Quantum Machine Learning Classifier

Avery Leider, Gio Abou Jaoude, Abigail E. Strobel, and Pauline Mosley, DPS

Seidenberg School of Computer Science and Information Systems, Pace University, New York, USA,
{al43110n, ga97026n, as66693n, pmosley}@pace.edu

Abstract. This Quantum Machine Learning Classifier (QMLC), uses the mathematics of quantum computing in a deep neural network to find and classify the specific flower type of the three different iris flower species: Versicolor, Setosa and Virginica, utilizing the SciKit-Learn dataset "Iris." In that dataset, there are four characteristic features of each iris type: petal length, petal width, sepal length, and sepal width. The quantum computing machine learning classifier out-performed the classical deep learning neural network methods. Significant is that this classifier trained in fewer epochs.

Keywords: Quantum Computing, Machine Learning, Deep Learning, Quantum Machine Learning Classifier, Quantum Computing Mathematics

1 Introduction

Those new to the study of quantum computing find the literature complex because of the many different variants of quantum computing. One variant is using the mathematics of quantum computing to achieve a result. These mathematics are often accomplished on a classical ordinary machine, possibly a high performance computer or a cloud system, and these are not automatically considered a quantum simulator. What they do to be considered "quantum computing" is that they exploit the principles of quantum mechanics during the process of computation. This can be done by using the mathematics expressed in quantum mechanics or the subatomic particles that exhibit quantum mechanical properties.

Another variant is using quantum computing simulators. These platforms are designed for programs using quantum mathematics but add realism by including error. Quantum computers today are probabilistic: as there is always error in these systems, it takes hundreds or thousands of attempts to run the same program to find the probability of a specific output. For a quantum simulator to be useful, it must include error into the calculations.

Then, we have real quantum computers, which is using subatomic or atomic particles to do the computing. Most systems today are not actually purely quantum, but rather a hybrid of classical machines that do some of the calculations and store results, and the separate part which is actually where the quantum bits or "qubits" are calculated.

In the deep learning quantum machine classifier described here, the first variant is used, using the mathematics of quantum computing, expressed in linear algebra, to achieve our deep learning neural network result. We used it first on the cloud computing platform of Google Colab to write and test the program. Then, we used a high performance computer, the AiMOS supercomputer, to run the same Python script. The use of quantum computing mathematics in deep learning improves the classical approach by a significant margin.

The next section is our Literature Review, followed by a section on our Methodology, followed by our Investigations and Findings.

2 Literature Review

Classical computers can today both produce and recognize patterns in data with neural networks. There is a hope that quantum computing methods can also both recognize and produce statistical patterns. These would be more complex and counter intuitive because of quantum mechanics and would be those that are computationally difficult for a classical computer [1]. Small quantum computers can produce such patterns, and the work now in machine learning is to train them to recognize them. The hope is to find quantum algorithms for machine learning that can deliver quantum speedup from classical computers. A quantum algorithm is a set of instructions solving a problem that can be performed on a quantum computer.

Exponential speedups have been predicted for implementing least squares fitting [2], quantum Boltzmann machines [3], quantum principal components analysis [4], and quantum support vector machines [5]. These are still theory.

Quadratic speedups have been theoretically demonstrated for Bayesian inference [6], online perceptron [7], classical Boltzmann machines [8], and quantum reinforcement learning [9]. These are also still theory.

All of these speedups need a low-error rate, universal, quantum computer with hundreds to thousands of qubits. In addition, the speedup of some of these algorithms requires quantum RAM, which would enable a quantum mapping of a classical vector into a quantum state, but this does not currently exist. These research publications cited are all written before 2019, which was a landmark year in which IBM realized actual working primitive quantum computers and made them available for researchers for free on the Internet. Since that time, there has been a gap, as the possibility of creating something from all that promise is a challenge made closer and also more difficult in its steep concreteness.

The promise of new quantum algorithms for an exponential speed-up for machine learning, involving clustering and finding patterns in big data with quantum computing, may in part be hyperbole. However there are currently only a few real quantum algorithms: Grover's primitive search algorithm, Shor's algorithm for breaking public-key cryptography (currently unrealized due to lacking a large enough quantum computer), and the idea that we could accelerate the simulation of quantum physics and chemistry with other undiscovered quantum algorithms, suggested by Richard Feyman [10].

However, the desire to find solutions drives researchers to push past the hyperbole into new areas, such as Kathuria et al.'s [11] efforts to use quantum machine learning to find the presence or absence of Alzheimer's disease genes in human genomes, which is a difficult problem, as they have encoded their massive data into strings of 0s and 1s that are expressed in blocks of 64 digits to ensure non-overlapping windows of data elements.

This quantum circuit used in the quantum computing classifier extends the work of William Cappelletti of Entropica Labs [12] as seen in Figure 7. Important distinctions from Cappelletti's work are the number of gates, the use of expectation values rather than measurements and the use of a gradient [13]. The Cappelletti circuit uses a method of optimization called Constrained Optimization by Linear Approximation (COBYLA). A distinction between COBYLA and gradient descent algorithms is that COBYLA is considered to be slower (in number of operations) in exchange for COBYLA requiring fewer weights for optimization. A COBYLA optimized classifier is not equivalent to a gradient descent based classifier.

2.1 Review of Quantum Computing Definitions

Quantum computing is a cross-section of computer science and quantum mechanics, a nascent science that is exploding with new terms and discoveries. To ensure the readers understand the quantum machine learning classifier that is our research contribution, included here is a section reviewing the terms of quantum computing that are directly relevant. This covers quantum bits ("qubits") and the two specific gates, the $CNOT$ gate, $CZNOT$ and the R_X gate, used in this quantum circuit for the quantum machine learning classifier.

Qubit: The qubit is the quantum computing analog of a classical bit represented as a vector of two numbers. The two numbers can be represented as a vector in three dimensional space. A qubit is represented as a wire in the graphical representation of a circuit.

