\mathbf{v}) h_j \times \cdots \\
&\times \sum_{h_K \in \{0,1\}} p(h_K|\mathbf{v}) \Big] v_i
\end{aligned} \tag{B-15}$$

Considering that $\mathbf{v}$ is given, looking at the possible values that h_j can take help us to better simplify the equation. As $h_j \in \{0, 1\}$, if it is zero, the whole equation is equal zero. Therefore, we are only concerned with $h_j = 1$. Due to this fact, we can replace the summation for h_j for just the conditional probability, as presented below.

$$\begin{aligned}
&\Big[\sum_{h_1\in\{0,1\}} p(h_1|\mathbf{v}) \times \cdots \times \sum_{h_j\in\{0,1\}} p(h_j|\mathbf{v})h_j \times \cdots \times \sum_{h_K\in\{0,1\}} p(h_K|\mathbf{v})\Big]v_i \\
&= \Big[\sum_{h_1\in\{0,1\}} p(h_1|\mathbf{v}) \times \cdots \times p(h_j=1|\mathbf{v})\cdot 1 \times \cdots \times \sum_{h_K\in\{0,1\}} p(h_K|\mathbf{v})\Big]v_i \\
&= \Big[\sum_{h_1\in\{0,1\}} p(h_1|\mathbf{v}) \times \cdots \times \sum_{h_K\in\{0,1\}} p(h_K|\mathbf{v})\Big]p(h_j=1|\mathbf{v})v_i
\end{aligned} \tag{B-16}$$

Concluding, as we are summing over all possible values for the terms different than h_j, we can use the fact that $\sum_{h_k\in\{0,1\}} p(h_k|\mathbf{v}) = 1$ to replace the product of sums to 1, yielding the final simplification for the first term of the partial derivative (in Equation B-4):

$$\frac{\partial}{\partial w_{ij}} \log\left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))\right) = p(h_j=1|\mathbf{v})v_i \tag{B-17}$$

Considering the other partial derivatives, over a_i and b_j, the main differences would be at the simplification step in Equation B-16. We will briefly explain them in the following paragraphs.

For the a_i partial derivative, $\frac{\partial}{\partial a_i}(-E(\mathbf{v}, \mathbf{h'^n}) = v_i$, meaning that at the end of Equation B-16, we would not have an h_j term. Therefore, we would also consider that $\sum_{h_j\in\{0,1\}} p(h_j|\mathbf{v}) = 1$, resulting in:

$$\frac{\partial}{\partial a_i} \log\left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))\right) = v_i \tag{B-18}$$

Finally, for the b_j gradient, $\frac{\partial}{\partial b_j}(-E(\mathbf{v}, \mathbf{h'^n}) = h_j$. Therefore, we would not have a v_i term as in Equation B-17, yielding:

$$\frac{\partial}{\partial b_j} \log\left(\sum_{\mathbf{n}} \exp(-E(\mathbf{v}, \mathbf{h'^n}))\right) = p(h_j=1|\mathbf{v}) \tag{B-19}$$

Simplifying the second term

Returning to Equation B-4, we can rewrite it with the first term manipulations, considering the partial derivative over w_{ij}, as presented below.

$$\frac{\partial}{\partial w_{ij}} \log p(\mathbf{v}) = p(h_j = 1|\mathbf{v})v_i \\ - \frac{\partial}{\partial w_{ij}} \log \left(\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})) \right) \tag{B-20}$$

Now, let us work on the second term of the equation.

$$\frac{\partial}{\partial w_{ij}} \log \left(\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})) \right)$$

$$= -\frac{1}{\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n}))} \sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})) \frac{\partial}{\partial w_{ij}} E(\mathbf{v'^m}, \mathbf{h'^n}) \tag{B-21}$$

From Equations 2-3 and 2-5, we know that:

$$p(\mathbf{v}, \mathbf{h}) = \frac{\exp(-E(\mathbf{v}, \mathbf{h})}{\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})} \tag{B-22}$$

Therefore, one can rewrite Equation B-21 as:

$$- \frac{\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})) \frac{\partial}{\partial w_{ij}} E(\mathbf{v'^m}, \mathbf{h'^n})}{\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n}))} \tag{B-23}$$

$$= - \sum_{\mathbf{m},\mathbf{n}} p(\mathbf{v'^m}, \mathbf{h'^n}) \frac{\partial}{\partial w_{ij}} E(\mathbf{v'^m}, \mathbf{h'^n})$$

Now, we can manipulate $p(\mathbf{v'^m}, \mathbf{h'^n})$:

$$
\begin{aligned}
&- \sum_{\mathbf{m},\mathbf{n}} p(\mathbf{v'^m}, \mathbf{h'^n}) \frac{\partial}{\partial w_{ij}} E(\mathbf{v'^m}, \mathbf{h'^n}) \\
&= - \sum_{\mathbf{m},\mathbf{n}} p(\mathbf{v'^m}) p(\mathbf{h'^n}|\mathbf{v'^m}) \frac{\partial}{\partial w_{ij}} E(\mathbf{v'^m}, \mathbf{h'^n}) \\
&= - \sum_{\mathbf{m}} p(\mathbf{v'^m}) \sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v'^m}) \frac{\partial}{\partial w_{ij}} E(\mathbf{v'^m}, \mathbf{h'^n})
\end{aligned}
\tag{B-24}
$$

