Benchmarking Neural Networks For Quantum Computations

Nam H. Nguyen and E.C. Behrman
Department of Mathematics and Physics, Wichita State University
Wichita, KS, USA 67260-0033

Mohamed A. Moustafa and J.E. Steck
Department of Aerospace Engineering,Wichita State University
Wichita, KS, USA 67260-0044

***Abstract*—The power of quantum computers is still somewhat speculative. While they are certainly faster than classical ones, the class of problems they can efficiently solve has not been mapped definitively onto known classical complexity theory, and there is a paucity of truly quantum algorithms. This is partly because finding algorithms that take advantage of the quantum nature of reality is very difficult. In previous work over the past three decades we have proposed, and developed, the idea of using techniques of machine learning to address this problem. Here we compare the performance of standard real- and complex-valued classical neural networks with that of one of our models for a quantum neural network, on both classical problems and on an archetypal quantum problem: the computation of an entanglement witness. The quantum network is shown to be considerably more powerful.**

***Index Terms*—quantum neural network, quantum machine learning, quantum computation, entanglement, complex neural network, benchmarking, complexity**

I. Introduction

In recent years there has been a growing interest in quantum computations, developing and deploying computations on quantum hardware, and performing quantum simulations and calculations using classical computers [1]. Often, these quantum problems have no closed-form or general solutions. The lack of overarching solutions to outstanding problems in quantum mechanics, coupled with the availability of a limited knowledge suggests an opportunity for a dynamically learning system to learn and approximate general solutions. Moreover, while the possibilities are easily imagined to be large, actual concrete results in terms of applications and algorithms are somewhat thin.

It has been known for twenty years [2] that quantum computers are at least as fast as classical computers, and widely believed that they are much faster, perhaps exponentially faster. A number of researchers have proved the existence of quantum codes in a variety of contexts [3]. But existence is not construction, much less optimization. There is a need for direct comparison, and benchmarking.

Here, we explore the power and the complexity of a quantum neural network computer (QNN), in both classical and quantum computations. We apply standard real- and complex-valued classical neural networks to well-understood classical tasks of computation (a single-layer perceptron), classification (iris identification), and function approximation (a sine). The complex-valued network, it will be seen, is considerably more powerful than a comparable real-valued network, at logic gates, at function approximation, and at pattern classification. A fully quantum mechanical network [4] (inherently complex valued) does even better.

More important, a QNN can efficiently do fully *quantum* calculations. Previously, we have shown [4] that a QNN of this type can successfully calculate a general experimental entanglement witness, applicable to mixed as well as pure states, which can be bootstrapped to multiqubit systems [5], and which is robust to noise and to decoherence [6]. We therefore compare also the performance of the classical neural nets to this problem. Our results show that a QNN is considerably more powerful than a classical neural network.

II. Neural Network Approaches

A. Classical real-valued neural networks

Even a classical, real-valued neural network (RVNN) is a "universal approximator". More precisely, what this means is as follows [7].

A real-valued neural network with a continuous sigmoidal activation function $\sigma : R^n \to R$ can be represented as

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma\big(w_j^T x + \theta_j\big) \tag{1}$$

where $x, w_j \in R^n$ are the inputs and weights, respectively; $\alpha, \theta \in R$ are the scaling factors and biases of the activation function, and where w_j^T represents the transpose of w_j.

It should be noted that σ is discriminatory. This means for a measure $\mu \in M(I_n)$

$$\int_{I_n} \sigma\big(w^T x + \theta\big) d\mu(x) = 0 \tag{2}$$

for all $x \in R^n$ and $\theta \in R$ implies that $\mu = 0$. Here $M(I_n)$ is the space of finite, signed, regular Borel measures on the unit interval, I_n. This requirement is to ensure that the activation can't send the affine space $w^T x + \theta$ to the set of measure zero.

Theorem: Let σ be any continuous sigmoidal function. Then finite sums of the form

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma\big(y_j^T x + \theta_j\big) \tag{3}$$

are dense in $C(I_n)$.

This theorem tells us that as long as the mapping between the inputs and targets can be described by a continuous function, $G : R^n \to R$, then for a real-valued neural network with just one hidden layer and any continuous sigmoidal function as the activation function there exist weights w, bias θ and scalar α to approximate this continuous map to any desired accuracy.

When the activation is a bounded, measurable sigmoidal function then [7] the finite sum $G(x) = \sum_{j=1}^{N} \alpha_j \sigma\big(y_j^T x + \theta_j\big)$ is dense in $L^1(I_n)$.

An (in house) implementation of a real-valued artificial neural network was used as a baseline. The software NeuralWorks Professional II [8] was also used to verify that the results obtained from commercially available software were comparable to our own implementation.

B. Complex-valued neural networks

We now extend from R^n to C^n. Clearly it ought to be true that if a calculation can be done in R^n it can be done in C^n, since the real valued neural network is a special case of the complex-valued neural net (CVNN). And it is true that the complex-valued net is a universal approximator also, though we need to be a little careful.

Theorem: Let $\sigma : C^n \to C$ be any complex continuous discriminatory function. Then the finite sums of the product of the form

$$F(z) = \sum_{k=1}^{m} \beta_k \prod_{l=1}^{s_k} \sigma\big(W_{kl}^T z + \theta_k\big) \tag{4}$$

are dense in $C(I_n)$.

