Knowledge Distillation in Quantum Neural Network using Approximate Synthesis

Mahabubul Alam  
The Pennsylvania State University  
University Park, USA  
mxa890@psu.edu

Satwik Kundu  
The Pennsylvania State University  
University Park, USA  
satwik@psu.edu

Swaroop Ghosh  
The Pennsylvania State University  
University Park, USA  
szg212@psu.edu

ABSTRACT
Recent assertions of a potential advantage of Quantum Neural Network (QNN) for specific Machine Learning (ML) tasks have sparked the curiosity of a sizable number of application researchers. The parameterized quantum circuit (PQC), a major building block of a QNN, consists of several layers of single-qubit rotations and multi-qubit entanglement operations. The optimum number of PQC layers for a particular ML task is generally unknown. A larger network often provides better performance in noiseless simulations. However, it may perform poorly on hardware compared to a shallower network. Because the amount of noise varies amongst quantum devices, the optimal depth of PQC can vary significantly. Additionally, the gates chosen for the PQC may be suitable for one type of hardware but not for another due to compilation overhead. This makes it difficult to generalize a QNN design to a wide range of hardware and noise levels.

An alternate approach is to build and train multiple QNN models targeted for each hardware which can be expensive. To circumvent these issues, we introduce the concept of knowledge distillation in QNN using approximate synthesis. The proposed approach will create a new QNN network with (i) a reduced number of layers or (ii) a different gate set without having to train it from scratch. Training the new network for a few epochs can compensate for the loss caused by approximation error. Through empirical analysis, we demonstrate ≈71.4% reduction in circuit layers, and still achieve ≈16.2% better accuracy under noise.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Mahabubul Alam, Satwik Kundu, and Swaroop Ghosh. 2023. Knowledge Distillation in Quantum Neural Network using Approximate Synthesis. In 28th Asia and South Pacific Design Automation Conference (ASPDAC ’23), January 16–19, 2023, Tokyo, Japan. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567877

1 INTRODUCTION
Quantum computing is advancing rapidly. The community is seeking computational advantages with quantum computers (i.e., quantum supremacy) for practical applications. Recently, Google claimed quantum supremacy with a 53-qubit quantum processor to complete a computational task in 200 seconds that might take 10K years [3] (later rectified to 2.5 days [13]) on the state-of-the-art supercomputers. Quantum machine learning (QML) is a promising application domain to archive quantum advantage with noisy quantum computers in the near-term.

A PQC is a quantum circuit with tunable parameterized gates as shown in Fig. 1(b). It generally consists of repeated layers of single-qubit rotations (to explore the search space) and multi-qubit operations (to create entanglement). The parameters of PQC can be tuned to attain desired outputs for given inputs (e.g., classifying data samples). QNN models are claimed to be more expressive compared to the classical neural networks [1, 6]. In other words, QNN models have higher capability to approximate a desired functionality compared to the classical models of similar scale (e.g., with same number of tunable parameters/weights). Therefore, QNN holds great potential to demonstrate quantum advantage in practical applications.

The existing QNNs faces two major adaptability challenges:

- **Noise adaptability:** A large number of PQC layers generally translates to a better QNN performance in noiseless simulation [1, 7]. However, the near-term quantum devices have a limited number of qubits, and they suffer from various errors such as, decoherence, gate, measurement and crosstalk errors. These noises build up quickly as a circuit is scaled up. Therefore, a deep QNN may perform poorly on an actual hardware compared to a shallower network. A considerable noise gradient might exist across different pieces of hardware from the same vendor [4]. On top of that, the noise levels of a device may vary over time. A trained QNN may perform differently on different devices from the same vendor or on the same device over time, making it difficult to select an optimal QNN depth for a particular ML task. Training multiple QNN models with different depths and then selecting one based on current target hardware noise levels can be a naive (and expensive) solution to this problem.