Dirac notation: The Dirac notation as described first by [14] is the standard notation for quantum computing by which vectors are represented through the use of $\langle Bras|$ and $|Kets\rangle$ to denote, respectively, column and row vectors.

Gate: A gate is an operation performed on a qubit. Mathematically the gate is described as a matrix that changes the values of a qubit. This value change corresponds with a rotation in three dimensional space. The degree of the rotation can be a function of an input value. In this case, the gate was parameterized with that value (called a parameter).

Circuit: A circuit is an ordered list of operations, gates, on a set of qubits to perform computational algorithms.

Bloch Sphere: The Bloch Sphere is the three dimensional representation of the possible orientations a qubit can have with a radius of one as described in the lectures of Felix Bloch [15]. It is analogous to the unit circle.

Expectation Value: The expectation value is the probabilistic value of a circuit.

Neural Network: A neural network is a learning algorithm designed to resemble the neural connections in the human brain.

Neuron: A neuron is the basic building block of a neural network. Connected neurons have weights to stimulate predetermined activation functions.

Weight: A weight can be thought of as the value denoting the strength of the connection between two neurons. It transforms the output signal of a neuron before it is fed into another neuron in the next layer.

AiMOS: AiMOS is a super computer at the Rensselaer Polytechnic Institute Center for Computational Innovations [16].

Backpropagation: The algorithm by which a neural network decreases the distance between the output of the neural network and the optimal possible output of the same neural network. Backpropagation is an amalgam of the phrase "backward propagation."

Epoch: Epoch is one iteration of training.

Training Set: The subset of data used to train the neural network.

Testing Set: The set of data used to verify the accuracy of the trained neural network.

R_X Gate: The R_X gate causes a rotation of a qubit about the X-axis to a degree specified by a parameter. The angle of rotation is specified in radians and can be positive or negative.

$$R_X(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

CZ Gate: The C_Z gate, controlled-Z gate, causes a rotation in a target qubit on the Z-axis based on the control qubit's position on the Z-axis.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

CNOT Gate: The CNOT gate, CX gate, causes a rotation in a target qubit based on the value of a control qubit.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

3 Objective and Contribution of this Research

The objective of this research is to find a quantum computing machine learning classifier that can surpass classical machine learning models in number of operations needed for training. This would be a type of "speedup" - though not necessarily the theoretical quadratic or exponential speedups referenced in the Literature Review section.

The contribution of this research is a quantum machine learning classifier. It uses quantum computing mathematics to classify data using deep learning.

4 Methodology

The development environment was Google Colab Jupyter Notebooks. The dataset used was the Iris Dataset from SciKit Learn. Google Colab allowed for comments and visualizations that helped open up understanding of how the neural network was operating as it searched through the data looking for characteristic features to make its classification decisions. First a benchmark program was written of a classical deep learning neural network, and the dataset run through that benchmark program. Then the quantum machine learning classifier program was written and the dataset run through the Quantum Machine Learning Classifier (QMLC). The results of the QMLC were significantly better, finding the correct classification in fewer epochs. The code was then downloaded from Colab in the .py file format and run on the AiMOS supercomputer, then the output file interpreted with a visualization program to see and compare results in an easy fashion. The results from the AiMOS computer were achieved faster, and the output was same as those from Colab, which helped validate our results.

We used the Pennylane library for our program. PennyLane is designed for quantum machine learning, using training of variational quantum circuits, as a type of differential quantum programming, an quantum analog to TensorFlow and PyTorch for classical computation. We imported the Python code libraries of numpy, pandas, seaborn and plotly. These were for, respectively, processing numbers, arrays, data plot visualizations and online graphing. We also imported matplotlib.

For our dataset, we imported from sklearn.dataset load_iris , as well as the preprocessing package of StandardScaler, the model selection of train_test_split and from the sklearn.metrics code library, we imported the confusion_matrix, accuracy_score, precision_score, recall_score and f1_score.

The Iris dataset [17] consists of 150 measurements taken by Edgar Anderson to quantify the variations in iris flower species. These measurements consist of four attributes: sepal length, sepal width, petal length, and petal width of the three species as seen in Figure 1. The irises consisted of three species: Versicolor, Setosa and Virginica. In the work displayed in this paper, each species is referred to by a target number: Veriscolor is target zero, Setosa is target one, and Virginica is target two. The statistics of the Iris dataset are seen in Figure 2. The pair plots of the attributes in the Iris dataset are seen in Figure 3. These are the visualization plots of the dataset to compare with our benchmark classical neural network classifier as well as the novel quantum classifier presented in this paper. The goal of the classifiers is to correctly classify a row of measurements as either target zero, one, or two correctly.

Figure 4 shows a correlation matrix that illustrates the relationships between the measured characteristics in the dataset. Red values are positive correlations and blue are negative correlations. Zero is a neutral relationship. This correlation

Fig. 1. Versicolor, Setosa and Virginica [17]

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Fig. 2. Statistics of Iris Dataset