Here the product is needed to ensure that the set $\tau = \big\{F : C^n \to C\big\}$ is an algebra since it will be closed under multiplication [9]. This is the difference between the real-valued neural network and complex-valued network approximation theorem. Similarly to the real-valued networks, if the activation function, $\sigma : C^n \to C$ is any bounded measurable discriminatory function then the finite sum $F(z) = \sum_{k=1}^{m} \beta_k \prod_{l=1}^{s_k} \sigma\big(W_{kl}^T z + \theta_k\big)$ is dense in $L^1(I_n)$.

In the complex valued case, the activation function may have essential singularities. But if $\sigma : C^n \to C$ is any complex function having an isolated and essential singularity then the finite sum above will be dense in compact subsets of analytic deleted neighborhood of the singularity.

This shows that there exists a representation of the complex valued neural network that can solve essentially any problem the real-valued net can do. Indeed, we expect that the complex-valued net will be more powerful or at least more efficient. The archetypal example is of course the XOR. Since it is a nonlinear problem, a real-valued net needs a hidden layer to solve, but the complex-valued net can do this in a single layer [10].

The implementation of the CVNN used here is largely based on the work of Aizenberg [10]. The major difference from the RVNN is that the neuron inputs, outputs, weights, and biases are numbers in the complex plane. In the RVNN signals are only scaled by their respective weights, while the CVNN allows the scaling and rotation of the signals. This leads to higher functionality in more compact architectures. Similar to a conventional RVNN, the input signals to any given neuron are summed by their respective weights:

$$z = \sum_n w_n \cdot x_n \tag{5}$$

where w and x are the complex-valued weights and signals. An activation function is then applied to the sum z at the neuron. The complex activation function P takes the inputs and maps them to a unit circle on the complex plane which constitutes the signal from the neuron to the consequent layer.

$$P(z) = e^{i \cdot arg(z)} \tag{6}$$

where as usual i refers to the imaginary unit. The network's scalar inputs and outputs/targets are condensed to the range [0,1] and mapped as points on the complex unit circle using the following mapping function:

$$M(r) = e^{i \cdot \pi r} \tag{7}$$

Unlike the RVNN, with this architecture there is no need to take the derivative of the activation function or to apply gradient descent in order for this network to learn. The resulting error from each training pair can be represented as the vector from the output to the target in the complex plane. In other words, given the target t and the output z of an output neuron, the error can be represented as the vector e

$$e = t - z = \sum_n \Delta w_n \cdot x_n \tag{8}$$

given that the resulting error is a contribution of all of the signals to the output neuron. Here Δw_n is the contribution of each individual weight to the error. Dividing the error equally among the N incoming weights we arrive at the following required weight adjustments for each training pair per epoch.

$$\Delta w_n = \frac{e}{N} \cdot x_n^{-1} \tag{9}$$

For each subsequent layer the error is "backpropagated" by dividing the error across the weights relative to the values of the signals they carry. Thus the error gets diluted as it propagates backwards, yet this is applicably manageable given that the higher functionality of the CVNN favors shallower networks for conventional problems.

C. Quantum Neural Network

Because the major goal here is the calculation of an entanglement witness, we consider the model for a QNN we first proposed in 2002 for this purpose [4]. The system is prepared in an initial state, allowed to evolve for a fixed time, then a (predetermined) measure is applied at the final time. The adjustable parameters necessary are the (assumed time-dependent) couplings and tunneling amplitudes of the qubits; the nonlinearity necessary is provided by the measurement, which we may take to be anything convenient without loss of generality. So, if we take the measurement whose output is $\in$ [0,1] (e.g., for the system of two qubits, A and B, the square of the qubit-qubit correlation function at the final time, $\langle \sigma_{zA}(t_f)\sigma_{zB}(t_f)\rangle^2$), we are considering the map

$$f : C^n \to [0,1] \tag{10}$$

This structure lends itself immediately to the perceptron problem (as well as the entanglement problem).

The network input is the initial state of the system, as given by the density matrix $\rho(0)$, then propagate it through time using Schrodinger's equation

$$\frac{d\rho}{dt} = \frac{1}{i\hbar}[H,\rho] \tag{11}$$

where H is the Hamiltonian. (Note: we work with the density matrix representation because it is computationally easier than using the state vector (or wave vector) representation. The density matrix form is more general, allowing mixed as well as pure states, but here we will in fact use only pure states. In terms of the state vector of the system $|\psi\rangle$, the density matrix is given by $\rho = |\psi\rangle\langle\psi|$.) For a two-qubit system, we take the Hamiltonian to be:

$$H = \sum_{\alpha=1}^{2} K_\alpha \sigma_{x\alpha} + \epsilon_\alpha \sigma_{x\alpha} + \sum_{\alpha\neq\beta=1}^{2} \zeta_{\alpha\beta}\sigma_{z\alpha}\sigma_{z\beta} \tag{12}$$

where $\{\sigma\}$ are the Pauli operators corresponding to each qubit, $\{K\}$ are the amplitudes for each qubit to tunnel between states, $\{\epsilon\}$ are the biases, and $\{\zeta\}$ are the qubit-qubit couplings. Generalization to a fully connected N qubit system is obvious. We choose the usual charge basis, in which each qubit's state is ±1, corresponding to the Dirac notation $|0\rangle$ and $|1\rangle$, respectively.

