On the Capabilities of Quantum Machine Learning

Sarah Alghamdi and Sultan Almuhammadi
College of Computer Science and Mathematics
King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
Emails: saraalghamdi497@gmail.com, sultan@almuhammadi.com

***Abstract*—Machine learning techniques give impressive results in many areas. However, due to the physical limitation of integrated circuits which restricts their computational power growth, and the rapid advances in quantum computing, lots of research studies on quantum machine learning (QML) have been done recently. QML is a technique that uses quantum algorithms as parts of the implementation. Quantum algorithms use quantum mechanics and have the potential to outperform classical algorithms for a given problem. In this paper, three widely used machine learning algorithms are discussed and their quantum versions are presented, namely: quantum neural network, quantum autoencoder, and quantum kernel method. In addition, we discuss the potential capabilities of these QML algorithms and review recent work employing them. Moreover, a quantum neural network prototype is implemented using Qiskit as a proof of concept and tested on a real quantum computer. Empirical results show that quantum neural networks can be trained efficiently.**

Keywords*—*Quantum Machine Learning, Near-term Quantum Computers, Neural Network, Autoencoder, Kernel Method.

I. Introduction

The rapid increasing computational power and data availability, as well as recent algorithmic advances, have led machine learning (ML) techniques to impressive results in regression, classification, data generation and reinforcement learning tasks. However, the physical limits of integrated circuit chips fabrication may restrict the growth of computational power in the future. This motivates researchers to look for computing alternatives that support the need of high computational power for ML applications. Quantum computing is one promising alternative for ML algorithms. It uses the properties of quantum mechanics and quantum states to perform calculations and represent data.

Since quantum systems produce counter-intuitive patterns believed not to be efficiently produced by classical systems, it is reasonable to postulate that quantum computers may outperform classical computers on ML tasks. The field of quantum machine learning (QML) explores how to devise and implement concrete quantum software that offers such advantages. Recent work has made clear that the hardware and software challenges are still considerable but has also opened paths towards solutions.

Quantum algorithms speed up classical computation drastically. For example, Shor's algorithm solves the integer factorization problem in polynomial time, which gives an exponential speedup to the factorization algorithm on classical computers [1]. Grover's algorithm gives a quadratic speedup to the search problem [2]. With the rapid advances in ML applications, and the complexity of datasets, classical ML might face a great challenge when processing big data. Therefore, many researchers believes that quantum computers can achieve high speedup in ML algorithms [3], [4], [5], [6].

QML models are deployed on quantum computers. They use quantum effects such as superposition, entanglement and interference to perform computation. These give great advantages in quantum neural network models, such as speedups in training and faster processing. Quantum neural networks serve as a newer class of machine learning models that are deployed on quantum computers. Although there has been much development in the growing field of QML, a systematic study of the trade-offs between quantum and classical models has yet to be conducted. In particular, the question of whether quantum neural networks are more powerful than classical neural networks is still open.

This paper aims to highlight and discuss the capabilities of QML models. We briefly explain few concepts in quantum mechanics and quantum computing that are necessary to understand and discuss the presented content. We review the literature in QML and discuss the possible mix of classical machine learning and quantum computation. In particular, we discuss three common ML algorithms and present their quantum versions, namely: quantum neural network, quantum autoencoder, and quantum kernel method. We discuss the potential capabilities of each algorithms and review recent work employing them. Moreover, we implement a quantum neural network prototype using Qiskit as a proof of concept. We test the prototype on a real quantum computer at IBM-Q Experience. We discuss the results of this experiment as well as the results of other studies related to QML techniques in the literature.

II. Background

Machine learning based algorithms are the core of data mining and visualization approaches. Such algorithms are built based on a minimization problem of a constrained function. By solving the associated optimization problem, a decision function can be obtained which maps between the input and output points. Modeling and simulating such systems gave rise to the idea of employing quantum mechanics for computing. To improve the learning process, QML enlists the support of nature on a subatomic level. Certain probabilistic operations could be accomplished more quickly in QML due to the quantum parallelism [7].

Computational intelligence is a field of ML inspired by nature methodologies to solve optimization problems. Examples include swarm intelligence [8], evolutionary computing [9], neural networks [10], autoencoders [3], and kernel methods [4]. A new research direction emerged recently that

borrows metaphors from quantum physics. These quantum-like methods in ML are in a way nature inspired, hence they are related to computational intelligence. There are also quantum versions of particle swarm optimization [11]. The particles in a swarm are agents with simple patterns of movements and actions, each one is associated with a potential solution. Relying on only local information, the quantum variant is able to find the global optimum for the optimization problem in question.

