# Quantum Algorithms for Feedforward Neural Networks

JONATHAN ALLCOCK and CHANG-YU HSIEH, Tencent Quantum Laboratory
IORDANIS KERENIDIS, CNRS, IRIF, Université Paris Diderot
SHENGYU ZHANG, Tencent Quantum Laboratory

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. 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 inner products between vectors in a robust way, and on implicitly storing intermediate values in quantum random access memory for fast retrieval at later stages. The running times of our algorithms can be quadratically faster in the size of the network than their standard classical counterparts since they depend linearly on the number of neurons in the network, and not on the number of connections between neurons. Furthermore, networks trained by our quantum algorithm may have an intrinsic resilience to overfitting, as the algorithm naturally mimics the effects of classical techniques used to regularize networks. Our algorithms can also be used as the basis for new quantum-inspired classical algorithms with the same dependence on the network dimensions as their quantum counterparts but with quadratic overhead in other parameters that makes them relatively impractical.

CCS Concepts: • **Theory of computation → Quantum computation theory**; • **Computing methodologies → Neural networks**; **Machine learning algorithms**;

Additional Key Words and Phrases: Quantum algorithms, qRAM

**ACM Reference format:**
Jonathan Allcock, Chang-Yu Hsieh, Iordanis Kerenidis, and Shengyu Zhang. 2020. Quantum Algorithms for Feedforward Neural Networks. *ACM Trans. Quantum Comput.* 1, 1, Article 6 (September 2020), 24 pages.
https://doi.org/10.1145/3411466

## 1 INTRODUCTION

Machine learning has been one of the great success stories of computation in recent times. From self-driving cars and speech recognition to cancer detection and product recommendation, the applications of machine learning have had a transformative impact on our lives. At the heart of many machine learning tasks are artificial neural networks: groups of interconnected computational nodes loosely modeled on the neurons in our brains. In a supervised learning scenario, a neural network is first trained to recognize a set of labeled data by learning a hierarchy of features that together capture the defining characteristics of each label. Once trained, the network

Authors' addresses: J. Allcock, C.-Y. Hsieh, and S. Zhang, Tencent Quantum Laboratory, Tencent, Shenzhen, Guangdong, China, 518057; emails: {jonallcock, kimhsieh, shengyzhang}@tencent.com; I. Kerenidis, CNRS, IRIF, Université Paris Diderot, Paris, France; email: jkeren@irif.fr.

2643-6817/2020/09-ART6 $15.00
https://doi.org/10.1145/3411466

can then be evaluated on previously unseen data to predict their corresponding labels. Although their origins can be traced back to the 1940s, artificial neural networks enjoyed a resurgence of interest in the 1980s when a method for computing gradients known as backpropagation [36] was popularized, which dramatically increased the efficiency with which networks could be trained to recognize data. The following decades have seen researchers develop ever more sophisticated techniques to improve the performance of neural networks, and the best algorithms can now outperform humans on realistic image recognition tasks [20]. Regardless of the additional techniques used, backpropagation remains the key algorithmic component for neural network training.

Despite the progress made over the years, significant resources are still required to train deep networks needed for industrial and commercial grade problems, and long training times on clusters of GPUs are often necessary. Furthermore, a network trained on a particular dataset for a particular task may not perform well on another task. For this, an entirely different network may need to be trained from scratch. There is thus considerable benefit in any method for speeding up or otherwise improving the training and evaluation of neural networks.

The developments in machine learning have been accompanied by the rise of another potentially transformative technology: quantum computing. From the original proposal of Feynman [13] in the 1980s for using the rules of quantum mechanics to carry out computation, quantum computing has developed into one of the most exciting fields of research today. The past several years have seen increased attention on applications of quantum computing for industry, of which quantum chemistry, material design, optimization, and quantum machine learning are leading candidates. A natural and fundamentally important question at the intersection of machine learning and quantum computing is whether quantum algorithms can offer any improvement on the classical algorithms currently used for neural networks.

Several challenges present themselves immediately. First, the process of training and evaluating neural networks is highly sequential. Quantum algorithms can be a natural fit for performing tasks in parallel by harnessing the phenomenon of quantum superposition. However, they are poorly suited to situations where data must be computed and stored at many intermediate steps, a process that destroys quantum coherence. Second, neural network performance relies critically on the ability to perform non-linear transformations of the data. Quantum mechanics, however, is linear, and effectively implementing non-linear transformations can be non-trivial [5]. Finally, if a quantum algorithm is to be used for training a classical neural network, the training data and the parameters that define the network must first be encoded in quantum states. Unless there is an efficient method for preparing these quantum states, the state-preparation procedure can be a time-consuming bottleneck preventing a quantum algorithm from running more quickly than classically. This is especially challenging for the parameters corresponding to the network weights that are so numerous as to rule out efficient methods for generating their corresponding quantum states directly.

There is some qualitative cause for optimism. The standard classical neural network algorithms typically employed make heavy use of linear algebra, for which quantum algorithms may have an advantage in certain cases [3]. Additionally, it is desirable for neural networks to be robust to noise and small errors, which can achieved classically by introducing perturbations during the training process [42]. In quantum computing, the randomness inherent in the outcome of measurements can have the effect of such introduced perturbations, and if harnessed correctly, a quantum algorithm may achieve a natural robustness to noise without additional cost. This is important, as in many cases, raw computational speed is not necessarily the main concern in practical machine learning. Other issues such as the size of the datasets, generalization errors, or how robust the algorithms are to noise and perturbations may indeed be of much greater practical concern. Nevertheless, there is an explicit connection between speed and performance. In many cases, the bottleneck is the fact that one can only spend, say, a day for training and not a year. With this restriction, one selects a neural network with a given size so that training can be completed in the allotted time. With faster training, larger neural networks can be trained, and more experiments can be conducted for different choices of network architecture, loss function, and hyper-parameters. Taken together, these can all lead to better eventual performance.

In this work, we consider quantum algorithms for feedforward neural networks, in which the propagation of information proceeds from start to finish, layer by layer, without any loops or cycles. Feedforward neural networks not only constitute a canonical category of neural network of fundamental importance in their own right but also form key components in practical machine learning architectures such as deep convolutional networks and autoencoders. We propose quantum algorithms for training and evaluating robust versions of classical feedforward neural networks—which we call $(\epsilon, \gamma)$-feedforward neural networks—based on the standard classical algorithms for feedforward and backpropagation, adapted to overcome the challenges and exploit the opportunities mentioned previously. To achieve this, we use the fact that quantum computers can compute approximate inner products of vectors efficiently and define a robust inner product estimation (RIPE) procedure that outputs such an estimate in a quantum data register. By having inner products stored in register rather than in the phases of quantum states, we are able to directly implement non-linear transformations on the data, circumventing the problem of non-linearity entirely. We address the state-preparation problem for short length vectors by storing them in a particular data structure [23, 24, 52] in quantum random access memory (qRAM) [15, 16, 32] so that their elements may be queried in quantum superposition and their corresponding states generated efficiently. For network weight matrices that are too big to be stored efficiently in this manner, we reconstruct their corresponding quantum states indirectly by superposing histories of shorter length vectors stored in qRAM.

In a recent important theoretical work, Tang [45] showed that the data structure required for fast qRAM-based inner product estimation can also be used to classically estimate inner products. Quantum algorithms based on such data structures can thus give rise to quantum-inspired classical ones as well [14, 44]—for instance, classical algorithms based on their quantum counterparts can be defined with only a polynomial slowdown in running time. However, in practice, the polynomial factors can make a difference, and analysis of a number of such algorithms [1] shows that care is needed when assessing their performance relative to the quantum algorithms from which they were inspired. The situation is the same with the quantum algorithms we propose here. New classical algorithms for training and evaluating $(\epsilon, \gamma)$-feedforward neural networks based on our quantum algorithms can indeed be defined, and in a later section we compare these with both their quantum versions and the standard classical algorithms. While interesting from a theoretical perspective, these quantum-inspired algorithms will always have worse performance than their quantum versions, and it is unclear whether they will ever perform well enough in practice to replace existing methods.

Quantum machine learning [3, 6, 11, 39, 40] in general and quantum supervised learning [22, 28, 29, 34, 41, 49] in particular are fast-growing areas of research. However, although quantum generalizations of neural networks have been proposed for feedforward networks [5, 7, 12, 27, 35, 38, 47], Boltzmann machines [25, 46, 50, 51], and Hopfield networks [33], the current work presents, to the best of our knowledge, the first proposed quantum algorithms for training and evaluating feedforward neural networks that can, in principle, offer an advantage over the canonical classical feedforward and backpropagation algorithms. Although it remains to be seen whether our algorithms can be used in the future to obtain a practical advantage over classical methods, we believe, backed with our theoretical analysis and initial simulations, that our results indicate one very promising way that quantum techniques can be applied to neural networks and that in the future more sophisticated practical techniques may be built on these or other ideas.

## 2 RESULTS

We will discuss $(\epsilon, \gamma)$-feedforward neural networks in more detail later. For now, it is enough to think of these as network in which, with probability at least $1 - \gamma$, inner products between vectors can be computed to within error $\epsilon$ of the true value. Our main results are the following.

Quantum Training (See Theorem 1). *There exists a quantum algorithm for training $(\epsilon, \gamma)$-feedforward neural networks in time $\tilde{O}((TM)^{1.5} N \frac{\log(1/\gamma)}{\epsilon} R)$, 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, and which numerical evidence suggests is small for many practical parameter regimes.*

Here and in what follows, $\tilde{O}$ hides polylogarithmic factors in $TMN$. 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). *For a neural network whose weights are explicitly stored in a qRAM data structure, there exists a quantum algorithm for evaluating an $(\epsilon, \gamma)$-feedforward neural network in time $\tilde{O}(N \frac{\log(1/\gamma)}{\epsilon} R_e)$, where $N$ is the total number of neurons in the network, and $R_e$ is a factor that depends on the network and training samples, and is expected to be small for practical parameter regimes.*

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 7,000 synaptic connections to other neurons [10].

The running time of our quantum algorithms also depends on $\log(1/\gamma)/\epsilon$ and on terms $R$ and $R_e$, respectively, which depend on the values of certain vector norms that appear during the network evaluation and will be explicated in later sections. Although the combined effect of these terms must be taken into account, we will give arguments and numerical evidence that indicate that, in practice, these terms do not contribute significantly to the running time.

### 2.1 Feedforward Neural Networks

Feedforward neural networks are covered by a vast amount of literature, and good introductory references are available [18, 30]. Here we simply give a brief summary of the concepts required for the current work.

