Quantum algorithms for feedforward neural networks

Jonathan Allcock,[1, *] Chang-Yu Hsieh,[1, †] Iordanis Kerenidis,[2, ‡] and Shengyu Zhang[1, 3, §]

[1]*Tencent Quantum Laboratory*
[2]*CNRS, IRIF, Université Paris Diderot, Paris, France*
[3]*The Chinese University of Hong Kong*

Quantum machine learning has the potential for broad industrial applications, and the development of quantum algorithms for improving the performance of neural networks is of particular interest given the central role they play in machine learning today. In this paper we present quantum algorithms for training and evaluating feedforward neural networks based on the canonical classical feedforward and backpropagation algorithms. Our algorithms rely on an efficient quantum subroutine for approximating the inner products between vectors, and on storing intermediate values in quantum random access memory for fast retrieval at later stages. The running times of our algorithms can be quadratically faster than their classical counterparts, since they depend linearly on the number of neurons in the network, as opposed to the number of edges as in the classical case. This makes our algorithms suited for large-scale, highly-connected networks where the number of edges in the network dominates the classical algorithmic running time.

I. INTRODUCTION

Feedforward neural networks play a key role in machine learning, with applications ranging from computer vision and speech recognition to data compression and recommendation systems. In a supervised learning scenario, a network is trained to recognize a set of labelled data by learning a hierarchy of features that together capture the defining characteristics of each label. Once trained, the network can then be used to recognize unlabelled data.

A natural question is whether quantum algorithms can offer any improvement on the classical algorithms currently used for the training and evaluation of feedforward neural networks. There is some qualitative cause for optimism. Firstly, the standard classical algorithms employed typically make heavy use of linear algebra, for which quantum algorithms may have an advantage in certain cases [1]. Secondly, and perhaps more interestingly, it is desirable for neural networks to be robust to noise and small errors, which can achieved by introducing noise during the training process [23]. In quantum computing, the effect of introduced noise can occur naturally. In this light, neural networks are a natural fit for quantum computing. On the other hand, neural networks are highly sequential, and data is often required to be measured and stored at many intermediate steps, a process which would destroy quantum coherence. Furthermore, classical data must first be encoded in quantum states if they are to be evaluated by a quantum algorithm, and in some cases this state-preparation procedure may be a bottleneck preventing a quantum algorithm from running more quickly than classically.

Quantum supervised learning is a fast growing area of research [10, 14, 15, 18, 22, 27], and quantum generalizations of neural networks have been proposed in the literature for feedforward networks [3, 6, 13, 19, 26], Boltzmann Machines [25, 28] and Hopfield networks [17]. The current work is, to our knowledge, the first proposed quantum algorithms for training and evaluating feedforward neural networks that can offer an advantage over the canonical classical feedforward and backpropagation algorithms.

[*] jonallcock@tencent.com
[†] kimhsieh@tencent.com
[‡] jkeren@irir.fr
[§] shengyzhang@tencent.com

More precisely, in this work we present quantum algorithms for training and evaluating robust versions of classical feedforward neural networks — which we denote (ϵ, γ)-feedforward neural networks — based on the standard classical algorithms for feedforward and backpropagation [20], and adapted to exploit the strengths and mitigate the downsides of quantum procedures. Firstly, we make use of the fact that quantum computers can compute approximate inner-products of vectors efficiently and we define a robust inner product estimation procedure that outputs such an estimate in a quantum register. Secondly, we address the state-preparation problem for short length vectors by storing them in a particular data-structure [11, 12], the Quantum Random Access Memory (QRAM), so that their elements may be queried in quantum superposition. For the weight matrices which are too big to be stored efficiently in this manner, we reconstruct their corresponding quantum states by superposing histories of shorter length vectors stored in QRAM, and perform a low-rank initialisation.

A consequence of our design choices is that our quantum algorithms have a running time that depends on a term $\log(1/\gamma)/\epsilon$ where, with probability at least $1-\gamma$, we are able to calculate inner-products to within ϵ accuracy (either relative or absolute, depending on the situation), and on terms R and R' respectively which depend on the values of certain vector norms which appear during the network evaluation and will be explicated in later sections. We give arguments and numerical evidence that indicate that, in practice, these terms may not contribute significantly to the quantum running time . In this case, our first quantum algorithm allows for efficiently training large fully-connected neural networks, where the term E dominates the running time. Informally, our result can be stated as follows:

Quantum Training: [See Theorem 1] *There exists a quantum algorithm for training (ϵ, γ)-feedforward neural networks in time $\tilde{O}\left((TM)^{1.5} N \frac{\log(1/\gamma)}{\epsilon} R\right)$, where T is the number of update iterations, M the number of input samples in each mini-batch, N is the total number of neurons in the network, and R is a factor that depends on the network and training samples.*

Once the neural network is trained it can be used to label new input data. This evaluation is essentially the same as the feedforward algorithm that is used in the training, and we obtain the following result.

Quantum Evaluation: [See Theorem 2] *There exists a quantum algorithm for evaluating an (ϵ, γ)-feedforward neural network in time $\tilde{O}\left(NT_U \frac{\log(1/\gamma)}{\epsilon} R'\right)$, where T_U is the time required to create the quantum states corresponding to the network weights, N is the total number of neurons in the network, and R' is a factor that depends on the network and training samples.*

For a neural network whose weights are already stored in a QRAM data structure or with weights whose corresponding quantum states are easy to construct, the time T_U can be polylogarithmic in TMN. For a network with parameters trained via our quantum training algorithm, we store the network weight matrices W in memory only implicitly, which leads to a time T_U scaling as $O(\sqrt{TM})$. In this case the overall running time equates to $\tilde{O}\left(\sqrt{TM} N \frac{\log(1/\gamma)}{\epsilon} R'\right)$.

In contrast, the classical training algorithm has running time $O(TME)$ while the evaluation algorithm takes time $O(E)$, where E is the total number of *edges* in the neural network. It is interesting to note that the complexity order reduces from the number of edges (neural connections) classically to the number of vertices (neurons) in the quantum algorithm. This gap can be very large; indeed, an average neuron in the human brain has 7000 synaptic connections to other neurons [5].

II. ROBUST FEEDFORWARD NEURAL NETWORKS

Feedforward neural networks are covered by a vast amount of literature (see [8, 16] for good introductory references), and here we simply give a brief summary of the concepts required for the current work. More precisely, we describe the classical algorithms for feedforward neural networks in more detail, generalized to be made robust to small errors, in anticipation of a quantum approach to their training and evaluation. Specifically, we will consider the case where inner products, such as the one appearing in equation (1) below, are not evaluated exactly but, with high probability $1 - \gamma$, can be estimated to within some error tolerance ϵ. We refer to such networks as (ϵ, γ)-feedforward neural networks, of which the standard classical feedforward neural network is a special case. Such robust training is similar in spirit to the technique of multiplicative Gaussian noise and dropout [23], where at every layer of the network noise is either introduced in the calculations, or neurons are probabilistically deactivated. Training in this manner can often improve network performance by both avoiding local minima and preventing over-fitting to the training data. When we turn to the quantum case, the noise comes from the fact that quantum subroutines for calculating inner products are not perfect, but rather output estimates of the true value to within some error tolerance. Our simulation results show that, for reasonable tolerance parameters, this noise similarly does not hurt the network performance.

Let us fix the notation that we will use in the remainder of this paper. For a vector $x \in \mathbb{R}^n$, denote its ℓ_2 norm by $\|x\|$, and for a matrix $A \in \mathbb{R}^{m \times n}$, let A_j and A^k denote its j-th row and k-th column respectively, $\|A\|$ its spectral norm and $\|A\|_F$ its Frobenious norm. We denote the Euclidean inner product between vectors $x, y \in \mathbb{R}^n$ by $\langle x, y \rangle$, and denote $\langle x | y \rangle := \frac{\langle x,y \rangle}{\|x\| \|y\|}$.

