Grover's Implementation of Quantum Binary Neural Networks

Brody Wrighter*, Sonia Lopez Alarcon†
KGCOE Department of Computer Engineering, Rochester Institute of Technology
Rochester, New York, United States of America
Email: †slaeec@rit.edu

2023 IEEE International Conference on Quantum Computing and Engineering (QCE) | 979-8-3503-4323-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/QCE57702.2023.00043

***Abstract*—Binary Neural Networks (BNNs) are the result of a simplification of network parameters in Artificial Neural Networks (ANNs). The computational complexity of training ANNs increases significantly as the size of the network increases. This complexity can be greatly reduced if the parameters of the network are binarized. Binarization, which is a one bit quantization, can also come with complications including error and information loss.**

The implementation of BNNs on quantum hardware could potentially provide a computational advantage over its classical counterpart. This is due to the fact that binarized parameters fit nicely to the nature of quantum hardware. Quantum superposition allows the network to be trained more efficiently, without using back propagation techniques, with the application of Grover's Algorithm for the training process. This paper looks into two BNN designs that utilize only quantum hardware, as opposed to hybrid quantum-classical implementations. It also provides practical implementations for both of them. Looking into their scalability, improvements on the design are proposed to reduce complexity even further.

***Index Terms*—Binary Neural Networks, Grover's algorithm, Machine Learning**

I. Introduction

ARTIFICIAL Neural Networks (ANNs) are a form of Machine Learning (ML) able to make decisions or classifications by using a set of tunable parameters (weights) in a network of artificial neurons. The key to reaching accurate decision making or classification comes from finding the right set of weights by exposing the ANN to a training set of inputs, so it can then classify new inputs on its own. Finding these network's weights is known as the *training* of the ANN, and comes with growing computational cost as the network or training data set grows.

Much research is focused on finding efficient ways of training ANNs, or efficient ANN approaches, hoping to reduce this computational cost of the training process without sacrificing the accuracy of the classification process. One of these approaches is Binary Neural Networks(BNN), which restricts the weights of the network to binary values. While BNNs can reduce the complexity of Neural Networks(NNs), efficient training is a challenge, given the step function nature of its input/output model, the potential loss of accuracy of this simplification, and, in any case, the still large data sets necessary for effective training. Various approaches are being explored, attempting to reduce this complexity or computation time further. One such method is the use of Quantum Hardware to implement BNNs.

BNNs are particularly suitable for quantum implementations, given the binary nature of the weights. The problem of finding the right set of weights can be framed as a Grover's search algorithm, in which the basis encoded solution(s) will stand out among all other possible combinations with the highest probability. This approach was already proposed [1], but naive implementations can lead to poor scalability of the quantum circuit. Two key parameter must be taken into account when evaluating the scalability of the designs: width and depth of the quantum circuit. Designing the oracle of the Grover's search algorithm to implement this specific search is the key and challenging piece.

This work takes a close look at the previously proposed algorithms [1], and expands on it by significantly reducing the complexity of the original implementation. Two distinct designs based on the Grover's Search are explored: one design uses *Phase Estimation*, while the other uses *Register Counting* to train the neural network. The new work includes implementations and two proof of concept test cases on real binary classification.

II. Background and Related Work

Artificial Neural Networks, are inspired by the brain's complex network of ∼86 billion neurons, and are the most successful of the Machine Learning implementations. When neural networks are exposed to data sets, they are able to "learn" from this data set to then make decisions or classifications on items that are new to them. Neural networks have made significant achievements and have led to steady improvement in the accuracy of the models for tasks like object classification [2], voice recognition [3], machine translation [4], and more. However, these improvements come at the cost of the models' size and complexity, with today's state-of-the-art models containing 100s of billions of parameters. Moreover, implementing training and inference of these state-of-the-art models on conventional CPU/GPU hardware incurs large overheads that are incompatible with application domains that have strict size, weight, and power (SWaP) constraints. Neural network quantization is a particularly attractive solution due to the super-linear reduction in compute resources with linear scaling of parameter bits. However, as neural network parameter precision is reduced, training with gradient descent becomes challenging due to the discrete nature of the loss landscape [5]. Therefore, methods that are able to efficiently

979-8-3503-4323-6/23/$31.00 ©2023 IEEE
DOI 10.1109/QCE57702.2023.00043

explore discrete parameter spaces are required to effectively scale neural network quantization.

The use of Quantum Computing as an accelerator of machine learning algorithms has been proposed as a hybrid algorithm. In particular, the use of Variational Quantum Eigensolver algorithms [6], [7] to solve the training of quantum neural networks. This approach and the Quantum Approximate Optimization Algorithm [8], [9] are quantum-classical implementations in which an initial quantum ansatz is implemented in a parameterizable quantum circuit, and evaluated against a classical cost function to adjust the quantum parameters. This is a Hamiltonian encoded quantum approach, for which practical uses are still being explored, with no quantum advantage being demonstrated to date. The work around this set of algorithms is extensively mathematical, exploring the many theoretical challenges of the topology in the quantum optimization algorithm, and of the extraction of information in the NISQ era [10], [11], [12], [13], [14], [15], [16], [17].

The work proposed and discussed in this paper is on the other hand, fully quantum, based on Grover's search algorithm. Grover's algorithm [18] was initially proposed as a search algorithm, but under the right angle, it can be reduced to the search for a solution of a given optimization problem. This approach is discussed in this paper, with special attention to the scalability of the problem.

III. Grover's Implementation of Quantum Binary Neural Networks

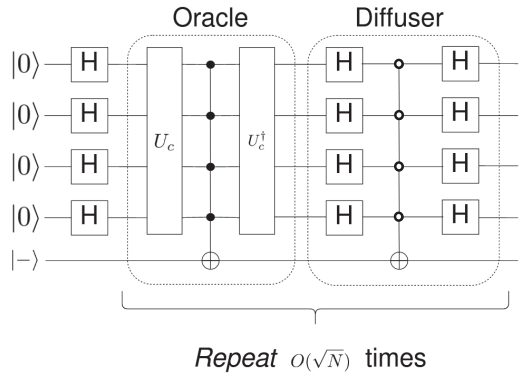


Fig. 1: Summarizing representation of Grover's algorithm circuit. The qubit register is placed in a superposition state. The oracle applied to the superposition state results in $|f(x)\rangle$. If the target qubit is in the $|-\rangle$ state, the phase kickback effect then leaves the target qubit unaffected and changes the quantum register's state to $(-1)^{f(x)}|x\rangle$. The diffuser through multiple executions amplifies the amplitudes of the right states out of the superposition.

Grover's algorithm [18] is one of the first quantum algorithms to show true potential acceleration when compared to similar classical problems. Its goal is to find a specific element in an unsorted list of elements using a qubit-based quantum computer. Grover's algorithm can search a list of N elements using $O(\sqrt{N})$ steps, compared to the $O(\frac{N}{2})$ steps needed for the average number of comparisons to search an unordered list using a classical computer.