A feedforward neural network consists of $L$ layers, with the $l$-th layer containing $n_l$ neurons. A weight matrix $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ is associated between layers $l-1$ and $l$, and a bias vector $b^l \in \mathbb{R}^{n_l}$ is associated to each layer $l$ (Figure 1). 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 $l$, let $a^l \in \mathbb{R}^{n_l}$ be the vector of outputs (activations) of the neurons. Given a non-linear activation function $f$, the network feedforward rule is given by $a_j^l = f(z_j^l)$, where $z_j^l = \sum_k W_{jk}^l a_l^{l-1} + b_j^l$ (Figure 2). It will be convenient to express this as

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

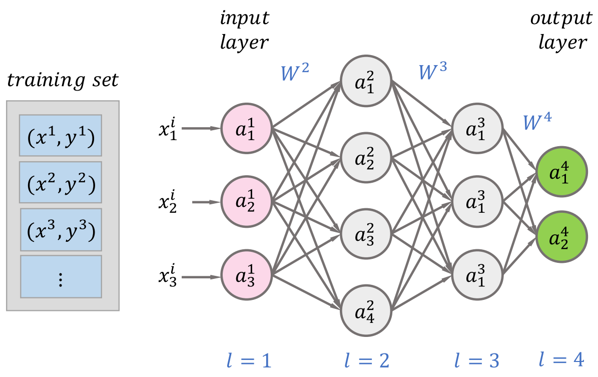


Fig. 1. Feedforward neural network with $L = 4$ layers. A weight matrix $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ is associated between levels $l-1$ and $l$, and a bias vector $b \in \mathbb{R}^{n_l}$ (not shown) is associated to each level $l$. During training, a (data, label) pair $(x^i, y^i)$ is chosen from the training set, and the data is fed into the input layer of neurons. The weights and biases in the network determine the activations of subsequent layers of neurons. Finally, the values from the output layer of neurons are compared with the true label $y^i$.

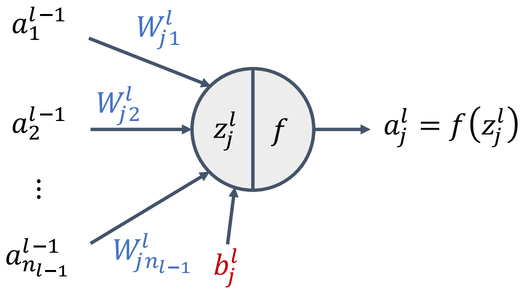


Fig. 2. The $j$-th neuron at level $l$ of a network. The output activation of the neuron is computed in two steps. First, the value $z_j = \sum_k W_{jk}^l a_k^{l-1} + b_j^l$ is calculated, which depends on the weights $W_{ji}^l$ and bias $b_j^l$, as well as the activations from the previous layer of neurons. A non-linear activation function $f$ is then applied to $z_j^l$, and the value $f(z_j)$ is output.

where $\langle x, y \rangle$ is the Euclidean inner product between vectors $x$ and $y$, and $W_j^l$ is the $j$-th row of $W^l$ (i.e., $(W_j^l)_k = W_{jk}^l$). For simplicity, we assume that the activation function is the same across all layers in the network, although our algorithm works more generally when the activation function is allowed to vary across layers. Common activation functions are the sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$ (when neurons take values in $[0, 1]$), the hyperbolic 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 $\mathcal{T} = \{(x^1, y^1), (x^2, y^2), \ldots\}$ 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 that is not part of the training set, it outputs the correct label with a high degree of accuracy.

Classically, network training consists of the following steps:

(1) *Initialization of weights and biases*: 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 $W_{jk}^l \sim \mathcal{N}(0, \frac{1}{\sqrt{n_{l-1}}})$ [17].

(2) *Feedforward*: Select a pair $(x, y)$ from the training set. Assign the neurons in the first layer of the network to have activations $a_j^1 = x_j$. Pass through the network layer-by-layer, at each layer $l$ computing and storing the vectors $z^l = \langle W_j^l, a^{l-1}\rangle + b_j^l$ and $a^l = f(z_j^l)$. The running time of this procedure is $O(\sum_{l=2}^{L} n_l \cdot n_{l-1}) = O(E)$: at each level $l$, one must evaluate $n_l$ activations, and each activation involves calculating an inner product of dimension $n_{l-1}$ that takes times $n_{l-1}$.

(3) *Backpropagation*: Given $z^L$ and $a^L$ at the end of the feedforward process, define vector $\delta^L$ with components $\delta_j^L = f'(z_j^L)\frac{\partial C}{\partial a_j^L}$, where $C$ is the chosen cost function. Then, proceeding backward through the network, compute and store the vectors $\delta_j^l = f'(z_j^l)\langle (W^{l+1})_j^T, \delta^{l+1}\rangle$, where $(W^{l+1})^T$ is the matrix transpose of $W^{l+1}$. The running time of the backpropagation algorithm is again $O(E)$, by the same reasoning as for the feedforward algorithm.

(4) *Update weights and biases*: Repeat the feedforward and backpropagation steps for a mini-batch of inputs of size $M$, then update the weights and biases by stochastic gradient descent, according to

$$W_{jk}^{t+1,l} = W_{jk}^{t,l} - \eta^{t,l}\frac{1}{M}\sum_{m}^{M} a_k^{t,m,l-1}\delta_j^{t,m,l}, \tag{2}$$

$$b_j^{t+1,l} = b_j^{t,l} - \eta^{l}\frac{1}{M}\sum_{m}^{M} \delta_j^{t,m,l}, \tag{3}$$

where the superscripts $t \in [T]$ and $m \in [M]$ denote the iteration number and mini-batch element, respectively, and $\eta^{t,l} > 0$ are update step sizes.

(5) *Iterate*: Repeat the previous weight and bias update step $T$ times, each time with a different mini-batch.

Note that the method for choosing each mini-batch of inputs is left to the user. A common choice in practice, and the one used in our numerical simulations, is to divide the training procedure into a number of periods known as epochs. At the beginning of each epoch, the training set is randomly shuffled and successive mini-batches of size $M$ are chosen without replacement. The epoch ends when all training examples have been selected, after which the next epoch begins. Alternatively, one may consider selecting the mini-batches by randomly sampling-with-replacement from the full training set.

The overall running time of the classical training algorithm is $O(TME)$ since there are $TM$ steps and each step requires performing algorithms to carry out the feedforward and backpropagation procedures. Note that the product $TM$ is equal to the number of epochs multiplied by the size $|\mathcal{T}|$ of the training set. The fact that the training of general feedforward neural networks depends linearly on the number of edges makes large-size fully connected feedforward networks computationally expensive to train. Our quantum algorithm reduces the dependence on the network size from the number of edges to the number of vertices.

Once the network training is complete, the classical evaluation algorithm consists of running the feedforward step on a previously unseen input (i.e., on data that lies outside the training set). The success of deep learning tells us that the network training produces weights and biases that perform well in classifying new data.

## 2.2 Robust Feedforward Neural Networks

A perennial concern in neural network training is the problem of overfitting. Since a large network may have many more free parameters than training data points, it is easy to train a network that recognizes the training data accurately yet performs poorly on data that lies outside the training set. Various techniques are used in practice to reduce overfitting. For instance, one may consider adding a term to the cost function that penalizes a network with too many large parameters. Alternatively, one can train the data on an ensemble of different networks and average the performance across the ensemble. Although this may be prohibitively costly to perform exactly, the effect can be approximated by randomly deactivating certain neurons in each layer, a technique known as dropout. A similar result can be achieved by adding random multiplicative Gaussian noise to the activation of each neuron during training. These techniques are both seen to be effective at preventing overfitting [42] and can also improve network performance by avoiding local minima in parameter space.

Motivated by the benefits of such random network perturbations during training, and anticipating the requirement of quantum processes to generate randomized outcomes, we consider a generalization of the training and evaluation algorithms where the inner products in the feedforward and backpropagation steps may not be evaluated exactly. Instead, with probability $1-\gamma$, they are estimated to within some error tolerance $\epsilon$, either relative or absolute, depending on whether the inner product is large or small. In other words, the feedforward step computes values $s_j^l$ satisfying

$$\left|s_j^l - \left\langle W_j^l, a^{l-1}\right\rangle\right| \leq \max\left\{\epsilon\left|\left\langle W_j^l, a^{l-1}\right\rangle\right|, \epsilon\right\} \quad \text{with probability } \geq 1-\gamma,$$

and similarly for the inner product calculation between $(W^{l+1})_j^T$ and $\delta^{l+1}$ in the backpropagation step. Note that we do not specify how the preceding $s_j^l$ are generated. The way this is realized will be left to specific implementations of the algorithm. For instance, a simple classical implementation is to first compute the inner product $\langle W_j^l, a^{l-1}\rangle$ and then add independent Gaussian noise bounded by the maximum of $\epsilon|\langle W_j^l, a^{l-1}\rangle|$ and $\epsilon$. In the quantum case, the $s_j^l$ will be generated by a quantum inner product procedure that is not perfect but rather outputs an estimate of the true inner product satisfying the $(\epsilon, \gamma)$ conditions required. The reason we allow for either a relative or absolute error is to ensure that the quantum procedure can be carried out efficiently regardless of the magnitude of the inner products involved.

We refer to such networks as $(\epsilon, \gamma)$-feedforward neural networks, of which the standard classical feedforward neural network is a special case. Our simulation results show that for reasonable tolerance parameters, the generalization to $(\epsilon, \gamma)$ estimates of inner products does not hurt network performance.

For small enough $\epsilon$, the running time of classically computing the feedforward and backpropagation steps remains $O(E)$, since in general one needs to look at a large fraction of the coordinates of two vectors to obtain an $\epsilon$-error approximation to their inner product. The classical training time for $(\epsilon, \gamma)$-feedforward neural networks is thus $O(TME)$, as for the original case where inner products are evaluated exactly.

## 2.3 Quantum Training

It is clearly desirable to improve on the $O(TME)$ classical result using a quantum algorithm; however, there are several obstacles to achieving this. As mentioned in Section 1, 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. Quantum algorithms, by contrast, are typically well suited to performing tasks in parallel but not for performing tasks that require sequential measurements to be performed, a process that destroys quantum coherence. In addition, a critical step classically is the application of a 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 $O(E)$.

We address these challenges by using a hybrid quantum-classical procedure that follows the classical training algorithm closely. In our algorithm, all sequential steps are taken classically. At each step, quantum operations are only invoked for estimating the inner products of vectors, and for reading and writing data to and from qRAM.