By introducing the Liouville operator, $L = \frac{1}{i\hbar}\left[\cdots, H\right]$, Eq. (11) can be rewritten as

$$\frac{\partial\rho}{\partial t} = -iL\rho \tag{13}$$

which has the general solution of

$$\rho(t) = e^{-iLt}\rho(0) \tag{14}$$

We propagate our initial state, $\rho(0)$, forward in time to the state $\rho(t_f)$ where we will perform a measurement on $\rho(t_f)$, which mathematically is an operator followed by a trace. That measurement, or some convenient mathematical function of that measurement, is the output of the net. That is, the forward operation of the net can be pictured as:

$$\rho(0) \xrightarrow{\frac{d\rho}{dt}=\frac{1}{i\hbar}[H,\rho]} e^{-iLt}\rho(0) = \rho(t_f) \to \mathrm{Trace}(\rho(t_f)M) = \langle M\rangle \tag{15}$$

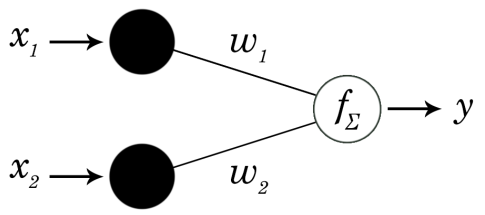


Fig. 1: Single-layer perceptron architecture

where M represents the particular measurement chosen. To train the net, the parameters K, ϵ and ζ are adjusted based on the error, using a quantum analogue [4] of Werbos's well-known back-propagation technique [11].

There is, clearly, no direct correspondence between these equations for the QNN with those of either the real- or complex-valued neural network, above. The nonlinearity necessary to do learning comes entirely from the (necessarily nonlinear) measurement step. We do not know exactly how the complexity of this model compares; work on these theorems is on-going [12].

However, as a first step, we can investigate the question experimentally, in what follows.

III. Benchmarking on Classical Problems

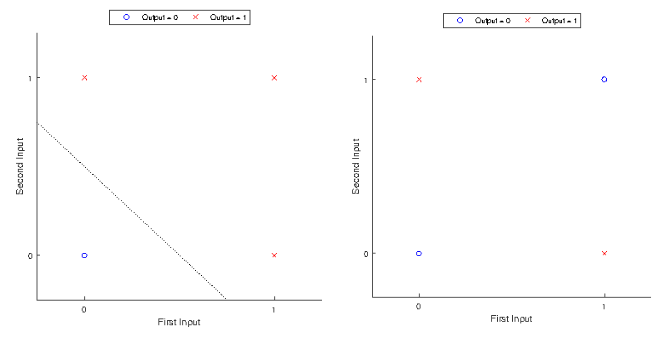
We compare the performance of our neural networks on three kinds of classical problems (logic gates, pattern classification, and function approximation).

A. Logic Gates

Perhaps the most basic of neural network architectures is the single-layer *perceptron*. See Fig. 1. The perceptron is a single output neuron connected to the input neurons by their respective weighted synapses. The real-valued perceptron is a linear classifier and therefore can approximate linearly separable logic functions such as AND and OR gates; however, because it essentially creates a single linear separator, it necessarily fails to approximate non-linear gates, like the XOR or XNOR. This point is made graphically in Fig.2: a straight line can separate points for which different outputs are desired. In contrast, a complex perceptron *can* calculate the XOR or XNOR, since it can create a nonlinear separation (or, equivalently, uses interference.)

The single layer perceptron was implemented on all four networks. The networks were trained on the 4 possible input pairs, {(00), (01), (10), and (11)}, for the outputs of the AND, NAND, OR, NOR, XOR and XNOR gates until the root mean square error (RMS) reached 1%. See Fig. 3. The learning rate of each network was increased to the "optimal", i.e, the training speed was maximized subject to the constraint that convergence was still achieved. This was done to draw a more equitable comparison on learning speed between the RVNN and CVNN, which have different learning schemes.

To implement logic gates on our QNN model, we take the inputs as characterizing the joint state of the two-qubit system,



(a) OR (b) XOR

Fig. 2: Visual representation of the linear separability of the OR gate versus the XOR gate. The dotted line represents linear classifier.

and the output measure as being the square of the qubit-qubit correlation function at the final time, i.e, $\langle\sigma_{zA}(t_f)\sigma_{zB}(t_f)\rangle^2$. Thus, for example, the input-output pairs for the XNOR gate are as shown in Table I. For this gate, we train the QNN qubits to be perfectly correlated to each other at the final time if the input state is either $|00\rangle$ or $|11\rangle$; if the input is either $|01\rangle$ or $|10\rangle$, the final state of the qubits is trained to be perfectly uncorrelated. Because of the inherent continuity of the QNN any initial state close to one of the input states will necessarily produce an final state close to the corresponding output state, and the computation is robust to both noise and decoherence [6].