Machine learning models can be divided into four different categories based on the data and the algorithm types as shown in Figure 1. In each category, we either have classical or quantum data, processed by a classical or a quantum algorithm. In the CC category, we have classical data processed on a classical computer. This category includes all of the classical machine learning that we already know. In the second category, QC, we have quantum data processed on a classical machine. Models in this category can be found in some quantum mechanical experiments that output some quantum data, which are further processed using classical machine learning algorithms to tweak some parameters of the experiment. In the CQ category, we have classical data processed on a quantum computer. Most of the recent QML studies fit in this category. Finally, in the QQ category, we have quantum data processed on a quantum computer. An example of a model in this category is the quantum autoencoder explained in Section IV-B.

Type of Data \ Type of Algorithm	Classical	Quantum
Classical	CC	CQ
Quantum	QC	QQ

Figure 1: Classical/Quantum Machine Learning Models

III. Preliminaries

Relative to classical computing, the capabilities of quantum computing are generally based on a number of principles and phenomena originated from quantum mechanics. Therefore, understanding the basic laws and principles of quantum mechanics is essential to realize and understand other complicated topics related to quantum computing such as complex probability amplitudes, quantum parallelism, quantum interference, and quantum evolution. Moreover, grasping the fundamentals of Hilbert space formalism is highly required to employ the quantum features in designing and developing quantum-based algorithms. In this section, we briefly introduce some important concepts in quantum computing related to QML.

A. Mathematical Framework

In this section, the mathematical tools of quantum computing are presented. Such tools are required for modeling and measuring the quantum systems.

1) Hilbert Spaces: The Hilbert space formalism serves as the foundation for quantum mechanical theories and their related phenomena and definitions. It is constructed by a collection of all possible states of the quantum system [12].

2) Quantum State: The quantum states are represented by vectors of a Hilbert space H. These states provide a full-description of the internal memory of the quantum computer. According to the formalism of Hilbert space H, any state X can be modeled as *bra-vector* or *ket-vector*. Such state can be represented in terms of any basis B and written as

$$|X\rangle = \sum_{i \in B} |i\rangle\langle i|X\rangle = \sum_{i \in B} \alpha_i |i\rangle \quad \text{and} \quad \langle X| = \sum_{i \in B} \alpha_i^* \langle i|$$

In the above expression, $\langle i|X\rangle$ represents the probability of qubit collapse to $|i\rangle$ when measuring X with respect to B.

3) Qubits: The states in the quantum computers are modeled by normalized vectors in $\mathbb{C}^n$. The two-dimensional state in $\mathbb{C}^2$ represents a qubit. The coordinates of such qubit can be represented with respect to the orthonormal basis as follows.

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \{|0\rangle, |1\rangle\}, \quad \text{where} \quad |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

B. Quantum Gates

A quantum register of n-qubits can be represented in any superposition of 2^n basis states, in which each qubit can be described in an infinite number of states. By implementing quantum gates and transformations, tremendous parallelism can be established. Quantum gates can be considered as unitary operation on qubits, or a simple rotation of the qubits along some axis using the *Bloch* sphere representation. For example, the X-Gate makes a 180° rotation around the X-axis as shown in Figure 2. While the Y-Gate makes a 180° rotation around the Y-axis. The $R_x(\theta)$ and $R_y(\theta)$ gates make rotations with arbitrary angles around the X and Y axes, respectively.

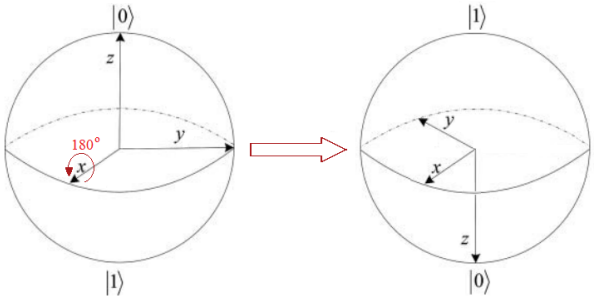

Figure 2: Rotation Around X-Axis

Moreover, quantum gates on multiple qubits can be mathematically represented by a matrix. For example the Controlled-Not (CNOT) gate is applied on two qubits, say (x, y), and transforms them into $(x, x \oplus y)$. Thus, if x is true, the value of y is negated. This creates *entanglement* with allows the quantum information to travel from one qubit to another. The CNOT gate is represented by the following matrix.

