QUBO-based Neural Networks on Quantum Computers

Master's Thesis in
Computer Science
by

Georgi Georgiev

16 August 2023

Institute for Anthropomatics and Robotics (IAR)
KARLSRUHE INSTITUTE OF TECHNOLOGY (KIT)

Supervisor:	Prof. Dr. Jan Niehues
Co-supervisor:	Prof. Dr. Rainer Stiefelhagen

Abstract

Machine Learning is one of the fastest-developing areas in computer science. It has a wide range of applications including automation, image recognition, language processing and many others. Deep learning in particular is a subset of machine learning which has been at the forefront of many breakthroughs. However, it has very high computational requirements as it utilizes large datasets and the size of the networks constantly increases in order to improve performance. Among neural network variants, binary neural networks have gained attention due to their reduced memory requirements, faster inference and energy efficiency. These are essential qualities for the implementation of neural networks on embedded systems and small devices such as smartphones. The training of binary neural networks is challenging, as they require additional algorithms which approximate gradient descent. We look into quantum-inspired formulations for a better alternative. Quantum computing has been gaining popularity in recent years as it holds great promises for optimisation, simulation and secure communication but also could be a great threat to the current security paradigm. Quantum computers perform computations over higher-dimensional search spaces, potentially leading to improvements in many areas of machine learning. We focus on a versatile mathematical formulation called a Quadratic Unconstrained Binary Optimisation (QUBO), which is required by quantum annealers, the most advanced current quantum devices in terms of number of qubits. It can also be solved on gate-based quantum computers and classical computers. There has been a recent formulation for the training of a binary neural network to be implemented as a QUBO, which does not require gradient descent at all. We offer a model that improves upon the original formulation, tackling the problem of bad scalability and achieve a QUBO which is capable of handling much bigger feature spaces for our networks. Furthermore, we allow for training over time which removes some volatility and produces more robust weights. We also add some quality-of-life changes to the model such as separately tunable penalty factors and more freedom when choosing the size of the feature vectors and the number of hidden neurons. Because of our improvements, we are, to the best of our knowledge, the first to fully train a binary neural network for image classification and sentiment analysis tasks by using a QUBO. We test our model on preprocessed data from the MNIST and IMDb datasets and we solve our QUBOs on a digital annealer, achieving promising results and in very specific cases outperforming an unoptimised classical neural network of the same size with the same loss and activation functions.

Zusammenfassung

Maschinelles Lernen zählt zu den am schnellsten wachsenden Bereichen der Informatik. Es findet vielfältige Anwendungen, darunter Automatisierung, Bilderkennung, Sprachverarbeitung und viele andere. Insbesondere im Bereich des Deep Learning, einer Teilbereich des Maschinellen Lernens, haben sich zahlreiche Durchbrüche ereignet. Allerdings hat Deep Learning sehr hohe Rechenanforderungen, da es sehr große Datensätze verwendet und die Größe der Netze ständig erhöht wird, um die Leistung zu verbessern. Unter den Varianten Neuronaler Netze haben Binäre Neuronale Netze aufgrund ihrer reduzierten Speicheranforderungen, schnelleren Inferenz und Energieeffizienz Aufmerksamkeit erregt. Dies sind wesentliche Eigenschaften für die Implementierung von Neuronalen Netze in eingebetteten Systemen und kleinen Geräten wie Smartphones. Das Training von Binärer Neuronaler Netze ist jedoch eine anspruchsvolle Aufgabe, da zusätzliche Algorithmen erforderlich sind, um den Gradientenabstieg zu nutzen. Wir betrachten quanteninspirierten Formulierungen, um eine bessere Alternative zu finden. Quantencomputing hat in den letzten Jahren auch an Popularität gewonnen, da es vielversprechende Ansätze für Optimierung, Simulation und sichere Kommunikation, aber auch potenzielle Bedrohungen für das aktuelle Sicherheitsparadigma bietet. Quantencomputern können Berechnungen in höherdimensionalen Räumen durchführen, was zu Verbesserungen in vielen Bereichen des maschinellen Lernens führen kann. Unter den heutigen Quantentechnologien ist das Quanten-Annealing am weitesten entwickelt und bietet die größte Anzahl an Qubits. Quadratisches Unbeschränktes Binäres Optimierungsproblem (QUBO) ist eine vielseitige mathematische Formulierung, die von Quanten-Annealern benötigt wird, aber auch auf gatterbasierten Quantencomputern und klassischen Computern lösbar ist. Wir präsentieren ein Modell, das eine existierende Formulierung für das Training eines Binären Neuronalen Netzes als QUBO verbessert. Wir versuhen das Problem der schlechten Skalierbarkeit zu lösen und bauen ein QUBO, das viel größere Featureräume für die Binären Neuronalen Netze nutzen kann. Darüber hinaus ermöglichen wir auch Online Training, was teilweise Volatilität beim Training entfernt und stabile Gewichte erzeugt. Wir fügen dem Modell auch einige Verbesserungen hinzu, wie separat einstellbare Straffaktoren und mehr Freiheit bei der Wahl der Größe der Merkmalsvektoren und der Anzahl der Hidden Neuronen. Dank unserer Verbesserungen sind wir, unserer Kenntnis nach die ersten, die ein Binäres Neuronales Netz für Bilderkennungs- und Sentimentanalyseaufgaben vollständig, mithilfe eines QUBOs trainieren können. Wir testen unser Modell anhand vorverarbeiteter Daten aus den MNIST- und IMDb-Datensätzen und lösen unsere QUBOs auf einem digitalen Annealer, wobei wir vielversprechende Ergebnisse erzielen und in sehr speziellen Fällen nicht optimierte klassische Neuronale-Netze mit derselben Größe, Loss- and Aktivierungsfunktionen übertreffen.

Contents

1 Introduction

Machine learning (ML) models are prominent in many fields today but take a lot of time to be trained and are usually computationally expensive. It is evident that they are also a crossing point for many topics like Geometry Processing, Embedded Systems, Context-sensitive Systems and Medicine Robotics. The focus of this work will be on Neural networks (NNs), because of their wide array of use cases in all of the above-mentioned fields. They are also arguably the best performing ML method in the last decade, driving most of the recent successes [1].

Among these, Binary neural networks (BNNs) are powerful and very efficient quantization models in terms of memory consumption, energy consumption and speed of inference [28], which makes them probably the best-suited network models for resource-constrained environments such as embedded systems and portable devices. They achieve this at the cost of slightly lower accuracy in comparison to full-precision models [28]. Their training presents challenges due to the discrete and non-differentiable nature of binary operations. Traditional approaches often employ a method called Straight Through Estimator (STE) to circumvent this issue by approximating gradient descent, but these estimators may introduce quantization errors and hinder training convergence [29]. In search of a way to circumvent these difficulties, we look into quantum and quantum-inspired technologies and formulations for a way to implement the training of a BNN without an STE and without gradient descent.

Even though large-scale fault-tolerant quantum computers are not expected to be realized in the near future, quantum computing is being researched in many fields of computer science. Naturally, its promises for exponential speed up for computation tasks have made it very alluring to ML researchers. In the last decade Quantum machine learning (QML) has been a rapidly growing field with efforts in many areas including Support Vector Machines (SVMs) [2], Principal component analysis (PCA) [3], K-nearest neighbour (KNN) algorithms [4],[5] and K-means clustering [6]. Finding an efficient way for NNs to be implemented on a quantum computer might be an important milestone for ML, considering the computational advantages promised by quantum technology and the widespread use of NNs. There have been many efforts to create Quantum neural networks (QNNs) [7], [8], but with the current quantum technology [9] scaling those to big sizes will likely be a challenging task.

It is no surprise that there also has been ongoing research on training classical NNs with gate-based quantum computers and it has gained significant attention in recent years [8], [10]. The objective is to leverage the computational power of quantum systems to enhance the training process for classical NNs. Because of the different principles of quantum computing, such as superposition, tunnelling and entanglement, researchers aim to develop novel algorithms and architectures that can utilize those to overcome the limitations of classical approaches. Although this field is still in its early stages of development, ongoing research explores various strategies, including quantum-inspired classical algorithms [11] and hybrid quantum-classical methods [12], [13], [14]. Currently, full-precision NNs are too big and too complex to be fully trained by today's quantum technology. However, successful hybrid implementations such as [15], [13] and [12] which improve classical training, show promising results.

Most current implementations for QNNs and the quantum training of classical NNs involve replacing parts of the NN with a variational quantum circuit. This is a type of quantum circuit that is using rotation operator gates with free parameters to perform various numerical tasks, such as approximation, optimization and classification. The variational quantum algorithm is a hybrid algorithm that is very similar to a NN in that it approximates functions through parameter learning, but also benefits from quantum phenomena. Some prominent examples would be the replacing of the generators [14] and/or the discriminators [16],[17] of generative models and the convolutional and pooling layers of Convolutional neural network (CNN) [18], [19] with quantum circuits. There have been some recent efforts to implement the policy training for reinforcement learning by using a variational quantum circuit [20]. There are also approaches that replace the input layer of a NN with artificial neurons [21] which use fewer parameters to learn the complex and non-linear patterns in data. The possible applications of these quantum technologies for ML and its use cases for the industry are gathering attention and are being investigated in many projects such as the German initiatives QuCUN [22] and QCHALLenge [23]. However, it is important to note that in all of these works, we see a significant reduction of the dimensions of the data either during preprocessing or via a hybrid implementation utilizing classical computers. This is required because of the limited maturity of state-of-the-art quantum computers, most experiments are performed with 4-6 qubits. Implementing the full training of a big NN on a gate-based quantum computer could lead to faster and more efficient training of NNs, allowing for further advancements in ML and artificial intelligence, however, with the rate at which gate-based technology is developing [9], this step will likely take a while.

Adiabatic quantum computing (AQC) is the second prominent type of quantum computation, which we will consider. It relies on the adiabatic theorem and utilizes the principles of superposition, tunnelling and entanglement that manifest in quantum physical systems in order to solve optimisation problems. There has been a lot of progress in adapting ML algorithms for AQC including SVMs [2], linear regression [24] and K-means clustering [6]. Physical realizations of AQC called quantum annealers are the most developed quantum technology in terms of number of qubits. The currently largest quantum annealer has more than 5000 qubits [25] compared to the largest gate-based machines having an order of magnitude less [9]. This is a major factor considering the representational power needed to train a NN and makes them suitable for scaling up the size of the trained networks. Additionally, quantum annealers being specifically designed for optimization problems, exploration of large solution spaces and finding optimal configurations, makes them well-suited for the optimization of the loss function of a NN. They are also more stable as annealers directly map the optimisation problem onto their hardware architecture, exhibiting greater noise resilience compared to gate-based quantum computers and being less prone to measurement errors and decoherence. Furthermore, because of this mapping, quantum annealers have reduced computational overhead compared to gate-based counterparts, potentially speeding up training times for real-world testing. Lastly, by exploring a wider range of configurations near the global minimum, quantum annealers could contribute to the discovery of parameters that exhibit improved generalization abilities.

A Quantum Unconstrained Binary Optimisation (QUBO) is a mathematical formulation for binary variables that aims to minimize a quadratic objective function subject to no constraints. It is the native input format for a quantum annealer, which is one of their limitations. Higher degree polynomials can also be solved on annealers by first internally transforming them to quadratic [26], [27], but this comes at the cost of increasing the size of the problem and the qubits required for solving it. However, the QUBO formulation offers a unique opportunity for a problem to be solved on a classical computer by using Simulated annealing (SA), genetic algorithms or Digital annealing (DA), on a quantum annealer or on a gate-based quantum computer by utilizing the Variational Quantum Eigensolver (VQE) or the Quantum Approximation Optimization Problem (QAOA) algorithms. Considering the early stages of research for combining quantum computing with ML the flexibility in terms of solution methods for the QUBO formulation is certainly advantageous. By formulating the training process of a BNN as a QUBO problem, the binary weights and activations can be directly optimized without relying on approximations. This approach offers several benefits, including a native binary environment and no rounding during the training, hopefully leading to fewer quantization errors and faster convergence. By utilizing the benefits of not using an STE for the training, there is a hope to enhance the robustness, efficiency, and overall performance of the trained BNNs, while maintaining or even improving the accuracy, opening up new possibilities for their application in real-world scenarios. The advantages quantum annealers have for training NNs which were mentioned above are even more pronounced for the training of BNNs because of the exploration over a binary solution space where no approximations need to be made, as it is the natural domain for annealers.

These natural connections have led to research being done trying to combine AQC and ML. There have been formulations for a few ML models such as KNN, linear regression and clustering [6]. And because Quantum annealing (QA) is also a very good generative tool there have been models, which utilize it for sampling in order to enhance the training of NNs [31], [32]. There have also been a few attempts to fully model the training of NNs [26] and BNNs [33] as a QUBO, but this is a very underdeveloped area of research.

In this work, we improve upon the QUBO formulation from [33]. It presents a natural and intuitive way to implement the training of a BNN and is very transparent with the values for all of the neurons which makes it a good model for research. We also explain the main differences, advantages and disadvantages compared to the approach presented in [26]. The main limitation that is reported in both approaches is connected to the scalability of the formulation and our main effort is to reduce the size of the resulting QUBO and also to allow for bigger feature vectors to be used. Even though the scaling of our formulation is still polynomial, limiting the size of the BNN and the amount of training data, we are nevertheless able to do realistic classification on preprocessed data.

We further improve the general utility of the formulation by developing an algorithm for training over time which allows for the training over multiple batches one after the other, continuously improving the solution. The training over time scheme theoretically adds possibilities for more complex weights, such as ternary weights $\{-1, 0, 1\}$ or even small integers, to be approximated after the training, even though the QUBO training itself remains completely binary. We also give more freedom of choice when it comes to the size of the feature vector, which is limited to one less than a power of two in [33]. Furthermore, we allow for fine-tuning of the penalty terms, which results in more opportunities for fine-tuning and also helps in penalizing some infeasible solutions. Lastly, we adapted our model for the two use cases of image classification and sentiment analysis, which are very prominent problems from computer vision and Natural language processing (NLP) respectively. These are widespread use cases, which provide a good baseline for the performance of our model. Furthermore, they also have some of the most rigorously tested and balanced low-complexity datasets such as the MNIST and IMDb datasets. Additionally, to the best of our knowledge, no work has presented a fully trained NN for these tasks achieved solely via a QUBO, with a feature vector size that is applicable to real-world scenarios.

We did our benchmarking and testing on the Fujitsu Digital annealer [34] which is a novel quantum-inspired technology that uses annealing to solve QUBO problems. Because it is fully classical we were able to use it for testing without fear of decoherence or quantum errors. The Digital annealer is capable of handling problems with qubit sizes in the thousands which allows it to solve problems in the same or higher realm of complexity as the D-Wave Advantage which has 5000 qubits. The quantum annealer offers limited connectivity [25], so when there are many 2-qubit connections in the QUBO matrix, extra qubits are needed in order to compensate for these interactions, effectively reducing the number of available qubits. The digital annealer, on the other hand, offers full connectivity and the QUBO matrix can be directly mapped on the hardware. We are doing tests up to 8000 qubits which is what the digital annealer should be able to solve without problems [34]. This gives us a good approximation for the performance of quantum annealers in the near future and produces dependable results for testing. Our formulation was tested on down-sampled versions of the datasets MNIST and IMDb used for image recognition and sentiment analysis respectively. Even though some reduction in the feature space was done to make the experiments take a shorter amount of time, we showed that our model can be used with the full-scale images from MNIST. For the sentiment analysis, the formulation was able to handle vocabulary size in the hundreds which, provided a good preprocessing, is enough for easier datasets.

We describe the theoretical backgrounds and relevant related work in chapter 2 and chapter 3. Then we explain the formulation of the original model in detail in chapter 4. We describe the different improvements we made and how to apply the new model to image classification and sentiment analysis in chapter 5. The setup for our experiments and the discussion of the observed results we present in chapter 6. Lastly, we briefly summarize our results in chapter 7 and offer some ideas and suggestions for future developments in chapter 8.