- **Hardware adaptability:** Different quantum hardware may exhibit varying degrees of noise. Furthermore, they may support a completely different set of basis gates (Fig. 1). All other gates must be broken down into their component parts. For example, the CX gate which is a native gate in IBM quantum computers will break into 1 CZ gate and 6 other single-qubit gates for Rigetti devices (Fig. 1). Similarly, a CZ gate will break down into 1 CX and 6 single-qubit gates for IBM devices. As a result, porting a QNN designed for one piece of hardware (e.g., tailored with CX gates for IBM devices) to another (e.g., Rigetti) is problematic. Creating and training multiple device-specific QNN models is also not a viable option.

A somewhat similar problem exists in the classical domain where an expensively trained (e.g., in a server environment with multiple GPU/CPU) deep neural network (DNN) cannot be run on resource constrained mobile/edge devices. Network pruning, weight sharing, quantization, layer fusion, etc. are some of the techniques available in the classical domain to reduce the size of a pre-trained DNN [12]. Knowledge distillation is another popular DNN compression technique [8]. In this case, a pre-trained large DNN is used as a guide to train a smaller DNN without sacrificing much performance.

Inspired from the classical approach, we introduce knowledge distillation in QNN using approximate synthesis to address the adaptability challenges in QNN. The traditional knowledge distillation technique uses original data to train the student network with the
original labels and pre-trained model outputs guiding the training procedure. However, this can result in a similar amount of training time as learning a new model from start. The proposed method avoids such costly training technique by employing approximate synthesis to mimic the pre-trained QNN behavior. Finding an equivalent unitary transformation for an existing quantum circuit using different gates is called approximate synthesis [9]. Once trained, the PQC of a QNN performs a fixed unitary transformation of the input states. Using approximate synthesis, we can find a comparable PQC that has fewer layers or an entirely different set of gates than the original. Here, the original PQC can act as a guide to create a new network that is more compact and noise-resistant, or that is customized for a completely different hardware. Performance may suffer due to approximation errors which can be recovered by training the new network for a few epochs (far cheaper than training a new QNN from scratch).

To the best of our knowledge this is the first work to demonstrate approximate synthesis for knowledge distillation in QNN to address adaptability and resilience challenges.

2 PRELIMINARIES

Qubits, Quantum Gates, Measurements & Quantum Circuit: Unlike a classical bit, a qubit can be in a superposition state i.e., a combination of $|0\rangle$ and $|1\rangle$ at the same time. Quantum gates (e.g., single qubit Pauli-X gate or 2-qubit CNOT gate) modulate the state of qubits and thus perform computations. These gates can perform a fixed computation (e.g., an X gate) or a computation based on a supplied parameter (e.g. the RY($\theta$)). A two-qubit gate (e.g., a CNOT gate) changes the state of one qubit (target qubit) based on the current state of the other qubit (control qubit). Qubits are measured to retrieve the final state of a quantum program.

Quantum Noise: Errors in quantum computing can be broadly classified into, (i) Coherence errors: a qubit can retain its state for a short period (coherence time). The computation needs to be done within this limit. (ii) Gate errors: quantum gates are realized as single-qubit or two-qubit gates that are not native to the target hardware. Hence, the gates in a quantum circuit need to be decomposed into the basis gates before execution. Besides, the native two-qubit operation may or may not be permitted between all the two-qubit pairs. These limitations in two-qubit operations are also known as coupling constraints. Conventional compilers add necessary SWAP gates to meet the coupling constraints. Thus, a compiled circuit depth and gate counts can be significantly higher than the original (synthesis overhead). The compilation overhead is dictated by the native gates/connectivity of the target hardware, and the efficiency of the compiler software.

