Quantum advantage in training binary neural networks

Yidong Liao,[1] Oscar Dahlsten,[2,1,*] Daniel Ebler,[1] and Feiyang Liu[1]

[1] *Institute for Quantum Science and Engineering, Department of Physics, Southern University of Science and Technology (SUSTech), Shenzhen, China.*
[2] *Wolfson College, University of Oxford, Linton Road, Oxford OX2 6UD*

Neural networks have become a corner stone of artificial intelligence. Once a task is defined, a neural network needs to undergo a training phase in order to calibrate the network parameters. On a classical computer, the number of training cycles is strongly correlated with the amount of parameters in the neural network, making an optimization of the parameters computationally expensive. Here, we propose the a fully quantum training protocol for quantum binary neurons. We show that for special cases this protocol yields a quadratic advantage over commonly used classical training methods, and numerics suggests that this advantage is generic for most instances. The source of this advantage is the possibility to run training cycles in a quantum superposition. Further, we present an extension of the training of single quantum binary neurons to feedforward binary neural networks.

1. INTRODUCTION

Since their proposal in the 1940s [1], Neural networks (NNs) have been proven immensely successful for a wide variety of tasks, notably including pattern recognition [2, 3], language processing [4, 5], medicine and pharmaceutics [6–9], and finance [10]. The basic idea of NNs is to use elementary computational units, called neurons, to span a network mimicking the human brain. Due to this analogy, neural nets are widely used in the field of artificial intelligence [11–13].

Computers were originally designed to process information according to a pre-defined algorithm. In contrast, NNs are able to learn how to process data themselves. This learning mechanism is refered as a training, during which parameters in the network are calibrated through good sets of training data. As the number of parameters is typically large, the training phase suffers from long run-times [14] and consumes large amounts of memory [15, 16]. Recently, simplified models such as binary neural networks (BNNs) [17] were introduced to resolve the performance problem. Yet, the performance problem in the processing time still exhausts large amounts of computational resources.

A novel approach to NNs is based on quantum technology, which has been shown to achieve performances beyond the possible of current classical implementations [18–21]. These so-called quantum neural networks (QNNs) strive not only for a more efficient learning routine [22], but also for learning [23, 24] and identifying new quantum protocols [25].

While the information is encoded in quantum systems, the training in recent proposals has remained classical. An appealing possibility is that quantum effects, such as superpositions and entanglement, can also improve the efficiency of trainings. The question for a quantum advantage is of substantial importance for the performance of neural networks, especially for complex tasks for which networks with large amounts of neurons are typically used.

In this work we propose the to our knowledge first fully quantum training protocol for binary neural networks. The protocol uses a generalization of Grover's search algorithm as a subroutine to find the optimal system parameters. The basic idea of the protocol is to run multiple training cycles in a quantum superposition. We show that this leads to a provably quadratic advantage over classical training algorithms for special instances. Strikingly, it turns out that numerics suggest this advantage to be generic for most cases.

The Letter is structured as follows. After a brief introduction to BNNs and Grover's search algorithm, we study a quantum extension of a binary neuron introduced in [25]. Then, the fully quantum training protocol is presented. Finally, an analysis of the performance in comparison to classical binary neurons is given and an extension from single neurons to feedforward quantum BNNs suggested.

2. PRELIMINARIES

The quantum extension of the classical binary neural network requires the knowledge of classical binary neural networks and Grover's search algorithm. Here, we give a technical introductory section is to give a brief overview of these two concepts.

2.1 Classical feedforward binary neural networks

Artificial neural networks (NNs) is a computational framework used to process data without pre-defining an algorithm. The task specific algorithm is constructed from the input data alone, which allows NNs to operate universally, up to restrictions due to the architecture of the NN. The construction of the algorithm is based on examples, often referred to as training data. The basic

building block of an NN, the neuron, is constructed in correspondence with the biological neuron in the human brain. Roughly speaking, a neuron acts as a computational cell by receiving a inputs and processing them according to a pre-set rule. Then, the output is forwarded to the connected neurons. Modelling the structure as a graph, the edge between two neurons carries a weight w, which describes the importance of the connection. We focus in this Letter on feedforward NNs (FNNs), for which the neurons are grouped into sequential layers. Every neuron in layer r shares an edge to every neuron in layer $r-1$ and $r+1$, while the neurons within a layer or across neighbouring layers do not interact (see Fig. 1). The first layer obtains a set of inputs, processes it and forwards it to the next layer. The cascade of computations terminates when the last layer yields the final output. Note, that the amount of neurons and the processing operation in a neuron can be different in each layer.