We consider feedforward neural network consisting of L layers, with the l-th layer containing n_l neurons. A weight matrix $W^l \in \mathbb{R}^{n_{l-1} \times n_l}$ is associated between layers $l-1$ and l, and a bias vector $b^l \in \mathbb{R}^{n_l}$ is associated to each layer l. The total number of neurons is $N = \sum_{l=1}^{L} n_l$ and the total number of edges in the network is $E = \sum_{l=2}^{L} n_l \cdot n_{l-1}$. For each level l, we define $a^l \in \mathbb{R}^{n_l}$ to be the vector of outputs (activations) of the neurons. Given an activation function f, the network feedforward rule is given by $a_j^l = f(z_j^l)$, where

$$z_j^l = \langle W_j^l, a^{l-1} \rangle + b_j^l. \tag{1}$$

Common activation functions are the sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$ (when neurons take values in $[0, 1]$), the function $\tanh(z) = 2\sigma(2z) - 1$ (when neurons take values in $[-1, 1]$), and the Rectified Linear Unit (ReLU function) $f(x) = \max(0, x)$.

A training set $\mathcal{T} = \{(x^1, y^1), (x^2, y^2) \dots (x^{|\mathcal{T}|}, y^{|\mathcal{T}|})\}$ for a learning task consists of vectors $x^i \in \mathbb{R}^{n_1}$ and corresponding labels $y^i \in \mathbb{R}^{n_L}$. This set is used, via a training algorithm, to adjust the network weights and biases so as to minimize a chosen cost function $C : \mathbb{R}^{n_L} \to \mathbb{R}$, which quantifies the network performance. The goal is to obtain parameters such that when the network evaluates a new input which is not part of the training set, it outputs the correct label with a high degree of accuracy.

Broadly speaking, network training proceeds via the following steps:

1. The weights and biases are initialized. Various techniques are used, but a common choice is to set the biases to a small constant, and to draw the weight matrices randomly from a normal distribution according to $W_{jk}^l \sim \mathcal{N}\left(0, \frac{1}{\sqrt{n_{l-1}}}\right)$ [7].

2. A mini-batch of size M of training pairs are chosen at random from the training set. Each input of the mini-batch is passed through the network and the output compared with its label according to the chosen cost function C. By averaging over the mini-batch one can determine an approximation to the gradient of C, and hence adapt the parameters to reduce the cost. Note that practitioners of stochastic gradient descent typically employ random shuffling, but this makes little difference in performance, nor does it affect the computational cost of the classical or our quantum algorithms.

3. The previous step is repeated T times, each time with a different mini-batch. For long enough T the hope is that the network parameters converge to values that give good network performance on data outside the training set.

In our analysis, we will make use of vectors and matrices indexed by iteration (t), mini-batch (m), and network layer (l). These indices will appear as superscripts e.g. vectors $a^{t,m,l}$, matrices $A^{t,l}$, or its j-th row $A_j^{t,l}$ and its k-th column $(A^{t,l})^k$. In what follows, $\tilde{O}$ hides polylogarithmic factors in NMT.

A. The Feedforward algorithm

We begin by describing how the neural network evaluates an input in the forward direction. Assume we are at iteration $t \in [T]$ of the training and are considering input $m \in [M]$ of the mini-batch, namely $x^{t,m} \in \mathbb{R}^{n_1}$. The goal is to evaluate and store the vectors $z^{t,m,l}$ and $a^{t,m,l}$ at each layer $l \in [L]$ in the network. To this end we apply a robust version of the standard feedforward algorithm where, during the calculation of the inner product between the vectors $W_j^{t,l}$ and $a^{t,m,l-1}$, an error bounded by ϵ (either relative or absolute, depending on whether the inner product is large or small) is introduced.

Subroutine 1. ((ϵ, γ)-**Feedforward**)

Inputs: indices $t \in [T], m \in [M]$; input pair $(x^{t,m}, y^{t,m})$; weight matrices $W^{t,l}$ and biases $b^{t,l}$ for $l \in [L]$; activation function f; accuracy parameters $\epsilon, \gamma > 0$.

```
1. For $j = 1$ to $n_1$ do:
2.     $a_j^{t,m,1} = x_j^{t,m}$
3. For $l = 2$ to $L$ do:
4.     For $j = 1$ to $n_l$ do:
5.         Compute $s_j^{t,m,l}$, such that $\left|s_j^{t,m,l} - \left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle\right| \leq \max\left\{\epsilon\left|\left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle\right|, \epsilon\right\}$ with probability $\geq 1-\gamma$.
6.         Compute and store $z_j^{t,m,l} = s_j^{t,m,l} + b_j^{t,l}$
7.         Compute and store $a_j^{t,m,l} = f(z_j^{t,m,l})$
```

Note that we have not completely specified how the $s_j^{t,m,l}$ are generated in step 5 above: the way this is realized will be left to specific implementations of the algorithm. For instance, one simple classical implementation would be to first compute the inner product $\left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle$ and then add independent Gaussian noise bounded by the maximum of $\epsilon\left|\left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle\right|$ and ϵ. In the quantum case, the $s_j^{t,m,l}$ will be generated by a noisy quantum inner product procedure whose error originates from the amplitude estimation procedure and fulfills the (ϵ, γ) assumptions of step 5. Additionally, the reason we allow for either a relative error $\epsilon\left|\left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle\right|$ or absolute error ϵ is to ensure that when the inner product is close to zero it is still possible to efficiently add this error in the quantum case.

The running time of the above procedure is $O\left(\sum_{l=2}^{L} n_l \cdot n_{l-1}\right) = O(E)$ where we recall that E is the total number of edges in the network. Indeed, at each level l one must evaluate n_l activations, and each activation involves calculating an inner product of dimension n_{l-1} which takes times n_{l-1}. Note that even for the approximate case and for small enough ϵ, the running time of the classical algorithm remains $O(E)$, since the inner product calculation is extremely sensitive and in general one would have to look at a large fraction of the coordinates to obtain an ϵ-error approximation.

B. The Backpropagation algorithm

The second stage of the training algorithm is known as backpropagation. We assume we are at iteration t of the training and using the m-th training input pair of the mini-batch. Moreover, we have already run the feedfoward algorithm for input $x^{t,m}$ and have stored the vectors $a^{t,m,l}$ and $z^{t,m,l}$ for all $l \in [L]$.

Subroutine 2. (**Backpropagation**)

Inputs: indices $t \in [T], m \in [M]$; input pair $(x^{t,m}, y^{t,m})$; vectors $a^{t,m,l}, z^{t,m,l}$, weight matrices $W^{t,l}$ and biases $b^{t,l}$ for $l \in [L]$; derivative f' of activation function f; cost function C; accuracy parameters $\epsilon, \gamma > 0$.

```
1. For $j = 1$ to $n_L$ do:
2.     $\delta_j^{t,m,L} = f'(z_j^{t,m,L}) \frac{\partial C}{\partial a_j^L}$.
3. For $l = L - 1$ to 1 do:
4.     For $j = 1$ to $n_l$ do:
5.         Compute $s_j^{t,m,l}$, such that
           $\left| s_j^{t,m,l} - \left\langle \left(W^{t,l+1}\right)^j, \delta^{t,m,l+1} \right\rangle \right| \le \max\left\{ \epsilon \left| \left\langle \left(W^{t,l+1}\right)^j, \delta^{t,m,l+1} \right\rangle \right|, \epsilon \right\}$ with probability $\ge 1 - \gamma$.
6.         Compute and store $\delta_j^{t,m,l} = f'(z_j^{t,m,l}) s_j^{t,m,l}$.
```

The running time of the back propagation algorithm is again $O(E)$, by the same reasoning as for the feedforward algorithm.

C. The training algorithm

We are now ready to describe the entire training algorithm. At a high level, the algorithm performs T iterations: at each iteration a new mini-batch of inputs is used of size M, and for every input the feedforward algorithm computes the vectors z and a, the backpropagation algorithm computes the vectors δ, and then the weights and biases are updated.