Grover's algorithm works by iteratively increasing the amplitude of the solution states out of a superposition of states. The maximum amplitude of the probability for the solution states is achieved at approximately $\sqrt{N}$ iterations. Grover's algorithm uses an oracle to determine which states will be amplified in each iteration. The oracle, along with the phase kickback with target qubit set to $|-\rangle$, results in $(-1)^{f(x)}|x\rangle$. Applying the oracle and the diffuser further increases the solution states' probabilities of being measured. The general structure of Grover's algorithm is shown in Figure 1. The basis state with higher probability corresponds to the search's solution.

Hence, this algorithm aids with the identification of solutions encoded as strings of 0s and 1s. In this work, Grover's algorithm is used as a mechanism to train a Binary Neural Network (BNN) by finding the set of weights that will result in the best classification accuracy. The problem is about correctly defining the oracle that will amplify the right set of binary weights.

A. Quantum Phase Estimation Binary Neural Network Implementation

Figure 3 summarizes the full implementation proposed in [1], broken down into three parts. Being this a Grover's search algorithm, it can be broken down into *initialization* (before the *O* border), *oracle*, and *diffuser* (*D* block after the *O'* border.) The *oracle* is represented by all the computational steps between *O* and *O'*.

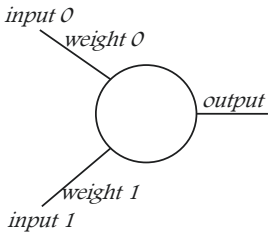
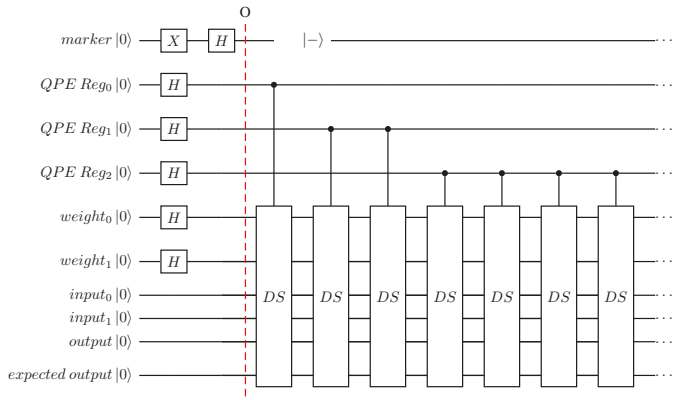


Fig. 2: Two input example neuron

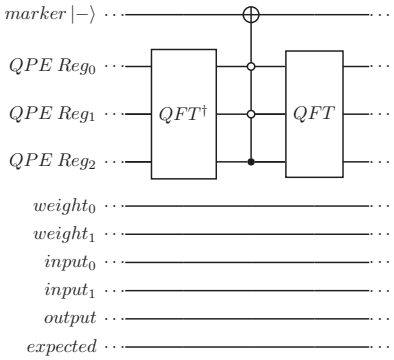
1) Qubit registers and initializations: Figure 3 implements a two input neuron example such as the one shown in Figure 2, and it uses six different sets of qubits:

- marker: Grover's must-have qubit to enable phase kickback, initialized to $|-\rangle$.
- weight: a (two) qubit register that contains the (two) binary weight vector, initialized to a uniform superposition. The *diffuser* is applied at the end to this register and will contain the solution to the training problem.
- input: a (two) qubit register containing the (two) binary inputs to the neuron. In a NN, these come from other neurons, and have corresponding weights associated with them.
- output: the output produced by the neuron.
- expected output: the expected output, known and provided for training.

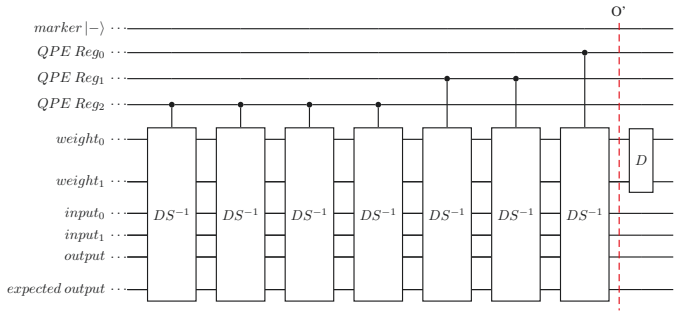
- QPE register: register of p qubits, used for quantum phase estimation, and initialized to a uniform superposition. p has an impact on the precision of the estimation.



(a) First part: The controlled-DS blocks contain the training dataset implementations, as shown in Figure 4.



(b) Second Part: Using IQFT and QFT to apply the accuracy threshold. IQFT converts the accumulated phases to a binary encoding with the least significant bit being the top most qubit.



(c) Third Part: Uncomputing the controlled-DS blocks, followed by Grover's diffuser applied on the weights.

Fig. 3: Complete original QBNN circuit implemented through Quantum Phase Estimation (QPE) for a two weight, two input, one output example. The three parts top to bottom can be connected left to right. The steps between O-O' represent the oracle that will help identify the correct set of weights to provide the desired classification accuracy.

2) Oracle: The idea behind this implementation is to accumulate phase rotations on the *QPE Register* when the output and expected output are equal, thanks to the phase kickback coming from the dataset training (DS blocks in Figure 3a). Then, quantum phase estimation is used to identify the sets of weights that produced correct outputs above a certain accuracy threshold (Figure 3b). Last, the dataset blocks are un-computed (Figure 3c.)

3) Dataset (DS) controlled unitary: The controlled *DS unitary* contains the full training dataset for the NN. Figure 4 shows the training for the input values 10 with expected output 1, and 11 with expected output 0. X-gates implement and reverse these specific training datasets.

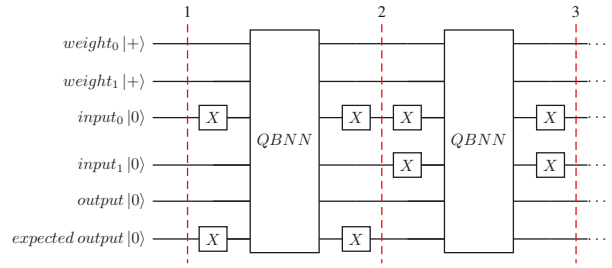


Fig. 4: Dataset (DS) implementation. The QBNN circuit shown above represents the circuit in Figure 5. This circuit is repeated for each entry in the training dataset. The dataset binary values are implemented with the X-gates placed before and after each QBNN block.

The *QBNN* blocks contain the actual Quantum Binary Neural Network implementation, as depicted in Figure 5 for a two input, two weight example. They take care of the training of the NN, for each dataset that will determine the values for the weights. Usually in NNs, the inputs and weights are multiplied ($a_i * w_i$), however, in this QBNN the operation is $a_i \oplus w_i$ which is implemented with a CNOT operation among the corresponding input and weight. The different computation does not change the result of the training, as long as the training is done consistently to this approach. A Toffoli gate implements the *activation function* of the neuron, in this case a discrete activation function that assigns a 1 value to the output if both inputs are 1. Next, a phase of $\frac{\pi}{n}$ is produced in two cases: when the expected output and the neuron output are both equal to 1, or when the expected output and neuron output are both equal to 0, where n is the size of the training dataset. Last, inputs and weights are returned to the original state, reversing the computations.