2 Background information

2.1 Quantum unconstrained binary optimisation (QUBO)

A QUBO is a mathematical formulation for binary variables that aims to minimize a quadratic objective function subject to no constraints. The general form of a QUBO problem can be expressed as:

$$\min \quad f(\mathbf{x}) = \sum_{i} q_{ii}x_i + \sum_{i<j} q_{ij}x_i x_j \tag{2.1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a binary decision vector representing the variables, and the coefficients q_{ii} represent the linear terms of the objective function, while q_{ij} represent the quadratic terms. We also add a constant E to eq. (2.1) for all coefficients q which are not dependent on any of the variables x. This constant does not affect the solution vector $\mathbf{x}$ of the QUBO but changes the minimum value of $f(\mathbf{x})$.

The solution of the QUBO problem is a vector of binary variables that minimizes the objective function $f(\mathbf{x})$ and finding it is an NP-hard problem. A solution can be achieved through various classical optimization techniques, such as SA, genetic algorithms and DA, quantum approaches like QA or hybrid approaches such as QAOA and VQE. It is also possible to use simulated QA which is performed solely on a classical machine and simulates the evolution of the Hamiltonian (see section 2.1.1). It is proposed that it also benefits from adiabatic speed-up [35] similar to QA. Notably, these are all probabilistic schemes, so the quality of the found solutions can vary. Considering the early stages of research for combining quantum computing with ML the flexibility in terms of solver options of the QUBO formulation presents a good base for future developments in many directions.

The QUBO formulation can also allow for the splitting of any problem of size n in two QUBO problems of size $n-1$ known as Divide-and-Conquer [36]. These can be solved independently of each other opening possibilities for parallelisation. By itself this simplification is not very efficient, however, if the structure of the problem is exploited each time such a simplification is made, the size of both QUBOs can be reduced by more than one variable making it an effective way of speeding up calculation. This reduces the overall qubit requirements while also improving the approximations of the solving heuristics [36].

Overall the QUBO formulation has a few notable advantages which make it a great candidate for implementing the training of BNNs. QUBO is a versatile framework capable of representing a broad spectrum of optimization problems, which is also naturally aligned to tackle binary problems and can become highly parallelisable by utilizing schemes such as Divide-and-Conquer. Furthermore, as mentioned above, there is a plethora of ways to solve the QUBO problem. This compatibility opens up avenues for harnessing quantum computing capabilities, classical computing and even hybrid approaches to explore new ways for training BNNs and to get insights into the future of training NNs as a whole.

We are going to go into further detail about some of the most popular ways to solve QUBOs which are crucial for our research, namely SA, with DA in particular which is going to be used for the experiments in chapter 6 and QA, which is the best candidate quantum technology for solving our formulation.

2.1.1 Quantum Annealing (QA)

Currently, there are two main quantum computing paradigms: Gate-based quantum computing (GQC) and AQC. Currently, the practical realisation of AQC in the form of quantum annealers is more developed and offers machines with a larger number of qubits. QA is an optimisation technique that utilizes quantum mechanical phenomena such as quantum tunnelling, quantum entanglement and quantum superposition and aims to find the lowest energy state, or the global minimum, of a given problem's objective function. This is achieved through the energy minimization of a physical system. QA as a computational technique was proposed by Tadashi Kadowaki and Hidetoshi Nishimori in 1998 [37]. One of the main developers of quantum annealers as of today is D-Wave Systems and it has played a prominent role in the development of QA technology. After a prototype was demonstrated in 2007, in 2011, D-Wave introduced its first commercial QA system, the D-Wave One [38]. Subsequent generations of featured improvements in qubit count, connectivity, and overall performance, with Advantage, reaching 5000 qubits [25], which outperforms the previous iterations of D-Wave machines in almost all metrics except for very sparse problems [39].

QA involves mapping an optimization problem onto a physical system composed of qubits. They represent the variables of the problem, and their states are manipulated to explore different configurations. The energy profile of any physical system is defined by a function called Hamiltonian. In the case of QA, the Hamiltonian is the sum of an initial Hamiltonian and a final Hamiltonian. The initial Hamiltonian is formulated to be very simple and with a known lowest energy state (ground state). The system's state is initialised at the ground state of the initial Hamiltonian, which is then annealed such that the final Hamiltonian, which represents the target optimization problem, dominates in the overall system. If the evolution is slow enough and there is minimal thermal noise, the final state of the system, which corresponds to the desired solution of the optimization problem, will be in the ground state of the final Hamiltonian:

$$H(t) = (1 - s(t))H_i + s(t)H_f \tag{2.2}$$

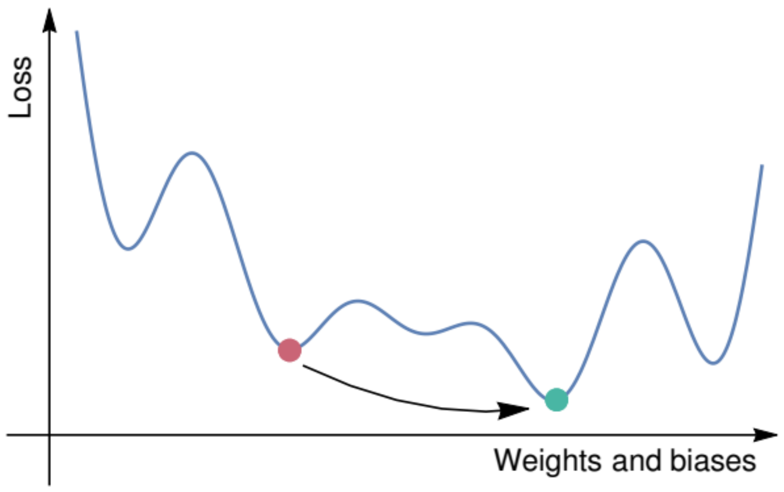


Figure 2.1: The idea behind QA training compared to classical training. Theoretically QA can tunnel from any other minimum to the global one. Image from [26]

$$H_i = \sum_n \sigma_{n,x} \tag{2.3}$$

$$H_f = \left(\sum_n \sigma_{n,z} + \sum_{n,m} J_{n,m} \sigma_{n,z} \sigma_{m,z} \right) \tag{2.4}$$

Where H_i is the initial Hamiltonian, H_f is the final Hamiltonian, $\sigma_{i,x}, \sigma_{i,z}$ are Pauli gates applied on qubit i, $J_{n,m}$ are couplings between qubits n and m and $s(t)$ is the annealing schedule. $s(t)$ increases from 0 to 1 as t goes from 0 to the maximal time chosen for the annealing. The adiabatic theorem states that "a physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum" [40]. So if our system starts at the ground state of H_i and the evolution of the Hamiltonian is slow enough to satisfy the requirements of the theorem, the system will end up in the ground state of H_f. In practice, many real-world problems have an infinitely small gap and therefore the evolution of $s(t)$ needs to be infinitely slow. In such cases, even small perturbations can get the system out of the ground state. This means that for most relevant problems real quantum annealers have a non-zero probability to end up in a state with higher energy than the ground state.

We can calculate the energy of the Hamiltonian and minimize it in order to find the ground state, by expressing it as a QUBO (see eq. (2.1)):

$$\min \quad f(\mathbf{q}) = \left(\sum_n Q_{nn} q_n + \sum_{n<m} Q_{nm} q_n q_m \right) \quad = \quad \min \quad \mathbf{q}^T \mathbf{Q} \mathbf{q} \tag{2.5}$$

Where $\mathbf{q} = [q_1, q_2, \ldots, q_N]$ is a quantum state, where each $q_n, q_m \in \{0, 1\}$, $\mathbf{Q} \in \mathbb{R}^{N \times N}$, and Q_{nm} is the element at the n-th row and m-th column of the QUBO matrix $\mathbf{Q}$. This is the only way to map a problem on a quantum annealer which is one of the main drawbacks of QA, as it limits the representation power of this technology.

The fundamental advantage of QA lies in its potential to efficiently explore complex solution spaces, particularly for optimization problems with a large number of variables or a rugged landscape of potential solutions. It has been applied to different optimization problems in areas such as finance, logistics, cryptography and drug discovery [41]. Efforts are ongoing to identify problem domains where QA excels and showcases a practical advantage over classical algorithms. However, while QA holds great promise, further research and development are necessary to fully exploit its capabilities and explore its practical implications [41].

2.1.2 Simulated Annealing (SA)

SA is a probabilistic technique for approximating the global optimum of a given function. It is a metaheuristic that helps to approximate the global minimum in a large search space for an optimization problem. It was introduced in 1983 in [42]. It draws inspiration from the cooling processes in metallurgy. The main idea is to overcome local optima and improve the likelihood of finding a global optimum by allowing "bad" moves early in the search but reducing their probability as the algorithm progresses. The amount of "bad" moves which are allowed is controlled by a parameter called temperature, which is dependent on the annealing time.

The hill climbing algorithm is a simple heuristic that looks at all neighbouring states and moves there if that solution is better than the current solution. SA is a direct improvement over such heuristics, as it significantly decreases the probability of getting stuck in a local optimum, by iteratively exploring the solution space and gradually decreasing the temperature which increases the probability of approximating the global minimum [42]. The idea behind this approach can be seen in fig. 2.2.

The main way to achieve this is by using a stochastic sampling method called Monte Carlo sampling [44], developed in 1953:

$$P(\text{accept}) = \exp\left(-\frac{\Delta E}{T}\right) \tag{2.6}$$

where $P(\text{accept})$ is the probability of accepting the candidate solution, ΔE is the change in energy between the current solution and the candidate solution and T is the current "temperature" parameter, controlling the level of randomness in the search. The temperature T can be decreased in many ways, called annealing schedules, but it is done usually either linearly or exponentially.

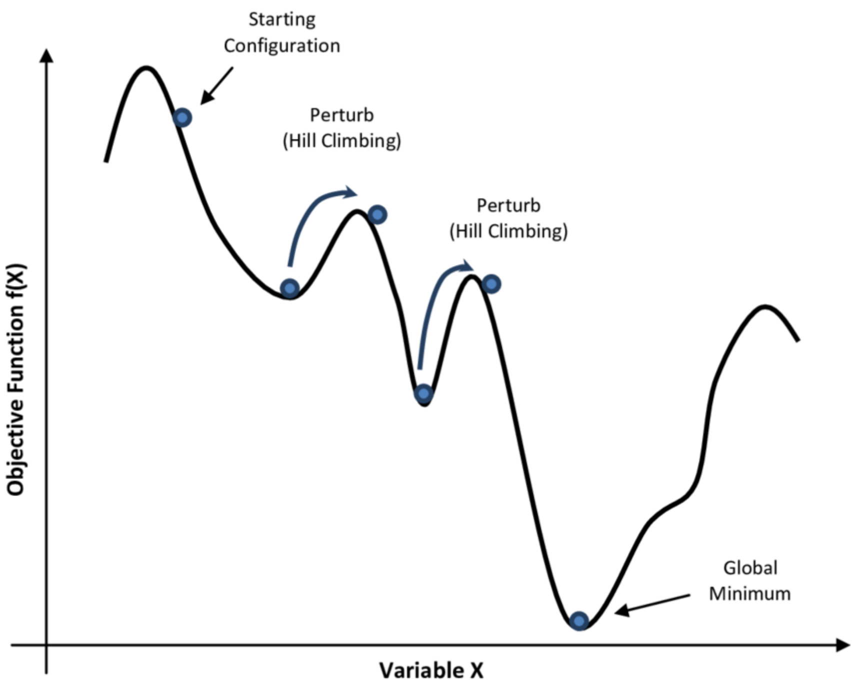


Figure 2.2: SA explores the objective function avoiding local minima and has a high probability of finding the global minimum. Image from [43]

Digital Annealing (DA)

DA is a new computing technology that Fujitsu Laboratories started researching and developing in the middle of the last decade. It is used for solving combinatorial optimization problems, which are difficult to solve with existing general-purpose computers. The digital annealer is a quantum-inspired classical technology that can be understood as accelerated SA. Where quantum computers repeatedly apply unitary operations to multiple qubits, the digital annealer repeatedly applies stochastic transitions to classical bits which correspond to the aforementioned binary values [34]. In order to accelerate this iterative operation of stochastic transitions for digital circuits, the latest generation model features an architecture that performs parallel computations of subtraction operations for the finite differences of the energy function and its stochastic evaluation.

DA solves combinatorial optimization problems mapped to an Ising model that is expressed by the energy function similarly to QA. The spin states $\{-1, +1\}$ are converted into binary values $\{0, 1\}$ and the digital annealer can rapidly solve those problems by searching for a ground state, based on the Markov-Chain Monte Carlo method [34]. Although it operates on classical hardware without relying on quantum effects like superposition and entanglement, Fujitsu's DA quantum-inspired technology offers the potential to provide efficient solutions to large-scale optimization problems which are formulated as a QUBO. This can prove beneficial for a problem like the training of a NN which requires high numbers of variables. It is also helpful for the purposes of research as there are no decoherence errors to take into account and no quantum noise. Furthermore, many of the parameters are tunable, such as the maximal annealing time, the number of solutions returned, the initial solution, the annealing schedule and many more. Additionally, the version that we used allows for one-hot encoding of penalties and auto-tuning for the factor applied to the penalties

[34]. This offers great flexibility for testing and benchmarking. For our experiments, we have not changed most of the parameters except for the maximal annealing time, in order to find the desired global minimum of our QUBO, essentially using the digital annealer as a black box. This was done because we wanted to minimize the effects of different setups for the DA on the solutions and focus on the quality of the formulation itself.

2.2 Neural Networks (NN)

A NN is a computational model inspired by the structure of the human brain. It consists of multiple layers of interconnected nodes called neurons which process and transmit information. These networks are used in various fields of artificial intelligence to learn patterns and make predictions. They solve complex tasks by gradually improving their performance, in a sequential process called training, where the network adjusts its internal parameters based on training data. The network's layered architecture and adaptive learning enable it to recognize and generalize patterns, making it a versatile tool for tasks such as image recognition, natural language processing and more. In the last decade, the size of NNs increased significantly both in terms of number of layers and the size of each layer. The training of these large networks was enabled by the increasing availability of large and diverse datasets.

A NN is the composition of L layers of neurons. The value for each neuron y_i in layer l can be computed as follows:

$$y_i^l = \sigma\left(\sum_j W_{ij}^l y_j^{l-1} + b_i^l\right) \tag{2.7}$$

Where W_{ij}^l is the weight connecting neuron j in layer $l-1$ to neuron i in layer l, $b_i^{(l)}$ is the bias term for neuron i in layer l and σ is the activation function.

The final output of the NN for an input x with n features is the result of the activation function applied on the last layer:

$$f(x^n) = y^L \tag{2.8}$$

Let (θ) include all the weights $W_{ij}^{(l)}$ and biases $b_i^{(l)}$ for all layers l.

The goal of training the NN is to find the optimal parameters θ that minimize a chosen loss function L over a training dataset D:

$$\theta^* = \min_\theta \sum_{(x,y)\in D} L(f(x;\theta), y) \tag{2.9}$$

where y is the label to datapoint x.

This minimization process is typically achieved through an optimization algorithm such as gradient descent. There are several types of gradient descent, including batch-gradient descent, stochastic gradient descent and mini-batch gradient descent. They all share the main idea of calculating the gradient of the loss function and then slowly adjusting the parameters θ of the NN in the opposite direction, so if we have a negative gradient the parameters should be increased and vice versa.

2.2.1 Binary Neural Networks (BNN)

BNNs are a class of NNs that have their network weights and activations constrained to binary values, either -1 and $+1$ or 0 and 1. This is the most extreme method for the quantization of NNs. Quantization is utilized mainly in order to reduce resource consumption of NNs in order to make them suitable for resource-constrained environments, including embedded systems, mobile devices and Internet of Things applications. BNNs offer advantages in terms of computational efficiency and compact model storage, [45], [46]. The memory efficiency is due to the fact that all of the inputs and parameters are represented by a single bit. This is especially important when dealing with expansive NNs. Furthermore, binary values can be manipulated using simple logic operations, such as bitwise XOR, bitwise AND and bit shifts, resulting in more energy-efficient computations compared to their non-binary counterparts. This also results in significantly faster inference, compared to classical NNs and other quantization schemes. The theoretical speed-up for XNOR-Net in [46] is calculated to be a factor of 58. Another thing to keep in mind is the fact that current hardware is optimized for floating point operations and BNNs are theorized to perform even better on specialized hardware optimized for bitwise operations [28], making them a candidate for real-time functionality, such as real-time computer vision systems or autonomous vehicles. An experiment was conducted in [45], where a 23 times speed up was achieved by a kernel utilizing specialized hardware, compared to a control unoptimized kernel. On top of this, BNNs enable higher hardware parallelism because of the efficient binary operations, which further improves processing speed and throughput.

