Quantum Annealing for Optimization Problems

Bachelor's Thesis
Academic Year 2023–2024

Author

Daniel Yllana Santiago

Advisor

Guillermo Botella Juan

Alberto Antonio del Barrio Garcia

Computer Engineering

Computer Science Faculty

Universidad Complutense de Madrid

Abstract

Quantum Annealing for Optimization Problems

Quantum computers are a new way that allow us to solve complex optimization problems that are intractable for classical computers. This bachelor thesis explores the use of quantum computers, more specifically, adiabatic quantum computing to optimize complex problems focusing on the Quantum Unconstrained Binary Optimization (QUBO) model. Through this model, quantum annealers can be an effective way of solving optimization problems.

In this thesis we will study the viability of using quantum annealers to optimize neural networks as well as its precision and efficiency. In the work, we will develop a mathematical model to represent neural networks of any size, and with different activation functions in such a way that it can be used in quantum computer by using the QUBO model.

We will also compare this method to other classical methods and see the benefits and downsides of using quantum annealers for this specific optimization task.

Keywords

Quantum Computing, QUBO, Adiabatic Computing, DWave Quantum Annealers, Quantum Neural Networks, Optimization Problems, Constraint Model

Contents

List of figures

List of tables

Chapter 1

Introduction

1.1. Background

In the past decade, the use and development of neural networks has exponentially increased due to its potential for solving all types of tasks. Traditional methods for solving neural networks almost always rely on iterative techniques, mainly, gradient descent. Gradient descent is incredibly useful for a lot of tasks, but it has a lot of bottlenecks and can be computationally intensive and prone to getting stuck in local optima. Now that neural networks and dataset sizes are increasing rapidly, this limitations become more pronounced, and new methods of solving these neural networks could be beneficial.

To deal with these challenges, a new approach is proposed in which quantum computing is being used. Quantum Neural Networks offers a new way to optimize neural networks using the principles and advantages of quantum mechanics. By representing the neural network as a Quadratic Unconstrained Binary Optimization (QUBO) problem, QNNs can use quantum properties, such as parallelism and superposition, to explore a huge solution space simultaneously.

Adiabatic Quantum Computers, such as DWave's quantum annealers, can be used to implement QNNs and provide several advantages over classical computers. First, the parallelism property of quantum computers enable QNNs to explore an array of candidate solutions in parallel, leading to a faster convergence and eliminating the need to have an iterative methodology. Also, the properties of quantum computation allow for quantum entanglement and tunneling, which can promote finding global optimal solutions.

However, it is important to mention that current quantum computing hardware, while promising, is still very limited by a couple of factors such as, the numbers of qubits, and the coherence time of qubits. This can constrain the scale of the QNN and the size of the dataset. As more powerful and less error-prone quantum computers are developed, QNNs hold the potential to radically change the way neural networks are optimized, offering significant advantages over traditional methods.

1.2. Objectives

This thesis aims to optimize a quantum neural network using a quantum annealear, as well as generalizing the methods used to solve this problem.

1.2.1. Specific objectives

- Modeling the QNN to a Quadratic Constrained Binary Optimization (QCBO)
- Converting the QCBO model to a Higher-order Unconstrained Binary Optimization (HUBO)
- Converting the HUBO model to a Quadratic Unconstrained Binary Optimization (QUBO)
- Solving the problem with DWave's Hybrid Solvers and with DWave's QPU
- Decode the solution provided by the Solver
- Extract the solution to the QNN

1.3. Workflow

In this thesis we will model and optimize the parameters of a NN and try to find the optimal values while also minimizing the runtime.

Firstly, we will get a basic understanding of the mathematical basis of Quantum Computing and Quantum Annealing (chapter 2) to get a grasp of how they can be used to solve optimization problems. We will also explain the mathematical tools to convert problems onto models that can be mapped onto a quantum computer (chapter 3).

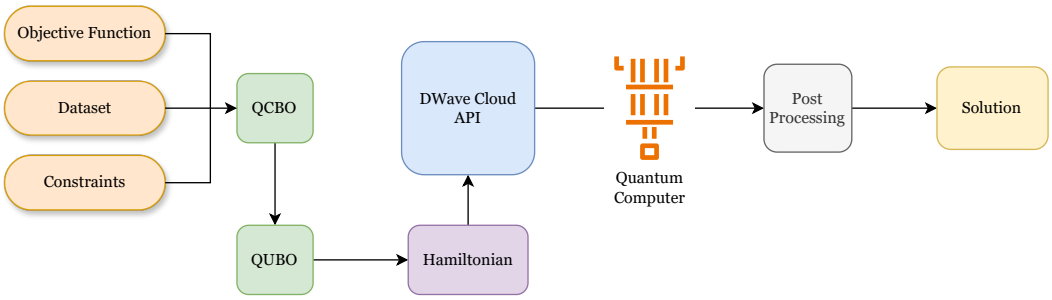


Figure 1.1: Workflow Diagram

In the second part, we will convert a traditional neural network to model of the Hamiltonian (chapter 4) as a QCBO (Quadratic Constraint Binary Optimization), which will be then transformed to a QUBO (Quadratic Unconstrained Binary Optimization) that can be mapped to a Quantum Computer as shown in (Fig. 1.1).

Lastly, we will use the conversion process to conduct three study cases:

1. **Study Case 1-** Optimize a neural network two classify points onto two possible types using different network types, parameters, and activation functions.
2. **Study Case 2-** Binary classification of images (handwritten numbers).
3. **Study Case 3-** Benchmarking classical algorithms and quantum annealing for a classification task. Comparing the runtime and solution correctness with different tasks, number of variables, network types, and parameters using a BNN.

Chapter 2

Adiabatic Quantum Computing

2.1. Quantum Computing

Quantum Computing is an emergent technology that combines classical computing with the fundamentals of quantum mechanics. It has the potential to solve problems that are extremely difficult to solve on classical computers. The main difference between quantum and classical computing, is the basic unit of information that is used, while classical computers use bits, quantum computers use quantum bits (Qubits).

Unlike classical bits, which can exist in states of either 0 or 1, qubits can exist in superposition, representing a probabilistic combination of both states simultaneously. This property enables quantum computers to process vast amounts of information in parallel, exponentially expanding their computational power compared to classical counterparts.

2.2. Qubit

A qubit is represented by a two-state quantum system, this is one of the fundamental systems that display the peculiarities of quantum mechanics. An example would be the spin of an electron where the two states can be taken as spin-up or spin-down. In a classical system, a bit has to be represented by either one state or the other, however, quantum mechanics allows the qubit to be in a superposition of multiple states simultaneously.

The general state of a qubit is represented by its two basis vectors. These vectors

are usually denoted by

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

and they are written in the conventional Dirac or bra-ket notation. The two orthonormal basis states $\{|0\rangle, |1\rangle\}$ together are called the computational basis.

A single qubit $|\psi\rangle$ can be described by a two-dimensional vector, more specifically, a linear combination of its two basis vectors $|0\rangle, |1\rangle$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.2}$$

where α and β are both complex numbers that represent the probability amplitudes. When the qubit is measured in its standard basis, the probability of the state $|0\rangle$ is $|\alpha|^2$ and the probability of $|1\rangle$ is $|\beta|^2$. Since the squares of the amplitudes equate to probabilities, it must be that α and β follow

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.3}$$

2.3. Superposition

Quantum superposition is one of the fundamental principles of quantum mechanics, where a quantum system can exist in multiple states simultaneously. This state can be represented as the sum of two or more different states. A qubit in a superposition state can be represented as a point on the Bloch Sphere (Fig. 2.1).

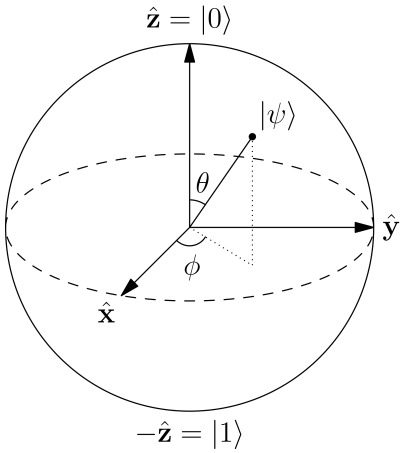


Figure 2.1: Position of the basis states $|0\rangle, |1\rangle$ and representation of the state $|\psi\rangle$ on the Bloch Sphere.

The Bloch sphere, associates every point with spherical coordinates (θ, ϕ) to a quantum state in the following way

$$|\psi\rangle \;=\; \cos(\theta/2)\,|0\rangle + e^{i\phi}\sin(\theta/2)\,|1\rangle \tag{2.4}$$

Having established the foundational principles of quantum computing, it is crucial to explore how these principles are applied in various quantum computing paradigms. While the discussion thus far has focused on the general attributes that differentiate quantum computing from classical computing, particularly the use of qubits and their ability to exist in superposition states, we now turn our attention to specific methodologies within quantum computing. One such approach is Adiabatic Quantum Computing (AQC), which leverages the principles of quantum mechanics differently compared to the quantum circuit model. AQC relies on the adiabatic theorem to gradually evolve a quantum system's Hamiltonian to find solutions to complex optimization problems. This paradigm, along with the related concept of Quantum Annealing, offers powerful tools for solving a variety of computational challenges by navigating through energy landscapes using quantum phenomena such as tunneling. In the following sections, we will delve into the mechanics of Adiabatic Quantum Computing, the adiabatic theorem, and the practical implementations of these concepts, providing a comprehensive understanding of their applications and implications in the field of quantum computing.

2.4. Adiabatic Quantum Computing

Adiabatic Quantum Computing is a quantum computing paradigm, distinct from quantum circuits, where the initial state is slowly evolved by a Hamiltonian dependent on time until a final state. It relies on the adiabatic theorem, which guarantees that if a given perturbation is acting on it slowly enough, the fundamental state of the initial Hamiltonian will evolve to the fundamental state of the final Hamiltonian, after a long enough time period.

The biggest difference with a circuit with quantum gates is that the intermediate states through which the system evolves are unknown. Adiabatic computing is useful to tackle problems in which the fundamental state of the final Hamiltonian contains the solution to the problem, for example, minimization or optimization problems.