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1}$$

C. Quantum Tunneling

Tunneling is a phenomenon which has shown to be very useful in applications. It is mainly used in quantum annealing to allow escaping from local minima situations. Such phenomenon does not exist in classical computers. Let us consider a moving particle that possesses mechanical energy E hits a barrier with potential energy $U(x)$. In classical systems, the particle will be unable to overcome the obstacle if $E < U(x)$. However, such a possibility exists in quantum systems. The location of the quantum particle is probabilistic and it may pass through the barrier even if $E \ll U(x)$ as if there is a tunnel cut across.

For example, consider the two turning points x_1 and x_2 on the two sides of a barrier as shown in Figure 3. In classical systems, the region $x_1 < x < x_2$ is inaccessible. However, in the quantum systems, the particle has an indeterminancy in its energy. Hence it may pass through the barrier with some probability $p > 0$. The probability of tunneling reduces as the potential barrier increases.

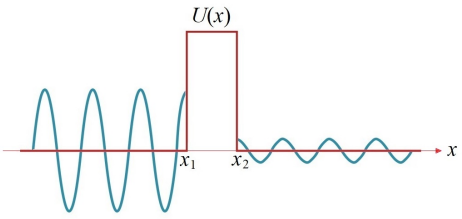

Figure 3: Quantum Tunneling

D. Quantum Algorithms

Generally speaking, a quantum algorithm computes all possible solutions in just one-step. This is achieved by establishing the computed state as a superposition of all possible classical states. However, the problem of wave function collapse shall be handled while developing quantum algorithms. In fact, only one of the results is generally observed in quantum computing and other states are collapsed and lost forever. Hence, quantum computations attempt to exploit the challenge by increasing the likelihood of a certain outcome and interpret it to get the desired results. Examples of such algorithms include Grover's and Shor's algorithms.

1) Grover Algorithm: Search problem basically is about finding an input of a one-way function, $f(x)$, to generate a desired output know as *target* value, t. We assume that the size of the co-domain of the function is M, thus, to find x such that $f(x) = t$ in a search space of size M. Classically, such a search problem requires $\frac{M}{2}$ steps. Grover's algorithm [2] achieves quadratic speedup using quantum computing, and it runs in $O(\sqrt{M})$ time.

2) Shor's Algorithm: Factorization problem can be solved in polynomial time by Shor's algorithm [1] using quantum Fourier transform (QFT) and modular arithmetic. QFT is used to find the period of a periodic function $f_{a,N}(x) = a_x \mod N$. Once the period r is found, the factors of N can be computed by the Euclidean algorithm to calculate $\gcd((a^{r/2}+1), N)$ and $\gcd((a^{r/2}-1), N)$ which give at least one of nontrivial factor of N. The time complexity of Shor's algorithm is $O(n^2 \log n \log\log n)$.

IV. Quantum Machine Learning

QML uses quantum algorithms as parts of the implementation. Quantum algorithms have the potential to outperform classical algorithms for a given problem. This potential is known as quantum speedup, which may vary from quadratic to exponential speedups, such as in Grover's and Shor's algorithms, respectively. However, in some cases, the best possible performance of a classical algorithm is not always known. For example, we assume that there is no polynomial-time classical algorithm known for integer factorization, but the possibility is not provably ruled out. Therefore, the speedup for Shor's algorithm to solve integer factorization is estimated as exponential based on this assumption. In this section, we present three QML algorithms and discuss their potential capabilities as well as some recent examples in the literature. These three algorithms are: quantum neural network, quantum autoencoder, and quantum kernel method.

A. Quantum Neural Network

In the classical neural network, we assume that there is a true relationship between the data (the features of an object) and the output (the target class of the object). This relationship can be considered as a function g such that $y = g(x)$, where x is a vector representing the features and y is the target output. In classical neural network models, we do not know the actual relationship function $g(x)$, so we want to approximate it by another function, say f, such that $\hat{y} = f(x, w)$, where $\hat{y}$ is an approximation of y and w is some set of weights that can be used as learning parameters.