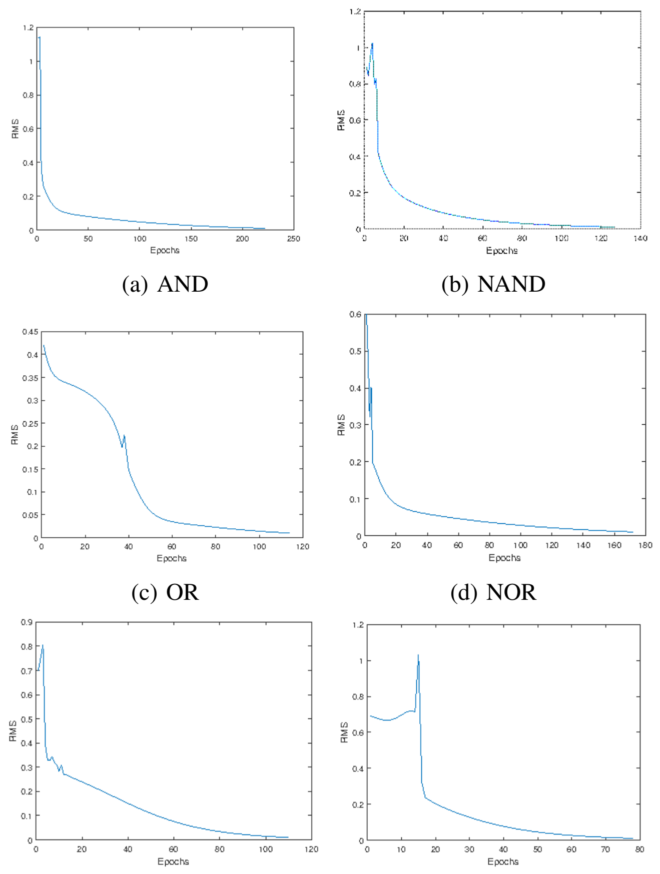
Inputs	Outputs
$\|00\rangle$	1
$\|01\rangle$	0
$\|10\rangle$	0
$\|11\rangle$	1

TABLE I: Inputs and Outputs for the QNN model of the XNOR gate. The input is the initial, prepared, state of the two-qubit system, at $t = 0$; the output, the square of the measured value of the qubit-qubit correlation function at the final time.

Table II shows the number of training epochs required by each "optimal" network to reach an RMS error of 1%, where an epoch is defined as one pass through the 4 training pairs.

The real-valued network would not train to an RMS error below 50% for the XOR and XNOR gates, given a number of epochs up to the order of 10^6. In addition, for the linearly separable gates the RVNN required 30-50 times more learning iterations than the CVNN to reach an RMS error of 1%, making the CVNN training runs computationally much more efficient than RVNN. Note that the single-layer complex-valued perceptron can learn to solve the non-linearly separable XOR and XNOR gates, and do so with an efficiency at least comparable to that for the linearly separable gates as shown in Table II, as mentioned above.

"Quantum-inspired" networks [13] generally are some version of a CVNN, and, therefore, do have increased power over a RVNN due to their ability to use interference. However



(a) AND (b) NAND (c) OR (d) NOR (e) XOR (f) XNOR

Fig. 3: Training RMS against training epochs for various logic gates using the (single layer) CVNN perceptron.

No. of Epochs to RMS error $\leq$1%				
Gate	N.Works	RVNN	CVNN	QNN
AND	11000	11146	222	1
NAND	11000	11145	127	1
OR	6000	5672	114	1
NOR	6000	5671	172	1
XOR	n/a	n/a	110	1
XNOR	n/a	n/a	78	1

TABLE II: Number of epochs needed for gate training to reach RMS error $\leq$1% for a (single layer) perceptron using RVNN, CVNN and NeuralWorks (to nearest 1000 epochs) implementations. Note that the nonlinear gates (XOR and XNOR) cannot be done by either of the real classical nets (RVNN and NeuralWorks) with only a single layer. The QNN reached $\leq$1% error in a single epoch.

a fully quantum network can do even better. Results of the training of the QNN for the linear and non-linear logic gates are shown in the last column of Table II. Note that the QNN requires only a single epoch to learn any of the linear or the nonlinear gates (the error continued to decrease below that level.) This is an experimental realization of our 2005 theoretical result deriving weights for a quantum perceptron [14].

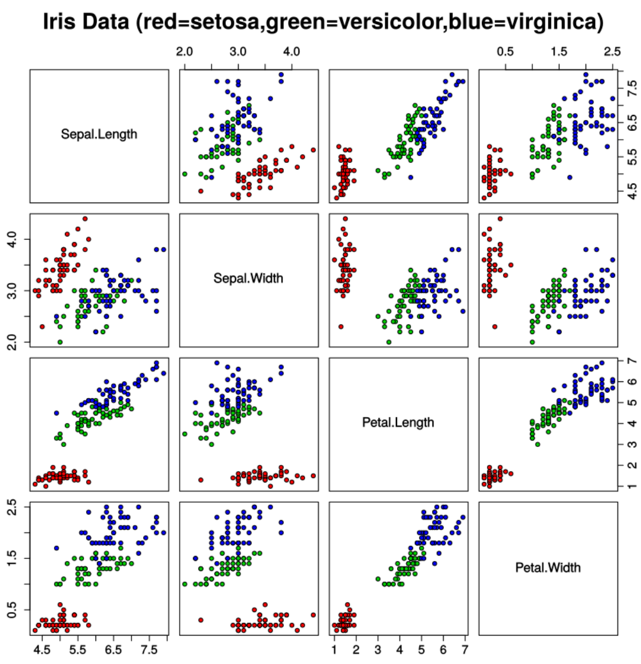


Fig. 4: Iris dataset cross-input classification scatterplot

B. Iris Classification

Another archetypal machine learning task is that of pattern classification, for which one of the most common benchmarking problems is the Iris flower classification dataset. The multivariate dataset contains four features (sepal length, sepal width, petal length, and petal width) as inputs and the corresponding taxonomy of Iris flower (setosa, virginica or versicolor) represented as a "one-hot" vector [15]. The set contains a total of 150 pairs, 50 of each species, as identified by Fisher in 1936 [16]. The difficulty in Iris classification is well demonstrated by the scatterplot shown in Fig. 4; two of the the three species cluster together and require a highly non-linear separability function.