Given a vector $x \in \mathbb{R}^n$, define the corresponding normalized quantum state $|x\rangle = \frac{1}{\|x\|} \sum_{j=1}^{n} x_j |j\rangle$, where $\| \cdot \|$ is the $\ell_2$ norm. The inner product of two quantum states therefore satisfies $\langle x|y\rangle := \frac{\langle x, y\rangle}{\|x\|\|y\|}$. Two key ingredients are the following: RIPE and qRAM.

*Robust inner product estimation.* If quantum states $|x\rangle$ and $|y\rangle$ can each be created in time $T_U$, and if estimates of the norms $\|x\|$ and $\|y\|$ are known to within $\epsilon/3$ multiplicative error, then a generalization of the inner product estimation subroutine of Kerenidis et al. [21] allows one to perform the mapping $|x\rangle|y\rangle|0\rangle \to |x\rangle|y\rangle|s\rangle$ where, with probability at least $1 - \gamma$,

$$|s - \langle x, y\rangle| \leq \begin{cases} \epsilon\,|\langle x, y\rangle| & \text{in time } \widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon} \frac{\|x\|\|y\|}{|\langle x, y\rangle|}\right) \\ \epsilon & \text{in time } \widetilde{O}\left(\frac{T_U \log(1/\gamma)}{\epsilon} \|x\|\,\|y\|\right) \end{cases}$$

The preceding inner product estimates $s$ are computed in a quantum register and not in the phase of a quantum state, enabling non-linear activation functions to be applied to it. When the data required is stored in qRAM (see the following), this inner product calculation is efficient and provides roughly a factor $O(n_l)$ saving in time per layer of the network.

*Quantum random access memory.* For the RIPE algorithm to efficiently compute an approximation of the inner product $\langle x, y\rangle$, we need an efficient way to prepare the states $|x\rangle$ and $|y\rangle$. A classical RAM is a device that takes an address $j$ and returns the data $x_j$ stored at that location. Analogously, a qRAM is a device that allows for such classical data to be queried in superposition. In other words, if the classical vector $x \in \mathbb{R}^N$ is stored in qRAM, then a query to the qRAM implements the unitary $\sum_j \alpha_j |j\rangle|0\rangle \to \sum_j \alpha_j |j\rangle|x_j\rangle$. Importantly, if the elements $x_j$ of $x$ arrive as a stream of entries $(j, x_j)$ in some arbitrary order, then $x$ can be stored in a particular data structure [23]—which we will refer to as an $\ell_2$-binary search tree ($\ell_2$-BST)—in a time linear in $N$ (up to logarithmic factors), and once stored, $|x\rangle = \frac{1}{\|x\|} \sum_j x_j |j\rangle$ can be created in time polylogarithmic in $N$ (Figure 3). In Section 3, we will comment on the feasibility of implementing qRAM in practice.

Conceptually, we would like to follow the classical training algorithm, modified so that the network biases $b^{t,l}$, weights $W^{t,l}$, pre-activations $z^{t,m,l}$, activations $a^{t,m,l}$, and backpropagation vectors $\delta^{t,m,l}$ are stored in a qRAM $\ell_2$-BST at every step. Their corresponding quantum states can then be efficiently created and the RIPE algorithm used to estimate the inner products required to update the network. However, there is a problem. The weight matrices $W^{t,1}, W^{t,2}, \ldots, W^{t,L}$ have a combined total of $E$ entries. Thus, it will take time $\tilde{O}(E)$ just to write all of the matrix elements into qRAM. This is true even for the initial weight matrices if they are generated by choosing independent random variables for each element, as is common classically. Each time the weights are updated, it will take an additional $\tilde{O}(E)$ time to write the new values to qRAM. Furthermore, the RIPE algorithm requires estimates of the norms of the vectors involved. This is not a problem for the $a^{t,m,l}$ and $\delta^{t,m,l}$ vectors, as their storage in the $\ell_2$-BST allows their norms to be accessed. However, if the weight matrices are not stored in this way, then the norms of their rows and columns are also not available to use explicitly.

We circumvent these issues in two ways: low-rank initialization and implicit storage of weight matrices.

*Low-rank initialization.* We set the initial weights $W^{1,l}$ to be low rank by selecting a small number $r$ of pairs of random vectors $a^{0,m,l}$ and $\delta^{0,m,l}$ $(m = 1, \ldots, r)$ and taking the sums of their outer

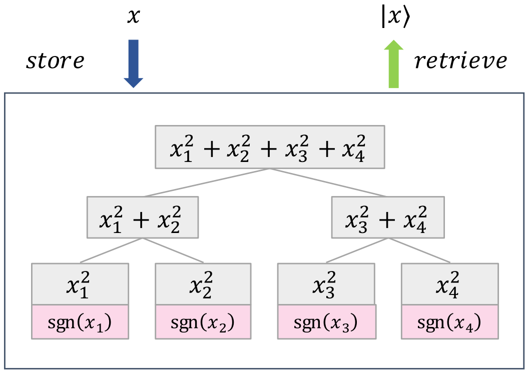


Fig. 3. $\ell_2$-BST data structure. A classical vector $x \in \mathbb{R}^{\mathbb{N}}$ is stored in a binary tree, with leaves containing the squared vector components $x_i^2$ and the signs of $x_i$. The tree is populated recursively, with each parent storing the sum of the values of its children. The root of the tree thus stores $\|x\|$. As shown in the work of Kerenidis and Prakash [23], it takes time $\tilde{O}(N)$ to store $x$ in the date structure. Once stored, qRAM access to the data structure allows the quantum state $|x\rangle = \frac{1}{\|x\|} \sum_j x_j |j\rangle$ to be generated in time polylogarithmic in $N$.

products. If we define $\eta^{0,1} = -1$, then Equation (2) for the weights can be expressed as

$$W_{jk}^{t,l} = \sum_{\tau=0}^{t-1} \sum_{m=1}^{M} \frac{-\eta^{\tau,l}}{M} \delta_j^{\tau,m,l} a_k^{\tau,m,l-1},$$

where for a fixed $t$ and $l$, only $r$ of the $M$ possible $\delta^{0,m,l}$ and $\delta^{0,m,l}$ are non-zero. We shall see that this low-rank initialization does not affect the network performance in practice, and in Section 3 we provide a classical justification for this initialization and numerical evidence supporting the use of $r = O(\log n_l)$ random pairs.

*Implicit storage of weight matrices.* For each $t \in [T], l \in \{2, \dots L\}, j \in [n_L]$, define the matrix $X^{[t,l,j]} \in \mathbb{R}^{t \times M}$ with matrix elements $(X^{[t,l,j]})_{\tau m} = \frac{-\eta^{\tau,l}}{M} \delta_j^{\tau,m,l} \|a^{\tau,m,l-1}\|$, with $\tau \in \{0, \dots t-1\}$, $m \in [M]$. We will store all matrix elements of each $X^{[t,l,j]}$ in qRAM (Figure 4), which allows for efficient computation of the states $|W_j^{t,l}\rangle$ on the fly, as opposed to explictly storing all values $W_{j,k}^{t,l}$, which is prohibitively expensive. More specifically, in Section 4, we show that by querying the rows of each matrix in superposition over iterations $\tau < t$, it is possible to generate the weight states $|W_j^{t,l}\rangle$ in time $T_W = \tilde{O}(\frac{\|X^{[t,l,j]}\|_F}{\|W_j^{t,l}\|} \sqrt{TM})$, and estimate $\|W_j^{t,l}\|$ to multiplicative error $\xi$ in time $O(T_W/\xi)$, respectively, where $||X^{[t,l,j]}||_F$ is the Frobenius norm $X^{[t,l,j]}$. Similar results apply to generating the states $|(W^l)_j^T\rangle$ corresponding to the columns of the weight matrices and estimating their norms.

With these ideas, one can define a quantum $(\epsilon, \gamma)$-feedforward algorithm that adapts the classical one to make use of RIPE, qRAM access, and the implicit storage of the weight matrices. This is shown in Algorithm 1.

The cost of performing this feedforward procedure is $O(\sum_{l=2}^{L} \sum_{j=1}^{n_l} T_{RIPE}(W_j^{t,l}, a^{t,m,l-1})) = O(N\overline{T}_{RIPE})$, where $T_{RIPE}(W_j^{t,l}, a^{t,m,l-1})$ is the time required to perform RIPE 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. Using the running time of RIPE that achieves the larger of the

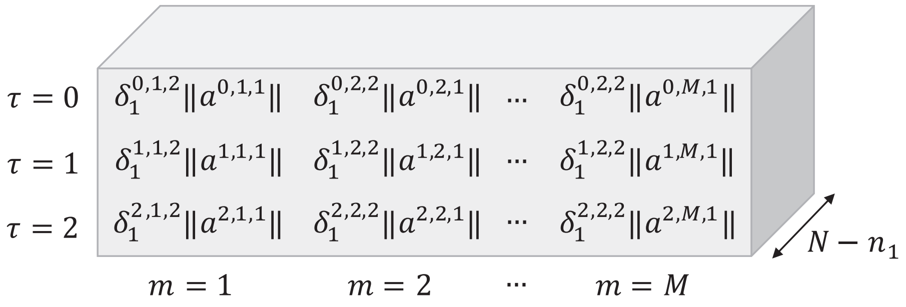

$\tau = 0$
$\tau = 1$
$\tau = 2$
$\delta_1^{0,1,2}\|a^{0,1,1}\|$ $\delta_1^{0,2,2}\|a^{0,2,1}\|$ $\cdots$ $\delta_1^{0,2,2}\|a^{0,M,1}\|$
$\delta_1^{1,1,2}\|a^{1,1,1}\|$ $\delta_1^{1,2,2}\|a^{1,2,1}\|$ $\cdots$ $\delta_1^{1,2,2}\|a^{1,M,1}\|$
$\delta_1^{2,1,2}\|a^{2,1,1}\|$ $\delta_1^{2,2,2}\|a^{2,2,1}\|$ $\cdots$ $\delta_1^{2,2,2}\|a^{2,M,1}\|$
$N - n_1$
$m = 1$ $m = 2$ $\cdots$ $m = M$

Fig. 4. Implicit storage of weight matrices via the matrices $X^{[t,l,j]}$. At any iteration $t$ there are $\sum_{l=2}^{L} n_l = N - n_1$ such matrices, which are stored in the $\ell_2$-BST data structure of Figure 3. Shown here are the matrix elements of $X^{[3,2,1]}$, with the factors of $-\eta^{\tau,l}/M$ omitted for clarity of presentation. The other matrices corresponding to different $l, j$ values are indicated behind. Storing these matrices in an $\ell_2$-BST allows for efficient creation of the states $|W_j^l\rangle$ and estimation of their norms. After each iteration of training another row of data is added to each of $N - n_l$ matrices, and the data structure is updated. The maximum size of data stored occurs after $T$ iterations and corresponds to $N - n_1$ matrices, each of size $TM$. Any element of these matrices can thus be coherently accessed by qRAM in time polylog($TMN$).