The simplest approximation method is the *linear model*, in which we multiply our data by weights and add some biases b to obtain the prediction $\hat{y} = wx + b$. However, the linear models do not give good approximations for non linear systems. Therefore, we introduce the *perceptron model* in a neural network, given in (2), which adds non-linearity to the model by applying some non-linear activation function σ.

$$\hat{y} = \sigma(wx + b) \tag{2}$$

The model takes an input x, and aggregates it by computing the weighted sum as well as some possible bias. It returns 1 only if the aggregated sum is greater than some threshold. Otherwise, it returns 0.

We may generalize this idea by introducing multiple hidden layers of perceptrons in the *feedforward neural network* model where we apply different activation functions in each layer with different learning parameters and biases. Equation 3 defines a feedforward neural network model with one hidden layer. Here, σ_1 is the activation function in the hidden layer, while σ_2 is the activation function for the output layer as illustrated in Figure 4.

$$\hat{y} = \sigma_2(w_2\sigma_1(w_1x + b_1) + b_2) \tag{3}$$

The quantum neural network (QNN) can be built using the same idea as classical ones. Figure 5 shows a general structure of a QNN. Let X be an n-dimensional vector representing the

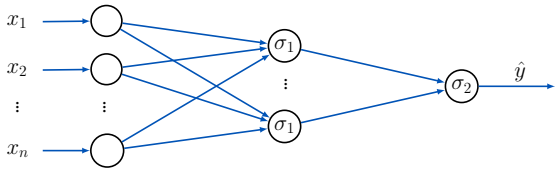

Figure 4: Feedforward Neural Network

input data (the values of the features). The model starts with n qubits initialized at state $|0\rangle$, one qubit for each feature. Then, it applies a unitary operation U on all these qubits using the values of each input feature in x. This encodes the input data in a quantum state, which is similar to the input layer in a classical neural network. Such unitary operation could be a simple of Pauli X-Gate for example. The operations $W(\theta_1)$ and $W(\theta_2)$ are similar to the hidden layers in the classical neural network, where the actual learning is done. The model uses another unitary operation W and a set of learning parameters, like $W\theta_1$ in the first hidden layer. After the last hidden layer is computed, the model measures the qubits and interprets the result to find the output $\hat{y}$ which is the predicted class. The measuring of a quantum state adds non-linearity to the model. Finally, it compares the prediction $\hat{y}$ with the actual class y and updates the learning parameters in all hidden layers $(\theta_1, \theta_2, \dots)$ accordingly.

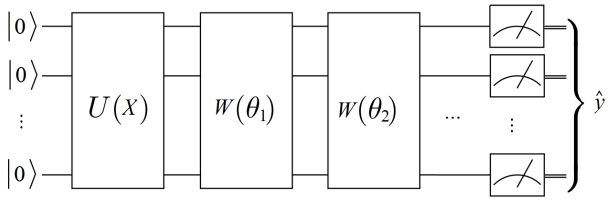

Figure 5: General Structure of QNN

QNNs have the potential to outperform their classical ones in many tasks. To allow the progressive membership of data instances, QNNs employ the superposition of quantum states. Moreover, using the concept of quantum tunneling, simulated quantum annealing avoids being caught in local minima.

Due to its effectiveness and simplicity, the QNN receives great attention in research [13]. Farhi et al. [14] proposed a QNN that can be trained by supervised learning to represent classical and quantum labeled data. Their proposed design is exploratory and relies on the classical simulation of small quantum systems.

Abbas et al. [5] discussed the advantages that can be achieved in near-term quantum computers machine learning tasks. They proposed a measure for how powerful and trainable QML models are in relation to popular classical neural networks, and proved that it can be used to assess any statistical model's ability to generalize on new data.

In [15], Zoufal et al. presented a hybrid quantum-classical algorithm to facilitate efficient learning and loading of generic probability distributions into quantum states. They used quantum simulation to demonstrate the use of the trained quantum channel in a quantum finance application

B. *Quantum Autoencoder*

Autoencoders are neural networks that can learn efficient low-dimensional representations of data in higher-dimensional space. An autoencoder takes and n-dimensional point x as an input and maps it to a lower dimensional point y such that x can be easily recovered from y. In classical ML, we choose the structure of the underlying autoencoder network to represent the data on a smaller dimension, and effectively compress the input. Figure 6 shows an example of a five-bit autoencoder with a two-bit latent space. The encoding map ($\mathcal{E}$) encodes a five-bit input (the red dots on the left) into a two-bit intermediate state (the yellow dots). Then, the decoding map ($\mathcal{D}$) attempts to reconstruct the input bits at the output (the green dots).