Then, applying the derivative for the energy function:

$$
\begin{aligned}
&- \sum_{\mathbf{m}} p(\mathbf{v'^m}) \sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v'^m}) \frac{\partial}{\partial w_{ij}} E(\mathbf{v'^m}, \mathbf{h'^n}) \\
&= \sum_{\mathbf{m}} p(\mathbf{v'^m}) \sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v'^m}) v_i h_j
\end{aligned}
\tag{B-25}
$$

Notice that the inner summation is similar to Equation B-10. Therefore, applying the same simplifications for the inner summation we get:

$$
\sum_{\mathbf{m}} p(\mathbf{v'^m}) \sum_{\mathbf{n}} p(\mathbf{h'^n}|\mathbf{v'^m}) v_i h_j = \sum_{\mathbf{m}} p(\mathbf{v'^m}) p(h_j = 1|\mathbf{v'^m}) v_i \tag{B-26}
$$

With these steps, we have achieved the following simplification for the second term of Equation B-4:

$$
\begin{aligned}
&\frac{\partial}{\partial w_{ij}} \log \left(\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})) \right) \\
&= \sum_{\mathbf{m}} p(\mathbf{v'^m}) p(h_j = 1|\mathbf{v'^m}) v_i
\end{aligned}
\tag{B-27}
$$

Furthermore, as presented in Equations B-18 and B-19, there are some changes when considering the gradient for the two other hyperparameters a_i and b_j.

Using the same arguments previously presented for the first term, we get the following equations for each partial derivative:

$$
\frac{\partial}{\partial a_i} \log \left(\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})) \right) = \sum_{\mathbf{m}} p(\mathbf{v'^m}) v_i \tag{B-28}
$$

$$\frac{\partial}{\partial b_j} \log \left(\sum_{\mathbf{m},\mathbf{n}} \exp(-E(\mathbf{v'^m}, \mathbf{h'^n})) \right) = \sum_{\mathbf{m}} p(\mathbf{v'^m}) p(\mathbf{v'^m}) p(h_j = 1 | \mathbf{v'^m}) \tag{B-29}$$

Final simplification

Finally, we can rewrite Equation B-4 as:

$$\frac{\partial}{\partial w_{ij}} \log p(\mathbf{v}) = p(h_j = 1|\mathbf{v}) v_i - \sum_{\mathbf{m}} p(\mathbf{v'^m}) p(h_j = 1|\mathbf{v'^m}) v_i \tag{B-30}$$

$$\frac{\partial}{\partial a_i} \log p(\mathbf{v}) = v_i - \sum_{\mathbf{m}} p(\mathbf{v'^m}) v_i \tag{B-31}$$

$$\frac{\partial}{\partial b_j} \log p(\mathbf{v}) = p(h_j = 1|\mathbf{v}) - \sum_{\mathbf{m}} p(\mathbf{v'^m}) p(h_j = 1|\mathbf{v'^m}) \tag{B-32}$$

It is important to notice that the first term of Equation B-30 considers an empirical distribution, i.e., it depends on the observed data $\mathbf{v}$, whilst the second considers a probabilistic distribution over the mathematical model of the RBM, with respect to the current weights $\mathbf{W}$, $\mathbf{a}$ and $\mathbf{b}$.

Thus, previous works on RBMs usually consider the following notation for the gradients:

$$\frac{\partial}{\partial w_{ij}} \log p(\mathbf{v}) = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$

$$\frac{\partial}{\partial a_i} \log p(\mathbf{v}) = \langle v_i \rangle_{data} - \langle v_i \rangle_{model} \tag{B-33}$$

$$\frac{\partial}{\partial b_j} \log p(\mathbf{v}) = \langle h_j \rangle_{data} - \langle h_j \rangle_{model}$$

C

Contrastive Divergence via Kullback-Leibler Divergence

In Chapter 2, Subsection 2.5.1, we introduce an algorithm for training RBMs called Contrastice Divergence (CD). In order to develop this method, Hinton presented that training an RBM is the same as minimizing the Kullback-Leibler (KL) divergence between the RBM model and the distribution of the dataset [16]. In this appendix we will go through the equations that lead to the updating rule for each hyperparameter of an RBM during CD training.

From Information Theory, the KL divergence is a measure of how distant two distributions $P(x)$ and $Q(x)$ are [56]. It is denoted by the following formula:

$$D_{KL}(P(x)||Q(x)) = \sum_{x} P(x) \log \frac{P(x)}{Q(x)} \tag{C-1}$$

Taking into account the concept of KL divergence, the goal of training an RBM is to approximate the distribution of visible variables samples in the dataset $\mathbf{D}$ ($p_0(\mathbf{v})$) to the distribution of the actual RBM model $p_{rbm}(\mathbf{v})$. Therefore, we would like to minimize the following:

$$\begin{aligned} D_{KL}(p_{rbm}(\mathbf{v})||p_0(\mathbf{v})) &= \sum_{\mathbf{v}\in\mathbf{D}} p_0(\mathbf{v}) \log \frac{p_{rbm}(\mathbf{v})}{p_0(\mathbf{v})} \\ &= \sum_{\mathbf{v}\in\mathbf{D}} p_0(\mathbf{v}) \log p_{rbm}(\mathbf{v}) - \sum_{\mathbf{v}\in\mathbf{D}} p_0(\mathbf{v}) \log p_0(\mathbf{v}) \end{aligned} \tag{C-2}$$

As we will be minimizing the KL divergence by updating the parameters in the RBM $\Theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$, let us take the derivative over Θ of Equation C-2.