Subroutine 3. (**Training**)

Inputs: input pairs $(x^{t,m}, y^{t,m})$ for $t \in T, m \in [M]$; step parameters $\eta^{t,l}$ for $t \in [T], l \in [L]$; accuracy parameters $\epsilon, \gamma > 0$.

```
1. For $l \in [L]$ do:
2.     Initialise the weights and biases $W^{1,l}, b^{1,l}$.
3. For $t = 1$ to $T$ do:
```

```
4.    For $m = 1$ to $M$ do:
5.        Run the feedfoward algorithm.
6.        Run the backpropagation algorithm.
7.    Update the weights and biases as
          $W_{jk}^{t+1,l} = W_{jk}^{t,l} - \eta^{t,l} \frac{1}{M} \sum_m a_k^{t,m,l-1} \delta_j^{t,m,l}$    (2)
          $b_j^{t+1,l} = b_j^{t,l} - \eta^{t,l} \frac{1}{M} \sum_m \delta_j^{t,m,l}$    (3)
```

The overall running time of the classical training algorithm is $O(TME)$ since there are TM steps and each step runs the feedforward and backpropagation algorithm. The fact that the training of general feedforward neural networks depends linearly on the number of edges makes large-size fully-connected feedforward networks difficult to train, despite its powerful representability. Our quantum algorithm alleviates this problem by reducing the dependence on the network size from the number of edges to the number of vertices.

III. QUANTUM TRAINING

There are several obstacles to improving on the $O(TME)$ classical result using a quantum algorithm. Feedforward neural network training and evaluation is a highly sequential procedure, where at each point one needs to know the results of previously computed steps. Thus, one cannot easily use techniques where the outcome of a quantum procedure is a quantum state (at least, not without incurring prohibitively expensive overhead resources to prevent errors from accumulating exponentially). In addition, a critical step classically is the application of the non-linear activation function to each neuron. Given that quantum mechanics is inherently linear, applying non-linearity to quantum states is non-trivial. Finally, the size of each weight matrix is $n_l \times n_{l-1}$, so even explicitly writing down these matrices for every step of the algorithm takes time $O(E)$.

Let us first fix some quantum notation. Given a vector $x \in \mathbb{R}^n$, define the corresponding normalized quantum state $|x\rangle = \frac{1}{\|x\|} \sum_{j=1}^n x_j |j\rangle$. Given a matrix $A \in \mathbb{R}^{m \times n}$, let A_j and A^k denote its j-th row and k-th column respectively. In this notation we have $|A_j\rangle = \frac{1}{\|A_j\|} \sum_{k=1}^n A_{jk} |k\rangle$ and $|A^k\rangle = \frac{1}{\|A^k\|} \sum_{j=1}^m A_{jk} |j\rangle$.

In what follows, we will make use of the well-known quantum procedures for amplitude amplification and estimation:

Proposition 1 (Amplitude amplification and estimation [2]). *Given access to a unitary operator U acting on k qubits such that $U |0\rangle^{\otimes k} = \sin(\theta) |x, 0\rangle + \cos(\theta) |G, 1\rangle$ (where $|G\rangle$ is arbitrary), $\sin^2(\theta)$ can be estimated to additive error $\epsilon \sin^2(\theta)$ in time $O\left(\frac{T(U)}{\epsilon \sin(\theta)}\right)$ and $|x\rangle$ can be generated in expected time $O\left(\frac{T(U)}{\sin(\theta)}\right)$,where $T(U)$ is the time required to implement U.*

Moreover, we assume that one can store classical data in a data structure such that for any $x \in \mathbb{R}^n$, a unitary operation can be implemented which effects the following transformation

$$|i\rangle |0\rangle \to |i\rangle |x_i\rangle \text{ for } x_i \in \mathbb{R} \text{ and } i \in [n]$$

We also assume that for any set of vectors $x^j, j \in [M]$, we have unitary access to their norms:

$$|j\rangle |0\rangle \rightarrow |j\rangle \left| \left\| x^j \right\| \right\rangle \text{ for } x^j \in \mathbb{R} \text{ and } j \in [M]$$

More precisely, the second register is a k-bit representation of the norm, which introduces an error of $\log(1/k)$, but we will show that these errors can be kept under control. In general, the notation $|0\rangle$ implies an all zero register of appropriate length. Proposition 2 shows that, given such oracular access to the data, which is the same oracular access used for a general Grover's search for example, one can efficiently create quantum states that correspond to these classical vectors, normalized to have unit norm. Each such classical vector is stored as a tree, where the leaves correspond to the vector coordinates, while the intermediate nodes store the square amplitudes that correspond to their subtree. We refer to [12] for more details on these QRAM constructions.

Proposition 2 ([12]). *Let $X \in \mathbb{R}^{M\times N}$. There exists a data structure to store the rows of X such that:*

1. *The size of the data structure is $O(NM)$*
2. *The time to store an element X_{ij} is $O(\log^2(NM))$, and the time to store the whole matrix X is thus $O(NM\log^2(NM))$*
3. *A quantum algorithm that can ask superposition queries to the data structure can perform in time $\mathrm{polylog}(NM)$ the following unitaries:*
   - $U_1 : |i\rangle |0\rangle \rightarrow |i\rangle |X_i\rangle = |i\rangle \frac{1}{\|X_i\|} \sum_j X_{ij} |j\rangle$ *for $i \in [N]$*
   - $U_2 : |0\rangle \rightarrow \frac{1}{\|X\|_F} \sum_{i\in[N]} \|X_i\| |i\rangle$

The same data structures can be of course used for the columns of a matrix as well.

In order to overcome the obstacles in the training of the feedforward neural networks described above we combine a number of ideas. First, we define a hybrid quantum-classical algorithm where all sequential steps are taken classically, and at each step quantum operations are only invoked for estimating the inner products between pairs of quantum states. During feedforward these are states $\left|a^{t,m,l-1}\right\rangle$ and $\left|W_j^{t,l}\right\rangle$ which encode the classical vectors $a^{t,m,l}$ and $W_j^{t,l}$ in their amplitudes, and during backpropagation the inner-products between states $\left|\delta^{t,m,l+1}\right\rangle$ and $\left|\left(W^{t,l+1}\right)^j\right\rangle$ are similarly estimated. This efficient inner product calculation provides, loosely speaking, a factor $O(n_l)$ saving in time per layer of the network. For the inner product estimation we make use of a robust version of the inner-product estimation subroutine of [9].

Lemma 1 (Inner Product Estimation (IPE) [9]). *Let a matrix $A \in \mathbb{R}^{M_1\times N}$ and a matrix $B \in \mathbb{R}^{M_2\times N}$. Assume there exist unitaries U_A and U_B that map $|i\rangle |0\rangle \rightarrow |i\rangle |A_i\rangle$ and $|j\rangle |0\rangle \rightarrow |j\rangle |B_j\rangle$ respectively, and also unitaries V_A and V_B that map $|i\rangle |0\rangle \rightarrow |i\rangle |\|A_i\|\rangle$ and $|j\rangle |0\rangle \rightarrow |j\rangle |\|B_j\|\rangle$, that run in time T_U. Then, there exists a quantum algorithm that computes, for any $\gamma > 0$ and $\epsilon > 0$, $|i\rangle |j\rangle |0\rangle \mapsto |i\rangle |j\rangle \left|\overline{\langle A_i, B_j\rangle}\right\rangle$ where $\left|\overline{\langle A_i, B_j\rangle} - \langle A_i, B_j\rangle\right| \leq \epsilon$ with probability at least $1-\gamma$, in time $\widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon} \|A_i\| \|B_j\|\right)$.*

We now define our Robust Inner Product Estimation procedure where for simplicity we assume we only have two vectors. We will need both a relative error and an absolute error in order to deal with the cases where the inner products are either very large in magnitude or close to zero.

Lemma 2 (Robust Inner Product Estimation (RIPE))**.** *Let $x^0, x^1 \in \mathbb{R}^n$, and some small $\epsilon > 0$. Assume there exists a unitary U_{x^0,x^1} that maps $|0\rangle|0\rangle \to |0\rangle|x^0\rangle$ and $|1\rangle|0\rangle \to |1\rangle|x^1\rangle$ and runs in time T_U, and we also know estimates of the norms of the vectors $\overline{\|x^0\|}, \overline{\|x^1\|}$ such that $\left|\overline{\|x^0\|} - \|x^0\|\right| \leq \frac{\epsilon}{3}\|x^0\|$ and $\left|\overline{\|x^1\|} - \|x^1\|\right| \leq \frac{\epsilon}{3}\|x^1\|$. Then, there exist quantum algorithms that, for any $\gamma > 0$, perform the mapping $|0\rangle|1\rangle|0\rangle \mapsto |0\rangle|1\rangle|s_{01}\rangle$ where with probability at least $1 - \gamma$*

1. $\left|s_{01} - \langle x^0, x^1\rangle\right| \leq \epsilon|\langle x^0, x^1\rangle|$ *in time* $\widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon}\frac{\|x^0\|\|x^1\|}{|\langle x^0,x^1\rangle|}\right)$.