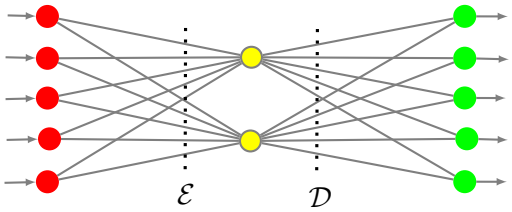

Figure 6: An Example of a 5-bit Autoencoder

Inspired by this idea, models of quantum autoencoders are introduced to perform similar tasks on quantum data [3], [4]. Quantum autoencoders compress a set of quantum states, then attempt to recover them. This sort of compression cannot be employed by classical computers due to the quantum mechanics. Figure 7 illustrates a circuit implementation of a quantum encoder that has 2-qubit latent space. The unitary operation $U(\theta)$ is to encode a 6-qubit input state $|X\rangle$ into a 2-qubit intermediate state (the latent space). We then need to design the decoder unitary operation $U'(\theta)$ such that it takes four qubits initialized at $|0\rangle$ and the two intermediate states, and attempts to reconstruct an output state $|X'\rangle$ that is similar to the input state $|X\rangle$. The motivation for a quantum autoencoder is to allow recognizing patterns beyond the capabilities of a classical autoencoder; since the input in this case has different properties of quantum mechanics.

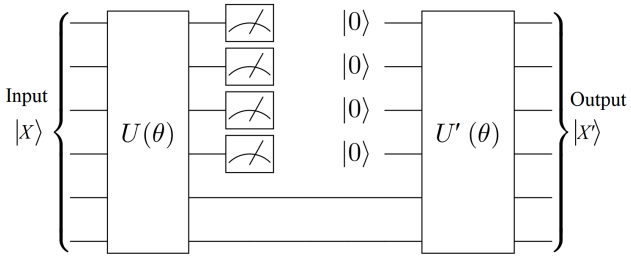

Figure 7: An Example of a Quantum Autoencoder

Quantum autoencoders have been studied extensively in recent research and proposed for many applications. In 2017, Romero et. al. [3] applied quantum autoencoders models in the context of quantum simulation to compress ground states of the Hubbard model and molecular Hamiltonians. The quantum autoencoder is trained to compress a particular data set of

quantum states, where a classical compression algorithm cannot be employed. The parameters of the quantum autoencoder are trained using classical optimization. The authors show that their proposed model is capable of learning a unitary circuit to create a quantum autoencoder.

In 2021, Bravo-Preito [4] presented an enhanced feature quantum autoencoder capable of compressing quantum states of different models with higher fidelity. The proposed model is a variational quantum algorithm to define a parameterized quantum circuit that depends upon adjustable parameters and a feature vector that characterizes such a model. The validity of the method is assessed in simulations by compressing ground states of the Ising model and classical handwritten digits. The results show that the proposed model improves the performance compared to the standard quantum autoencoder using the same amount of quantum resources, but at the expense of additional classical optimization. Therefore, the model makes the task of compressing quantum information better suited to be implemented in near-term quantum devices.

C. Quantum Kernel Method

Kernel functions are well-established methods in ML that are essentially utilized to facilitate data analysis [6], [16]. Such methods are based on mapping the input data into a higher-dimensional feature space (possibly infinite-dimensional) resulting in simpler models. The support vector machine (SVM) is one of the most popular examples of kernel methods. It is mainly employed to linearly separate two classes of input data by mapping the data into a different space in which a suitable decision boundary can be made to classify them with high accuracy. Similar to quantum computing, the kernel functions conduct the computation implicitly in a potentially intractably large Hilbert space by using efficient manipulation of data vectors.

An example of a kernel function defined by classical and QML models is illustrated in Figure 8. The nodes (A, B, ..., etc.) represent data points in different spaces. The edges between these nodes represent the similarity measure (the kernel function) between data. The geometric difference, g, is a difference between similarity measures (edges) in different ML models, and it is computed by

$$g_{i,j} = g(K_i||K_j) = \sqrt{||\sqrt{K_j}K_i^{-1}\sqrt{K_j}||_\infty}$$

where $||.||_\infty$ is the spectral norm of the resulting matrix. When the model is embedded into quantum Hilbert space, the quantum kernel functions conduct the computation by using efficient manipulation of data vectors, where the effective dimension of the dataset in the quantum Hibert space is $d \leq N$. The model then is projected back to the classical space and the projected quantum kernel is computed, which gives better learning results.