The RVNN and CVNN implementations were trained on the dataset to compare their performance for a non-linear multivariate classification problem. The dataset was divided into two randomly selected sets of 75 pairs containing an equal number of each species for training and testing. The networks were first trained on the entirety of the training set. The training pairs were then reduced to 30 and 12 pairs while keeping the testing set at 75 pairs. Both the RVNN and CVNN had a single hidden layer, that is, an architecture (4, N_h, 3), where the number of neurons in the hidden layer, N_h, was increased from three on up. Unlike the RVNN, the CVNN's performance improved substantially as N_h was increased, up to about 100; these are the results reported in Table III.

For implementation on the QNN, we used the feature data as the coefficients of the input state $|\psi(0)\rangle$ in the Schmidt decomposition. That is, the initial quantum state vector for each input was given by

$$|\psi(0)\rangle = a|00\rangle + be^{i\theta_1)}|01\rangle + ce^{i\theta_2}|10\rangle + de^{i\theta_3}|11\rangle \quad (16)$$

We set all the phase offsets $\{\theta_i\}$ equal to zero (i.e., only real inputs here), and take each amplitude to correspond to a features (i.e., a, b, c, d correspond to sepal length, sepal width, petal length, and petal width.) For the output, we again choose the final time correlation function, taken to be the polynomial $0.01(a^2 + b^2) + c^2 + 1.8d^2$. Most of the separation can be accomplished with only the petal data; with this function, almost complete separation is achieved. Results are shown in Table III.

On this classification problem the advantage of the CVNN to the RVNN is not nearly as pronounced. The RMS training errors for the CVNN are consistently below those of the RVNN, but the testing errors are not. In terms of the classification the CVNN does do better, but not as significantly as with the perceptron. The reason the classification improves while the RMS error does not is that the CVNN tends to a flatter output: thresholding gives the "correct" answer, but the separation between classification of "yes" (i.e., a value near 1) and "no" (i.e. a value near zero) is smaller with the CVNN.

Performance in training, testing, and classification by the QNN is also comparable, to both nets. The major difference among the three is in the amount of training necessary to achieve the results. The RVNN required 50,000 epochs, while the CVNN required only 1000, and the QNN only 100. The ratios of iterations necessary is remarkably similar to the results for the perceptron, shown in Table II. In this sense we can say that the QNN is significantly more powerful, at classification as well as at computation.

C. Sine Approximation

Yet another standard benchmarking task is function approximation. We used simply the basic sine function to compare the real-valued and complex-valued neural network implementations and to evaluate their ability to generalize from limited datasets. Similarly sized networks with a single hidden layer of 5 neurons were implemented on the RVNN, CVNN and NeuralWorks software. The Sine training and testing sets were randomly generated and the pairs were bound to the domain of zero to 2π. The networks were trained and tested on two sets of 100 pairs each for 10^3, 10^4 and 10^5 epochs and the RMS values were recorded in Table IV. The testing set was kept at 100 pairs to represent an adequately large sample of the selected domain. Results are shown in Table IV. Note that the RMS error values are presented as a percentage of the maximum possible error, the function range, and that the presented values are the cross-run averages for five separate training/testing runs of each test case.

The results show that the CVNN was consistently able to reach a training and testing RMS error comparable to, or less than, that of the real-valued networks in an order of magnitude fewer training epochs. This confirms the aforementioned increase in speed exhibited while benchmarking the networks on logic functions, and extends the finding to larger network architectures. It was also observed that the deviation between the training and testing percentage RMS errors for the CVNN was larger than that for the real-valued networks: where the difference averaged 0.214 as a fraction of the training RMS

	Training RMS (%)			Testing RMS (%)			Testing Accuracy (%)		
Training Pairs	RVNN	CVNN	QNN	RVNN	CVNN	QNN	RVNN	CVNN	QNN
75	3.45	2.06	0.96	3.71	7.78	2.31	100	100	97.3
30	0.51	0.41	1.1	4.97	9.47	9.78	93.3	96.0	97.5
12	0.69	0.09	0.62	11.9	16.4	7.48	85.3	94.7	85.5

TABLE III: Iris dataset training and testing average percentage RMS and identification accuracy for different training set sizes using similar RVNN and CVNN networks, and compared with the QNN. The RVNN was run for 50,000 epochs; the CVNN, for 1000 epochs; and the QNN, for 100 epochs.

	Training RMS (%)			Testing RMS (%)		
Epochs	N.Works	ANN	CVNN	N.Works	RVNN	CVNN
1000	12.67	10.16	5.00	12.83	10.95	6.64
10000	9.97	4.74	2.19	10.43	4.90	2.52
100000	2.86	2.31	1.54	2.92	2.38	1.79

TABLE IV: Average percentage RMS for Sine function training and testing sets for different numbers of epochs using similar Neural Works, RVNN, and CVNN networks.

error for the CVNN compared to 0.026 for Neural Works and 0.047 for RVNN. This suggests that the CVNN may be "overfitting" the training data compared to the real-valued classical networks, and possibly an indicator it may attain even better performance and generalizability with a smaller architecture.

The QNN is in a sense "tailor-made" for the sine function approximation: with the coupling and field variables set equal to zero, the qubit-qubit correlation function is in fact exactly the cosine function [17]; thus, the QNN's ability to reproduce this particular function is trivial, andnot evidence of quantum complexity or efficiency.

IV. Quantum computation problem

We now consider our networks' performance on a purely quantum calculation: an entanglement witness.