2. $\left|s_{01} - \langle x^0, x^1\rangle\right| \leq \epsilon$ *in time* $\widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon}\|x^0\|\|x^1\|\right)$

Proof. From Lemma 1, for normalized vectors $|x^0\rangle, |x^1\rangle$ there is an algorithm that runs in time $\widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon'}\right)$ and outputs $\left|s_{01} - \langle x^0|x^1\rangle\right| \leq \epsilon'$. By taking $\epsilon' = \frac{\epsilon}{4}\left|\langle x^0|x^1\rangle\right|$ we obtain a relative error algorithm in time $\widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon}\frac{1}{|\langle x^0|x^1\rangle|}\right)$. We can now output the estimator $s'_{01} = \overline{\|x^0\|}\,\overline{\|x^1\|}s_{01}$ and have

$$\begin{aligned}\left|s'_{01} - \langle x^0, x^1\rangle\right| &\leq \left|s'_{01} - \|x^0\|\|x^1\| s_{01}\right| + \left|\|x^0\|\|x^1\| s_{01} - \|x^0\|\|x^1\|\langle x^0, x^1\rangle\right| &(4)\\ &\leq [(1+\epsilon/3)^2 - 1]\|x^0\|\|x^1\|(1+\epsilon/4)\langle x^0|x^1\rangle + \epsilon/4\langle x^0, x^1\rangle &(5)\\ &\leq \epsilon\left|\langle x^0, x^1\rangle\right| &(6)\end{aligned}$$

for small enough ϵ. For the second part just use the first part and consider error $\epsilon/|\langle x^0, x^1\rangle|$. Note also that this implies that there is an algorithm that runs in time $T_{\text{IPE}}(x^0, x^1) = \widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon}\frac{\|x^0\|\|x^1\|}{\max\{1,|\langle x^0,x^1\rangle|\}}\right)$ and achieves an error of $|s_{01} - \langle x^0, x^1\rangle| \leq \max\left\{\epsilon\left|\langle x^0, x^1\rangle\right|, \epsilon\right\}$. □

The reason we need both types of errors is that these inner products could be rather small, in which case the running time for achieving a relative error is prohibitive, or they could be rather large, in which case the running time for achieving an absolute error becomes prohibitive. In practice, we find that the networks can be well trained even when we achieve a relative error when the inner products are large and an additive error when they are very small.

Notice also that we compute the inner products in a register and not as a phase, which will enable us to easily apply a non-linear activation function on it. After computing the estimated inner-products, we write the resulting z, a and δ vectors in a QRAM data structure, which takes time linear in their dimension (up to logarithmic factors), in order to be able to construct the corresponding quantum states efficiently, i.e. in time polylogarithmic in the dimensions.

The last idea of our quantum algorithm is a way to update and store the weight matrices implicitly, without having to explicitly update each element in the matrix, which would have taken time $\tilde{O}(E)$. The trade-off in avoiding an $\tilde{O}(E)$ scaling is an extra cost factor of $O(\sqrt{TM})$, which makes the overall running time of our algorithm basically $\tilde{O}\left((TM)^{1.5}N\right)$ and, while the exact running time also depends on various other factors, loosely speaking has an advantage over the classical training when we have $\sqrt{TM} << N$.

A. Constructing the weight matrices

Before describing the quantum training, we first show how one can create quantum states corresponding to the weight matrices even though, unlike the bias vectors, we will not explicitly store their matrix elements in memory. We will also show how to compute estimates of the norms of the rows and columns of the weight matrices.

Let us start with the initialization of the weight matrices $W^{1,l}$. The common way of initializing these matrices is by choosing independent random variables for each element, but this would take time $O(E)$ to record. To circumvent this, we instead set $W^{1,l}$ to be low rank by selecting a small number r of pairs of random vectors $a^{0,\mu,l}$ and $\delta^{0,\mu,l}$ ($\mu = 1,\dots,r$) and taking the sums of their outer-products. In principle, $r = O(\log n_l)$ random pairs suffice and in section VI we provide a classical justification for this initialization as well as numerical evidence supporting its use in practice. For the purpose of making the notation concise, we assume we choose M such pairs, of which only $r = \max(M, \log n_l)$ are non-zero. Also define $\eta^{0,l} = -1$. According to (2), with this notation the weights can be expressed as

$$W^{t,l}_{jk} = \sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M} \frac{-\eta^{\tau,l}}{M}\delta^{\tau,\mu,l}_j a^{\tau,\mu,l-1}_k$$

The meaning of having access to the weight matrices is the ability to create the states

$$\left|W^{t,l}_j\right\rangle = \frac{1}{\left\|W^{t,l}_j\right\|}\sum_{k=1}^{n_l} W^{t,l}_{jk}\left|k\right\rangle = \frac{1}{\left\|W^{t,l}_j\right\|}\sum_{k=1}^{N}\left(\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M}\frac{-\eta^{\tau,l}}{M}\delta^{\tau,\mu,l}_j a^{\tau,\mu,l-1}_k\right)\left|k\right\rangle$$

Note that at every iteration we can store all the vectors $a^{t,m,l}$ and $\delta^{t,m,l}$ in QRAM in time $O(N\,\mathrm{polylog}(TMN))$ and later we can create the quantum states $\left|a^{t,m,l}\right\rangle$ and $\left|\delta^{t,m,l}\right\rangle$ corresponding to these vectors in time $\mathrm{polylog}(TMN)$.

Let us also define for each t, l, j the matrix $X^{[t,l,j]} \in \mathbb{R}^{t\times M}$, with $\left(X^{[t,l,j]}\right)_{\tau\mu} = \frac{-\eta^{\tau,l}}{M}\delta^{\tau,\mu,l}_j\left\|a^{\tau,\mu,l-1}\right\|$ for $\tau \in \{0,\dots,t-1\}$,$\mu \in [M]$. Then, we define the corresponding quantum state

$$\left|X^{[t,l,j]}\right\rangle = \frac{1}{\left\|X^{[t,l,j]}\right\|_F}\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M} -\frac{\eta^{\tau,l}}{M}\delta^{\tau,\mu,l}_j\left\|a^{\tau,\mu,l-1}\right\|\left|\tau\right\rangle\left|\mu\right\rangle$$

Again, a QRAM data structure for this matrix can be maintained in the same asymptotic running time, since all required quantities have already been stored in QRAM. In other words, whenever a new element $\delta^{\tau,\mu,l}_j$ is stored in QRAM in a way to be able to construct the quantum state corresponding to the vectors $\delta^{\tau,\mu,l}$ we also store in QRAM the value $-\frac{\eta^{\tau,l}}{M}\delta^{\tau,\mu,l}_j\left\|a^{\tau,\mu,l-1}\right\|$ which is the τ,μ-th element of any of the matrices $X^{[t,l,j]}$ for $t > \tau$. We now give a subroutine that shows how the ability to create the state $\left|X^{[t,l,j]}\right\rangle$ can be used to create $\left|W^{t,l}_j\right\rangle$.