The kernel method has received little attention in the literature on QML which explores the use of quantum computing as a ML resource. In [16], Huang et al. used prediction error bounds to assess a potential quantum advantage. They show a significant prediction advantage over classical models on synthesized datasets designed to have a maximal quantum advantage.

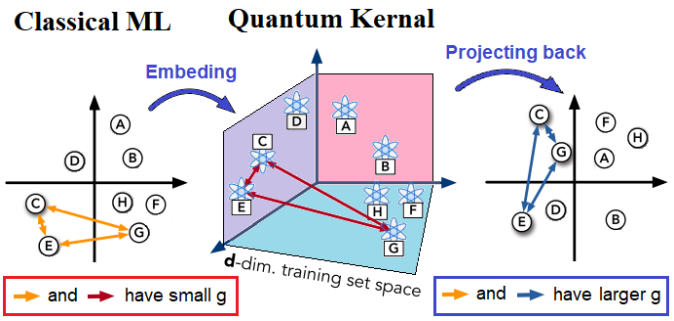

Figure 8: Quantum Kernel Method

In [6], the relationship between feature mapping, kernel functions and quantum computing is employed to the full potential. In view of the same, encoding conventional information into a quantum state is modeled as a data mapping into the Hilbert space of the quantum system. It is worth noting that instead of conducting the computations explicitly in the feature space, the kernel function is utilized here, which is the trick of this approach. As a result, two different strategies for developing quantum-based ML have emerged. In the first strategy, the inner products of the quantum states are estimated using a simple special-purpose quantum device for which the results are then fed into the classical kernel models. In the second strategy, the data are examined directly in the feature Hilbert space of quantum states, where basic classifier like linear models have great strength. With the approaching quantum computation, the two strategies result in classically intractable hybrid QML algorithms with near-term quantum technology.

V. Experimenting with Quantum Neural Network

In this section, we explain an experiment conducted in this work as a proof of concept. We implement a prototype of a QNN using Qiskit and test it on a real quantum computer provided by IBM-Quantum Experience.

A. Data Preparation

We use the Iris dataset available at Scikit-Learn website [17] for three types of irises (Setosa, Versicolour, and Virginica). The data set has 150 samples, 50 samples for each type. Each sample has four features, namely: Sepal Length, Sepal Width, Petal Length, and Petal Width. In this experiment, however, we decided to do binary classification. Therefore, we only take the first 100 samples for the first two types of irises (Setosa and Versicolour). We divide the dataset into two sets, with 67 samples in the training test (X_train), and 33 samples in the testing set (X_test).

B. Quantum Neural Network Implementation

We design and implement a QNN with n qubits and one hidden layer. The quantum circuit shown in Figure 9 is an instance of our implementation with $n = 4$ for the Iris dataset. In the input layer, we use rotation around the X-axis $R_x(x_i)$ as the unitary operation where the angle x_i equals the i^{th} input feature value. Then we create a variational quantum circuit by applying a number of CNOT gates to entangle the qubits

and allow the quantum information to travel from one qubit to another. We apply rotation operations around the Y-axis, $R_y(\theta_i)$, where θ_i is the angle representing the i^{th} learning parameter. These angels are initialized at $45°$, and we want to learn the optimal angles that give the best approximation of the target class. Then we measure the first qubit q_0 since the CNOT gates entanglement allows the information to travel from all qubits to the first qubit. The prediction of the class is based on the measurement of the first qubit, and the learning parameters are updated accordingly.

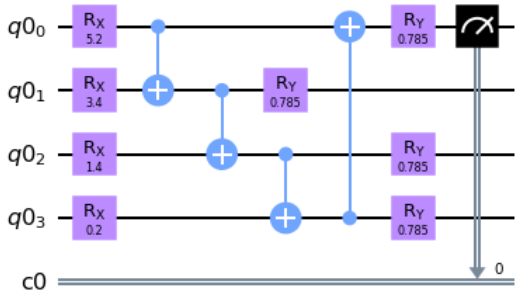

Figure 9: An Experimental QNN Circuit with 4 Qubits

C. Testing and Results

We test our implementation on a real quantum computer at IBM-Quantum Experience. We choose the *ibm_bogota* system, which has 5 qubits, 32 QV (quantum volume), and 2.3K CLOPS (circuit layer operations per second). This is good enough to test our neural network with up to five input features. We set the number of shots = 1000, which runs the quantum circuit 1000 times, and compute the probability of the measured classical bit C_0 accordingly.