Quantum Neural Network: QNN generally consists of three segments: (i) a classical to quantum data encoding circuit, (ii) a parameterized circuit (PQC), and (iii) measurement operations. A variety of encoding methods are available in the literature [15]. For continuous variables, the most widely used encoding scheme is angle encoding [1, 11, 15] where a continuous variable classical feature is encoded as a rotation of a qubit along a desired axis (X/Y/Z). For example, RZ($f_1$) on a qubit in superposition is used to encode a classical feature ‘I’ in Fig. 2(b). We can also encode multiple continuous variables in a single qubit using sequential rotations (Fig. 2(c)). States produced by a qubit rotation along any axis will repeat in $2\pi$ intervals (Fig. 2(a)). Therefore, features are generally scaled within 0 to $2\pi$ in a data pre-processing step. One can restrict the values between $-\pi$ to $\pi$ to accommodate features with both negative and positive values.

The PQC consists of multiple layers of entangling operations and parameterized single-qubit rotations. The entanglement operations are a set of multi-qubit operations between the qubits to generate correlated states [11]. The parametric single-qubit operations search through the solution space. This combination of entangling and rotation operations is referred to as a parametric layer (PL) in the literature. The optimal number of PL for any given ML task is generally unknown and is similar to choosing the number of hidden layers/neurons in a classical DNN. In practice, one needs to go through multiple training iterations with different number of PL’s to come up with a compact network. A compact network can offer better noise-resilience/reliability, and lower latency/faster execution.

Figure 1: A toy QNN is shown next to two hardware coupling graphs from IBM and Rigetti. They each offer a distinct set of basis gates. A PQC with CX gates may be a good choice for IBM devices. In Rigetti hardware, however, each CX gate is decomposed into one CZ and six additional single qubit gates. This PQC may not be a good fit for the Rigetti devices due to the noise introduced by each gate during computation on actual hardware. Training multiple hardware-specific QNNs from scratch can be costly.

Figure 2: (a) Bloch sphere representation of a qubit. A qubit can be rotated along the X, Y, or Z axis. The states repeat in $2\pi$ intervals, (b) angle encoding 1:1 (i.e., one continuous variable encoded in a single qubit state), and (c) angle encoding 2:1 (i.e., two continuous variables encoded in a single qubit state).
during inference. There is a wide variety of choices available for PL. We use 19 architectures (proposed in [16]) to evaluate our proposed methodologies (cX refers to circuit X in [16]). Due to the differences in compilation overhead, a PL that works well for one hardware platform may not be ideal for another.

**Knowledge Distillation:** This is a model compression technique [8] where a small (student) model is trained to match a large (teacher) pre-trained model. The teacher model’s knowledge is transferred to the student by minimizing a loss function that aims to match both softened teacher logits and ground-truth labels. The logits are softened by using a “temperature” scaling function in the softmax, effectively smoothing out the probability distribution and revealing the teacher-learned inter-class relationships.

**Approximate Synthesis:** The intended state transformation of a quantum gate can be represented by a unitary matrix. An n-qubit quantum circuit can be represented by a unitary matrix $U$ of size $2^n \times 2^n$ (tensor product of the gate matrices). Quantum circuit synthesis is the process of finding a new circuit with a different set of gates whose unitary matrix representation ($V$) is exact (exact synthesis) or approximate (approximate synthesis) equivalent of the original ($U$). A synthesis algorithm tries to minimize the distance between $U$ and $V$ ($||U - V||$) using metrics such as L1 norm, L2 norm, and frobenius norm [5, 9, 10]. L1 norm is the sum of the absolute element-wise differences ($\sum_{i=1}^{n} \sum_{j=1}^{n} |U_{ij} - V_{ij}|$). L2 norm is the euclidean distance between the matrices ($\sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (U_{ij} - V_{ij})^2}$). If $V$ is the unitary matrix representation of a PQC, the parameters can be updated by a conventional optimizer (e.g., Nelder-Mead, BFGS etc.) to minimize $||U - V||$. Note that, it is not guaranteed that the optimizer will converge to an exact solution (distance = 0) for any given $U$ and the choice of PQC for $V$. More often than not, it will generate a $V$ that is a close approximation of $U$. The outcome will largely depend on the choice of the PQC for $V$, dimension of $U$, chosen optimization method (local/global), and the computing resource/time allocated for optimization.