Neurons in this shape are connected to form a Neural Network, accumulating phase on the right sets of binary weights out of the superposition. Repeated instances of these QBNN are used to apply the training dataset (DS) (Figure 4), and multiple instances of these DS will accumulate an approximate phase (precision depending on the number of qubits in the QPE register [19]) through phase kickback on the *QPE register* (Figure 3).

4) Quantum phase estimation and diffuser: The rest of the algorithm (Figures 3b and 3c) completes the quantum phase estimation (QPE) through a QFT^{-1} applied to the QPE register to transfer the phase to a binary number. The phase is really expressed as a multiplier θ applied to the $\frac{\pi}{n}$. An accuracy threshold is applied, to set a minimum phase value above which the set of weights is expected to behave. For example, as shown in Figure 3b, if the expectation is that out

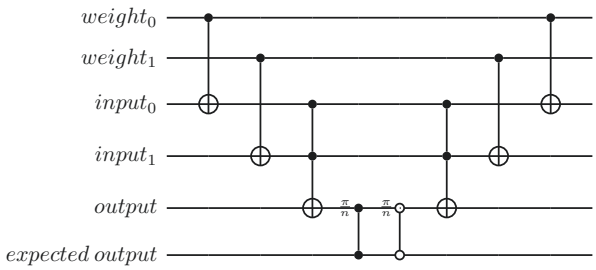


Fig. 5: Quantum phase estimation binary neural network two input neuron example. This circuit implements the neuron shown in Figure 2 for the QPE training implementations.

of $n = 4$ training cases, all four of them will be correct, then a triple-controlled NOT gate encoding the number $4_d = 100_b$ will *mark* (as in the phase inversion of Grover's search) all weight sets that result in $\theta = 4$ accumulated in a phase $4 * \frac{\pi}{n}$.

The only thing left to do is to revert all the steps (QFT and DS^{-1}) to then apply the diffuser that will grow the probability of the correct binary weight sets. $O(\sqrt{N})$ repetitions of these steps will be sufficient to measure them, being N the size of all possible weight combinations.

B. Improved Quantum Phase Estimation QBNN

There are two improvements to the Quantum Phase Estimation implementation of the Quantum Binary Neural Network: The first one is just the elimination of an unnecessary ancilla qubit that was used in the original design for the phase accumulation. Figure 5 already shows the design without that extra qubit and the original design and discussion can be seen in the original paper [1]. The second improvement removes the many repetitions of the DS blocks, as explained next.

1) Dataset (DS) phase accumulation: In Figure 3 the controlled DS blocks —with potential phase shift $\phi = \frac{\pi}{n}$— are applied repeatedly to the QPE register for quantum phase estimation [19], depending on the bit significance. As in the example (Figure 3), three QPE qubits require $1 + 2 + 4$ DS blocks. If a fourth qubit were necessary to improve the precision of the phase estimation, an additional eight $(1+2+4+8$ total) DS blocks would be necessary. However, with this information, it is possible to define the blocks with the specific phase rotations needed, instead of accumulating $\phi = \frac{\pi}{n}$ increments in each qubit. This is shown in Figure 6. The DS blocks now implement a specific $2^i\phi$, where i is the significance of the qubit and $\phi = \frac{\pi}{n}$, and n is the number of cases in the training set, known beforehand. Only p controlled DS blocks are necessary for a QPE register with p qubits, as opposed to

$$\sum_{i=0}^{p-1} 2^i$$

DS blocks for QPE register with p qubits. This reduces the depth of the design very significantly, as it will be shown in Section IV.

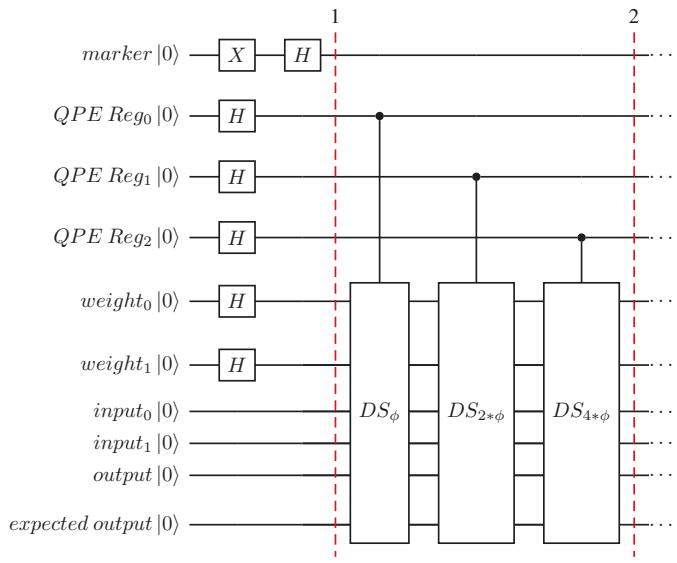


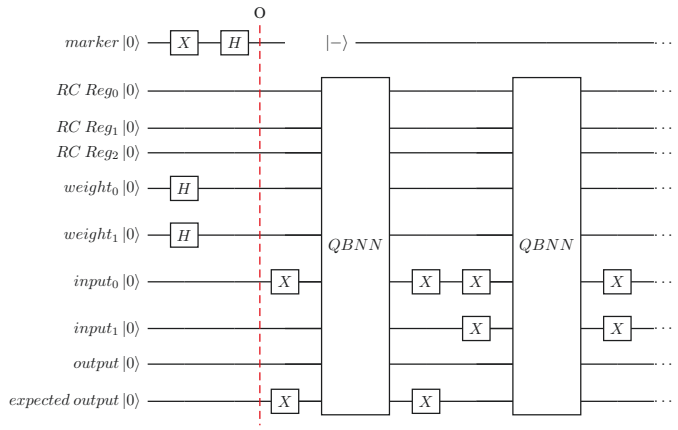
Fig. 6: The QPE Improved training circuit follows the same structure and order as the original QPE training circuit shown in [1]. The improvement of this circuit can be seen within the QPE portion of the training. Due to the fact that the accumulation oracle is a set phase based on the number of training dataset entries, the oracle phase can simply be doubled for each higher significance qubit in the QPE register as seen above. This reduces the depth of the circuitry greatly.