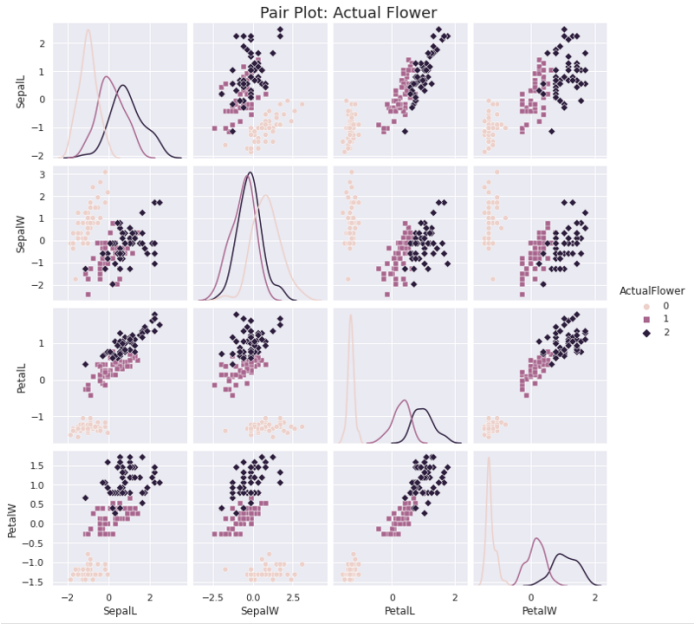

Fig. 3. Pair Plots of Attributes of the Iris Dataset

matrix is showing how strongly one characteristic feature attribute is directly correlated to another. The data for this matrix is standardized, by bounding them between the real numbers of -1 and positive 1. A positive number (signified by a red square) means that if one attribute is present in a certain degree, then the second attribute on the corresponding row or column is also present to that degree with the likelihood of that standardized value. Otherwise, there is a negative correlation.

5 Investigation and Findings

5.1 The Classic Deep Learning Neural Network

The metrics of the classifier were measured after every backpropagation of the entire neural network. This occurs after training on every data point in every epoch. No testing set data was used for training. The testing set is meant to gather metrics only. In Figure 5 the blue line that denotes accuracy is plotted over the recall as they are equivalent in every measurement taken. The highest accuracy reached in the 13 epochs on the classical deep learning neural network was 89%.

The weights of the classifier were also recorded after every backpropagation. Observing a few of the weights closely (Figure 6) shows that patterns appear in the repeated fluctuations of weight values. This is a direct result of gradient

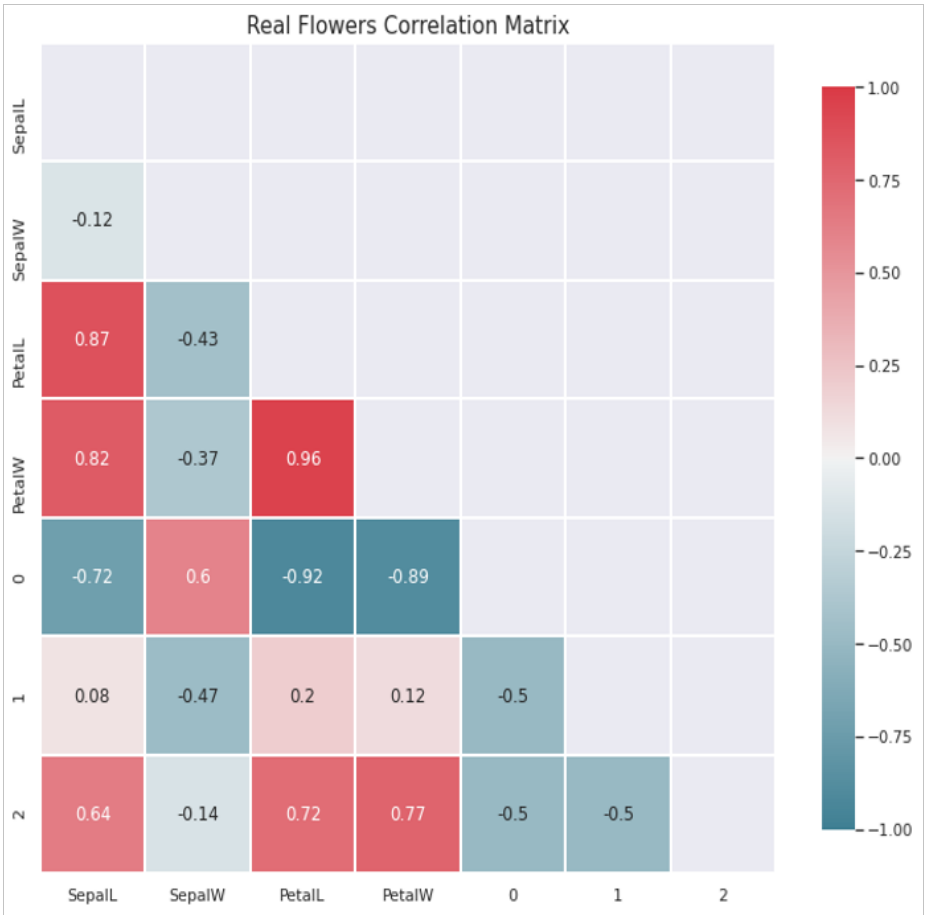

Fig. 4. Correlation Matrix of Iris Dataset

descent causing small changes to the weights in the same order over multiple epochs. Also of note, some weights experience minor inflections over the course of training. In addition, the right hand side of the image shows increasing discernment by the classifier over its initial understandings of the data on its left hand side.

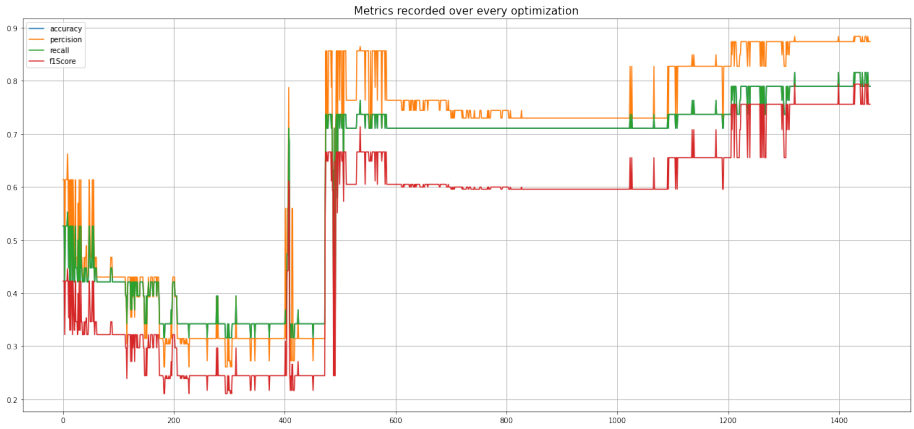

Fig. 5. Recording the classic neural network metrics over the course of training.

