# Quantum Neurons: analyzing the building blocks of quantum deep learning algorithms

Zachary Cetinic, Daniel Hidru, Marta Skreta

December 7, 2019

## 1 Abstract

The massive computational power of modern computers and large volume of data generated in recent years has resulted in a surge of deep learning algorithms that have demonstrated success in a variety of tasks, from classification and segmentation to generative modelling. As the amount of data grows and the number of transistors in silicon chips approaches the physical limit, solutions are needed to speed up deep learning algorithms and reduce their resource consumption. Quantum computation is an attractive solution as it has been shown to speed up algorithms in related fields. In this paper we analyze the field of quantum neurons, the building blocks of deep learning networks. We note that developing a quantum neuron is a challenging task since classical neurons require procedures that are difficult to simulate with quantum algorithms. We found that while quantum neurons have been proposed to simulate classical neurons, they often pose no advantage in terms of runtime over classical algorithms for a single input vector, but may provide a benefit when data is trained in superposition. We also found that there is potential in the ability of quantum neurons to reduce the runtime of learning network parameters; however, it is too soon to tell whether it these algorithms will be beneficial in practice.

## 2 Introduction

In recent years, both machine learning (ML) and quantum computing have, each in their own right, attracted a significant amount of attention in research and media outlets. Quantum machine learning is the marriage of these two domains: it concerns the use of quantum algorithms to achieve significant speedups of machine learning algorithms compared to their classical counterparts [1]. The paradigm of quantum machine learning has been to replace classical subroutines with a faster quantum equivalent. This gave rise to algorithms such as the HHL algorithm (named after its inventors: Harrow, Hassidim, and Lloyd), which can solve linear equations faster on a quantum computer than on a classical one [2], and quantum support vector machines, which can learn to classify data with an exponential speedup over

classical algorithms [3]. However, these algorithms aren't perfect: oftentimes, they require assumptions to be made about the dataset that may or may not hold in practice, and they might not return the algorithm output in a form that is useful to the user [4].

Deep learning algorithms constitute a class of ML algorithms that have experienced a boom in popularity over the last 7 years due to their ability to learn nontrivial patterns in complex and high-dimensional data [5]. While papers describing a quantum-style neural network have been proposed since the 1990s, progress in the development of a quantum neural network has been slow and has not reached a level of maturity as other sectors of quantum machine learning [6]. Reasons for this may include the difficulty of mapping deep learning algorithms, which are sequential and nonlinear, into the linear and parallel framework of quantum mechanics. Perhaps the single greatest challenge to developing a quantum neural network is the difficulty of implementing nonlinear activation functions in quantum neurons, the quantum equivalent of classical neurons in classical neural networks. In this section, we analyze the current state of quantum neurons and outline challenges currently hindering this field from progressing.

In this paper, we focus on recent developments in quantum neuron architecture. Neurons are the fundamental building blocks of deep neural networks. Any speedup obtained at the neuron level has the potential to have a significant impact in the runtime of the overall algorithm. We begin this paper with an overview of classical deep learning and neurons. We then provide an overview of the current quantum neuron literature, analyzing the pros and cons of each method compared to their classical counterpart. Finally, we end with a section on how quantum neurons are implemented in practice and the difficulties that accompany running these algorithms on a quantum computer.

# 3 Deep Learning

Deep learning is a class of machine learning algorithms that extracts features from an input and maps them to an output using an artificial neural network (ANN) [5]. While ANNs make up a diverse set of models, their unifying factor is that they are composed of multiple linear operations joined together with nonlinear transformations, which are called activation functions. Examples of commonly used nonlinear transformations are the sigmoid function ($f(x) = \frac{1}{1+e^{-x}}$) and ReLU function ($f(x) = max(0, x)$) [7]. These non-linearities are an integral part in the ability of an ANN to be a universal function approximator [8, 9]. The term "deep" is used to signify multiple linear and nonlinear layers joined together to form a single network.

Deep learning algorithms are used to solve diverse problems, such as classification, regression, segmentation, and generative modelling. They are used in a diverse set of fields, such as healthcare, finance, and chemistry [5]. The deep neural network boom of the last 7 years was catalyzed by the development of AlexNet, a deep convolutional neural network trained on millions of images to predict 1000 different categories [10].