$$\begin{aligned}&\frac{\partial}{\partial\Theta}D_{KL}(p_{rbm}(\mathbf{v})||p_0(\mathbf{v}))\\&=\frac{\partial}{\partial\Theta}\Big(\sum_{\mathbf{v}\in\mathbf{D}}p_0(\mathbf{v})\log p_{rbm}(\mathbf{v})\Big)-\frac{\partial}{\partial\Theta}\Big(\sum_{\mathbf{v}\in\mathbf{D}}p_0(\mathbf{v})\log p_0(\mathbf{v})\Big)\end{aligned}\tag{C-3}$$

Now, as $p_0(\mathbf{v})$ is the distribution of the dataset, not depending on any parameter from the RBM, the derivative of the second term in Equation C-3 over Θ is zero. Moreover, we can exclude $\sum_{\mathbf{v}\in\mathbf{D}}p_0(\mathbf{v})$ out of the derivative for the first term.

$$\frac{\partial}{\partial\Theta}D_{KL}(p_{rbm}(\mathbf{v})||p_0(\mathbf{v}))=\sum_{\mathbf{v}\in\mathbf{D}}p_0(\mathbf{v})\frac{\partial}{\partial\Theta}\Big(\log p_{rbm}(\mathbf{v})\Big)\tag{C-4}$$

At this step, we can make use of the calculations performed in Appendix B for Section 2.5, expanding the derivative for $\log p_{rbm}(\mathbf{v})$, for the gradient over w_{ij}.

$$\begin{aligned}&\frac{\partial}{\partial w_{ij}}D_{KL}(p_{rbm}(\mathbf{v})||p_0(\mathbf{v}))\\&\quad=\sum_{\mathbf{v}\in\mathbf{D}}p_0(\mathbf{v})\Big(p_{rbm}(h_j=1|\mathbf{v})v_i-\sum_{\mathbf{m}}p(\mathbf{v'^m})p(h_j=1|\mathbf{v'^m})v_i\Big)\end{aligned}\tag{C-5}$$

Now, we can make use of the expected value notation as follows:

$$\begin{aligned}&\sum_{\mathbf{v}\in\mathbf{D}}p_0(\mathbf{v})\Big(p_{rbm}(h_j=1|\mathbf{v})v_i-\sum_{\mathbf{m}}p(\mathbf{v'^m}p(h_j=1|\mathbf{v'^m})v_i\Big)\\&=\Big\langle p_{rbm}(h_j=1|\mathbf{v})v_i\Big\rangle_{p_0}-\Big\langle\sum_{\mathbf{m}}p(\mathbf{v'^m})p(h_j=1|\mathbf{v'^m})v_i\Big\rangle_{p_0}\end{aligned}\tag{C-6}$$

However, as the second term goes over all possible values for $\mathbf{v}$, it does not vary according to p_0, making it a constant. That being said, we can remove the expectation value notation from it.

$$\begin{aligned}&\sum_{\mathbf{v}\in\mathbf{D}}p_0(\mathbf{v})\Big(p_{rbm}(h_j=1|\mathbf{v})v_i-\sum_{\mathbf{m}}p(\mathbf{v'^m}p(h_j=1|\mathbf{v'^m})v_i\Big)\\&=\Big\langle p_{rbm}(h_j=1|\mathbf{v})v_i\Big\rangle_{p_0}-\sum_{\mathbf{m}}p(\mathbf{v'^m})p(h_j=1|\mathbf{v'^m})v_i\end{aligned}\tag{C-7}$$

From Section 2.5, we have seen that the second term is actually intractable, as we are iterating over an exponential amount ($2^{dim(\mathbf{v})}$) of possible values for the visible vector. With that in mind, Hinton proposed a new method called Contrastive Divergence (CD) that has a different gradient [16; 57].

Instead of the learning gradient being the derivative of the KL divergence $D_{KL}(p_{rbm}||p_0)$, it is the gradient for a subtraction between two KL divergences:

$$D_{KL}(p_{rbm}||p_0) - D_{KL}(p_{rbm}||p_k), \tag{C-8}$$

where p_k is the distribution of the dataset being sampled by k Gibbs Sampling steps (see Section 2.4).

Following the same mathematical steps taken for the $\frac{\partial}{\partial w_{ij}} D_{KL}(p_{rbm}||p_0)$, we can arrive in the following gradient for this novel KL divergence:

$$\frac{\partial}{\partial w_{ij}} D_{KL}(p_{rbm}||p_k) = \Big\langle p_{rbm}(h_j = 1|\mathbf{v^k}) v_i^k \Big\rangle_{p_k} - \sum_{\mathbf{m}} p(\mathbf{v'^m}) p(h_j = 1|\mathbf{v'^m}) v_i \tag{C-9}$$

Notice that in Equation C-9 there is a k superscript on the $\mathbf{v}$ and v_i elements in the first term. This means that they are visible variable values sampled from k Gibbs Sampling steps.