Utility and security

The usefulness of BNNs does not end with their extreme efficiency. They offer many benefits in terms of security, compression and generalization capability. Binary weights help address privacy concerns by attenuating the sensitivity of the model to malicious attacks. Due to their binary nature, the weights hide information regarding the original training data and it has been shown that quantization helps protect from adversarial attacks [47], [48], [49]. The compact representation of BNNs permits efficient model compression, thereby facilitating easier deployment and transfer across networks characterized by limited bandwidth or storage capacity. It is also considered that BNNs exhibit greater resilience to noise and quantization errors compared to full-precision networks and it is even suggested that the added noise allows for better generalization [29].

As a downside BNNs have reduced representational capacity relative to their non-binary counterparts, potentially leading to marginal decreases in accuracy. Full-precision NNs can express a wide range of subtle variations, allowing them to capture intricate patterns in the data, whereas BNNs are restricted to the "black and white" relations of binary variables. Ongoing research endeavours focus on developing techniques to mitigate this limitation. Approaches encompass the introduction of supplementary model parameters [46] or the utilization of more advanced binary activation functions [50].

Training

One of the main challenges when it comes to BNNs is their training. Binary activation functions introduce non-differentiability, interfering with the straightforward use of traditional gradient-descent-based optimization techniques. Nevertheless, researchers have devised approximation methods, such as the STE, to enable backpropagation and gradient-based learning in BNNs. It was introduced in [51] and is a technique used during the training of BNNs to handle the non-differentiability of binary activation functions. It is formally defined as follows:

$$g = \frac{\partial L}{\partial h}, \quad h = \mathrm{bin}(f(x)) \tag{2.10}$$

Where g is the estimated gradient, L is the loss function, h is the output of a neuron, $f(x)$ is the forward pass and $bin()$ is the rounding function.

During the forward pass, the binary activation function is applied to obtain binary outputs. In the backward pass, instead of directly propagating non-differentiable gradients, the STE approximates the gradients as seen in eq. (2.10). Essentially the full-precision weights are stored in parallel with the binary weights. The full-precision ones are used for the backpropagation, while the binarized ones are used for the forward propagation. This allows the gradients to be backpropagated through the network in a meaningful way and also avoids oscillating gradients. As a result, even though the binary activation function is not differentiable, the STE provides a way to approximate gradients and continue the training process. The STE technique is one of the staple ways used in BNNs to address the challenges posed by discrete and non-differentiable operations. For a more in-depth understanding, refer to the original paper [51], which discusses the STE concept in detail.

Even so, the training of BNNs takes longer than other quantization methods as found out in [45] which can be seen in fig. 2.3, and remains arguably the most problematic quality they have. This is to be expected as multiple changes to the real-valued references in the STE are required in order to change the binary weight.

2.3 Image Classification

Image classification refers to the computational task of assigning predefined labels or classes to input images based on their visual content. It is used for many tasks such as object recognition for autonomous vehicles, robotics, manufacturing quality control, medical diagnoses, content moderation on online platforms and social media, security and surveillance, augmented reality and many more [52],[53]. It involves the development of ML models that can effectively recognize and differentiate between various objects, scenes, or patterns depicted in images, enabling accurate classification. Formally it can be defined as developing a predictive model that finds a mapping:

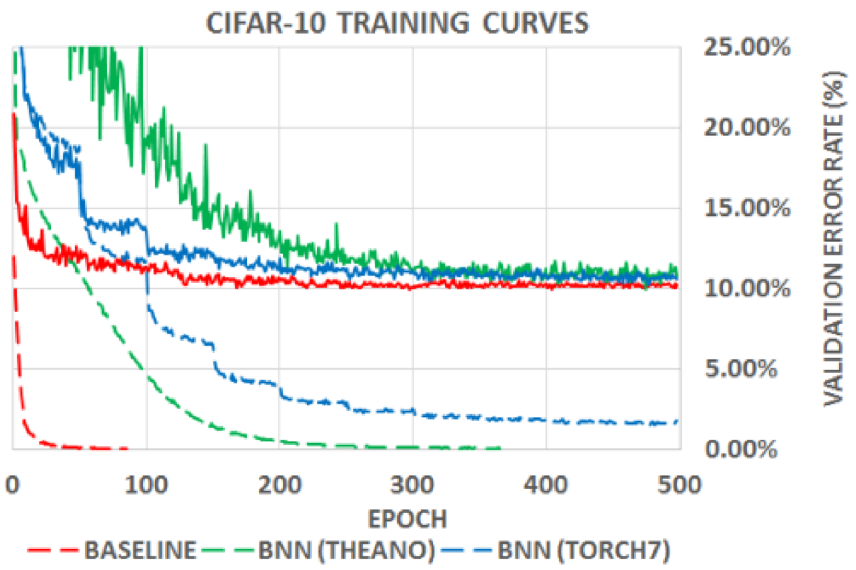


Figure 2.3: Training curves of a BNNs showing that they are slower to train but nearly as accurate as 32-bit float NNs [45]. Dotted lines represent the training costs and the continuous lines represent validation error rates. Image taken from [45]

$$f : X \to Y \tag{2.11}$$

where $x_i \in X$ is an image, represented as a matrix of pixel values and $y_i \in Y$ is the corresponding label.

While traditional ML techniques have been used for image classification, such as SVMs, Decision trees, Naive Bayes and Random Forests, the main improvements in accuracy and performance were led by the development of deep learning. In this work, we are only going to focus on image classification utilizing NNs.

2.3.1 History

In 1998 LeNet [54] was introduced, which was one of the first CNNs architectures, designed specifically for handwritten digit recognition. It showcased the successful application of deep learning techniques for image classification tasks. LeNet's architecture included convolutional layers, pooling layers, and fully connected layers, which proved to be highly effective for pattern recognition in images and inspired a lot of similar architectures.

One of the main breakthroughs utilizing deep learning came in 2012, with the introduction of AlexNet [55], a deep CNN architecture that significantly outperformed existing methods on the ImageNet dataset, which is one of the most complex benchmark datasets for image classification. AlexNet is most famous for the use of the ReLU activation function and GPU acceleration.

VGGNet [56], developed in 2014, demonstrated that increasing the depth of the network led to improved accuracy. With its 16- and 19-weight layers, it set new benchmarks on various image classification tasks and provided insights into the importance of network depth in deep learning models.

ResNet [57] addressed the problem of degradation in deep NNs. By utilizing skip connections and residual blocks, ResNet enabled the training of deeper networks without suffering from the vanishing gradient problem, which allowed for even deeper architectures to be developed and also significantly improved the accuracy of the classification.

After constant improvements and rising popularity NNs, the focus shifted to trying to widen their applications. In 2017 MobileNet [58] was introduced, which is a lightweight CNN architecture designed for mobile and embedded devices. MobileNet utilized depthwise separable convolutions, which significantly reduced the computational cost and model size without sacrificing accuracy. It can be argued that [58] popularized the concept of efficient models for resource-constrained environments.

With the rising popularity of quantum computers, there has also been a lot of research focusing on image classification such as the development of the Quantum convolutional neural network (QCNN) [13] and utilizing quantum circuits for the training of classical NNs [12] for medical image classification.

Furthermore, image classification has been used as an important benchmark for the development of BNNs,[45], [46]. For more information about the development of BNNs refer to section 2.2.1.

2.3.2 MNIST

The MNIST dataset is a widely employed benchmark dataset for image classification in ML and computer vision [59]. It comprises grayscale images of handwritten digits, ranging from 0 to 9, with each image having dimensions of 28x28 pixels. It contains 60,000 training samples and 10,000 testing samples. It offers a diverse set of writing styles, different digit shapes, and variations in stroke thickness. The dataset provides a balanced distribution of digits, ensuring an equal number of examples for each class. It is one of the simplest datasets and provides a good benchmark for new approaches such as the one presented in this work. It is extensively studied and therefore provides established baselines that can serve as references and guidelines for the performance of the tested models. It was also chosen because it offers one of the smallest but still realistic image sizes. Having a smaller number of features allows us to formulate our model while still utilizing a few images for the training, especially in the context of the polynomially scaling QUBO sizes when training BNNs [26],[33].

2.4 Sentiment Analysis

Sentiment analysis, also referred to as opinion mining, is a computational technique that involves employing ML algorithms to categorize the sentiment or emotional orientation expressed in textual data, such as customer reviews, social media posts or survey responses. It is an important part of NLP, which has many use cases such as social media monitoring, political analysis, customer feedback analysis, brand monitoring and market research. Its primary objective is to automatically ascertain and classify the subjective information conveyed within the text into positive, negative or neutral sentiments. In this work, in order to reduce complexity the focus is solely on classifying positive and negative reviews. Usually, sentiment analysis enables the extraction of valuable insights concerning the opinion of the person who wrote the text by analysing its full content and training on vast amounts of data, so in the context of limited resource environments, it should be a challenging classification problem. Formally it can be defined as developing a predictive model that finds a mapping:

$$f : X \to Y \tag{2.12}$$

where $x_i \in X$ is a text and $y_i \in Y$ is the corresponding binary sentiment label.

One popular approach to sentiment analysis is the bag-of-words model, which represents text as a collection of individual words, disregarding grammar and word order. The bag-of-words approach relies on the assumption that the overall sentiment of a document can be inferred from the frequencies or presence of certain words or phrases within it. The simplest bag-of-words model is a n-hot encoding for the vocabulary, signifying either the presence or absence of a word, losing the information about how many times it has been mentioned. This is a simple yet effective method for sentiment analysis, which was chosen for this work because it allows for a small vocabulary size and a binary input vector, which in turn drastically reduces the size of the BNN and the QUBO. Due to the limited resources available, this is essential in order to have at least a few datapoints to train on, while also keeping the problem small enough to be solvable on current quantum annealers.

2.4.1 History

In 2002 an unsupervised learning approach to sentiment analysis was introduced in [60]. This work laid the foundation for using statistical measures to extract sentiment from textual data.

In [61] supervised ML techniques were used for sentiment analysis. In the paper a dataset of movie reviews was introduced, which had sentiment labels and were used to evaluate various ML algorithms including Naive Bayes, SVMs and Maximum Entropy.

In [62], in which the IMDb dataset was introduced, the effectiveness of deep learning in capturing sentiment information was highlighted. The authors trained Recurrent neural networks (RNNs) and CNNs on this dataset and achieved state-of-the-art results in sentiment analysis. This work paved the way for subsequent research in sentiment analysis and contributed to the development of more sophisticated NN models for this task.

In [63], Recursive Neural Tensor Networks for sentiment analysis were developed. These are RNNs that capture compositional meaning in sentences by analysing the hierarchical structure of sentences, allowing for more nuanced sentiment analysis. This model outperformed previous state-of-the-art methods on sentiment analysis tasks, showcasing the effectiveness of NNs for capturing sentiment compositionality.

In [64] a Gated RNN model was proposed for sentiment analysis at the document level. It incorporated gated units to capture long-term dependencies in sequential data, enabling it to model the sentiment of an entire document. The authors evaluated the model on multiple sentiment analysis datasets, including Twitter sentiment classification, achieving competitive performance.

The field of quantum NLP is in its very early stages of development but there have been some resent developments in building Long Short-Term Memory networks such as [15] and [65] which are well suited for tasks such as text classification and sentiment analysis.

And similarly to the task of image classification, there has been a lot of research concentrated for compressing NLP models in order to make them more viable for resource constraint environments. As early as 2016 there were proposed Long short-term memory (LSTM) implementations [66], where binarizing the weights was achieved with very good results but binarizing both weights and inputs proved challenging. In [67], where a 2-4 bit representation was used for the weights and activations, very good accuracy was observed for NLP and in [68] state-of-the-art results were achieved while using the binarized representations of all the components of a text classification architecture.

2.4.2 IMDb

The IMDb Movie Review Dataset is a collection of movie reviews sourced from the IMDb website. Each review is accompanied by a sentiment label indicating whether the sentiment expressed in the review is positive represented by 1 or negative represented by 0. It is simple with small sizes of the reviews and it has a specific domain that allows for a simple bag-of-words approach with a very small vocabulary and n-hot encoding to be effective, which makes it a good match for our research purposes.

The dataset was initially introduced by Maas et al. in [62] and has since become a standard benchmark for evaluating sentiment analysis algorithms. This allows for comparison against established results and general guidelines. It consists of 50,000 movie reviews, with 25,000 reviews labeled as positive and 25,000 reviews labeled as negative. This balanced distribution ensures an equal representation of positive and negative sentiments, enabling objective evaluation and comparison of sentiment analysis models.

3 Related work

There is very limited literature on the specific problem of formulating the whole training of a BNN as a QUBO. In [33] the authors introduce the formulation which we are improving upon and in [26] a fundamentally different approach is suggested, which we will briefly explain in this section.

The focus of this approach is on utilizing non-trivial activation functions and training on a dataset that can easily be larger than the number of qubits on any device currently in existence. This is achieved by calculating the loss function for the whole NN over an approximated activation function for each of the neurons, over the whole training dataset, which results in a big high-degree polynomial that represents the whole training process for the BNN. After that, they remove terms that are not quadratic by introducing auxiliary variables (see section 4.2.3 eq. (4.12)).

This approach allows for "one-shot" training over large amounts of training data. However, it is very hard to utilize data that has a large number of features or has not been through significant preprocessing. As stated in one of the examples, where a quadratic approximation of the RELU activation function is used, the number of additional auxiliary variables that need to be introduced is:

$$(N_h+1)(N_f+1)+(N_h+1)(N_f+1)^2 \tag{3.1}$$

[26]

Where N_h is the number of hidden neurons and N_f is the number of input features. The authors report that the number of auxiliary variables scales geometrically with the degree of the term that is being reduced, which depends on the activation function. For example, there is an approximation of the RELU activation function which is 4th degree and there is an approximation of the sigmoid activation function which is 3rd degree.

4 QUBO modelling for training Binary neural networks (BNN)

The usual way of training a NN described in section 2.2 is to perform forward propagation with the initial weights and biases, then calculate the loss function for the given input data and adjust the parameters during the backpropagation using gradient descent. This process is repeated until the training is finished. Gradient descent cannot be used on binary parameters so for a BNN the training is usually done via a method called STE (see section 2.2.1). Now we are going to describe the model presented in [33] for training a BNN without backpropagation, gradient descent or an STE, achieved by formulating the problem as a QUBO and then finding an optimal or close to an optimal setup for the parameters. We are going to present the original scheme as we have implemented it, so the ordering of the variables in the solution vector and the names of some functions and variables are different, but the definitions are unchanged from the original formulation.

4.1 Definition of the Binary Neural Network (BNN)

The authors do not use biases for the formulation which also aligns with other classical BNN models such as [46]. They can be included as additional terms but this will add additional complexity and later on, could interfere with our objective to reduce the size of the QUBO and allow for bigger feature vectors.

For the BNN a simple activation function was chosen in order to introduce non-linearity in the easiest and most efficient way possible. The activation the authors chose is the sign function, which for $x \in \mathbb{R}$ is denoted as $\operatorname{sgn}(x)$ and is defined as:

$$\operatorname{sgn}(x) = \frac{x}{|x|} \tag{4.1}$$

For a BNN with binary weights $W^l \in \{-1,1\}$, inputs $y^l \in \{-1,1\}$ and layers $l \in [0,1,2,...L]$, the output of each neuron is computed as the sign of the weighted sum of inputs multiplied by the weights:

$$y_j^{l+1} = \operatorname{sgn}\left(\sum_i W_{ij}^l y_i^l\right) \tag{4.2}$$

where y_j^{l+1} is the output of neuron j, y_i are the inputs from previous layer neuron i, W_{ij} is the weight connecting neuron i to neuron j.