Every year computers are becoming increasingly more powerful, allowing them to easily process very large datasets and learn billions of features that represent subtle and complex

patterns that would be otherwise difficult to identify. These computers are even capable of beating human experts in some cases [11]. To allow for even greater computational power, specialized architectures are being implemented specifically for deep learning. The latest Nvidia graphical processing units (GPU) include specialized tensor cores for deep learning problems [12]. Moreover, Google announced the tensor processing unit (TPU), a circuit created specifically to speed up artificial intelligence applications [13].

However, as the size of datasets increases and as the number of transistors per classical circuit approaches its limit, other hardware-based solutions will be needed [6]. This is why there exists an effort to develop quantum versions of machine learning algorithms for quantum computers. Progress in this direction requires the development of efficient quantum algorithms that can be used to implement quantum neural networks. For quantum neural networks to be useful, they need to provide some advantage over classical neural networks, whether it be time reduction, storage reduction, or superior pattern recognition.

# 4 Classical Neurons

The fundamental building block of an ANN is a neuron. In its simplest form, a neuron takes in an $N$-dimensional input vector $\vec{x}$, combines it linearly with a weight vector $\vec{w}$ using the dot product and adds a bias $b$. This results in an intermediate value $\theta$ that is passed into an activation function $a$, such as a sigmoid or ReLU function, to introduce a nonlinearity. (Fig. 1) The output of the neuron is:

$$y = a(\theta) = a(\vec{x} \cdot \vec{w} + b) \tag{1}$$

The output $y$ can either be fed as a learned input into another layer of neurons in a deep network or be interpreted as the prediction of the network for a given input. The goal of training a neuron is to learn weights that maximize the probability that the final neuron in the network will produce the correct output for a given input. During training, weights are tweaked to maximize this probability using sequential algorithms such as backpropagation [14].

# 5 Quantum Deep Learning Building Blocks

## 5.1 Quantum Perceptrons

One of the earliest examples of a neuron was the perceptron, first described in the 1950s by Rosenblatt [15]. Perceptrons use the step-activation function as the nonlinearity, which returns an output $y$ such that:

$$y = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

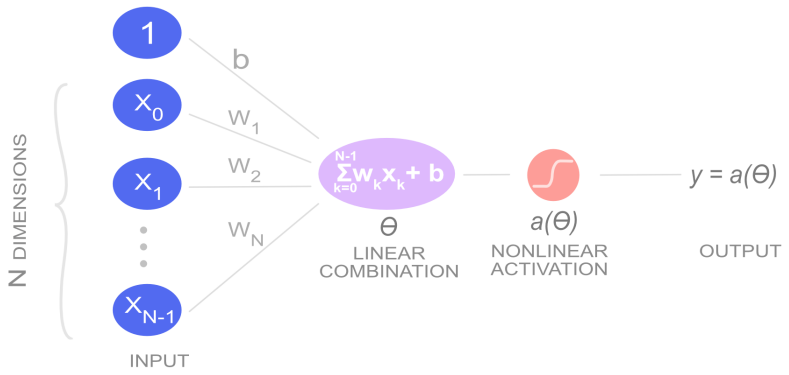


Figure 1: Schematic showing classical neuron. An $N$-dimensional input $\vec{x}$ is linearly combined with an $N$-dimensional weight vector $\vec{w}$ and a bias term $b$. The resulting dot product, $\theta$, is passed though an activation function to introduce a nonlinearity, $a(\theta)$, to produce the output, $y$.

Schuld et al. proposed a quantum perceptron imitating the step-activation function using the phase estimation algorithm (Fig. 2) [16]. The entries of the input $\vec{x}$ are restricted to have binary values $x_k \in \{-1, 1\}$ so the input can be encoded in the state $|\Psi_{\vec{x}}\rangle = |x_0, ..., x_{N-1}\rangle$, where entries of -1 are represented by a quantum state of 0. The input state is passed into the circuit along with ancilla qubits, $|0\rangle^{\otimes\tau}$, where $\tau$ is the precision of the phase estimation algorithm. A Hadamard gate is applied to each of the ancilla qubits. The resulting state is $\frac{1}{\sqrt{2^\tau}} \sum_{j=0}^{2^\tau-1} |j\rangle|\Psi_{\vec{x}}\rangle$.