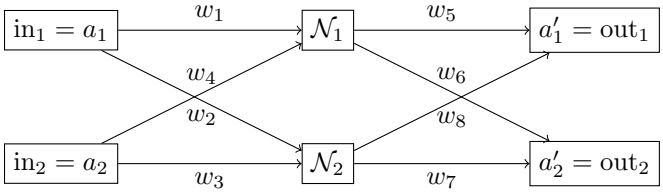

FIG. 1. *Simple instance of a FNN:* the inputs a_1, a_2 are forwarded to the neurons $\mathcal{N}_1$, $\mathcal{N}_2$, which process them and combine their outputs to a'_1, a'_2. Each of the three layers is connected with edges carrying weights $w \in \{w_i\}_{i=1}^{8}$.

The impressive capability of NNs to learn certain tasks like pattern recognition [2, 3] and classification [26] comes with a high computational cost. There are several reasons for this. First of all, the inputs to a neuron and weights w can be arbitrary real numbers. Second, the processing in a neuron implements an arbitrary function f based on the operations "addition", "subtraction" and "multiplication". Only recently, Bengio *et al* [17] proposed with binary neural networks (BNNs) a simplified version of NNs, which can process data faster and with fewer resources than for NNs with continuous parameters. In particular, BNNs limit the outputs of the neuron and the weights to be bits $b \in \{1, -1\}$. In addition, the operations in a neuron is restricted to two operations: applying the `XNOR` gate, which is only one if both inputs to the gate are zero, and counting the bits with a `bitcount` operation.

A typical processing f is given by the sign function, acting on an input x as

$$f(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} .$$

As input values below a certain threshold yield the same constant value, while inputs above the threshold "activate" the neuron to the value $+1$, such a processing is called activation function.

Remarkably, despite the strong nature of the simplifications the accuracy of BNNs has been shown to be similar to NNs [17].

Training a NN: gradient descent and backpropagation

During the training phase of an NN, a set of dummy input data with known outputs is sent through the NN and the outputs a' are compared to the desired outputs a^*. The weights w in the NN are refined according to the deviation of the output from the ideal ones. To quantify the deviation, a task-specific cost function $C(a', a^*)$ is defined. A widely used approach is to approach the minimum cost through the method of gradient descent, which updates the weight w_{ij} between neuron i and neuron j from the t-th iteration of the NN to the next one as

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}} . \tag{1}$$

Here, η is a constant, usually referred to as the learning rate.

The direct evaluation of the term $\partial C/\partial w_{ij}$ is computationally expensive: even for BNNs the gradient of the cost function is in general real-valued. A frequently used method is backpropagation, which approximates the derivative by evaluating a sequence of local computations. On a high level, the chain rule for derivatives allows to rewrite

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial o^{(n)}} \frac{\partial o^{(n)}}{\partial i^{(n)}} \frac{\partial i^{(n)}}{\partial o^{(n-1)}} \cdots \frac{\partial i^{(m)}}{\partial w_{ij}} , \tag{2}$$

where where we defined $o^{(n)} = \{o_j^{(n)}\}$ as the set of all outputs of the neurons in the n-th layer, and $i^{(n)} = \{i_i^{(n)}\}$ as the corresponding set of inputs. Then, each term can be evaluated locally. The multiplication finally yields an approximation of the partial derivative in Eq. (1). This concept can be applied to arbitrary NNs, as long as the processing f of each neuron is differentiable.

2.2 Grover's search algorithm

For an unsorted list of length N, identifying the index of an element requires $O(N)$ computational steps for the best known classical algorithms. If quantum effects are allowed for, only $O(\sqrt{N})$ steps are needed. This is the result of the famous search algorithm proposed by Grover [19], who proved a speedup by running multiple tests in parallel by initializing items of a list in a quantum superposition.