2.5. Adiabatic Theorem

Lets consider a quantum system with a Hamiltonian that has evolved from H_0 to H_f in a period of time T. In such a way that we have:

$$\begin{aligned} H(t=0) &= H_0 \\ H(t=T) &= H_t \\ H(t) &= (1-\frac{t}{T})H_f + \frac{t}{T}H_0 \end{aligned} \tag{2.5}$$

Now we call $|\phi_n(t)\rangle$ to the eigenstate corresponding to the Hamiltonian $H(t)$:

$$H(t)\,|\phi_n(t)\rangle = E_n(t)\,|\phi_n(t)\rangle \tag{2.6}$$

Considering $E_1(t) < E_2(t) < \ldots < E_n(t)$ such that no eigenvalues are repeated. If the system at $t=0$ is in state $|\psi(t=0)\rangle = |\phi_n(0)\rangle$ for some n and the Hamiltonian has evolved from H_0 to H_f over a period T, then by the adiabatic theorem, in $t=T$, the state of the system is $|\psi(t=T)\rangle \cong |\phi_n(T)\rangle$.

If the state of the system starts off in the n-th lowest eigenvalue with state $|\phi_n(0)\rangle$ which corresponds with Hamiltonian H_0, then at time T the system will remain in the state with n-th lowest eigenvalue corresponding with the final Hamiltonian H_f if the evolution was performed slowly enough.

2.5.1. Algorithm

The most basic operating scheme of adiabatic quantum computing can be summarized in the following steps:

1. Choose a final Hamiltonian H_f that represents the problem to be solved. Normally this will be a binary optimization problem, such as QUBO, that are easily mapped to a Hamiltonian.

2. An initial Hamiltonian H_0 with an easily prepared fundamental state.

3. A intermediate Hamiltonian $H(s)$ that slowly evolves with s, and such that $H(0) = H_0$ and $H(T) = H_f$. In the trivial case, $H(s)$ is the linear interpolation $H(s) = (1-s)H_0 + sH_f$.

4. The parameter s is chosen very carefully as a function of time. Normally, this parameter is simply $s = t/T$, where T is the final time.

5. The system is evolved under these conditions over a time T.

2.6. Quantum Annealing

The required conditions for adiabatic computing, such as zero temperature, are extremely difficult to physically obtain, which is why there are new algorithms based on the adiabatic concepts in which the conditions are relaxed in exchange of losing the universality of adiabatic computing.

Quantum Annealing is one of these procedures that allow us to find the minimum of a Hamiltonian by utilizing Quantum Tunneling. Although the ideal is for the system to evolve around the minimum energy, in some cases the probability that it will remain in the minimum is low, however, these solutions can still be relatively close the the minimum.

Quantum Annealing is primarily used to solve optimization or sampling problems, where the problem is modeled with a Hamiltonian from which we want to find the lowest energy state, which approximates to the optimal solution to the initial problem.

2.6.1. Quantum Tunneling

Quantum Tunneling plays a crucial role in traversing the energy barriers and reaching the lowest energy state of a Hamiltonian. Its a phenomenon that comes from the principles of quantum mechanics, and it allows particles to pass through classically impassable barriers by exploiting their wave-like nature.

In adiabatic computing, the Hamiltonian represents the energy landscape of the problem being solved. The lowest energy state of a Hamiltonian corresponds to the optimal solution of a problem. However, in more complex optimization tasks, the system may encounter energy barriers that hinders its transition to the lowest energy state as can be seen in (Fig. 2.2).

Quantum tunneling enables the system to overcome the barriers by allowing the wave function of the quantum system to propagate through energy barriers, rather than getting stuck in local minima. As a result, the adiabatic evolution of a quantum system can navigate through the energy landscape more efficiently,

compared to classical methods.

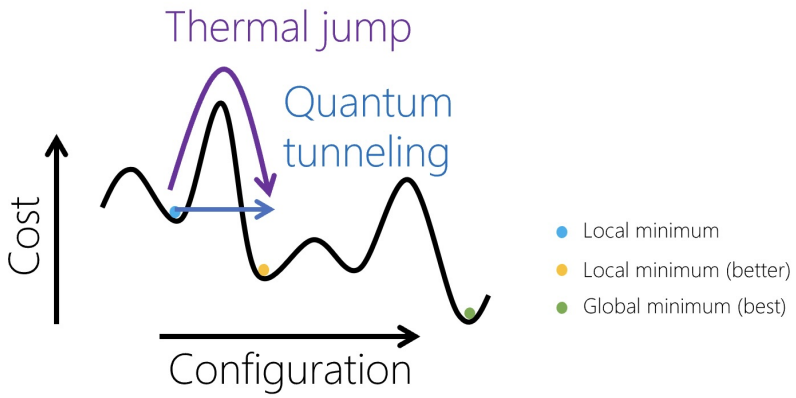


Figure 2.2: Visual representation of finding the minimum of a function with and without the quantum tunneling phenomena.

Chapter 3

Mathematical Models for Optimization Problems

In order to take advantage of the benefits of the adiabatic theorem, we need to be able to encode a given problem into a the final Hamiltonian. The main step in this process, is to convert an optimization problem to some sort of Binary Optimization problem, since this subset of problems can be easily converted to a Hamiltonian.

There are different types of Binary Optimization Problems or BQM, each with different characteristics that are beneficial for different types of optimization tasks. The most used model is the QUBO model.

Most optimization problems can be expressed the following way

$$min_x \ f(x) \ = \ x^T Q x \tag{3.1}$$

where x is a vector of unknown binary variables, and Q is a known square matrix that contains the constants of the problem. There are two different ways of building the Q matrix.

Symmetric Matrix

$$\begin{pmatrix} q_{00} & q_{01} & \cdots & q_{0n-1} \\ q_{10} & q_{11} & \cdots & q_{1n-1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n-1\ 0} & q_{n-1\ 1} & \cdots & q_{n-1n-1} \end{pmatrix} \rightarrow \begin{pmatrix} q_{00} & \frac{q_{01}+q_{10}}{2} & \cdots & \frac{q_{0n-1}+q_{n-1\ 0}}{2} \\ \frac{q_{01}+q_{10}}{2} & q_{11} & \cdots & \frac{q_{1n-1}+q_{n-1\ 1}}{2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{q_{0n-1}+q_{n-1\ 0}}{2} & \frac{q_{1n-1}+q_{n-1\ 1}}{2} & \cdots & q_{n-1n-1} \end{pmatrix}$$

Upper Triangular Matrix

$$\begin{pmatrix} q_{00} & q_{01} & \cdots & q_{0\ n-1} \\ q_{10} & q_{11} & \cdots & q_{1\ n-1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n-1\ 0} & q_{n-1\ 1} & \cdots & q_{n-1 n-1} \end{pmatrix} \rightarrow \begin{pmatrix} q_{00} & q_{01}+q_{10} & \cdots & q_{0\ n-1}+q_{n-1\ 0} \\ 0 & q_{11} & \cdots & q_{1\ n-1}+q_{n-1\ 1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-1\ n-1} \end{pmatrix}$$

3.1. QUBO

The QUBO model, or Quadratic Unconstrained Binary Optimization [6] models can be codified as a matrix Q, where the main diagonal represents the weights of the linear terms of the function to be minimized, and the rest of the values represent the weights of the quadratic terms of the function.

The QUBO model can also be formulate the following way

$$f(x) \ = \ \sum_i Q_{i,i}x_i + \sum_{i<j} Q_{i,j}x_ix_j \tag{3.2}$$

Let's see an example, suppose we have the following function to be minimized

$$minf(x_3, x_2, x_1, x_0) \ = \ -7x_0 - 3x_1 - 10x_2 - x_3 - 5x_0x_1 + 10x_0x_2 + 3x_2x_3$$

since the variables of the vector x are binary, we have $x_i = x^2$, then

$$minf(x_3, x_2, x_1, x_0) = -7x_0^2 - 3x_1^2 - 10x_2^2 - x_3^2 - 5x_0x_1 + 10x_0x_2 + 3x_2x_3$$

Now, we can express the function as a matrix

$$minf(x_3, x_2, x_1, x_0) \ = \ \underbrace{\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \end{pmatrix}}_{x^T} \underbrace{\begin{pmatrix} -7 & -5 & 10 & 0 \\ -5 & -3 & 0 & 0 \\ 10 & 0 & -10 & 3 \\ 0 & 0 & 3 & -1 \end{pmatrix}}_{Q} \underbrace{\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x}$$

3.2. QCBO

The problem with the QUBO formulation, is that most optimization problems not only have a function to minimize but they also have some sort of constraints that the x vector must follow. The most common solution to adding constraints is the penalty model, in which the constraints are manipulated and added to the minimization function.

The penalty model penalizes the solutions that do not comply with the constraints.

Continuing with the previous example, if we want to add the constraint

$$x_0 + x_3 \ = \ 1$$

first, we make the equation one-sided, and then we square it, resulting in

$$(x_0 + x_3 - 1)^2 \ = \ 0$$

finally we add the new equation to the objective function

$$minf(x_3x_2x_1x_0) + P * (x_0 + x_3 - 1)^2$$

where P is the penalty coefficient, the value of P has to be big enough to have an impact on the energy levels of the system if the constraint is not meet.

Chapter 4

Neural Networks on Quantum Annealers

The proposed algorithm focuses on feed forward quantized neural networks (QNN). This kind of neural network has a uni-directional flow, from the input layer, through the hidden layers, and finally to the output layer, meaning that it can be represented with fewer equations than more complex networks such as Transformers. The input data to the network has to be represented with qubits, and since there is a hardware limitation on the number of qubits that modern quantum computers have, the input data will be quantized to use the least amount of qubits possible to represent it.

The QNN is composed of two parts, the loss function, and feed forward network. The loss function is used to calculate how the current predictions of the network compare to the truth values. The feed forward network is comprised of three components, weights, biases, and activation function.

Current methods for training feed forward neural networks involve a back-propagation algorithm, that using the loss function, modifies the weights and biases of the networks to achieve a better prediction, however, we can use quantum annealers instead of back-propagation to find the optimal solutions to the network.

