Distributed Quantum Machine Learning

Wai Ming Chan[1]

[1]Department
of Electrical and Computer Engineering, University of Kansas, Lawrence, KS 66045, USA

April 24, 2024

Abstract

Quantum computing in machine learning and optimization has attracted great attention, particularly in the variational quantum algorithms (VQAs) which are implemented by parameterized quantum circuits (PQCs). However, simulation of large-scale quantum circuits is still a challenge. In this project, we investigate distributed quantum machine learning and optimization, and provide a practical model to accelerate the training of VQAs. We first review the distributed method of VQAs called Quantum Distributed Optimization (QUDIO) algorithm and then modify the quantum circuit to a smaller size. We implement the distributed VQA with a 1-layer quantum circuit in pennylane and compare the speedup of the distributed VQA with different numbers of local nodes. We observe that the distributed VQA has a speedup of 3 times with $Q=2,4$ local nodes. We can see the advantage of the distributed VQA in terms of the speedup, but the simulation of middle to large-scale quantum circuits is still a challenge. The current result of the distributed VQA with 1-layer quantum circuit is still worse than classical neural networks.

1 Introduction

Quantum computing in machine learning and optimization has attracted great attention, particularly in the variational quantum algorithms (VQAs) which are implemented by parameterized quantum circuits (PQCs). VQAs have shown efficient implementations on noisy NISQ machines, as well as theoretical guarantees for quantum superiority. Nevertheless, modern VQAs are still facing challenges in terms of expensive or unaffordable quantum resources, such as the number of qubits, the depth of quantum circuits, and the number of quantum measurements. For example, in classical machine learning, the training data is usually in a large volume, and the current quantum circuit might not be able to handle the large-scale data in a single quantum chip. For example for solving classification problems with n labels in quantum neural networks (QNNs), the conventional optimizer requires to query the single quantum circuit at least $O(n)$ times. This overhead prohibits the scalability of QNNs in large-scale classification problems.

A natural way to accelerate the training of VQAs is to parallelize the quantum circuits to fulfill the joint optimization. However, designing such a parallel quantum circuit is non-trivial and it is sharply inconsistent with classical distributed optimization or quantum distributed computing. There are three reasons. First, the gradient operated in VQAs is biased, induced by system imperfection such as sample

error and gate noise, whereas most classical distributed methods assume unbiased gradients. Second, unlike the classical distributed optimization, VQAs are immune to the communication bottleneck, due to the small number of trainable parameters. Third, quantum distributed computation focuses on using multiple less powerful quantum circuits to simulate a standard quantum circuit. Therefore, the focus of this project is to investigate distributed quantum machine learning and optimization, and to provide a survey and outlook on this topic.

2 Problem Statement

We consider the distributed quantum machine learning consists of a central server and multiple local clients $\{\mathcal{Q}_i\}_{i=1}^{Q}$ where each local client consists of a quantum circuit $\mathcal{Q}_i$. The central server is responsible for the global optimization of the quantum circuits, and the local clients are responsible for the local optimization of the quantum circuits. The goal of distributed quantum machine learning is to minimize the global cost function $\mathcal{L}(\theta,\mathcal{D})$, where θ is the global parameter and $\mathcal{D}$ is the global dataset. The dataset $\mathcal{D}=\{\mathbf{x}_i,y_i\}_{i=1}^{N}$ is distributed among the local clients, and the cost function $\mathcal{L}_i(\theta,\mathcal{D})$, for the classification task, is defined as

$$\mathcal{L}(\theta,\mathcal{D})=\frac{1}{N}\sum_{i=1}^{N}(y_i-\mathcal{Q}(\mathbf{x}_i;\theta))^2+\lambda\|\theta\|^2, \tag{1}$$

where $\mathcal{Q}(\mathbf{x}_i;\theta)$ is the quantum circuit, and $\lambda\|\theta\|^2$ is the regularization term.

For original central QNN training, we want to minimize the cost function $\mathcal{L}(\theta,\mathcal{D})$ by solving the optimization problem

$$\min_{\theta}\ \mathcal{L}(\theta,\mathcal{D}), \tag{2}$$

by using gradient-based optimization algorithms, such as stochastic gradient descent (SGD).