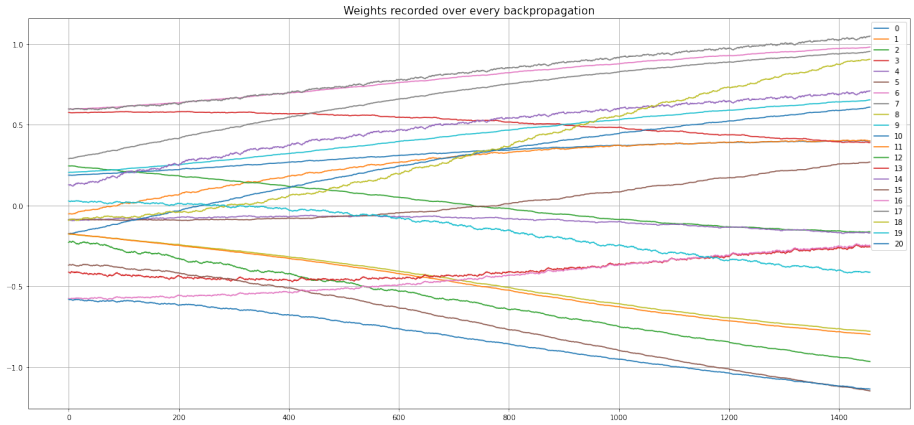

Fig. 6. Recording the classic neural network weights over the course of training.

This was the methodology used to create an accurate benchmark of a classic deep learning neural network with which to compare the new quantum computing machine learning classifier.

5.2 Quantum Machine Learning Classifier

Our work is expressed in full in two Google Colab Jupyter Notebooks online, the first is the classical neural network we used for validation and benchmarking posted online [18] and the second is our contribution, the novel quantum machine learning classifier posted online [19]. After writing the code, it was run first in the Google Colab environment, then downloaded and run on the AiMOS supercomputer, where it worked equally well.

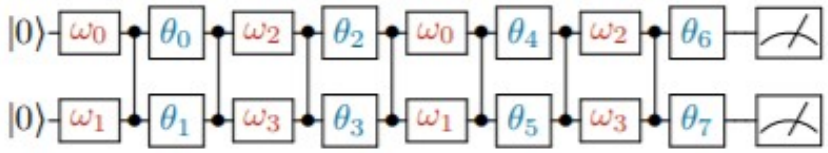

Fig. 7. The quantum circuit for the classification of Iris dataset. Vector $\boldsymbol{\omega}$ encodes for (sepal length, sepal width, petal length, petal width). Note that the features are *re-uploaded*.

Fig. 7. The Cappelletti Circuit [12]

Cappelletti et al. outline two rules in their paper used to guide the design of the quantum circuit classifiers. They write, "First, we want to minimize the number of gates. Second, we need enough parametric gates to encode the input vector and provide adequate learning capacity." These 'guidelines' do not limit the number of features or qubits that can be used in the circuit.

As noted in the Figure 7 showing a representation of the Cappelletti circuit, the omega vector (ω) denotes the features of the Iris dataset and where they are fed into the circuit as radians for R_X gate rotations. Notice that they are each encoded into the circuit twice. This is the same for the gradient based classifier in Figure8, and in the same order. In the Cappelletti circuit, after each two feature values, we see a gate connecting both wires. Cappelletti leaves the specific axis of the NOT gate used vague but the gradient based quantum classifier on the bottom uses $ZNOT$ gates in alternating directions after each pair of R_X rotation gates.

The alternating $ZNOT$ gates in Figure 8 means that the roles of target and control qubit alternate after each R_X gate. The theta vector (θ) in the Cappelletti circuit denotes the weights. Notice that in the gradient quantum classifier, there are only 7 weights (not 8). Finally, the Cappelletti gate uses a measurement of both wires to predict the value of an iris measurement. This is done by attributing certain binary combinations to each iris; '00' to Iris zero, '01' to Iris one and '10' to Iris two. The gradient based classifier does not do this.

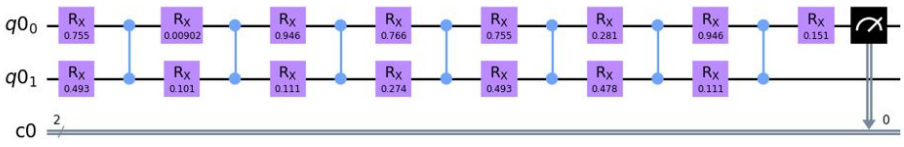

Fig. 8. The Cappelletti Circuit in QisKit

It is a pseudo binary classifier that uses the expectation value of the top wire to generate a value denoting the confidence value of the circuit. This means that, unlike the single circuit Cappelletti Classifier, our classifier uses three circuits to classify a measurement.

5.3 Our Novel Quantum Machine Learning Classifier

The novel quantum machine learning classifier described here consists of 3 similar circuits, shown in Figure 9. Each circuit is assigned to a target iris. The first circuit was assigned to Iris zero, the second circuit was assigned to Iris one and the third circuit was assigned to Iris two. Each circuit consists of 7 weights each. The process of classifying a measurement is similar to the forward propagation process of the neural network: each circuit is fed the measurements of sepal length, sepal width, petal length and petal width within the form of gate rotations in two parts of the circuits each, as described in the Cappelletti circuit. The weights of each circuit and the features of the irises cause rotations in each qubit, including $ZNOT$ rotations.

Then, the expectation value (also called the confidence value) of the top qubit in each circuit is calculated. The classifier then determines the circuit with the highest confidence value and classifies the measurement as the target iris assigned to that circuit. This may sound similar to the final layer of the neural network (the soft-max layer). This was done intentionally to help parallel the classification process of the benchmarking neural network. An important distinction between the confidence values and the soft-max layer is that the output of the soft-max layers were the result of normalizing the values and then rounded to give the neural network more simplicity. The confidence values do not sum to one and can individually range from 0 to 1 as per the original explanation of qubit expectation values. The goal is to optimize each circuit so that they produce higher expectation values when feature values associated with their target irises are fed into the gates.