**ALGORITHM 1:** Quantum $(\epsilon, \gamma)$-Feedforward

**Input:** 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
    $a_j^{t,m,1} = x_j^{t,m}$
end for

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 with probability $\geq$
        $1 - \gamma$,
            $$\left|s_j^{t,m,l} - \left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle\right| \leq \max\left\{\epsilon\left|\left\langle W_j^{t,l}, a^{t,m,l-1}\right\rangle\right|, \epsilon\right\}$$
        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
    end for
end for
```

absolute or relative estimation errors, one obtains

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

where $T_U$ is the time required to prepare state $|W_j^{t,l}\rangle$ and $|a^{t,m,l-1}\rangle$. $|a^{t,m,l-1}\rangle$ can be created in time polylogarithmic in $N$ since the required data is, by assumption, stored in qRAM. By the implicit storage of weight matrices, $|W_j^{t,l}\rangle$ can be created in time $\tilde{O}(\frac{\|X^{[t,l,j]}\|_F}{\|W_j^{t,l}\|}\sqrt{TM})$. The overall running time of the quantum feedforward algorithm is therefore

$$\tilde{O}\left(\sqrt{TM}N\frac{\log(1/\gamma)}{\epsilon}R_a^{t,m}\right),$$

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

The factor $R_a^{t,m}$ does not appear in the classical algorithms, and although it is *a priori* not clear what impact this will have on the running time, we give evidence in Section 3 that this does not impact the running time significantly in practice. The upside is that we save a factor of $O(N)$ compared with the classical case, since our algorithm depends linearly on $N$ and not on the number of edges $E$. Last, there is an overhead of $\sqrt{TM}$ that is a consequence of only saving the weight matrices implicitly. For large neural networks, one expects that $N >> \sqrt{TM}$.

We can similarly define a quantum backpropagation algorithm (Algorithm 2). Similar to the feedforward case, 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 $R_\delta^{t,m} = \frac{1}{N-n_l}\sum_{l=1}^{L-1}\sum_{j=1}^{n_l}\frac{\|\tilde{X}^{[t,l+1,j]}\|_F\|\delta^{t,m,l+1}\|}{\max\{1,|\langle (W^{t,l+1})_j^T,\delta^{t,m,l+1}\rangle|\}}$.

**ALGORITHM 2:** Quantum $(\epsilon,\gamma)$-Backpropagation

**Input:** 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 $\overline{\left\|(W^{t,l+1})^j\right\|}$ and $\overline{\left\|\delta^{t,m,l+1}\right\|}$ 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 $\epsilon, \gamma > 0$.

```
for $j = 1$ to $n_L$ do
    $\delta_j^{t,m,L} = f'(z_j^{t,m,L})\frac{\partial C}{\partial a_j^L}$
end for

for $l = L-1$ to 1 do
    for $j = 1$ to $n_l$ do
        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 with probability $\geq 1-\gamma$,
            $\left|s_j^{t,m,l} - \left\langle \left(W^{t,l+1}\right)^j, \delta^{t,m,l+1}\right\rangle\right| \leq \max\left\{\epsilon\left|\left\langle\left(W^{t,l+1}\right)^j, \delta^{t,m,l+1}\right\rangle\right|, \epsilon\right\}$
        Compute $\delta_j^{t,m,l} = f'(z_j^{t,m,l})s_j^{t,m,l}$
        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.
    end for
end for
```

The quantum training algorithm (Algorithm 3) consists of running the feedforward and the backpropagation algorithms for all inputs in a mini-batch of size $M$ and iterating this procedure $T$ times. 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 $\tilde{O}(E)$. 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 $\delta$ vectors in memory so that we can create the quantum states corresponding to the weights on the fly.

**ALGORITHM 3:** Quantum $(\epsilon, \gamma)$-Training

```
Input: 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 $\epsilon, \gamma > 0$.

Initialize the weights and biases $W^{1,l}, b^{1,l}$ for $l \in [L]$ with a low-rank initialization.