### 3 RELATED WORKS

The work in [9] introduced approximate synthesis for quantum circuit compression/deep reduction, arbitrary quantum state preparation, and noise reduction. This work showed that quantum circuits with high depth/gate-count but small number of qubits (two/three) can often be approximated with a smaller circuit (lesser gates/depth). Because quantum gates are erroneous, the benefits of circuit size reduction can outweigh the approximation error from such compression. However, the approximation error becomes too large when the number of qubit is increased.

Recent works have used the divide and conquer approach to apply approximate synthesis on larger circuits [5, 17]. Here, a large quantum circuit is divided into smaller sub-circuits (2-4 qubits). Smaller representation of this sub-circuits are found through approximate synthesis. Later, these smaller representations replace the original sub-circuits [17]. It’s worth noting that most previous works have used approximate synthesis to improve arithmetic quantum circuits or state-preparation circuits. These circuits have a very small tolerable margin of error. Since QNN or any other ML model approximates a function (e.g., classification/regression), the margin of error in synthesis can be higher. In fact, it may provide better performance on hardware if the benefits with reduction of gate count/depth outweighs the loss incurred due to approximation. However, there is a lack of research on approximation synthesis of QNN circuits.

### 4 PROPOSED METHODOLOGY

We follow a teacher-student knowledge distillation framework (Fig. 3) [8] where the teacher is the PQC of a pre-trained QNN. The teacher PQC’s unitary matrix representation is the knowledge that is learnt during the training. The student learns to mimic the unitary transformation of the teacher PQC through approximate synthesis without being trained on the original dataset. Later, the loss in performance due to approximation error is mitigated through training the student network for a small number of epochs.

**Overview:** The knowledge distillation process (Fig. 4) starts with a pre-trained QNN and a desired PQC architecture (PQC-2). The pre-trained network can be a large/short-depth QNN with its basic building blocks - a quantum-to-classical data encoding circuit, a PQC with fixed/trained parameters (PQC-1), and measurements. PQC-1 is separated from the pre-trained QNN. We also freeze its parameters. Note that both these PQC’s have the same number of qubits (n) and can be represented by unitary matrices of dimension $2^n \times 2^n$. Since, PQC-1 parameters are frozen, its matrix representation is a fixed unitary. On the other hand, PQC-2 parameters are trainable. Therefore, its corresponding unitary transformation is a function of its parameters. For any random set of values of PQC-2 parameters, the distance between these two unitary matrices can be large. This distance can be calculated as a single scaler quantity using various types of norms, e.g., L1, L2, etc.

After choosing a suitable norm as the distance metric, we can use a classical optimizer to update the PQC-2 parameters and minimize this distance. If the optimization procedure ends with a sufficiently low value of this distance, then we have a new PQC (PQC-2 with optimized parameters) that is a close approximation of PQC-1. After plugging in PQC-2 in place of PQC-1, we get a new QNN that performs approximately similar to the original QNN. If the distance
is not sufficiently small, the new QNN performance can be inferior to the original QNN. To recuperate this loss, we can train the new QNN with the original dataset for a few epochs.

**Distance Metric:** We use following metric (based on the Hilbert-Schmidt inner product) as cost function for approximate synthesis:
\[
d = 1 - \frac{\text{Tr}(U^\dagger V)}{\dim(U)}
\]
where $U$ and $V$ are the unitary matrix representations of PQC-1 and PQC-2. The inverse of a unitary matrix is its conjugate transpose. Therefore, if the synthesis succeeds and $U$ is close to $V$, the product $U^\dagger V = I_N$ where $I$ is the identity matrix and $N = \dim(U)$. Furthermore, the maximum magnitude that the trace of a unitary matrix can have is its size $N$, which occurs at the identity (up to a phase). The closer $U^\dagger V$ is to identity, the closer $\text{Tr}(U^\dagger V)$ is to $N$, and thus, the distance ($d$) closer to $0$. Note, existing works on approximate synthesis have also used this metric [5, 9] because (i) it has computational advantage over other methods, (ii) the distance becomes $0$ if and only if the matrices match exactly, (iii) it scales well with the size of the problem, and (iv) it has operational meaning.