Finally, we can subtract the two gradients as Hinton proposed. As the second terms in Equations C-7 and C-9 are equal, the subtraction cancels them out, yielding:

$$\begin{aligned} &\frac{\partial}{\partial w_{ij}} \Big(D_{KL}(p_{rbm}||p_0) - D_{KL}(p_{rbm}||p_k) \Big) \\ =&\Big\langle p_{rbm}(h_j = 1|\mathbf{v}) v_i \Big\rangle_{p_0} - \Big\langle p_{rbm}(h_j = 1|\mathbf{v^k}) v_i^k \Big\rangle_{p_k} \end{aligned} \tag{C-10}$$

Considering that during the learning process the hyperparameters will be updated for each sample $\mathbf{v}$, we can change them according to the following equations, where $p_{rbm}(\cdot) = p(\cdot)$ and ε is a learning rate.

$$
\begin{aligned}
w_{ij} &= w_{ij} + \varepsilon\Big(p(h_j = 1|\mathbf{v})v_i - p(h_j = 1|\mathbf{v^k})v_i^k\Big) \\
a_i &= a_i + \varepsilon\Big(v_i - v_i^k\Big) \\
b_j &= b_j + \varepsilon\Big(p(h_j = 1|\mathbf{v}) - p(h_j = 1|\mathbf{v^k}\Big)
\end{aligned}
\tag{C-11}
$$

D

QUBO Reformulation for non-binary values

Much effort has been made to develop reformulation techniques for general optimization problems to QUBOs, taking into account different types of constraints, variable encodings, and objective functions with a polynomial degree greater than two [58; 59; 60; 61; 62; 63]. It is worth mentioning that these conversion steps often lead to more variables added to the problem and loss of precision.

A brief variable encoding introduction

There are different variable mapping techniques that expand integer and continuous variables into binary ones, such as the Binary, Unary and Domain Wall [58]. Each encoding has its pros and cons, considering the number of binary variables required and the target QUBO solver that is going to be used.

In this appendix we will focus on the Binary Encoding technique, that was the one used for this research project. Equations D-1 and D-2 present the integer and continous Binary Encodings respectively.

$$\begin{aligned} &\text{Let } I \in [a,b] \subset \mathbb{Z}, n = \lceil \log_2(b-a)+1 \rceil \text{ and } \mathbf{y} \in \mathbb{B}^n \\ &I = a + (b - a - 2^{n-1} + 1)y_n + \sum_{i=1}^{n-1} 2^{i-1} y_i \end{aligned} \tag{D-1}$$

$$\begin{aligned} &\text{Given a value } \tau \text{ for the encoding error be less or equal to,} \\ &\text{Let } C \in [a,b] \subset \mathbb{R}, n \geq \log_2 \left[1 + \frac{b-a}{4\tau}\right] \text{ and } \mathbf{y} \in \mathbb{B}^n \\ &C = a + \frac{b-a}{2^n - 1} \sum_{j=1}^{n} 2^{j-1} y_j \end{aligned} \tag{D-2}$$

For easier understanding of how these encodings work, we will expand a variable $x \in [0, 3]$, for $x \in \mathbb{Z}$ and $x \in \mathbb{R}$.

Integer to Binary encoding

$$
\begin{aligned}
&a = 0, b = 3 \implies n = \lceil \log_2(3) + 1 \rceil = \lceil 2.58 \rceil = 3 \\
&I = 0 + (3 - 2^{3-1} + 1)y_3 + \sum_{i=1}^{3-1} 2^{i-1} y_i \\
&\quad = 0y_3 + 2^0 y_1 + 2y_2 = y_1 + 2y_2
\end{aligned}
\tag{D-3}
$$

As $[y_2, y_1] \in \mathbb{B}^2 : [0, 0] = 0;\ [0, 1] = 1;\ [1, 0] = 2;\ [1, 1] = 3$

Continuous to Binary encoding

Let us start with $\tau = 0.25$

$$a = 0, b = 3 \implies n \geq \log_2(4)$$

Using $n = 2$: (D-4)

$$C = 0 + \frac{3}{3} \sum_{1}^{2} 2^{i-1} y_i = y_1 + 2y_2$$

Using $\tau = 0.25$ we have the same representation achieved with the integer to binary encoding, which means that using this mapping we are not able to represent a non-integer number.

That being said, let us be more precise, with $\tau = 0.15$

$$a = 0, b = 3 \implies n \geq \log_2(1 + {}^{3}\!/_{0.6}) \implies n \geq 2.58$$

Using $n = 3$: (D-5)

$$C = 0 + \frac{3}{7} \sum_{1}^{3} 2^{i-1} y_i = \frac{3}{7}(y_1 + 2y_2 + 4y_3)$$

Now we have the following assignments:

y	C
[0,0,0]	0.0
[0,0,1]	0.43
[0,1,0]	0.86
[0,1,1]	1.29

y	C
[1,0,0]	1.71
[1,0,1]	2.14
[1,1,0]	2.57
[1,1,1]	3.0

Notice that there is a trade-off between encoding precision and the number of variables added to the model.

Thermal Dispatch to QUBO

Now we will present how these encodings work on an actual optimization problem, using a famous example from the energy sector. For simplicity, we will be using only integer variables and, due to the model that we are going to manipulate, a simple constraint mapping technique will have to be performed.