And the output of the whole network looks like this:

$$y^L = \text{sgn}\left(\sum W^L \text{sgn}\left(\sum W^{L-1} ... \text{sgn}\left(\sum W^0 y^0\right)\right)\right) \tag{4.3}$$

For easier notation, in [33] the authors define the product of the weights and inputs for the layers l and $l+1$ as follows:

$$Z_{ij}^{l+1} = W_{ij}^l y_i^l \tag{4.4}$$

For the loss function the simple $0-1$ loss function is chosen, which is defined as follows:

$$L_{0-1}(y^L, l) = \begin{cases} 0 & \text{if } y^L = l \\ 1 & \text{if } y^L \neq l \end{cases} \tag{4.5}$$

where L is the loss, l is the true label and y^L is the predicted output.

Then the loss is formulated as a quadratic function and squared in order to avoid negative loss:

$$L = (y^L - l)^2 \tag{4.6}$$

4.2 Model for one datapoint

We will first explain some of the important elements and intuitions for building the QUBO from [33] and in the rest of the section we are going to explain in detail how to construct the QUBO matrix for a BNN that has 1 input layer, 1 hidden layer and a single output neuron for binary classification.

4.2.1 Prerequisites

One thing, which should be noted for the training scheme from [33] is the switching from binary notation $\{-1, 1\}$, which is also called spin notation in the literature [33], [26], referencing quantum spins, to the binary notation $\{0, 1\}$ which is also known as logical bits/qubits as it is the one used for QUBO formulations [27]. When the spin notation is used for the weights it denotes a positive correlation via a "spin up" - (1) and a negative correlation via a "spin down" - (-1). The two notations can be used interchangeably and can be both used throughout the whole formulation, but it is often the case that swapping between the two grants a simpler mathematical formulation.

When swapping between the notations, the conversion of a spin variable to a binary $\{0, 1\}$ variable or vice versa is done as follows:

$$q = \frac{s+1}{2}, \quad q \in \{0, 1\}, \quad s \in \{-1, 1\} \tag{4.7}$$

When swapping from the $\{-1,1\}$ notation to the $\{0,1\}$ notation, the multiplications of the weights and inputs in the sign function need to be converted to XNOR operations. This can be illustrated as follows:

$$
\begin{aligned}
Z = W * y: \quad & W = 1, \quad y = 1, \quad Z = 1, \\
& W = 1, \quad y = -1, \quad Z = -1, \\
& W = -1, \quad y = 1, \quad Z = -1, \\
& W = -1, \quad y = -1, \quad Z = 1
\end{aligned}
$$

$$
\begin{aligned}
Z = \mathrm{XNOR}(W,y): \quad & W = 1, \quad y = 1, \quad Z = 1, \\
& W = 1, \quad y = 0, \quad Z = 0, \\
& W = 0, \quad y = 1, \quad Z = 0, \\
& W = 0, \quad y = 0, \quad Z = 1
\end{aligned}
$$

The sign function applied over the sum of the bits, has to be converted to a counting operation for the ones, which establishes a majority. This can be achieved by looking at the most significant bit of the binary sum in the $\{0,1\}$ notation, with the restriction that the bits which are being counter are one less than a power of 2:

$$
\operatorname{sgn}\left(\sum_{i=0}^{2^n-1} a_i\right), \quad a_i \in \{-1,1\} \iff \operatorname{msb}\left(\operatorname{bin}\left(\sum_{i=0}^{2^n-1} a_i\right)\right), \quad a_i \in \{0,1\}, \quad n \in \mathbb{N} \tag{4.8}
$$

This can be illustrated by the use of a simple example for the addition of seven bits. The possible sums expressed as three bits in binary notation are $\{000,001,010,011,100,101,110,111\}$. Here we can see that if the majority of bits are ones, 4 in our case, the most significant bit of the sum is 1 and if the majority of bits are zeros the most significant bit of the sum is 0. What happens here is that we "count" how many bits are one and then represent the number in base two. After that, the most significant bit represents the majority.

4.2.2 Objective function

First, the simple squared loss function needs to be incorporated in the QUBO which is defined as follows:

$$
L = (y^L - l)^2 \tag{4.9}
$$

Where l is the true label, y^L is the prediction of our BNN.

Because this is a QUBO formulation for a binary classification task, there is only one neuron per datapoint that is needed for the classification. Consequently, the objective function is only applied to a single qubit in the QUBO per datapoint.

With only the objective function included our QUBO matrix Q looks like this:

$$\min_{q\in\{0,1\}} Q = ({a_L}^1 - l)^2 \tag{4.10}$$

Where ${a_L}^1$ is the output neuron and l is the true label.

4.2.3 Penalty Terms

First, a penalty term needs to be introduced, ensuring the multiplications in QUBO represent the multiplications in the BNN correctly. As discussed in section 4.2.1 XNOR operations have to be used for that. In order to enforce the correctness of $q_3 = \text{XNOR}(q_1, q_2)$ the following penalty term is used, where $P \in \mathbb{R}^+$:

$$P_{XNOR}(q_1, q_2, q_3) = P(1 - q_1 - q_2 - q_3 + 2q_1q_2 + 2q_2q_3 + 2q_1q_3 - 4q_1q_2q_3) \tag{4.11}$$

Because there cannot be any cubic terms in a QUBO, additional variables b need to be introduced, to allow the cubic term $q_1q_2q_3$ to be replaced by the quadratic bq_3, where $b = q_1q_2$. This is enforced by using the following penalty:

$$P_{cubic}(q_1, q_2, q_3, b) = bq_3 + P(3b + q_1q_2 - 2q_1b - 2q_2b) \tag{4.12}$$

Here, q_1, q_2, and q_3 are binary variables, and b is an auxiliary variable in the QUBO problem.

These two penalties are applied to every multiplication in the model:

$$Z_{ij}^{l+1} = \text{XNOR}(W_{ij}^l, a_{iL}^l) \tag{4.13}$$

In order to better illustrate the ordering of the binary variables in the QUBO, we collate them in a binary vector q:

$$\begin{aligned} q = ({Z_{00}}^0, {Z_{01}}^0, \ldots {Z_{nm}}^0, {a_{0L}}^0, {a_{1L}}^0 \ldots {a_{nL}}^0, {W_{00}}^0, {W_{01}}^0 \ldots {W_{mn}}^0, b_00^0, b_01^0 \ldots b_nm^0, \\ {Z_0}^1, {Z_1}^1, \ldots {Z_n}^1, {a_L}^1, W_0^1, W_1^1 \ldots W_n^1, {b_0}^1, {b_1}^1 \ldots {b_n}^1) \end{aligned} \tag{4.14}$$

where a_{nL}^l are the values for the neuron n in layer l and n is the number of hidden neurons, while m is the number of features.

After adding both of these penalties to the QUBO, it looks as follows:

$$\min_{q\in\{0,1\}} Q = \left({a_L}^1 - l\right)^2 + \sum_{i,j,l} P_{XNOR}\left({a_i}^l, {W_{ij}}^{l+1}, {Z_i}^{l+1}\right) + \sum_{i,j,l} P_{cubic}\left({a_i}^l, {W_{ij}}^{l+1}, {Z_i}^{l+1}, {b_i}^{l+1}\right) \quad (4.15)$$

Next, penalty terms are introduced for ensuring the additions in the QUBO represent the additions in the forward pass for the BNN correctly. As explained in section 4.2.1 the most significant bit of the binary notation is observed in order to enforce the addition of the binary inputs in each neuron. Here is an example of counting three bits by using two:

$$P_{add}\left(a, a^{out}, q_0^{in}, q_1^{in}, q_2^{in}\right) = P\left(-a - 2a^{out} + q_0^{in} + q_1^{in} + q_2^{in}\right)^2 \quad (4.16)$$

where a, a^{out} is the representation of a number in base 2, a^{out} being the most significant bit and $q_0^{in}, q_1^{in}, q_2^{in} \in \{0,1\}$.

In the general case, if the number of inputs of a neuron is restricted to values $2^n - 1$ and a are auxiliary bits, the penalty term, as described in [33] for the activation of the sum becomes:

$$P_{add}(a_i, q_k) = P\left(-\sum_{i=0}^{n-1} 2^i a_i + \sum_{k=0}^{2^n-1} q_k\right)^2 \quad (4.17)$$

After adding the penalty terms for the addition, the binary vector q is complete:

$$q = ({Z_{00}}^0, {Z_{01}}^0, \ldots {Z_{nm}}^0, {a_{00}}^0 \ldots {a_{0L}}^0, {a_{10}}^0 \ldots {a_{nL}}^0, {W_{00}}^0, {W_{01}}^0 \ldots {Wmn}^0, {b_{00}}^0, {b_{01}}^0 \ldots {b_{nm}}^0$$
$$\quad {Z_0}^1, {Z_1}^1, \ldots {Z_n}^1, {a_0}^1, {a_1}^1, \ldots {a_L}^1, W_{00}^1, W_{01}^1 \ldots W_{0n}^1, {b_0}^1, {b_1}^1 \ldots {b_n}^1) \quad (4.18)$$

From it, we can calculate the number of variables required for this representation. Let us have a feature vector of size m as input and n hidden neurons. The total number of variables for one datapoint is $3nm + n\log_2 m + 3n + \log_2 n$.

In the end, the simple notation for the whole QUBO looks as follows:

$$\min_{q\in\{0,1\}} Q = \left({a_L}^1 - l\right)^2 + \sum_{i,j,l} P_{XNOR}\left({a_i}^l, {W_{ij}}^{l+1}, {Z_i}^{l+1}\right) + \sum_{i,j,l} P_{cubic}\left({a_i}^l, {W_{ij}}^{l+1}, {Z_i}^{l+1}, {b_i}^{l+1}\right) + \sum_{i,j,l} P_{add}\left({a_i}^l, {Z_i}^l\right) \quad (4.19)$$

4.3 Model for multiple datapoints

The model for a single datapoint can be extended for multiple datapoints, as described in [33], in order to have the opportunity for training over larger amounts of data. It should be noted that the weights W_{ij}^0 and W_{ij}^1 are shared between the datapoints, but all other variables are datapoint specific. So the binary vector needs to be extended for multiple datapoints as follows:

$$q = \left(\vec{Z}_0^0, \vec{Z}_1^0, ...\vec{Z}_d^0, \vec{a}_0^0, \vec{a}_1^0 ... \vec{a}_d^0, W^0, \vec{b}_0^0, \vec{b}_1^0 ... \vec{b}_d^0, \vec{Z}_0^1, \vec{Z}_1^1, ...\vec{Z}_d^1, \vec{a}_0^1, \vec{a}_1^1, ...\vec{a}_d^1, W^1, \vec{b}_0^1, \vec{b}_1^1 ... \vec{b}_d^1\right) \tag{4.20}$$

Where d is the number of datapoints and for a datapoint x: $\vec{a}_x^l$ are all variables a_{ij}, $\vec{Z}_x^l$ are all variables Z_{ij} and $\vec{b}_x^l$ are all variables b_{ij} in a layer l.

The locations where the additional variables are placed in the QUBO are not specified in the original model. The idea behind the ordering of the variables we used is that each set of variables for a datapoint is followed by the same type of variables for the next datapoint. So after the Z variables for the first datapoint follow the Z variables for the second etc., after the b variables for the first datapoint, follow the b variables for the second and so on.

5 Modelling the training of BNNs for real-world problems as a QUBO

We improved the original model in several aspects with the goal of making the formulation applicable to realistic problems such as image classification and sentiment analysis.

5.1 Structural improvements

The original QUBO can be split into an objective function matrix and a penalty matrix, which can also be weighted differently as follows:

$$Q = w_{obj}L + w_pC \tag{5.1}$$

This allows us to control how big of an importance is placed on correct classifications of the training data versus correct interactions between the neurons. This is most significant for solutions with non-zero energy as usually lower energy solutions are preferred. However, as we can see in chapter 6 it makes a difference even when focussing only on zero energy solutions.

Furthermore, we can also make the observation that the different types of penalty terms for our penalty matrix C can be classified as XNOR penalties expressed in eq. (4.11), corresponding cubic term penalties from eq. (4.12) and addition penalties explained in eq. (4.17). They can also be separately weighted by w_{XNOR}, w_{cubic} and w_{add} respectively. The general structure for the QUBO we are building is as follows:

$$Q = w_{obj}L + w_{XNOR}P_{XNOR}() + w_{cubic}P_{cubic}() + w_{add}P_{add}() \tag{5.2}$$

We opt to use different penalty factors for each penalty type as this allows us to penalize some possible incorrect solutions, which would not be penalized otherwise. We will illustrate this with a simple example of an infeasible solution. Let $Z_0^1 = 1$, $a_{0L}^0 = 1$, $b_1^1 = 1$ and everything else in our binary solution vector, including W_0^1 be 0. The relation between these variables should be:

$$Z_0^1 = \text{XNOR}\left(a_{0L}^0, W_0^1\right) \tag{5.3}$$

while b is the auxiliary variable for the cubic terms which appear in the penalty. In this case Z_0^1 should be 0 and therefore this solution should be penalized.

From eq. (4.11) and eq. (4.12) from the original formulation, we get a penalty:

$$P_{XNOR}\left(a_{0L}^0, W_0^1, Z_0^1\right) = P\left(1 - a_{0L}^0 - W_0^1 - Z_0^1 + 2a_{0L}^0 W_0^1 + 2W_0^1 Z_0^1 + 2a_{0L}^0 Z_0^1 - 4b_1^1 Z_0^1\right) + \\ + P\left(3b_1^1 + a_{0L}^0 W_0^1 - 2a_{0L}^0 b_1^1 - 2W_0^1 b_1^1\right) \quad (5.4)$$

which after we substitute our values simplifies to the following:

$$-3P + P = -2P \quad (5.5)$$

Which, regardless of the value for the penalty factor P, will lower the energy and thus be accepted as a valid solution.

If we take the same example with our suggested weights from eq. (5.2) we get:

$$P_{XNOR}\left(a_{0L}^0, W_0^1, Z_0^1\right) = w_{XNOR}\left(1 - a_{0L}^0 - W_0^1 - Z_0^1 + 2a_{0L}^0 W_0^1 + 2W_0^1 Z_0^1 + 2a_{0L}^0 Z_0^1 - 4b_1^1 Z_0^1\right) + \\ + w_{cubic}\left(3b_1^1 + a_{0L}^0 W_0^1 - 2a_{0L}^0 b_1^1 - 2W_0^1 b_1^1\right) \quad (5.6)$$

which after the substitution simplifies to:

$$-3w_{XNOR} + w_{cubic} \quad (5.7)$$

Considering that this solution should be penalized we can choose values for w_{cubic} and w_{XNOR} such that $w_{cubic} > 3w_{XNOR}$ and therefore $w_{cubic} - 3w_{XNOR} > 0$.

Additionally, this way of specifying the factors for the different penalties allows for more possibilities for customisation and optimisation, as expressed in [27]. There might be a setup for the penalty factors which improves the performance of our model. This, however, is something that we will not attempt in this work and briefly discuss in chapter 8.

5.2 Flexible number of neurons in each layer

The original scheme is restricted to allowing feature vectors and number of hidden neurons which are of the form $2^n - 1$ (see section 4.2.3 eq. (4.17)). This has the consequence that the trained BNN needs to be adapted to the QUBO formulation rather than the other way around. To solve this, we propose a change to the addition penalties, which allows for the use of any number of odd input bits and hidden neurons. The reason for having an odd number of inputs is that the addition with the sign function as an activation is essentially a majority vote and to have a clear majority, an odd number is required. The idea behind our formulation is to change the factors of the most significant bit and the bit before that in order to ensure that the only time when the most significant bit changes, is when a majority is reached and then use the bit before that as a "buffer" to compensate for the changes we made, while still allowing for all possible sums to be represented. Let us clarify by using 49 input bits as an example, which would not be possible in the original scheme. If we insert these values in eq. (4.17) we get:

$$P_{add}(a_i, q_k) = P\left(-\sum_{i=0}^{5} 2^i a_i + \sum_{k=0}^{49} q_k \right)^2 \tag{5.8}$$

The only value in the binary vector which is used as the output of a neuron is the bit a_5. If we have 25 bits that are 1, we can see that a_5 is 0 as the binary notation for 25 is 011001, not registering that majority is reached.