4.1. Training Problem

Suppose that we have multilayer QNN for which we want to find the optimal parameters θ. The training problem is described by

$$\min_{\theta} \ \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 \tag{4.1}$$

where f is the QNN, N is the size of the sampleset, and (x_i, y_i) is the i-th data sample.
The output of the network is given by the function f and is computed by the composition of each layer

$$f(x) \ = \ f^{(L)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x) \tag{4.2}$$

where f^k is the function for the k-layer, and has the form

$$f^{(k)}(x) \ = \ g\big(W^{(k)}x + b^{(k)}\big) \tag{4.3}$$

This contains linear transformation with $b^{(k)}$ and $W^{(k)}$, the biases and weights of the layer , and nonlinear transformation with g, the activation function of the layer.

Because the function of the network is a composition of its layer's function, this results in a high-order nonlinear loss function in respect with θ rather than a quadratic loss function. This means that it cannot be trivially mapped to a QUBO.

The conversion process is shown in (Fig. 4.1), where we first take as input the data and the neural network topology, which is converted to a QCBO problem and then a QUBO problem. After the QUBO problem has been created and the Q matrix is generated, it is executed on a Adiabatic Quantum Computer. The output of the quantum computer is a binary solution, that has to be decoded to extract the network parameters to form the neural network.

In order to convert the problem to a QCBO, there are two techniques to achieve this, rewrite the network topology as constraints functions, and to represent in binary the variables.

Once the problem has been converted to a QCBO, we then convert it to a QUBO that can be mapped to a Hamiltonian. To do this, we first use the penalty method to deal with the constraints, and then use Rosenberg order reduction method to make sure there are no high-order terms.

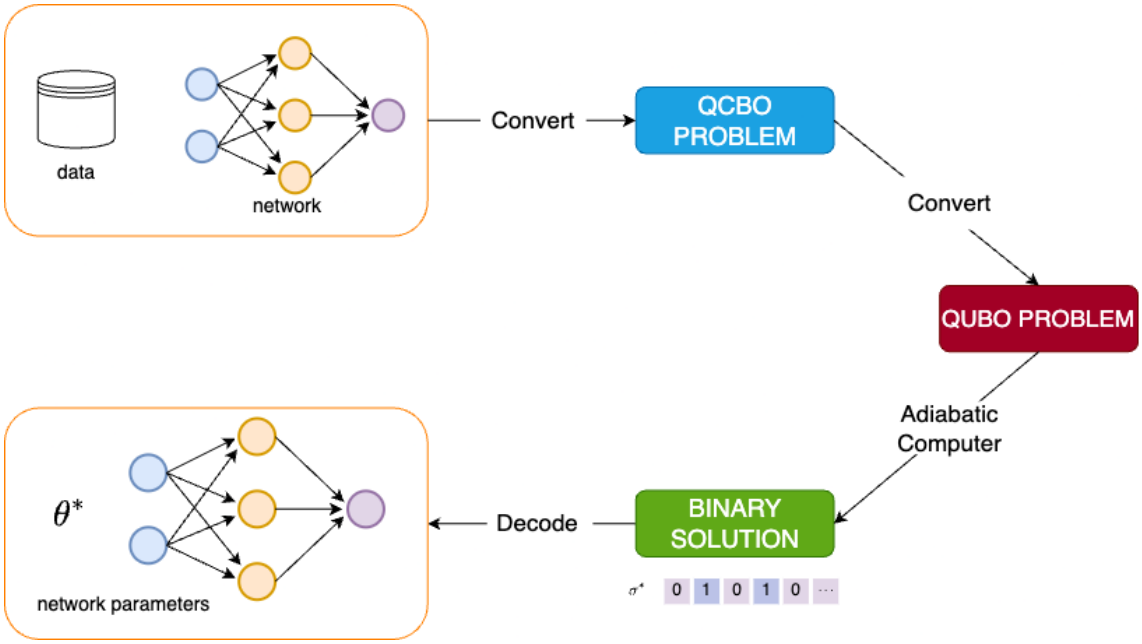


Figure 4.1: Workflow of the training algorithm.

4.2. Network Loss Function

The loss function that we will be using is one of the most common ones, Mean Squared Error (MSE):

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.4}$$

where $\hat{y}_i$ is a decision variable representing the output of the network for a given x_i input data. When minimizing (4.4), we have to ensure that the value of $\hat{y}_i$ matches to that of $f(x_i)$, therefore, some constraints are needed to ensure this behaviour of the network topology.

All of the variables needed throughout this process are shown in (Table 4.1).

Variable	Description
N	size of sampleset
L	number of layers
H	number of nodes per hidden layer
n	number of nodes in input layer
m	number of nodes in output layer
x	quantized input
$\hat{y}$	quantized predicted output
$W^{(k)}$	weights of the k-th layer
$b^{(k)}$	biases of the k-th layer
$s^{(k)}$	pre-activation value of k-th layer
$a^{(k)}$	post-activation value of k-th layer

Table 4.1: Symbols of variables in QNN

4.3. Network Topology Constraints

In each layer, there are two operations, the linear transformation and the activation function. To represent the topology, the two operations are converted to equality constraints. The linear transformation can be written as:

$$W^{(k)}a^{(k-1)} + b^k \;=\; s^{(k)} \tag{4.5}$$

there are a lot of different activation functions [10], we will use the *sign* activation function which has the following behaviour:

$$sign(x) \;=\; \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases} \tag{4.6}$$

To represent the behaviour of the *sign* activation function, two constraint functions are needed:

$$a^{(k)} \odot s^{(k)} \;=\; r^{(k)} \tag{4.7a}$$

$$a^{(k)} + 2r^{(k)} \;\geq\; 1 \tag{4.7b}$$

where $\odot$ represents element-wise multiplication, $a^{(k)} \in \{-1, 1\}^H$, $s^{(k)} \in \mathbb{Z}^H$, and $r^{(k)} \in \mathbb{N}^H$ is an auxiliary variable.

The constraint (4.7a) guarantees that $a^{(k)}$ and $s^{(k)}$ have the same sign, because $r^{(k)}$

is non-negative, and the value of $r^{(k)}$ will be equal to the absolute value of $s^{(k)}$. The inequality constraint (4.7b) guarantees $a^{(k)} = +1$ when $s^{(k)} = 0$. These two constraints together guarantee that $a^{(k)} = sign(s^{(k)})$.

In order for the penalty method to work, the inequality constraint (4.7b) must be converted to a equality constraint by introducing an auxiliary variable:

$$a^{(k)} + 2r^{(k)} = 1 + t^{(k)} \tag{4.8}$$

The equality constraints (4.5), (4.7a), (4.8) need to be defined for all layers except the output layer, except for (4.5) that will also need to be defined for the last layer as seen in (Fig. 4.2).

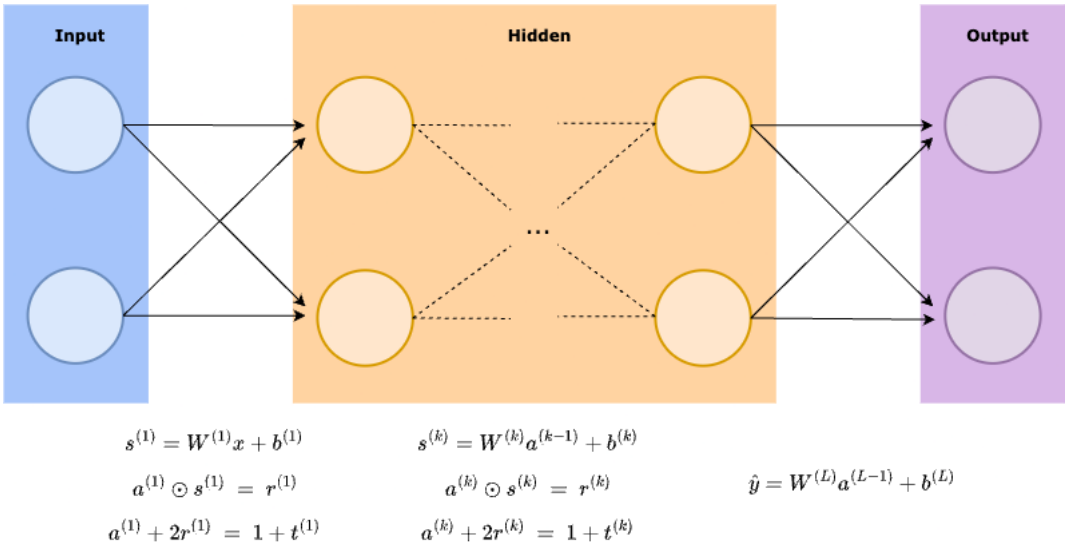


Figure 4.2: **Network Topology of QNN**

These equality constraints have to be defined for all samples in the sampleset, for example, the constraint (4.5) becomes:

$$W^{(k)}a_i^{(k-1)} + b^k = s_i^{(k)} \quad \forall i \tag{4.9}$$

Finally, the QCBO model for the neural network is:

$$
\begin{aligned}
&\text{Minimize} \\
&\qquad \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \\
&\text{subject to} \\
&\qquad W^{(k)}a^{(k-1)} + b^k \;=\; s^{(k)} \\
&\qquad a^{(k)} \odot s^{(k)} \;=\; r^{(k)} \\
&\qquad a^{(k)} + 2r^{(k)} \;=\; 1 + t^{(k)}
\end{aligned}
\tag{4.10}
$$

The set of variables that need to be optimized to minimize (4.4) is $\{W^{(k)}, b^{(k)}, s_i^{(k)}, r_i^{(k)}, t_i^{(k)}, a_i^{(k)}, \hat{y}_i \mid k = 1, 2, \ldots, L \text{ and } i = 1, 2, \ldots, N\}$.