Let us encode the items of a list into quantum states $|0\rangle, |1\rangle, \ldots, |N-1\rangle$. Furthermore, let $|\omega\rangle \in \{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$ be the state corresponding to the index of the search query. The algorithm then proceeds as follows:

- Initialize the register in the superposition

$$\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\} \mapsto |X\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

- For $O(\sqrt{N})$ times, repeat:
  1. Apply the quantum oracle Λ
  2. Apply the diffusion transform D
- Apply the measurement Ω to the register

The quantum oracle Λ flips the sign of the state $|x\rangle$ in the superposition if it coincides with the correct index ω, and does nothing otherwise. The action of the diffusion operator D is described by the matrix with entries

$$D_{i,j} = \begin{cases} 2/N & \text{if } i \neq j \\ 2/N - 1 & \text{if } i = j \ . \end{cases} \tag{3}$$

The diffusion operator amplifies the predicted list index x^*, such that the concluding measurement Ω yields the outcome $x^* = \omega$ with high probability. A generalization of Grover's search is used later to identify the optimal set of weights in a quantum BNN.

3. QUANTUM BINARY NEURONS AND QUANTUM TRAINING

In the following, we propose a novel training protocol utilizing quantum effects. More precisely, applying the superposition principle to the network parameters allows to parallelize multiple training cycles.

3.1 Design of a quantum binary neuron

A classical binary neuron (CBN) implements a nonlinear function, which takes a set of input values and produces a single output. Quantum mechanics, on the contrary, is a linear and reversible theory. Following the recent proposal of Ref. [25], we present an extension of CBNs, which is compatible with quantum theory and includes CBNs as a special case.

Finding a reversible extension: Let us consider the elementary instance of a classical neuron depicted in Fig. 2.

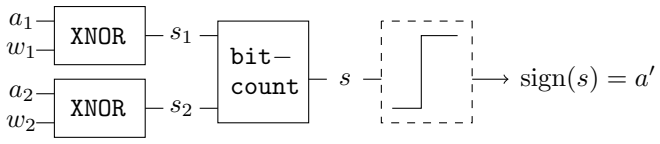

FIG. 2. *Functioning of a CBN:* an `XNOR` operation multiplies the inputs a_1, a_2 with the corresponding weights w_1, w_2. The results $s_1 = w_1 a_1$, $s_2 = w_2 a_2$ are forwarded to the `bitcount`, which outputs $s = s_1 + s_2$. Finally, the sign function determines the activation of the neuron.

The neuron takes two inputs a_1 and a_2 and has weights w_1 and w_2 on the edges. An `XNOR` gate multiplies the inputs with the weights, and the operation `bitcount` sums the results to the value $s = w_1 a_1 + w_2 a_2$. Finally, the function $f(s)$ determines the activation value a' of the neuron. Now, we introduce ancillary inputs and outputs for each operation in the CBN, such that the number of inputs coincides with the number of outputs. These ancillas carry the output values of the operations, while the inputs are preserved. This yields a reversible embedding $U_{\texttt{XNOR}}$ of `XNOR` and $U_{\texttt{bit+}}$ of `bitcount`, see Fig. 3. Here, $U_{\texttt{bit+}}$ includes both the `bitsum` operation and the activation through the sign function.

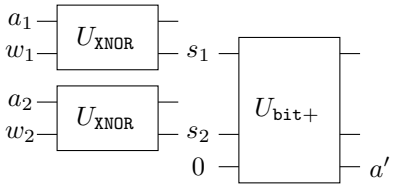

FIG. 3. *Reversible embedding of CBN:* the gate $U_{\texttt{XNOR}}$ takes an input-weight pair (a_i, w_i) and encodes their multiplication value in the ancillary output s_i, with $i \in \{1, 2\}$. On the second output, either a_i or w_i is forwarded, making the full operation reversible. The gate $U_{\texttt{bit+}}$ takes the inputs s_1, s_2 and the ancilla 0 and encodes the activation $\text{sign}(s)$ in the output a', while preserving s_1 and s_2.

The basic operations in a QBN: in order to find a quantum circuit implementation of a QBN, we reduce the operations $U_{\texttt{XNOR}}$ and $U_{\texttt{bit+}}$ to quantum gates.

The multiplication operation can be achieved by combining a NOT gate on each input $|a_i\rangle$, controlled by the state of the corresponding weight $|w_i\rangle$, $i \in \{1, 2\}$ (see Fig. 4).

It can be easily seen from the truth table of the `XNOR` and CNOT gate that the operations indeed coincide, if the following change of basis is made: we shift the bits as $1 \to 0$ and $0 \to 1$. The gate $U_{\texttt{bit+}}$ is realized by the Toffoli gate [27] on the three input qubits.