for $t = 1$ to $T$ do
  for $m = 1$ to $M$ do
    Run the quantum $(\epsilon, \gamma)$-feedfoward algorithm.
    Run the quantum $(\epsilon, \gamma)$-backpropagation algorithm.
    Compute the biases and update the qRAM with
      $b_j^{t+1,l} = b_j^{t,l} - \eta^{t,l} \frac{1}{M} \sum_m \delta_j^{t,m,l}$
    Compute the estimates of the norms $\overline{\|W_j^{t+1,l}\|}$ and $\overline{\|(W^{t+1,l})^j\|}$ with relative error $\xi = \epsilon/3$.
  end for
end for
```

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

Proof. The terms $R_a$ and $R_\delta$ come from the feedforward and backpropagation algorithms. The $R_W$ term comes from the estimation of the norms $\|W_j^{t,l}\|$ and $\|(W^{t,l})_j^T\|$, which only happens once for each mini-batch. For a given $t, l, j$, the estimation of the norm $\overline{\|W_j^{t+1,l}\|}$ takes time $O(T_W/\xi)$, with $T_W = \tilde{O}(\frac{\|X^{[t,l,j]}\|_F}{\|W_j^{t,l}\|} \sqrt{TM})$, and we take $\xi = \epsilon/3$. Hence, we get the ratio $R_{W_r}^t = \frac{1}{M} \frac{1}{N-n_1} \sum_{l=2}^L \sum_{j=1}^{n_l} \frac{\|X^{[t,l,j]}\|_F}{\|W_j^{t,l}\|}$ and similarly for the estimation of the columns $R_{W_c}^t = \frac{1}{M} \frac{1}{N-n_1} \sum_{l=2}^L \sum_{j=1}^{n_l} \frac{\|\tilde{X}^{[t,l,j]}\|_F}{\|(W^{t,l})_j^T\|}$. Then we can define $R_W = \frac{1}{T} \sum_t (R_{W_r}^t + R_{W_c}^t)$ as needed. □

The tradeoff in avoiding an $O(E)$ scaling is an extra cost factor of $O(\sqrt{TM})$, which makes the overall running time of our quantum training algorithm essentially $\tilde{O}((TM)^{1.5} N)$ and, although the exact running time also depends on various other factors, loosely speaking has an advantage over the classical training when $\sqrt{TM} \ll N$. Note that up to now, we have assumed that the input pairs $(x^{t,m}, y^{t,m})$ are stored in qRAM for all $t \in [T], m \in [M]$. If this is not the case, then as each $x^{t,m}$ has dimension $n_1 \leq N$ and each $y^{t,m}$ has dimension $n_L < N$, in the worst case one must spend an additional time $\tilde{O}(TMN)$ to store all of the required data in qRAM, which is subsumed by the $(TM)^{1.5} N$ term in the running time in Theorem 1. Furthermore, in the event that data is received sequentially (i.e., as a stream), it takes time $O(TMN)$ just to receive all of the data. In this case, if the data is stored in qRAM as it is received, then up to polylogarithmic factors, no additional time is lost in this storage phase of the algorithm.

## 2.4 Quantum Evaluation

Once a neural network has been trained, it can be evaluated on new data to output a predicted label. Although the initial training may only occur once, evaluation of new data labels may occur thousands or millions of times thereafter. There are thus large gains to be had from even small

Table 1. $(\epsilon, \gamma)$-Feedforward Neural Network Accuracy (Percentage of Correctly Labeled Test Points) on the MNIST Handwriting Dataset for Various Values of $\epsilon$, and $\gamma = 0.05$

| $\epsilon$ | 0 | 0.1 | 0.3 | 0.5 |
|---|---|---|---|---|
| Standard | 96.9% | 97.1% | 96.9% | 96.2% |
| Low rank | — | 96.8% | 96.8% | 96.3% |
| $\log(1/\gamma)/\epsilon$ | — | 30 | 10 | 6 |
| $R_a$ | — | 24.4 | 21.9 | 21.3 |
| $R_\delta$ | — | 0.6 | 0.1 | 1.3 |
| $R_W$ | — | 0.1 | 0.1 | 0.1 |

*Note*: A network with $L = 4$ layers and dimensions $[n_1, n_2, n_3, n_4] = [784, 100, 30, 10]$ was used, with $M = 100$, $T = 7{,}500$, $\eta = 0.05$, tanh activation function, and mean squared error cost function.

improvements to the efficiency of network evaluation, and we realize such an improvement with our second quantum algorithm.

The quantum procedure for neural network evaluation is essentially the same as the quantum feedforward algorithm. Assume that 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 $|W_j^l\rangle$ and $|a^{l-1}\rangle$ in time $T_U$, estimates of their norms $\overline{\|W_j^l\|}$ and $\overline{\|a^{l-1}\|}$ to relative error at most $\xi = \epsilon/3$, and vectors $b^l$ in qRAM for $l \in [L]$. The running time of the quantum feedforward algorithm implies the following.

Theorem 2. *There exists a quantum algorithm for evaluating an $(\epsilon, \gamma)$-feedforward neural network in time $\tilde{O}(T_U N \frac{\log(1/\gamma)}{\epsilon} R_e)$, where $R_e = \frac{1}{N-n_1} \sum_{l=2}^{L} \sum_{j=1}^{n_l} \frac{\|W_j^l\| \|a^{l-1}\|}{\max\{1, |\langle W_j^l, a^{l-1}\rangle|\}}$ and $T_U$ is the time required to prepare any of the states $|W_j^l\rangle$ and $|a^{l-1}\rangle$. For a neural network whose weights are already explictly stored in an $\ell_2$-BST, $T_U$ is polylogarithmic in E, and the total running time is $\tilde{O}(N \frac{\log(1/\gamma)}{\epsilon} R_e)$.*

In contrast, the classical evaluation algorithm requires running time $O(E)$. 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 $\tilde{O}(\sqrt{TM} N \frac{\log(1/\gamma)}{\epsilon} R_e)$.

## 2.5 Simulation

In this section, we show that $(\epsilon, \gamma)$-networks can achieve comparable results to standard neural networks and give numerical evidence that the hard-to-estimate parameters $R_a$, $R_\delta$, and $R_W$ may not be large for problems of practical interest. We give further arguments to support this in Section 3.

We classically simulate the training and evaluation of $(\epsilon, \gamma)$-feedforward neural networks on the MNIST handwritten digits dataset 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 $C = \frac{1}{2M} \sum_m \|y^{t,m} - a^{t,m,L}\|^2$. For this network size, we have $N = 924$, $E = 81, 700$. Our results are summarized in Table 1. "Standard" weight initialization refers to drawing the initial weight matrix elements from appropriately normalized Gaussian distributions $W_{jk}^l \sim \mathcal{N}(0, \frac{1}{\sqrt{n_{l-1}}})$ [17], and "Low rank" refers to weight initialization as described in Section 4.2, with rank $r = 6$. In all cases, Gaussian noise drawn from $N(0, \epsilon/2)$ was added to each inner product evaluated throughout the network. Note that no cost function regularization was used to improve the network performance,

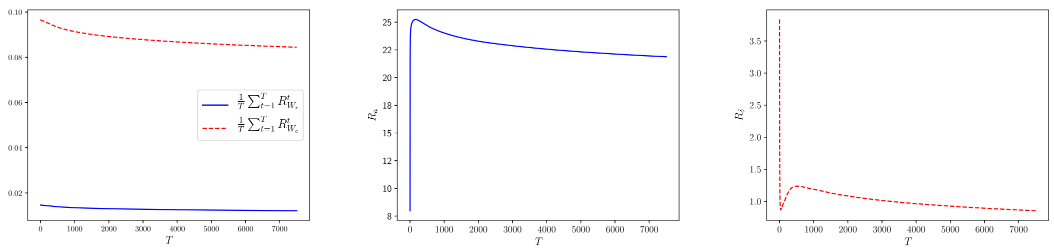


Fig. 5. Evolution of the terms $R_W$, $R_a$, and $R_\delta$ as a function of the number of iterations $T$ for the $(\epsilon, \gamma) = (0.3, 0.05)$ network of Table 1 (MNIST dataset).

and as $\epsilon$ was varied, the other network hyper-parameters were not re-optimized. We find that for $\gamma = 0.05$ and various values of $\epsilon$, the network achieves high accuracy while incurring only modest contributions to the running time from the quantum-related terms $R_a, R_\delta$, and $R_W$. For the $\epsilon = 0.3$ case, we calculate the values of $R_a$, $R_\delta$ and the two components of $R_W$, as a function of the number of gradient update steps $t$. These results appear in Figure 5, where we see that these values quickly stabilize to small constants during the training procedure. Note that the percentage accuracy obtained in these experiments is significantly outperformed by state-of-the-art classical methods based on more complex network architectures. For instance, Wan et al. [48] achieved an accuracy of 99.79% by using multiple independent convolutional neural networks and utilizing data augmentation and decaying learning rates. An interesting avenue for future research would be to see what advantages our methods could bring to more sophisticated networks, and how the inner product estimations involved would impact the generalization error and accuracy in these cases.

The MNIST dataset is evaluated using Gaussian distributed noise; however, we also numerically evaluate the performance of $(\epsilon, \gamma)$-feedforward networks on the Iris flower dataset using both Gaussian noise and, for comparison, noise corresponding to the quantum RIPE subroutine where the norms of all vectors are known exactly (see the appendix). This dataset—consisting of 120 training examples and 30 test examples, with each data point corresponding to a length 4 real vector and having one of three labels—is small enough that classically sampling from the RIPE distribution (which corresponds to simulating the output of a quantum circuit) during the network training can be performed efficiently on a standard desktop computer. A network with $L = 3$ layers, dimension $[n_1, n_2, n_3] = [4, 10, 3]$, tanh activation function, and mean squared error cost function was chosen, with $M = 10$, $T = 1200$, and $\eta = 0.07$. For each $(\epsilon, \gamma)$ pair, we repeated the network training and evaluation five times and show the average number of correctly labeled test points in Table 2. For $\gamma = 0.05$ and the same choices of $\epsilon$ as in the MNIST case, we find that the choice of noise distribution, Gaussian or RIPE, does not lead to significant difference in network performance.

## 2.6 Quantum-Inspired Classical Algorithms

Our quantum training and evaluation algorithms rely on the $\ell_2$-BST data structure to efficiently estimate inner products via the RIPE procedure. By the following result of Tang [45], such a data structure in fact allows inner products to be efficiently estimated classically as well.

*Classical $\ell_2$-BST-based inner product estimation* [45]. If $x, y \in \mathbb{R}^n$ are stored in $\ell_2$-BSTs, then with probability at least $1 - \gamma$, a value $s$ can be computed satisfying

$$|s - \langle x, y\rangle| \leq \begin{cases} \epsilon\, |\langle x, y\rangle| & \text{in time } \widetilde{O}\left(\frac{\log(1/\gamma)}{\epsilon^2} \frac{\|x\|^2 \|y\|^2}{|\langle x, y\rangle|}\right) \\ \epsilon & \text{in time } \widetilde{O}\left(\frac{\log(1/\gamma)}{\epsilon^2} \|x\|^2 \|y\|^2\right) \end{cases}.$$

Table 2. Average Number of Correctly Labeled Test Points (Out of a Total of 30) for the Iris Flower Dataset, for Gaussian Distributed Noise and Noise from the RIPE Distribution

| $\epsilon$ | 0 | 0.1 | 0.3 | 0.5 |
|---|---|---|---|---|
| Gaussian | 28.6 | 29.8 | 29.2 | 29.0 |
| RIPE | — | 28.6 | 28.4 | 28.0 |
| $\log(1/\gamma)/\epsilon$ | — | 30 | 10 | 6 |
| $R_a$ | — | 6.0 | 4.4 | 4.0 |
| $R_\delta$ | — | 0.7 | 0.6 | 0.6 |
| $R_W$ | — | 0.4 | 0.4 | 0.4 |

*Note*: For the case of RIPE, the $R_a$, $R_\delta$, and $R_W$ terms presented are also averaged over five network evaluations. $(\epsilon, \gamma)$-Feedforward neural network accuracy (percentage of correctly labeled test points) was trained on the Iris dataset for various values of $\epsilon$, and $\gamma = 0.05$. A network with $L = 3$ layers and dimensions $[n_1, n_2, n_3] = [4, 10, 3]$ was used, with $M = 10$, $T = 1{,}200$, $\eta = 0.07$, tanh activation function, and mean squared error cost function.

We can use this concept to derive quantum-inspired classical analogues of our algorithms. If the network weights are explicitly stored in $\ell_2$-BSTs, then the quantum RIPE procedure can be directly replaced by this classical inner product estimation routine to give a classical algorithm for network evaluation that has running time

where $R_e^{cl} = \frac{1}{N-n_1}\sum_{l=2}^{L}\sum_{j=1}^{n_l}\frac{\|W_j^l\|^2\|a^{l-1}\|^2}{\max\{1, |\langle W_j^l, a^{l-1}\rangle|^2\}} \geq R_e^2$.