**Distance Minimization:** Any classical optimizer can be used to minimize the distance. We can choose any gradient-based (e.g., Adam, AdaGrad, SGD, BFGS, etc.) or gradient-free (e.g., Nelder-Mead) optimizer for this task. These algorithms, however, execute local searches and are prone to get stuck in local minima. To increase the chances of getting closer to a global minima, one can run multiple procedures with different random seeds with these local optimizers. Alternatively, one can use global optimization methods, e.g., differential evolution, simulated annealing, etc., for better chances of getting an optimal solution at the cost of higher runtime. We use the dual annealing optimizer from the SciPy library which is an extension of the classic simulated annealing.

**Choice of PQC:** Approximate synthesis can be used to (i) compress a QNN (same PL architecture, lower number of layers), and (ii) find an efficient QNN for a target hardware platform (PL tailored for the target hardware). A PQC may be an excellent choice for one hardware platform but unsuitable for another. To further illustrate this issue, we have used two basis gate-sets (IBM & Rigetti), and compiled a total of 19 benchmark PQC circuits from [16] with the qiskit compiler for a hypothetical 4-qubit fully-connected hardware. Fig 5 shows the relative depth and gate counts with identical number of layers. Note that, c2 has the lowest-depth and gate-count with IBM gate-set as shown in Fig 5(a)(&b) (we omit c1 from comparison since it does not have any entanglement). The c6 PQC can be 8.6X deeper and can have 5.6X larger gate count due to compilation overhead. Note that c2 PQC uses CX operation between the neighboring qubits to create entanglement which is native to the IBM gate-set, and therefore, more suitable for this gate-set. Similarly, c9 is suitable for Rigetti gate-set since it uses CZ for entanglement which is native to this gate-set (Fig. 5(c)&(d)). The c2 PQC with a similar number of PL can be 1.7X deeper, and the gate count can be 1.3X higher compared to c9 on Rigetti gate-set. As demonstrated by these results, a QNN optimized for one type of hardware may perform poorly on another if executed naively following compilation due to compilation overhead.

**Approximation Error:** Approximate synthesis may not always produce an exact solution. The outcome will depend on the choice of PQC, the size of the PQC’s in terms of qubits, the chosen optimization procedure, and the computational resource/time allocated for optimization. A very shallow PQC-2 may not have sufficient expressive power to mimic the unitary transformation of PQC-1 [16]. It is noted that approximate synthesis does not scale well with qubit-size [5, 9] due to significant approximation error. Moreover, it may search through a local minima. If a global optimizer is terminated after a certain number of function evaluations, it may not be able to reach an optimal solution. Therefore, approximate synthesis may incur approximation error and compromise the QNN performance.

**Training the Student Network:** Performance loss due to approximation error can be mitigated by training the student model. The compressed model parameters can be used as the initial seed. If the approximation error is minor, a small number of training epochs with the original dataset may be sufficient to recover the majority of the loss else a large number of epochs might be necessary. Nonetheless, the process can be significantly less expensive than training a new model from scratch using random seeds.

## 5 EVALUATION

### 5.1 Setup

**Framework:** We use the PennyLane and TensorFlow packages to train the QNN models. Qiskit is used for circuit compilations and noise simulations. SciPy package is used for optimization. All numerical experiments are run on a Intel Core-i7-10875H CPU with 16GB of RAM. The teacher models are trained for 10 epochs and the student models are trained for 2 epochs using the Adam optimizer (learning rate = 0.2, beta_1 = 0.9, beta_2 = 0.999, epsilon = e^-6). We use the dual annealing optimizer as the classical optimizer in approximate synthesis (initial_temp = 52300.0, restart_temp_ratio = 2e-45, visit = 2.62, accept = -5.0). The parameter values are bounded between -π to π and the maximum number of function calls is restricted to 1000.