To map these variables to qubits, they need to be represented as binary variables, using the binary encoding protocol for decimal numbers. For example, variable $s^{(k)}$ is encoded by:

$$
s^{(1)} \;=\; \sum_j 2^j \sigma_s^{(1)}(j) \tag{4.11}
$$

where $\sigma_s^{(1)}(j) \in \{0,1\}^H$ is a binary variable that will be mapped to a qubit. Through binary representation, $W^{(1)}$, $b^{(1)}$ and $s^{(1)}$ are all encoded as quantized values. Then the constraint $W^{(1)}x_i + b^{(1)} = s_i^{(1)}$ requires that x_i must be quantized itself, therefore, x_i is quantized as an integer by rounding with a value range of $[-2^B, 2^B]$ where B is the number of bits that will be used to represent input data.

4.4. Converting to QUBO

Now that we have all of restrictions necessary to represent the problem in QCBO as in (4.10), now we have to convert it to a QUBO by removing the constraints. To do that, the main technique is removing the constraints using the penalty function method as explained in algorithm (1), as well as an order reduction method in case there are higher order polynomials.

Algorithm 1 Penalty Function Method

Input: loss function $\mathcal{L}_{MSE}$, the set of all constraints to be removed Ω, and P penalty coefficient

1: **for** all constraints ω in Ω **do**

2: $\quad \mathcal{L} \leftarrow \mathcal{L}_{penalty} + P \cdot \omega^2$

3: **end for**

4: $\mathcal{L}_{penalty} \leftarrow \mathcal{L}_{penalty} + \mathcal{L}_{MSE}$

Output: $\mathcal{L}_{penalty}$ quadratic loss function with the constraints embedded on it

Using the penalty function method to eliminate the constraints, the loss function (4.4) is rewritten as:

$$\begin{aligned}
\mathcal{L}_{penalty} &= \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \\
&+ P\sum_{i=1}^{N}\left\|W^{(1)}x_i + b^{(1)} - s_i^{(1)}\right\|^2 \\
&+ P\sum_{i=1}^{N}\sum_{k=2}^{L-1}\left\|W^{(k)}a_i^{(k-1)} + b^{(k)} - s_i^{(k)}\right\|^2 \\
&+ P\sum_{i=1}^{N}\left\|W^{(L)}a_i^{(L-1)} + b^{(L)} - \hat{y}_i\right\|^2 \\
&+ P\sum_{i=1}^{N}\sum_{k=1}^{L-1}\left\|a_i^{(k)} \odot s_i^{(k)} - r_i^{(k)}\right\|^2 \\
&+ P\sum_{i=1}^{N}\sum_{k=1}^{L-1}\left\|a_i^{(k)} + 2r_i^{(k)} - 1 - t_i^{(k)}\right\|^2
\end{aligned} \tag{4.12}$$

where P is the penalty coefficient, with a large enough value of P the cost of breaking the constraints is higher than that of of increasing the accuracy. However, the loss function (6.5) has an order greater than 2, and is therefore a higher-order unconstrained binary optimization problem.

Now using the second technique, we reduce the order of the loss function by using the Rosenberg polynomial [7]. The form of the Rosenberg polynomial is:

$$h(v_1, v_2, w) = 3w + v_1v_2 - 2v_1w - 2u_2w \tag{4.13}$$

with $v_1, v_2, w \in \{0, 1\}$, the polynomial has two properties that are useful for

reducing the order of a function:

1. $h(v_1, v_2, w) \geq 0 \quad \forall v_1, v_2, w$
2. $h(v_1, v_2, w) = 0 \iff w = v_1 v_2$

The properties result in $h(v_1, v_2, w) = 0$ when $w = v_1 v_2$ and $h(v_1, v_2, w) > 0$ when $w \neq v_1 v_2$, so h takes its minimum value when $w = v_1 v_2$. Due to this two properties, we can substitute a multiplication of two variables $v_1 v_2$ with the auxiliary variable w, and then adding the polynomial h to the loss function with a large coefficient following the idea of the penalty function method. Same as with the penalty method, if the coefficient is big enough, $w = v_1 v_2$ will hold.

As a result, a second order term $v_1 v_2$ can be replaced by a linear term w. This method applied iteratively reduces the order of the polynomial to a quadratic order, therefore converting it to a QUBO, at the expense of adding more auxiliary variables, and more terms to the loss function.

QUBO Verification

To verify whether QUBO's solution is identical to QCBO's, we use the verification method [9] shown in (Fig. 4.3). The solution identity is a strong evidence of the correctness of the conversion process from QCBO to QUBO. The conversion process is supposed to eliminate the constraints without altering the problem's optimality.

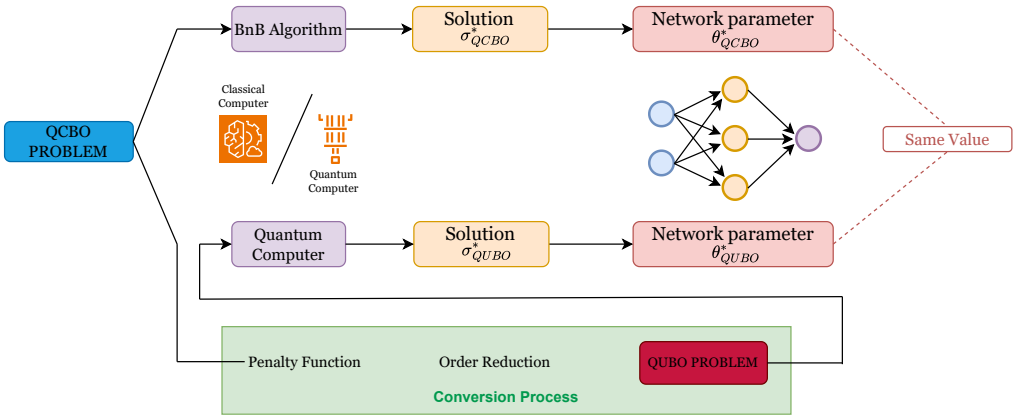


Figure 4.3: Verification Strategy

4.5. Binary Neural Networks

A Binary Neural Network (BNN) [11] is a type of neural network where the weights and features are 1-bit values (often they can also be $\{-1, 1\}$) in all of the hidden layers. The purpose of BNN's is to reduce the matrix computation costs by applying XNOR operations [8] which leads to a smaller model with less storage needed as well as faster layer computations.

Just like NNs, BNNs follow equation (4.5), however, the weights and biases are binary values. In this case, we will be using $\{-1, 1\}$ as the binary values since they are easily mapped to a QPU.

If we remove the bias from (4.5), we are left with:

$$W^{(k)} a^{(k-1)} \;=\; s^{(k)} \tag{4.14}$$

and since the input data, and the weights are binary variables, we have that the output of each layer will also be a binary value. Because of this, an activation function is not need, thus, simplifying the neural network and removing restrictions (4.7a) and (4.8).

Finally, we can now rewrite equation (6.5) for a BNN:

$$\begin{aligned} \mathcal{L}_{penalty\ BQNN} \;=\; & \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \\ & + P\sum_{i=1}^{N}\left\|W^{(1)}x_i - s_i^{(1)}\right\|^2 \\ & + P\sum_{i=1}^{N}\sum_{k=2}^{L-1}\left\|W^{(k)}a_i^{(k-1)} - s_i^{(k)}\right\|^2 \\ & + P\sum_{i=1}^{N}\left\|W^{(L)}a_i^{(L-1)} - \hat{y}_i\right\|^2 \end{aligned} \tag{4.15}$$

Not only have we reduced the number of equations, we have also greatly reduced the number of variables, and the qubits necessary to represent each variable, since now, they are all SPIN variables that are directly mapped to qubits.

4.6. Training Algorithm

The overall idea or workflow to optimize a problem on a Quantum Annealer is shown in algorithm (2). The computational complexity of converting the problem to a model that can be mapped to the Quantum Annealer and the parameter decoding of the solution returned by the Quantum Annealer is low, even for large problems. This, coupled with the speed of quantum anealing means that the learning algorithm can be very fast.

Algorithm 2 Training for Quantum Neural Networks

Input: data points x

1: Convert the NN into a QCBO problem ▷ CPU
2: Transform the QCBO into a QUBO problem using (6.5)
3: Create Hamiltonian
4: Embedded QUBO onto Quantum Computer Topology
5: Solve on Quantum Annealer ▷ QPU
6: Decode parameters ▷ CPU

Output: optimal parameters for network θ

Chapter 5

DWave's Quantum Annealers

5.1. Solvers

Dwave provides multiple solvers made for different types of optimization problems, therefore, easing the development of algorithms and applicability of the systems. Dwave has two main types of solvers, its direct QPU solvers, and its Hybrid-solvers.

The problems that are directly executed on quantum annealers are in the QUBO model. However, hybrid solvers accept other types of quadratic models, such as constrained or unconstrained, with not only binary variables, but also real, integer, or discrete variables. These solvers closely combine classical and quantum computing to solve large and complex problems that are very computationally costly on classical computers, but not yet possible using only quantum computers. This approach uses classical computing for tasks such as pre and post processing of the problem, while using the quantum computers for the main optimization task. They are designed with classical algorithms that allocate subproblems to the quantum computer where it benefits most.

The different types of Hybrid solvers available on Dwave are:

- **Constrained Quadratic Model (CQM)** This extends the QUBO formulation by allowing constraints. This allows the user to create problems with linear and quadratic constraints, which the hybrid solver will satisfy during the optimization problem. This solver, address the need for penalty function method that is commonly used to create QUBO models. These model also allows for non-binary variables such as real and integer numbers.

- **Discrete Quadratic Model (DQM)** It is typically used to represent problems with sever distinct options instead of continuous variables like integer numbers.

5.2. Quantum Processing Unit

The Quantum Processing Unit is the core of Quantum Annealers, where quantum computation is performed. These systems are designed to exploit the quantum mechanical phenomena to solve optimization problems. Essentially, a QPU contains an Ising system of qubits, where the strength of the connections between the qubits can be modified. DWave's QPU have been quickly evolving due to continuous innovation in this field. DWave's state-or-the-art QPU is the Advantage Quantum Computer, which has over 5000 qubits, 35000 couplers, and uses the Pegasus topology.

5.3. QPU Architectures

The current architecture used by DWave latest QPU's is called Pegasus as seen in (Fig. 5.1). It allows for more complex interactions due to its greater connectivity between qubits. The enhanced connectivity in Pegasus reduces the necessity for minor embedding, which is the process of mapping logical qubits to physical qubits on the QPU.