C. Quantum Register Counting Binary Neural Network Implementation

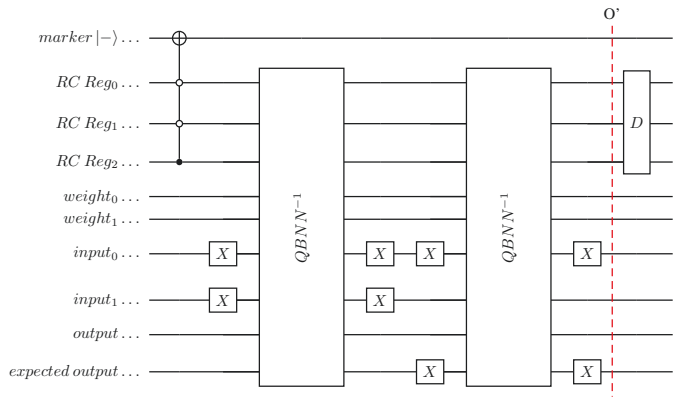
The Register Counting implementation follows a similar structure to the Quantum Phase Estimation case. The difference, however, is that instead of accumulating phase to then be estimated through QPE, this RC approach simply keeps a count of the number of times that a set of weights returns an output equal to the expected output. Figure 7 summarizes this approach. Due to the fact that the RC implementation records each successful weight string with a binary encoding, per-qubit phase accumulation is not necessary. This means that the training dataset only needs to be implemented once before the accuracy threshold. The caveat is that the Register Counting implementation needs to have enough qubits to represent the number of training data in a binary encoding.

The QBNN circuit is shown in Figure 8. Each QBNN block in this case contains the circuitry necessary to increment the counting register by one on each correct *output-expected output* match. Just like in the QPE case, the weights are applied to the inputs through CNOT gates ($a_i \oplus w_i$), followed by a threshold that implements the activated function, a Toffoli gate, in this case, implements $output = 1$ when both $a_i \oplus w_i$ are equal to 1. When both output and expected-output are equal (00 or 11) then an incrementing controlled unitary is applied to the counting register. The incrementing unitary is shown in Figure 9. The activation function and XOR gates are reversed to train the next data point.

After the system has been trained on all training data points, the accuracy threshold is applied, just like in the QPE case. Last, all computations are un-computed before



(a) First Part: The Circuit is initialized like any Grover's search. The QBNN block implements the increments for each correct *output-expected output* match, and it is applied on each data point of the training dataset implemented with x-gates. The counting register is directly related to the maximum size of the training dataset. The above example can only handle up to 7 dataset entries.



(b) This circuit continues the circuit shown in 7a. The triple-controlled NOT gate implements the accuracy threshold for the QBNN training, in this case the threshold is $4_d = 100_b$. Other thresholds can be set by varying the sets of CNOT gates. The training is then uncomputed to return the states of all qubits to their original state. The Grover's diffuser is then applied.

Fig. 7: QBNN implemented with the Register Counting approach.

applying the Grover's search diffuser. Reversing the order of the incrementing circuit in Figure 9 will decrement the RC register by one for each correct training case.

IV. Scalability Results

The scalability of these implementations is crucial to prove their usefulness. The original design proposed in [1] could not be applied to any practical implementations due to their poor scalability, from two perspectives: its width, or number of qubits required; and its depth, or number of computational steps necessary for completion. This scalability had such an impact on the circuit's width and depth of the original implementation that its simulation was prohibitive. In this work, the size of these designs, lack of access to large enough quantum hardware, and current hardware's noise levels do not allow for actual performance metrics. However, depth is a good

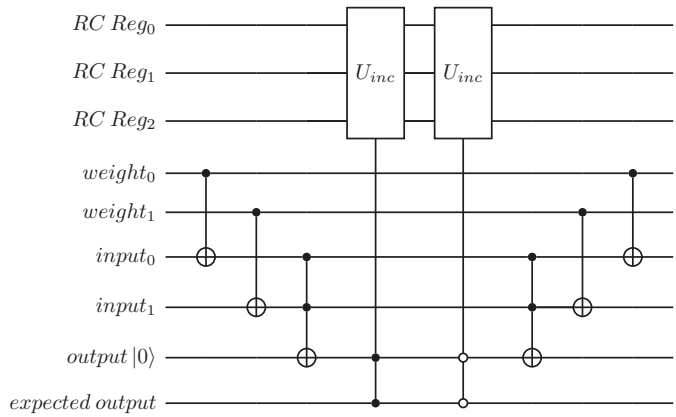


Fig. 8: The complete register counting QBNN implementation is shown, implementing the 2-input neuron example. Here the RC oracle is implemented instead of the QPE oracle.

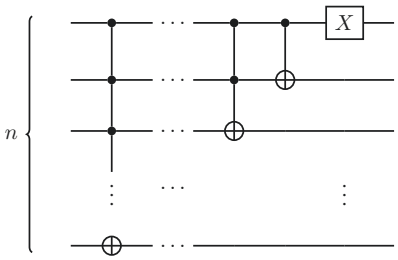


Fig. 9: Register incrementing circuit. When a number n is presented at the input, $n+1$ is produced at the output. This circuit can then be converted to a controlled circuit and used as the oracle for the RC QBNN.

first approximation to performance, given that it represents the number of steps to completion. Therefore, lower depth numbers are better from a performance perspective. They are also better from a noise perspective, since depth challenges coherence times. As this work will show in Section V, the new implementations allowed to demonstrate their accuracy on a practical problem, convolution edge detection, with a significant reduction of the descriptive depth of the circuit design.

Therefore, three different implementations are compared in this section:

- Quantum Phase Estimation BNN: This is the original implementation as presented in [1], in which the neurons accumulate phase on the QPE register on each correct output-expected output match.
- Improved QPE BNN: This is the improved QPE version proposed in this work. Since the phase rotation in each QPE qubit is known, and only dependent on the training dataset size, there is no need to repeat generic DS blocks, but implementing the right rotation is sufficient.
- Register Counting BNN: This new implementation of the QBNN uses a direct binary count of the number of correct outputs rather than a phase accumulation. This allows to train the NN with the dataset only once, as opposed to the repeated training for each phase in the cases above.

There are multiple parameters that need to be balanced in the design of Binary Neural Networks, and that have an impact on the width and depth of the circuit's design. These are:

- Size of the network's architecture: number of nodes per hidden layer and number of hidden layers,
- Activation function,
- Size of the training dataset, and
- Quantum phase estimation precision.

An approximate depth (in number of computational descriptive steps) and width have been calculated through expressions that depend on the parameters above. These expressions are not shown here for the sake of space, and for being all different factors rather convoluted to explain. Details can be found in [20]. The next sections explain their trends and the reasons behind them.

A. Size of the network's architecture

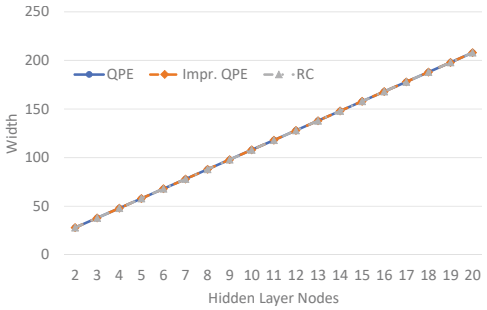
Two aspects are considered regarding the size of the neural network: the number of nodes per hidden layer and the number of hidden layers. With the parameters shown in Table I, the number of nodes per hidden layer and the number of hidden layers were varied.