5.4 Gradient Parameter-Shift Rule

The key characteristic of the quantum classifier is in its ability to optimize each circuit through the use of taking the gradient with respect to each measurement in the training set. This is done using the gradient parameter-shift rule. This rule (more of a formula) is the Gradient Parameter-Shift Rule at Equation 1

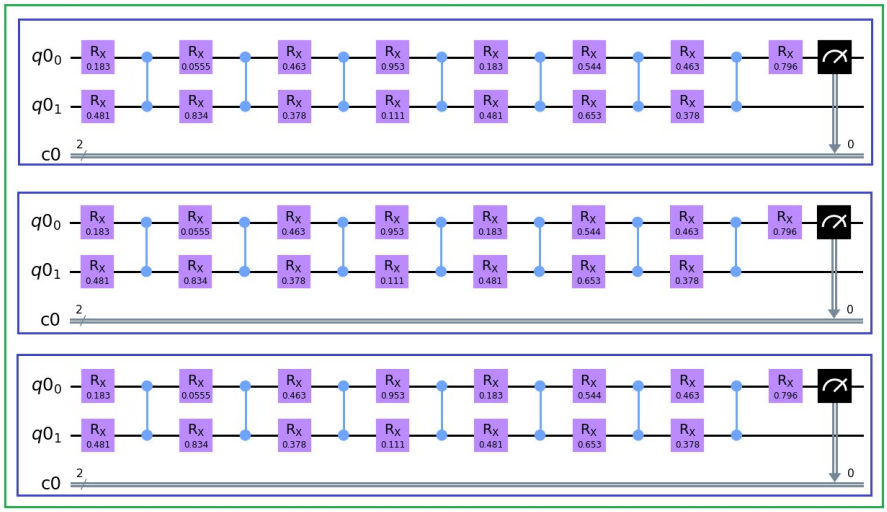

Fig. 9. Quantum Machine Learning Classifier Circuits

The first step in the rule is to take the expectation value of a circuit twice, once after moving a single parameter forward by pi/2 and once after moving the same parameter backward by pi/2. Half of the difference between these two expectation values is the gradient of that parameter. If we add this gradient to the parameter and run the circuit again, the expectation value will be larger.

$$\nabla_{\Theta_i} \left\langle \widehat{B} \right\rangle (\Theta) = \frac{1}{2} \left[\left\langle \widehat{B} \right\rangle \left(\Theta + \frac{\pi}{2} \widehat{e_i} \right) - \left\langle \widehat{B} \right\rangle \left(\Theta - \frac{\pi}{2} \widehat{e_i} \right) \right] \tag{1}$$

To optimize the entire circuit, we repeat the gradient shift rule with every weight in the circuit. It is important to note that the parameters do not change until the gradient is calculated for every weight. This could allow for the gradient parameter-shift rule to be applied to each weight independently and of optimizing the circuit by parallel processing. This would greatly decrease the time needed to change all weights because the expectation value must be estimated twice for every weight in the circuit (42 times).

The circuit at Figure 10 was given values for each gate at random. The weights were changed using the gradient parameter-shift rule as seen in Figure 11. Notice that only the parameters located in a gate that would contain a weight are changed. This still leads to a circuit with a higher expectation value. This process can be repeated until the expectation value reaches the highest possible value, 1.

The code used to take the gradient of every weight was taken from Penny Lane with some alterations to account for Iris measurements being fed into the circuit as parameters that do not change with a gradient [20]. The weights are referred to in the code as parameters and the measurements as features. Notice that the circuit function, which returns an expectation value, takes both features

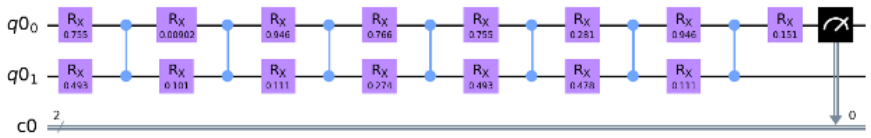

Fig. 10. Initial values for Gradient Parameter-Shift circuit application

and parameters as arguments. The gradient for each parameter is calculated sequentially but could be done in parallel.

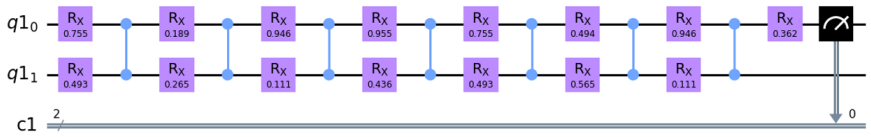

Fig. 11. Values After Gradient Parameter-Shift circuit application

5.5 Classifier Optimization

Given the parameter-shift rule, it is possible to optimize each circuit individually based on the target of each row in the training data. But this procedure did not work. The lack of a cohesive cost function that tied the circuits together meant that the expectation values only increased over the course of training. Each circuit learned to interpret the features associated with irises but not distinguish between iris targets. This led to our novel method for classifier optimization outlined with the following example.

The circuits are trained with data from the training set with the actual value of the iris and a learning rate smaller than 1. The circuit associated with that iris is optimized in the procedure mentioned previously; the features of the data are placed into the appropriate gates and held as fixed while the weights are changed using the parameter-shift rule to determine the gradient and increase the expectation value of that circuit with respect to the data. For instance, when the target value is zero, circuit zero is optimized. The novel approach of this method is to repeat the same process for the other circuits while multiplying the gradients by a value of -0.33. This is referred to as the 'de-optimization' of a circuit. The goal is to have the circuits produce a lower expectation value when features of a different target are fed into the non-corresponding circuits. A parallel could be drawn to a similar concept in reinforced learning called negative reinforcement. The 'de-optimization' of the circuits kept the expectation values

from rising perpetually. Both the learning rate and the 'de-optimization' factor, the -0.33 value, were estimated by trial and error.