The weights of the model $\vec{w}$ are constrained to be continuous values $w_k \in [-1, 1)$. The weights are encoded in the unitary $U_{\vec{w}}$, which is then applied to the input conditioned on the ancilla qubits. $U_{\vec{w}}$ can be decomposed into a product of unitaries: $U_{\vec{w}} = U_{w_{N-1}}...U_{w_0}U_0$, where $U_{w_k}$ is the unitary applied to $x_k$. $U_0$ is a global phase shift of $\pi i$ and each $U_{w_k}$ has the form:

$$U_{w_k} = \begin{bmatrix} e^{-2\pi i \frac{w_k}{2N}} & 0 \\ 0 & e^{2\pi i \frac{w_k}{2N}} \end{bmatrix}$$

After applying $U_{\vec{x}}$ in the circuit, the ancilla qubits will have the state $\frac{1}{\sqrt{2^\tau}} \sum_{j=0}^{2^\tau-1} e^{2\pi i j\phi}|j\rangle$, where $\phi = \frac{\vec{x}\cdot\vec{w}}{2N} + 0.5$. Since $x_k \in \{-1, 1\}$ and $w_k \in [-1, 1)$, the dot product $\vec{x} \cdot \vec{w} \in [-N, N]$ and $\phi \in [0, 1]$. Therefore, $\phi$ is in the correct range so that it may be extracted using the phase estimation algorithm. If a single bit of precision is used to extract the value of $\phi$, then $\phi_1 = \mathbb{1}_{\frac{\vec{x}\cdot\vec{w}}{2N} \geq 0}$ and the result may be used to generate the output of the model:

$$y = \begin{cases} 1 & \text{if } \frac{\vec{w}\cdot\vec{x}}{2N} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

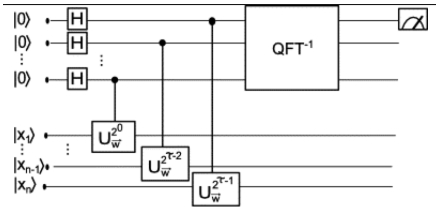


Figure 2: Phase estimation algorithm used to implement a quantum perceptron. The dot product of $\vec{x}$ and $\vec{w}$ is stored in the eigenvalue of the qubit state. Inverse QFT is used to extract the dot product ($\phi$); the output becomes 1 if it is above the threshold and -1 otherwise. Figure taken from [16].

While Schuld et al. comment on classical methods to train neurons and potential quantum methods, they don't provide a learning method that is specific to their perceptron.

Since only one bit of precision of required for the threshold function, only two gates are needed in the phase estimation algorithm. However, this may become problematic for an increasing input dimension, $N$. For example, given a data-set that is a uniform random distribution in [-1, 1), the majority of points in the output distribution are centered at $\frac{1}{2}$. Greater precision around the mean is required, and so the value of $\tau$ must increase. Specifically, Schuld et al. found that the circuit should be applied a number of times proportional to $O(\log\sqrt{N})$ [16]. To implement $QFT^{-1}$, the number of gates that are required is $\frac{\tau(\tau+1)}{2} + 3\frac{\tau}{2}$, which is $O(\log^2\sqrt{N})$ [17]. To implement the neuron on a quantum computer using this algorithm, $O(N\ \log^2\sqrt{N}) = O(N\ \log(N))$ time is required, whereas an upper bound of $O(N)$ time is required to implement the classical equivalent. Thus, this neuron does not pose any advantage from a time perspective since it requires more resources than its classical equivalent for a single input. However, the paper claims that the training data can be represented in a superposition and so the circuit needs to be called only once. Thus, this algorithm may provide a global benefit in terms of reducing the overall runtime since all the data can be processed in one step, even though that step is longer than a single step on a classical computer, where we would need to run the algorithm on multiple batches of the data.