**Datasets:** We use the Iris, MNIST, and Fashion-MNIST datasets. The Iris dataset has 150 samples of 3 different classes (50/class). Each sample has 4 features. The MNIST and Fashion-MNIST datasets both have 60000 training samples and 10000 test samples that belong to 10 different classes. Each sample has 28x28 features. To reduce simulation time, we have created 6 smaller 3-class datasets from the MNIST and Fashion-MNIST datasets - MNIST179, MNIST246, MNIST358, FASHION012, FASHION345, and FASHION678. First, we reduce the dimension of MNIST and Fashion-MNIST datasets to 8 using a convolutional auto-encoder [2]. Later, we pick 750 samples of 3 different classes (250/class) to create a smaller dataset (e.g., 750 samples are drawn from classes 0, 1, and 2 for FASHION012). We divide this 750 samples to a train (600) and validation set (150).

**QNN Networks, Qubits, Encoding & Loss Function:** We use 4-qubit QNN models in all the QNN’s. The 19 PQC architectures from [16] are used in our work. We use 1 variables/qubit encoding (Fig. 2(b)) for Iris, and 2 variables/qubit encoding (Fig. 2(c)) for MNIST and Fashion-MNIST. Pauli-Z expectation values of the qubits are taken as the QNN outputs. We feed these outputs to a fully-connected classical dense layer with 3 neurons to facilitate 3-class classification problems [2]. A Softmax layer transforms the output of this neurons to class probabilities. We use the categorical cross-entropy loss. We prefer to evaluate teacher models, student models, and trained student models using classification accuracy on the training and validation sets.

### 5.2 Results

**QNN Retargeting:** Approximate synthesis can be used to find an equivalent QNN to a pre-trained QNN that is more suitable for a target quantum device. The pre-trained QNN might have been trained for a different device. The PL of the new PQC can be chosen based on the basis gates of the new hardware. For example, the c6 PQC consists of several CRX gates which are not native to IBM and Rigetti hardware. A single-layer c6 PQC has 8.6X, and 8.4X higher number of gates than a single-layer c2, and c9 PQC when compiled
Figure 5: Depth and gate-count of the 19 single-layer 4-qubit PQC’s in [16] after compilation with a hypothetical 4-qubit hardware with 3 different gate sets (IBM & Rigetti). Here, cX refers to circuit X in [16].

Figure 6: In (a), (b), and (c), we show the distance between a 4-qubit 2-layer c6 PQC, and its compressed versions with 1-8 layer c2, c9, and c12 PQC’s. In (d), we show the distance between a 10-layer c15 PQC and its compressed versions. (e) shows a similar analysis with c2. In all cases, the base PQC is trained to classify the Iris dataset.

QNN Compression: Since noise increases with the circuit size, compression can be extremely useful to mitigate the impact of noise during inference of the model on an actual hardware. For example, we take 10-layer c15, and c2 PQC’s and trained them to classify the Iris dataset. Later, we approximate these PQC’s with 1-8 layers in Fig. 6(a). Note that, with a single-layer, the distance is quite large (0.093) which decreases consistently with added layers (0.023 with 8-layers). Also note that an 8-layer c2 PQC has less than half the gates of a 2-layer c6 PQC when compiled with the IBM gate-set. We may get to an even lower distance (closer to exact synthesis) by increasing the c2 layer and still maintain the number of gates lower than 2-layer c6 PQC. Note that, with every PQC, we may not explore similar spaces in the Hilbert-space [16]. Therefore, a PQC may perform better over another to approximate the target PQC. For instance, with 8-layers, the c12 PQC gets much closer to the 2-layer c6 PQC compared to c9 as shown in Fig. 6(b)&(c) (0.008 vs. 0.072). Therefore, c12 can be a better choice to approximate c9 for Rigetti devices, both having considerably lower gate count/depth upon compilation with Rigetti gate-set (Fig. 5).