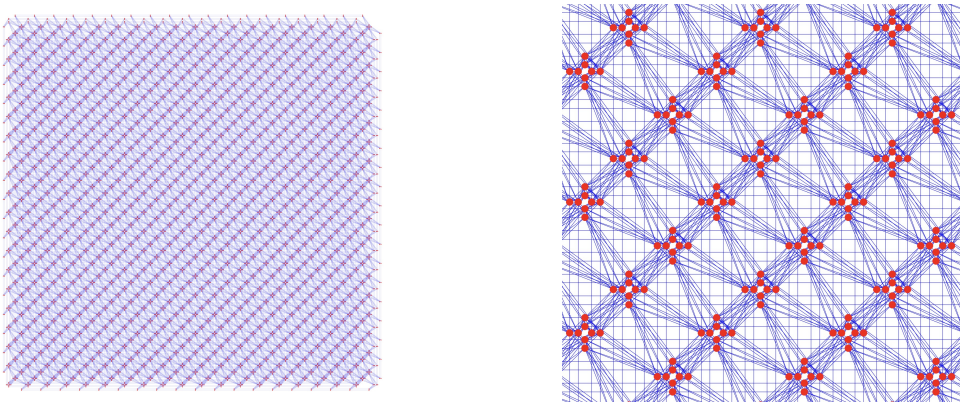


Figure 5.1: Architecture of Pegasus on DWave's Advantage QPU's. The blue lines indicate the qubit interconnections and the red dots indicate the qubits.

The interaction between the qubits is created by couplers. The strengths of the couplers are some of the values that are programmed and are selected by the values of the Hamiltonian.

5.4. Embeddings

In the current quantum computers, one of the limitations is that the qubits aren't connected in an all-to-all way. Even in the most advanced computers, qubits are only connected to some of its closest neighbours as can be seen in (Fig. 5.1) and we can only use couplers (coefficients of the QUBO model) between qubits that are interconnected. The way in which the qubits are connected in a certain quantum computer is its topology.

For example, older DWave systems used the Chimera topology (Fig. 5.2) which is much simpler than the current Pegasus topology.

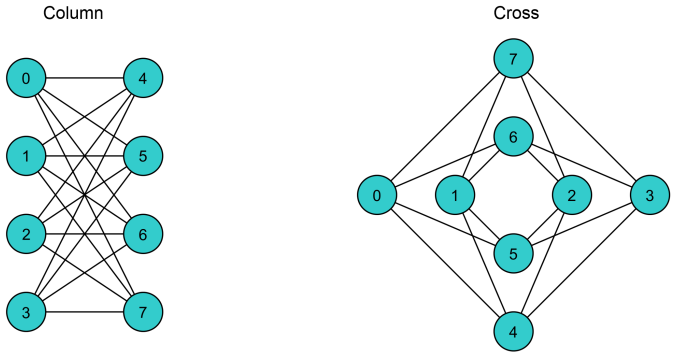


Figure 5.2: DWave's Chimera Topology Graph.

An embedding is a way of mapping logical qubits (the qubits in the Hamiltonian) to the physical qubits that exist in the topology. The problem with this, is that it is not always possible to find a one to one mapping. A lot of the times, it is needed to use several physical qubits (called a chain) to represent a single logical qubits. In that case, it is needed for all the qubits of the chain to have the same value for them to correctly represent the logical qubit. To achieve this, it is necessary to set the coupler strength between the qubits of the chain to a strength that is negative and big in absolute value.

For example, since the Chimera topology doesn't have triangular connections, we can see in (Fig. 5.3) that it is necessary to create a chain to map the original

problem to the topology by using two physical qubits acting as one logical one.

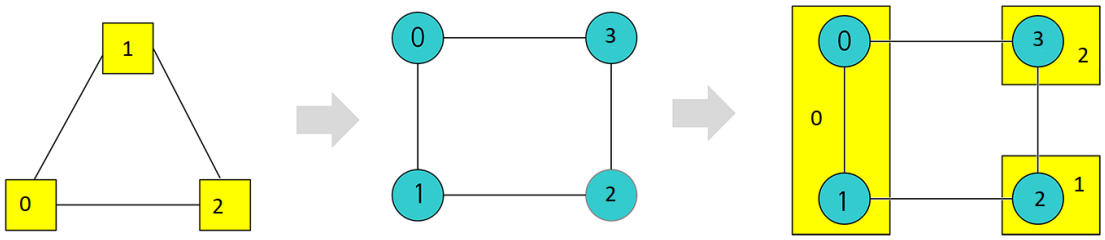


Figure 5.3: Example of embedding using a chain.

Chapter 6

Results from case studies conducted

For the implementation of the algorithm, the main library that will be used is Dwave's Dimod library [1].

Dwave Systems offer a quantum cloud computing platform named Leap [3] where by using its API, you can access Dwave's QPU computers as well as its Hybrid Solvers. Since the implementation will be using CQM, we need to use Dwave's Hybrid Solvers [2] to not only execute the algorithm in a Quantum Computer, but also do some pre processing of the mathematical model.

Once the NN has been optimized by the Quantum Annealer through Leap, the data we get is all of the solutions the Quantum Computer found. However, in some cases, many of these solutions don't meet the requirements set by the constraints. Each of the solutions, contains the values of each of the variables that were optimized, the energy level of the system, and some other useful data such as which of the constraints were satisfied and which weren't.

After the solutions are found, and the best one is picked, we have to extract the useful variables, since some of the variables were only auxiliary variables. The only variables that we need to then create the NN are the weights $W^{(k)}$ and biases $b^{(k)}$.

For the following examples, two NN Topologies are proposed:

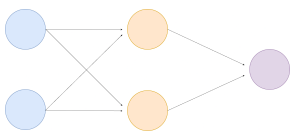


Figure 6.1: Neural Network Topology 1

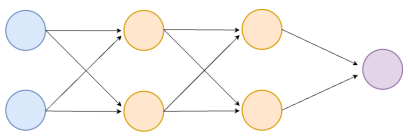


Figure 6.2: Neural Network Topology 2

6.1. Case Study 1 - Binary Classification

In this example, the NN previously created is evaluated using binary classification of data points with two dimensions x and y. The objective of this problem is to separate any given point into two different classes. The way this will be done is by creating a decision boundary that separates the two classes, then, any given point can be classified by seeing where the point is relative to the decision boundary. The optimal decision boundary is given by the line that has the maximum distance between the two closest points of different class.

6.1.1. Case Study 1.a - *Sign* Activation Function

Two datasets will be used. A smaller one with 10 data points (Fig. 6.8), and a slightly bigger one with 14 data points (Fig. 6.9). Each of the datasets have half of the points belonging to one class, and half belonging to the other class.

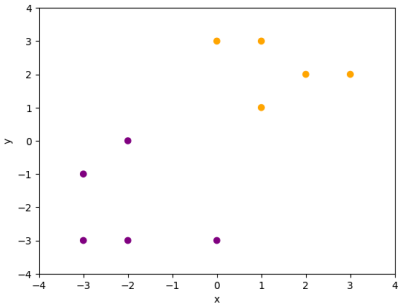


Figure 6.3: Dataset 1. 14 data points in two classes

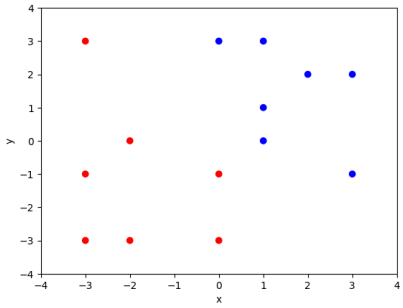


Figure 6.4: Dataset 2. 20 data points in two classes

Using dataset 1, multiple models with different parameters were executed to see the limitations of the model.

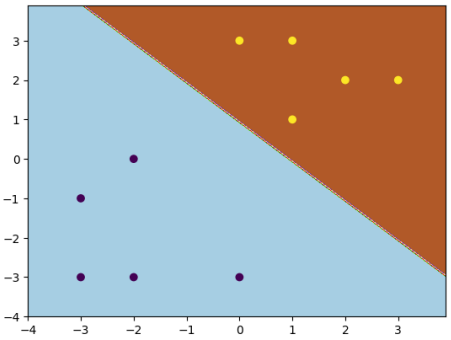


Figure 6.5: Classification of data points with an anneal time of 20μs using network topology (fig 6.1).

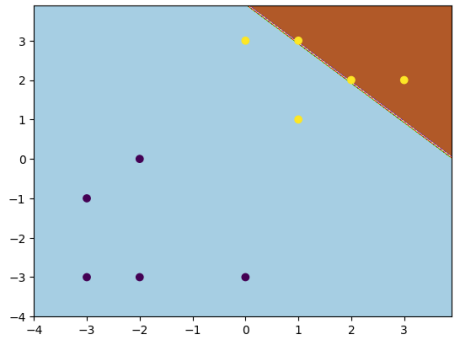


Figure 6.6: Classification of data points with an anneal time of 40μs using network topology (fig 6.2).

In (Fig. 6.10 and 6.6) we can see that having different network topologies with small changes and different parameters such as the time limit can have a huge impact on the solution correctness.

Now using data set 2 that contains more points and is more complex we get the following model.

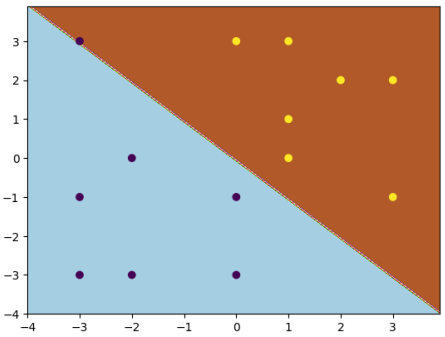


Figure 6.7: Classification with an anneal time of 20μs and using network topology (Fig. 6.1).

From all three models (Fig. 6.10, 6.6, and 6.7), we can observe the linear nature of the decision boundary. This linear behaviour is due to the fact that the activation function of the neural network is the sign activation function.