Quantum computing may also be used to train perceptron models faster than classical algorithms. Rather than define a new way to compute a perceptron model in a quantum computer, Kapoor et al. developed two quantum algorithms for perceptron learning that provided improvements in the computational and statistical complexity in the scenario where they have access to an oracle that computes the output of the perceptron [1]. One of the algorithms achieves these improvements by utilizing the parallelism of quantum computation to identify which training samples are misclassified and need to be used by the model to update its weights. Given a set of $S$ training examples and a function which computes the misclassification of an training example, Grover's search algorithm may be used to identify a training example which has been misclassified in $O(\sqrt{S})$. The author's were able utilize this speed up to create a quantum algorithm that resulted in a $\sqrt{S}$ reduction in the number of training examples required to learn a perceptron with a sufficiently small error over the

classical perceptron algorithm. This algorithm also does not follow the paradigm of classical machine learning algorithms, as classical machine learning algorithms update weights by measuring their gradient with respect to every training example, which takes $O(S)$ time. Instead, it only looks for samples that were misclassified. While this algorithm reduces the number of training examples required for the computation, techniques like this that constrain quantum methods to mimic traditional machine learning methods might be slowing down the progress of quantum machine learning, as the full potential of quantum algorithms has not yet been exploited for this task.

## 5.2 Quantum Neurons

A number of works have outlined the simulation of artificial neurons jointly using both classical and quantum algorithms. A notable recent contribution in this area is by Tacchino et al., who demonstrated that an artificial neuron able to analyze patterns in binary images can be implemented on a real quantum computer: one of IBM's quantum processors [18]. The input to their algorithm is a $N = 2^q$ vector $\vec{x} \in \{-1, 1\}^N$, which is mapped to the following wave function:

$$|\Psi_{\vec{x}}\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j |j\rangle \tag{2}$$

where $N$ is the dimension of the input string, $x_j$ is the value of the $j$-th input, and $|j\rangle$ represents the states $|j\rangle \in \{|00...00\rangle, |00...01\rangle, ..., |11...11\rangle\}$ that form the basis vectors for $q$ qubits. Due to implementation difficulties, they were only able to simulate $q = 2$ qubits on the quantum compute. The benefit of this is that it allows the dynamics of the system to be controlled more robustly on account of lower error rates [18, 19]. Similarly, the weight vector $\vec{w} \in \{-1, 1\}^N$ is mapped to the state

$$|\Psi_{\vec{w}}\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} w_j |j\rangle \tag{3}$$

The next step involves computing the inner product between $\vec{w}$ and $\vec{x}$. First, a unitary matrix $U_w$ rotates $|\Psi_{\vec{w}}\rangle$ to the $|11...11\rangle$ axis, i.e. $U_w|\Psi_{\vec{w}}\rangle = |1\rangle^{\otimes q} = |N-1\rangle$. Subsequently applying $U_w$ to $|\Psi_{\vec{x}}\rangle$ corresponds to the projection of the vector $U_w|\Psi_{\vec{x}}\rangle$ over the $|11...11\rangle$ axis. It follows that the dot product between the two quantum states is

$$\begin{aligned} \langle \Psi_{\vec{w}} | \Psi_{\vec{x}} \rangle &= \langle \Psi_{\vec{w}} | U_w U_w^\dagger | \Psi_{\vec{x}} \rangle \\ &= \langle N-1 | \phi_{x,w} \rangle \\ &= c_{N-1} \end{aligned} \tag{4}$$

where $c_{N-1}$ is the coefficient of the last state of $|\phi_{x,w}\rangle$ and $|\phi_{x,w}\rangle = U_w^\dagger |\Psi_{\vec{x}}\rangle$. To compute the dot product between $\vec{x}$ and $\vec{w}$, equations 2 and 3 are combined such that:

$$
\begin{aligned}
\langle \Psi_{\vec{w}} | \Psi_{\vec{x}} \rangle &= \left(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \langle j|\right)\left(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} w_j |j\rangle\right) \\
&= \frac{1}{N} \sum_{j=0}^{N-1} x_j w_j \langle j|j\rangle \\
&= \frac{1}{N} \sum_{j=0}^{N-1} x_j w_j \\
&= \frac{1}{N} \vec{w} \cdot \vec{x}
\end{aligned}
\tag{5}
$$

Thus, we have that the dot product $\vec{w} \cdot \vec{x} = N\langle \Psi_{\vec{w}} | \Psi_{\vec{x}} \rangle$. Since $\langle \Psi_{\vec{w}} | \Psi_{\vec{x}} \rangle = c_{N-1}$, the normalized dot-product can be extracted from the last state of $|\phi_{x,w}\rangle$. This can be implemented in a circuit using an ancilla qubit and a multi-controlled NOT gate that changes the state of the ancilla qubit depending on the state of the qubit (Fig. 3). Specifically, this will only flip the ancilla qubit to the state $|1\rangle$ for the final state of $|\phi_{x,w}\rangle$, which is where $c_{N-1}$ is stored. After the multi-controlled NOT gate is applied, the state of the system is:

$$
|\phi_{x,w}\rangle|0\rangle \rightarrow \sum_{j=0}^{N-2} c_j |j\rangle|0\rangle + c_{N-1}|N-1\rangle|1\rangle
$$