A. Benchmarking with entanglement

An outstanding problem in quantum information is quantifying one of the key elements responsible for the powerful capabilities of quantum computational systems: entanglement. Entanglement is the feature whereby two or more subsystems' quantum states cannot be described independently of each other, that is, that the state of a system AB cannot be correctly represented as the (state of A)$\otimes$(state of B), where $\otimes$ indicates the outer product. Non-multiplicativity, or entanglement, of states, is fundamental to quantum mechanics. Indeed, all of quantum mechanics can be derived from this characteristic; in that sense, entanglement is equivalent to non-commutativity of operators, or to the Heisenberg Uncertainty Principle.

Even though quantifying entanglement has been a prominent subject of research, there is no accepted closed-form representation that is generalizable to quantum states of arbitrary size or properties. Because a quantum state is unentangled if it is separable into its component states, entanglement *measures* are used to determine the "closeness" of a quantum state to the subspace of separable product states. Most of the developed measures involve a subset of possible compositions or require extremizations which are difficult to implement [19], especially for large systems. One well-known measure is the entanglement of formation which quantifies the resources utilized to construct a specific quantum state. For a two-qubit system we have an explicit formula for the entanglement of formation [18], but generalizations to larger systems are difficult.

The lack of a general analytic solution combined with a base of knowledge on specific sets of states suggests an opportunity for a machine learning approach. Given that the entanglement is an inherent property of the quantum mechanical system, that system in a sense "knows" what its entanglement is. Thus, if we are clever, we can devise a measurement or measurements that will extract this information, and, it is hoped, somewhat more efficiently than complete quantum tomography would require. It is a theorem [4] that a single measurement cannot determine the entanglement completely; however, one measurement can serve as a "witness": that is, a measurement giving a lower bound to the correct entanglement.

Using a training set of only four pure states, we successfully showed [4] that this model of a QNN can "learn" a very general entanglement witness, which tracks very well with the entanglement of formation [18], and which not only works on mixed as well as pure states but can be bootstrapped to larger systems [5] and, most importantly, is robust to noise and to decoherence [6].

Of course, any computation done by a quantum system can be simulated. How complex is this calculation? For the two-qubit system, this is fairly easy to characterize. Note that for the time-propagation part of the computation, each step is linear. Thus, if we have a two qubit system Eq. 15 can be written out as follows. Let M be the 4x4 matrix $M = \{m_{ij}\}$, let the time propagator $T = e^{-iLt}$ be $T = \{T_{ij}\}$, and let $\rho(0) = \{\rho_{ij}\}$.

Then $\rho(t_f) = T\rho(0) =$

$$\begin{bmatrix} \sum_{j=1}^4 T_{1j}\rho_{j1} & \sum_{j=1}^4 T_{1j}\rho_{j2} & \sum_{j=1}^4 T_{1j}\rho_{j3} & \sum_{j=1}^4 T_{1j}\rho_{j4} \\ \sum_{j=1}^4 T_{2j}\rho_{j1} & \sum_{j=1}^4 T_{2j}\rho_{j2} & \sum_{j=1}^4 T_{2j}\rho_{j3} & \sum_{j=1}^4 T_{2j}\rho_{j4} \\ \sum_{j=1}^4 T_{3j}\rho_{j1} & \sum_{j=1}^4 T_{3j}\rho_{j2} & \sum_{j=1}^4 T_{1j}\rho_{j3} & \sum_{j=1}^4 T_{3j}\rho_{j4} \\ \sum_{j=1}^4 T_{4j}\rho_{j1} & \sum_{j=1}^4 T_{4j}\rho_{j2} & \sum_{j=1}^4 T_{4j}\rho_{j3} & \sum_{j=1}^4 T_{4j}\rho_{j4} \end{bmatrix}.$$

Therefore,

$$\left(Tr[M\rho(t_f)]\right)^2 = \left(\sum_{l=1}^4 \left[\sum_{k=1}^4 \left(m_{lk}\sum_{j=1}^4 T_{kj}\rho_{jl}\right)\right]\right)^2 \quad (17)$$

Eq. 17 forms a second order polynomial in terms of the input elements. It is easy to show [20] that the entanglement of formation for a two qubit pure state can be rewritten as a quadratic also:

$$E(|\psi\rangle) = 4a^2d^2 + 4b^2c^2 - 8abcd\cos(\theta_3 - \theta_2 - \theta_1) \quad (18)$$

where the (real) numbers $\{a, b, c, d, \theta_1, \theta_2, \theta_3\}$ are defined in Eq. 16. So it should not be altogether surprising that the QNN can learn an entanglement indicator for the two-qubit pure system. But clearly this function, according to the theorems in section II, ought to be able to be approximated by both the RVNN and the CVNN, especially for the relatively simple case where all the phase offsets, $\{\theta_i\}$, are zero. As we will see, however, this turns out to be no easy task for either to learn.

B. Results

In our previous work [4] we saw that the QNN was able to learn entanglement for a two-qubit system from a training set of only four pure states. It should also be noted that the QNN had also successfully generalized to mixed states, as well. Here we train, and test, only on pure states; nevertheless, neither the real- nor complex-valued classical net was able to accomplish this. See Tables V and VI. Each network was given the entire $4x4$ density matrix of the state as input. The classical nets again were given a hidden layer, and a single output. For the minimal training set of four, all three classical nets trained below 1% error. However the testing error was quite bad. Increasing the size of the training set did enable marginally better fitting by all the classical networks, but testing error remained an order of magnitude larger than with the QNN. Increasing N_h did not help. Note that increasing the size of the training set affected training and testing very little for the QNN, which had already successfully generalized with a set of only four. Neither the NeuralWorks nor the RVNN could learn pure state entanglement well, even with a training set as large as 100. And while the ("quantum-inspired") CVNN could use interference, it did not generalize efficiently or well, either. This is not surprising when one considers that it is precisely access to the entangled part of the Hilbert space that the CVNN lacks; that is, the state of CVNN cannot itself be entangled. In that sense it is inherently classical.