Our model would also use 6 bits for the addition and we keep the first four factors before the bits the same as eq. (4.17), which leads to a penalty that looks like this:

$$P_{add}(a_i, q_k) = P\left(-(+2a_1 + 4a_2 + 8a_3 + ba_4 + ma_5) + \sum_{k=0}^{49} q_k \right)^2 \tag{5.9}$$

Now we have to choose values for b and m. Formally we will define them for n input bits as follows:

$$m = \frac{n+1}{2}, \quad b = m - \sum_{i=0}^{\log_2 n - 2} 2^i \tag{5.10}$$

So in our example, for m we have to choose the value 25 as it is the majority of 49. For the buffer b, in order to still be able to represent all possible sums, we need to cover the numbers between 16 and 25 which leaves us with $b = 9$. After this, our penalty looks as follows:

$$P_{add}(a_i, q_k) = P\left(-(1a_0 + 2a_1 + 4a_2 + 8a_3 + 9a_4 + 25a_5) + \sum_{k=0}^{49} q_k \right)^2 \tag{5.11}$$

5.3 Reducing Variables

We can make the observation that when we calculate the values for the first layer of hidden neurons $\sum Z_{ij}^0$, the inputs ${y_i}^0$ are available before the annealing begins and are not affected by any of the variables which are changed in the solution vector, thus there is no need for them to be present in our QUBO vector which represents only variables. This leaves the variables Z_{ij}^0 only dependent on the weight variables W_{ij}, which allows us to pack the whole information of the Z_{ij}^0 variables into the part of the QUBO which is representing the weights W_{ij}. After that change, we need to readjust the addition to be connected to the weights as there will no longer be Z_{ij} variables. We also need to pay attention to the fact that the multiplications that we are skipping are in the binary $\{0,1\}$ notation and therefore should be represented by an XNOR.

To formalize this approach we first make the observation that the XNOR in eq. (4.13), applied to the input ${y_i}^0$ which we know, can be expressed as follows:

$$Z_{ij}^0 = \begin{cases} W_{ij}^0 & \text{if } {y_i}^0 = 1 \\ 1 - W_{ij}^0 & \text{if } {y_i}^0 = 0 \end{cases} \tag{5.12}$$

After that, the addition penalty for layer 0 described in eq. (4.17) is changed to use W_{ij} instead of Z_{ij} as follows:

$$P_{add}\left({a_{ij}}^0, {W_{ij}}^0\right) \tag{5.13}$$

After all of this is taken into account we can remove all Z_{ij}^0 variables, along with the auxiliary b_{ij}^0 variables described in eq. (4.12), which are no longer needed.

After the simplification, our binary vector, without losing any information becomes:

$$q = \left({a_{0L}}^0, {a_{1L}}^0 ... {a_{nL}}^0, {W_{00}}^0, {W_{01}}^0 ... Wmn^0, {Z_0}^1, {Z_1}^1, ... {Z_n}^1, {a_0}^1, {a_1}^1, ... {a_L}^1, W_{00}^1, W_{01}^1 ... W_{0n}^1, {b_0}^1, {b_1}^1 ... {b_n}^1\right) \tag{5.14}$$

Where n is the number of hidden neurons and m is the number of inputs.

We can calculate the number of variables required for this representation similarly to the original. If we have a feature vector of size m as input and n hidden neurons, the total number of variables required for the model with one datapoint is $nm + n\log_2 m + 3n + \log_2 n$ which is with $2nm$ less than the original. The connections between the input layer and the first hidden layer usually contain the largest number of parameters, so our reformulation helps us to significantly reduce the number of QUBO elements that are needed to represent the training. For a network with 7 inputs and 7 hidden neurons for 8 datapoints, the original scheme requires 760 variables [33] and our scheme reduces that number to 360. We performed most of our experiments with a BNN with 49 inputs and 7 hidden neurons which required just 822 qubits. This is the most significant improvement enabling us to classify image and text data as can be seen in chapter 6.

5.4 Training over time

The usual way a NN is trained is a sequential process described in section 2.2. One of the benefits of the training being formulated as a QUBO is that it can be done in a "one-shot" fashion for multiple images at the same time. It, however, is still limited in the amount of data which can be utilised. In order to learn over an extensive number of datapoints, while maintaining or improving the accuracy, we propose a hybrid scheme for training over time by utilizing multiple batches. This was left by the authors of the original formulation [33] as future work, where they were envisioning a different approach to ours, which only focuses on previous solutions which repeat exactly. In comparison, we propose a scheme that takes advantage of all previous solutions which can be the result of previous training with annealing or even from classical training. The general idea of our formulation is to keep optimal weights found during training as a reference and each time we construct a QUBO for a new batch of datapoints, to introduce a bias in that QUBO towards those weights. This bias has to be chosen carefully so that the option to settle for different, better weights is still available, but in every other case favour the old weights.

A direct approach of letting the weights of each iteration influence the next will result in weights which change a lot and are also very subjective to the order in which the batches are trained. To tackle these problems we propose having a floating point solution matrix V. After each annealing run we add the suggested weights W_{sij} of all solutions suggested by the digital annealer, multiplied by a learning rate α to the solution matrix V:

$$V_{ij} = V_{ij} + \sum_s \alpha W_{sij}, \quad s \in \{0, ..., 25\} \tag{5.15}$$

where s is a solution from the digital annealer.

This scheme is devised to work for binary weights which are in $\{-1, 1\}$ notation in order to avoid infinite scaling and also allow for the possibility for a weight in V to flip from negative to positive or vice versa.

Now that we have our reference weights V_{ij}, we can add a bias towards them the next time we construct a QUBO. For that, we use an additional penalty function, which affects only the diagonal elements of the matrix which looks like this:

$$-B \sum_{ij} W_{ij} V_{ij} \tag{5.16}$$

Where B is a factor for the bias towards the weights that are in the matrix V_{ij} containing the reference weights and W_{ij} are the weights in the new QUBO. The penalty is multiplied by -1 because the weights which are positive need to reduce the energy and the weights which are negative need to increase the energy if we want to have a bias towards finding those same weights. So our simplified notation for formulating a QUBO for training over multiple batches is:

$$\begin{aligned} \min_{q \in \{0,1\}} Q = \left(a_L{}^1 - l\right)^2 + \sum_{i,j,l} P_{XNOR}\left(a_i{}^l, W_{ij}{}^{l+1}, Z_i{}^{l+1}\right) + \\ + \sum_{i,j,l} P_{cubic}\left(a_i{}^l, W_{ij}{}^{l+1}, Z_i{}^{l+1}, b_i{}^{l+1}\right) + \sum_{i,j,l} P_{add}\left(a_i{}^l, Z_i{}^l\right) - B \sum_{ij} W_{ij} V_{ij} \end{aligned} \tag{5.17}$$

We should also add that, for general intuition, we adjust the energy constant with the same amount that we subtract from the diagonal elements in order to still have global optima at 0 energy and no solutions with negative energy. This is just for convenience and does not change the functionality of the formulation.

Here we made the interesting observation that we don't need to round off the elements in the solution matrix V, as they are stored classically and are not variables in the QUBO, meaning they don't have to be binary. We can even use the solution matrix V directly, which will somewhat approximate integer or floating point weights. This, however, would defeat the purpose of using a BNN in the first place. One more feasible precision allowed by this modelling is the use of ternary weights $\{-1,0,1\}$, which depending on the problem might be beneficial. This can be achieved by setting a boundary around 0 where we assume that the weight is undetermined giving it a value of 0 and outside of that boundary we round the weights to $\{1,-1\}$. Essentially we could get 2-bit approximations for the ideal weights without the need to enlarge the QUBO size. This however is something we have not explored in this work.

5.5 Modelling for Image Classification

For the image classification, we used the MNIST dataset and each image was reduced from $28x28$ to a dimension $7x7$ by applying $MaxPooling$ on each $4x4$ grid. $MaxPooling$ is a very popular preprocessing technique where a grid is applied over the input and only the maximal value in that grid is stored in the new matrix. Therefore it is often used in CNNs for reducing the size of the inputs. Additionally, because we are going to be binarizing the values for the inputs, there is further incentive to get as close to polarized binary values as possible in order to lose as little information as possible when rounding. After the $MaxPooling$, we tried to further increase the number of ones artificially by setting any number that passes a threshold (we use 0.25) to 1. This final step is done so that the input vector has a similar number of ones and zeros in it. We do not have any evidence that this improves or hurts the performance of our model but we wanted to create very balanced feature vectors, in contrast to the ones for sentiment analysis, in order to test our model under different conditions.

We also performed a few tests on full-scale images, but due to the amount of time needed for solving those big instances, we mainly focused on working with preprocessed data with reduced dimensions and complexity. For the full-scale images there was minimal preprocessing. The first step was removing a single bit from the bottom right corner of the image (the last bit), as it is one of the most likely bits to hold no valuable information. This was needed as our scheme only works with an odd number of features and so we were working with 783 inputs rather than 784. The last step we did was increasing the contrast by again setting any number that is larger than our threshold of 0.25, instead of the expected 0.5, to 1.

Considering that we are doing binary classification we had to choose pairs of digits to be compared. We decided on two pairs namely $0,1$ and $5,6$. We expected the former to be relatively easy to

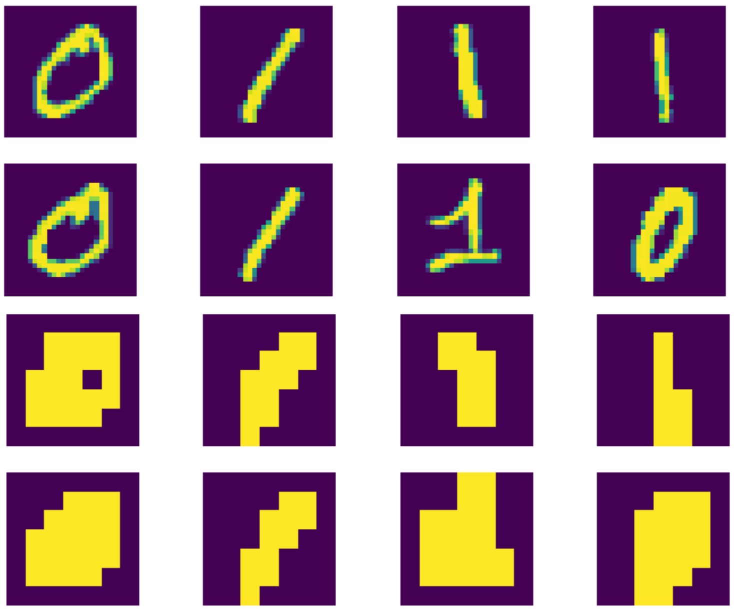


Figure 5.1: Preprocessing applied to numbers 0 and 1

classify and the latter to be challenging, again for the purpose of testing as many different scenarios as possible.

Lastly, we had to decide on a way for choosing a set of weights because the digital annealer has the option to return a few solutions, which we set to 25. We observed only the solutions with the lowest energy and then we used them to classify the training data. The best-performing weights on the training data were then tested on the test data in order to benchmark the performance of the scheme. This is not the case for our multibatch model, where all 25 solutions are utilized to influence the final optimal weights (see section 5.4).

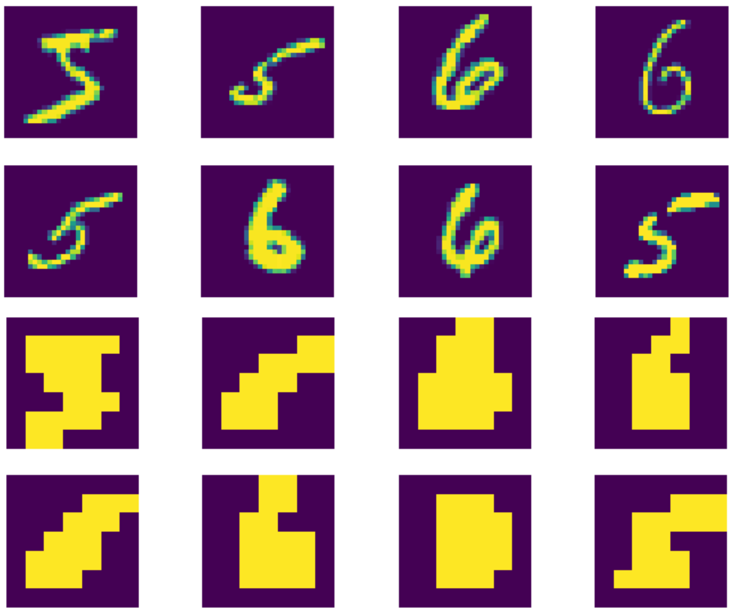


Figure 5.2: Preprocessing applied to numbers 5 and 6

5.6 Modelling for Sentiment Analysis

For the sentiment analysis, we used the IMDb dataset. Here, the preprocessing is not as straightforward as with the image classification because we have to find a way to keep the dimensionality of the input data very low, which is why we chose a bag-of-words approach. In order to select a good ensemble of words we calculated an index for every word that is found in the interviews and adjusted the bounds for it in order to control the number of words which were selected. This index is calculated by dividing the occurrences of a word in positive reviews by the occurrences in negative reviews or vice versa:

$$I_p(x) = O_p(x)/O_n(x), \quad I_n(x) = O_n(x)/O_p(x) \tag{5.18}$$

where x is a word, $O_p()$ and $O_n()$ are functions that calculate the occurrences of a word in the positive and negative reviews respectively.

We choose appropriate values for these indices, guaranteeing us that the words we choose occur often in positive but occur rarely in negative reviews and vice versa. In our case, we opted for words x which either have $I_p(x) > 3$ or $I_n(x) > 3$. Furthermore, we selected only words that appear often in order to reduce the effects of outliers. We simply considered only words x for which $O_p(x) + O_n(x) > 80$. Our next step was removing any names and very specific words from the dataset, as they are something that cannot be used in generalized cases and would make our sparse vectors even sparser. Our vocabulary after this preprocessing consisted of 203 words and can be seen in fig. 5.3. After that, when choosing reviews for training and testing we chose only reviews, which have more than 10 occurrences of words from our vocabulary in order to avoid extremely sparse or even empty feature vectors. After our preprocessing, we were left with feature vectors of length 203 which were quite sparse, and had 394 vectors for training and 363 for testing. We should note that after the preprocessing there were 126 words in the vocabulary, characteristic for negative reviews compared to the 77 characteristic for positive reviews. Also, there were more negative reviews overall in both our training and test datasets after the preprocessing. These are significantly more unbalanced feature vectors and datasets compared to the ones we used for image classification in section 5.5, which will present our formulation with a different challenge and hopefully give us more insights into its strengths and weaknesses.