Dequantizing the quantum training algorithm is slightly more complicated as the network weights are only stored implicitly. Nonetheless, the required inner products can still be estimated efficiently, and in Section 4 we show a quantum-inspired algorithm can be given that runs in time

$$\tilde{O}\left((TM)^2 N\frac{\log(1/\gamma)}{\epsilon^2}\left(R_a^{cl} + R_\delta^{cl}\right)\right),$$

where $R_a^{cl} \geq R_a^2$ and $R_\delta^{cl} \geq R_\delta^2$. The quantum-inspired algorithm is thus slower than the quantum one by a factor of $\frac{\sqrt{TM}}{\epsilon}\frac{R_a^2+R_\delta^2}{(R_a+R_\delta+R_W)}$. We further analyze their relative performance next.

## 3 DISCUSSION

We have presented quantum algorithms for training and evaluating $(\epsilon, \gamma)$-feedforward neural networks that take time $\tilde{O}((TM)^{1.5}N\frac{\log(1/\gamma)}{\epsilon}R)$ and $\tilde{O}(NT_U\frac{\log(1/\gamma)}{\epsilon}R_e)$, respectively. These are the first algorithms based on the canonical classical feedforward and backpropagation procedures with running times better than the number of inter-neuron connections, opening the way to utilizing larger-sized neural networks. Furthermore, our algorithms can be used as the basis for quantum-inspired classical algorithms that have the same dependence on the network dimensions but a quadratic penalty in other parameters. Let us now make several remarks on our design choices and the performance of our algorithms.

*Performance of $(\epsilon, \gamma)$-feedforward neural networks.* Although the notion of introducing noise in the inner product calculations of an $(\epsilon, \gamma)$-feedforward neural network is, as mentioned previously, similar in spirit to known classical techniques for improving neural network performance, there are certain differences. Classically, dropout and multiplicative Gaussian noise [42] 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); however, 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. Yet numerical simulation (see Tables 1 and 2) of the noise model present in our quantum algorithm shows that $(\epsilon, \gamma)$-networks may tolerate modest values of noise, and values of $\epsilon$ and $\gamma$ can be chosen for which the network performance does not suffer significantly, but 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 to avoid a time of $\tilde{O}(E)$ 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 [8, 37, 53], as well as in network training [43], 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 $\epsilon$ values (see Table 1).

*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 $R = R_a + R_\delta + R_W$ that are not featured in the classical algorithm. Although the $\epsilon$ and $\gamma$ can be viewed as hyper-parameters, which can be freely chosen, the impact of the $R$ terms warrant further discussion.

The ratio $||X^{[t,l,j]}||_F/||W_j^{t,l}||$ appears in both $R_a$ and $R_W$ (in the $R_{W_r}^t$ contribution), and similarly the ratio $||\tilde{X}^{[t,l,j]}||_F/||(W^{t,l})_j^T||$ appears in $R_\delta$ and the $R_{W_r}^t$ contribution to $R_W$. Although exact values may be difficult to predict, one can expect the following large $t$ behavior: 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 $||W_j^{1,l}|| \approx 1$, 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 $\eta$ one expects $||W_j^{t,l}||$ to roughly grow proportionally to $\sqrt{t\eta}$. The norm $||X^{[t,l,j]}||_F$ 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, $||a^{t,l-1}|| \leq \sqrt{n_{l-1}}$. We thus expect $||X^{[t,l,j]}||_F$ to saturate for large $t$ and not grow in an unbounded fashion. Figure 5(a) showing the time averaged values of $R_{W_r}^t$ and $R_{W_c}^t$ for an $(\epsilon, \gamma)$-network trained on the MNIST dataset is consistent with these ratios saturating to very small values over time—in fact, values less than 0.014. A similar result can be seen in Figure 6(a) for the Iris dataset.

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

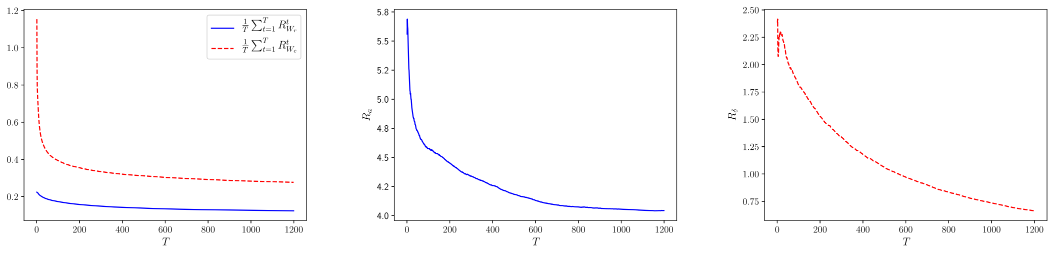


Fig. 6. Evolution of the terms $R_W$, $R_a$, and $R_\delta$ as a function of the number of iterations $T$ for one typical instance of an $(\epsilon, \gamma) = (0.3, 0.05)$ network of Table 2 (Iris dataset).

Table 3. Comparison of Running Times Between the Quantum, Quantum-Inspired, and Standard Classical Algorithms

| | Training | Evaluation |
|---|---|---|
| Quantum | $\tilde{O}\left((TM)^{1.5} N \frac{\log(1/\gamma)}{\epsilon} (R_a + R_\delta + R_W)\right)$ | $\tilde{O}\left(N \frac{\log(1/\gamma)}{\epsilon} R_e\right)$ |
| Quantum Inspired | $\tilde{O}\left((TM)^{2} N \frac{\log(1/\gamma)}{\epsilon^2} \left(R_a^{cl} + R_\delta^{cl}\right)\right)$ | $\tilde{O}\left(N \frac{\log(1/\gamma)}{\epsilon^2} R_e^{cl}\right)$ |
| Standard Classical | $O(TME)$ | $O(E)$ |

*Note*: $R_a^{cl} \geq R_a^2$, $R_\delta^{cl} \geq R_\delta^2$, and $R_e^{cl} \geq R_e^2$.

and vector norms:

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

As discussed, we expect $||X^{[t,l,j]}||_F$ to saturate for large $t$, and for activations in $[-1, 1]$ we have $||a^{t,m,l-1}|| \leq \sqrt{n_{l-1}}$. However, as the network becomes well trained, one expects the inner products $\langle W_j^{t,l}, a^{t,m,l-1} \rangle$ to become large in magnitude so that the neurons have post-activation values close to $\pm 1$. It is thus reasonable to expect the $R_a$ to saturate or even decline for large $t$. This is consistent with our results in Figures 5(b) and 6(b). One expects similar large $t$ behavior for $R_\delta$, except intuitively $R_\delta$ should be much smaller than $R_a$ since $||\delta^{t,m,l}||$ should become very small as the network becomes well trained. Figures 5(c) and 6(c) display this expected behavior. Although these simulation results are already promising, we expect the quantum advantage in the running time to become more prominent for larger-sized neural networks.

*Quantum evaluation running time.* The classical running time of the evaluation algorithm is $O(E)$. In contrast, our quantum evaluation algorithm runs in time $\tilde{O}(N \frac{\log(1/\gamma)}{\epsilon} R_e)$ if the entries of the weight matrices are explicitly stored in qRAM. By the same arguments given previously for the training algorithm, we expect the penalty term $R_e$ to be small for well-trained networks, and in this case we expect the quantum training algorithm to be able to provide a significant speedup over its classical counterpart.

*Quantum vs. quantum-inspired running times.* Although both the quantum and quantum-inspired classical algorithms presented here have a running time linear in $N$, the quantum-inspired algorithms come with a quadratic overhead in other parameters. A comparison of their running times is given in Table 3. The standard classical training time of $TME$ is outperformed by our quantum training algorithm when $\sqrt{TM} \frac{N}{E} \frac{\log(1/\gamma)}{\epsilon} (R_a + R_\delta + R_W) \ll 1$, whereas the quantum-inspired classical training algorithms can only outperform the standard classical algorithm when $TM \frac{N}{E} \frac{\log(1/\gamma)}{\epsilon^2} (R_a^{cl} + R_\delta^{cl}) \ll 1$. To understand the significance of the quadratic overheads required by the quantum-inspired algorithms, consider a back-of-the-envelope calculation based on the landmark neural network of Krizhevsky et al. [26]. The convolutional neural network they use to recognize the ImageNet LSVRC dataset has $TM \approx 10^8$ and fully connected final layers corresponding to $N \approx 2 \times 10^4, E \approx 6 \times 10^7$. Taking $\epsilon = 0.1, \gamma = 0.05$ and assuming values of $R_a, R_\delta$, and $R_W$ the same order of magnitude as we numerically evaluated for the MNIST case gives $\sqrt{TM}\frac{N}{E}\frac{\log(1/\gamma)}{\epsilon}(R_a + R_\delta + R_W) \approx 10^3$ and $TM\frac{N}{E}\frac{\log(1/\gamma)}{\epsilon^2}(R_a^{cl} + R_\delta^{cl}) \approx 1.4 \times 10^9$, a factor of $10^6$ in favor of the quantum algorithm. Although neither the quantum nor the quantum-inspired classical algorithms can compete with the standard classical algorithm in this particular case, plausible changes to the network and dataset parameters can be chosen where the quantum training algorithm has an advantage. The quantum-inspired algorithm, however, would require network parameters many orders of magnitude different from the practical example considered here. In fact, it remains to be seen if there are any real cases where quantum-inspired algorithms can be better in practice than the standard classical ones, and the evidence so far is negative [1], although more benchmarking and careful comparison with state-of-the-art classical algorithms is needed.

*Practicality of qRAM.* Our proposed quantum algorithms are based on access to a working qRAM. Although several advances [2, 9, 19, 31] have been made since the original qRAM proposal [15, 16], the theory and experiment of qRAM remain in relative infancy. From an engineering standpoint, developing a fully error-corrected qRAM is no more difficult than the (considerable) challenge of developing a fault-tolerant quantum computer. However, analysis [9] of the resources required to implement such a qRAM with a large storage size suggests time-space tradeoffs that are tremendously expensive, at least for current qRAM proposals. Yet further advances in qRAM design and quantum error correction may bring the required resources down, and understanding the conditions under which qRAM-based algorithms can give practical advantages over classical methods is an open area of research.

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.

## 4 METHODS

### 4.1 RIPE Running Time

Kerenidis et al. [21] give a quantum inner product estimation (IPE) algorithm that allows one to perform the mapping $|x\rangle|y\rangle|0\rangle \to |x\rangle|y\rangle|s\rangle$, where $s$ satisfies $|s - \langle x, y\rangle| \leq \epsilon$ with probability at least $1 - \gamma$. The running time is $\widetilde{O}(\frac{T_U \log(1/\gamma)}{\epsilon}\|x\|\|y\|)$, where $T_U$ is the time require to implement unitary operations for creating states $|x\rangle, |y\rangle, |\|x\|\rangle, |\|y\|\rangle$.

The RIPE algorithm is a generalization to the case where one only has access to estimates $\overline{\|x\|}, \overline{\|y\|}$ of the vector norms, satisfying $|\overline{\|x\|} - \|x\|| \leq \frac{\epsilon}{3}\|x\|$ and $|\overline{\|y\|} - \|y\|| \leq \frac{\epsilon}{3}\|y\|$ and where one would like to obtain inner product estimates to either additive or multiplicative error. For normalized vectors $|x\rangle, |y\rangle$, the IPE algorithm runs in time $\widetilde{O}(\frac{T_U \log(1/\gamma)}{\epsilon'})$ and outputs $|s - \langle x|y\rangle| \leq \epsilon'$. By taking $\epsilon' = \frac{\epsilon}{4}|\langle x|y\rangle|$, we obtain a relative error algorithm in time $\widetilde{O}(\frac{T_U \log(1/\gamma)}{\epsilon}\frac{1}{|\langle x|y\rangle|})$. Outputting the estimator $s' = \overline{\|x\|}\,\overline{\|y\|}s$ then satisfies

$$\begin{aligned} |s' - \langle x, y\rangle| &\leq |s' - \|x\|\,\|y\|\,s| + |\|x\|\,\|y\|\,s - \|x\|\,\|y\|\,\langle x, y\rangle| \\ &\leq [(1 + \epsilon/3)^2 - 1]\,\|x\|\,\|y\|\,(1 + \epsilon/4)\langle x|y\rangle + \epsilon/4\langle x, y\rangle \\ &\leq \epsilon\,|\langle x, y\rangle| \end{aligned}$$