Consider the formulation for a simplified Thermal Dispatch problem, presented in Equation D-6. There are N thermal plants, each with a cost of operation per Megawatt ($\frac{\$}{MW}$) c_i and with a maximum generation capacity G_i^{max}. In order to minimize expenses, the goal is to find how much energy each thermal plant needs to generate in order to supply a demand D.

$$\begin{aligned} \min \quad & \sum_t \mathbf{c}'\mathbf{g} \\ s.t. & \sum \mathbf{g} = D \\ & g_i \in [0, G_i^{max}], \forall i \end{aligned} \tag{D-6}$$

For this example, there are three thermal plants, each generating $g1, g2, g3$ MW, to meet a demand of 6 MW. Additionally, the cost of operation for each plant is $c_1 = 1$, $c_2 = 2$ and $c_3 = 3$ $\frac{\$}{MW}$, while the maximum generation is $G_1^{max} = 2$, $G_2^{max} = 3$ and $G_3^{max} = 5$ MW. Thus, according to Equation D-6, this problem can be expressed as follows.

$$\begin{aligned} \min \quad & g_1 + 2g_2 + 3g_3 \\ s.t. \quad & g_1 + g_2 + g_3 = 6 \end{aligned} \tag{D-7}$$

As seen in Equation D-7, this problem is comprised of a constraint and non-binary variables which, for the sake of simplicity, are all integer ($g_i \in \mathbb{Z}$). Therefore, before mapping this model into a QUBO, it needs to have its variables discretized into binary ones and become unconstrained.

First, one can remove the constraint requires shifting it to the objective function with a penalty factor ρ. Thus, one can add the following term.

$$\rho(g_1 + g_2 + g_3 - 6)^2 \tag{D-8}$$

Notice that if the total generation is greater or less than the actual demand, having a large penalty factor ρ would leave the objective value far from the desired minimum. After this step, the problem can be represented as follows.

$$\min \quad g_1 + 2g_2 + 3g_3 + \rho(g_1 + g_2 + g_3 - 6)^2 \tag{D-9}$$

Subsequently, the integer variables g_i need to be encoded into binary ones. Adhering to Equation D-1, they are mapped in the following way.

$$\begin{aligned} &g_1 \in [0, 2], n = \lceil log_2(2) + 1 \rceil = 2 \\ &g_1 = g_{12} + g_{11} \end{aligned} \tag{D-10}$$

$$\begin{aligned} &g_2 \in [0, 3], n = \lceil log_2(3) + 1 \rceil = 3 \\ &g_2 = (0)g_{23} + g_{21} + 2g_{22} = 2g_{22} + g_{11} \end{aligned} \tag{D-11}$$

$$\begin{aligned} &g_3 \in [0, 5], n = \lceil log_2(5) + 1 \rceil = 4 \\ &g_3 = -2g_{34} + 4g_{33} + 2g_{32} + g_{31} \end{aligned} \tag{D-12}$$

Once this stage is finished, it is possible to rearrange the final equation.

$$\begin{aligned} \min \quad & g_{11} + g_{12} + 2g_{21} + 4g_{22} + 3g_{31} + 6g_{32} + 12g_{33} - 6g_{34} \\ & + \rho(g_{11} + g_{12} + g_{21} + 2g_{22} + g_{31} + 2g_{32} + 4g_{33} - 2g_{34} - 6)^2 \end{aligned} \tag{D-13}$$

Expanding the second term

$$
\begin{aligned}
\min \quad & 4g_{34}^2\rho - 16g_{33}g_{34}\rho - 8g_{32}g_{34}\rho - 4g_{31}g_{34}\rho - 8g_{22}g_{34}\rho - 4g_{21}g_{34}\rho - 4g_{12}g_{34}\rho \\
& - 4g_{11}g_{34}\rho + 24g_{34}\rho + 16g_{33}^2\rho + 16g_{32}g_{33}\rho + 8g_{31}g_{33}\rho + 16g_{22}g_{33}\rho + 8g_{21}g_{33}\rho \\
& + 8g_{12}g_{33}\rho + 8g_{11}g_{33}\rho - 48g_{33}\rho + 4g_{32}^2\rho + 4g_{31}g_{32}\rho + 8g_{22}g_{32}\rho + 4g_{21}g_{32}\rho \\
& + 4g_{12}g_{32}\rho + 4g_{11}g_{32}\rho - 24g_{32}\rho + g_{31}^2\rho + 4g_{22}g_{31}\rho + 2g_{21}g_{31}\rho + 2g_{12}g_{31}\rho \\
& + 2g_{11}g_{31}\rho - 12g_{31}\rho + 4g_{22}^2\rho + 4g_{21}g_{22}\rho + 4g_{12}g_{22}\rho + 4g_{11}g_{22}\rho - 24g_{22}\rho \\
& + g_{21}^2\rho + 2g_{12}g_{21}\rho + 2g_{11}g_{21}\rho - 12g_{21}\rho + g_{12}^2\rho + 2g_{11}g_{12}\rho - 12g_{12}\rho + g_{11}^2\rho \\
& - 12g_{11}\rho - 6g_{34} + 12g_{33} + 6g_{32} + 3g_{31} + 4g_{22} + 2g_{21} + g_{12} + g_{11} + 36\rho
\end{aligned}
\tag{D-14}
$$

As this reformulation has not introduced any higher-order terms, the quadratization step is not required. Thus, the $\mathbf{Q}$ matrix from Equation 4-2 can be constructed as follows.