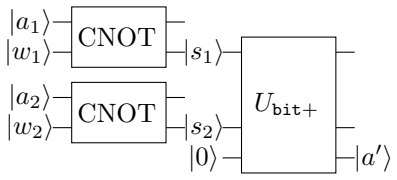

FIG. 4. *Functioning of QBN:* the input data is encoded into quantum states. The multiplication succeeds by the CNOT gate and outputs the states $|s_1\rangle$, $|s_2\rangle$. Finally, the Toffoli gate executes $U_{\mathtt{bit+}}$, leading to the output state $|a'\rangle$.

3.2 The quantum training protocol

All proposals of quantum neural networks were accompanied to our knowledge with a fully classical [28, 29] or semi-classical [23] training phase. As the training is a determining factor of the over-all performance of the network, it is highly desirable to find more cost efficient ways for training the network.

Here, we propose the to our knowledge first fully quantum training protocol for QBNs. We restrict the training to a single neuron and discuss the extension to networks later.

Marking the target weights: The first subroutine marks good weights $w_1^*, w_2^*, \ldots, w_k^*$ from the set of all weights. This is done similarly to Grover's algorithm, where the oracle Λ marks good indexes by flipping their sign.

In the following, we discuss the marking subroutine step by step and refer to Fig. 5.

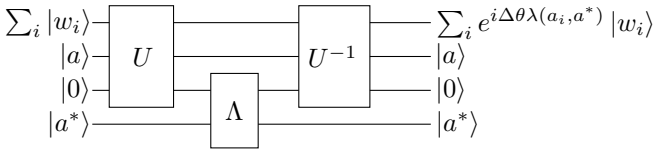

FIG. 5. *Marking of the target weights:* the action U of the QBN acts on the input $|a\rangle$, ancilla $|0\rangle$ and coherent weight state $|W\rangle$. The output $|a'\rangle$ is compared to the optimal out $|a^*\rangle$ by the oracle Λ. To decouple the weights, an uncomputation U^{-1} is applied. This gives the weighted vector $|W'\rangle$.

Step 1: initialize all N weights $\{w_i\}$ in a coherent superposition $|W\rangle = 1/\sqrt{N}\sum_i |w_i\rangle$. This yields the total initial state

$$|\mathrm{In}_1\rangle = |W\rangle|a\rangle|a^*\rangle|0\rangle \ , \tag{4}$$

where $|a^*\rangle$ is the desired output state and $|0\rangle$ is an ancillary qubit. Due to this superposition, all weight states act simultaneously on the input $|a\rangle$. Afterwards, the QBN acts on $|W\rangle$, the input $|a\rangle$ and an ancilla $|0\rangle$, yielding the output $|a'\rangle$. The overall state then reads

$$|\mathrm{Out}_1\rangle = \sum_i |w_i, a_i, a'\rangle|a^*\rangle \ , \tag{5}$$

where the QBN transforms the input $|a\rangle$ into $|a_i\rangle$ if the control state is $|w_i\rangle$. Note that the systems in Eq. (5) are entangled.

Step 2: call the oracle Λ to compare the output $|a'\rangle$ with the desired output $|a^*\rangle$. If the two states coincide then Λ adds a small phase $\Delta\theta$ to $|a'\rangle$. This leads to

$$|\mathrm{Out}_2\rangle = \frac{1}{\sqrt{N}}\sum_i |w_i, a_i, a'\rangle e^{i\Delta\theta\lambda(a',a^*)}|a^*\rangle \ , \tag{6}$$

where $\lambda(a', a^*) = 1$ if the oracle was successful and zero otherwise.

Step 3: decouple the weights. By reversing the unitary action U of the QBN, the weights get decoupled and are in the state

$$|W'\rangle = \frac{1}{\sqrt{N}}\sum_i e^{i\Delta\theta\lambda(a',a^*)}|w_i\rangle \ . \tag{7}$$

Step 4: accumulation of phases. The state $|W'\rangle$ is used as the initial weight state for a new round of marking with a new set of input data. By repeating Step 1 - 3 for n times, the small phases add up for the elements. The phases $\Delta\theta$ are chosen such that $n\Delta\theta = \pi$. Hence, the most frequently marked weights are closest to a prefactor of -1.