for small enough $\epsilon$. An absolute error estimate can similarly be obtained by taking replacing $\epsilon$ above with $\epsilon/|\langle x,y\rangle|$. We thus have an algorithm that runs in time $T_{\mathrm{IPE}}(x,y) = \widetilde{O}(\frac{T_U \log(1/\gamma)}{\epsilon} \frac{\|x\|\|y\|}{\max\{1,|\langle x,y\rangle|\}})$ and achieves an error of $|s - \langle x,y\rangle| \le \max\{\epsilon|\langle x,y\rangle|, \epsilon\}$.

## 4.2 Constructing the Weight Matrix States and Estimating Their Norms

If, at each iteration $t$, the values $a_j^{t,m,l}$ and $-\frac{\eta^{t,l}}{M}\delta_j^{t,m,l}\|a^{t,m,l-1}\|$ are stored in qRAM for all $m \in [M]$, then the states $|a^{t,m,l-1}\rangle$ and

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

can be created coherently in time polylogarithmic in $TMN$. In other words, unitary operators $U_X$ and $U_a$ can be implemented in this time that affect the transformations:

$$U_X |t\rangle |l\rangle |j\rangle |0\rangle |0\rangle \to |t\rangle |l\rangle |j\rangle \left|X^{[t,l,j]}\right\rangle$$
$$U_a |l\rangle |t\rangle |m\rangle |0\rangle \to |l\rangle |t\rangle |m\rangle \left|a^{t,m,l-1}\right\rangle .$$

Each of the kets $|\cdot\rangle$ in an expression such as $|l\rangle|t\rangle|m\rangle|0\rangle$ is referred to as a register. Application of $U_X$ on the first five registers of state $|t\rangle|l\rangle|j\rangle|0\rangle|0\rangle|0\rangle$, followed by application of $U_a$ on registers 2, 4, 5, 6 on the resulting state, gives

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

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

$$\begin{aligned} &|t\rangle |l\rangle |j\rangle \frac{1}{\left\|X^{[t,l,j]}\right\|_F \sqrt{Mt}} \sum_x \sum_y \left(\sum_{k=1}^{N}\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M} -\frac{\eta^{\tau,l}}{M}(-1)^{\tau\cdot x+\mu\cdot y}\delta_j^{\tau,\mu,l} a_k^{\tau,\mu,l-1} |k\rangle\right) |x\rangle |y\rangle \\ &= |t\rangle |l\rangle |j\rangle \left(\sin\theta \left|W_j^{t,l}\right\rangle |0\rangle |0\rangle + \cos\theta \left|\mathrm{junk}\right\rangle \left|00^{\perp}\right\rangle\right), \end{aligned} \tag{4}$$

where

$$\left|W_j^{t,l}\right\rangle = \frac{1}{\left\|W_j^{t,l}\right\|}\sum_{k=1}^{n_l} W_{jk}^{t,l} |k\rangle = \frac{1}{\left\|W_j^{t,l}\right\|}\sum_{k=1}^{N}\left(\sum_{\tau=0}^{t-1}\sum_{\mu=1}^{M}\frac{-\eta^{\tau,l}}{M}\delta_j^{\tau,\mu,l} a_k^{\tau,\mu,l-1}\right)|k\rangle ,$$

$\sin^2\theta = \frac{\|W_j^{t,l}\|^2}{\|X^{[t,l,j]}\|_F^2 Mt}$ and $|00^{\perp}\rangle$ is a state orthogonal to $|0\rangle|0\rangle$. By the well-known quantum procedures of amplitude amplification and amplitude estimation [4], 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 $O(\frac{T(U)}{\epsilon\sin(\theta)})$ and $|x\rangle$ can be generated in expected time $O(\frac{T(U)}{\sin(\theta)})$,where $T(U)$ is the time required to implement $U$. Amplitude amplification applied to the unitary preparing the state in (4) allows one to generate $|W_j^{t,l}\rangle$ in time

$$T_W = O\left(\frac{\left\|X^{[t,l,j]}\right\|_F}{\left\|W_j^{t,l}\right\|}\sqrt{TM}\,\mathrm{polylog}(TMN)\right).$$

Similarly, amplitude estimation can be used to find an $s$ satisfying $|s - \frac{\|W_j^{t,l}\|^2}{\|X^{[t,l,j]}\|_F^2 Mt}| \leq \xi \frac{\|W_j^{t,l}\|^2}{\|X^{[t,l,j]}\|_F^2 Mt}$ in time $O(T_W/\xi)$. Outputting $\overline{\|W_j^{t,l}\|} = \|X^{[t,l,j]}\|_F\sqrt{Mts}$ then satisfies $|\overline{\|W_j^{t,l}\|} - \|W_j^{t,l}\|| \leq \xi\|W_j^{t,l}\|$, since $|\sqrt{s}\|X^{[t,l,j]}\|_F\sqrt{Mt} - \|W_j^{t,l}\|| \leq \xi\|W_j^{t,l}\|$.

If the values $\delta_j^{t,m,l}$ and $\frac{-\eta^{t,l}}{M}a_j^{t,m,l-1}\|\delta^{t,m,l}\|$ are also stored at every iteration, analogous results also hold for creating quantum states $|(W^{t,l})_j^T\rangle$ corresponding to the columns of $W^{t,l}$, except in this case the key ratio $\|W_j^{t,l}\|/\|X^{[t,l,j]}\|_F$ that appeared in the previous proof is replaced by $\|(W^{t,l})^j\|/\|\tilde{X}^{[t,l,j]}\|_F$, where $\|\tilde{X}^{[t,l,j]}\|_F$ is the norm of the quantum state

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

## 4.3 Quantum-Inspired Classical Algorithm for $(\epsilon,\gamma)$-Feedforward Network Training

Suppose that for a given $l \in \{2,\ldots,L\}$ and $j \in [n_l]$, the following are stored in an $\ell_2$-BST:

$$\begin{aligned} &a^{\tau,\mu,l-1} && \forall\tau \in \{0,1\ldots,t-1\}, \forall\mu \in [M] \\ &a^{t,m,l-1} && \\ &X^{[t,l,j]}. && \end{aligned}$$

It is then possible to sample $(\tau,\mu)$ with probability $P(\tau,\mu) = \frac{(X^{[t,l,j]})_{\tau,\mu}^2}{\|X^{[t,l,j]}\|_F^2}$ and $k$ with conditional probability $P(k \mid \tau,\mu) = (\frac{a_k^{\tau,\mu,l-1}}{\|a^{\tau,\mu,l-1}\|})^2$ in time $O(\text{polylog}(TMN))$. The random variable $Z(k,\tau,\mu) \overset{\text{def}}{=} a_k^{t,m,l-1}\frac{\|a^{\tau,\mu,l-1}\|}{a_k^{\tau,\mu,l-1}}\frac{\|X^{[t,l,j]}\|_F^2}{(X^{[t,l,j]})_{\tau,\mu}}$ can be computed in time $O(\text{polylog}(TMN))$ and has expectation

$$\begin{aligned} \langle Z\rangle &= \sum_{k,\tau,\mu} P(k \mid \tau,\mu)P(\tau,\mu)Z(k,\tau,m) \\ &= \sum_{k,\tau,\mu}\frac{a_k^{\tau,\mu,l-1}}{\left\|a^{\tau,\mu,l-1}\right\|}\left(X^{[t,l,j]}\right)_{\tau,\mu}a_k^{t,m,l-1} \\ &= \sum_k\left(\sum_{\tau,\mu} -\frac{\eta^{\tau,l}}{M}\delta_j^{\tau,\mu,l}a_k^{\tau,\mu,l-1}\right)a_k^{t,m,l-1} \\ &= \langle W_j^{t,l}, a^{t,m,l-1}\rangle \end{aligned}$$

and variance

$$\sigma^2 \leq \sum_{k,\tau,\mu}\left(a_k^{t,m,l-1}\right)^2\left\|X^{[t,l,j]}\right\|_F^2 = MT\left\|a^{t,m,l-1}\right\|^2\left\|X^{[t,l,j]}\right\|_F^2.$$

By the Chebyshev and Chernoff-Hoeffding inequalities, taking the median of $O(\log(1/\gamma))$ averages, each an average of $O(\frac{1}{\epsilon'^2})$ independent copies of $Z$, produces an estimate $s$ within $\epsilon'\sigma$ of $\langle Z\rangle$. Taking $\epsilon' = \frac{\epsilon\max\{|\langle W_j^{t,l}, a^{t,m,l-1}\rangle|,1\}}{\sqrt{MT}\|a^{t,m,l-1}\|\|X^{[t,l,j]}\|_F}$ then allows an $s_j^{t,m,l}$ to be computed satisfying $|s_j^{t,m,l} - \langle W_j^{t,l}, a^{t,m,l-1}\rangle| \leq \epsilon\max\{|\langle W_j^{t,l}, a^{t,m,l-1}\rangle|,1\}$ with probability at least $1-\gamma$.

Replacing the RIPE procedure in Algorithm 1 with this method for computing $s_j^{t,m,l}$ gives a quantum-inspired classical $(\epsilon,\gamma)$-feedforward subroutine that runs in time

$$\tilde{O}\left(TMN\frac{\log(1/\gamma)}{\epsilon^2}R_a^{cl\,t,m}\right),$$

where $R_a^{cl\,t,m} = \frac{1}{N-n_1}\sum_{l=2}^{L}\sum_{j=1}^{n_l}(\frac{\|X^{[t,l,j]}\|_F\|a^{t,m,l-1}\|}{\max\{1,|\langle W_j^{t,l},a^{t,ml-1}\rangle|\}})^2$. Using $(\sum_{i=1}^n x_i)^2 \leq n\sum_{i=1}^n x_i^2$ for $x_i \geq 0$, it follows that $R_a^{cl\,t,m} \geq (R_a^{t,m})^2$.

The RIPE procedure in Algorithm 2 can similarly be replaced to obtain a quantum-inspired classical $(\epsilon,\gamma)$-backpropagation subroutine that runs in time

$$\tilde{O}\left(TMN\frac{\log(1/\gamma)}{\epsilon^2}R_\delta^{cl\,t,m}\right),$$

where $R_\delta^{cl\,t,m} \geq (R_\delta^{t,m})^2$.

Finally, by substituting the quantum $(\epsilon,\gamma)$-feedforward and quantum $(\epsilon,\gamma)$-backpropagation subroutines with their quantum-inspired classical counterparts in Algorithm 3, one obtains a quantum-inspired classical $(\epsilon,\gamma)$-training algorithm that runs in time

$$\tilde{O}\left((TM)^2\,N\frac{\log(1/\gamma)}{\epsilon^2}\left(R_a^{cl}+R_\delta^{cl}\right)\right),$$

with $R_a^{cl} \geq R_a^2$ and $R_\delta^{cl} \geq R_\delta^2$.

## APPENDIX

## A CLASSICALLY SAMPLING FROM THE RIPE DISTRIBUTION

The quantum $(\epsilon,\gamma)$-feedforward and quantum $(\epsilon,\gamma)$-backpropagation subroutines (Algorithms 1 and 2, respectively) of the main text make use of the RIPE procedure to compute values $s$ that satisfy

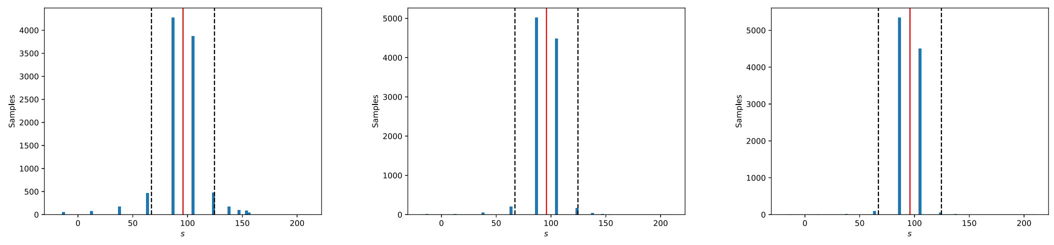
$$|s-\langle x,y\rangle| \leq \max\{\epsilon\,|\langle x,y\rangle|\,,\epsilon\}$$

with probability at least $1-\gamma$ for given input vectors $x,y\in\mathbb{R}^n$. In this section, we give a classical subroutine for generating samples from RIPE distribution, for the special case of RIPE where one has exact knowledge of the the norms $\|x\|$ and $\|y\|$ (the generalization to the case where we only have estimates of the norms is straightforward). At a high level, this procedure is based on the following ideas (see the work of Kerenidis et al. [21] for more details):

- The inner product estimation procedure of Kerenidis et al. [21] on which RIPE is based implicitly assumes access to a unitary operator for efficiently creating the state $|\phi\rangle = \frac{\|x\|\,|0\rangle|x\rangle+\|y\|\,|1\rangle|y\rangle}{\sqrt{\|x\|^2+\|y\|^2}}$. Applying a Hadamard operator to the first register produces the state

$$|\Psi\rangle = \sqrt{a}\,|1\rangle\,|\Psi_1\rangle + \sqrt{1-a}\,|0\rangle\,|\Psi_0\rangle\,,$$

  where $a=\frac{\|x\|^2+\|y\|^2-2\langle x,y\rangle}{2(\|x\|^2+\|y\|^2)}$. Obtaining an estimate $\bar{a}$ to $a$ satisfying $|\bar{a}-a|\leq\epsilon_a=\frac{\epsilon\max\{1,|\langle x,y\rangle|\}}{\|x\|^2+\|y\|^2}$ will therefore allow us to compute an estimate $s$ to $\langle x,y\rangle$ satisfying $|s-\langle x,y\rangle|\leq\epsilon\max\{1,|\langle x,y\rangle|\}$, by taking $s=(\|x\|^2+\|y\|^2)(1-2\bar{a})/2$.
- Performing amplitude estimation [4] on $|\Psi\rangle$ with $\log M$ ancilla qubits returns a value $\tilde{a}$ satisfying $|\tilde{a}-a|\leq\frac{\pi}{M}+\left(\frac{\pi}{M}\right)^2$ with probability at least $8/\pi^2$. This process requires time $\tilde{O}(MT)$, where $T$ is the time required to implement the unitary required for the creation of state $|\phi\rangle$. Taking $M=\lceil\frac{\pi}{2\epsilon_a}\left(1+\sqrt{1+4\epsilon_a}\right)\rceil$ suffices to ensure that $|\tilde{a}-a|\leq\epsilon_a$.



(a) $\gamma = 0.2, Q = 1, f = 86\%$. (b) $\gamma = 0.05, Q = 3, f = 97\%$. (c) $\gamma = 0.01, Q = 5, f = 99\%$.

Fig. 7. Concentration of 10,000 samples of estimates $s$ of the inner product of vectors $x, y$ given in (5), drawn from the RIPE distribution, for $\epsilon = 0.3$ and various values of $\gamma$. The vertical red line indicates the true value of the inner product $\langle x, y\rangle = 95.83$, and the vertical dashed lines are located at $\pm\epsilon \max\{1, |\langle x, y\rangle|\}$. The actual percentage of points sampled that lay within the desired range in each case is denoted by $f$. Note that in the case of $(a)$, $\gamma = 0.2$ corresponds to $1 - \gamma < 8/\pi^2$, so $Q = 1$ suffices and there is no need to take medians.

- By the Hoeffding bound, repeating the preceding procedure $Q = \lceil \frac{\log(1/\gamma)}{2(8/\pi^2 - 1/2)^2} \rceil_{odd}$ times and taking the median of the results gives a value $\bar{a}$ satisfying $|\bar{a} - a| \le \epsilon_a$ with probability at least $1 - \gamma$. The notation $\lceil z \rceil_{odd}$ denotes the smallest odd integer greater than or equal to $z$.

The amplitude amplification part of the sampling subroutine makes use of a distance function $d : \mathbb{R}^2 \to \mathbb{Z}$ given by $d(\omega_0, \omega_1) = \min_{z \in \mathbb{Z}}\{z + \omega_1 - \omega_0\}$.

**ALGORITHM 4:** Classically Sampling from the RIPE Distribution