6.1.2. Case Study 1.b - *Absolute* Activation Function

Now we propose to use the absolute value function as the activation function of the neural network. The function is:

$$abs(x) \;=\; \begin{cases} x, \;\; x \geq 0 \\ -x, \;\; x < 0 \end{cases} \tag{6.1}$$

To capture the behaviour of *absolute*, the following constraint should be added to (4.10):

$$r^{(k)} \odot s^{(k)} \;=\; a^{(k)} \tag{6.2}$$

With this activation function, the model should be able to have a non-linear behaviour. Two new datasets will be used. Both of the datasets have 20 data points, half of the points belonging to one class, and half belonging to the other class as seen in (Fig. 6.8) and (Fig. 6.9).

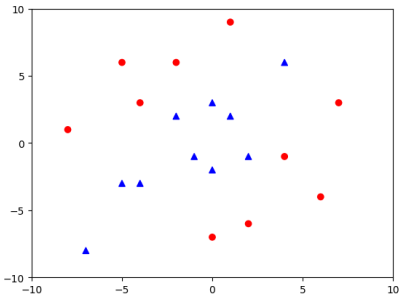


Figure 6.8: Dataset 3 with 20 data points divided into three parts with 2 classes

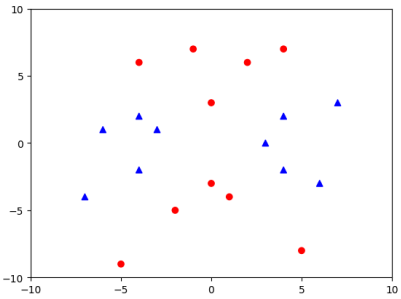


Figure 6.9: Dataset 4 with 20 data points divided into four parts 2 classes

The training results on dataset 3 are shown in (Fig. 6.10).

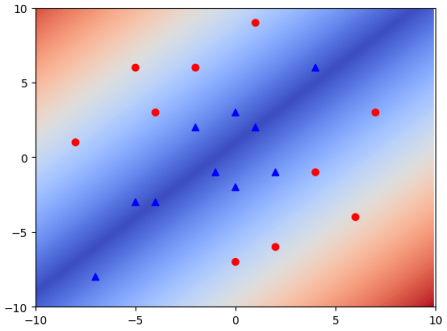
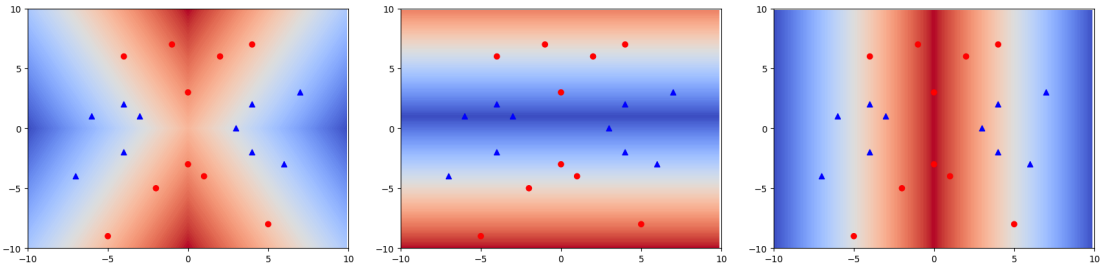


Figure 6.10: Classification of dataset 3 with an anneal time of 20μs using neural network topology (Fig. 6.1).

In (Fig. 6.11) we can see the three different training results, (a) corresponds to

the result from the quantum computer with the lowest energy level, which as can be observed, is the best model out of the three. (b) and (c) corresponds to results of higher energy level, and thus, are not the optimal results.



(a) (b) (c)

Figure 6.11: Classification of dataset 4 with an anneal time of 30μs and using neural network topology (Fig. 6.2).

The confusion matrix of the neural network model obtained is shown in (Fig. 6.12), where we can observe the model obtains high accuracy with just a few datapoints.

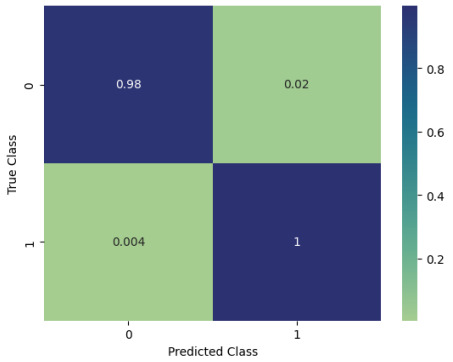


Figure 6.12: Confusion matrix of classification of dataset 4.

6.1.3. Proposal of third activation function

Now we propose a third activation function, *Exp-ReLU* given by the function:

$$\textit{Exp-ReLU}(x) = \begin{cases} x^2, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{6.3}$$

To capture the behaviour of *Exp-ReLU*, we add the following two constraints to (4.10):

$$\frac{1}{2}(r^{(k)} + s^{(k)}) \;=\; a^{(k)} \tag{6.4a}$$

$$t^{(k)} \odot s^{(k)} \;=\; r^{(k)} \tag{6.4b}$$

6.2. Comparison of activation functions

The final form of the QUBO formulation for the neural network with the three proposed activation functions, *sign*, *absolute*, and *Exp-ReLU* respectively is given by:

$$\begin{aligned}
\mathcal{L}_{penalty} \;=\; & \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \\
& + P\sum_{i=1}^{N}\left\|W^{(1)}x_i + b^{(1)} - s_i^{(1)}\right\|^2 \\
& + P\sum_{i=1}^{N}\sum_{k=2}^{L-1}\left\|W^{(k)}a_i^{(k-1)} + b^{(k)} - s_i^{(k)}\right\|^2 \\
& + P\sum_{i=1}^{N}\left\|W^{(L)}a_i^{(L-1)} + b^{(L)} - \hat{y}_i\right\|^2 \\
& \boxed{\begin{array}{l}
+\; P\sum_{i=1}^{N}\sum_{k=1}^{L-1}\left\|a_i^{(k)} \odot s_i^{(k)} - r_i^{(k)}\right\|^2 \\
+\; P\sum_{i=1}^{N}\sum_{k=1}^{L-1}\left\|a_i^{(k)} + 2r_i^{(k)} - 1 - t_i^{(k)}\right\|^2
\end{array}} \\
& \boxed{+\; r^{(k)} \odot s^{(k)} \;=\; a^{(k)}} \\
& \boxed{\begin{array}{l}
+\; \frac{1}{2}(r^{(k)} + s^{(k)}) \;=\; a^{(k)} \\
+\; t^{(k)} \odot s^{(k)} \;=\; r^{(k)}
\end{array}}
\end{aligned} \tag{6.5}$$

In (Table 6.1) we observe the huge impact that the activation function has on the model's accuracy. We also see that increasing the anneal time may result in a higher accuracy, but after a certain point, it doesn't provide much improvement.

Activation Function	Anneal Time	Train set Accuracy	Test set Accuracy
Sign	20μs	0.65	0.552
Sign	40μs	0.75	0.602
Sign	60μs	0.75	0.622
Absolute	20μs	1	0.988
Absolute	10μs	0.95	0.956
Absolute	5μs	0.95	0.934
Exp-ReLU	5μs	0.90	0.835
Exp-ReLU	10μs	0.90	0.902
Exp-ReLU	20μs	0.95	0.924

Table 6.1: Accuracy comparison of different models on dataset 4.

6.3. Case Study 2 - Classification of Images

Another example that can be solved using the QNN on a Adiabatic Quantum Computer is handwritten number classification. Since there are current limitations on the number of qubits adiabatic quantum computers have, only two numbers will be chosen to be classified, in this case the numbers 6 and 7.

The dataset used in this example is the MNIST dataset [4]. The MNIST dataset is an ample dataset of handwritten numbers ranging from 0 to 9 (only one digit numbers) that is commonly used as the basic example for image classification. The MNIST dataset contains 70,000 samples, each sample is a 28 x 28 pixel image representing a number. Each pixel ranges from 0 to 255.

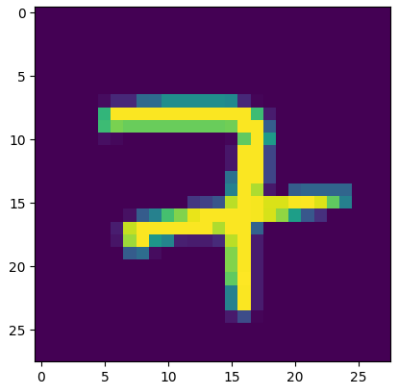


Figure 6.13: Sample of the MNIST dataset.

To train the model on a adiabatic computer and achieve a reasonable accuracy,

two actions are needed. First, in order to reduce the number of output neurons, we will convert the problem to a binary classification by selection only two numbers to be classified. Secondly, since the default size of each sample in the dataset is 28 x 28 = 784 a lot of input neurons would be needed, so each image will be down sampled.

To reduce the number of input neurons, each image will be down sampled to a 2 x 2 image as seen in (Fig. 6.14). The first step in the down-sampling process is to divide the image into 4 quadrants, then, the pixel value of the down-sampled image will be the mean of the pixels of the corresponding quadrant.

Until this step, the current sample is now a 2 x 2 pixel image, however, since the numerical range of the pixels is 0 to 255 and a quantum computer deals with qubits, each pixel would need to be represented by 6 qubits. 2 x 2 x 6 = 24 qubits needed to represent the down sampled image. To decrease the number of qubits need, each pixel will be normalized to a smaller range of values.

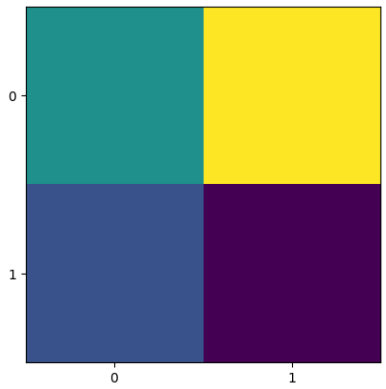


Figure 6.14: Down-sample of (Fig. 6.13)

The network topology used is a 3-layered neural network, with 4 input neurons, 2 neurons in the hidden layer, and 1 output neuron. In order to train the model, 20 samples were chosen, 10 sample for each number.