The nonlinearity is applied by measuring the ancilla qubit. This will cause the qubit to collapse to $|1\rangle$ with probability $|c_{N-1}|^2$. Therefore, the output of the system is probabilistic with $P(y=1) = (\frac{\vec{w}\cdot\vec{x}}{N})^2$. The value of this probability can also be estimated by sampling the output multiple times and calculating the frequency of $y = 1$. We note that there is a potential downside to this method during the learning process. Suppose the ground truth label is 0, but the algorithm predicts the coefficient is more than 0.5 and misclassifies it as 1. To update the weight parameters for the next iteration, the algorithm employed by Tacchino et al. searches for instances where $\vec{w}$ and $\vec{x}$, which caused the inner product to increase. However, to reduce the probability that the algorithm outputs 1, they want the inner product to decrease. Thus, they find the subset where they coincide and randomly flip some proportion of bits in $\vec{w}$. The issue in this problem arises when a very large subset of $\vec{w}$ and $\vec{x}$ *don't* coincide, since the inner product is a large negative value and consequently the square of the inner product is a large positive value. Now, when the algorithm searches for places where they do coincide, very few values coincide. Flipping any $\vec{w}$ thus makes the inner product more negative, which makes the square more positive and so the probability of output being 1 is higher, which is the opposite of what should happen.

An overview of the circuit can be viewed in Figure 3. While they were successfully able to implement their model in an IBM processor to classify patterns, the time it took for the algorithm to run was linear in the dimension of the input vector, which is the same as classical models. Thus, there is no time advantage of their model over what currently exists.

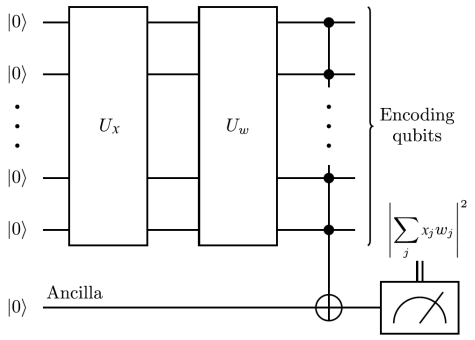


Figure 3: Circuit used by Tacchino et al. to implement a quantum neuron. The dot product between $\vec{w}$ and $\vec{x}$ is computed and stored as the coefficient $c_{N-1}$ of the last qubit of the input. This information is trasnferred to an ancilla qubit using a multi-controlled CNOT gate. The ancilla qubit is then measured; it collapses to $|1\rangle$ with probability proportional to $c_{N-1}$ and $|0\rangle$ otherwise. The act of measuring simulates the nonlinearty. Figure adapted from [18].

The authors do, however, gain an advantage over classical methods in terms of parameter storage during training by encoding their data in an exponentially smaller number of qubits. Note that during when saving the model to memory this advantage would disappear since the space required to store the weights is linear; however, if memory was a bottleneck during training (for example, all parameters couldn't fit into qRAM), this method would provide an advantage.

Cao et al. took the quantum neuron one step further by enabling it to simulate a sigmoid function while processing inputs in quantum superposition [20]. Their algorithm encodes an input $\vec{x} \in \{0,1\}^N$ in the quantum state $|\Psi_{\vec{x}}\rangle = |x_1 ... x_n\rangle$ and use a Pauli Y operator is used to rotate each input qubit proportionally by the weights and bias. The operator has the form

$$R_y(a\frac{\pi}{2} + \frac{\pi}{2})|0\rangle = cos(a\frac{\pi}{4} + \frac{\pi}{4})|0\rangle + sin(a\frac{\pi}{4} + \frac{\pi}{4})|1\rangle$$