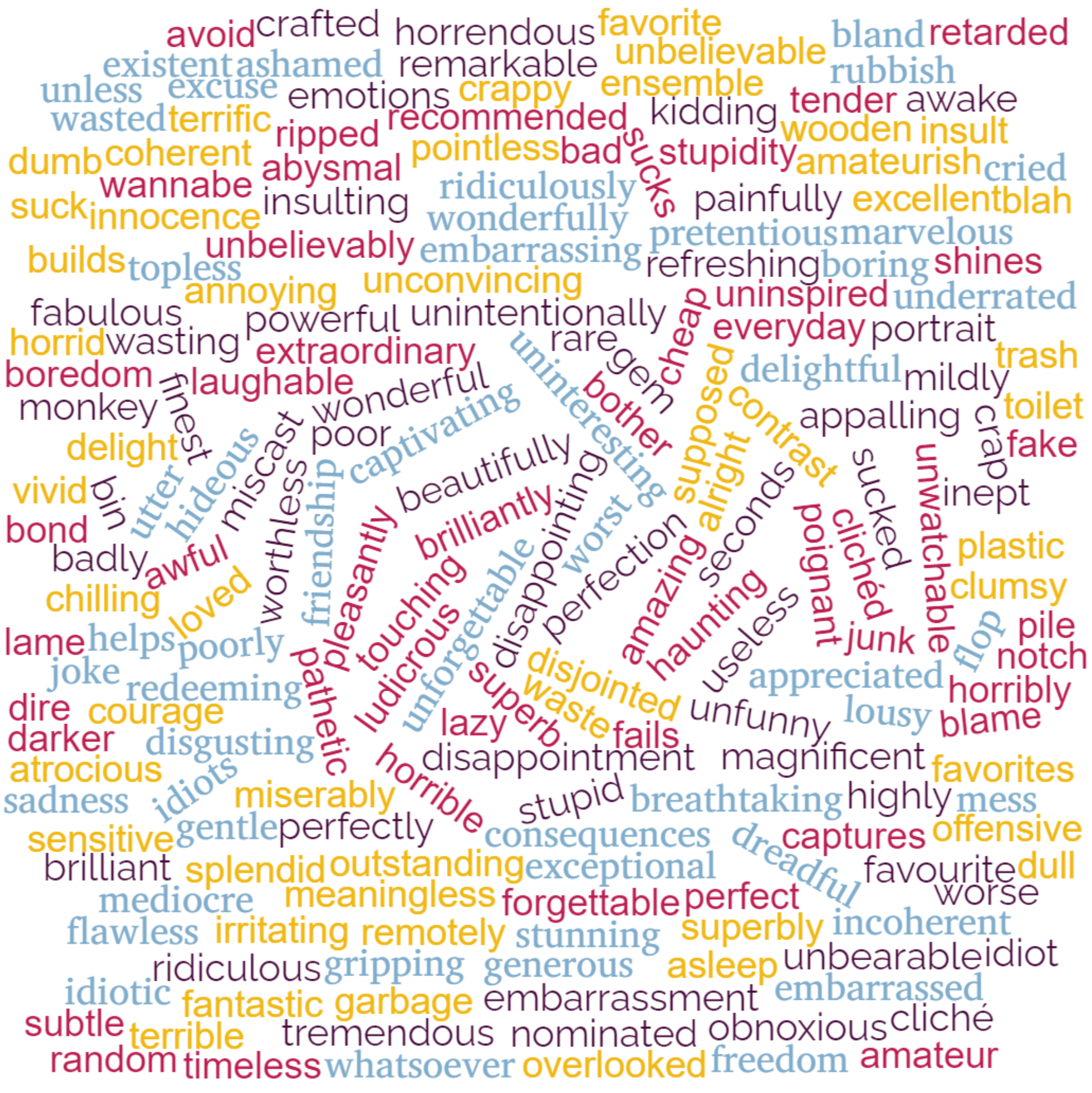


Figure 5.3: Words in our vocabulary after the preprocessing is done.

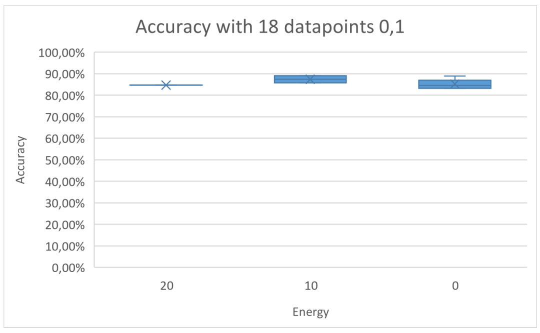
This is a very simple, hand-crafted approach and we recognise that there are many preprocessing possibilities that will hopefully produce much better results. Formulating a binary feature vector with such low dimensions is a very challenging task and, in our opinion, a very interesting avenue for future research (see chapter 8).

The same method for choosing a solution from the 25 returned by the digital annealer, which was described for image classification, is also applied here. After choosing the solutions with the lowest energy, the weights are used to classify on the training data and then the best classifier is benchmarked on the test data. This is not the case for our multibatch model, where all 25 solutions are utilized to influence the final optimal weights (see section 5.4).

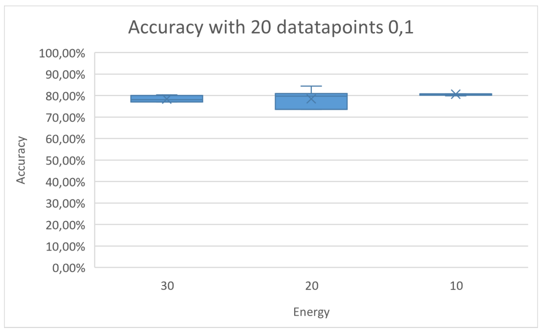
6 Experiments

In order to have a fair comparison with a classical NN we have implemented a simple Multilayer perceptron (MLP) with the same dimensions as the BNN that we are training with our QUBO. The input that we are using for the classical network is preprocessed and binarized the same way as the input data for the QUBO and BNN. Then we train the MLP with the squared loss function and the sign activation function, using classical weights which are slightly reduced to $float16$. We train it over 50 epochs and with a learning rate of 0.1. After we have finished with the training, we test the performance with classical weights in order to have a benchmark. After that, we round the classical weights to get binary weights $\{-1,1\}$ and have a second test of the performance of the classical training with binary precision, which is the closest comparison to our scheme.

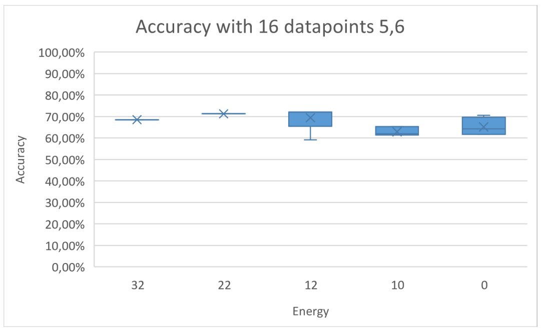
For the annealing we decided to focus on solutions suggested by the DA solver which have an energy of zero, meaning that there were no violations in both the objective function and the penalty terms. This means that all multiplications and additions in the represented BNN are executed correctly and that all of the training data was classified correctly. We can see when comparing fig. 6.1b with fig. 6.1a that with an increasing number of datapoints it gets more difficult to find solutions with no violations. Even though we cannot verify this, we strongly believe that there is a number of datapoints, which cannot have no objective function violations. For some cases when no zero-energy solutions were found, we chose the ones with the lowest energy. We observe that, generally, the quality of the solutions remains stable as long as the violations of penalty terms and the objective function are not too many as can be seen in fig. 6.1.



(a) 18 datapoints for digits 0 and 1



(b) 20 datapoints for digits 0 and 1



(c) 16 datapoints for digits 5 and 6

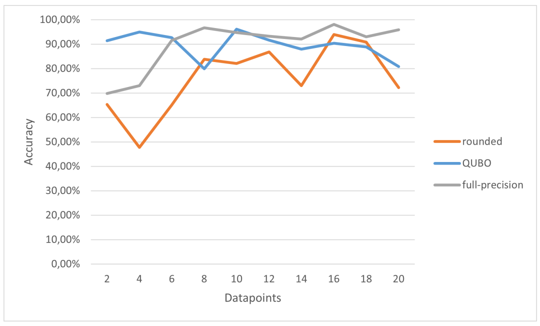
Figure 6.1: Accuracy of test samples depending on their energies with mean and standard deviation. Achieved by changing the maximum annealing time, resulting in states with different energies.

6.1 Image classification

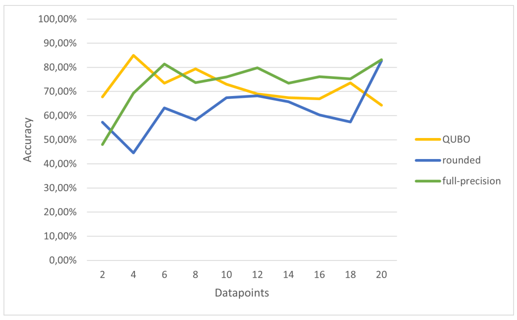
For our testing as described in section 5.5 we focused on 4 labels $0, 1$ and $5, 6$. We used the first 10000 images from the training images and all 10000 test images from the MNIST dataset. Overall, the model successfully learned to differentiate between the digits for both pairs. Depending on the problem and the setup, the average accuracy of our model was between 65% and 90%. It also achieved a maximal observed accuracy of 96%. Most of our tests were focused on observing how the model responds to different internal parameters and datasets, as most intuitions for classical ML are not applicable to such formulation.

One of the most interesting observations was how the accuracy of our model changes with the addition of new training data. For limited amounts of training data - 2, 4 and 6, our model outperforms both the classical training with $float16$ weights and the binarized weights, which can be seen in fig. 6.2. This is a very interesting observation, which might point to a niche scenario where the QUBO modelling leads to better utilization of limited data. The performance is overall similar to a binarized version of the trained MLP but tends to fall behind the full-precision MLP. In the same figures we can also see that the performance of the QUBO model is getting worse with the addition of more training data. This might be due to the higher volatility of BNNs for example.

If we use the same data but split it in small batches we get considerably better results. This can be seen if we compare the performance of our multibatch online training fig. 6.7 with fig. 6.2. These examples are both using the same 20 images of 0s and 1s and 5s and 6s. The training with 5 batches containing 4 datapoints, achieves 15% and 14% higher accuracy respectively.

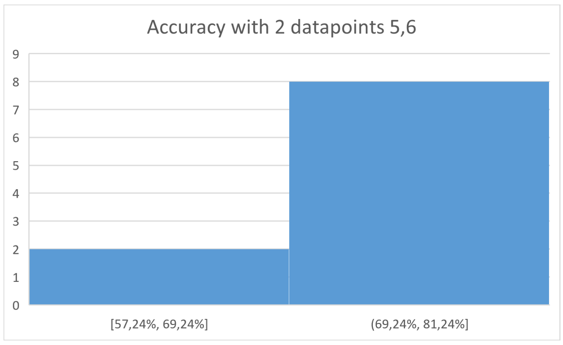


(a) For digits 0 and 1

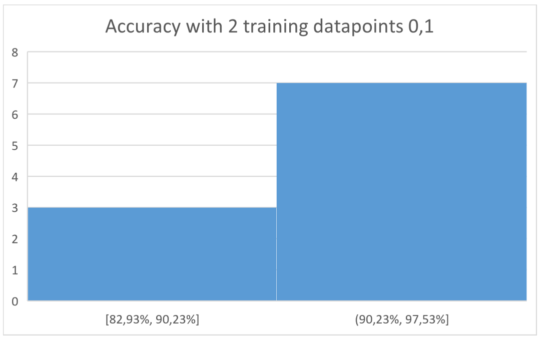


(b) For digits 5 and 6

Figure 6.2: A comparison between the accuracy of the QUBO model and that of a classical model with and without binarization on different numbers of datapoints.



(a) For digits 5 and 6

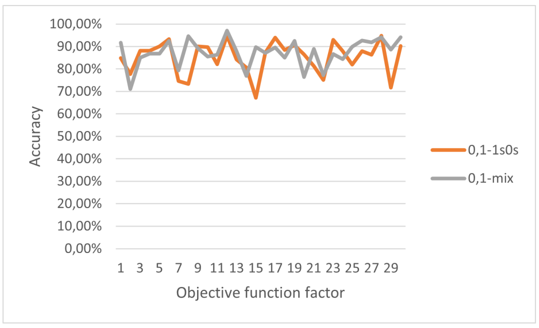


(b) For digits 0 and 1

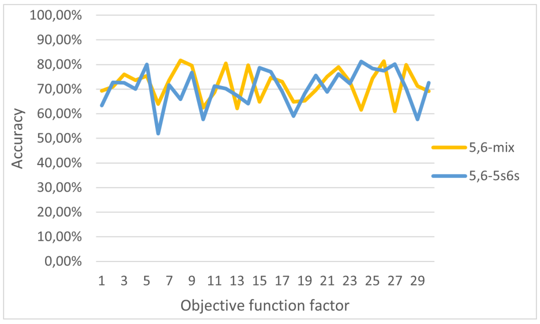
Figure 6.3: Histograms showing the how consistently the model performs for two datapoints on different randomly chosen pairs of data. There is always one datapoint of each class.

There was also a consideration for the possibility of some data being an outlier in terms of how well it represents the general problem and thus allowing for better training. After testing different pairs of data separately which can be seen in fig. 6.3 we concluded that this is likely not the case, because even though there are some differences in the accuracy, when all the data is used together these differences should be balanced out.

One last possibility we considered was the difficulty in finding a good solution for generalization. As an example for 20 datapoints finding a solution with 0 energy took approximately 1200 seconds, while for 4 datapoints each anneal requires 90 seconds or less. This suggests that with an increasing number of datapoints there are fewer and fewer solutions without any violations, which is to be expected as each datapoint increases the requirements to be met by the weights, removing some of the previously possible solutions. It might be the case that the quality of complex solutions which fit many datapoints perfectly is worse for generalization compared to the solutions achieved on a few datapoints. If this is the case, this is similar to the effects of overfitting for classical ML.

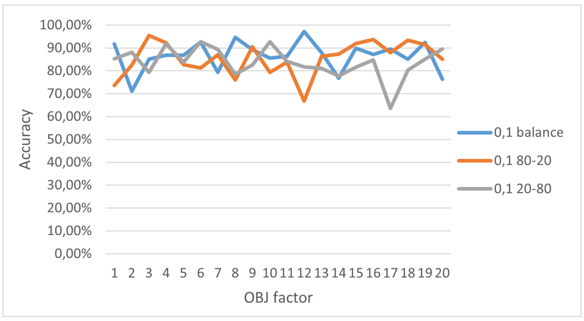


(a) For digits 0 and 1

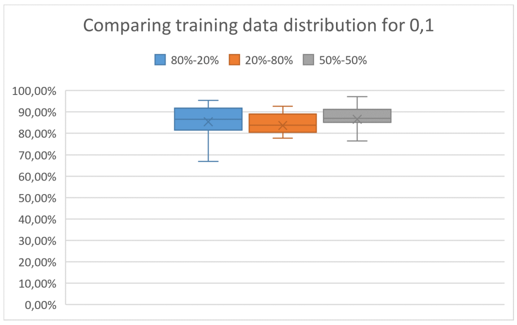


(b) For digits 5 and 6

Figure 6.4: Changes to accuracy depending on the relation between penalty factor and objective function factor. The penalty factor is 10 and we are training on 10 datapoints. Additionally we are comparing two ways to order the data in the input vector-alternating or segregated.



(a) Across different objective factors



(b) Average accuracies with mean and standard deviation

Figure 6.5: Performance depending on different proportions of the input being from one class, in this case 0 and 1. We tested on 10 datapoints.

We tested how the accuracy of our model changes depending on how the penalties and objective function are weighted in comparison to each other, represented by the penalty factor and the objective function factor described in section 5.1 in order to find an optimal setup for these parameters. What we observe in fig. 6.4 is that the values of the penalty factor and the objective function factor don't have observable best values when we compare the quality of the suggested weights. They also don't show any linear trends in terms of accuracy of the trained BNN.

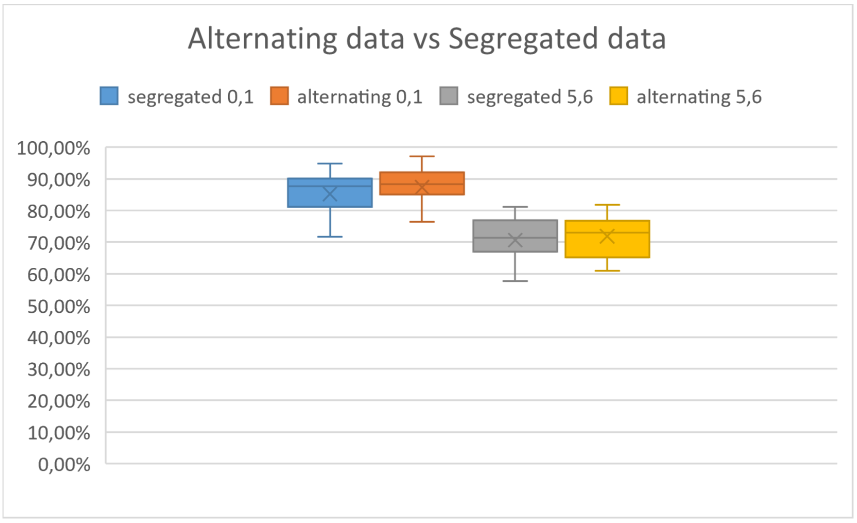


Figure 6.6: Average accuracies with mean and standard deviation achieved on 10 datapoints with different order for the input data - segregated versus alternating for classifying 5,6 and 0,1.