	Training RMS error (%)			
Training Pairs	N.Works	RVNN	CVNN	QNN
100	5.66	3.74	0.97	0.04
50	5.96	5.89	0.53	0.09
20	6.49	4.97	0.04	0.2
4	0.00	0.93	0.01	0.2

TABLE V: Training on entanglement of pure states with zero offsets, for the NeuralWorks, RVNN, CVNN, and QNN. The architectures used for the NeuralWorks, RVNN, and CVNN were 16 (input layer), 8 (hidden layer), 1 (output layer). Numbers of epochs were 5 to 10 thousand for the real valued nets; 1000 for the CVNN, and only 20 for the QNN. Again we note the increase in efficiency of training in going from RVNN to CVNN, and in going from CVNN to QNN.

	Testing RMS error(%)			
Training Pairs	N.Works	ANN	CVNN	QNN
100	7.56	5.39	3.61	0.2
50	7.91	10.7	6.00	0.3
20	13.6	15.5	9.48	0.4
4	48.2	51.9	55.0	0.4

TABLE VI: Testing on entanglement of pure states. Each network was tested on the same set of 25 randomly chosen pure states with zero offset.

V. Conclusions and Future Work

The marriage of quantum computing and machine learning can be enormously beneficial to both subfields: with machine learning techniques we can find "algorithms" [4], do automatic scale-up [5], and increase robustness significantly [6]; while with quantum systems and their access to the enormously greater Hilbert space, machine learning can be both more efficient and more powerful [4]. In this paper we provide evidence of both. We have shown, here, that a fully quantum neural network can do standard classical benchmarking tasks more efficiently than either a real-valued or a complex-valued classical net. Moreover, a fully QNN can do *quantum* calculations efficiently, that is, without explicit simulation. In any physical implementation there will also be additional, natural, nonlinearities we can take advantage of, and, possibly, control and use for machine learning.

There is currently a lot of good research being done on "quantum-inspired" machine learning [13]. Certainly it is of great value to tease out exactly whence the efficiency, and the complexity, of quantum computing - and specifically quantum neural computing - arise. One way of doing this is precisely to do "quantum-inspired" calculations: to allow, e.g., the state of the system to be complex-valued in some sense. Clearly without allowing full superposition this will not be fully quantum mechanical, because it will not include entangled states. There are many more states that are at least partially entangled than there are classical states (in the sense of being product states of subsystems.) But fully quantum mechanical calculations can quickly become dauntingly enormous, so the question is, how much can we get from how little?

There are, of course, many more kinds of machine learning tasks that we did not explore here. It will be of great interest to see what kinds of problems can benefit significantly from a fully quantum mechanical approach. It is not at all obvious what these are. And while our earlier results [6], [20] seem to imply that the calculations here would be similarly robust, we have not (yet) shown this. There is much still to do.

Acknowledgment

The authors would like to thank William Ingle for helpful discussions.

References

[1] M. A. Nielsen and I. L. Chuang (2001), *Quantum Computation and Quantum Information*, Cambridge University Press (Cambridge, England).

[2] E. Bernstein and U. Vazirani (1997) *Quantum complexity theory*, SIAM J. Comput. **26**, pp 1411-1473.

[3] See, e.g, A.R. Calderbank and P. Shor (1996) *Good quantum error correction codes exist*, Phys. Rev. A **54**, pp 1098-1105; G. Ortiz, J. Gubernatis, E. Knill, and R. Laflamme (2001), *Quantum algorithms for femionic simulations* Phys. Rev. A **64** 022319.

[4] E.C. Behrman, V Chandrashekar, Z. Wang, C.K. Belur, J.E. Steck, and S.R. Skinner (2002), *A quantum neural network computes entanglement*, arXiv: quant-ph/0202131; E.C. Behrman, J.E. Steck, P. Kumar, and K.A. Walsh (2008), *Quantum algorithm design using dynamic learning*, Quantum Inf. Comput. **8** pp. 12-29; E.C. Behrman, R.E.F. Bonde, J.E. Steck, and J.F. Behrman (2014), *On the correction of anomalous phase oscillation in entanglement witnesses using quantum neural networks*, IEEE Trans. on Neural Networks and Learning Systems **25**, pp 1696-1703.

[5] E. C. Behrman and J. E. Steck (2013), *Multiqubit entanglement of a general input state*. Quantum Inf. Comput. **13**, 1-2, pp. 36-53.

[6] E.C. Behrman, N.H. Nguyen, J.E. Steck, M. McCann (2016), *Quantum neural computation of entanglement is robust to noise and decoherence*, in Quantum Inspired Computational Intelligence: Research and Applications, S. Bhattacharyya, ed. (Morgan Kauffman, Elsevier) pp. 3-33.

[7] G. Cybenko (1989) *Approximation by superpositions of a sigmoidal function*, Math. Control Signal Systems **2**, pp. 303-314.