where $a \in [-1, 1]$ and $R_y(t) = e^{-itY/2}$. The operator rotates a qubit by the angle $t$ around the $Y$ axis, where $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$. To apply the weights and bias onto the inputs, each input qubit $|x_i\rangle$ is rotated by the gates $R_y(2w_i)$ and $R_y(2b)$. Note that the weights and bias are multiplied by a factor of 2 to cancel out the factor $1/2$ in the operator. This is equivalent to applying a gate $R_y(2\theta)$ on an ancilla qubit conditioned on the input $|\Psi_{\vec{x}}\rangle$. The ancilla qubit is then rotated by a gate corresponding to the activation function, $R_y(2q(\theta))$, where $\theta = w_1 x_1 + ... + w_n x_n + b$ and $q$ is a nonlinear function $q(\theta) = \arctan(\tan^2\theta)$ that maps the input to a value between 0 and $\frac{\pi}{2}$. These gates are implemented using a *repeat-until-success* (RUS) circuit, which works as follows: the operator $R_y(2q(\theta))$ is applied to an ancilla qubit conditioned on the input $|\Psi_{\vec{x}}\rangle$. The ancilla qubit is then measured. If it is in state $|0\rangle$, the output qubit is close to $R_y(\pi)|0\rangle = |1\rangle$, which is the right state. If it measures $|1\rangle$, the output qubit has been rotated by $R_y(\pi/2)$ by the circuit, and so the qubit is rotated by $R_y(-\pi/2)$

to undo this rotation. This circuit is applied $k$ times until the ancilla qubit is measured to be $|0\rangle$, which indicates that the output qubit is in state $R_y(0)|0\rangle = |0\rangle$. Equivalently, this process can be thought of rotating the angle $\theta$ closer to 0 or $\pi/2$ depending on whether its less than or greater than the threshold, $\pi/4$, which simulates the sigmoid function. The closer $\theta$ is to the threshold, the more times the circuit needs to be applied. The number of iterations required to rotate $\theta$ to be within a distance $\epsilon$ of 0 or $\pi/2$ is $k = O(\log\frac{1}{\delta\epsilon})$, where $\delta$ is the distance to 0 or $\pi/2$. This work can also be extended to implement other important nonlinear activation functions, such as ReLU [20].

To implement this algorithm and learn the optimal weights, $n$ quantum neurons are required, in addition to $k$ ancilla qubits. The input vector is passed thorugh a neuron and an RUS circuit activates the output to -1 or +1. Neurons can be stacked together in multiple layers to form a deeper network; the RUS circuit is applied at each transition from the $i$-th layer to the $i+1$-th layer. Note that at each layer in the circuit, the neurons must be rescaled to $[0, \pi/2)$ for the RUS circuit to be implemented correctly. The accuracy of the network can be determined by taking the expectation value between the network output and ground truth values. Based on the training accuracy, the weights are updated using the Nead-Medler algorithm, which is a gradient-free optimization tool.

The significant contributions of this paper are that it the presents a quantum equivalent of simulating nonlinearities that are commonly used in modern deep learning, and can be adapted to simulate multiple nonlinearities. These nonlinearities are essential for learning nontrivial patterns in complex distributions. However, this algorithm comes at a cost for speed. While an equivalent classical neuron can compute a nonlinearity in $O(N)$, this quantum neuron does it in $O((\frac{N}{\delta})^{2.075}(\frac{1}{\epsilon})^{3.15})$, where $N$ is the dimension of the input vector, $\delta$ is a resolution parameter associated with settings weight and bias values, and $\epsilon$ is the distance from the circuit output to the attractor (0 or $\pi/2$). Thus, for a single input vector, the runtime is slower on a quantum computer than on a classical computer. However, Cao et al. also note that their algorithm can learn from a superposition of training data, meaning that the algorithm may be overall faster than a classical one if we are allowed to process all the data simultaneously. This demonstrates a method of learning that is different from classical algorithms, which rely on batches of data being fed sequentially to the model. From initial results, it seems as though their network was able to learn appropriate weights from a superposition of training data, but further analysis is required to understand the advantage this strategy has over classical methods in terms of runtime and storage.

# 6 Limitations of training a quantum neuron

The algorithms presented in this paper make many assumptions about available resources and existing implementations of key quantum constructs. They assume that they have access to some oracle that presents them with input data already in a quantum state. However, this task is nontrivial and is one of the reasons it is difficult to implement these algorithms on a quantum computer. In this section we will analyze two of those assumptions: qRAM implementations and encoding classical data into a quantum ready state.