```
Input: $x, y \in \mathbb{R}^n$

Compute $a = \frac{\|x\|^2 + \|y\|^2 - 2\langle x, y\rangle}{2\left(\|x\|^2 + \|y\|^2\right)}$, $\theta_a = \sin^{-1}\left(\sqrt{a}\right)$, $\epsilon_a = \frac{\epsilon \max\{1, |\langle x, y\rangle|\}}{\|x\|^2 + \|y\|^2}$, $M = \left\lceil \frac{\pi}{2\epsilon_a}\left(1 + \sqrt{1 + 4\epsilon_a}\right)\right\rceil$

for $j = 1, \dots, m$ do
    $a_j = \sin^2(\pi j/M)$
    $p(a_j) = \left|\frac{\sin(Md(j/M, \theta_a/\pi))}{M \sin(d(j/M, \theta_a/\pi))}\right|^2$
end for

for $q = 1, \dots, Q$ do
    Sample $\tilde{a}^{(q)} \sim p$
end for

Compute $\bar{a} = \mathrm{median}(\tilde{a}^{(1)}, \dots \tilde{a}^{(Q)})$

Return: $s = \left(\|x\|^2 + \|y\|^2\right)(1 - 2\bar{a})/2$
```

Examples of such samples are shown in Figure 7 for randomly chosen vectors $x$ and $y$ given by

$$\begin{aligned} x &= (5.88414114, 2.0327562, 1.68155901, 7.91848042, 1.61922687), \\ y &= (5.15610287, 7.2034771, 9.88496245, 3.46281654, 4.20607662), \end{aligned} \tag{5}$$

$\epsilon = 0.3$, and various values of $\gamma$. These vectors have norms $\|x\| = 10.34$ and $\|y\| = 14.35$, and inner product $\langle x, y\rangle = 95.38$.

## REFERENCES

[1] Juan Miguel Arrazola, Alain Delgado, Bhaskar Roy Bardhan, and Seth Lloyd. 2019. Quantum-inspired algorithms in practice. arXiv:1905.10415.

[2] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. 2015. On the robustness of bucket brigade quantum RAM. *New Journal of Physics* 17, 12 (2015), 123010.

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

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

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

[6] Carlo Ciliberto, Mark Herbster, Alessandro Davide Ialongo, Massimiliano Pontil, Andrea Rocchetto, Simone Severini, and Leonard Wossnig. 2018. Quantum machine learning: A classical perspective. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474, 2209 (2018), 20170551.

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

[8] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*. 1269–1277.

[9] Olivia Di Matteo, Vlad Gheorghiu, and Michele Mosca. 2020. Fault-tolerant resource estimation of quantum random-access memories. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–13.

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

[11] Vedran Dunjko and Hans J. Briegel. 2018. Machine learning and artificial intelligence in the quantum domain: A review of recent progress. *Reports on Progress in Physics* 81, 7 (2018), 074001.

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

[13] Richard P. Feynman. 1982. Simulating physics with computers. *International Journal of Theoretical Physics* 21, 6 (1982), 467–488.

[14] András Gilyén, Seth Lloyd, and Ewin Tang. 2018. Quantum-inspired low-rank stochastic regression with logarithmic dependence on the dimension. arXiv:1811.04909

[15] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. 2008. Architectures for a quantum random access memory. *Physical Review A* 78, 5 (2008), 052310.

[16] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. 2008. Quantum random access memory. *Physical Review Letters* 100, 16 (2008), 160501.

[17] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. 249–256.

[18] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep Learning*. Vol. 1. MIT Press, Cambridge, MA.

[19] Connor T. Hann, Chang-Ling Zou, Yaxing Zhang, Yiwen Chu, Robert J. Schoelkopf, Steven M. Girvin, and Liang Jiang. 2019. Hardware-efficient quantum random access memory with hybrid quantum acoustic systems. *Physical Review Letters* 123, 25 (2019), 250501.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[21] Iordanis Kerenidis, Jonas Landman, Alessandro Luongo, and Anupam Prakash. 2019. q-means: A quantum algorithm for unsupervised machine learning. In *Advances in Neural Information Processing Systems*. 4136–4146.

[22] Iordanis Kerenidis and Alessandro Luongo. 2018. Quantum classification of the MNIST dataset via Slow Feature Analysis. arXiv:1805.08837.

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

[24] Iordanis Kerenidis and Anupam Prakash. 2020. Quantum gradient descent for linear systems and least squares. *Physical Review A* 101, 2 (2020), 022316.

[25] Mária Kieferová and Nathan Wiebe. 2017. Tomography and generative training with quantum Boltzmann machines. *Physical Review A* 96, 6 (2017), 062327.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.

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

[28] Yang Liu and Shengyu Zhang. 2015. Fast quantum algorithms for least squares regression and statistic leverage scores. In *Proceedings of the International Workshop on Frontiers in Algorithmics*. 204–216.

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

[30] Michael Nielsen. 2015. *Neural Networks and Deep Learning*. Determination Press.

[31] Daniel K. Park, Francesco Petruccione, and June-Koo Kevin Rhee. 2019. Circuit-based quantum random access memory for classical data. *Scientific Reports* 9, 1 (2019), 1–8.

[32] Anupam Prakash. 2014. *Quantum Algorithms for Linear Algebra and Machine Learning*. Ph.D. Dissertation. University of California at Berkeley.

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

[34] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. 2014. Quantum support vector machine for big data classification. *Physical Review Letters* 113, 13 (2014), 130503.

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

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

[37] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'13)*. IEEE, Los Alamitos, CA, 6655–6659.

[38] Maria Schuld and Nathan Killoran. 2019. Quantum machine learning in feature Hilbert spaces. *Physical Review Letters* 122, 4 (2019), 040504.

[39] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. 2014. The quest for a quantum neural network. *Quantum Information Processing* 13, 11 (2014), 2567–2586.

[40] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. 2015. An introduction to quantum machine learning. *Contemporary Physics* 56, 2 (2015), 172–185.

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

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

[43] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. 2015. Convolutional neural networks with low-rank regularization. arXiv:1511.06067.

[44] Ewin Tang. 2018. Quantum-inspired classical algorithms for principal component analysis and supervised clustering. arXiv:1811.00414.

[45] Ewin Tang. 2019. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 217–228.

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

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

[48] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of neural networks using DropConnect. In *Proceedings of the International Conference on Machine Learning*. 1058–1066.

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

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

[51] Nathan Wiebe and Leonard Wossnig. 2019. Generative training of quantum Boltzmann machines with hidden units. arXiv:1905.09902.

[52] Leonard Wossnig, Zhikuan Zhao, and Anupam Prakash. 2018. Quantum linear system algorithm for dense matrices. *Physical Review Letters* 120, 5 (2018), 050502.

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

Received November 2019; revised June 2020; accepted July 2020