QPE, RC Qubits	5
Dataset Size	16
Grover's Iterations	1
Accuracy Threshold	3
Activation Function Threshold	2

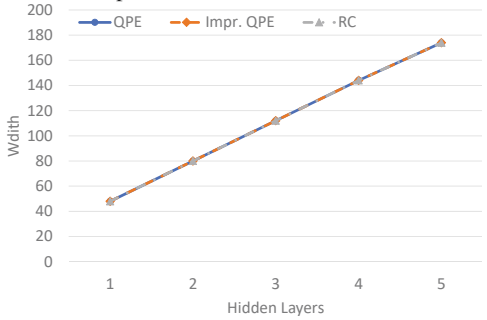
TABLE I: NN parameters used for the variation of hidden layer nodes.

The width, or number of qubits required, is shown in Figure 10a and Figure 10b. In this case and all other comparisons, it is the same for all three implementations —as long as the RC and QPE registers are using the same number of qubits, see Section IV-C.— The width increases linearly with the number of nodes and hidden layers. The number of nodes is the critical factor in the growth in this case, since each node implements a neuron with its own outputs, inputs, and associated weights. In the case when those nodes are distributed across multiple layers, the number is slightly lower (174 qubits in 20 nodes across 5 hidden layers in Figure 10b, as opposed to 208 qubits for 20 nodes in one hidden layer in Figure 10a). This is due to the fact that some outputs of the neurons of one layer are inputs to the next, so some input qubits are saved in that case. In any case, the width is not a critically limiting factor to the implementation of this size BNN, although the trend predicts quickly surpassing the 1000 qubit width for approximately 100 node implementations.

The depth on the other hand displays different behaviors depending on the quantum implementation of the BNN. Figure 11 shows exponential growth of the approximate depth of the circuit with the number of nodes for all three cases. However, the original design, QPE, reaches over 13M (million) computational steps for 19 nodes, while, the improved QPE barely surpasses 2M and RC requires about 422K steps. The cost in number of steps of these implementations is prohibitive. The 20 node case made the trends impossible to display reasonably due to the high exponential of the QPE case, and it was left out for that reason.



(a) Variation of the number of nodes within one single hidden layer containing 2-20 nodes. 4 input nodes-1 output node.



(b) Variation of the number of hidden layer containing 4 nodes each. 4 input nodes-1 output node.

Fig. 10: Number of hidden layer

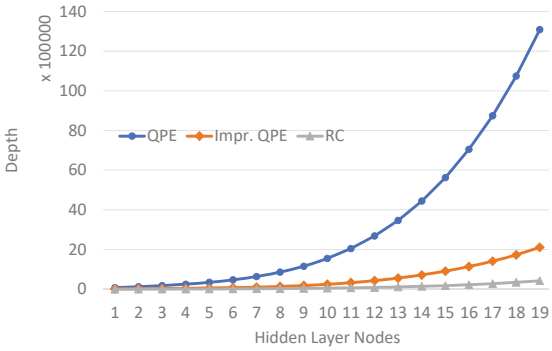


Fig. 11: Approximate depth with variation of the number of nodes within one single hidden layer containing 2-19 nodes. 4 input nodes-1 output node.

When the approximate depth is looked at per hidden layer as opposed to nodes in one single hidden layer, the trend is linear, still demonstrating much better behavior for the new approaches. The reason for this linear trend as opposed to exponential is that the nodes are distributed in different layers in this case. For the 20 node case (5 layers with 4 nodes each) there is no need to implement all possible combinations of the 20 nodes, but just all possible combinations of 4 nodes, as many times as layers are implemented. In this plot, RC is approximately 30x better than QPE and 5x better than Impr. QPE. The RC depth value for the 5 layer case is approximately 25.5K computational steps.

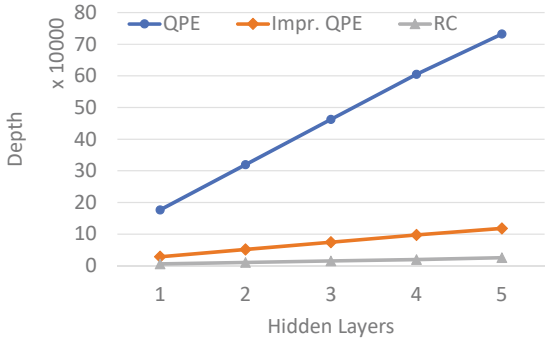


Fig. 12: Approximate depth with variation of the number of hidden layer containing 4 nodes each. 4 input nodes-1 output node.

B. Size of the training dataset

When the size of the training dataset is varied from 4 to 64 cases, again the RC case shows the best approximate depth numbers. In this case, a 4-4-3-1 BNN is implemented: four input nodes, four node hidden layer followed by a three node hidden layer, and finally one single output node. Figure 13 shows again a linear trend with the size of the training dataset when all other parameters are fixed. It is notable that in this case again, RC is approximately 30x better than QPE and approximately 5x better than QPE Impr. There is a reason for this:

For QPE and Impr. QPE, the size of the QPE register is fixed to five qubits for these experiments. The Width of the counting register in RC is also fixed to five qubits. Remember from Section III that in the QPE and Impr. QPE methods, phase was accumulated in the QPE register through repetitions of the dataset (DS) training blocks. In the QPE case, the number of repetitions doubles for each of the qubits in the QPE registers. For five qubits, that is $1+2+4+8+16=31$. For the Impr. QPE these repetitions were avoided by applying the correct phase shift for each qubit in the QPE register, for a total of 5 DS blocks. In the case of RC, the DS blocks only have to be implemented once. Hence, the $30\times$ and $5\times$ better approximate the depth of the RC case. One key conclusion is that the depth of the QPE cases is highly dependent on the precision of the phase estimation.

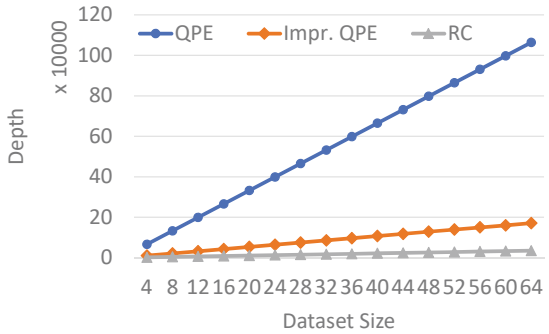


Fig. 13: Approximate depth with variations of the dataset size implemented on a 4-4-3-1 BNN.

The size of the training dataset does have an impact on the width of the implementation, as the counting register of the implementation will count correct outputs for each case and needs at most $ln(n)$ qubits where n is the size of this training dataset. Therefore, in this case, only five qubits are necessary to cover the 32-63 dataset size cases. This is represented in Figure 14.