5.6 Metrics

Refer to Figure 13 and Figure 12. The testing set was meant to gather metrics only. The quantum classifier was trained for 13 epochs to allow for comparison with the neural network. The quantum classifier achieved 100% accuracy by the 9th epoch. It is important to note that the classifier did not constantly improve. The accuracy of the classifier repeatedly dropped between epochs 0 (with randomly chosen weights) and epoch 3.

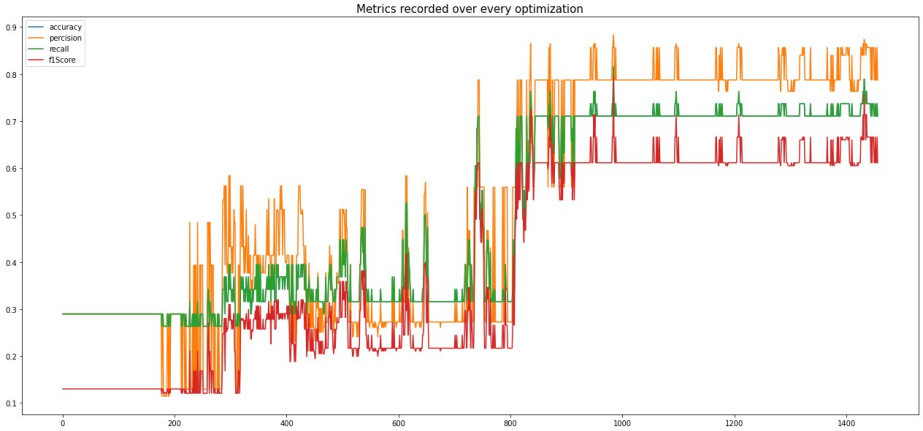

Fig. 12. Recordings of the classical neural network metrics over the course of training.

QML vs Classical Neural Network Metrics: The metrics of the quantum classifier outperform that of the neural network in Figure 13. Notice the vast juxtaposition of the wildly fluctuating weights in the quantum classifier graph and the more smooth quantum classifier metrics graph as compared against the neural networks graphs which follow an almost opposite trend.

QML vs Classical Neural Network Weights: The weights of the classifier were recorded after every optimization of the classifier. There are two glaring differences between the neural network weights and the quantum classifier weights. The weights have units and radians, and they can be examined separately by circuit number. Figure 14 displays all the weights in one graph to highlight the clearly nonlinear nature of the gradient parameter-shift rule. The 'static' from the ordered gradient descent is visible in both graphs but much more so in the quantum classifier even with a larger scale y-axis. This may indicate a method

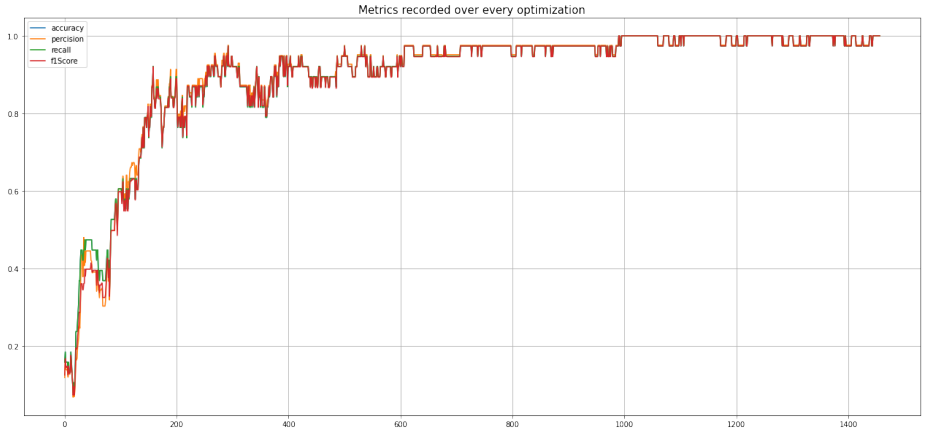

Fig. 13. Recordings of the quantum machine learning classifier metrics over the course of training.

involving momentum operators or stochastic methods could contribute greatly to stabilizing the weights of the quantum classifier.

Circuit 0 Weights: An assumption that proved false was that the weights would follow some clear trigonometric function, like a sine, cosine or tan function. As seen in Figure 15, there are several inflection points of the weights on a scale larger than can be attributed to a gradient descent approach. For example, weight number 4 starts to repeat the gradient descent 'static' common from a basic descent without using any momentum or stochastic approach after the 800th recording of the weight. The inflection of weights 1 and 3 between the 1st and 300th measurement cannot be attributed to this same phenomena. These drastic inflections happen in repeated patterns in other iterations of the classifier (using other random starting weights). The more smooth inflection of weight 2 is also common among other iterations.

Circuit 1 Weights: In the Figure 17, one can see that weight 8 shows another interesting pattern. The units of the weight are in radians and the weight repeatedly peaks above and then dips below the value of pi. Another assumption proven false was that this value would have some sort of strong significance given its role in trigonometry, like an asymptotic limit for the weights. This is not the case. Placing the iris features before the weights in the circuit removed any associations with typical 'magic numbers' such as in trigonometry.

Circuit 2 Weights: In the circuit in Figure 17 one can see a trend found in some 'curse of dimensionality' problems. The value starts at 0 and increases, almost linearly, until it hovers at a value just under 4 after the 1200th measurement.