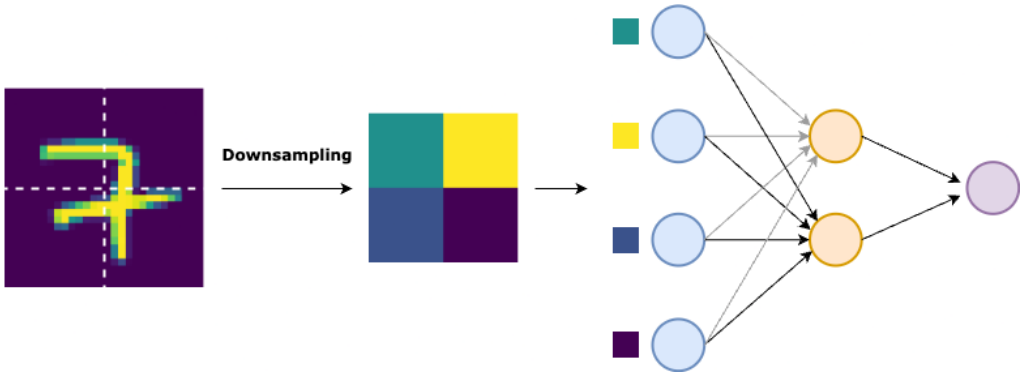


Figure 6.15: Downsampling process to convert the original MNIST sample to a more usable sample for the NN topology.

Once all the samples are ready to be plugged into the neural network, the CQM is created and then executed on Dwave's Hybrid CQM Solver. The solver resulted in 133 solutions, from which 99 met all of the constraints of the CQM. From the valid solutions, we extract the one with the lowest energy, which corresponds to the solution that best minimizes the objective function. The best solution had a energy of 0, meaning that the model fit perfectly the 20 samples given the to neural network, however, this doesn't mean that the neural network will achieve a 100% accuracy.

To now use the model, the decision variables have to be extracted from the best result given by the Hybrid CQM Solver. Once the decision variables are extracted, we create a normal neural network with the same topology and we set the weights and biases of the model to the values extracted from the Quantum Computer. Using samples different from the training samples, we can evaluate the true accuracy of the model.

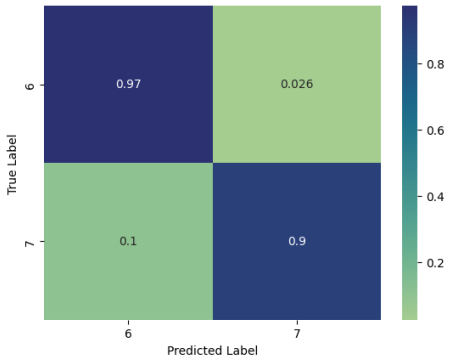


Figure 6.16: Confusion Matrix.

From the confusion matrix (Fig. 6.16), we can observe that the model achieved a high accuracy from only 20 samples with only 4 pixels each.

Now we can use different combinations of parameters and see the accuracy they produce, as seen in (Table 6.2). It is clear that with very few pixels and images, the model doesn't have a great accuracy, however, when we increased the number of images or number of pixels per image, the model got a much better accuracy over the test data.

Pixels	Training size	Test set Accuracy
4	10	0.63
4	20	0.93
8	10	0.95
8	20	0.91

Table 6.2: Accuracy comparison of different models on mnist dataset, where pixels represents the number of pixels use to represent the image, and the training size is the number of images used to train the model.

6.4. Case Study 3 - Classical Algorithms vs Quantum Annealing

In this experiment, we will compare a few metrics (mainly runtime) of the results of executing classical optimization algorithms and quantum annealing on a BNN.

We will use the UCI adult dataset [5], the prediction task of this dataset is to determine if a person earns more than 50k $ a year. The dataset contains 14 data points such as age, work class, education, etc. In order to use the features of this dataset in the BNN, first, they need to be binarized. The output of the BNN will be a single binary neuron that will determine whether the prediction is that the person makes more or less than 50k $.

Two different networks will be used for comparing results:

- **(Task 1)** 10 batches with 3 input attributes and $N = 4$ data points per batch.

- **(Task 2)** 10 batches with 3 input attributes and $N = 15$ data points per batch.

The BNN used will have two layers with 3 input neurons, 3 hidden neurons, and 1 output neuron for both tasks.

The following algorithms will be used:

- Branch-and-bound for QCBO, QUBO, and eQUBO formulations.
- Simulated annealing for the QUBO and eQUBO formulations. The number of anneals of will be set to both 500 and to 4000.
- Quantum Annealing for eQUBO with 500 samples.

Branch and Bound is a classical method for solving optimization problems by breaking them down onto smaller sub problems and using a bounding function to discard the sub problems that cannot contain the optimal solution, thereby, greatly reducing the search space. As we will see, this method is extremely quick for small problems, but as the number of variables increases, the cost of the algorithm grows exponentially. The Branch and Bound method guarantees to find the optimal solution to the optimization problem.

eQUBO refers to the QUBO formulation but embedded on a specific graph as can be seen in (Fig 6.17), in this case, it will be embedded on DWave's Pegasus QPU topology. This is to mimic as close as possible the problem that the QPU will solve, since the QPU is restricted to its topology.

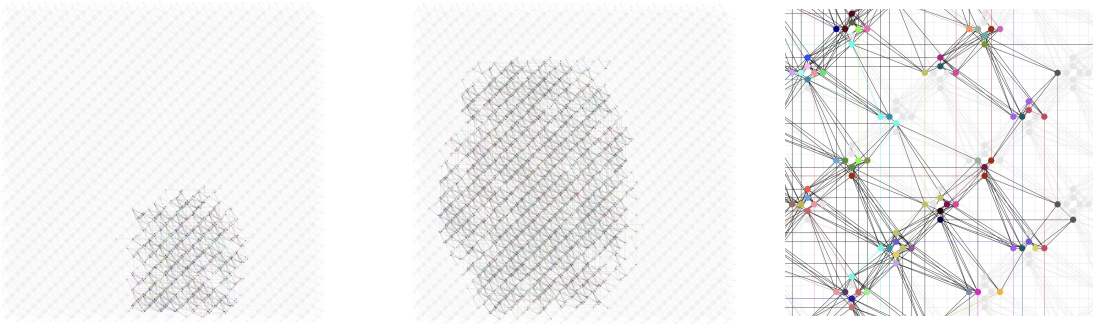


Figure 6.17: (left) The embedding of **Task 2** on the Advantage System used to solve the problem. (Middle) Embedding of an instance of the BQNN training problem with 50 data points employing 1212 qubits. (Right) Closer look to the embedding of a problem in the quantum computer.

Let $\theta_{BnB},\ \theta_{SA},\ \theta_{QA}$, since BnB on QCBO conducts a deep search, it finds the global optimum for the weights of the BNN ($\theta_{BnB.QCBO}$). We will take this value as the ground truth, use it as the baseline to compare the rest of the results, and compute the distance of another solution with weights (θ) to the ground truth as:

$$d(\theta) \ = \ \sum_{i=1}^{N} \left|\hat{y}_i - f_{BNN}(\theta_{BnB.QCBO})(x_i)\right| \ - \ \left|\hat{y}_i - f_{BNN}(\theta)(x_i)\right|$$

In (Fig 6.18, 6.19, 6.20) are shown the results of using different algorithms to solve **Task 1** in blue and **Task 2** in orange. (Fig 6.18) shows that the distance of BnB on QUBO and eQUBO is zero which verifies that the QUBO formulation (4.15) matches the QCBO as seen in (Fig 4.3).

The setup for executing all of the classical algorithms has been a MacBook Air with an Apple M2 Chip, 8GB of RAM and macOS Sonoma.

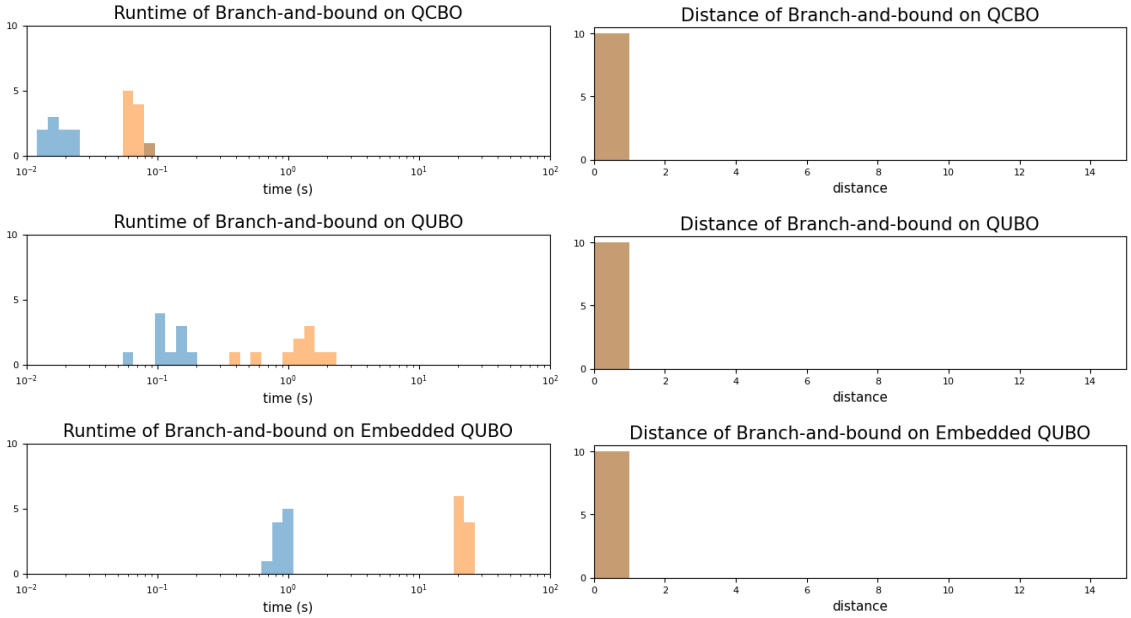


Figure 6.18: (left) Histogram of runtime using the Branch and Bound algorithm in log scale with different formulations of the problem. (right) Histogram of the distance to the ground truth of the solution.