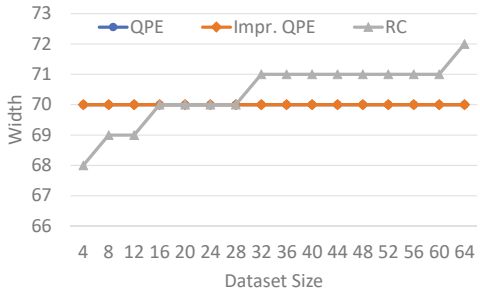
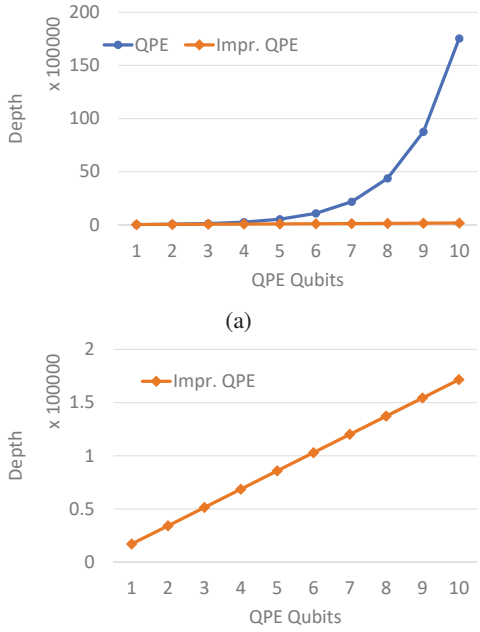


Fig. 14: Width as the size of the training dataset grows. With a smaller dataset, the counting register does not need 5 qubits to account for correct cases.

C. Precision of the phase estimation

The precision of the phase estimation is only a parameter in QPE and Impr. QPE., and depends directly on the size of the QPE register. Given that QPE requires $\sum_{i=0}^{p-1} 2^i$ for a $p-qubit$ register, the trend is exponential in this case, while it is linear in the case of Impr. QPE, which requires only p repetitions if the DS blocks. This can be seen in Figure 15



(b) Closer look at the Impr. QPE only

Fig. 15: QPE and Impr. QPE approximate depth with varying QPE register size (QPE qubits) for the 4-4-3-1 BNN.

D. Activation function

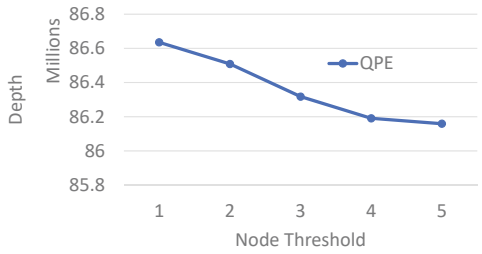
For each neuron with N inputs and corresponding N weights, the weights are applied to the inputs by performing

$a_i \oplus w_i$ of each pair. Then, an activation function will set the output to 1 when the number of 1's among the XORed pairs is higher than a threshold t. In Figure 5 for example, this threshold is set to 2 using a Toffoli gate. Other thresholds could be used, and there is more margin for different activation functions when the neuron has multiple (as in more than two) inputs.

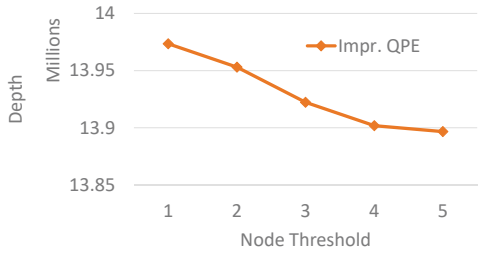
Neural Network	4-16-16-1
QPE, RC Qubits	5
Dataset Size	16
Grover's Iterations	1
Accuracy Threshold	3

TABLE II: Parameters for the variation of node activation threshold.

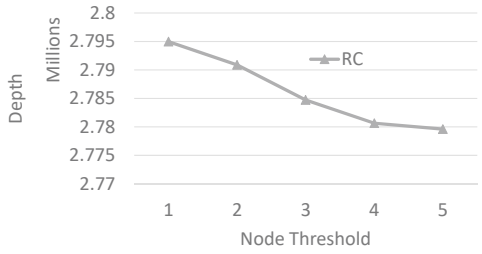
To be able to vary the activation function within a range of parameters, a BNN according to Table II is tested. The large number of nodes in the hidden layers naturally leads to neurons with multiple inputs (up to 16 possible input-weight pairs). Figure 16 shows the three different plots, one for each design, showing very similar depth trends but at very different scales. As the node threshold of the activation function grows, the depth decreases. The implementation of the activation function makes it such that if the threshold is set to 16, one multi-controlled Toffoli gate will suffice, while, if the threshold is set to eight for example, the output qubit is set to 1 for the 16 input case, for the 15 input case, for the 14 input case,..., for the 9 input case, each one requiring its own multi-controlled Toffoli gate. Hence, higher thresholds result in lower depth numbers. Higher thresholds also result in better accuracy in the proof of concept cases, as it will be shown in Section V



(a) QPE



(b) Impr. QPE



(c) RC

Fig. 16: Approximate depth with varying activation function (node threshold).

V. Convolution Edge Detection

The designs of these QPE, Impr. QPE and RC implementations were developed using IBM's tools and quantum gate model [21]. These implementations were tested on a small scale, real world problem to demonstrate the potential of Grover's search in solving the BNN problem: the convolutional edge detection image filters, as simple edge detection networks. In order to check the correctness of the implementations, they were tested using Qiskit's simulation tools without noise. The simulations were performed in RIT's Research Computing resources [22].

Two filters, 3x3 and 2x2 convolution edge detection circuits, were tested using the three versions of the QBNN training circuits. For these tests, a neural network was created to identify vertical edges from given pixel values. The output of the network is a 1 when the pixels in the filter make a vertical edge, and a zero when the pixels do not represent a vertical edge. The binary strings of weights extracted out of these quantum implementations were compared with those found in classical training. For the 3x3 case, the weights were also used to find the accuracy of the BNN trained in this manner. The findings are summarized next.

A. 2x2 Convolutional edge detection

Figure 17 shows all possible 16 combinations of the 2x2 pixel structure, highlighting the two that are vertical edges. This case is not a true test of the training of the BNN, since the training dataset was this full set of possible inputs, and hence, the prediction was tested against the same set. But this test case was a good first approximation to check the viability of the training process.

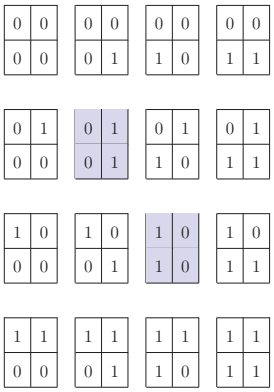


Fig. 17: 2x2 convolutional edge detection dataset.