While in Grover's search the oracle contributes an exact sign flip to good items, the pre-factors obtained after the marking subroutine introduced above are generally different from ± 1. More precisely, for n rounds of marking, the quality of each weight string is quantified on a fine grained scale from 0 to n. This way, differences among good weight strings bad weight strings are accounted for. A binary sign flip as in Grover's search, on the contrary, does not make any statement about how good the good weights and how bad the bad weights are. For this reason, it is easy to find examples where Grover's search for optimal weights fails, whereas the generalized phase approach always succeeds.

Amplitude amplification of the target weights: to increase the chance of picking the optimal weights, we amplify the pre-factors of the target weights. To this purpose, we use the diffusion operator D defined in Eq. (3).

More precisely, after each cycle consisting of n marking iterations the diffusion matrix D is applied to the output state of the weights. Hence, after m rounds of amplification the state of the weights is given

by $(DM)^m|W\rangle$, where M denotes the action of a full marking cycle on the state of the weights. Supported by numerical evidence, the protocol reaches the optimal amplification after $m = \sqrt{N}\pi/4$ rounds, in agreement with the optimal number of amplifications in Grover's search. In Fig. 6 we present an example of the quantum circuit for a single neuron.

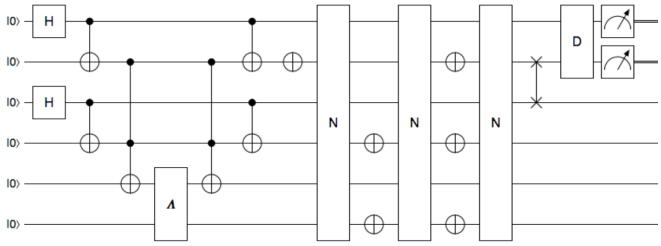

FIG. 6. *Quantum circuit for the training of one neuron:* the input qubits (from top to bottom) are $|w_1\rangle$, $|a_1\rangle$, $|w_2\rangle$, $|a_2\rangle$, ancilla qubit $|0\rangle$, and $|a^*\rangle$. In the first step, Hadamard gates are applied to $|w_1\rangle$ and $|w_2\rangle$, creating a coherent superposition of two weights for both $|w_1\rangle$ and $|w_1\rangle$. In step two and three, the circuit depicts the unitary actions of the neurons. After the oracle Λ is called, in step five and six the uncomputation of the neurons is performed. The full action of step two to six is denoted by the gate N. New input data is initialized in the circuit by applying X gates on $|a_1\rangle$, $|a_2\rangle$, $|a^*\rangle$ (we use $\oplus$ to denote X gate), and N is applied after for each new input. In this example we are using a training set $\{(a_1, a_2, a^*)\} = \{(0,0,0),(1,0,0),(0,1,1),(1,1,1)\}$. After the phase accumulation of all the training data, the diffusion transform D is applied on the two weight qubits (after a swap gate) to amplify the amplitude of the optimal weight string. Note that only a single amplification is necessary, as we have four possible choices of weights, and by Grover's search the optimal number of amplifications is given by $\sqrt{4}\pi/4$. Finally, measurements are performed to read out the probabilities of weight strings.

3.3 Performance

One way to quantify the time requirement for training is to count the number of calls to the neuron/neural net. Additionally, the number of calls to the oracle in our algorithm should also be included in the count. Each such call involves passing a quantum object through a circuit. The probability of a system being lost, or the amount of noise more generally, will often scale as the number of calls. Moreover, the cost to experimenters is often the amount of time the experiment takes, which may also scale as the number of calls.

In the quantum training presented above there are two separate contributions to the number of calls: (i) for each input i there is a call to the neuron, with a total number of N_i calls, and (ii) the Grover-like search protocol on the weight state is expected to take a number of calls scaling as the $\sqrt{N_w}$, with N_w the number of weight states (for t weights there are 2^t weight states). Thus the total number of calls is expected to be $N_i \times \sqrt{N_w}$.

For a comparable classical training, we consider searching an unstructured list, the list being that of weight state, input pairs with assignment of a cost value to each of these pairs. Comparing it to searching a structured list, e.g. via gradient descent, appears unfair as the Grover search can handle unstructured lists. Such a classical search amounts to searching a list of size $N_i N_w$ which requires checking either all or some fraction of the entries in the list, depending on one's error tolerance. In our set-up each evaluation of the list corresponds to jointly inserting a given input and weight state.

Because of the design of the neural net, each layer has at least as many weights as inputs. If all the neurons are connected to the each neuron in the next layer, we have N_i^2 weights per layer and LN_i^2 weights in total, for L layers. Then the quantum training is proportional to N_i steps and the classical training to N_i^3 steps. This polynomial speed-up can be significant in terms of real run-time. Compare for example $10^3 s$ (approx. 17minutes) with $10^6 s$ (278hrs).

Note, that classically testing all possible weight strings yields the optimal string at the end of the training. In the quantum protocol, the outcome of the final measurement on the amplified weight states depends on the set of training data. For the special case where a small amount of weight strings get marked in every round (corresponding to a pre-factor of -1) and the remaining strings are not called (corresponding to a pre-factor of 1) we retrieve Grover's search. Hence, in these situations, the advantage is exactly the one described above.

Intriguingly, numerics showed the same quantum advantage for all instances in which most weight strings have less than $N/2$ phase counts. This yields the conjecture that for all these cases we have the same quantum advantage as Grover's search on the special instances discussed above. On the other hand, Grover is known to fail when the number of good weights is equal or larger than half of the number total weight strings. We observe the same behaviour for our phase accumulation approach.

4. GENERALIZATION TO QUANTUM FBNNS

So far, the training of a single QBN was studied. Here, an extension to quantum FBNNs is suggested. Instead of identifying the optimal weights for the inputs of a single neuron the goal is to find the optimal string of weights $\ell = \tilde{w}_1 \tilde{w}_2 \dots \tilde{w}_j$ for each link among two neurons in the network. It is apparent that the design of quantum FBNNs is more subtle. While in FBNNs the output of a neuron is copied and forwarded to every input of the

next layer, this is not possible in the quantum case due to the no-cloning theorem [30]. Instead, a so-called fan-out operation is used which creates imperfect copies of the output state [25]. Clearly, the quality of the copies decreases with the their number. One option of such a fan-out operation is the CNOT gate, which creates a single perfect copy if the input state is either $|0\rangle$ or $|1\rangle$, and copies parts of the information otherwise.

For a classical FBNN, there are 2^j different choices of ℓ. Hence, testing all of them for optimality becomes infeasible very quickly. Using the proposed quantum training, it is sufficient to initialize a single superposition of all weight states, as $|W\rangle = 1/\sqrt{2^j}\sum_{\underline{a}}|\underline{a}\rangle$, where $\underline{a} = a_1, a_2, \ldots, a_j$. The oracle Λ then acts as

$$\Lambda(\underline{a}', \underline{a}^*) = \begin{cases} e^{i\Delta\theta} & \text{if } \underline{a}' = \underline{a}^* \\ 1 & \text{else .} \end{cases} \tag{8}$$

A final j-outcome measurement Ω outputs the optimal weights after the marking and amplification subroutines. The single neuron training method of Fig. 5 works also for this network design, with the unitaries U replaced by the total network unitary. In Appendix we discuss one example of QBNN training.

5. SUMMARY AND OUTLOOK

In this Letter we presented the to our knowledge first fully quantum training protocol for quantum binary neural networks. Different from other proposals, the protocol quantizes both the training data and the weights of the network. It was shown that this extension to the quantum domain yields in most cases to a quadratic speedup over classical protocols by running multiple trainings in a quantum superposition. While the training method was analyzed for single quantum binary neurons, an extension to quantum feedforward neural networks was suggested.

This novel approach opens drastically new possibilities, for example increasing the training data set or the size of the neural networks to improve the performance.

Several questions arise from our work. The generalised Grover search algorithm deserves further analysis as to performance in the general cases where it is not identical to Grover. As for these cases numerics strongly hint at the same advantage as for the Grover case, a deeper understanding of the training performance is an important direction of research. It is also necessary to consider searches of structured lists, possibly using the tools of quantum random walks to replace Grover's search with alternative algorithms for that case. Furthermore, experimental implementations in quantum computing type set-ups and optimization of the scheme in terms of experimental realizability sound appealing. Finally it is important to identify in which useful real-world tasks this training speed advantage can be best put to use.

ACKNOWLEDGEMENTS

We would like to thank Yu Hao and Jon Allcock for useful discussions.