Subroutine 4. (Creating $\left|W^{t,l}_j\right\rangle$ and estimating $\left\|W^{t,l}_j\right\|$)

Inputs: indices $t \in [T], l \in \{2,\dots,L\}, j \in [n_l]$. Error parameter $\xi > 0$. Unitary operators U_X and U_a that effect the transformations:

$$U_X\left|t\right\rangle\left|l\right\rangle\left|j\right\rangle\left|0\right\rangle\left|0\right\rangle \to \left|t\right\rangle\left|l\right\rangle\left|j\right\rangle\left|X^{[t,l,j]}\right\rangle$$
$$U_a\left|l\right\rangle\left|\tau\right\rangle\left|\mu\right\rangle\left|0\right\rangle \to \left|l\right\rangle\left|\tau\right\rangle\left|\mu\right\rangle\left|a^{\tau,\mu,l-1}\right\rangle, \quad \text{for } \tau = \{0,\dots,t-1\}, \mu \in [M]$$

1. Start with state $\left|t\right\rangle\left|l\right\rangle\left|j\right\rangle\left|0\right\rangle\left|0\right\rangle\left|0\right\rangle$

2. Apply U_X on the first five registers to get the state

$$|t\rangle|l\rangle|j\rangle\left|X^{[t,l,j]}\right\rangle|0\rangle = |t\rangle|l\rangle|j\rangle\frac{1}{\left\|X^{[t,l,j]}\right\|_F}\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M}-\frac{\eta^{\tau,l}}{M}\delta_j^{\tau,\mu,l}\left\|a^{\tau,\mu,l-1}\right\||\tau\rangle|\mu\rangle|0\rangle$$

3. Apply U_a on registers $\{2,4,5,6\}$ to get

$$|t\rangle|l\rangle|j\rangle\frac{1}{\left\|X^{[t,l,j]}\right\|_F}\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M}-\frac{\eta^{\tau,l}}{M}\delta_j^{\tau,\mu,l}|\tau\rangle|\mu\rangle\sum_k a_k^{\tau,\mu,l-1}|k\rangle$$

4. Apply the Hadamard transformations $|\tau\rangle \to \frac{1}{\sqrt{t}}\sum_x(-1)^{\tau\cdot x}|x\rangle$ and $|\mu\rangle \to \frac{1}{\sqrt{M}}\sum_y(-1)^{\mu\cdot y}|y\rangle$ to produce

$$|t\rangle|l\rangle|j\rangle\frac{1}{\left\|X^{[t,l,j]}\right\|_F\sqrt{Mt}}\sum_x\sum_y\left(\sum_{k=1}^{N}\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M}-\frac{\eta^{\tau,l}}{M}(-1)^{\tau\cdot x+\mu\cdot y}\delta_j^{\tau,\mu,l}a_k^{\tau,\mu,l-1}|k\rangle\right)|x\rangle|y\rangle$$

   Note that the probability of measuring the $|x\rangle$ and $|y\rangle$ registers and obtaining $x=0, y=0$ is

$$\frac{1}{\left\|X^{[t,l,j]}\right\|_F^2 Mt}\sum_{k=1}^{N}\left(\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M}-\frac{\eta^{\tau,l}}{M}\delta_j^{\tau,\mu,l}a_k^{\tau,\mu,l-1}\right)^2 = \frac{\left\|W_j^{t,l}\right\|^2}{\left\|X^{[t,l,j]}\right\|_F^2 Mt}$$

   and the post-measurement state in that case is $|t\rangle|l\rangle|j\rangle\left|W_j^{t,l}\right\rangle$.

5. Use amplitude amplification to produce $\left|W_j^{t,l}\right\rangle$

6. Use amplitude estimation to find an s with $\left|s-\frac{\left\|W_j^{t,l}\right\|^2}{\left\|X^{[t,l,j]}\right\|_F^2 Mt}\right| \leq \xi\frac{\left\|W_j^{t,l}\right\|^2}{\left\|X^{[t,l,j]}\right\|_F^2 Mt}$, and output $\overline{\left\|W_j^{t,l}\right\|} = \left\|X^{[t,l,j]}\right\|_F\sqrt{Mts}$

Lemma 3. *Let $t\in[T], l\in\{2,\dots,L\}, j\in[n_l]$. Assume that the matrix $X^{[t,l,j]}\in\mathbb{R}^{t\times M}$ and vectors $a^{0,l-1}, a^{\tau,\mu,l-1}$ for all $\mu\in[M], \tau\in\{0,\dots,t-1\}$, as well as their norms are stored in QRAM. Then, the above algorithm generates $\left|W_j^{t,l}\right\rangle$ in time $T_W = O\left(\frac{\|X^{[t,l,j]}\|_F}{\|W_j^{t,l}\|}\sqrt{TM}\,\mathrm{polylog}(TMN)\right)$, and returns an estimate $\overline{\left\|W_j^{t,l}\right\|}$ satisfying $\left|\overline{\left\|W_j^{t,l}\right\|}-\left\|W_j^{t,l}\right\|\right| \leq \xi\left\|W_j^{t,l}\right\|$ in time $O(T_W/\xi)$.*

Proof. With the above mentioned data in QRAM, the unitaries U_X and U_a can both be implemented in time $O(\mathrm{polylog}(TMN))$, and hence the quantum state after the Hadamard transformation can be created in this time. This state can be expressed as

$$|\psi\rangle = \sin\theta\left|W_j^{t,l}\right\rangle|0\rangle + \cos\theta|\mathsf{junk}\rangle|1\rangle$$

where $\sin^2\theta = \frac{\|W_j^{t,l}\|^2}{\|X^{[t,l,j]}\|_F^2 Mt}$. The running time for creating $\left|W_j^{l,t}\right\rangle$ then follows directly from amplitude amplification (Theorem 1). The same theorem shows that amplitude estimation may be used to obtain the estimate s in time $O(\|T_W/\xi\|)$ and note that $|\sqrt{s}\left\|X^{[t,l,j]}\right\|_F\sqrt{Mt}-\left\|W_j^{t,l}\right\|| \leq \xi\left\|W_j^{t,l}\right\|$. □

Analogous results clearly also hold for creating quantum states $\left|(W^{t,l})^j\right\rangle$ corresponding to the columns $(W^{t,l})^j$ of the matrix $W^{t,l}$, except in this case the key ratio $\left\|W_j^{t,l}\right\| / \left\|X^{[t,l,j]}\right\|_F$ that appeared in the previous proof is replaced by $\left\|(W^{t,l})^j\right\| / \left\|\tilde{X}^{[t,l,j]}\right\|_F$, where $\left\|\tilde{X}^{[t,l,j]}\right\|_F$ is the norm of the quantum state

$$\left|\tilde{X}^{[t,l,j]}\right\rangle = \frac{1}{\left\|\tilde{X}^{[t,l,j]}\right\|_F} \sum_{\tau=0}^{t-1} \sum_{\mu=1}^{M} -\frac{\eta^{\tau,l}}{M} \left\|\delta^{\tau,\mu,l}\right\| a_j^{\tau,\mu,l-1} \left|\tau\right\rangle \left|\mu\right\rangle$$

B. The quantum feedforward algorithm

The quantum feedforward algorithm is in essence the same as the classical one, adapted to make use of the quantum inner product estimation and quantum access to the weight matrices. We denote by U_{x^0,x^1} a unitary, as in Lemma 2, that maps $|0\rangle|0\rangle \mapsto |0\rangle\left|x^0\right\rangle$ and $|1\rangle|0\rangle \mapsto |1\rangle\left|x^1\right\rangle$, for any two vectors x^0, x^1.

Subroutine 5. (Quantum (ϵ,γ)**-Feedforward**)

Inputs: indices $t \in [T], m \in [M]$; input pair $(x^{t,m}, y^{t,m})$ in QRAM; unitaries $U_{W_j^{t,l},a^{t,m,l-1}}$ for creating $\left|W_j^{t,l}\right\rangle$ and $\left|a^{t,m,l-1}\right\rangle$ in time T_U, estimates of their norms $\overline{\left\|W_j^{t,l}\right\|}$ and $\overline{\left\|a^{t,m,l-1}\right\|}$ to relative error at most $\xi = \epsilon/3$, and vectors $b^{t,l}$ in QRAM for $l \in [L]$; activation function f; accuracy parameters $\epsilon, \gamma > 0$.

```
For $j = 1$ to $n_1$ do:
    $a_j^{t,m,1} = x_j^{t,m}$
For $l = 2$ to $L$ do:
    For $j = 1$ to $n_L$ do:
        Use the RIPE algorithm with unitary $U_{W_j^{t,l},a^{t,m,l-1}}$ to compute $s_j^{t,m,l}$, such that
            $\left|s_j^{t,m,l} - \langle W_j^{t,l}, a^{t,m,l-1}\rangle\right| \leq \max\{\epsilon|\langle W_j^{t,l}, a^{t,m,l-1}\rangle|, \epsilon\}$ with probability $\geq 1-\gamma$
        Compute $z_j^{t,m,l} = s_j^{t,m,l} + b_j^{t,l}$ and store $z_j^{t,m,l}$ in QRAM
        Compute $a_j^{t,m,l} = f(z_j^{t,m,l})$ and store $a_j^{t,m,l}$ in QRAM
```

Lemma 4. *The running time of the quantum feedforward algorithm is* $\tilde{O}\left(\sqrt{TM} N \frac{\log(1/\gamma)}{\epsilon} R_a^{t,m}\right)$*, where we have* $R_a^{t,m} = \frac{1}{N-n_1}\sum_{l=2}^{L}\sum_{j=1}^{n_l} \frac{\|X^{[t,l,j]}\|\|a^{t,m,l-1}\|}{\max\left\{1,\left|\left\langle W_j^{t,l},a^{t,m,l-1}\right\rangle\right|\right\}}$.