The Neural Network is shown in Figure 18. It is a 4-2-1 NN, that is: four input nodes, one hidden layer with two additional nodes, and one final output layer with one node. This adds up to ten different weights to be determined. The details of the three implementations are shown in Table III.

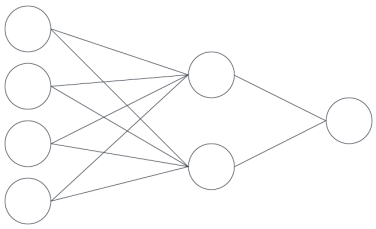


Fig. 18: 2x2 Convolution edge detection filter neural network. 4-2-1

QBNN Implementation	Width	Depth	Runtime (h:m:s)
QPE	28	41714	13:23:50
Impr. QPE	28	6770	2:46:39
RC	28	1620	1:22:05

TABLE III: 4-2-1 Network width, depth and simulation runtime (in hours:minutes:seconds).

Something to keep in mind is that with ten possible weights, the number of possible outcomes out of the Grover's search implementation adds up to 1024 binary strings. Besides the cost in time of the simulation to generate the state vector, building the histogram was challenging when done through simulated "shots", and it was also hard to represent. Instead, the state vector was manipulated to extract the probabilities of each possible binary string. The abbreviated version of the histogram is represented in Figure 19. The results of all three implementations correctly found the best weight strings for the 4-2-1 NN and the dataset shown in Figure 17. In this case, the accuracy testing resulted in 100% accuracy, which is expected, given that the training dataset is the full dataset.

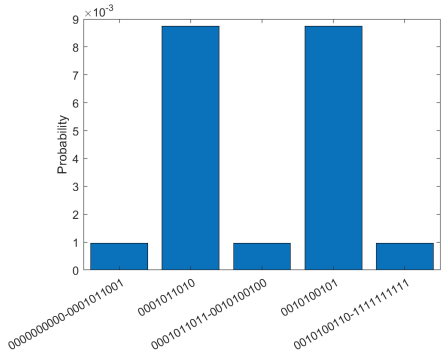


Fig. 19: 2x2 Convolution histogram results. Two states have a much higher probability than all other states, meaning that those are the two resulting weight strings. All states that are not the two amplified states are represented by the lower bars.

B. 3x3 convolutional edge detection

The 3x3 edge detection filter was implemented using the quantum neural network. With nine input values associated with the 3x3 filter, there is a potential of $2^9 = 512$ possible input combinations. However, the training dataset is reduced to 16 entries. They are shown in Figure 20, with the two that were selected as vertical edges highlighted. As shown in Figure 21, the neural network is not fully connected. There are only 3 input nodes connected to each hidden layer node instead of the full nine. This was found to be the best configuration that could be reasonably simulated while providing good accuracy.

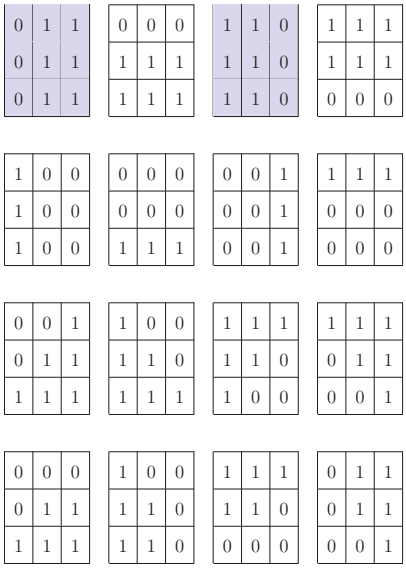


Fig. 20: 3x3 convolutional edge detection dataset.

QBNN Implementation	Width	Depth	Runtime
QPE	32	41702	∼10 Days
Impr. QPE	32	6758	∼48 hrs
RC	32	1608	22:02:49

TABLE IV: Depth, width and simulation runtime (∼Days or hours:minutes:seconds) of the 9-3-1 neural network training methods. The QPE method described in [1] was not able to complete within the allowed runtime of the research computing resources [22]. This shows the benefits of using the designs proposed.

The details of the BNN implementations are shown in Table IV. This case requires a total of 32 qubits used for all three implementations, with QPE and Impr. QPE using the same QPE register size as needed for the counting register. The QPE and Impr. QPE simulation times were only estimated, and only the RC case was fully simulated to extract the trained weight set.

There is a total of twelve weight values in this circuit, meaning that there are $2^{12} = 4096$ binary weight strings that this circuit evaluates. Similar to the 2x2 edge detection example, the state vector was manipulated to extract the probabilities associated with each of the 4096 binary strings (or basis states). The abbreviated histogram results of the training are shown in Figure 22.

Out of the set of 30 probable weights, the weight strings $[0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1]$ and $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ were also classically produced. These result in a 96.4% accuracy when testing against the complete dataset of 512 entries. Given that the classes were highly unbalanced, the full confusion matrix is reproduced in Figure 23. These numbers

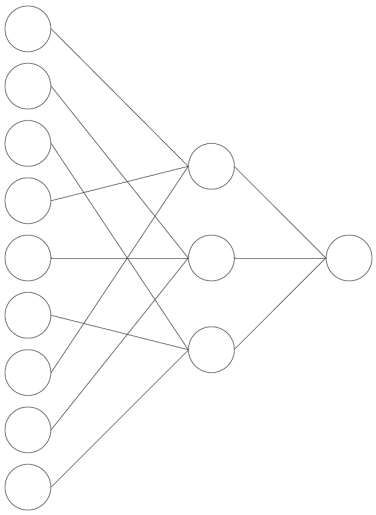


Fig. 21: 3x3 Convolution edge detection filter. This is a not-fully connected 9-3-1 BNN

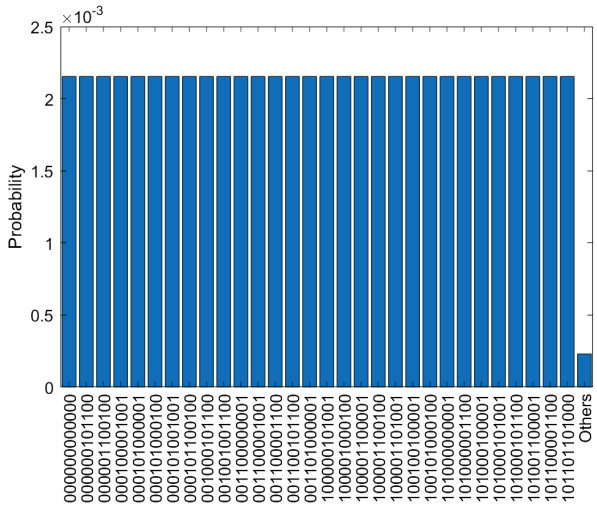


Fig. 22: 9-3-1 Neural network weight strings. These 30 weight strings are able to correctly identify all of the edges in the dataset. The 'others' bar represents the probability of each weight string that was not amplified.

result in good accuracy (0.96) and specificity (0.96) values, but bad precision (0.09) due to the 20 false negative results. However, this is not due to the quantum implementation, since both classical and quantum implementations result in the same weight strings for this NN configuration. The number of false negatives is resolved with a fully connected BNN, as opposed to this partially connected one.