$$
\begin{array}{c}
 \\ g_{11} \\ g_{12} \\ g_{21} \\ g_{22} \\ g_{23} \\ g_{31} \\ g_{32} \\ g_{33} \\ g_{34}
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
g_{11} & g_{12} & g_{21} & g_{22} & g_{23} & g_{31} & g_{32} & g_{33} & g_{34}
\end{array} \\
\left[\begin{array}{ccccccccc}
-11\rho+1 & 2\rho & 2\rho & 4\rho & 0 & 2\rho & 4\rho & 8\rho & -4\rho \\
2\rho & -11\rho+1 & 2\rho & 4\rho & 0 & 2\rho & 4\rho & 8\rho & -4\rho \\
2\rho & 2\rho & 11\rho-12 & 4\rho & 0 & 2\rho & 4\rho & 8\rho & -4\rho \\
4\rho & 4\rho & 4\rho & -20\rho+4 & 0 & 4\rho & 8\rho & 16\rho & -8\rho \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2\rho & 2\rho & 2\rho & 4\rho & 0 & -11\rho+3 & 4\rho & 8\rho & -4\rho \\
4\rho & 4\rho & 4\rho & 8\rho & 0 & 4\rho & -20\rho+6 & 16\rho & -8\rho \\
8\rho & 8\rho & 8\rho & 16\rho & 0 & 8\rho & 16\rho & -32\rho+12 & -16\rho \\
-4\rho & -4\rho & -4\rho & -8\rho & 0 & -4\rho & -8\rho & -16\rho & 28\rho-6
\end{array}\right]
\end{array}
\tag{D-15}
$$

Bibliography

[1] AJAGEKAR, A.; YOU, F.. **Quantum computing assisted deep learning for fault detection and diagnosis in industrial process systems**. Computers amp; Chemical Engineering, 143:107119, Dec. 2020.

[2] ABBAS, A.; AMBAINIS, A.; AUGUSTINO, B.; BÄRTSCHI, A.; BUHRMAN, H.; COFFRIN, C.; CORTIANA, G.; DUNJKO, V.; EGGER, D. J.; ELMEGREEN, B. G.; FRANCO, N.; FRATINI, F.; FULLER, B.; GACON, J.; GONCIULEA, C.; GRIBLING, S.; GUPTA, S.; HADFIELD, S.; HEESE, R.; KIRCHER, G.; KLEINERT, T.; KOCH, T.; KORPAS, G.; LENK, S.; MARECEK, J.; MARKOV, V.; MAZZOLA, G.; MENSA, S.; MOHSENI, N.; NANNICINI, G.; O'MEARA, C.; TAPIA, E. P.; POKUTTA, S.; PROISSL, M.; REBENTROST, P.; SAHIN, E.; SYMONS, B. C. B.; TORNOW, S.; VALLS, V.; WOERNER, S.; WOLF-BAUWENS, M. L.; YARD, J.; YARKONI, S.; ZECHIEL, D.; ZHUK, S. ; ZOUFAL, C.. **Challenges and opportunities in quantum optimization**. Nature Reviews Physics, Oct. 2024.

[3] CAO, Y.; ROMERO, J.; OLSON, J. P.; DEGROOTE, M.; JOHNSON, P. D.; KIEFEROVÁ, M.; KIVLICHAN, I. D.; MENKE, T.; PEROPADRE, B.; SAWAYA, N. P. ; OTHERS. **Quantum chemistry in the age of quantum computing**. Chemical reviews, 119(19):10856–10915, 2019.

[4] RIEFFEL, E. G.; ASANJAN, A. A.; ALAM, M. S.; ANAND, N.; BERNAL NEIRA, D. E.; BLOCK, S.; BRADY, L. T.; COTTON, S.; GONZALEZ IZQUIERDO, Z.; GRABBE, S.; GUSTAFSON, E.; HADFIELD, S.; LOTT, P. A.; MACIEJEWSKI, F. B.; MANDRÀ, S.; MARSHALL, J.; MOSSI, G.; BAUZA, H. M.; SAIED, J.; SURI, N.; VENTURELLI, D.; WANG, Z. ; BISWAS, R.. **Assessing and advancing the potential of quantum computing: A nasa case study**. Future Generation Computer Systems, 160:598–618, 2024.

[5] PRESKILL, J.. **Quantum computing in the nisq era and beyond**. Quantum, 2:79, Aug. 2018.

[6] CEREZO, M.; ARRASMITH, A.; BABBUSH, R.; BENJAMIN, S. C.; ENDO, S.; FUJII, K.; MCCLEAN, J. R.; MITARAI, K.; YUAN, X.; CINCIO, L. ;

COLES, P. J.. **Variational quantum algorithms**. Nature Reviews Physics, 3(9):625–644, Aug. 2021.

[7] CEREZO, M.; LAROCCA, M.; GARCÍA-MARTÍN, D.; DIAZ, N. L.; BRACCIA, P.; FONTANA, E.; RUDOLPH, M. S.; BERMEJO, P.; IJAZ, A.; THANASILP, S.; ANSCHUETZ, E. R. ; HOLMES, Z.. **Does provable absence of barren plateaus imply classical simulability? or, why we need to rethink variational quantum computing**, 2024.

[8] DAWID, A.; LECUN, Y.. **Introduction to latent variable energy-based models: a path toward autonomous machine intelligence**. Journal of Statistical Mechanics: Theory and Experiment, 2024(10):104011, Oct. 2024.

[9] KERGER, P.; MIYAZAKI, R.. **Quantum image denoising: a framework via boltzmann machines, qubo, and quantum annealing**. Frontiers in Computer Science, 5, 2023.

[10] AJAGEKAR, A.; YOU, F.. **Quantum computing based hybrid deep learning for fault diagnosis in electrical power systems**. Applied Energy, 303:117628, 2021.

[11] HINTON, G. E.. **A Practical Guide to Training Restricted Boltzmann Machines**, p. 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[12] RIPPER, P.; NETO, J.. **Niteq/qarbom.jl: v0.0.1**, Feb. 2025.