Proof. The cost of performing this feedforward procedure is $O\left(\sum_{l=2}^{L}\sum_{j=1}^{n_l} T_{RIPE}(W_j^{t,l}, a^{t,m,l-1})\right) = O\left(N\overline{T}_{RIPE}\right)$, where $T_{RIPE}(W_j^{t,l}, a^{t,m,l-1})$ is the time required to perform Robust Inner Product Estimation between vectors $W_j^{t,l}$ and $a^{t,m,l-1}$, and $\overline{T}_{RIPE}$ denotes the average time (over all layers and neurons) to perform this inner product estimation. From Lemma 2 and using the minimum running time algorithm that achieves the maximum error, we have:

$$T_{RIPE}(W_j^{t,l}, a^{t,m,l-1}) = \widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon} \frac{\left\| W_j^{t,l} \right\| \left\| a^{t,m,l-1} \right\|}{\max\left\{1, \left| \left\langle W_j^{t,l}, a^{t,m,l-1} \right\rangle \right| \right\}} \right).$$

Since the data required to create the $\left| W_j^{t,l} \right\rangle$ and $\left| a^{t,m,l-1} \right\rangle$ states is stored in QRAM, it follows from Lemma 3 that $T_U = O\left(\frac{\|X^{[t,l,j]}\|_F}{\left\| W_j^{t,l} \right\|} \sqrt{TM} \,\mathrm{polylog}(TMN) \right)$. This concludes the proof. □

The factor $R_a^{t,m}$ does not appear in the classical algorithms, and while it is a priori not clear what impact this will have on the running time, we give evidence in the discussion section that this does not impact the running time significantly in practice. On the bright side, our running time saves a factor of $O(N)$ since it depends linearly on N and not on the number of edges E. Last, there is an overhead of $\sqrt{TM}$ which comes from the fact that we only save the weight matrices implicitly. In practice and for large neural networks, we expect that $N >> \sqrt{TM}$.

Note also that when when we say "store $a_j^{t,m,l}$ in QRAM", we mean that the corresponding value is stored as a leaf in a tree data structure and all intermediate nodes of the tree are updated (thus the time to store each element is $\log^2(TMN)$), so that the quantum states corresponding to the normalized vector $a^{t,m,l}$ can be efficiently prepared in time $O(\log(TMN))$ and the norm of the vector is also calculated and stored.

C. The quantum backpropagation algorithm

The quantum backpropagation algorithm again closely follows the steps of its classical counterpart.

Subroutine 6. (Quantum (ϵ, γ)-**Backpropagation**)

Inputs: indices $t \in [T], m \in [M]$; input pair $(x^{t,m}, y^{t,m})$ in QRAM; vectors $a^{t,m,l}, z^{t,m,l}, l \in [L]$ in QRAM; unitaries $U_{(W^{t,l+1})^j, \delta^{t,m,l+1}}$ for creating $\left|(W^{t,l+1})^j\right\rangle$ and $\left|\delta^{t,m,l+1}\right\rangle$ in time T_U, estimates of their norms $\overline{\|(W^{t,l+1})^j\|}$ and $\overline{\|\delta^{t,m,l+1}\|}$ to relative error at most $\xi = \epsilon/3$, and vectors $b^{t,l}$ in QRAM for $l \in [L]$; derivative activation function f'; parameters $\eta^{t,l}$ for $l \in [L]$; accuracy parameters $\epsilon, \gamma > 0$.

```
1. For $j = 1$ to $n_L$ do:
2.     $\delta_j^{t,m,L} = f'(z_j^{t,m,L}) \frac{\partial C}{\partial a_j^L}$.
3. For $l = L - 1$ to 1 do:
4.     For $j = 1$ to $n_l$ do:
5.         Use the RIPE algorithm with unitary $U_{(W^{t,l})^j, \delta^{t,m,l+1}}$ to compute $s_j^{t,m,l}$, such that
               $\left| s_j^{t,m,l} - \langle \left(W^{t,l+1}\right)^j, \delta^{t,m,l+1} \rangle \right| \leq \max\{\epsilon |\langle \left(W^{t,l+1}\right)^j, \delta^{t,m,l+1} \rangle|, \epsilon\}$ with probability $\geq 1 - \gamma$
6.         Compute $\delta_j^{t,m,l} = f'(z_j^{t,m,l}) s_j^{t,m,l}$ and store $\delta_j^{t,m,l}$, $-\frac{\eta^{t,l}}{M} \delta_j^{t,m,l} \left\| a^{t,m,l-1} \right\|$, and $-\frac{\eta^{\tau,l}}{M} \left\| \delta^{t,m,l} \right\| a_j^{t,m,l-1}$
   in QRAM.
```

The running time is similar to the feedforward algorithm for an appropriate definition of $R_\delta^{t,m}$.

Lemma 5. *The running time of the quantum backpropagation algorithm is* $\tilde{O}\left(\sqrt{TM}N\frac{\log(1/\gamma)}{\epsilon}R_\delta^{t,m}\right)$, *where we have* $R_\delta^{t,m} = \frac{1}{N-n_l}\sum_{l=1}^{L-1}\sum_{j=1}^{n_l}\frac{\left\|\tilde{X}^{[t,l+1,j]}\right\|_F\left\|\delta^{t,m,l+1}\right\|}{\max\left\{1,\left|\left\langle (W^{t,l+1})^j,\delta^{t,m,l+1}\right\rangle\right|\right\}}$.

Again, when we say that we store the values $-\frac{\eta^{t,l}}{M}\delta_j^{t,m,l}\left\|a^{t,m,l-1}\right\|$ and $-\frac{\eta^{\tau,l}}{M}\left\|\delta^{t,m,l}\right\|a_j^{t,m,l-1}$ in QRAM, we mean we store these values as leaves of a tree and update the intermediate tree nodes, so that the states $\left|X^{[t,l,j]}\right\rangle$ and $\left|\tilde{X}^{[t,l,j]}\right\rangle$ can be constructed in $\operatorname{polylog}(TMN)$ time and Lemma 3 can be applied.

D. The quantum training algorithm

The quantum training algorithm consists of running the feedforward and the backpropagation algorithms for all inputs in a mini-batch of size M. After each mini batch has been processed, we explicitly update the biases b in QRAM, but we do not explicitly update the weights W, since this would take time $O(E)$. Instead, we compute an estimate of the norm of the rows and columns of the weight matrices and keep a history of the a and δ vectors in memory so that we can create the quantum states corresponding to the weights on the fly.