## 6.1 qRAM

Quantum random access memory (qRAM) is a theoretical device that can perform memory accesses in coherent quantum superposition and is one of the key elements needed in many quantum machine learning algorithms to access input data. Several implementations have been proposed for efficient qRAM. The most popular implementation, by Giovannetti et al., uses an algorithm called bucket-brigade, which claims to reduce the number of switches needed from $O(N^{1/d})$, which is the complexity of implementing qRAM using conventional architecture patterns, to $O(log(N))$ [21]. Here, $d$ is the dimension of the memory array. This algorithm works by reading out the memory address using a binary search tree like structure, where each leaf is a memory cell, and only activates switches in the path of the tree. This leads to an exponential decrease in the number of gates required in the running-time computational complexity and they also argue it reduces the need for expensive error correcting measures.

However, Arunachalam et al. later explored the robustness of the aforementioned qRAM implementation and found significant limitations [22]. The bucket-brigade algorithm assumes that its routing components will have a negligible error rate. Arunachalam et al. show that if the algorithm using the qRAM requires more than a polynomial number of queries, each gate used in the lookup path will require error correction. If each gate requires error correction during routing, this will result in an exponential number of gates that will need to be activated, negating any of the performance gained from using the bucket-brigade model.

## 6.2 Encoding Classical Data

Encoding classical data in an efficient manner is another problem when implementing quantum machine learning algorithms. Quantum machine learning algorithms use data encoded into a superposition of states: when this data is read, due to the measurement principle, the superposition is destroyed, meaning that the data has to be re-encoded into a quantum state. One recent approach into a better encoding strategy, by Park et al. [23], proposed a flip-flop qRAM, which can transfer classical data and superimpose it into quantum basis states where the amplitudes encode the data. This scheme takes $O(N)$ qubits and $O(MN)$ steps to encode $M$ entries of classical data consisting of $N$ bits each. Moreover, since their implementation involves a single quantum circuit, they do not have to perform error correction similar to the bucket-brigade model. However, their qRAM is rendered unusable after accessing the data once. To circumvent this for specific use cases, they introduced the idea of quantum forking, where a qubit can be processed in a series of independent processes in superposition. The quantum forking process works especially well at calculating inner products, a kernel used frequently in machine learning, but is not very useful for other algorithms due to the fact that the qRAM state can not be reused.

# 7 Conclusion

In this review, we analyzed the recent developments in quantum neurons, which are the foundation of quantum neural networks. We found that the state of the field is optimistic since there exists a quantum neuron that can mimic a classical neuron using commonly used nonlinear activation functions. However, we note that there is still a long way to go to achieve an advantage over classical neurons, since no quantum algorithm has yet proposed a faster runtime than its classical equivalent at the level of a single step. That being said, there are efforts to reduce training time in other aspects, such as training on data in a superposition and figuring out which parameters to update using Grover's search. Thus, there is still a great deal of opportunity to make large contributions to the field.

# References


[1] Ashish Kapoor, Nathan Wiebe, and Krysta Svore. Quantum perceptron models. In *Advances in Neural Information Processing Systems*, pages 3999–4007, 2016.

[2] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), Oct 2009.

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

[4] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

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

[7] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[8] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[9] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6231–6239. Curran Associates, Inc., 2017.

[10] Alex Krizhevsky, I Sutskever, and G Hinton. Imagenet classification with deep convolutional neural. In *Neural Information Processing Systems*, pages 1–9, 2014.

[11] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[12] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S. Vetter. NVIDIA tensor core programmability, performance & precision. In *Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018*, 2018.

[13] Norm Jouppi. Google supercharges machine learning tasks with tpu custom chip.

[14] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[15] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[16] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Simulating a perceptron on a quantum computer. *Physics Letters A*, 379(7):660 – 663, 2015.

[17] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011.

[18] Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):26, 2019.

[19] Maria Schuld. Machine learning in quantum spaces. *Nature*, 567(7747):179–181, mar 2019.

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

[21] Vittorio Giovannetti, Seth Lloyd, and Lorenzo MacCone. Quantum random access memory. *Physical Review Letters*, 2008.

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

[23] Daniel K. Park, Francesco Petruccione, and June Koo Kevin Rhee. Circuit-Based Quantum Random Access Memory for Classical Data. *Scientific Reports*, 2019.