[13] DAWID, A.; LECUN, Y.. **Introduction to latent variable energy-based models: A path towards autonomous machine intelligence**, 2023.

[14] BISHOP, C. M.. **Pattern Recognition and Machine Learning (Information Science and Statistics)**. Springer-Verlag, Berlin, Heidelberg, 2006.

[15] FISCHER, A.; IGEL, C.. **Training restricted boltzmann machines: An introduction**. Pattern Recognition, 47(1):25–39, 2014.

[16] HINTON, G. E.. **Training products of experts by minimizing contrastive divergence**. Neural Computation, 14(8):1771–1800, 08 2002.

[17] TIELEMAN, T.. **Training restricted boltzmann machines using approximations to the likelihood gradient**. In: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ICML

'08, p. 1064–1071, New York, NY, USA, 2008. Association for Computing Machinery.

[18] LIAO, R.; KORNBLITH, S.; REN, M.; FLEET, D. J. ; HINTON, G.. **Gaussian-bernoulli rbms without tears**, 2022.

[19] TAYLOR, G. W.; HINTON, G. E. ; ROWEIS, S.. **Modeling human motion using binary latent variables**. In: PROCEEDINGS OF THE 20TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, NIPS'06, p. 1345–1352, Cambridge, MA, USA, 2006. MIT Press.

[20] LAROCHELLE, H.; BENGIO, Y.. **Classification using discriminative restricted boltzmann machines**. In: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ICML '08, p. 536–543, New York, NY, USA, 2008. Association for Computing Machinery.

[21] HINTON, G. E.; OSINDERO, S. ; TEH, Y.-W.. **A fast learning algorithm for deep belief nets**. Neural Comput., 18(7):1527–1554, July 2006.

[22] NAIR, V.; HINTON, G. E.. **3d object recognition with deep belief nets**. In: PROCEEDINGS OF THE 23RD INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, NIPS'09, p. 1339–1347, Red Hook, NY, USA, 2009. Curran Associates Inc.

[23] HUA, Y.; GUO, J. ; ZHAO, H.. **Deep belief networks and deep learning**. In: PROCEEDINGS OF 2015 INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING AND INTERNET OF THINGS, p. 1–4, 2015.

[24] MORO, L.; PRATI, E.. **Anomaly detection speed-up by quantum restricted boltzmann machines**. Commun. Phys., 6(1), Sept. 2023.

[25] NIELSEN, M. A.; CHUANG, I. L.. **Quantum computation and quantum information: 10th anniversary edition**, 2010.

[26] DIVINCENZO, D. P.. **The Physical Implementation of Quantum Computation**. Fortschritte der Physik, 48(9-11):771–783, 2000.

[27] AHARONOV, D.; VAN DAM, W.; KEMPE, J.; LANDAU, Z.; LLOYD, S. ; REGEV, O.. **Adiabatic quantum computation is equivalent to standard quantum computation**. SIAM Journal on Computing, 37(1):166–194, 2007.

[28] MCGEOCH, C. C.. **Adiabatic quantum computation and quantum annealing**. Synthesis lectures on quantum computing. Springer International Publishing, Cham, 2014.

[29] HOLMES, Z.; SHARMA, K.; CEREZO, M. ; COLES, P. J.. **Connecting ansatz expressibility to gradient magnitudes and barren plateaus**. PRX Quantum, 3(1), Jan. 2022.

[30] RAGONE, M.; BAKALOV, B. N.; SAUVAGE, F.; KEMPER, A. F.; MARRERO, C. O.; LAROCCA, M. ; CEREZO, M.. **A unified theory of barren plateaus for deep parametrized quantum circuits**, 2023.

[31] ZIMBORÁS, Z.; KOCZOR, B.; HOLMES, Z.; BORRELLI, E.-M.; GILYÉN, A.; HUANG, H.-Y.; CAI, Z.; ACÍN, A.; AOLITA, L.; BANCHI, L.; BRANDÃO, F. G. S. L.; CAVALCANTI, D.; CUBITT, T.; FILIPPOV, S. N.; GARCÍA-PÉREZ, G.; GOOLD, J.; KÁLMÁN, O.; KYOSEVA, E.; ROSSI, M. A. C.; SOKOLOV, B.; TAVERNELLI, I. ; MANISCALCO, S.. **Myths around quantum computation before full fault tolerance: What no-go theorems rule out and what they don't**, 2025.

[32] PERUZZO, A.; MCCLEAN, J.; SHADBOLT, P.; YUNG, M.-H.; ZHOU, X.-Q.; LOVE, P. J.; ASPURU-GUZIK, A. ; O'BRIEN, J. L.. **A variational eigenvalue solver on a photonic quantum processor**. Nature Communications, 5(1), July 2014.

[33] HELGAKER, T.; JØRGENSEN, P. ; OLSEN, J.. **Molecular electronic structure theory**, 2000.

[34] JORDAN, P.; WIGNER, E.. **Über das paulische äquivalenzverbot**, 1928.

[35] FARHI, E.; GOLDSTONE, J. ; GUTMANN, S.. **A quantum approximate optimization algorithm**, 2014.

[36] HADFIELD, S.; WANG, Z.; O'GORMAN, B.; RIEFFEL, E. G.; VENTURELLI, D. ; BISWAS, R.. **From the quantum approximate optimization algorithm to a quantum alternating operator ansatz**. Algorithms, 12(2):34, Feb. 2019.