For distributed QNN training, we want to minimize the cost function $\mathcal{L}(\theta,\mathcal{D})$ by combining the local model updates or gradients from the local clients, i.e.,

$$\operatorname*{argmin}_{\theta}\mathcal{L}(\theta,\mathcal{D})=f\Big(\{\mathcal{L}_i(\theta_i,\mathcal{D}_i)\}_{i=1}^{Q}\Big), \tag{3}$$

where $f(\cdot):\theta_1\times\cdots\times\theta_Q\mapsto\theta$ is the aggregation function, such as the average or the weighted average, and the local model updates or gradients are computed by the local clients, i.e.,

$$\text{(Local model update)}\qquad \theta_i^{(t)}=\theta_i^{(t-1)}-\eta\nabla\mathcal{L}_i(\theta_i^{(t-1)},\mathcal{D}_i), \tag{4}$$

where η is the learning rate, and $\nabla\mathcal{L}_i(\theta_i^{(t-1)},\mathcal{D}_i)$ is the gradient of the local cost function $\mathcal{L}_i(\theta_i^{(t-1)},\mathcal{D}_i)$.

The distributed quantum machine learning problem is to design the aggregation function $f(\cdot)$ and the local model updates or gradients $\nabla\mathcal{L}_i(\theta_i^{(t-1)},\mathcal{D}_i)$, such that the global cost function $\mathcal{L}(\theta,\mathcal{D})$ is minimized.

Once the local model updates or gradients are computed, the central server aggregates the local model updates or gradients to update the global parameter θ, i.e.,

$$\text{(Global model update)}\qquad \theta^{(t)}=f\Big(\{\theta_i^{(t)}\}_{i=1}^{Q}\Big). \tag{5}$$

3 Basic Review and References

We start with a distributed method of VQAs called proposed **Qu**antum **Di**stributed **O**ptimization (QUDIO) algorithm [1] which designs a distributed VQA to improve the runtime efficiency by distributing the optimization process across multiple local nodes. The QUDIO algorithm adopts the average aggregation function to combine the local model updates, i.e.,

$$\theta^{(t+1)} = f\Big(\{\theta_i^{(t)}\}_{i=1}^{Q}\Big) = \frac{1}{Q}\sum_{i=1}^{Q}\theta_i^{(t)}. \tag{6}$$

The QUDIO algorithm is shown in Algorithm 1, and the circuit of the local node in QUDIO method is shown in Fig. 1.

Algorithm 1: The Pseudocode of QUDIO.

Input: Initial parameters $\theta^{(0)} \in [0,2\pi)^{d_Q}]$, employed loss function $\mathcal{L}$, given dataset $\mathcal{D}$, and the hyper-parameters $\{Q,\mu,T\}$

The central server partitions the given problem into Q parts and allocates them to Q local clients;
the dataset $\mathcal{D}$ is partitioned into $\mathcal{D}_i$ for each local client $\mathcal{Q}_i$;
for $t=0,\cdots,T-1$ **do**
    for *Quantum processor* $\mathcal{Q}_i$, $\forall i \in [Q]$ **do**
        `// Local optimization`
        $\theta_i^{(t,0)} = \theta^{(t)}$; `// Broadcast model parameters to local clients`
        for $x \in \mathcal{D}_i$ **do**
            Compute the estimate gradient $g_i^{(t)}(x) = \nabla \mathcal{L}_i(\theta_i, x)$;
            Update $\theta_i^{(t)} = \theta_i^{(t)} - \mu g_i^{(t)}(x)$;
        end
    end
    $\theta^{(t+1)} = \frac{1}{Q}\sum_{i=1}^{Q}\theta_i^{(t,W)}$; `// Model aggregation`
end

Output: The optimal parameters $\theta^{(T)}$

3.1 Issues and Challenges

We implemented the QUDIO algorithm with the circuit shown in Fig. 1 in pennylane and tried to benchmark the performance with the centralized VQA. However, we found that the running time of the simulation of 10 qubits is too long for MNIST dataset. For the first few iterations, the running time took about a few hours to finish, and the classification accuracy was not as good as the classical neural network.