Subroutine 7. (**Quantum Training**)

Inputs: input pairs $(x^{t,m}, y^{t,m})$ for all $t \in T, m \in [M]$, parameters $\eta^{t,l}$, for $t \in [T], l \in [L]$, and $\epsilon, \gamma > 0$.

```
1. Initialise the weights and biases $W^{1,l}, b^{1,l}$ for $l \in [L]$ with a low-rank initialization.
2. For $t = 1$ to $T$ do:
3.     For $m = 1$ to $M$ do:
4.         Run the quantum feedfoward algorithm.
5.         Run the quantum backpropagation algorithm.
6.     Compute the biases and update the QRAM with
           $b_j^{t+1,l} = b_j^{t,l} - \eta^{t,l}\frac{1}{M}\sum_m \delta_j^{t,m,l}$
7.     Compute the estimates of the norms $\overline{\left\|W_j^{t+1,l}\right\|}$ and $\overline{\left\|(W^{t+1,l})^j\right\|}$ with relative error $\xi = \epsilon/3$ such
       that the assumptions in the feedforward and backpropagation algorithms are met.
```

Theorem 1. *The running time of the quantum training algorithm is* $\tilde{O}\left((TM)^{1.5}N\frac{\log(1/\gamma)}{\epsilon}R\right)$, *where* $R = R_a + R_\delta + R_W$, $R_a = \frac{1}{TM}\sum_{t,m}R_a^{t,m}$, $R_\delta = \frac{1}{TM}\sum_{t,m}R_\delta^{t,m}$, *and* $R_W = \frac{1}{T}\sum_t(R_{W_r}^t + R_{W_c}^t)$, *with* $R_{W_r}^t = \frac{1}{M}\frac{1}{N-n_1}\sum_{l=2}^{L}\sum_{j=1}^{n_l}\frac{\left\|X^{[t,l,j]}\right\|_F}{\left\|W_j^{t,l}\right\|}$ *and* $R_{W_c}^t = \frac{1}{M}\frac{1}{N-n_1}\sum_{l=2}^{L}\sum_{j=1}^{n_l}\frac{\left\|\tilde{X}^{[t,l,j]}\right\|_F}{\left\|(W^{t,l})^j\right\|}$.

Proof. The first two terms of the running time come from the feedforward and backpropagation. The last term comes from the estimation of the norms which only happens once for each mini-batch. Let's fix t, l, j and look at the estimation of each of the norms $\overline{\left\| W_j^{t+1,l} \right\|}$. For this we have that the algorithm takes time $O(T_w/\xi)$, with $T_W = O\left(\frac{\left\| X^{[t,l,j]} \right\|_F}{\left\| W_j^{t,l} \right\|} \sqrt{TM} \, \text{polylog}(TMN) \right)$, and we take $\xi = \epsilon/3$. Hence, we get the ratio $R_{W_r}^t = \frac{1}{M} \frac{1}{N-n_1} \sum_{l=2}^{L} \sum_{j=1}^{n_l} \frac{\left\| X^{[t,l,j]} \right\|_F}{\left\| W_j^{t,l} \right\|}$ and similarly for the estimation of the columns $R_{W_c}^t = \frac{1}{M} \frac{1}{N-n_1} \sum_{l=2}^{L} \sum_{j=1}^{n_l} \frac{\left\| \tilde{X}^{[t,l,j]} \right\|_F}{\left\| (W^{t,l})^j \right\|}$. Then we can define $R_W = \frac{1}{T} \sum_t (R_{W_r}^t + R_{W_c}^t)$ as needed.

□

IV. QUANTUM EVALUATION

The quantum procedure for evaluating a trained neural network, where the goal is to output the predicted label of a new input data vector, is essentially the same as the quantum feedforward algorithm. Assume there is a new input pair (x, y) that we want to evaluate, and that we have unitaries $U_{W_j^l, a^{l-1}}$ for creating $\left| W_j^l \right\rangle$ and $\left| a^{l-1} \right\rangle$ in time T_U, estimates of their norms $\overline{\left\| W_j^l \right\|}$ and $\overline{\left\| a^{l-1} \right\|}$ to relative error at most $\xi = \epsilon/3$, and vectors b^l in QRAM for $l \in [L]$. We have also fixed an activation function f and accuracy parameters $\epsilon, \gamma > 0$. Then, by Lemma 4, we have the following for the running time of the quantum evaluation.

Theorem 2. *There exists a quantum algorithm for evaluating an (ϵ, γ)-feedforward neural network in time $\tilde{O}\left(T_U N \frac{\log(1/\gamma)}{\epsilon} R_E \right)$, where $R_E = \frac{1}{N-n_1} \sum_{l=2}^{L} \sum_{j=1}^{n_l} \frac{\left\| W_j^l \right\| \left\| a^{l-1} \right\|}{\max\left\{ 1, \left| \left\langle W_j^l, a^{l-1} \right\rangle \right| \right\}}$*

V. SIMULATION

We have proposed a quantum algorithm for training and evaluating feedforward neural networks and in this section we perform numerical simulations to provide evidence that the quantum training algorithm works well in practice, and to understand the factors entering its running time, since it depends on a number of parameters that need to be estimated.

We classically simulate the training and evaluation of (ϵ, γ)-feedforward neural networks on the MNIST handwritten digits data set consisting of 60,000 training examples and 10,000 test examples. A network with $L = 4$ layers and dimensions $[n_1, n_2, n_3, n_4] = [784, 100, 30, 10]$ was used, with $M = 100, T = 7500, \eta = 0.05$, tanh activation function and mean squared error cost function $C = \frac{1}{2M} \sum_m \left\| y^{t,m} - a^{t,m,L} \right\|^2$. For this network size we have $N = 924$, $E = 81,700$. Our results are summarized in Table I. 'Standard' weight initialization refers to drawing the initial weight matrix elements from appropriately normalized Gaussian distributions $W_{jk}^l \sim \mathcal{N}\left(0, \frac{1}{\sqrt{n_{l-1}}} \right)$ [7], and 'Low rank' refer to weight initialization as described in Section III A, with rank $r = 6$. In all cases, Gaussian noise drawn from $N(\epsilon/2, 0)$ was added to each inner product evaluated throughout the network. Note that no regularization was used to improve the network performance, and as ϵ was varied the other network hyper-parameters were not re-optimized. We find that for $\gamma = 0.05$ and various values of ϵ, the network achieves high accuracy whilst incurring only modest contributions to the running time from the quantum-related terms R_a, R_δ and R_W. For the $\epsilon = 0.3$ case we calculate the values of R_a, R_δ and the two components of R_W, as a function of the number of gradient update steps t. These results appear in Figs. 1.

ϵ	0 %	0.1	0.3	0.5
Standard	96.9%	97.1%	96.9%	96.2%
Low rank	-	96.7%	96.6%	96.3%
$\log(1/\gamma)/\epsilon$	-	30	10	6
R_a	-	23.6	22.6	21.7
R_δ	-	0.1	0.1	0.2
R_W	-	0.03	0.03	0.03

TABLE I. (ϵ,γ)-feedforward neural network accuracy on the MNIST handwriting data set, for various values of ϵ, and $\gamma = 0.05$. A network with $L = 4$ layers and dimensions $[n_1, n_2, n_3, n_4] = [784, 100, 30, 10]$ was used, with $M = 100, T = 7500, \eta = 0.05$, tanh activation function and mean squared error cost function.

VI. DISCUSSION

We have presented a quantum algorithm for training an (ϵ,γ)-feedforward neural network which takes time

$$\tilde{O}\left((TM)^{1.5} N \frac{\log(1/\gamma)}{\epsilon} R\right)$$

This is the first algorithm for training feedforward neural networks with a running time that is better than quadratic in the number of neurons, opening the way to training larger size neural networks. We provide below a number of remarks on our algorithm.

Performance of (ϵ,γ)-feedforward neural networks. Firstly, while the notion of introducing noise in the inner product calculations of an (ϵ,γ)-feedforward neural network is, as previously mentioned, similar in spirit to known classical techniques for improving neural network performance, there are certain differences. Classically, dropout and multiplicative Gaussian noise [23] involve introducing perturbations post-activation: $a_j^l \to a_j^l r_j$, where $r_j \sim N(1,1)$ (multiplicative Guassian noise) or $r_j \sim Bernoulli(p)$ (dropout), whereas in our case the noise due to inner-product estimation occurs pre-activation. Furthermore, classical methods are typically employed during the training phase but, once the network parameters have been trained, new points are evaluated without the introduction of noise. This is in contrast to our quantum algorithm where the evaluation of new points inherently also involves errors in inner-product estimation. However, numerical simulation (see Table I) of the noise model present in our quantum algorithm shows that (ϵ,γ)-networks may tolerate modest values of noise, and values of ϵ and γ can be chosen for which the network performance does not suffer significantly, whilst at the same time do not contribute greatly to the factor of $\log(1/\gamma)/\epsilon$ that appears in the quantum running time.