[8] Neuralware (2000), *Getting started: a Tutorial in NeuralWorks Professional II/Plus*

[9] Taehwan Kim and Tlay Adali (2001), *Complex backpropagation neural network using elementary transcendental activation function* Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 7-11 May 2001 **2** pp. 1281-1284; Taehwan Kim and Tlay Adali (2003)*Approximation by fully complex multilayer perceptrons*, Neural Comput. **15** pp. 1641-1666. DOI=http://dx.doi.org/10.1162/089976603321891846

[10] I. Aizenberg (2011), *Complex-Valued Neural Networks with Multi-Valued Neurons*, Berlin-Heidelberg: Springer.

[11] Paul Werbos, in *Handbook of Intelligent Control*, Van Nostrand Reinhold, p. 79 (1992); Yann le Cun, *A theoretical framework for back-propagation* in *Proc. 1998 Connectionist Models Summer School,* D. Touretzky, G. Hinton, and T. Sejnowski, eds., Morgan Kaufmann, (San Mateo), pp. 21-28 (1988).

[12] N.H. Nguyen (2019), *Complexity and power of quantum neural networks*, Ph.D. Thesis, Wichita State University (in progress.)

[13] See, e.g, S. Dey, S. Bhattacharyya, and U. Maulik (2014), *Quantum inspired genetic algorithm and particle swarm optimization using chaotic map model based interference for gray level image thresholding*, Swarm and Evol. Comput. **15** pp 38-57; S. Dey, S. Bhattacharyya, and U. Maulik (2017), *Efficient quantum inspired meta-heuristics for multi-level true colour image thresholding*, Appl. Soft Comput. **56** pp 472-513; and references cited therein.

[14] K.-L. Seow, E.C. Behrman, and J.E. Steck (2015), *Efficient learning algorithm for quantum perceptron unitary weights*, arXiv:1512.00522

[15] D. Harris and S. Harris (2012) *Digital design and computer architecture (2nd ed.)*, San Francisco, Calif.: Morgan Kaufmann, p. 129.

[16] R. A. Fisher (1936), *The use of multiple measurements in taxonomic problems* Ann. Eugenics **7**, pp 179188.

[17] E.C. Behrman, G.A. Jongeward, and P.G. Wolynes (1983), *A Monte Carlo approach for the real time dynamics of tunneling systems in condensed phases* J. Chem. Phys. **79**, 6277-6281.

[18] W.K. Wootters (1998), *Entanglement of Formation of an Arbitrary State of Two Qubits* Phys. Rev. Lett. **80**, 2245.

[19] V. Vedral, M.B. Plenio, M.A. Rippin, and P.L. Knight (1997) *Quantifying entanglement* Phys. Rev. Lett. **78**, pp. 2275-2279 (1997); S. Tamaryan, A. Sudbery, and L. Tamaryan (2010), *Duality and the geometric measure of entanglement of general multiqubit W states* Phys. Rev. A **81** 052319

[20] N.H. Nguyen, E.C. Behrman, and J.E. Steck (2016), *Robustness of quantum neural calculation increases with system size*, arXiv:1612.07593

Nam H. Nguyen earned a B.S in Mathematics from Eastern Oregon University, and an M.Sc in Applied Mathematics from Wichita State University. He is currently a Ph.D candidate in applied mathematics at Wichita State University under the supervision of professor E.C. Behrman. His research interests and publications are in the area of quantum neural networks and machine learning, partial differential equations, and graph theory. He has done research work in inorganic and physical chemistry as well.

E.C. Behrman earned an Sc.B from Brown University in mathematics, and an M.S. in chemistry and Ph.D. in physics from University of Illinois at Urbana-Champaign. She did postdoctoral work at SUNY Stony Brook, and worked for four years at NYS College of Ceramics at Alfred University. She is currently professor of mathematics and of physics at Wichita State University. Her research interests and publications are broad, with over 80 papers in subjects ranging from chemical kinetics and reaction pathways to ceramic superconductors to nuclear waste vitrification. She was the first to predict the stability of inorganic buckyballs and buckytubes, and among the first to design and computationally test models for quantum neural networks and machine learning.

Mohamed A. Moustafa holds a B.S. in Aerospace Engineering from Wichita State University and B.S. in Physics from Emporia State University. His research interests include machine learning, quantum computations, aviation and space applications; he particularly enjoys bridging the gap between academia and software development. He has contributed to research on aircraft loss-of-control prediction and developed pilot-assistive warning displays using augmented reality devices at Wichita State University. He has experience implementing various machine learning models including GA for solving the facility layout problem and various implementations of artificial neural networks. Mohamed is currently an Applications and Front-End engineer at a cloud HPC (High Power Computations) company based in San Francisco, CA.

J.E. Steck has a B.S. and M.S. in Aerospace Engineering and a Ph.D. in Mechanical Engineering from the University of Missouri-Rolla where he performed research in the use of finite element methods in mechanics, fluids and aero-acoustics. He has done postdoctoral work in artificial neural networks as control systems for the Intelligent Systems Center at the University of Missouri-Rolla, and worked for two years at McDonnell Aircraft Company in the Aerodynamics department doing flight dynamics support for flight simulation, wind tunnel and flight testing of the AV-8B aircraft. He is currently professor of Aerospace Engineering at Wichita State University where he has taught for twenty-three years. He teaches undergraduate and graduate courses in flight dynamics and controls, artificial neural networks, and computational methods. His current work includes: intelligent adaptive control systems for general aviation aircraft, integrated aircraft structural health monitoring, quantum neural computing, optical aircraft ice detection, optical neural computing, and the use of artificial neural networks for system modeling and control.