The main issue is that the quantum circuit used amplitude encoding which is not efficient for large-scale data. In the MNIST dataset, the input vector is of size $28 \times 28 = 784$, and the amplitude encoding requires $\texttt{ceil}(\log_2(784)) = 10$ qubits. Running the quantum circuit with 10 qubits is intense CPU consuming. Thus, we have to change the quantum circuit to a reasonable size so that we can benchmark the performance of the QUDIO algorithm.

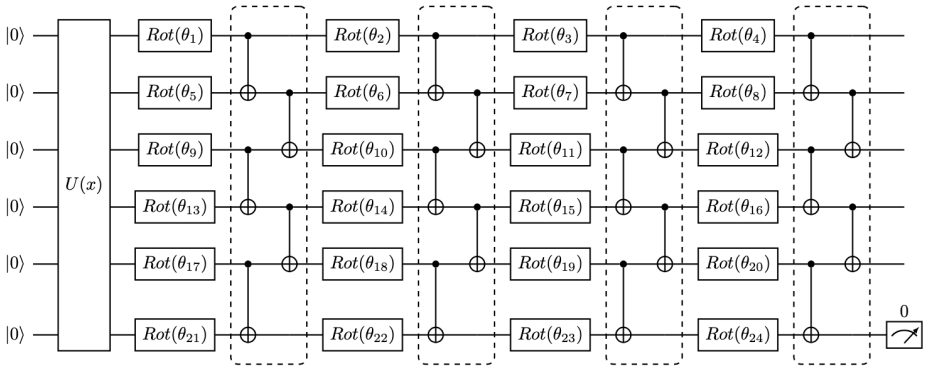

Figure 1: Circuit of the local node in QUDIO method [1]. The amplitude embedding method is represented by the block $U(x)$, and the quantum observable is set as $O = I_{2^5} \otimes |0\rangle\langle 0|$.

4 Modifications

We modified the QUDIO algorithm by changing the quantum circuit to a smaller size. We first simplified the MNIST dataset.

4.1 Data Preprocessing

We first simplified the MNIST dataset by reducing the size of the image to $8 \times 8 = 64$ pixels as shown in Fig. 4. Then, we only consider the first 4 classes of the MNIST dataset, i.e., the digits 0,1,2,3, so that the quantum circuit is reduced to 2 qubits to obtain basis-4 measurements.

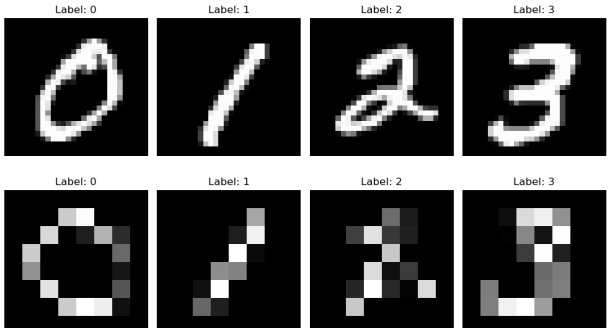

Figure 4: (Top) Original MNIST dataset, $28 \times 28 = 784$ pixels. (Bottom) Resized MNIST dataset, $8 \times 8 = 64$ pixels.

4.2 Quantum Circuit

We then modified the quantum circuit to a smaller size, i.e., 2 qubits. The quantum circuit is shown in Fig. 5. Since the input vector is of size 64, we use the phase encoding method to encode 3 features in each rotation gate, thus we need $6 \times 2 = 12$ rotation gates for phase encoding. After each rotation gate, we rotate the model parameters by model parameters, and then apply the CNOT gates. In the case

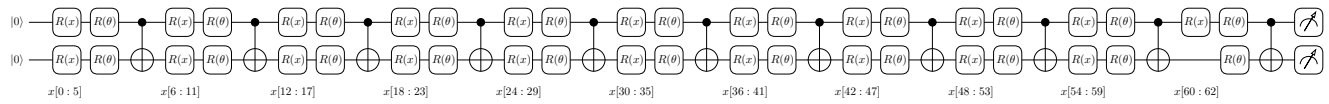

Figure 5: Circuit of the local node with 1 layer case in our modified QUDIO method. The phase encoding is adopted in the quantum circuit followed by the rotation of the model parameters and CNOT gates. The output measurements are used to compute the likelihood of the data belonging to the classes.

of 1 layer quantum circuit, there are a total $36+4=40$ (with bias) trainable parameters. The output measurements are used to compute the likelihood of the data belonging to the classes.