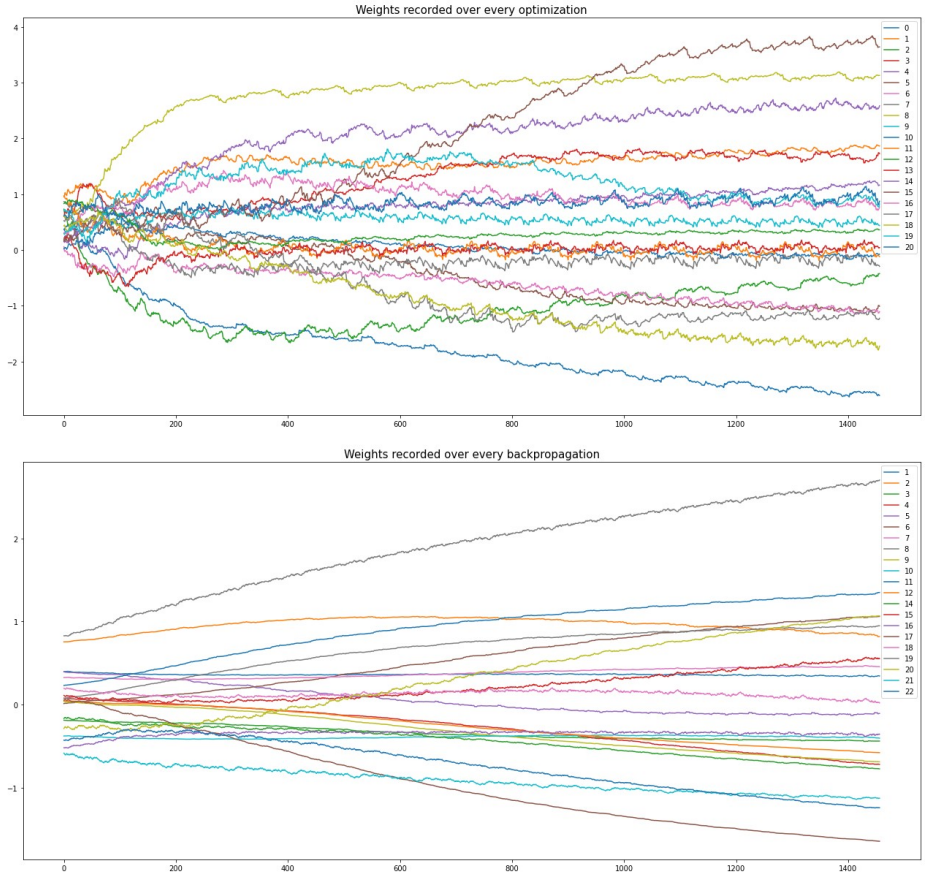

Fig. 14. Recordings of the QML classifier weights and classic neural network weights over the course of training.

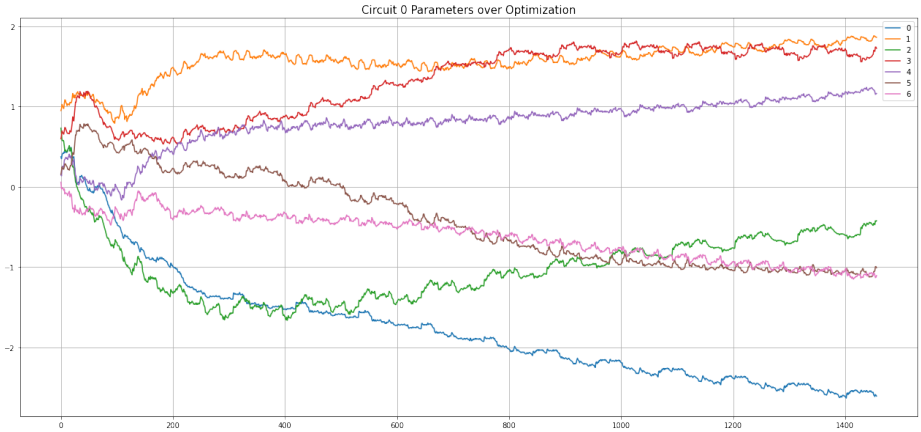

Fig. 15. Recordings of the circuit associated with target 0.

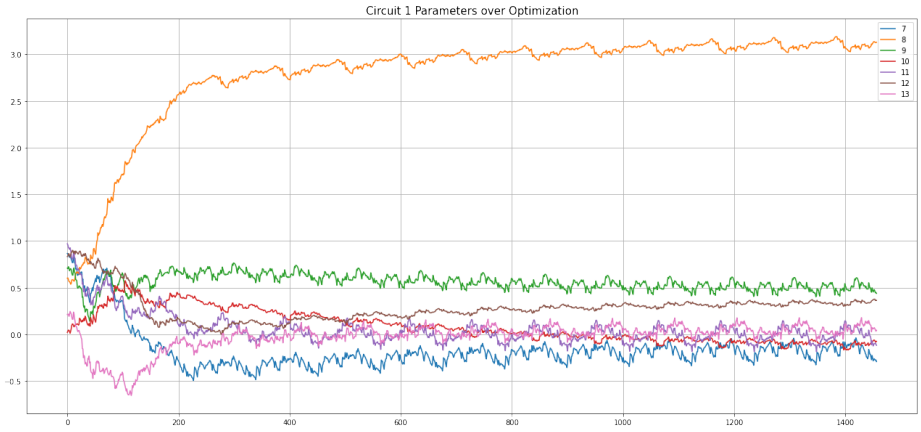

Fig. 16. These are the recordings of the circuit associated with target 1.