We also tested if there are any observable effects on the accuracy of the model depending on the way the training data is ordered, which should not be the case. There are no significant effects that would suggest a trend in the performance which can be seen in fig. 6.4 and fig. 6.6. This is a good sign because it means that training is quite stable and there are no unnecessary dependencies in the QUBO. Furthermore, having a balanced amount of data for each class versus having predominantly data from one class was investigated and seems to have no significant negative effects which can be seen in fig. 6.5a. After the results of the aforementioned tests, based on the small differences in the average accuracy seen in fig. 6.5b and fig. 6.6, we concluded that the optimal performance of our model is reached when we have a balanced distribution of training data which are in alternating order.

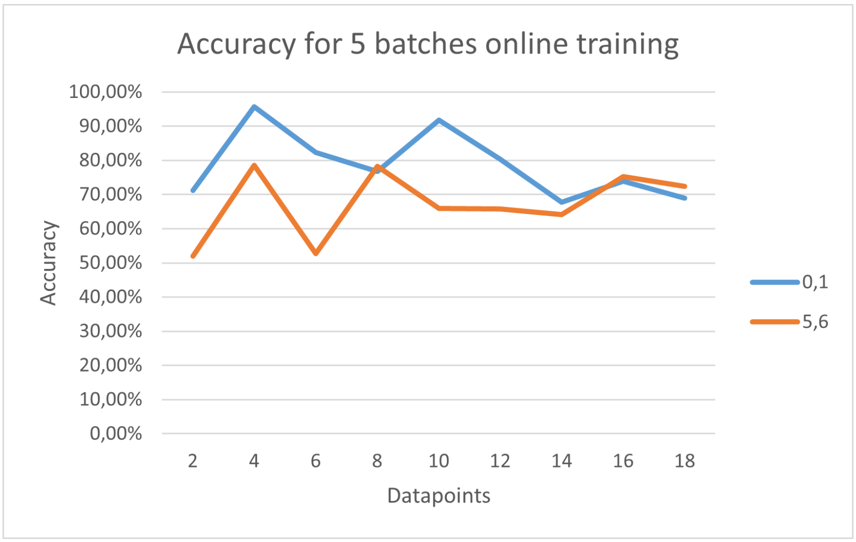


Figure 6.7: Accuracies achieved for training 5 batches with different number of input data in each batch.

The online multibatch training scheme, which was described in section 5.4, shows good performance as can be seen in fig. 6.7. However, overall, it also struggles with the problem described earlier where the accuracy starts decreasing after a certain amount of training data is used in each batch. The benefits of this scheme are twofold. First, this scheme provides a good way to create a robust solution that is not susceptible to outliers and shows good results. And second, it seems to be the case that even if we use the same training data but split it in a few batches it finds better parameters for generalization. This can be illustrated by training with 20 datapoints of 1s and 0s in one batch as seen in fig. 6.2a. If we split the same data in 5 batches we achieve a 15% increase in accuracy, which can be seen in fig. 6.7. Not only that but it takes a lower amount of annealing time to find each separate solution. The single batch takes about 1200 seconds while each of the 5 batches takes 90 seconds. Overall, this scheme is the best bet to train a model for real applications when there is no verification on the choice of good weights on vast amounts of training data as we do in section 5.5. This is due the the robustness of the achieved solutions and the efficient utilization of the solving hardware.

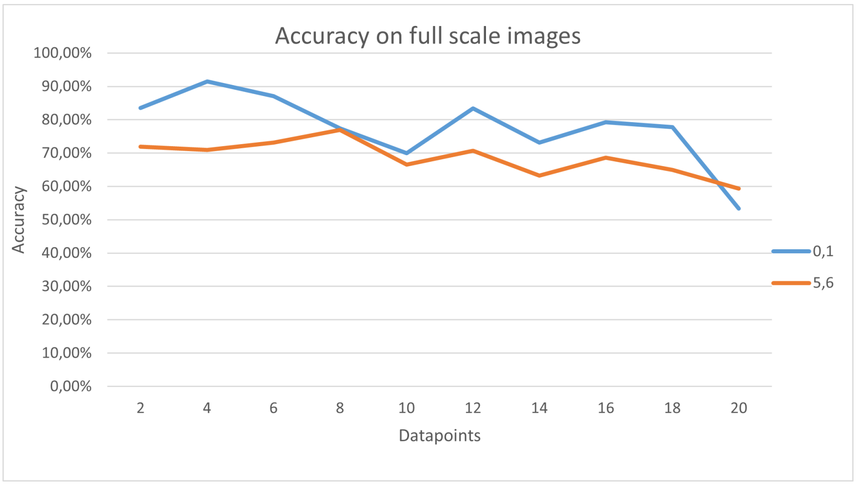


Figure 6.8: Accuracy achieved for training and testing on full-scale MNIST images depending on the number of training datapoints.

The motivation for our scheme was to allow for bigger feature vectors and for more training data to be included. After observing the high accuracy achieved with limited training data and having gathered some information about the stability of the algorithm even when some penalties are violated, we decided to test the performance of our scheme on full-size MNIST data. We achieve training on a few datapoints and our QUBOs require between 5600 and 8600 qubits. The results are shown here fig. 6.8. This is beyond the capabilities of currently available quantum annealers but it will hopefully be achievable in the near future. Therefore, we kept the main part of our testing with smaller-size QUBOs in order to have problems which are solvable on current QA technology. Furthermore, smaller problems enable us to have more consistent tests, as finding the ground state of such big instances gets very difficult and takes approximately 1800s for the digital annealer to find. These results show that our scheme will be able to scale to larger feature spaces and produce good results, with the caveat that the hidden and output layers should be kept relatively small. Furthermore, we observed the same tendencies discussed throughout section 6.1 for preprocessed images to persist for full-scale MNIST images, but the changes in accuracy are less drastic and more smooth in comparison.

6.2 Sentiment Analysis

For sentiment analysis, we had only the two labels of "positive" and "negative" reviews. After doing some hand-crafted preprocessing, described in section 5.6, over the whole IMDb dataset, we were left with 394 reviews for training and 363 for testing, each in the form of a 203-element binary vector. Overall, the model successfully learned to differentiate between positive and negative reviews. Depending on the setup, the average accuracy of our model was about 70% and it had a maximal observed accuracy of 82%. The accuracy achieved for this problem was significantly lower than the differentiation between 0 and 1 and similar to the differentiation between 5 and 6. Similar to image classification most of our tests were focused on observing how the model responds to different internal parameters and we observed similar patterns for both tasks.

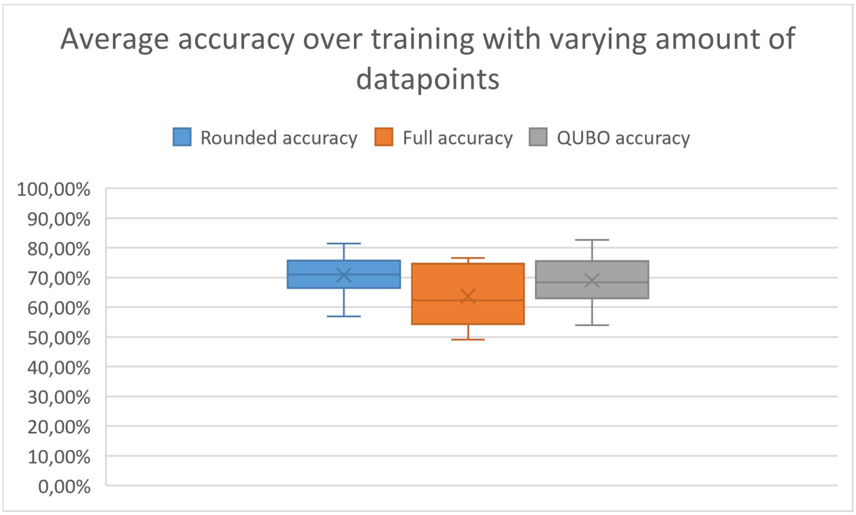


Figure 6.9: Average accuracies achieved over different amounts of training data with full-precision, rounded binary weights and our QUBO results.

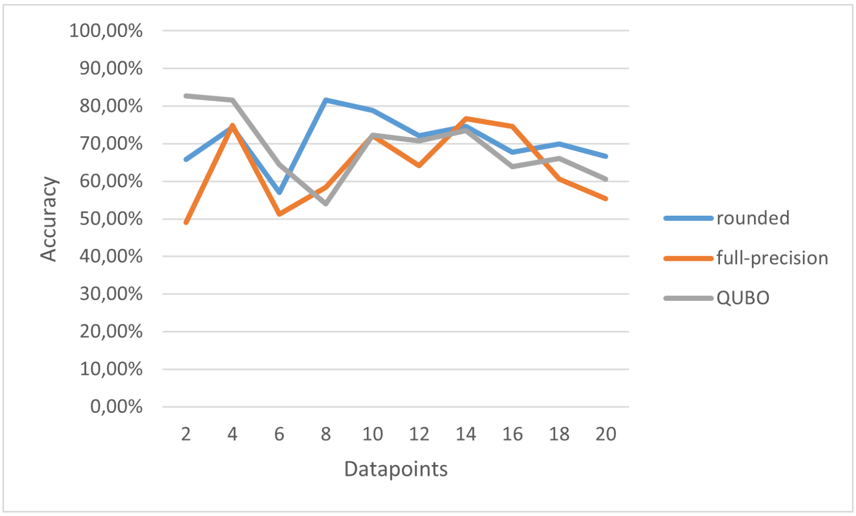


Figure 6.10: Accuracies achieved with different amounts of training data with full-precision, rounded binary weights and our QUBO results.

We observed that for few datapoints, between 2 and 6, our model again outperforms both the classical floating point weights and the binarized weights as can be seen in fig. 6.10 and it achieves good results on average fig. 6.9. However, we again see the pattern discussed in section 6.1, where the accuracy drops after a certain amount of training data. In this case, we observe a similar tendency also for the classical training, which may suggest that the preprocessing has some innate imbalance.

We also tested if the correlation between the penalty factor and objective function factor has any significant effects on the accuracy similar to what we tested in fig. 6.4 for image classification. The results of fixing the penalty factor and testing different values for the objective factor can be observed in fig. 6.11. This test was done for 6 datapoints, which showed a dip in accuracy in fig. 6.10. When we did the same test for the image classification we used 10 datapoints which was showing high accuracy. The idea behind this choice was to find if there is a change in the importance of the correlation depending on the overall expected accuracy. Similar to image classification, there

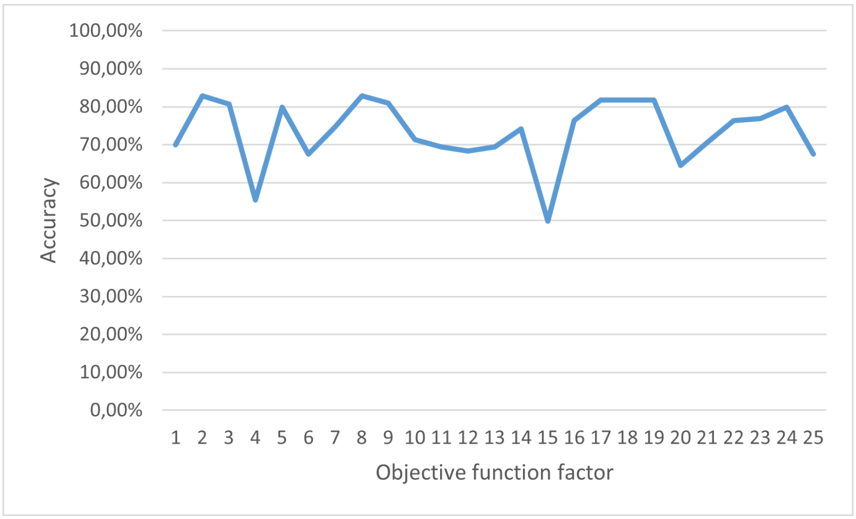


Figure 6.11: Accuracies depending on the objective function factor when the penalty factor is fixed to 10. The testing was done for 6 datapoints.

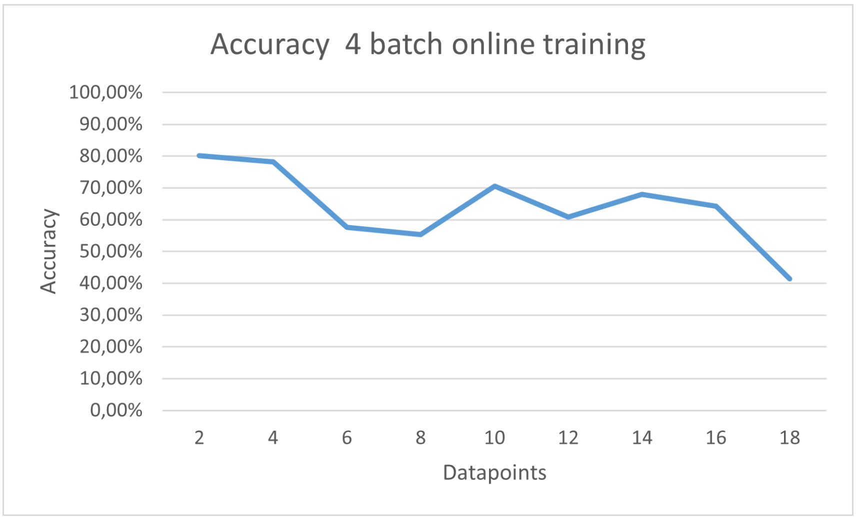


Figure 6.12: Accuracies achieved for training 4 batches with different number of input data in each batch

are differences in the accuracy achieved, however, no trend was observed. For our further tests, we kept our objective factor at 18 while the penalty factor was fixed to 10, as it showed the best results.

The performance of our online multibatch training model, explained in section 5.4, for this task can be seen in fig. 6.12. Similar to the image classification task, the performance of the scheme is satisfactory and achieves similar accuracy to the best solutions achieved via "one-shot" training. We also continue to observe that smaller batch sizes perform better if we keep the number of batches the same.

6.3 Final remark

We conclude with a final remark, considering the limitations of the tests that we have done in section 6.1 and section 6.2. Because of the many tunable parameters in the formulation of the QUBO and also in the annealing setup, we have to admit that more rigorous testing is required in order to properly benchmark the performance of our model in the future. Lets us give an example to illustrate this point. In fig. 6.10 the accuracy for 6 datapoints is $64,46\%$ as we were using an objective function factor 20 for all tests. When we look at fig. 6.11 we find that this result is indeed repeated, but there is a higher accuracy achieved with an objective function factor of 18, at $81,81\%$, which is a significant difference. Now if we add to that, the possible dimensions of solutions with some violations of penalties and different possibilities for penalty factors, the volatility of the possible accuracies becomes really high. Furthermore, there were no clear trends in the accuracy of the model as we have elaborated in section 6.1 and section 6.2. Due to time and resource constrictions, it was not feasible to exhaustively search for all dependencies or the best accuracies achieved in all scenarios. This, in our opinion, does not diminish our findings but has to be taken into consideration when looking into the performance of our model.

7 Conclusions

We investigated the problem of formulating the training of BNNs as a QUBO, with the aim to allow for the training to be done with quantum or quantum-inspired devices. We improved upon the formulation presented in [33] by allowing for a significant increase in the size of the input data, which allows for much bigger feature vectors to be used for training, compared to other approaches. Furthermore, we made some structural improvements to the formulation to add more freedom for optimization, penalize some infeasible solutions and better utilize the quantum-inspired hardware we used. The first was allowing for independent weighting of the objective function and the different penalties. For the penalty factors, this was done out of necessity to penalize states which represent an infeasible setup for the BNN. This also opens an additional avenue for further refinement of our model, considering the difference in performance achieved by solely changing the factor for the objective function. We also made a change to the original model, which allows for freely choosing the size of the input vector and the number of hidden neurons as long as it is an odd number. This minor improvement gives more freedom in choosing the size of the BNN which needs to be trained and ensures that the QUBO model will be able to adapt to the desired size for the NN and not the other way around.

Combined all of these changes allowed us to model the training for two of the most popular ML tasks - image classification and sentiment analysis. To the best of our knowledge, the full training of a BNN for these tasks has never been done with a QUBO until now. We achieved promising results of an average accuracy of about 80% for image classification and about 70% for sentiment analysis. Furthermore, in specific circumstances, namely when very few datapoints are available for training, our model outperforms an unoptimised classical MLP of the same size with the same loss and activation functions. However, the model shows some signs of instability like solutions, which lose accuracy with the addition of new data or with longer annealing time and therefore lower amount of errors, which is something that would need to be further investigated in the future.