Low rank initialization. One assumption we make in our quantum algorithm is a low-rank initialization of the network weight matrices, compared with freedom to choose full-rank weights classically. This assumption is made in order to avoid a time of $O(E)$ to input the initial weight values into QRAM. Low rank approximations to network weights have found applications classically in both speeding up testing of trained networks [4, 21, 29] as well as in network training [24], in some cases delivering significant speedups without sacrificing much accuracy. We find numerically that the low-rank initialization we require for our quantum algorithm works as well as full rank for a range of ϵ values (see Table I).

Quantum training running time. Compared with the classical running time of $O(TME)$, our quantum algorithm scales with the number of neurons in the network as opposed to the number of edges. However, this comes at the cost of a square root penalty in the number of iterations and mini-batch size, and it is an open question to see how to remove this term. The quantum algorithm also has additional factors of $\log(1/\gamma)/\epsilon$ and $R = R_a + R_\delta + R_W$ which do not feature in the classical algorithm. While the ϵ and γ

can be viewed as hyperparameters, which can be freely chosen, the impact of the R terms warrant further discussion.

The ratio $\left\|X^{[t,l,j]}\right\| / \left\|W_j^{t,l}\right\|$ appears in both R_a and R_W (in the $R^t_{W_r}$ contribution) and similarly the ratio $\left\|\tilde{X}^{[t,l,j]}\right\| / \left\|\left(W^{t,l}\right)^j\right\|$ appears in R_δ and the $R^t_{W_r}$ contribution to R_W. While exact values may be difficult to predict, one can expect the following large t behaviour: Classically, initial weight matrices are typically chosen so that the entries are drawn from a normal distribution with standard deviation $1/\sqrt{\text{row length}}$, which would give $\left\|W_j^{1,l}\right\| \approx 1$, and a similar scenario can hold with low rank initialization. As the weights are updated according to equation (2), and since changes in individual matrix elements may be positive or negative, for a constant step size η one expects $\left\|W_j^{t,l}\right\|$ to roughly grow proportionally to $\sqrt{t\eta}$. The norm $\left\|X^{[t,l,j]}\right\|$ has value

$$\left\|X^{[t,l,j]}\right\|_F = \sqrt{\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M}\left(\frac{\eta}{M}\delta_j^{\tau,\mu,l}\left\|a^{\tau,\mu,l-1}\right\|\right)^2}$$

and, as t increases and the network becomes close to well trained, we expect $\delta_j^{\tau,\mu,l} \to 0$, whereas for activations bounded in the range $[-1,1]$, as is the case for the tanh function, $\left\|a^{t,l-1}\right\| \leq \sqrt{n_{l-1}}$. We thus expect $\left\|X^{[t,l,j]}\right\|$ to saturate for large t and not grow in an unbounded fashion. Fig. 1(a) showing the time averaged values of $R^t_{W_r}$ and $R^t_{W_c}$ is consistent with these ratios saturating to very small values over time, in fact values less than 0.014.

The term $R_a = \frac{1}{TM}\sum_{t,m} R_a^{t,m}$ is an average over iterations and mini-batch elements of terms $R_a^{t,m}$, which themselves are averages over neurons in the network of ratios and products of matrix and vector norms:

$$R_a^{t,m} = \frac{1}{N-n_1}\sum_{l=2}^{L}\sum_{j=1}^{n_l}\frac{\left\|X^{[t,l,j]}\right\|\left\|a^{t,m,l-1}\right\|}{\max\left\{1, \left|\left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle\right|\right\}}$$

As discussed, we expect $\left\|X^{[t,l,j]}\right\|$ to saturate for large t, and for activations in $[-1,1]$ we have $\left\|a^{t,m,l-1}\right\| \leq \sqrt{n_{l-1}}$. However, as the network becomes well trained, one expects the inner products $\left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle$ to become large in magnitude so that the neurons have post-activation values close to ±1. It is thus reasonable to expect the R_a to saturate or even decline for large t. This is consistent with our results in Fig. 1(b). One expects similar large t behaviour for R_δ, except intuitively R_δ should be much smaller than R_a since $\left\|\delta^{t,m,l}\right\|$ should become very small as the network becomes well trained. Fig. 1(c) displays this expected behaviour. While these simulation results are already promising, we expect the quantum advantage in the running time to become more prominent for larger size neural networks.

Let us add a final remark. In our training algorithm we used classical inputs and showed that the number of iterations required for convergence is similar to the case of classical robust training. One can also consider using superpositions of classical inputs for the training, which could conceivably reduce the number of iterations or size of mini-batch required. We leave this as an interesting open direction for future work.

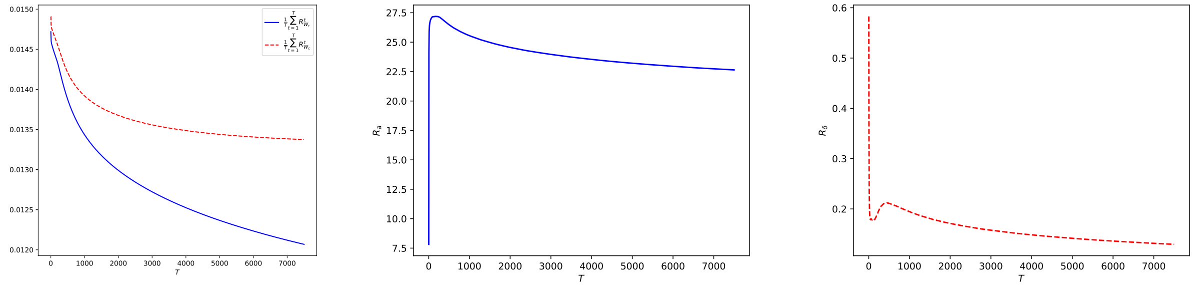

FIG. 1. R_{W_r}, R_a, and R_δ for the $(\epsilon, \gamma) = (0.3, 0.05)$ network of Table I.

[1] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195, 2017.

[2] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[3] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. *arXiv preprint arXiv:1711.11240*, 2017.

[4] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[5] David A Drachman. Do we have brain to spare? *Neurology*, 64(12):2004–2005, 2005.

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

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[8] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[9] Iordanis Kerenidis, Jonas Landman, Alessandro Luongo, and Anupam Prakash. q-means: q-means: A quantum algorithm for unsupervised machine learning. *Manuscript 2018.*

[10] Iordanis Kerenidis and Alessandro Luongo. Quantum classification of the mnist dataset via slow feature analysis. *arXiv:1805.08837*, 2018.

[11] Iordanis Kerenidis and Anupam Prakash. Quantum gradient descent for linear systems and least squares. *arXiv:1704.04992*, 2017.

[12] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 67. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[13] Yidong Liao, Oscar Dahlsten, Daniel Ebler, and Feiyang Liu. Quantum advantage in training binary neural networks. *arXiv preprint arXiv:1810.12948*, 2018.

[14] Yang Liu and Shengyu Zhang. Fast quantum algorithms for least squares regression and statistic leverage scores. In *International Workshop on Frontiers in Algorithmics*, pages 204–216. Springer, 2015.

[15] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013.

[16] Michael Nielsen. Deep learning. *Neural Networks and Deep Learning. Retrieved Sep*, 15, 2017.

[17] Patrick Rebentrost, Thomas R Bromley, Christian Weedbrook, and Seth Lloyd. Quantum hopfield neural network. *Physical Review A*, 98(4):042308, 2018.

[18] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[19] Jonathan Romero, Jonathan P Olson, and Alan Aspuru-Guzik. Quantum autoencoders for efficient compression of quantum data. *Quantum Science and Technology*, 2(4):045001, 2017.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[21] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE, 2013.

[22] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Prediction by linear regression on a quantum computer. *Physical Review A*, 94(2):022342, 2016.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[24] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[25] Guillaume Verdon, Michael Broughton, and Jacob Biamonte. A quantum algorithm to train neural networks using low-depth circuits. *arXiv preprint arXiv:1712.05304*, 2017.

[26] Kwok Ho Wan, Oscar Dahlsten, Hlér Kristjánsson, Robert Gardner, and MS Kim. Quantum generalisation of feedforward neural networks. *npj Quantum Information*, 3(1):36, 2017.

[27] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *Quantum Information & Computation*, 15(3-4):316–356, 2015.

[28] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum deep learning. *Quantum Information & Computation*, 16(7-8):541–587, 2016.

[29] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.