Performance after Approximation and Training: The approximated model performances can be inferior in noiseless simulation. To illustrate this, we trained the MNIST and Fashion-MNIST datasets with 2-layer C6 PQC’s and approximated them with 6-layer c2 and c9 PQC’s (QNN Retargeting). Similarly, we have also trained a 7-layer c15 PQC and later approximated it with 2-5 layers c15 PQC’s (QNN Compression). The results are tabulated in Table 1 and 2. For some datasets (e.g., FASHION345), the approximated models provide similar performance compared to the base model (0.91 and 0.925 accuracy over the training set with 6-layer c2 and c9 compared to 0.9316 with 2-layer c6). However, in some cases (e.g., MNIST179), the performance is significantly compromised (0.5550 and 0.6899 accuracy over the training set with 6-layer c2 and c9 compared to 0.9033 with 2-layer c6). As expected, the performance loss is higher with more compression (Table 2). For example, with 4-layers, the accuracy on MNIST179 training set is 0.4933 which increases to 0.6499 with 5-layers. However, the compressed models performance remains inferior to the 7-layer c15 model for all the datasets. After training the retargeted/compressed models only for two epochs (with approximated model parameters as the initial seeds), we get closer to the base model as evident from Table 1 and 2.

Performance under Noise: To study the performance after approximation, we pick all the 7-layer c15 base models, 2/4 layer approximated models (Ap), and the approximated and trained models (Ap/Tr) from Table 2 and measured their accuracy on two hardware emulators available in Qiskit - FakeMelbourne and FakeAlmaden. We used emulators rather than actual hardware because (i) access to actual hardware is still extremely limited and (ii) generating statistically significant results would require an exorbitantly lengthy period of access to hardware. These hardware emulators model the gate errors (using depolarizing noise channel), decoherence errors (using thermal relaxation), and measurement errors (using bit-flipping probabilities) of the devices. The noise parameters are obtained from actual hardware calibration. FakeMelbourne has the following noise attributes - avg. single/two-qubit gate error: 0.104%/3.14%, avg. T1: 56.07 μs, avg. T2: 55.5 μs, avg. single/two-qubit gate-execution.
Figure 7: Measured average accuracy of the 7L c15 base models, 2L/4L approximated (Ap), and approximated & trained models (Ap/Tr) across all the datasets on two emulators - FakeMelbourne, and FakeAlmaden available in the Qiskit framework.

time: 68.57ns/902.9ns, and avg. measurement error: 5.63%. FakeAlmaden - avg. single/two-qubit gate error: 0.09%/2.38%. Avg. T1: 86.78 μs, avg. T2: 64.31 μs, avg. single/two-qubit gate-execution time: 35.55ns/405.8ns, and avg. measurement error: 5.35%. We show the average of the results over all the datasets in Fig. 7. Note that, for both the training and validation sets, the approximated and trained student models provide significantly higher accuracy compared to the base model. For example, we get 0.726/0.737 (0.68/0.677) average accuracy with 2L/4L models compared to 0.637 (0.573) of the base 7L model on the training (validation) sets.

6 DISCUSSION

The approximation error can grow with qubit size [9, 10]. To illustrate this issue, we took 200 instances of 2, 3, 4, 5, and 6-qubit & 6-layer c2 PQC with random parameters and approximated them with 4-layer PQC’s (40/qubit-size). Later, we measured the fidelity between the states prepared by the 6-layer PQC’s and their 4-layer approximations.