[37] KIRKPATRICK, S.; GELATT, C. D. ; VECCHI, M. P.. **Optimization by simulated annealing**. Science, 220(4598):671–680, 1983.

[38] APOLLONI, B.; CARVALHO, C. ; DE FALCO, D.. **Quantum stochastic optimization**. Stochastic Processes and their Applications, 33(2):233–244, 1989.

[39] FINNILA, A. B.; GOMEZ, M. A.; SEBENIK, C.; STENSON, C. ; DOLL, J. D.. **Quantum annealing: A new method for minimizing multidimensional functions**. Chem. Phys. Lett., 219(5-6):343–348, Mar. 1994.

[40] RAY, P.; CHAKRABARTI, B. K. ; CHAKRABARTI, A.. **Sherrington-Kirkpatrick model in a transverse field: Absence of replica symmetry breaking due to quantum fluctuations**. Phys. Rev. B Condens. Matter, 39(16):11828–11832, June 1989.

[41] KADOWAKI, T.; NISHIMORI, H.. **Quantum annealing in the transverse ising model**. Phys. Rev. E, 58:5355–5363, Nov 1998.

[42] XAVIER, P. M.; RIPPER, P.; ANDRADE, T.; GARCIA, J. D.; MACULAN, N. ; NEIRA, D. E. B.. **Qubo.jl: A julia ecosystem for quadratic unconstrained binary optimization**, 2023.

[43] LUBIN, M.; DUNNING, I.. **Computing in Operations Research Using Julia**. INFORMS Journal on Computing, 27(2):238–248, 2015.

[44] XAVIER, P. M.; RIPPER, P.. **psrenergy/qiskitopt.jl: v0.3.0**, 2024.

[45] XAVIER, P. M.; RIPPER, P.. **Juliaqubo/pysa.jl: v0.3.2**, 2024.

[46] XAVIER, P. M.; RIPPER, P.. **`DWave.jl`**, 11 2022.

[47] DOWNS, J.; VOGEL, E.. **A plant-wide industrial process control problem**. Computers Chemical Engineering, 17(3):245–255, 1993. Industrial challenge problems in process control.

[48] RUSSELL, E. L.; CHIANG, L. H. ; BRAATZ, R. D.. **Fault detection in industrial processes using canonical variate analysis and dynamic principal component analysis**. Chemometrics and Intelligent Laboratory Systems, 51(1):81–93, 2000.

[49] RIETH, C. A.; AMSEL, B. D.; TRAN, R. ; COOK, M. B.. **Additional tennessee eastman process simulation data for anomaly detection evaluation**, 2017.

[50] XIE, D.; BAI, L.. **A hierarchical deep neural network for fault diagnosis on tennessee-eastman process**. In: 2015 IEEE 14TH INTERNA-

TIONAL CONFERENCE ON MACHINE LEARNING AND APPLICATIONS (ICMLA), p. 745–748, 2015.

[51] ZHANG, Z.; ZHAO, J.. **A deep belief network based fault diagnosis model for complex chemical processes**. Computers amp; Chemical Engineering, 107:395–407, Dec. 2017.

[52] CHEBEL-MORELLO, B.; MALINOWSKI, S. ; SENOUSSI, H.. **Feature selection for fault detection systems: application to the tennessee eastman process**. Applied Intelligence, 44(1):111–122, July 2015.

[53] YIN, S.; DING, S. X.; HAGHANI, A.; HAO, H. ; ZHANG, P.. **A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process**. Journal of Process Control, 22(9):1567–1581, Oct. 2012.

[54] INNES, M.. **Flux: Elegant machine learning with julia**. Journal of Open Source Software, 2018.

[55] KLYMKO, C.; SULLIVAN, B. D. ; HUMBLE, T. S.. **Adiabatic quantum programming: minor embedding with hard faults**. Quantum Information Processing, 13(3):709–729, Nov. 2013.

[56] COVER, T. M.; THOMAS, J. A.. **Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)**. Wiley-Interscience, USA, 2006.

[57] CARREIRA-PERPIÑÁN, M. A.; HINTON, G.. **On contrastive divergence learning**. In: Cowell, R. G.; Ghahramani, Z., editors, PROCEEDINGS OF THE TENTH INTERNATIONAL WORKSHOP ON ARTIFICIAL INTELLIGENCE AND STATISTICS, volumen R5 de **Proceedings of Machine Learning Research**, p. 33–40. PMLR, 06–08 Jan 2005. Reissued by PMLR on 30 March 2021.

[58] TAMURA, K.; SHIRAI, T.; KATSURA, H.; TANAKA, S. ; TOGAWA, N.. **Performance comparison of typical binary-integer encodings in an ising machine**. IEEE Access, 9:81032–81039, 2021.

[59] CHANCELLOR, N.. **Domain wall encoding of discrete variables for quantum annealing and qaoa**. Quantum Science and Technology, 4(4):045004, aug 2019.

[60] KARIMI, S.; RONAGH, P.. **Practical integer-to-binary mapping for quantum annealers**. Quantum Information Processing, 18(4), Feb. 2019.

[61] DATTANI, N.. **Quadratization in discrete optimization and quantum mechanics**, 2019.

[62] GLOVER, F.; KOCHENBERGER, G. ; DU, Y.. **Quantum bridge analytics i: a tutorial on formulating and using qubo models**. 4OR, 17(4):335–371, 2019.

[63] LUCAS, A.. **Ising formulations of many np problems**. Frontiers in Physics, 2, 2014.