1) Testing the Loss and Accuracy of the QNN: Since the dataset has 4 input features, we set the number of qubits $N = 4$ initially, run the test for 25 epochs, and compute the loss and the accuracy of the model. The accuracy is given in percentage and computed by

$$accuracy = 100 \times \frac{\text{number of correct predictions}}{\text{number of samples}}$$

Figure 10 shows the accuracy we get in each epoch. We notice that the model learns fast. It achieves 82% accuracy after in Epoch 2 and stays above 80% afterward.

The loss is calculated by $loss = (prediction - target)^2$, which indicates how far is our predictions from the actual targets regardless of the accuracy. Figure 11 shows the loss we get in each epoch. We notice that the loss drops to 0.21 at Epoch 12 and remains below 0.21 afterward.

2) Testing the performance of the QNN: Since we have five qubits in our quantum computer (*ibm_bogota*), we create datasets with number of features ranging from 2 to 5 from the original dataset to evaluate the performance of the model. For the dataset of 5 input features, we just add one feature and set the values in all entries to 1.0 which does not affect the accuracy. Table I shows the execution time taken by each test,

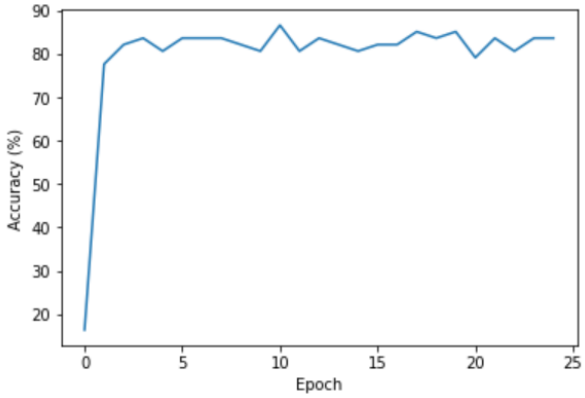

Figure 10: The Accuracy Graph of the QNN

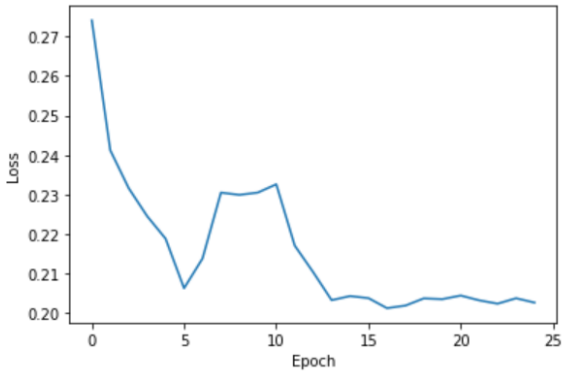

Figure 11: The Loss Graph of the QNN

which implies that there is no much difference in the execution time.

It is worth noting that when running the quantum circuit on a real quantum computer, the actual quantum circuit might be different from the one we design. For example, when we run our circuit in Figure 9, it is converted (compiled) to the circuit shown in Figure 12 to be executed on the *ibm_bogota* quantum computer. This is because some of the qubits are not neighbor to each other in the actual topology of *ibm_bogota* hardware. Therefore, some swapping should be done to create a circuit equivalent to ours that can be executed on this quantum computer.

VI. Analysis and Discussion

Empirical results in Figure 10 show that the QNN can learn fast and achieve accuracy above 80% after 3 epochs. The accuracy fluctuates up and down after the third epoch. This is mainly due to the noise in the qubits [5]. However, we notice from Table I that the quantum machine takes a much longer time than classical machines in this task. A classical neural network may achieve higher accuracy on such a small dataset in much shorter times (a fraction of a second). On the other hand, it may take several more epochs to achieve the same accuracy. In the near-term quantum computer, and with a good number of qubits, we may build fault-tolerant QNNs to reduce the effect of noise, and achieve high accuracy