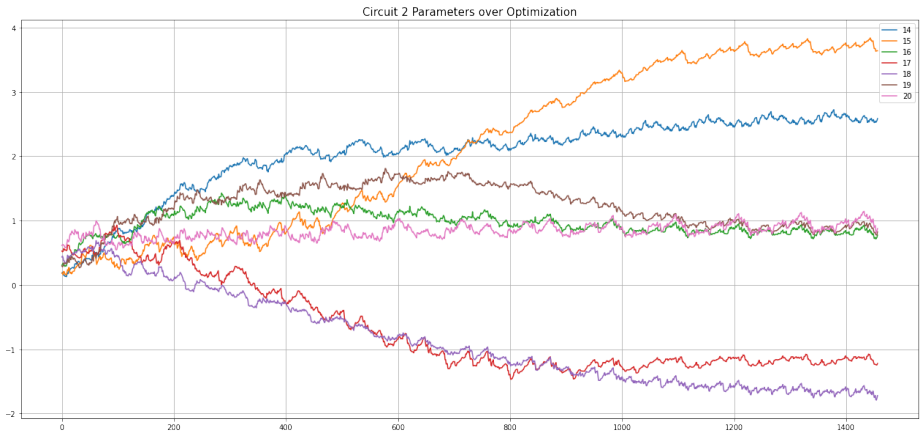

Fig. 17. Recordings of the circuit associated with target 2.

Trigonometry tells us that a rotation of 4 radians is a larger radial distance than the equivalent rotation in the negative direction. A simpler example in other units could be; why move 270 degrees in one direction when you could move 90 degrees in the other direction and reach the equivalent spot? One conclusion, that requires more research, is the curse of dimensionality made this weight take the longer path to reach the same global minimum. This is a major proposition that requires more research but could eventually be the ultimate proof of this and other quantum circuits' potentials as classifiers.

6 Conclusions

The quantum machine learning classifier trained in four fewer epochs than the traditional neural network.

7 Future Work

This quantum computing machine learning classifier is more efficient than its classic deep learning neural network counterpart when compared on classical computers using the Iris dataset as it delivers optimal results in fewer epochs. Future work would be to use it with larger datasets that consist of more targets and rows.

8 Acknowledgments

The authors are especially grateful to the CAREERS (Cyberteam to Advance Research and Education in Eastern Regional Schools) program for supporting Gio Abou Jaoude with a stipend and with tutoring by sponsoring our research

project. Our work here will be extended to its next phase of code optimization on the AiMOS supercomputer hosted at Rensselaer Polytechnic institute (RPI), the largest supercomputer in New York State. The CAREERS Cyberteam Program is a 3-year initiative funded by the National Science Foundation (Award No. 2018873) to build a regional pool of Research Computing Facilitators to support researchers at small and mid-sized institutions in Connecticut, Delaware, New Jersey, New York, Pennsylvania, and Rhode Island, leveraging the work of the Northeast Cyberteam and national programs including XSEDE Campus Champions, CaRCC and others [21].

References

1. J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
2. N. Wiebe, D. Braun, and S. Lloyd, "Quantum algorithm for data fitting," *Physical review letters*, vol. 109, no. 5, p. 050505, 2012.
3. M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, "Quantum Boltzmann machine," *Physical Review X*, vol. 8, no. 2, p. 021050, 2018.
4. S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum principal component analysis," *Nature Physics*, vol. 10, no. 9, pp. 631–633, 2014.
5. N. Spagnolo, C. Vitelli, L. Sansoni, E. Maiorino, P. Mataloni, F. Sciarrino, D. J. Brod, E. F. Galvao, A. Crespi, R. Ramponi, *et al.*, "General rules for bosonic bunching in multimode interferometers," *Physical review letters*, vol. 111, no. 13, p. 130503, 2013.
6. G. H. Low, T. J. Yoder, and I. L. Chuang, "Quantum inference on bayesian networks," *Physical Review A*, vol. 89, no. 6, p. 062315, 2014.
7. N. Wiebe, A. Kapoor, and K. M. Svore, "Quantum perceptron models," *arXiv preprint arXiv:1602.04799*, 2016.
8. N. Wiebe, A. Kapoor, and K. M. Svore, "Quantum deep learning," *arXiv preprint arXiv:1412.3489*, 2014.
9. V. Dunjko, J. M. Taylor, and H. J. Briegel, "Quantum-enhanced machine learning," *Physical review letters*, vol. 117, no. 13, p. 130501, 2016.
10. S. Aaronson, "Read the fine print," *Nature Physics*, vol. 11, no. 4, pp. 291–293, 2015.
11. K. Kathuria, A. Ratan, M. McConnell, and S. Bekiranov, "Implementation of a Hamming distance–like genomic quantum classifier using inner products on ibmqx2 and ibmq_16_melbourne," *Quantum machine intelligence*, vol. 2, no. 1, pp. 1–26, 2020.
12. W. Cappelletti, R. Erbanni, and J. Keller, "Polyadic quantum classifier." {https://arxiv.org/pdf/2007.14044.pdf}, 2020.
13. S. Doshi, "Various optimization algorithms for training neural network." {https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6}, 2019.
14. P. A. M. Dirac, "A new notation for quantum mechanics," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, pp. 416–418, Cambridge University Press, 1939.
15. F. Bloch and J. D. Walecka, *Fundamentals of statistical mechanics: manuscript and notes of Felix Bloch*. World Scientific, 2000.

16. SCER Staff, "AiMOS, Most Powerful Supercomputer at a Private University, To Focus on AI Research." {https://news.rpi.edu/content/2019/12/05/aimos-most-powerful-supercomputer-private-university-focus-ai-research}, 2019.
17. SciKit Learn, "Iris dataset." {https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html}, 2021.
18. O. Aflack, "neural_network.ipynb." {https://colab.research.google.com/drive/10y6glU28-sa-OtkeL8BtAtRlOITGMnMw#scrollTo=oTrTMpTwtLXd}, 2018.
19. G. Abou Jaoude, "Quantum machine learning classifier in a Google Colab Jupyter Notebook." {https://colab.research.google.com/drive/1YMPn7LTtfd73WpWc4XmGUsAxHruD0WBf?usp=sharing}, 2021.
20. PennyLane Dev Team, "Quantum gradients with backpropagation." {https://pennylane.ai/qml/demos/tutorial_variational_classifier.html}, 2021.
21. NSF, "CAREERS: Cyberteam to Advance Research and Education in Eastern Regional Schools." {https://careers-ct.cyberinfrastructure.org/}, 2021.