We also devised a scheme that allows for the training to be done over time similar to a classical NN. It produces more robust weights and allows us to propose an additional way to utilize our scheme, which enables us to approximate good low precision weights for the NN, such as ternary weights $\{-1, 0, 1\}$ or small integers, while still training with binary weights and activations and keeping the sizes of the QUBOs the same. Even though this scheme sacrifices the "one-shot" style training it might be a good stepping stone for future research. It achieves consistently good results for small batches and has less volatility compared to our single-batch training. That being the case, the multibatch model still suffers from the instabilities mentioned earlier.

Compared to the formulation from [26] presented in chapter 3, our model does not allow for huge amounts of training data, but allows for the utilization of significantly bigger inputs, such as the full-scale images of the MNIST dataset which can be seen in section 6.1. Also in the case of new data being introduced, our multibatch approach can just train with the new batch and upgrade the optimal weights, while in the other formulation the loss function, would have to be recalculated and then a new full-size QUBO would need to be solved. Hopefully in the future finding a way to circumvent the restrictions of one or both of these approaches will be possible, which would result in very efficient and fast quantum training schemes for BNNs, with the possibility to train with real world data over many datapoints.

Overall, we showed that training BNNs for the tasks of image classification and sentiment analysis is possible with a QUBO, which fulfils our objective of showing that realistic BNNs can be trained on quantum devices in the future. Our formulation for the QUBO model is quite generalizable and we hope that it can be used for training BNNs for other ML problems in the future.

8 Outlook

Allowing for more than binary classification is a logical step for improving this model. This could be done by implementing a one-hot classification for the output layer with n neurons. We wanted to focus on extending the applicability of the model to tasks such as image classification and sentiment analysis for which input size was the biggest priority. Furthermore, allowing for more classes by adding additional neurons would make our model much more expensive considering that our scheme for reducing the variables tackles only the interaction between the input layer and the hidden layer. If there is a way to further reduce the needed qubits for the representation of the hidden and output layer, multiclass problems should be the next priority for such formulation.

As we explained in section 5.1 there is a need to differentiate between the factors for different penalties applied to our QUBO in order to penalize infeasible solutions. This also leaves another avenue for optimizing this formulation, as it might be the case that by fine-tuning the penalty factors better performance of the model could be achieved. This is often the case with QUBOs, as is explained in [27].

We considered a different formulation for how the additions are verified in the model, which requires fewer qubits than the current version explained in section 5.2. The currently used penalties require qubits equal to the second logarithm of the feature space. We are going to briefly present this idea for three bits being added, requiring only one bit rather than 2, here:

Let our binary vector be:

$$q = \begin{pmatrix} a & q_0 & q_1 & q_2 \end{pmatrix} \tag{8.1}$$

where a is the sign function over the sum $(q_0 + q_1 + q_2)$.

We will show how the matrix looks for this particular simple case:

$$\begin{bmatrix} 3 & -2 & -2 & -2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.2}$$

In the following we are checking what values the energy E has for the possible combinations of the variables q_0, q_1, q_2. We are going to show only one example where for the case where one or two of the bits are set to 1 to avoid unnecessary clutter. The energy is the same for the other cases.

$$\begin{aligned} a = 0: \quad q_0 = q_1 = q_2 = 0 &\rightarrow E = 0 \\ q_0 = 1, \quad q_1 = q_2 = 0 &\rightarrow E = 0 \\ q_0 = q_1 = 1, \quad q_2 = 0 &\rightarrow E = 1 \\ q_0 = q_1 = q_2 = 1 &\rightarrow E = 1 \end{aligned}$$

$$\begin{aligned} a = 1: \quad q_0 = q_1 = q_2 = 0 &\rightarrow E = 3 \\ q_0 = 1, \quad q_1 = q_2 = 0 &\rightarrow E = 1 \\ q_0 = q_1 = 1, \quad q_2 = 0 &\rightarrow E = 0 \\ q_0 = q_1 = q_2 = 1 &\rightarrow E = 0 \end{aligned}$$

We observe that a correctly calculates the sign of the sum $q_0 + q_1 + q_2$ and the energy is positive in all incorrect setups while 0 in the correct ones. This result is promising but creating a generalized scheme proves to be a challenging task and we leave it as an interesting topic for future research.

As we described in section 5.6 we are using a very simple preprocessing for our text data. Some ideas we had include utilizing embeddings such as *Word2vec* or *GloVe* and then group the vectors we get into clusters. If there is a word from one of the clusters in the review then the corresponding bit in the feature vector is set to 1. Clustering in this context could prove tricky as it might be hard to create balanced clusters that are semantically similar, have roughly the same density and also their number is not too high - ideally less than 100. Another possibility is applying clustering after using a tf-idf vectorization. However, clustering such sparse data is very dangerous because it would be hard to capture semantic similarities and also it would be hard to avoid grouping dissimilar words together. Here we should also note that an expensive model for binary word encoding might produce dense feature vectors which represent the data very well but such models might be in conflict with most of the reasons for using BNNs in the first place. Because of these reasons, we stuck with a simple and straight-forward approach for preprocessing and we leave finding an optimal balance between complexity, resource consumption and optimal feature extraction for future research.

Lastly, the unexpected behaviour which we observed throughout section 6.1 and section 6.2, where the accuracy achieved with our model was dropping after a certain amount of datapoints used in a single batch seen in fig. 6.10 and fig. 6.2 remains unexplained. We observed the effects when using different datapoints and when using different internal parameters for our formulation, which simply confirmed our observation and did not show any biases in the data. It should be noted that this behaviour persisted even when we were using our multibatch online training scheme, where we essentially average all of the solutions after each batch. Therefore, this is a phenomenon that we leave for further research, as it might hold some insights about restrictions in the formulation of the training of a BNN or even any NN as a QUBO.

9 Abbreviations

QUBO	Quadratic Unconstrained Binary Optimisation
ML	Machine learning
QML	Quantum machine learning
AQC	Adiabatic quantum computing
GQC	Gate-based quantum computing
VQE	Variational Quantum Eigensolver
QAOA	Quantum Approximation Optimization Problem
BNN	Binary neural network
NN	Neural network
QNN	Quantum neural network
QCNN	Quantum convolutional neural network
QA	Quantum annealing
SA	Simulated annealing
DA	Digital annealing
STE	Straight Through Estimator
CNN	Convolutional neural network
RNN	Recurrent neural network
NLP	Natural language processing
LSTM	Long short-term memory
MLP	Multilayer perceptron
KNN	K-nearest neighbour
PCA	Principal component analysis
SVM	Support Vector Machine

10 References

[1] Stefan Leijnen and Fjodor van Veen. The neural network zoo. In *Proceedings*, volume 47, page 9. MDPI, 2020.

[2] Taylor Simons and Dah-Jye Lee. A review of binarized neural networks. *Electronics*, 8(6):661, 2019.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[4] Dennis Willsch, Madita Willsch, Hans De Raedt, and Kristel Michielsen. Support vector machines on the d-wave quantum annealer. *Computer physics communications*, 248:107006, 2020.

[5] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.

[6] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv preprint arXiv:1401.2142*, 2014.

[7] Jing Li, Song Lin, Kai Yu, and Gongde Guo. Quantum k-nearest neighbor classification algorithm based on hamming distance. *Quantum Information Processing*, 21(1):18, 2022.

[8] Date, P., Arthur, D., Pusey-Nazzaro, and L. Qubo formulations for training machine learning models. *Sci Rep 11, 10029*, 2021.

[9] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.

[10] Yunseok Kwak, Won Joon Yun, Soyi Jung, and Joongheon Kim. Quantum neural networks: Concepts, applications, and challenges. In *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 413–416. IEEE, 2021.

[11] Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. Quantum computing: A taxonomy, systematic review and future directions. *Software: Practice and Experience*, 52(1):66–114, 2022.

[12] Somayeh Bakhtiari Ramezani, Alexander Sommers, Harish Kumar Manchukonda, Shahram Rahimi, and Amin Amirlatifi. Machine learning algorithms in quantum computing: A survey. In *2020 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2020.

[13] Manuel S Alvarez-Alvarado, Francisco E Alban-Chacón, Erick A Lamilla-Rubio, Carlos D Rodríguez-Gallegos, and Washington Velásquez. Three novel quantum-inspired swarm optimization algorithms using different bounded potential fields. *Scientific Reports*, 11(1):11655, 2021.

[14] Natansh Mathur, Jonas Landman, Yun Yvonna Li, Martin Strahm, Skander Kazdaghli, Anupam Prakash, and Iordanis Kerenidis. Medical image classification via quantum neural networks. *arXiv preprint arXiv:2109.01831*, 2021.

[15] Yanxuan Lü, Qing Gao, Jinhu Lü, Maciej Ogorzałek, and Jin Zheng. A quantum convolutional neural network for image classification. In *2021 40th Chinese Control Conference (CCC)*, pages 6329–6334. IEEE, 2021.

[16] Carlos A Riofrío, Oliver Mitevski, Caitlin Jones, Florian Krellner, Aleksandar Vučković, Joseph Doetsch, Johannes Klepsch, Thomas Ehmer, and Andre Luckow. A performance characterization of quantum generative models. *arXiv preprint arXiv:2301.09363*, 2023.

[17] Riccardo Di Sipio, Jia-Hong Huang, Samuel Yen-Chi Chen, Stefano Mangini, and Marcel Worring. The dawn of quantum natural language processing. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8612–8616. IEEE, 2022.

[18] He-Liang Huang, Yuxuan Du, Ming Gong, Youwei Zhao, Yulin Wu, Chaoyue Wang, Shaowei Li, Futian Liang, Jin Lin, Yu Xu, et al. Experimental quantum generative adversarial networks for image generation. *Physical Review Applied*, 16(2):024051, 2021.

[19] Smit Chaudhary, Patrick Huembeli, Ian MacCormack, Taylor L Patti, Jean Kossaifi, and Alexey Galda. Towards a scalable discrete quantum generative adversarial neural network. *Quantum Science and Technology*, 8(3):035002, 2023.

[20] Seunghyeok Oh, Jaeho Choi, and Joongheon Kim. A tutorial on quantum convolutional neural networks (qcnn). In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 236–239. IEEE, 2020.

[21] Iris Cong, Soonwon Choi, and Mikhail D Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.

[22] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8:141007–141024, 2020.

[23] Avinash Chalumuri, Raghavendra Kune, and BS Manoj. Training an artificial neural network using qubits as artificial neurons: a quantum computing approach. *Procedia Computer Science*, 171:568–575, 2020.

[24] Das quantum computing user network in deutschland. `https://www.qucun.de/`, 2022.

[25] Quantum-classical hybrid optimization algorithms for logistics and production line management. `https://qarlab.de/en/qchallenge-en/`, 2022.

[26] Prasanna Date and Thomas Potok. Adiabatic quantum linear regression. *Scientific reports*, 11(1):21905, 2021.

[27] Catherine McGeoch and Pau Farre. Advantage processor overview. Technical report, 2022.

[28] Steve Abel, Juan C Criado, and Michael Spannowsky. Completely quantum neural networks. *Physical Review A*, 106(2):022601, 2022.

[29] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models. *arXiv preprint arXiv:1811.11538*, 2018.

[30] Steven H Adachi and Maxwell P Henderson. Application of quantum annealing to training of deep neural networks. *arXiv preprint arXiv:1510.06356*, 2015.

[31] Max Wilson, Thomas Vandal, Tad Hogg, and Eleanor G Rieffel. Quantum-assisted associative adversarial network: Applying quantum annealing in deep learning. *Quantum Machine Intelligence*, 3:1–14, 2021.

[32] Michele Sasdelli and Tat-Jun Chin. Quantum annealing formulation for binary neural networks. In *2021 Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–10. IEEE, 2021.

[33] Hiroshi Nakayama, Junpei Koyama, Noboru Yoneoka, and Toshiyuki Miyazawa. Description: third generation digital annealer technology, 2021.

[34] Elizabeth Crosson and Aram W Harrow. Simulated quantum annealing can be exponentially faster than classical simulated annealing. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 714–723. IEEE, 2016.

[35] Gian Giacomo Guerreschi. Solving quadratic unconstrained binary optimization with divide-and-conquer and quantum algorithms. *arXiv preprint arXiv:2101.07813*, 2021.

[36] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355, 1998.

[37] Johnson, Amin, Gildert, S., M., and L. Quantum annealing with manufactured spins. *Nature 473, 194–198*, 2011.

[38] Dennis Willsch, Madita Willsch, Carlos D Gonzalez Calaza, Fengping Jin, Hans De Raedt, Marika Svensson, and Kristel Michielsen. Benchmarking advantage and d-wave 2000q quantum annealers with exact cover problems. *Quantum Information Processing*, 21(4):141, 2022.

[39] Max Born and Vladimir Fock. Beweis des adiabatensatzes. *Zeitschrift für Physik*, 51(3-4):165–180, 1928.

[40] Sheir Yarkoni, Elena Raponi, Thomas Bäck, and Sebastian Schmitt. Quantum annealing for industry applications: Introduction and review. *Reports on Progress in Physics*, 2022.

[41] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[42] Omid Ghasemalizadeh, Seyedmeysam Khaleghian, and Saied Taheri. A review of optimization techniques in artificial networks. *Int. J. Adv. Res*, 4(9):1668–1686, 2016.

[43] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[44] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[45] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[46] Kirsty Duncan, Ekaterina Komendantskaya, Robert Stewart, and Michael Lones. Relative robustness of quantized neural networks against adversarial attacks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[47] Faiq Khalid, Hassan Ali, Hammad Tariq, Muhammad Abdullah Hanif, Semeen Rehman, Rehan Ahmed, and Muhammad Shafique. Qusecnets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks. In *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 182–187. IEEE, 2019.

[48] Adnan Siraj Rakin, Jinfeng Yi, Boqing Gong, and Deliang Fan. Defend deep neural networks against adversarial examples via fixed and dynamic quantized activation functions. *arXiv preprint arXiv:1807.06714*, 2018.

[49] Hyungjun Kim, Jihoon Park, Changhun Lee, and Jae-Joon Kim. Improving accuracy of binary neural networks using unbalanced activation distribution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7862–7871, 2021.

[50] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[51] A complete guide to image classification in 2023. `https://viso.ai/computer-vision/image-classification/`, 2023.

[52] Image classification: 6 applications and 4 best practices in 2023. `https://research.aimultiple.com/image-classification/`, 2023.

[53] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[57] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[58] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[59] Peter D Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. *arXiv preprint cs/0212032*, 2002.

[60] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*, 2002.

[61] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[62] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[63] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[64] Samuel Yen-Chi Chen, Shinjae Yoo, and Yao-Lung L Fang. Quantum long short-term memory. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8622–8626. IEEE, 2022.

[65] Weiyi Zheng and Yina Tang. Binarized neural networks for language modeling. Technical report.

[66] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[67] Kumar Shridhar, Harshil Jain, Akshat Agarwal, and Denis Kleyko. End to end binarized neural networks for text classification. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 29–34, 2020.

Acknowledgments

We would like to express our sincere appreciation to Fujitsu for providing us with access to their Digital Annealer during the course of this project. Their generous support and collaboration have been instrumental in the successful execution of our research work. Their digital annealing technology significantly enhanced our ability to scale the proposed model to a good size, considering the limitations of the currently available quantum technologies, enabling us to achieve our research objectives.

I would like to express my gratitude to my two supervisors, Dr. Peter Limacher and Prof. Dr. Jan Niehues, for their guidance, support, and expertise throughout the duration of this research project. I am grateful to Dr. Limacher for the continuous support and insightful discussions. I am also thankful to Prof. Dr. Niehues, for the mentorship, encouragement, and expertise. I am grateful to both of them for their valuable input and consistent support in every stage of this thesis. Their knowledge has greatly enriched my understanding of the subject matter and I am thankful for their guidance and assistance in overcoming various challenges. I am honoured to have had the opportunity to work under your supervision.

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 16. August 2023

Georgi Georgiev: