Quantum speed-up in global optimization of binary neural nets

Yidong Liao, Daniel Ebler, Feiyang Liu and Oscar Dahlsten

1 Institute for Quantum Science and Engineering, Department of Physics, Southern University of Science and Technology (SUSTech), Shenzhen, People’s Republic of China
2 Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong Special Administrative Region of China
3 Wolfson College, University of Oxford, St Aldate’s, Oxford OX1 1HD, United Kingdom
4 Center for Quantum Computing, Peng Cheng Laboratory, Shenzhen, 518000, People’s Republic of China
5 London Institute for Mathematical Sciences, 35a South Street Mayfair, London W1K 2XE, United Kingdom

* Author to whom any correspondence should be addressed. E-mail: dahlsten@sustech.edu.cn

Keywords: quantum computation, quantum neural nets, binary neural nets, unitary training

Abstract
The performance of a neural network (NN) for a given task is largely determined by the initial calibration of the network parameters. Yet, it has been shown that the calibration, also referred to as training, is generally NP-complete. This includes networks with binary weights, an important class of networks due to their practical hardware implementations. We therefore suggest an alternative approach to training binary NNs. It utilizes a quantum superposition of weight configurations. We show that the quantum training guarantees with high probability convergence towards the globally optimal set of network parameters. This resolves two prominent issues of classical training: (1) the vanishing gradient problem and (2) common convergence to sub-optimal network parameters. We prove that a solution is found after approximately $4n^2 \log(\frac{n}{\delta}) \sqrt{\tilde{N}}$ calls to a comparing oracle, where $\delta$ represents a precision, $n$ is the number of training inputs and $\tilde{N}$ is the number of weight configurations. We give the explicit algorithm and implement it in numerical simulations.

1. Introduction

Artificial neurons are elementary many-to-one logical gates whose action is controlled by tunable parameters, often referred to as weights. Networks of interconnected neurons, called (artificial) neural networks (NNs) [1–3], have proven immensely successful for a large variety of tasks, notably including pattern recognition [4, 5], language processing [6, 7], and simulation of molecular dynamics [8, 9], leading to changes in practice in fields such as medicine, pharmaceutical research [10–13], and finance [14]. As digital logic naturally employs binary encoding, binary neural networks (BNNs) [15], for which the weights and outputs can only assume bit values, were recently introduced. Such simplifications drastically reduce the memory size and run time during the execution of the network [16].

A key reason for the success of NNs in the tasks mentioned above is their trainability. While computers were originally built to process information according to a pre-defined algorithm, NNs, including BNNs [15] can undergo training to learn how to process data themselves. In a standard set-up, data, for which the desired output is already known, is input and the network parameters are adjusted until the outputs of the NN coincide with the desired ones. This adjustment is performed according to training algorithms [1–3].

In the literature, such a calibration of the learning architecture falls under the umbrella of supervised learning.

Designing and understanding training algorithms for NNs is accordingly a critical area of research. Training NNs optimally was shown to be NP-complete [17, 18], leading to long training times [17] and large consumption of memory [19, 20]. Furthermore, common training methods, such as gradient descent, heavily rely on the shape of the optimization landscape of the network parameters. Since the landscape is
usually not convex, ‘greedy’ training methods performing step-by-step local optimisation find suboptimal choices of parameters, corresponding to local extrema rather than global ones [21]. It is known that there are NN scenarios where local extrema lead to performances which are far from globally optimal [22, 23]. In addition, the gradient of the cost function with respect to a specific weight can be vanishingly small or explodingly large—both preventing convergence to a solution [24, 25]. Hybrids of local and global searches are also considered, see e.g. [26], but it remains a key open question in NN development to devise alternative training methods that are both faster than classical global search and guaranteed to converge to a global optimum.

A potentially powerful approach to enhancing the training is the use of quantum technologies. Quantum technology has been shown theoretically to achieve performances beyond current classical implementations for a variety of tasks [27–31]. So-called quantum neural networks (QNNs), and certain closely related tunable deep unitary circuits, strive not only for an improved learning capacity [32], but also for learning [33, 34] and identifying new quantum protocols [33, 35–39]. Proposals of QNNs are commonly accompanied by a fully classical [35, 39–41] or semi-classical training algorithms assisted with quantum effects [37]. Recent proposals for quantum training algorithms showing speed-ups over classical trainings [42] take advantage of a quantum speed-up for gradient descent [43, 44]. Such results taken together illuminate that quantum NNs can be designed which outperform classical NNs for certain tasks, including training. Nevertheless the performance issues discussed above concerning gradient-descent remain, including the vanishing gradient/barren plateau problem [45]. An alternative approach to finding a quantum advantage for NN training is to consider a superposition of NNs whose amplitudes get modified according to their performance on the training data. This was suggested e.g. in [46]. However it has remained an open question how to transform supervised learning into a such a quantum training and what the performance would be.

In order to address this question, we take inspiration from recent developments in the field of coherent control of quantum processes [47–49]. More concretely, it has been shown that quantum superpositions of processes yield advantages for a variety of tasks, such as error filtration [50] and quantum communication [51]. Casting NNs in the language of processes, we seek for analogous improvements in training neural nets by initializing multiple NNs in superposition. The quantum state of the network parameters then determines which of the NNs is applied to the input data. This way, multiple networks can be probed simultaneously on the inputs if the quantum state of the network parameters is in a superposition.

The main result of this work is the concrete design of a purely quantum optimisation algorithm for BNNs. Rather than initializing the NN in a fixed weight configuration and updating the network parameters sequentially, we initialize all possible networks in a quantum superposition. The protocol maintains coherence throughout the training, in stark contrast to gradient-based training methods which generally exert measurement-based update rules for the parameters. We further provide analytical statements for the convergence time of our algorithm, as a function of the number of training inputs and number of network parameters. We prove a significant speed-up, provided that the number of training inputs scales as a polynomial of the number of weights. We do not account for experimental imperfections, such as noise and decoherence, in this work. By reformulating the training as a global quantum optimisation problem, we (1) are guaranteed (with high probability) to find the globally optimal set of network parameters and (2) achieve a provable polynomial speed-up in the number of possible choices of parameters. This resolves the following open problems of NN training. Firstly, (1) prevents convergence to parameters corresponding to local (possibly strongly suboptimal) minima of the optimisation landscape. Secondly, (2) guarantees convergence to the globally optimal solution within a known time.

The main text of this work presents the key parts of the algorithm. Further, the performance of the training is analysed. More concretely, we give analytical expressions for the convergence time of the training algorithm, as well as the number of qubit systems needed for the training. The functioning of the algorithm is illustrated with simple instances of NNs. Certain more technical components of the algorithm, as well as additional simulations of the quantum training on several concrete examples are described in the appendix.

2. The training algorithm

We now present the key parts of the algorithm and its performance, beginning with the set-up.

*Feed-forward neural nets.* Artificial NNs constitute a computational framework used to process data without pre-defining an algorithm. The task-specific algorithm is constructed from the input data alone. On a fundamental level, NNs consist of a collection of elementary computational units, called neurons, which process input data \( x \) in a fixed way—often denoted by a classical function \( f(x) \). The neurons are
interlinked in a specific way, constituting the architecture of the network. This allows neurons to exchange information: after a neuron operated on some input it can send the resulting output to the next neuron for further processing. The strength of such an interaction is described by the network parameters \( \{w_i\}_{i=1}^{N} \), commonly referred to as the weights, which are most generally real numbers.

One prominent architecture is feed-forward neural networks (FNNs). Here, the neurons are grouped into consecutive layers. Neurons within the same layer do not interact, while all neurons between two consecutive layers are connected with another. The amount of neurons can be different in each layer. The time flows linearly in a sense that each neuron in layer \( i \) receives inputs at one single time \( t_i \)—no back-action or cycles are allowed—and the outputs of the neurons are forwarded to the next layer \( i+1 \) only. This way, the user inputs data to the input layer, and after a cascade of intermediate layers (commonly referred to as hidden layers) the output layer returns the response of the FNN. Figure 1 depicts a simple instance of a FNN with two inputs and outputs and one hidden layer. The FNN architecture has proven immensely successful for tasks like pattern recognition [4, 5] and classification [52].

The task of training binary neural nets. The ability of NNs, including FNNs, to learn certain tasks like pattern recognition [4, 5] and classification [52] comes with a high computational cost. One of the reasons is that this for the inputs to a neuron, and its weights \( u \) implicitly depend on the choice of weights. Hence, the task is to find the configuration of weights which minimizes the cost function \( C \). Typically, this is done via methods based on gradient descent. Here, a weight \( w \) is updated according to its induced change in the cost

\[
C(w + \delta w) - C(w) \\
\approx \frac{\partial C(w)}{\partial w} \delta w \\
\Rightarrow w \rightarrow w - \eta \frac{\partial C}{\partial w}.
\]

The training of classical NNs is known to have three main drawbacks: (1) there is no efficient method, and indeed training has been shown to be NP-complete even for small networks [17, 18] (see appendix D for more explanations). (2) The cost landscape is usually non-convex. The solutions found in training methods based on gradient descent are only locally optimal and can be globally very suboptimal [22, 23]. (3) The gradient of the cost function with respect to a specific weight can be vanishingly small or explodingly large, preventing convergence to a solution. This issue is often referred to as the problem of vanishing or exploding gradients [53].

Quantum neural networks. The processing of neurons in classical NNs is commonly highly non-linear. In computational models, however, quantum processing of information is fundamentally reversible. Modelling the action of a single neuron by a channel \( \mathcal{N} \) this would mean that \( \mathcal{N} \) is required to be unitary. Any channel can, with the addition of reference systems, be seen as a unitary dynamics on a larger system [54]. For this reason, we treat quantum extensions of classical NNs in this work as a collection of interlinked unitary channels corresponding to the quantumly extended neurons. In the section 3 (see also appendix E for

![Figure 1. Simple instance of an FNN. The inputs \( a_1, a_2 \) are forwarded to the neurons \( N_1 \) and \( N_2 \), which process them and forward their outputs to the next layer, consisting of neurons \( N_3, N_4 \) and \( N_5 \). Afterwards, the outputs of the second layer is forwarded to \( N_6 \) and \( N_7 \), which output \( a'_1, a'_2 \). Each of the three layers is connected with edges carrying weights \( w \in \{w_1\}_{i=1}^{12} \).](image)
further concrete examples) we illustrate this point by constructing an explicit quantum extension of the most common instance of classical feed-forward BNNs (FBNNs).

Since every neuron is unitary, the full network is unitary. This allows us to treat the training framework in large generality: the only assumptions we make is that the network subject to training (1) acts unitarily, has (2) a finite number \( N \) of parameters and (3) the parameters assume values from a finite set of cardinality \( d \) (as we consider BNNs each parameter takes a bit value—the framework considered here is, however, more general and allows for any arbitrary finite number \( d \) of values). In other words, although we focus on training classical FBNNs, the results of this paper hold in general for the more abstract task of discriminating the optimal unitary evolution, out of a set of \( d^N \) candidates, for executing a certain task. In the remainder of this paper we denote by \( U \) the full action of the quantumly extended FBNN (QFBNN).

The quantum training algorithm. Here, we propose a fully quantum training protocol for QFBNNs. In contrast to previously suggested protocols [35–37, 43] we maintain coherence throughout the training and do not collapse the state with each update of parameters. For illustration purposes we restrict the training to a single neuron. This readily generalises to networks, as both a single quantum-extended binary neuron (QBN) and a full QFBNN are, on an abstract level, described by a unitary with a finite set of two-level parameters. We comment in appendix E further on how the direct translation to QFBNNs works.

The general training strategy proceeds in the following three steps

(a) Marking of the target weights: the initialization of the full set of parameters is denoted as a string of values (weights). Each possible weight string is multiplied by a phase factor which depends on how many times it leads to the correct output, for the input data in the training set. This is achieved coherently by initializing a quantum superposition of different QFBNNs and acting on it unitarily.

(b) Binarisation of the marking: the phase factors from step 1 are binarised to the values \( \{1, -1\} \) using an algorithm based on quantum phase estimation. Strings with negative sign are candidates for solving the training problem.

(c) Extracting the optimal set of weights: the binarisation step can in general mark \( M \geq 1 \) different weight states, some of which are good but not optimal. In the last step, a binary search reduces \( M \) to the minimum by sorting out the suboptimal marked states. The remaining \( M \) states all perform equally well. Afterwards, the corresponding amplitudes are amplified and measured.

We prove that this yields with high probability the globally optimal set of weights, quadratically—in the number of weight strings—faster than classical global search. While the main text discusses the subroutines in detail, appendix G illustrates the functioning of the training method with concrete examples.

Marking the target weights. Let us assume we have in total \( N \) binary parameters (weights) \( \{w_1, w_2, \ldots, w_N\} \), with values encoded as \( |0\rangle \) and \( |1\rangle \). Consequently, there are in total \( N = 2^N \) different combinations of values for the weights. Each of these combinations will be denoted by a bit string \( w \) of length \( N \), for which the corresponding quantum state reads \( |w\rangle \). Furthermore, let \( n \) be the number of data pairs \((a_1, a'_1), (a_2, a'_2), \ldots, (a_n, a'_n)\) used for the training. Here, the value \( a'_i \) (sometimes called label) denotes the corresponding output of the given input \( a_i \) (note that a single QBN produces exactly one output \( a' \) for a given set of inputs). Then, a QBN takes as inputs one set of training data \( \{a_i\}_{i=1}^n \), the corresponding weights \( |w\rangle \) of the incoming data, and an ancilla qubit \( |0\rangle \) to realize the unitary embedding of the classical binary neuron (CBN) (see appendix E for more information).

The first subroutine works as a phase accumulation approach to identify the goodness of a string, see figure 2.

Step 1: initialize all \( N = 2^N \) possible weight strings in a coherent superposition \( |W\rangle = \frac{1}{\sqrt{N}} \sum_i |w_i\rangle \). This yields the overall initial state

\[
|\text{In}_1\rangle = |W\rangle |a_1\rangle |0\rangle |a^*_1\rangle,
\]

Figure 2. Marking of the target weights. A QBN acts as a unitary \( U \) on the input \( |a_1\rangle, \) ancilla \( |0\rangle \) and coherent weight state \( \sum_i |w_i\rangle \). The output of \( U \) is compared to the desired output \( |a^*_1\rangle \) by the oracle \( \Lambda \). To decouple the weights, an uncomputation \( U^{-1} \) is applied.
where $|0\rangle$ is an ancillary qubit and $(\bar{a}_i, a^*) \in \{(a_1, a^*_1), (a_2, a^*_2), \ldots, (a_n, a^*_n)\}$ is one pair of training data. Due to the superposition, all weight states are simultaneously combined with the input $|a\rangle$. Next, the unitary action $U$ of the QBN acts on $|W\rangle$, the input $|a\rangle$ and an ancilla $|0\rangle$, encoding the output $|a^*\rangle$ on the ancillary system. The overall state is then given by

$$|	ext{Out}_1\rangle = U|\text{In}_1\rangle = \frac{1}{\sqrt{N}} \sum_i |w_i, \tilde{a}_i, a^*_i\rangle |a^*\rangle,$$

where the QBN transforms the input $|a\rangle$ and ancilla $|0\rangle$ into $|\tilde{a}_i\rangle$ and $|a^*_i\rangle$ respectively if the control state is $|w_i\rangle$. Note that except for desired out $|a^*\rangle$ the systems in equation (2) are entangled in general.

**Step 2:** call the oracle $\Lambda$ to compare the output $|a^*_i\rangle$ with the desired output $|a^*\rangle$. If the two states coincide $\Lambda$ adds a phase $e^{i\pi/n}$ to $|a^*_i\rangle$. This leads to

$$|	ext{Out}_2\rangle = \frac{1}{\sqrt{N}} \sum_i |w_i, \tilde{a}_i, a^*_i\rangle e^{i\Delta_w} |a^*\rangle,$$

where $\Delta_w = \pi/n$ if the oracle comparison between $a^*_i$ and $a^*$ was successful, and $\Delta_w = 0$ otherwise.

**Step 3:** decouple the weights by uncomputation. By inverting the unitary action $\tilde{U}$ of the QBN, the weights get decoupled from the remaining system and assume the state

$$|W^\prime\rangle = \frac{1}{\sqrt{N}} \sum_i e^{i\Delta_w} |w_i\rangle.$$

The full output state is given by $|	ext{Out}_3\rangle = |W^\prime\rangle |a\rangle |0\rangle |a^*\rangle$.

**Step 4:** accumulation of phases. The state $|W^\prime\rangle$ is used as the initial weight state for the marking with a new training pair. By repeating steps 1–3 for all $n$ pairs of training data, we achieve a coherent phase accumulation for all weight strings, resulting in the output weight state

$$|\tilde{W}\rangle = \frac{1}{\sqrt{N}} \sum_i e^{iN_i/n} |w_i\rangle.$$

Here, $N_i \in [0, n]$ denotes the number of times the oracle comparison $\Lambda$ was successful for weight string $w_i$ during the $n$ rounds. The most frequently marked weights are closest to a phase of $-1$, whereas bad weight strings maintain a phase close to $+1$. In the appendix G we discuss the quantum circuit implementations of several basic examples of the above marking scheme and their simulation results. The simulations have been done via Huawei’s quantum computing cloud platform HiQ (hiq.huaweicloud.com), as numerical evidence of the proposed quantum training algorithm.

Now, let $\tilde{U}$ be the unitary describing the full accumulation process, namely

$$\tilde{U} \left( |W\rangle \otimes \left( \bigotimes_{j=1}^{n} |a_j\rangle \otimes |a^*_j\rangle \right) \otimes |0\rangle \right) = |\tilde{W}\rangle \otimes \left( \bigotimes_{j=1}^{n} |a_j\rangle \otimes |a^*_j\rangle \right) \otimes |0\rangle,$$

where $|0\rangle^\otimes n$ is the ancillas’ state. It can be seen easily that the states $|w_j\rangle \otimes \left( \bigotimes_{j=1}^{n} |a_j\rangle \otimes |a^*_j\rangle \right) \otimes |0\rangle$, $i = 1, 2, \ldots, n$ are eigenvectors of $\tilde{U}$ with corresponding eigenvalues $e^{i\pi N_i/n}$. This is an important insight for the next step in which the phases will be converted to a binary classifier for the quality of the strings.

**Binarising the marking.** The output state in equation (3) features a coherent encoding of the performance of each possible weight string for the given training data set. Next, we need to find a way to extract this information without destroying the coherence of the state. Standard quantum searches fail in this situation: it was shown in [55] that for arbitrary phases, the optimal number $k'$ of iterations of the quantum search is in general unknown.

Instead, we propose a subroutine to binarise the phases in the weight state $|\tilde{W}\rangle$. To this purpose, we define a threshold count $N_T$, which denotes the minimum required number of successful oracle calls for a weight string to get marked. If $N_i \geq N_T$, the corresponding weight state will get a phase of $-1$. If $N_i < N_T$, the phase is set to $1$. This conversion can be done via phase estimation (PE) followed by a threshold oracle $O_{\pm 1}$. The complete process of converting the phases in equation (5) to a binary marking is visualised in figure 3.

The PE is executed on the unitary $\tilde{U}$ corresponding to the full phase accumulation routine (see equation (6)), writing PE($\tilde{U}$). The eigenvectors of $\tilde{U}$ are given by $|u_i\rangle = |w_i\rangle \otimes \bigotimes_{j=1}^{n} (|a_j\rangle \otimes |a^*_j\rangle) \otimes |0\rangle$, with corresponding eigenvalues $e^{2\pi i u_i\phi(j)}$. Note that $\phi(j)$ can be represented in binary as $0, \phi(j)^0, \phi(j)^1, \phi(j)^2, \ldots$. Omitting normalisation for readability, we input the superposition $|u\rangle = \sum_{i=1}^{N}|u_i\rangle$ of
the standard diffusion operator
marking state of standard Grover search. Hence, in order to amplify the amplitudes of the marked elements
appendix C.

Then, for \( t \) large enough it can be determined from the estimate \( |\phi_1^{(i)} \phi_2^{(i)} \ldots \phi_l^{(i)}| \) if the condition \( N_i \geq N_I \) is met.

For illustration purposes, let us analyse some simple examples. Assume first we have \( t = 1 \), such that
PE(\( \tilde{U} \)) gives the first decimal bit \( \phi_1 \). Then, using \( \phi = N_i/(2n) \), we have \( \phi_1 = 1 \) if \( N_i = n \), and \( \phi_1 = 0 \) if
\( N_i < n \). If \( t = 2 \) PE gives \( \phi_1 \phi_2 \), which is \( \phi_1 \phi_2 = 01 \) if \( N_i \geq n/2 \) and \( \phi_1 \phi_2 = 00 \) if \( N_i < n/2 \). For \( t = 3 \), PE(\( \tilde{U} \)) yields \( \phi_1 \phi_2 \phi_3 \), which assumes the value 011 if \( N_i \geq 3n/4 \), and 010 if \( n/2 \leq N_i < 3n/4 \). Thus, even though the number of uses of \( \tilde{U} \) grows exponentially in \( t \), the search interval (up to errors in the PE discussed later) is narrowed down exponentially fast in \( t \). Hence, in order to single out the largest \( \phi_i \) only a small number \( t \) of ancilla qubits is necessary. Note that \( t \) only depends on our required precision, not on \( n \).

We comment later in greater detail how to choose \( t \) and its impact on the performance.

Finally, after PE and the oracle \( O_{2,1} \), the weight state together with the phase register state reads

\[
|\tilde{W}\rangle = \frac{1}{\sqrt{N}} \sum_i (-1)^{N_i \geq N_I} |\vec{w}_i\rangle |\phi_1^{(i)} \phi_2^{(i)} \ldots \phi_l^{(i)}|,
\]

(7)

where the exponent \( N_i \geq N_I \) is to be understood as a Boolean variable which is 1 in case the condition is true and 0 else.

The state in equation (7) is entangled between the weight strings and the binary encoding of phases from the PE. In order to decouple the two systems while maintaining coherence of the weight state, we uncompute the PE, which transforms (7) to

\[
\text{PE}^{-1}(\tilde{U}) : |\tilde{W}\rangle \mapsto |\hat{W}\rangle = \frac{1}{\sqrt{N}} \sum_i (-1)^{N_i \geq N_I} |\vec{w}_i\rangle.
\]

(8)

In the next step, the amplitudes of the marked strings in the state \( |\hat{W}\rangle \) are amplified such that the concluding measurement detects one of the marked strings with high probability.

Full training cycle. It can be seen from equation (8) that the marked state \( |\hat{W}\rangle \) is equivalent to the marked state of standard Grover search. Hence, in order to amplify the amplitudes of the marked elements the standard diffusion operator \( D = H^\otimes n \Lambda_0 H^\otimes n \) can be used, with \( \Lambda_0 = 2 (|0\rangle \langle 0|)^\otimes n - I \) (see the appendix for a brief revision of Grover search). The quantum training for the binary neuron then succeeds by applying \( k \approx \sqrt{N/M} \) iterations of the binary marking via PE and the subsequent diffusion, assuming the number \( M \) of target weight strings is known. This is generally not the case: \( M \) depends greatly on the training data. Yet, even without knowledge of \( M \) we show in the appendix that one can find an optimal weight string in an expected time that scales as \( O \left( \sqrt{N/M} \right) \) with a subroutine based on Boyer et al [56]. Figure 4 summarizes the full training cycle with PE. A circuit implementation of the full training is given in appendix C.

Finally, we would like to mention an alternative to the training via PE, based on counter registers.

Instead of encoding the quality of a set of parameters into phases, it is possible to encode the absolute value
of matches \( N_1 \) for a weight configuration \( w \) into qubit states. We elaborate in the appendix F on this method and compare it with the training algorithm from the main text.

3. Circuit implementation

We illustrate the training algorithm with an elementary instance of a NN, consisting of a single neuron with two inputs. In addition to studying the dynamics of the training with a small data set, we present the explicit circuit implementations of the phase accumulation sub-routine, as well as the full quantum training. Further details and more involved examples are given in the appendix.

Consider the example of a single QBN with two inputs and two weights. Figure 5 illustrates the example, and a circuit implementation of the phase accumulation subroutine is given in figure 6. In the circuit, the six input qubits are (from top to bottom): the two weights \( |w_1\rangle \) and \( |w_2\rangle \), the two training inputs \( |a_1\rangle \) and \( |a_2\rangle \), the ancilla storing the output of the neuron computation, and the desired output \( |a'\rangle \). The initial training data point is \((a_1, a_2, a') = (0, 0, 0)\). In the first step, the Hadamard transformations create the coherent superposition of all possible weight strings, \( |W\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \) (the individual weight strings are simply \( |w\rangle = |w_1 w_2\rangle \)). Next, the circuit in the first dashed blue box depicts the unitary actions of the neurons: the CNOT gates implements the multiplication between weights and inputs, the following Toffoli gate implements the addition (bitcount operation) of the weighted inputs and the activation function by sign function. The output is then saved in the ancillary qubit on wire 5. After the oracle \( \Lambda \) is called, the uncomputation of the neuron is performed (the circuit in the second blue box). A single phase accumulation cycle is then collectively denoted by the gate \( N \).

The full phase accumulation subroutine succeeds by repeating \( N \) for \( n = 4 \) times, where \( n \) is the number of different training inputs. New input data is initialized in the circuit by applying \( X \) gates on \( |a_1\rangle, |a_2\rangle, |a'\rangle \). In this example, the training set we adopt is: \( \{a_1, a_2, a'\} = \{(0, 0, 0), (1, 0, 0), (0, 1, 1), (1, 1, 1)\} \). Due to the small numbers of inputs, we exhaust all four possible data points as the training data.

As an illustrative example of the full quantum training algorithm consider figure 7.

Referring to figure 7 we first identify the circuit’s input qubits (all initialized in state \( |0\rangle \)), from top to bottom. The very first qubit \( |0\rangle_1 \) plays the role of a control for the binary marking oracle \( O_{\pm 1} \) and is only needed later in the training algorithm. As described in the main text, the oracle executes a binary classification of the weight strings. This is achieved by comparing the performance of the weight strings with a threshold \( N_t \). If the performance is above \( N_t \) then \( O_{\pm 1} \) flips the sign of the quantum state encoding the string.

The next three input qubits \( |0\rangle_2, |0\rangle_3, |0\rangle_4 \) are the ancillary control systems for the PE. This means that in the example here the subroutine estimates \( t = 3 \) binary digits of the number \( \phi \) in the phase \( e^{2\pi i \phi} \) (up to some error described in the appendix when introducing the PE algorithm). The remaining qubits play the following role: the multi-qubit register \( |0\rangle_5 \) represents the weight register, which is transformed into the coherent superposition \( |W\rangle = \frac{1}{\sqrt{N}} \sum |w\rangle \) of all weight strings. The last qubits \( |0\rangle_6 \) are the remaining inputs (\( n \) pairs of training data and ancillas for saving the output of the NN). The training inputs and desired outputs are all initialized in zero. As we will see later based on concrete examples, the unitary action \( \hat{U} \) can be designed to include changing the training input qubits into the intended training pairs.

The first subroutine in the blue box executed the PE. The PE algorithm first initializes the \( t = 3 \) control qubits in the \(|+\rangle\) state. Each qubit controls \( 2^{t-1}, j = 0, 1, 2 \) uses of the full marking unitary \( \hat{U} \) acting on the full set of inputs to \( \hat{U} \) (see the main text for more details). The circuit implementation of \( \hat{U} \) is discussed in detail in appendix G. Then, the inverse quantum Fourier transform (QFT) converts the control ancillas into the state \(|\phi_1 \phi_2 \phi_3\rangle\), which represents a truncated binary encoding of the factor \( N_t/(2n) \).
**Figure 5.** Single neuron with two inputs and two weights. In binary neural nets, the inputs, weights, and output are all binary.

**Figure 6.** Quantum circuit implementation of the phase accumulation sub-routine for the single neuron with two inputs and two weights. The first blue box implements the neuron. Then the Λ-oracle is applied to add a phase if the output is correct. In the second blue box, the neuron is uncomputed. Next, the inputs are altered by a Pauli $X$, followed by a repeat of neuron-Λ-uncompute neuron, labelled as $N$, and so on.

**Figure 7.** Circuit implementation for the full quantum training algorithm. After initialising the threshold oracle’s workspace qubit (qubit 1) and the weight states (qubits 5), the threshold-oracle is fed the output of a PE circuit, which includes the neural net within the controlled unitary. The PE is then uncomputed. The Grover diffusion operator is applied. Those steps after the initialisation, called $Q$, here, are then repeated a number of times.

Next, the red box implements the oracle $O_{±1}$. In case the integer $N_τ$ supersedes the threshold $N_τ$, the state is changed as $|φ_1 φ_2 φ_3⟩ → −|φ_1 φ_2 φ_3⟩$. In this specific example depicted in figure 7, the threshold is set to be $N_τ = n$ (namely the optimal weight should be good for all the inputs). When converting to binary digits, this condition translates to $φ_1 = 1, φ_2 = 0, φ_3 = 0$. Therefore, we apply a multi-controlled gate to realise the sign flip: only when $|φ_1 = 1, φ_2 = 0, φ_3 = 0⟩$ the controlled-NOT gate acts on the ancilla. For this reason, the ancilla has been pre-set to $|−⟩$ by applying $X$ and a Hadamard gate on it. In this case, $|−⟩ → −|−⟩$ if the threshold is met. Different thresholds can be implemented by switching between controls and anti-controls in figure 7 (whilst a controlled unitary $U$ has form $|0⟩⟨0| ⊗ I + |1⟩⟨1| ⊗ U$, an anti-controlled $U$, drawn with a white dot, has form $|1⟩⟨1| ⊗ I + |0⟩⟨0| ⊗ U$). The resulting 8 possible $O_{±1}$ can be seen to implement the desired threshold by binary expansion of the corresponding $|φ_1 φ_2 φ_3⟩$.

Example: $N_τ = (3/4)n$ is associated with ‘threshold’ state $|011⟩$, the binary encoding of $(3/4)n/(2n) = 1/8$, such that the top two dots are black and the lowest white. The full routine is thereafter to be uncomputed, with the purpose of decoupling the marked weight state $|W⟩ = \frac{1}{\sqrt{N}} \sum (−1)^N φ_1 φ_2 φ_3 |w⟩$ from the other systems. The uncomputation of PE is depicted by the second blue box.

After the decoupling of the weight state, the marked superposition $|W⟩$ undergoes amplitude amplification: the diffusion operator $D = −HΛ_0 H$, $Λ_0 = 2(|0⟩⟨0|) ⊗ n − I$ amplifies the amplitude of the good weight strings and dampens the bad ones. The full cycle—marking and amplifying—is then collated into one box, called $Q$, which is repeated for $k^*$ times, followed by a measurement in the computational
basis on the weight states. This yields an optimal weight string, up to some errors induced by the precision \( t \) of the PE algorithm and \( \delta \) of the binary search (see the appendix for more details).

4. Quantum training performance

While speed-ups over classical training algorithms were achieved in gradient based algorithms utilizing quantum effects \([35–37, 43]\), convergence related issues remained unsolved. Classically, failure of convergence as well as convergence to locally optimal parameters rather than globally optimal ones can only be addressed by global search techniques. However, this will only fuel the issue of efficiency rather than shortening run-times.

We show in the following that the quantum training algorithm presented in this work addresses all of the drawbacks of classical training methods. As we will see shortly, the algorithm trains classical FNNs while experiencing a provable speedup. Moreover, it overcomes prominent issues of gradient-based training methods, both classical and quantum.

As a performance measure, we firstly describe the time, and secondly the number of qubits required for the quantum training algorithm.

One way to quantify the time requirement for the training is to count (1) the number of calls to the NN and (2) the number of calls to the oracles involved in the training algorithm. Counting the calls to as a way of quantifying performance is reasoned since 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.

There are two ‘elementary’ oracles. The oracle \( \Lambda \) compares two quantum states \( |\psi\rangle, |\phi\rangle \) in a binary way—they are either found to be equal or different from another. In case of equality \( \Lambda \) ‘tags’ one of the states by a label (by adding a fixed phase). The second oracle \( O_{\pm1} \) functions as a threshold check: given a number \( q \) encoded into a quantum state \( |q\rangle \), and a threshold \( \alpha \) encoded into \( |\alpha\rangle \), the oracle \( O_{\pm1} \) acts as \( |q\rangle \rightarrow (-1)^{\alpha|q\rangle} \). In other words, if the number \( q \) is found to be larger than the threshold \( \alpha \), then the oracle flips the sign of the quantum state encoding \( q \).

In the quantum training presented above there are two separate contributions to the number of calls: (i) for each training input \((a, a')\) there is one call to the accumulation subroutine (consisting of one call to the comparing oracle and two NN calls: one call and one inverse call for the uncomputation), and there are \( n \) accumulation cycles in total (recall \( n \) is the number of training inputs), (ii) the quantum search protocol on the weight states is expected to take a number of calls scaling as the \( O\left(\sqrt{\tilde{N}/M}\right) \), with \( M \) being the estimated number of solutions and \( \tilde{N} = 2^N \) being the total number of weight states.

Hence, in terms of the number of training data pairs, \( n \), and the total number of weight states \( \tilde{N} \), we call the combined Grover oracle

\[
\sum_{i=1}^{\log n/\delta} \sqrt{\frac{\tilde{N}}{M_i}} \leq \left\lfloor \log \left( \frac{n}{\delta}\right) \right\rfloor \sqrt{\tilde{N}} \equiv N_G,
\]

times. Here, \( M_i \) is the number of optimal weights for the \( i \)th loop in the binary search and \( M_i \geq 1 \). The sum in equation (9) comes from the binary search with precision \( \delta \) described in the appendix. The comparing oracle is then called

\[
N_C = N_G \left( 2n \left( \sum_{i=1}^{t-1} 2^i \right) \right) = N_G \left( 2n \left( 2^t - 1 \right) \right) \approx 4N_Gn^2,
\]

times (the factor of 2 comes from the uncomputation of the PE), and the QBNN is called \( 2N_C \) times. The last approximation in equation (10) follows from the fact that the estimated phases are given by \( N_i/2n \) and the smallest possible precision of \( N_i \) is \( \delta = 1 \). In order to be able to give an estimate up to \( \delta = 1 \), we require the \( i \)th binary digit in \( \phi_i^{(0)} \phi_i^{(1)} \ldots \phi_i^{(\tilde{N})} \) to be such that \( 2^{-i} = \delta/2n = 1/2n \). This gives \( 2^t - 1 \approx 2n \). This is not taking into account the small randomness in the PE algorithm itself, which can be avoided with a small number of extra ancillary qubits—see the appendix.

The number of qubits needed consists of (i) the number of qubits needed for the PE and (ii) the number of qubits needed for QBNN. For (i), the PE requires \( t = \log(2n/\delta) \) ancillary qubits plus a small extra number—see the appendix—to estimate with precision \( \delta \) of the binary search. For (ii) the number of qubits needed for the QBNN, \( Q_{\text{QBNN}} \), consists of the amount of qubits needed for the training inputs, the desired outputs, the \( N \) weight qubits, and the ancilla qubits:

\[
Q_{\text{QBNN}} = Q_{\text{Input}} + Q_{\text{Ancilla}} + Q_{\text{Weight}} + Q_{\text{Desired output}}.
\]
The following lemma is useful to calculate $Q_{\text{QFBNN}}$ in terms of the number of weights and the number of input bits:

**Lemma 1.** The number of input qubits $Q_{\text{input}}$, ancilla qubits $Q_{\text{ancilla}}$, weight qubits $Q_{\text{weight}}$ and output qubits $Q_{\text{output}}$ in the QFBNN satisfy the relation

$$Q_{\text{input}} + Q_{\text{ancilla}} = Q_{\text{weight}} + Q_{\text{output}}. \quad (12)$$

The proof is given in the appendix C. Noting that $Q_{\text{desired output}} = Q_{\text{output}}$, we then see

$$Q_{\text{QFBNN}} = 2(Q_{\text{weight}} + Q_{\text{output}}) \leq 2(Q_{\text{weight}} + Q_{\text{input}}). \quad (13)$$

Thus once the number of weights and input/output size is determined the number of qubits can be readily calculated.

**Quantum training faster than classical.** The outcome of the quantum training methods discussed above is guaranteed with high probability to yield the globally optimal set of weights. To ensure classically that the globally optimal parameters are found would require a brute-force search over a list of size $n \times N$, where $N = 2^n$ and $N$ is the number of weights in the network, $n$ is the number of data pairs in the training set. This list consist of all weight strings and input pairs with assignment of a cost value. As each call of that list requires comparing desired outputs with outputs, such a brute force search calls the comparing oracle $N^{cl} = n \times N$ times. On the other hand, from equations (9) and (10), our quantum training calls the comparing oracle $N^{qm} \approx 4n^2 \log \left(\frac{\delta}{\epsilon}\right) \sqrt{N}$ times, where $\delta$ represents the precision of the threshold in the binary search for globally optimal weights (see the binary search in the appendix). The ratio of $N^{cl}$ and $N^{qm}$ is:

$$\frac{N^{cl}}{N^{qm}} \approx \frac{\sqrt{N}}{4n \log \left(\frac{\delta}{\epsilon}\right)} = \frac{2^{N/2}}{4n \log \left(\frac{\delta}{\epsilon}\right)}. \quad (14)$$

Note that, for training to be practical, $n$ should scale reasonably with the number of weights, and is thus expected to be bounded by a polynomial of $N$, at least in cases where the training data has benign scaling. Learning theory suggests that a number of samples—the sample complexity—that is linear in the number of weights is optimal [57–59] and in some practical scenarios a rule-of-thumb of 10 training inputs per model variable (here weight) is employed [60]. Thus equation (14) can be termed an exponential advantage in $N$ for such cases. If $n$ were to increase exponentially with $N$ our approach may not have an advantage as $N^{qm} \propto n^2$ whereas $N^{cl} \propto n$. We are not aware of cases with exponential scaling of $n$ in $N$. As an example of linear or sub-linear scaling, within the architecture of a two-hidden-layer feed-forward network that are enough to learn $n$ training samples [61], we have the relation $n < N$ (see appendix H for details). Therefore for this example we have:

$$\frac{N^{cl}}{N^{qm}} > \frac{2^{N/2}}{4N \log \left(\frac{\delta}{\epsilon}\right)}. \quad (15)$$

The value of $N$ is often large, in the order of thousands to millions [62], such that the quantum algorithm can exhibit dramatic reductions in training time relative to global classical search.

**Overcoming vanishing and exploding gradients.** During the classical training, the weights are updated with respect to the change they cause in cost. Binary activation functions are usually replaced by continuous approximations such as the sigmoid function. Yet, it can be shown that the gradient of the sigmoid function is bound to the interval $(0, 1)$. Hence, in a network with $L$ layers, approximating the derivative $\partial C/\partial w$ of the cost function $C$ with respect to a weight $w$ in layer $\ell$ by back-propagation leads to a multiplication of $\gamma = L - \ell$ terms, all smaller than one—the gradient vanishes exponentially in the number $\gamma$. Conversely, in certain cases the derivatives can be very large and the derivative explodes. Both situations obstruct convergence and cause the training algorithm to fail.

Common approaches to address this issue is using specific activation functions, such as the ReLU function, which improves the issue but does not resolve it in general [63]. Other solutions were proposed, such as classical global search methods [26, 64] and special architectures for NNs [65–67], but also these approaches were shown to suffer from significant drawbacks [24] or fail to resolve the exploding gradient problem [25].

The quantum training method proposed in this work is a global search. It, therefore, finds globally optimal solutions while at the same time avoiding gradient related issues. While classically it is still an active field of research to improve on this problem, the quantum search completely resolves it. This makes the quantum search a powerful option to train NNs with parameters from a finite set of values.
5. Summary and outlook

In this work we formulated a fully quantum algorithm to train classical feed-forward binary neural networks. The working principle behind the training algorithm was to establish a quantum superposition of all possible NNs for a given architecture—in the light of recent progress in the field of coherent control of processes [47–49]. The algorithm then manages to single out the networks with the best performance by amplifying their weightings in the superposition. This guarantees with high probability the convergence of the training to one network which fit the training data best.

The algorithm avoids two prominent issues of gradient-descent based training (both classical and quantum): (1) convergence towards a suboptimal choice of training parameters and (2) training failures due to vanishing or exploding gradients.

Further, a performance analysis of the algorithm is presented. Here, we derive analytical expressions for the resources—run-time and number of qubits—needed to execute the algorithm. It is proven that the algorithm achieves a quadratic saving in training time—compared to classical training algorithms with the same convergence guarantees.

Finally, numerical simulations for several instances of BNN training were given. The algorithm was shown to function correctly and the quantum speed-ups were observed in the simulations.

We see several experimental and theoretical possibilities for developments. Once larger and less noisy quantum computers can be built in specialised laboratories these could use this kind of training to identify optimal BNNs off-line, which are then built classically and implemented classically on-line. Whilst implementing large networks experimentally is currently not feasible, training a single binary neuron—an elementary module of the network—with the methods described above requires approximately 13 qubits. This is within range of current state-of-the-art experiments, albeit with noise [68, 69]. Error-correction would of course add further experimental complexity e.g. as in [70]. A theoretical question concerns the fact that the ability of finding the globally optimal weights may also induce a risk of over-fitting. There are several methods for avoiding over-fitting classically, including dropping out certain layers at times during the training, restricting the number of weights and weight regularisation techniques [71, 72]. It thus seems well-motivated to build on this quantum training framework to explore analogous quantum techniques. For example quantum versions of drop-out could involve dynamic quantum network architectures e.g. using quantum control systems or teleportation between non-successive layers.

Finally we note that our algorithm could be employed more generally to find globally optimal unitary dynamics, given a task and a finite set of candidate unitaries. The task itself is encoded in a set of training data and might be unknown to the experimenter. For instance, one might think of an unknown device which was tested by sending in a list of inputs and recording the corresponding outputs. The algorithm in this work then identifies which of the unitary dynamics from the candidates is the best fit for the data. From this perspective the algorithm sounds promising for tasks like hypothesis testing [73], regression [74, 75] and causal discovery [76].

Code availability

The numerical implementations were done via Huawei’s quantum computing cloud platform HiQ (hiq.huaweicloud.com) and the open-source version of our code is available on www.github.com/phylyd/QBNN.

Author contributions

All authors contributed substantially to the research presented in this paper and to the preparation of the manuscript.

Acknowledgments

We would like to thank Yu Hao, Jon Allcock, Joshua Morris, Sina Salek and Christa Zoufal for useful discussions. We acknowledge funding from the National Natural Science Foundation of China (NSFC) and a collaborative grant with HiSilicon.
We commence with a brief introduction to Grover search and quantum phase estimation.

**Grover's search algorithm.** For an unsorted list of length $N$, identifying the index of a given element in the list requires $O(N)$ computational steps classically. 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 [28], in which a speed-up is achieved by evolving a superposition of list elements. In the following, we give a brief overview of the algorithm. Let us encode the number $M$ items of a list into quantum states in the computational basis $|0\rangle, |1\rangle, \ldots, |N-1\rangle$. The quantum oracle $\Lambda_u$ , which is applied on the system $|u\rangle$, marks the state $|u\rangle$ by flipping its sign if it coincides with one of the solutions from the set $\{|\omega_i\rangle\}_{i=1}^M$, and does nothing otherwise. The diffusion operator amplifies the amplitude of the marked elements, such that the concluding measurement yields the outcome $x' \in \{|\omega_i\rangle\}_{i=1}^M$ with high probability.

The optimal number of iterations of the oracle and the diffusion before the concluding measurement is given by $k^* \approx \sqrt{\frac{N}{2^m}}$ [28]. It is, hence, crucial to have knowledge about $M$, as stopping at the wrong time may result in a random output rather than a valid solution.

**Quantum phase estimation.** Suppose $|u\rangle$ is the eigenvector of a unitary operator $K$, with eigenvalue $e^{i \phi}$. To estimate $\phi$ up to some error $\epsilon$, we initiate $t$ qubits in the state $|0\rangle$. The number $t$ is determined by the precision of the estimation of $\phi$, as well as the probability of success of the estimation. It can be shown [54] that for a PE with probability of error $\epsilon$, in order to obtain the phase $N_i/(2n)$ accurate to $m$ bits, we need a precision (and number of ancillary control systems)

$$t = m + \left(\log\left(2 + \frac{1}{2\epsilon}\right)\right).$$

The PE algorithm succeeds in two steps and is depicted in figure A1.

**Step 1.** Apply a Hadamard gate to each of the $t$ qubits in the first register, yielding the states $|+\rangle^\otimes t$. Then, the first ancilla acts as a control for $2^0 = 1$ use of the unitary operator $K$, which is applied to $|u\rangle$. The second ancilla acts as a control for $2^1 = 2$ uses of $K$, and so on. In general, the $l$-th ancilla acts as a coherent control for $2^{l-1}$ uses of $K$. In the end, the collective output state $|T\rangle$ of the $t$ ancillas yields [54]

$$|T\rangle = \frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} e^{i \pi k |l\rangle} |k\rangle.$$
We now present in detail the binary search algorithm to train a QFBNN without knowing the number of good parameter configurations. For a fixed precision \( t \) of the PE subroutine, the number of solutions \( M \) depends on the training set and is generally unknown. Indeed, there might be multiple marked weight strings, or even none which is above the threshold \( N_r \). To resolve this problem, we propose a binary search on the threshold intervals as a subroutine to ensure, up to the precision of the search and up to imprecisions in the PE algorithm, that the optimal solution is marked and amplified.

For a solution \( w^* \) to the training the possible range of the number of matches \( N^* \) in the accumulative marking scheme is initially an integer from the interval \([0, n]\). At every step of the search the possible range is cut evenly into two parts. Thus the size of the possible range goes as \( n2^{-i} \) for the \( i \)-th step of the binary search. The search stops when the size of the possible range reaches a small enough number which quantifies our desired precision, and which we call \( \delta < n \), i.e. the search runs until \( i \) is large enough so that \( n2^{-i} \leq \delta \). To determine whether the updated possible range is the lower or upper half of the cut, we mark the weight strings in the upper half by \(-1\) using the procedures described above. If after the search the measured weight string is in the upper half this becomes the new possible range. Else if it is not in the upper half the possible range becomes the lower half. Thus eventually this binary search identifies with a high probability an \( N^* \) which is within \( \delta \) of the global optimum. This takes \( \lceil \log \left( \frac{n}{\delta} \right) \rceil \) steps, which scales efficiently in the number of \( n \).

The binary search then proceeds as follows:

- Initialize \( N_r = n/2, i = 0 \).
- Set \( m = 1 \) and pick \( \mu \in (1, 4/3) \).
- For \( i \leq \lceil \log(n/\delta) \rceil \), loop the following steps:
  - Choose an integer \( s \in [1, m - 1] \) uniformly at random.
  - With current \( N_r \), apply \( s \) iterations of the full training cycle from figure 4. Measure the weight state and obtain an candidate solution \( w^* \).
  - Run the classical BNN on \( w^* \) only to obtain \( N^* \), the number of times \( w^* \) gave correct outputs.
  - If \( N^* \geq N_r \), this means the observed \( w^* \) is indeed a solution for current \( N_r \), and there might be better solutions for larger \( N_r \), so update \( N_r \) as: \( N_r \rightarrow N_r + \Delta N_i /2 \), in which \( \Delta N_i = n2^{-i} \), and go to next step to update \( i \).
  - else (i.e. \( N^* < N_r \)), set \( m \) to \( \min(\mu m, \sqrt{N}) \), and go back to step (i).
- Time-out rule for this inner loop on \( m \): when the loop of steps 1–4 continues to the case that the full training cycle has been totally executed \( O(\sqrt{N}) \) iterations, but \( N^* < N_r \) still, this means there is no solution for the current \( N_r \), then update \( N_r \) as: \( N_r \rightarrow N_r - \Delta N_i /2 \) in which \( \Delta N_i = n2^{-i} \), and go to next step to update \( i \).
  - Update \( i \) as: \( i \rightarrow i + 1 \).
- Output the last value of \( w^* \) that has \( N^* \geq N_r \) and end training. This \( w^* \) is the optimal weights the algorithm found.

The above search homes in on a solution that is globally optimal up to the \( \delta \) precision.

### Appendix C. Equality for the number of qubits needed in the QBNN

To calculate the total number of qubits needed by the algorithm which we derived in equation (13), we made use of the following lemma:

\[
Q_{\text{input}} + Q_{\text{ancilla}} = Q_{\text{weight}} + Q_{\text{output}}
\]  \hfill (C.1)

where \( Q_{\text{input}} \) was the amount of input qubits, \( Q_{\text{ancilla}} \) the amount of ancilla qubits, \( Q_{\text{weight}} \) the amount of weight qubits and \( Q_{\text{output}} \) the number of output qubits. Here, we derive this equality and illustrate it by two examples.
Let us begin with the elementary case of a single neuron, depicted in figure C1 below. It is straightforward to count and verify the equality in equation (C.1), by noting that $Q_{\text{ancilla}} = Q_{\text{output}} = 1$ and that every input comes on an edge with an assigned weight, i.e. $Q_{\text{input}} = Q_{\text{weight}}$. Thus equality (C.1) holds for the case of a single neuron.

Next, we consider a more involved network with three neurons in two layers, see figure C2. Direct counting verifies that equation (C.1) also holds here. The reason in general is that in the first layer the input qubits and weight qubits always come in pairs, i.e.

$$Q_{\text{input}} = Q_{\text{weight}}^{(\text{in})}, \quad \text{(C.2)}$$

where the superscript denotes that the count is for the weights in the first layer.

In the first layer and hidden layers, the ancilla qubits from the previous layer carry the output of the neuron computation, together with fan-out ancilla qubits, serve as inputs for the next layer, and each of them is paired with a weight qubit in the hidden layers.

Let the number of weight qubits $Q_{\text{weight}}$ be broken into two parts, those associated with the input layer and the rest, associated with the hidden layers:

$$Q_{\text{weight}} = Q_{\text{weight}}^{(\text{in})} + Q_{\text{weight}}^{(\text{hidden})}. \quad \text{(C.3)}$$

For example, in figure C2, $Q_{\text{weight}} = 6$, $Q_{\text{weight}}^{(\text{in})} = 4$ and $Q_{\text{weight}}^{(\text{hidden})} = 2$.

Similarly let the number of ancilla qubits $Q_{\text{ancilla}}$ be broken into three parts, those associated with the input layer, those associated with the hidden layers, and those associated with the output layer.

$$Q_{\text{ancilla}} = Q_{\text{ancilla}}^{(\text{in})} + Q_{\text{ancilla}}^{(\text{hidden})} + Q_{\text{ancilla}}^{(\text{out})}. \quad \text{(C.4)}$$

For example, in figure C2, $Q_{\text{ancilla}} = 3$, $Q_{\text{ancilla}}^{(\text{in})} = 2$, $Q_{\text{ancilla}}^{(\text{hidden})} = 0$ and $Q_{\text{ancilla}}^{(\text{out})} = 1$. In general some of $Q_{\text{ancilla}}^{(\text{hidden})}$ may be associated with fan-out gates, copying classical information to send to several neurons in later layers.

For the output layer each output qubit corresponds to an ancilla qubit in this layer:

$$Q_{\text{ancilla}}^{(\text{out})} = Q_{\text{output}}. \quad \text{(C.5)}$$
All ancillas except the output ancilla are paired up with weight qubits of the hidden layers, as they turn into inputs to hidden layers and each input is paired with a weight. Accordingly,

\[ Q_{\text{ancilla}}^{(\text{in})} + Q_{\text{ancilla}}^{(\text{hidden})} = Q_{\text{weight}}^{(\text{hidden})}. \]  

(C.6)

Altogether, we see that the equality

\[ Q_{\text{input}} + Q_{\text{ancilla}} = Q_{\text{weight}} + Q_{\text{output}} \]  

(C.7)

follows, for general QBNN according to our design. Equation (C.7) is the lemma we wished to prove.

Appendix D. Standard classical training of neural networks: gradient descent

A widely used approach in training NNs is to approach the minimum cost through the method of gradient descent. Here, each weight \( w \) is updated between the \( t \)-th iteration of the NN to the next one as

\[ w(t + 1) = w(t) - \eta \cdot \frac{\partial C(a^t, a^t_\ast)}{\partial w}. \]  

(D.1)

Above, \( \eta \) is a positive number, usually referred to as the learning rate.

The direct evaluation of the term \( \frac{\partial C}{\partial w} \) is computationally expensive: approximating the derivatives numerically requires to loop over all network inputs. In addition, even if the values of the network parameters and inputs is restricted (e.g. for BNNs), commonly the weights are kept real-valued and only discretised after the evaluation of the partial derivatives [15, 77]. Hence, improvements of memory consumption and execution speed are missing in the training stage of BNNs.

Moreover, gradient descent based training algorithms have the disadvantage of converging towards local minima in the cost landscape, rather than global ones. It was shown that locally optimal solutions to the training algorithm can be strongly sub-optimal [21]. To find a global minimum is in general NP-hard, consistent with the exponential—in the number of neurons—number of weight configurations [18, 78].

There are several methods to simplify the evaluations in gradient descent. The most prominent way is back-propagation. Here, the evaluation of the gradients is approximated: while the value \( C(a^t, a^t_\ast) \) depends on the full history of weights between the layers, back-propagation computes gradients only relying on the latest input of the neuron rather than the full processing history of the network. Such an evaluation gives a chain of layer-wise derivatives, which, multiplied with another, often leads to good approximations of \( C(a^t, a^t_\ast) \). Yet, back-propagation is still subject to slow training times, such that for deep networks training is usually done on supercomputers before distributing the NN to the end user. Further, back-propagation suffers from gradient-related problems such as the ones mentioned above. This can be seen easily. As the derivatives are approximated through a product of several local derivatives, the direction of convergence remains the same, leaving the issue of finding local minima open. Moreover, if exploding and vanishing gradients occur then also the evaluation of local gradients suffer from the problems.

Appendix E. Quantum embedding of classical neural networks

A classical neuron implements a function, which takes multiple inputs and produces a single output. Hence, this many-to-one mapping makes the processing irreversible, which is a priori not compatible with reversible dynamics of quantum computing. However, this mapping can be generalised to a quantum neuron in the same way that quantum computing generalises classical computing, along the paradigm of [35]. To define the quantum generalisation, first, classical gates are extended to reversible gates, which are a subset of quantum unitaries. Afterwards, general unitaries are allowed for. In this section we illustrate the quantum extension of CBNs with a concrete example.

Finding a reversible extension. We consider the most common instance of binary neuron processing, where each CBN \( j \) implements the Heaviside activation function \( H(x) \) with activation threshold \( \alpha_j \). More precisely, for a vector \( x \), for a threshold \( \alpha_j \) the activation succeeds as

\[ H(x) = \begin{cases} 1, & \text{if } \sum_i x_i \geq \alpha_j \\ 0, & \text{else.} \end{cases} \]  

(E.1)

The corresponding CBN is depicted below in figure E1.

For the instance depicted in the figure, the neuron takes the input \( a = (a_1, a_2) \) with the two components \( a_1 \) and \( a_2 \), and has weights \( w_1, w_2 \in \{1, -1\} \) on the edges. An XNOR gate multiplies the inputs with the
weights, yielding $s_i = w_i a_i$, $i = 1, 2$. Afterwards, the operation bitcount sums the results to the value $s = w_1 a_1 + w_2 a_2$. Finally, the sign 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. The result is a reversible embedding $U_{\text{XNOR}}$, $U_{\text{bitcount}}$, and $U_{\text{bit}^+}$ of bitcount, see figure E2. Here, $U_{\text{bit}^+}$ includes both the bitcount operation and the activation through the activation function $f$.

**Unitary embedding of reversible binary neurons.** The reversible logical gates in figure E2, with inputs $x_i$ and outputs $y_i$, can be implemented as quantum unitaries $U = \sum |x_i\rangle \langle y_i|$. This defines a quantum binary neuron (QBN). In order to find a quantum circuit implementation of a QBN, we decompose the operations $U_{\text{XNOR}}$ and $U_{\text{bit}^+}$ into elementary quantum gates.

The multiplication of the input and weight can be achieved by a controlled NOT (CNOT) gate on each input $|a_i\rangle$, controlled by the state of the corresponding weight $|w_i\rangle$ (see figure E3). It can be easily seen from the truth table of the XNOR and CNOT gate that the operations indeed coincide. The gate $U_{\text{bit}^+}$ is realized by the Toffoli gate [79] on the ancilla, where the states $|s_1\rangle$ and $|s_2\rangle$ act as control systems.

**Quantum binary neurons can form networks.** So far, we proposed a unitary extension of a classical neuron. A NN consists of multiple interlinked neurons. Hence, the quantumly extended NN acts, overall, unitarily on inputs as well. Generalising figure E3, a quantum feed-forward binary neural network (QBFNN) takes a set of $N$ weights $\{w_1, w_2, \ldots, w_N\}$ as inputs, together with a set of data $a = \{a_1, a_2, \ldots, a_p\}$. The set $\{w_1, w_2, \ldots, w_N\}$ denotes all weights in the networks—including input layer, hidden layers and output layer. The set of data $a = \{a_1, a_2, \ldots, a_p\}$ assumes that there are $p$ neurons in the input layer (with assigned weights $w_1, w_2, \ldots, w_p$). Each of the $r$ neurons in the last layer of the QBFNN outputs one value, leading to the overall outputs $a' = \{a'_1, a'_2, \ldots, a'_r\}$. Then, by depicting the action of the QBFNN as a single unitary $U$, the fully quantum training method introduced in the main text generalises straightforwardly from a single neuron to general QBFNNs. The only change that requires some more
explanation is the comparison oracle $\Lambda$. The direct generalisation would be to define the action as

$$\Lambda(a', a^*) = \begin{cases} e^{i\pi/n} |a'\rangle|a^*\rangle, & \text{if } a' = a^* \\ |a'\rangle|a^*\rangle, & \text{else.} \end{cases} \quad (E.2)$$

Other generalisations that only require a certain number $r \in [0, p]$ of outputs to coincide are possible as well.

When generalising from a single neuron to a FNN, the output values of the neurons are to be copied and distributed to each neuron in the next layer. While classically this is trivial to do, quantum mechanics prohibits exact copying of data—a phenomenon called no-cloning theorem [80]. For classical BNNs, this issue is resolved by applying CNOT gates to the output of each neuron, one for each copy operation. The CNOT acts on an ancillary qubit initialized in $|0\rangle$, controlled by the output qubit $|a'\rangle$ of the neuron. Hence, for bit valued $a'$ the CNOT gate acts as

$$\text{CNOT}(|a'\rangle|0\rangle) = |a'\rangle|a'\rangle. \quad (E.3)$$

In total from one layer with $\ell_1$ neurons to the next with $\ell_2$ neurons, we need $\ell_1 \times \ell_2$ ancillas. When the overall state has coherence, the CNOT gates in general do not produce perfect copies but rather entanglement between the states and the ancillas carrying the output values. We call this imperfect spread of information fan-out operation [35].

Appendix F. Alternative approach to quantum training algorithms: counter registers

Quantum training via PE includes an overhead of neuron calls that scales exponentially in the precision $t$ of the estimation. There is an alternative approach which is sensitive to the size of the training set, but avoids that overhead: rather than accumulating phases, the number of matches between the QFBNN outputs and the desired output can be encoded directly into a multi-qubit register.

The algorithm is similar to the training cycle presented in the results, up to some changes: in figure 2 the oracle $\Lambda$ adding incremental phases to good weight strings is replaced by a new oracle $\mathcal{O}_{\text{count}}$, acting on a $\lceil\log(n + 1)\rceil$-qubit register as

$$\mathcal{O}_{\text{count}}|h\rangle = \begin{cases} |h + 1\rangle & \text{if } a' = a^* \\ |h\rangle & \text{else} \end{cases}. \quad (F.1)$$

The number $h$ of matches is then encoded in binary into the qubits. Initially the counter is set to $|0\rangle$ and increased coherently during the marking. This way, after the full set of training data, the output weight state reads

$$|\tilde{W}\rangle = \frac{1}{N} \sum_i |w_i\rangle |N_i\rangle, \quad (E.2)$$

where $N_i$ is the amount of matches for the weight string $w_i$.

Next, we again define a threshold count $N_r$ and a new oracle $\tilde{\mathcal{O}}_{\text{count}}$. If the condition $N_i \geq N_r$ holds, the oracle marks the corresponding weight string $w_i$ by adding a factor of $-1$. Otherwise, the string is left invariant. The resulting state reads

$$|\tilde{\tilde{W}}\rangle = \frac{1}{\sqrt{N}} \sum_i (-1)^{N_i \geq N_r} |w_i\rangle |N_i\rangle. \quad (E.3)$$

In order to decouple the weight states from the counter register, the counter needs to be reset, in analogy with the uncomputation of the PE in the main text. To maintain coherence, such a reset needs to be unitary. This can be achieved by uncomputing the $n$ rounds of comparing the QBN outputs with the desired outputs.

The weight state then becomes

$$|\tilde{\tilde{W}}\rangle = \frac{1}{\sqrt{N}} \sum_i (-1)^{N_i \geq N_r} |w_i\rangle, \quad (E.4)$$

and the same amplification procedure as for the training algorithm with PE can be applied. The choice of $N_r$ can found by adapting the binary search presented in appendix B.

The alternative training algorithm with counter registers has the advantage of avoiding the exponential overhead in the number of neuron calls in the PE. However, there is one fundamental drawback: for the
training with PE we extracted the fraction $N_i/N$, $N_i \in [0, n]$, whereas here we would encode the absolute number of counts $N_i$. The number of training data pairs $n$ is typically large. Hence, while small numbers $t$ of ancilla qubits for the PE subroutine are sufficient, we need a great amount of qubits to encode the number of matches. Such an overhead could in practice be the bottleneck for implementations on real quantum computers.

Appendix G. Implementations of QBNN examples

We now go into detail how the phase accumulation $\tilde{U}$ is implemented in a quantum circuit. We illustrate this with several concrete examples.

G.1. Single neuron with 3 weights and 3 inputs

Next, we consider a single neuron with 3 weights and 3 inputs, as shown in figure G1. We will see below that this slight increase in inputs manifests itself in a significantly more complex gate implementation of the neuron action, implying an extra overhead in terms of numbers of gates that is proportional to the number of neurons, without impacting the number of oracle calls. Figure G2 shows the quantum circuit for the phase accumulation subroutine of this example.

Referring to figure G2, the qubits with index 1–3 take the role of the weights and qubits with index 4–6 are the three training inputs. Qubit 7 is an ancilla storing the output of the QBN (sometimes also referred to in this work as $a'$). Qubit 8 carries the desired outcome. As before, the dashed blue box depicts the action of the QBN consisting of (1) multiplying inputs with the weights, (2) adding up the weighted inputs and (3) the activation function (the gate implementation of the addition and activation is included in the dashed red box). The oracle $\Lambda$ (figure G3) is now applied:

- Ancilla is initialised in $|1\rangle$.
- For every pair of qubits $(a'_i, a^*_i)$ in the output and desired output respectively,
  - Undertake CNOT (control, target) as follows: (a) CNOT($a'_i$, ancilla), (b) CNOT($a^*_i$, ancilla).
  - if $|a'_i\rangle = |a^*_i\rangle = |0\rangle$ ancilla has stayed $|1\rangle$.
  - if $|a'_i\rangle = |a^*_i\rangle = |1\rangle$ the ancilla has been flipped twice and is also $|1\rangle$.
  - In the other two cases ancilla ends as $|0\rangle$. 

![Figure G1. Single neuron with 3 inputs and 3 weights. Again the weights, inputs and output are all binary in the classical neuron and qubits in the quantum generalisation.](image1)

![Figure G2. Quantum circuit implementation of phase accumulation sub-routine for the example of a single neuron with 3 inputs and 3 weights.](image2)
Figure G3. Circuit for oracle $\Lambda$. The first qubit is an output to be compared with the second qubit which is the desired output. An ancilla system—the third qubit—is also used. In circuit (i) controlled $X$s (CNOTs) are applied followed by a controlled phase gate and uncomputation of the CNOTs. Circuit (ii) is an alternative equivalent representation of circuit (i).

Figure G4. Training data used for the example of a single neuron with 3 weights and 3 inputs. Each training tuple consists of three inputs $a_1, a_2, a_3$ and one desired output $a^*$. The aim of the training is to find weight configurations that can generate the desired output of the corresponding inputs. We consider two independent tasks (enumerated by 1 and 2) and study the training according to the data.

Figure G5. Task 1: the evolution of the probabilities to measure the weight strings. For the first task there exists only one optimal weight configuration, namely (000). The vertical bars present the probabilities of the weight configurations, for each iteration of the full training cycle. The probability of the optimal weight is amplified and reaches its maximum after two iterations, in accordance to the optimal stopping time $k^* = \sqrt{8\pi/4} \approx 2.22$ of the quantum search.

* Apply control phase gate $|1\rangle \langle 1| \otimes \exp(i\phi)I + |0\rangle \langle 0| \otimes I$. This adds phase $\exp(i\phi)$ if ancilla is in state $|1\rangle$.

* Uncompute CNOTs, i.e. redo (a) and (b). Now the ancilla is certainly in state $|1\rangle$ again.

Next the action of the QBN is uncomputed (the circuit in the second dashed blue box). Finally, the full phase addition is repeated for $n$ times, where $n$ denotes again the amount of training input data. For readability, we only depict two data points $\{(a_1, a_2, a_3, a^*)\} = \{(0, 0, 0, 0), (1, 0, 0, 0)\}$ of the training set in the figure and used ellipsis to indicate that the loop indeed succeeds over all remaining training pairs. The two independent sets of full training data considered for this architecture are specified in figure G4.

During the loops of the full training (see figure 3 in the main text), the probabilities of the weight strings change. Here we present the probability evolution of the weight strings, in the quantum training with the highest $N_\tau$ for which there exists at least one optimal weight string: for task 1, figure G5 shows the probabilities of the weight strings evolving over the training cycles. Correspondingly, figure G6 illustrates the evolution for task 2.

G.2. Three-layer five-neuron network with 6 weights, 2 inputs and 1 output

Next, we study the circuit implementation of the phase accumulation subroutine for a network with five neurons and layer configuration 2-2-1. This is depicted in figures G7 and G8. In total, there are two inputs.
Figure G6. Task 1: the evolution of the probabilities to measure the weight strings. For the second task there are two optimal weight configurations: (000) and (101). The probabilities of the optimal weights are amplified and reach their maximum after the first iteration, in accordance to the optimal stopping time $k^* = \sqrt{8/\pi}/4 \approx 1.57$.

Figure G7. Five-neuron network with layer configuration 2-2-1. The network takes two inputs and has in total 6 weights.

Figure G8. Circuit implementation for a network with layer configuration 2-2-1, with 2 inputs and 6 weights. The qubits indexed 1–4 are the weights for the first layer and qubits with index 5 and 6 are the weights for the second layer. Qubits with index 7–10 are 4 training inputs duplicated from 2 inputs in the first layer, for the fan-out to the second layer. To save space in the diagram, the fan-outs in the first layer take place to the left, outside the diagram. Thus the inputs are already copied when they enter the depicted circuit. Qubits 11 and 12 are ancillas storing the outputs within the QBNN. Qubit 13 encodes the desired outcome. The dashed blue box depicts the action of the QBNN. After applying the oracle $\Lambda$, the action of the QBNN is uncomputed. The circuit depicts one round of marking for one data point in the training set, for the phase accumulation process.

$|a_1\rangle, |a_2\rangle$ and six weight states $|w_1\rangle, |w_2\rangle, \ldots, |w_6\rangle$, leading to $2^6 = 64$ possible weight strings $|w_1w_2\ldots w_6\rangle$. The two dashed blue boxes show the circuit for the computation and uncomputation of the NN.

For the QBNN in this example, we use the training set $\{(a_1, a_2, a^*)\} = \{(0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 1)\}$. In figure G9, we present the probability evolution of the weight strings, in the quantum training with the highest $N_T$ for which there exists at least one optimal weight string.
Figure G9. The probability evolution of the weight strings in the five-neuron QBNN with layer configuration 2-2-1, with 2 inputs and 6 weights. For the training set \( \{a_1, a_2, a^*\} = \{(0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 1)\} \) there are 7 optimal weight configurations among the total \( 2^6 = 64 \) configurations in the QBNN. In the figure, the horizontal axis presents all the 64 weight configurations. The configurations which have the same weights in the first layer are grouped together, with different colours representing the different weights (00, 01, 10, 11) in the second layer. The vertical bars presents the probabilities of the weight configurations, for each iteration during the training. The probabilities of the optimal weights are amplified over the iterations and reaches its largest amplification after two iterations. This result agrees with the expected stopping point given by \( \sqrt{N/M \pi/4} \approx 2.37 \) for this instance.

G.3. Three-layer six-neuron network with 8 weights, 3 inputs and 1 output

We conclude the set of examples with a six-neuron network in configuration 3-2-1, with 3 inputs and 8 weights (see figure G10). In total, this network uses 23 qubits. The circuit implementation of a single cycle of phase accumulation is shown in figure G11. We trained the QBNN in this example with the same two tasks presented in figure G4 above. For task 2 we present the probability evolution of the weight strings in figure G12. The threshold \( N_T \) is set such that there exists a single solution of weights to the training algorithm.

For this example there are 8 optimal weight configurations among the total of \( 2^8 = 256 \) configurations. The horizontal plane presents all the 256 weight configurations. Each point in the grid represents one weight configuration, with two axes indicating the weights in the two layers respectively. The vertical bars present the probabilities of all the weight configurations, for each iteration during the training. The probability of the optimal weights are amplified over the iterations and reaches its maximum at iteration 4, in agreement with the expected stopping point given by \( \sqrt{256/8 \pi} \approx 4.44 \).

Figure G13 shows the probability of obtaining an optimal weight string, as a function of the training cycle for both tasks. Recall the comparison between our quantum training and classical training stated in the performance analysis in the main text. Classical global searches call the comparing oracle \( N_{cl}^{\text{qd}} = n \times \tilde{N} \), whereas our quantum training algorithm only needs approximately \( N_{Cl}^{\text{qm}} = \tilde{N} \times 2n \times (2n - 1) \) calls, with
Figure G10. NN with 6 neurons in layer configuration 3-2-1.

Figure G11. Circuit implementation for a network with layer configuration 3-2-1. Qubits 1–6 are the weights for the first layer and qubits 7 and 8 are the weights for the second layer. Qubits 9–14 are the 6 training inputs, duplicated from 3 inputs in the first layer, for the fan-out to the second layer. (To save space in the diagram, the fan-outs in the first layer take place to the left, outside the diagram. Thus the inputs are already copied when they enter the depicted circuit.) Qubits with index 15 and 16 are ancillas storing the outputs of the neurons within the QBNN and qubit 17 encodes the output of the QBNN. Qubit 18 encodes the desired output. The dashed blue box contains the action of the QBNN, consisting of weighing the inputs with the weights, adding up the weighted input together with the subsequent activation function (the gate implementation of the addition and activation is included in the dashed red box). After applying the oracle $\Lambda$, the action of the QBNN is uncomputed (the circuit in the second dashed blue box). The entire circuit depicts one round of marking for one data point in the training set, for the phase accumulation process.

\[ N_G = \lceil \log(n/\delta) \rceil \sqrt{\tilde{N}} \] and $\delta$ being the precision of the binary search to single out optimal weight configurations (see the appendix).

We begin with task 1. Inserting the value of $n = 8$ and $\tilde{N} = 256$ for this example, one can see that even for this small network there is a quantum advantage: while $N_{cl}^{\text{cl}} = 8 \times 256$ we find for our quantum training algorithm $N_{qm}^{\text{cl}} = 8 \times 180$. Note that for large networks the quadratic advantage in $\tilde{N}$ will be much more significant, as discussed in the main text.

Appendix H. Correlation between $n$ and $N$ for a two hidden layer feed-forward network

In [61] the author proved that for a two-hidden-layer feed-FNN with $m$ output neurons, the number of hidden nodes that are enough to learn $N$ samples with negligibly small error is given by:

\[ 2\sqrt{(m + 2)N}. \] (H.1)
Figure G12. Probability evolution of the weight strings in the six-neuron QBNN with configuration 3-2-1, task 2. Initially all weights have the same amplitude but the quantum search technique increases the amplitudes of the best weights until the 4th iteration which is the optimal moment to measure the system. (The quantum search technique we employ does not require knowing at which moment to measure).

Specifically, the sufficient number of hidden nodes in the first layer is suggested to be

\[ L_1 = \sqrt{(m+2)N} + 2\sqrt{N/(m+2)} \]  \hspace{1cm} (H.2)

and in the second layer

\[ L_2 = m\sqrt{N/(m+2)}. \]  \hspace{1cm} (H.3)

In this architecture, the number of weights between the two hidden layers is the product of the number of nodes in the two hidden layers:

\[ L_1 \times L_2 \]  \hspace{1cm} (H.4)

\[ = (\sqrt{(m+2)N} + 2\sqrt{N/(m+2)}) \times (m\sqrt{N/(m+2)}) \]  \hspace{1cm} (H.5)

\[ = mN + 2Nm/(m+2). \]  \hspace{1cm} (H.6)
Figure G13. Probability of obtaining an optimal set of weights, for a six-neuron QBNN with layer configuration 3-2-1. The graphs show the optimal stopping time of the training, for both task 1 and 2. We see that the probability of success is close to unity after 6 iterations for task 1 and 4 iterations for task 2.

The total number of weights $N_{\text{total}}$ in this optimal architecture is larger than $L_1 \times L_2$ (since we also have weights between input layer and first hidden layer, weights between second hidden layer and output layer). Thus, we find

$$N_{\text{total}} > L_1 \times L_2 > mN.$$  \hfill (H.7)

Using the notation in our work, i.e. $N_{\text{total}} \mapsto N, N \mapsto n$ we have

$$N > m \times n,$$  \hfill (H.8)

where $N$ is the number of weight qubits and $n$ is the number of training samples, $m$ is the number of neurons in the output layer. Since $m$ is an integer larger than 1, so $N > mn > n$. This gives us the correlation $n < N$ used in the main text.

References

[1] Grossberg S E 1988 Neural Networks and Natural Intelligence (Cambridge, MA: MIT Press)
[2] Haykin S S 2009 Neural Networks and Learning Machines vol 3 (Upper Saddle River, NJ: Pearson)
[3] Russell S J and Norvig P 2016 Artificial Intelligence: A Modern Approach (Malaysia: Pearson Education)
[4] Bishop C M et al 1995 Neural Networks for Pattern Recognition (Oxford: Oxford University Press)
[5] Nasrabadi N M 2007 J. Electron. Imaging 16 049901
[6] Collobert R and Weston J 2008 A unified architecture for natural language processing: deep neural networks with multitask learning Proc. of the 25th Int. Conf. on Machine Learning (ACM) pp 160–7
[7] Mikolov T, Karafiát M, Burget L, Černocký J and Khudanpur S 2010 Recurrent neural network based language model 11th Annual Conf. of the Int. Speech Communication Association
[8] Kozuch D J, Stillinger F H and Debenedetti P G 2018 Proc. Natl Acad. Sci. 115 13252–7
[9] Degeraci T and 2019 Structure 27 1034–40
[10] Baxt W G 1991 Ann. Intern. Med. 115 843–8
[11] Karabatak M and Ince M C 2009 Expert Syst. Appl. 36 3465–9
[12] Zupan J, Gasteiger J and Zupan J 1999 Neural Networks in Chemistry and Drug Design (New York: Wiley)
[13] Agatonovic-Kusnir S and Beresford R 2000 J. Pharm. Biomed. Anal. 22 717–27
[14] Tripp R R and Turban E 1992 Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance (New York: McGraw-Hill)
[15] Hubara I, Courbariaux M, Soudry D, El-Yaniv R and Bengio Y 2016 Binarized neural networks Advances in Neural Information Processing Systems pp 4107–15
[16] Rastegari M, Ordonez V, Redmon J and Farhadi A 2016 arXiv:1603.05279
[17] Blum A and Rivest R L 1989 Training a 3-node neural network is NP-complete Advances in Neural Information Processing Systems pp 494–501
[18] Rojas R 2013 Neural Networks: A Systematic Introduction (Berlin: Springer)
[19] Rumelhart D E, Hinton G E and Williams R J 1985 Learning internal representations by error propagation Tech. Rep. California Univ San Diego La Jolla Inst for Cognitive Science
[20] Werbos P J 1990 Proc. IEEE 78 1550–60
[21] Choromanska A, LeCun Y and Arous G B 2015 Open problem: the landscape of the loss surfaces of multilayer networks Conf. on Learning Theory pp 1756–60
[22] Svitsitz G, Czarnecki W M and Pascanu R 2016 arXiv:1611.06310 [stat.ML]
[23] Alizadeh M, Fernández-Marqués J, Lane N D and Gal Y 2019 A systematic study of binary neural networks' optimisation Int. Conf. on Learning Representations
[24] Hochreiter S 1998 Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 06 107–16
[25] Pascanu R, Mikolov T and Bengio Y 2013 On the difficulty of training recurrent neural networks Int. Conf. on Machine Learning pp 1310–8
[26] Shang Y and Wah B W 1996 Computer 29 45–54