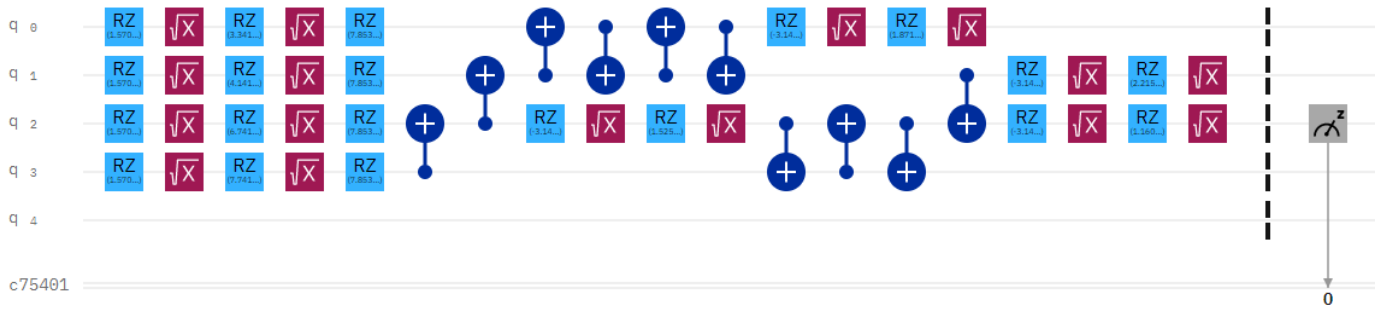

Figure 12: Actual Circuit Running On IBM_Bogota Quantum Computer

Table I: QNN Performance Test Results

Number of Features (N)	Execution Time (Seconds)
2	6.3
3	6.3
4	6.4
5	6.5

much faster than classical neural networks since they need less number of epochs.

The advantages of quantum computing in ML are quite promising in the near-term quantum era. In [5], a comparison between classical and quantum machine learning shows that the QNN gives much lower loss than the classical one. Other studies suggest QML has the potential to outperform classical ones in other ML algorithms [3], [4], [6], [8], [9], [11].

However, when considering quantum capabilities for a certain application, we should know how to achieve these advantages. Huang et al. [16] presented a foundation for understanding opportunities for quantum advantage in a learning setting. They showed that the advantage for quantum models is not guaranteed in some cases, and the classical algorithms can be computationally more powerful.

VII. Conclusion

Despite the noise and long execution time of today quantum computers, applying quantum computing in machine learning remains an interesting issue. Many studies show that QML is very promising for near-term quantum computers. However, we should be careful when considering quantum advantage for certain applications. Since in some cases, the advantages of applying quantum computing in machine learning are not guaranteed. Therefore, understanding the foundation of quantum advantage is essential.

As for future work, we suggest benchmarking QML algorithms and comparing them with classical ones. We also suggest implementing and testing other QML algorithms on real quantum computers. Moreover, we better test these algorithms on near-term quantum devices once available.

Acknowledgment

The authors would like to thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for supporting this research.

References

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[3] J. Romero, J. P. Olson, and A. Aspuru-Guzik, "Quantum autoencoders for efficient compression of quantum data," *Quantum Science and Technology*, vol. 2, no. 4, p. 045001, 2017.

[4] C. Bravo-Prieto, "Quantum autoencoders with enhanced data encoding," *Machine Learning: Science and Technology*, 2021.

[5] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, "The power of quantum neural networks," *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, 2021.

[6] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," *Physical review letters*, vol. 122, no. 4, p. 040504, 2019.

[7] D. Deutsch, "Quantum theory, the church–turing principle and the universal quantum computer," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[9] D. E. Goldberg, "Genetic and evolutionary algorithms come of age," *Communications of the ACM*, vol. 37, no. 3, pp. 113–120, 1994.

[10] D. E. Rumelhart, B. Widrow, and M. A. Lehr, "The basic ideas in neural networks," *Communications of the ACM*, vol. 37, no. 3, pp. 87–93, 1994.

[11] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," in *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, vol. 1. IEEE, 2004, pp. 325–331.

[12] J. Gruska *et al.*, *Quantum computing*. McGraw-Hill London, 1999, vol. 2005.

[13] 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.

[14] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," *arXiv preprint arXiv:1802.06002*, 2018.

[15] C. Zoufal, A. Lucchi, and S. Woerner, "Quantum generative adversarial networks for learning and loading random distributions," *npj Quantum Information*, vol. 5, no. 1, pp. 1–9, 2019.

[16] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean, "Power of data in quantum machine learning," *Nature communications*, vol. 12, no. 1, pp. 1–9, 2021.

[17] "Scikit-learn machine learning in python," Available: https://scikit-learn.org/, [Online]. Accessed 1 November 2021.