VI. Conclusion

In this paper, we have discussed two different quantum implementations for the training of binary neural networks, based on the Grover's search algorithm. As we look at the post-NISQ era, depth becomes more relevant, as a first approximation of the performance of the circuit implementations. For

		Actual Values	
		Positive	Negative
Predicted Values	Positive	2	20
	Negative	0	490

Fig. 23: 3x3 confusion matrix results on all 512 test cases.

current quantum systems, the depths discussed in this paper are unreachable, but this work puts the focus on the reductions in number of approximate steps that are achieved when different approaches are used.

The oracles of these two Grover's search implementations are based on two different approaches. Quantum Phase Estimation (QPE) accumulates phase rotations on a register when the output of training that data point out of a dataset is correct. The phase will then be examined by the Grover's oracle to select the best possible weight string out of the superposition of all possible strings. An improved version of QPE was also proposed and implemented, with more efficient phase accumulation. The Register Counting (RC) approach uses a much more "classical" perspective, simply counting the number of correct cases.

The effectiveness of this training approach for the proof of concept case —implementing convolutional edge detection— was proven to be the same as could be achieved classically with this network design. The network was not fully connected, and therefore, a number of false negative results were found. The most significant finding is reflected in the simulation time, which was prohibitive in the original design. The QPE approach required repeated implementations of the training dataset, while the RC implementation can train the NN with only one implementation of the dataset block. This results in a reduction of the simulation time of this quantum circuit from approximately 10 days in the original case to 22 hours in the RC case.

As quantum systems evolve to support more complex computations with higher number of qubits, it is necessary to identify the best practices that result in increased parallelism of the computation and reduced depth of the circuit. Future work will look into exact depth calculations, and the trade-offs between phase accumulation repeated DS implementation and the incrementing implementations of the RC case.

REFERENCES

[1] Y. Liao et al, "Quantum Speed-up in Global Optimization of Binary Neural Nets," *New J. Phys.*, vol. 23, no. 063013, 2021.

[2] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of Continuous-Valued Deep Networks to Efficient Event-driven Networks for Image Classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[3] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent Pre-trained Deep Neural Networks for Large-Vocabulary Speech Recognition," *IEEE Transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 30–42, 2011.

[4] F. Seide, G. Li, and D. Yu, "Conversational Speech Transcription Using Context-dependent Deep Neural Networks," in *Twelfth annual conference of the international speech communication association*, 2011.

[5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[6] M. Cerezo, K. Sharma, A. Arrasmith, and P. J. Coles, "Variational Quantum State Eigensolver," *npj Quantum Information*, vol. 8, no. 1, pp. 1–11, Sep. 2022. [Online]. Available: https://www.nature.com/articles/s41534-022-00611-6

[7] J. Tilly, H. Chen, S. Cao, D. Picozzi, K. Setia, Y. Li, E. Grant, L. Wossnig, I. Rungger, G. H. Booth, and J. Tennyson, "The Variational Quantum Eigensolver: A Review of Methods and Best Practices," *Physics Reports*, vol. 986, pp. 1–128, Nov. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0370157322003118

[8] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, and K. Temme, "Quantum Optimization using Variational Algorithms on Near-term Quantum Devices," *Quantum Science and Technology*, vol. 3, no. 3, p. 030503, Jul. 2018. [Online]. Available: https://iopscience.iop.org/article/10.1088/2058-9565/aab822

[9] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," 2014, publisher: arXiv Version Number: 1. [Online]. Available: https://arxiv.org/abs/1411.4028

[10] F. Sauvage, M. Larocca, P. J. Coles, and M. Cerezo, "Building Spatial Symmetries into Parameterized Quantum Circuits for Faster Training," Jul. 2022, arXiv:2207.14413 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2207.14413

[11] M. C. Caro, H.-Y. Huang, M. Cerezo, K. Sharma, A. Sornborger, L. Cincio, and P. J. Coles, "Generalization in Quantum Machine Learning from Few Training Data," *Nature Communications*, vol. 13, no. 1, p. 4919, Aug. 2022. [Online]. Available: https://www.nature.com/articles/s41467-022-32550-3

[12] E. Fontana, M. Cerezo, A. Arrasmith, I. Rungger, and P. J. Coles, "Non-trivial Symmetries in Quantum Landscapes and their Resilience to Quantum Noise," *Quantum*, vol. 6, p. 804, Sep. 2022, arXiv:2011.08763 [quant-ph, stat]. [Online]. Available: http://arxiv.org/abs/2011.08763

[13] M. Larocca, P. Czarnik, K. Sharma, G. Muraleedharan, P. J. Coles, and M. Cerezo, "Diagnosing Barren Plateaus with Tools from Quantum Optimal Control," *Quantum*, vol. 6, p. 824, Sep. 2022, arXiv:2105.14377 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2105.14377

[14] Q. T. Nguyen, L. Schatzki, P. Braccia, M. Ragone, P. J. Coles, F. Sauvage, M. Larocca, and M. Cerezo, "Theory for Equivariant Quantum Neural Networks," Oct. 2022, arXiv:2210.08566 [quant-ph, stat]. [Online]. Available: http://arxiv.org/abs/2210.08566

[15] L. Schatzki, M. Larocca, Q. T. Nguyen, F. Sauvage, and M. Cerezo, "Theoretical Guarantees for Permutation-Equivariant Quantum Neural Networks," Nov. 2022, arXiv:2210.09974 [quant-ph, stat]. [Online]. Available: http://arxiv.org/abs/2210.09974

[16] L. Leone, S. F. E. Oliviero, L. Cincio, and M. Cerezo, "On the practical usefulness of the Hardware Efficient Ansatz," Nov. 2022, arXiv:2211.01477 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2211.01477

[17] C. Moussa, M. H. Gordon, M. Baczyk, M. Cerezo, L. Cincio, and P. J. Coles, "Resource Frugal Optimizer for Quantum Machine Learning," Nov. 2022, arXiv:2211.04965 [quant-ph, stat]. [Online]. Available: http://arxiv.org/abs/2211.04965

[18] L. K. Grover, "A Fast Quantum Mechanical Algorithm for Database Search," *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC 96*, 1996.

[19] A. Y. Kitaev, "Quantum Measurements and the Abelian Stabilizer Problem," *arXiv:quant-ph/9511026*, 1995.

[20] B. A. Wrighter, "Improved Grover's Implementation of Quantum Binary Neural Networks," 2023. [Online]. Available: https://scholarworks.rit.edu/theses/11459

[21] IBM, "IBM Quantum Experience." [Online]. Available: https://www.research.ibm.com/ibm-q/

[22] Rochester Institute of Technology, "Research computing services," 2022. [Online]. Available: https://www.rit.edu/researchcomputing/