* Correspondence: Oscar Dahlsten (dahlsten@sustc.edu.cn)

[1] W. S. McCulloch and W. Pitts, The bulletin of mathematical biophysics **5**, 115 (1943).
[2] C. M. Bishop *et al.*, *Neural networks for pattern recognition* (Oxford university press, 1995).
[3] N. M. Nasrabadi, Journal of electronic imaging **16**, 049901 (2007).
[4] R. Collobert and J. Weston, in *Proceedings of the 25th international conference on Machine learning* (ACM, 2008) pp. 160–167.
[5] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, in *Eleventh Annual Conference of the International Speech Communication Association* (2010).
[6] W. G. Baxt, Annals of internal medicine **115**, 843 (1991).
[7] M. Karabatak and M. C. Ince, Expert systems with Applications **36**, 3465 (2009).
[8] J. Zupan, J. Gasteiger, and J. Zupan, (1999).
[9] S. Agatonovic-Kustrin and R. Beresford, Journal of pharmaceutical and biomedical analysis **22**, 717 (2000).
[10] R. R. Trippi and E. Turban, *Neural networks in finance and investing: Using artificial intelligence to improve real world performance* (McGraw-Hill, Inc., 1992).
[11] S. E. Grossberg, *Neural networks and natural intelligence.* (The MIT press, 1988).
[12] S. S. Haykin, *Neural networks and learning machines*, Vol. 3 (Pearson Upper Saddle River, 2009).
[13] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach* (Malaysia; Pearson Education Limited,, 2016).
[14] A. Blum and R. L. Rivest, in *Advances in neural information processing systems* (1989) pp. 494–501.
[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, Tech. Rep. (California Univ San Diego La Jolla Inst for Cognitive Science, 1985).
[16] P. J. Werbos, Proceedings of the IEEE **78**, 1550 (1990).
[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, in *Advances in neural information processing systems* (2016) pp. 4107–4115.
[18] D. Deutsch and R. Jozsa, Proc. R. Soc. Lond. A **439**, 553 (1992).
[19] L. K. Grover, in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (ACM, 1996) pp. 212–219.
[20] P. W. Shor, SIAM review **41**, 303 (1999).
[21] S. Bravyi, D. Gosset, and R. Koenig, arXiv preprint arXiv:1704.00690 (2017).
[22] M. Schuld, I. Sinayskiy, and F. Petruccione, Quantum Information Processing **13**, 2567 (2014).
[23] K. H. Wan, F. Liu, O. Dahlsten, and M. Kim, arXiv preprint arXiv:1806.10448 (2018).
[24] M. E. Morales, T. Tlyachev, and J. Biamonte, arXiv preprint arXiv:1805.09337 (2018).

[25] K. H. Wan, O. Dahlsten, H. Kristjánsson, R. Gardner, and M. Kim, npj Quantum Information **3**, 36 (2017).
[26] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas, Artificial Intelligence Review **26**, 159 (2006).
[27] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, Physical review A **52**, 3457 (1995).
[28] D. Yuan, L. Cai, M. Li, C. Liang, and X. Hou, Neuro-Quantology **16** (2018).
[29] J. Zhao, Y. Sun, F. Feng, F. Zhao, D. Sui, and J. Xu, in *IOP Conference Series: Earth and Environmental Science*, Vol. 108 (IOP Publishing, 2018) p. 032008.
[30] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," (2002).

Appendix: quantum circuit for QBNN training

Here we present one example of a quantum circuit for QBNN training. Specifically, we look at the quantum instance of the following classical neural network

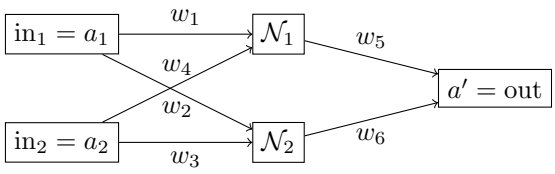

FIG. 7. *Example of classical neural network:* the inputs a_1, a_2 are forwarded to the neurons $\mathcal{N}_1$, $\mathcal{N}_2$, which process them and combine their outputs to a'. Each of the three layers is connected with edges carrying weights $w \in \{w_i\}_{i=1}^{6}$.