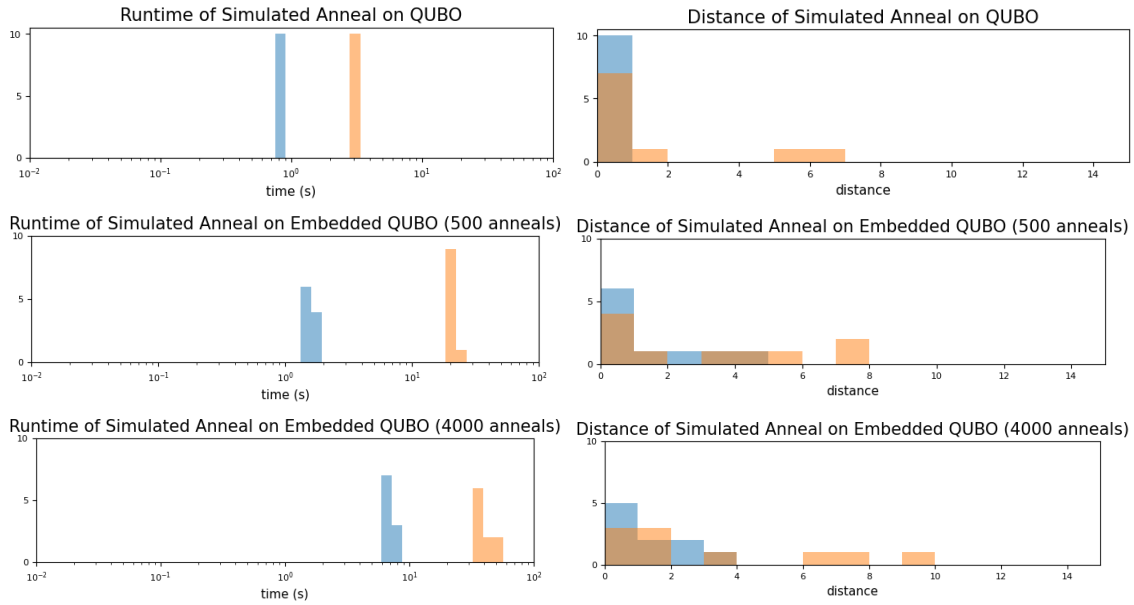


Figure 6.19: (left) Histogram of runtime using Simulated Annealing in log scale with different formulations of the problem and different number of anneals (number of samples taken). (right) Histogram of the distance to the ground truth of the solution.

In (Fig 6.19) we see that the runtime of Simulated Annealing is at least one order of magnitude bigger than that of the BnB algorithm. The distances of is not always zero, which means that the simulator also simulated noise that is experienced by quantum computers to more closely simulate them. We can also see that the more samples or anneals we do in the simulator, the longer it takes, however, taking more samples, can also lead to a better result.

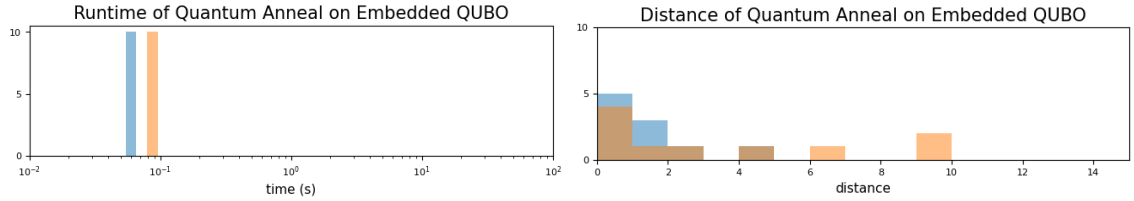


Figure 6.20: (left) Histogram of runtime using Quantum Annealing in log scales. (right) Histogram of the distance to the ground truth of the solution.

Finally, in (Fig 6.20) we see the results of executing **Task 1** (blue) and **Task 2** (orange) on the DWave Advantage 6.1 Solver. We can see how much faster the runtime is on the quantum computer compared to the other methods.

Training size	Samples	Features	Anneal time (μs)	% of available qubits used	Test set accuracy
4	100	3	20	3.99 (230)	0.4344
4	200	7	20	7.96 (459)	0.4682
15	100	3	20	15.45 (890)	0.5212
15	200	3	20	15.45 (890)	0.5412
15	2000	3	20	15.45 (890)	0.6124
15	2000	3	40	15.45 (890)	0.4387
15	200	7	20	35.19 (2027)	0.4896
15	2000	7	20	35.19 (2027)	0.5068
15	2000	7	40	35.19 (2027)	0.5274
30	2000	3	10	29.72 (1712)	0.6124
30	2000	3	20	29.72 (1712)	0.6280
30	2000	3	40	29.72 (1712)	0.5768
50	2000	3	20	49.86 (2872)	0.5768

Table 6.3: Accuracy of models with different parameter combinations. Training size is the number of datapoints used to train the NN, the number of samples is how many times the model was run on the QPU from which the best one is chosen. The anneal time is how slowly the Hamiltonian was evolved to its final state. And also shown is the percentage and the number of qubits used.

Now using the same methods, we will run a series of tests using different parameters to see the effect they have on the accuracy of the resulting model as seen in (Table 6.3). The two main things two notice is how the training size has a big impact on the accuracy, this is because with more data, the model is able to generalize better. Also, the number of samples taken by the QPU also has a noticeable effect on the accuracy, if more samples are taken, there is a higher probability of one of them being a lower energy solution.

Runtime Scaling

Comparing the runtime of the different algorithms, we see that BnB on QUBO and eQUBO is a lot higher than that of the QPU. This means that if the problem was even bigger, the difference in runtime would be grow extremely quickly, and thus, showing that Quantum Computers have a huge advantage when it comes to the runtime of optimization problems.

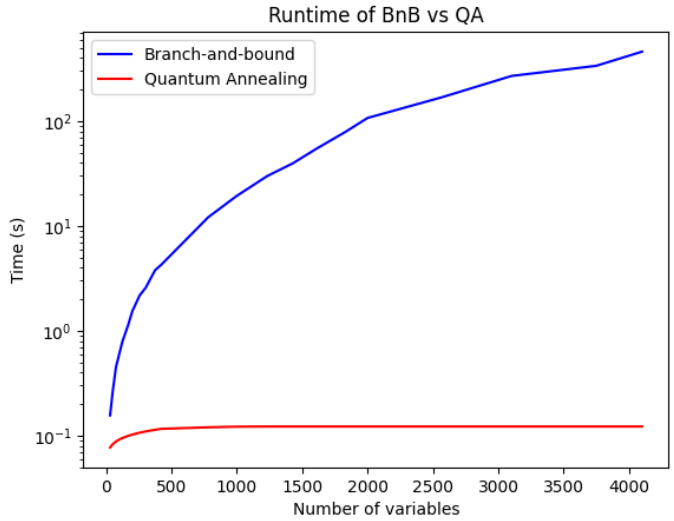


Figure 6.21: Runtime of BnB vs QA as the number of variables increases on a BNN with 3 input neurons, 3 hidden neurons, and 1 output neuron.

In (Fig. 6.21) we can see that the runtime of one of the fastest classical optimization algorithms increases exponentially with the number of variables, while the estimated runtime on the Quantum Annealer barely increases when the number of variables increases. The small change of runtime of the QA is only due to the readout time per sample which is affected by the number of qubits to read after execution. The readout time increases linearly, and as we can see, it has almost no effect, since it is measured in microseconds. However, the number of variables (or qubits in the QA) is limited by the current Quantum Computers developed.

Chapter 7

Conclusion

This thesis has aimed to provide a fundamental mathematical basis about optimization problems in quantum computers using formulations like QUBO, and QCBO, and to show how these mathematical methods can be used to find the optimal parameters of a NN of any size with linear layers, and a sign activation function.

Through the different study cases, we have shown the feasibility of using quantum annealers to train a NN and achieve a good accuracy as well as the benefits in terms of runtime of using quantum annealers to solve optimization problems compared with classical algorithms.

A significant portion of the study has involved implementing a NN on a quantum annealer. We discussed the problems and benefits of this approach compared to classical methods. Specifically, we showed that while classical optimization algorithms like Branch and Bound are powerful, they can be slower for larger datasets. While, on the other hand, quantum annealing offers a faster alternative.

One of the experiments focused on is binary classification of the UCI Adult dataset. We compared the runtime and accuracy of training a BNN using classical algorithms versus quantum annealing. The results showed that quantum annealing has the potential to provide significant speed advantages when it comes to solving this kind of problems, especially as quantum computing technology continues to improve.

Bibliography

[1] Dimod quadratic models documentation. https://docs.ocean.dwavesys.com/en/stable/docs_dimod/sdk_index.html.

[2] Dwave hybrid samplers documentation. https://docs.ocean.dwavesys.com/en/stable/docs_hybrid/sdk_index.html.

[3] Leap - dwave's quantum cloud computing platform. https://cloud.dwavesys.com/leap/.

[4] Mnist dataset. http://yann.lecun.com/exdb/mnist/.

[5] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[6] Fred W. Glover and Gary A. Kochenberger. A tutorial on formulating QUBO models. *CoRR*, abs/1811.11538, 2018.

[7] Avradip Mandal, Arnab Roy, Sarvagya Upadhyay, and Hayato Ushijima-Mwesigwa. Compressed quadratization of higher order binary optimization problems, 2020.

[8] Michele Sasdelli and Tat-Jun Chin. Quantum annealing formulation for binary neural networks, 2021.

[9] Xujie Song, Tong Liu, Shengbo Eben Li, Jingliang Duan, Wenxuan Wang, and Keqiang Li. Training multi-layer neural networks on ising machine, 2023.

[10] Tomohiro Yokota, Makiko Konoshima, Hirotaka Tamura, and Jun Ohkubo. Derivation of qubo formulations for sparse estimation. *Journal of the Physical Society of Japan*, 89(3):034801, March 2020.

[11] Chunyu Yuan and Sos S. Agaian. A comprehensive review of binary neural network. *CoRR*, abs/2110.06804, 2021.