We obtain very low approximation error with 2/3 qubit PQCs, as indicated by the higher average fidelity scores of 0.999 and 0.949, respectively. For 4/5 qubit PQCs the average fidelity are moderate at 0.796 and 0.775. However, 6-qubit PQCs show poor average fidelity score of 0.497. The results indicate that the vanilla approximate synthesis approach can be very useful for QNNs that have higher depth (many PQC layers) but a smaller number of qubits. Many NISQ-era QNN models sacrifice depth for smaller number of qubits [11, 14]. Approximate synthesis can be extremely beneficial for such QNNs. Our approach may be less useful for PQCs with a large number of qubits due to the large approximation error. However, we can always choose the divide and conquer approach in such cases [9, 17].

Table 1: Accuracy of the trained models with two-layer c6 PQC, their approximated counterparts with six-layer c2/c12 PQC’s (Ap), and the approximated & trained models (Ap/Tr)

Table 2: Accuracy of the trained models with seven-layer c15 PQC, its approximated versions with 2, 4, 5 layer PQC’s (Ap), and the approximated & trained models (Ap/Tr)

7 CONCLUSION

We introduce knowledge distillation in QNNs using approximate synthesis to compress a pre-trained QNN for a target device or retrain a pre-trained QNN for a new device, avoiding the expensive training of new networks from scratch. Approximation error can degrade the performance of compressed or retargeted QNNs which can be largely compensated by training for a few epochs. We conduct a comprehensive numerical analysis on 7 datasets with and without noise to evaluate the efficacy of the proposed method.

ACKNOWLEDGMENTS

The work is supported in parts by NSF (CNS-1722557, CNS-2129675, CCF-2210963, CCF-1718474, OIA-2040667, DGE-1723687, DGE-1812766 and DGE-2113839), Intel’s gift and seed grants from Penn State ICDS and Huck Institute of the Life Sciences.

REFERENCES

[1] Amira Abbas et al. 2021. The power of quantum neural networks. Nature Computational Science 1, 6 (2021), 483–489.
[2] Mahabubul Alam et al. 2021. Quantum-Classical Hybrid Machine Learning for Image Classification. arXiv preprint arXiv:2109.02662 (2021).
[3] Frank Arute et al. 2019. Quantum supremacy using a programmable superconducting processor. Nature 574, 7779 (2019), 505–510.
[4] Abdullah Ash-Saki et al. 2019. QURE: Qubit re-allocation in noisy intermediate-scale quantum computers. In 56th Annual Design Automation Conference 2019.
[5] Marc G Davies et al. 2020. Towards optimal topology aware quantum circuit synthesis. In IEEE International Conference on Quantum Computing and Engineering.
[6] Yuxuan Du et al. 2020. Expressive power of parametrized quantum circuits. Physical Review Research 2, 3 (2020), 033125.
[7] Edward Farhi and Hartmut Neven. 2018. Classification with quantum neural networks on near term processors. arXiv preprint arXiv:1802.06062 (2018).
[8] Geoffrey Hinton et al. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015).
[9] Sameer Khatri et al. 2019. Quantum-assisted quantum compiling. Quantum 3 (2019), 140.
[10] Vadym Kliuchnikov et al. 2014. Asymptotically optimal topological quantum compiling. Physical Review Letters 112, 14 (2014), 140504.
[11] Seth Lloyd et al. 2020. Quantum embeddings for machine learning. arXiv preprint arXiv:2001.03622 (2020).
[12] James O’ Neill. 2020. An overview of neural network compression. arXiv preprint arXiv:2006.08649 (2020).
[13] Edwin Pednault et al. 2019. On quantum supremacy. IBM Research Blog 21 (2019).
[14] Adrián Pérez-Salinas et al. 2020. Data re-uploading for a universal quantum classifier. Quantum 4 (2020), 226.
[15] Miaowei Schedl et al. 2021. Effect of data encoding on the expressive power of variational quantum-machine-learning models. Physical Review A (2021).
[16] Sukin Sim et al. 2019. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. Advanced Quantum Technologies 2, 12 (2019), 1900078.
[17] Xin-Chuan Wu et al. 2020. QGo: Scalable Quantum Circuit Optimization Using Automated Synthesis. arXiv preprint arXiv:2012.09893 (2020).