The quantum circuit of the corresponding QBNN looks as follows

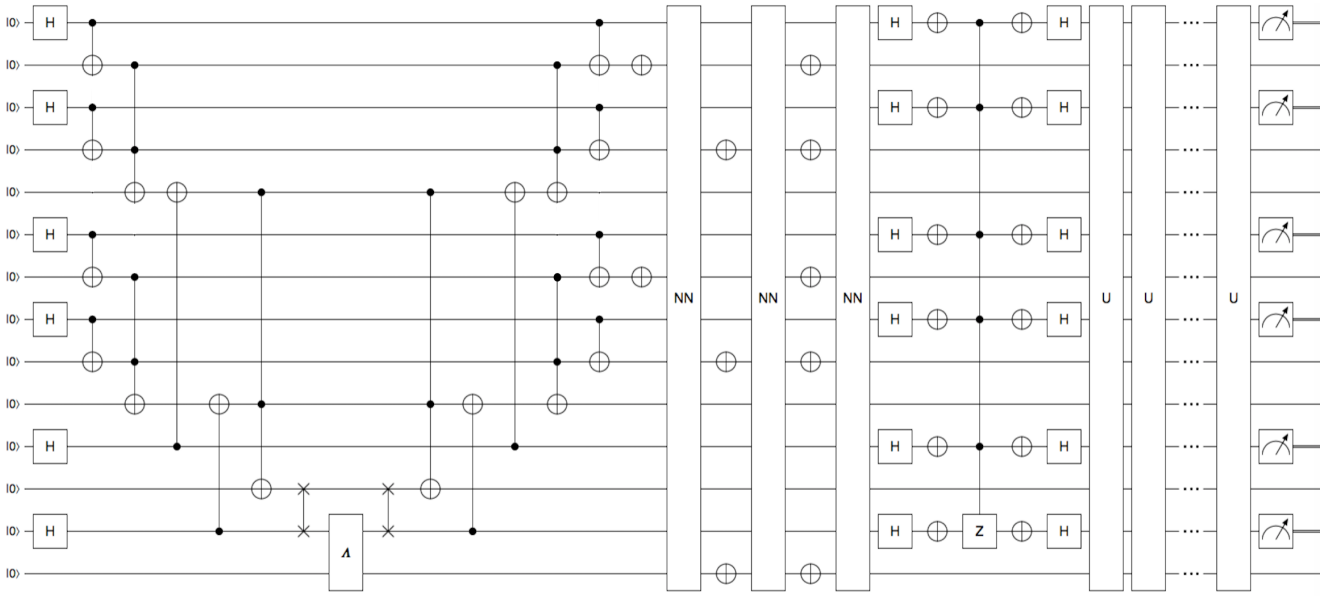

FIG. 8.

The circuit takes the inputs $|0\rangle^{\otimes 14}$, which correspond to (from top to bottom) $|w_1\rangle, |a_1\rangle, |w_2\rangle, |a_2\rangle$, ancilla qubit $|0\rangle, |w_3\rangle, |a_1\rangle, |w_4\rangle, |a_2\rangle$, ancilla qubit $|0\rangle, |w_5\rangle$, ancilla qubit $|0\rangle, |w_6\rangle$, and $|a^*\rangle$. In step 1, Hadamard gates are applied to the states $|w_1\rangle - |w_6\rangle$ to create the coherent superposition of weights $(|0\rangle + |1\rangle)/\sqrt{2}$ for every weight state $|w_1 > |w_2\rangle, \ldots, |w_6\rangle$. Then, in step 2-6 the circuit depicts the unitary action of the neurons, followed by a swap operation in step 7. Then, in step 8 the oracle Λ is called, adding a phase if the state of the system coincides with the desired output a^*, as in Eq. (8). Then, step 9-14 is the uncomputation of the action of the neurons.

The full action of the steps 2-14 is thereafter denoted by the gate NN. In order to initialize new input data, bit flips (denoted as $\oplus$ in the circuit) are applied on the input wires. More precisely, in the example shown in Fig. 8, the bit flips between the unitaries NN creates the input training set $\{(a_1, a_2, a^*)\} = \{(0,0,0), (1,0,0), (0,1,1), (1,1,1)\}$.

In total, this gives four rounds of phase accumulation. Then, the block consisting of the sequece H-$\oplus$-controlled Z-$\oplus$-H implements the diffusion matrix D from Eq. (3) on the six weight qubits, amplifying the amplitudes of the good weight strings.

All the steps above (from step 1 onwards) are denoted by the gate U. Next, the amplitude amplification subroutine repeats U for $\frac{\pi}{4}\sqrt{N} \approx 7$ times. Finally, measurements on the weight states yield with high probability the optimal choices of weights for the network, by picking the weights which turn out to be best for the largest amount of inputs.