5 Simulation Results and Simulation Code

5.1 Centralized VQA vs Classical Neural Network

Our original plan was to first implement this quantum circuit in pennylane with the same number of trainable parameters as the classical neural network to observe if there is any improvement in the classification accuracy. We consider the classical neural network with 1 input layer and 1 output layer, results in $64\times4+4=260$ trainable parameters. In our VQA circuit, we expand the quantum circuit to 6 layers, results in $66\times3+4=202$ trainable parameters. We then compare the classification accuracy of the classical neural network and the VQA circuit.

The training plot is shown in Fig. 6. It is observed that the VQA has a faster convergence rate than the classical neural network in the loss function. The classification accuracy of the VQA is also better than the classical neural network. However, the VQA has worse performance in the testing dataset, which indicates that the VQA is overfitting the training dataset.

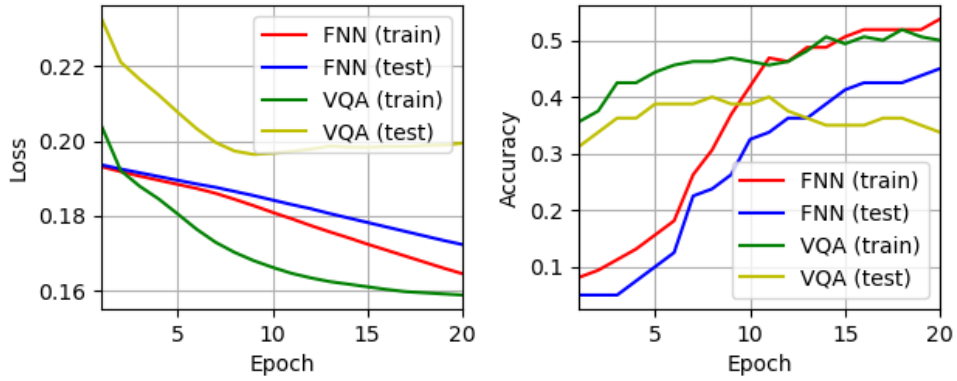

Figure 6: Classical neural network vs centralized VQA: Classical feedforward neural network (FNN) with 260 trainable parameters vs quantum circuit (VQA) with 202 trainable parameters.

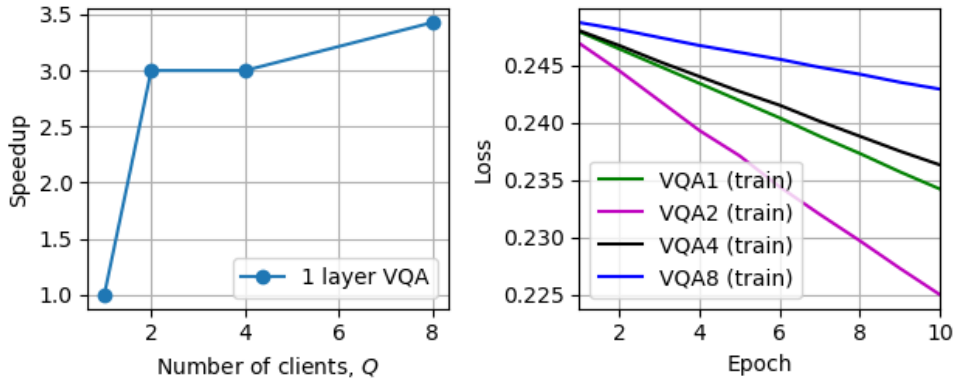

Figure 7: (Left) Speedup of the distributed VQA: 1-layer quantum circuit, different number of local nodes, $Q = 1,2,4,8$. (Right) Training loss of the distributed VQA with 1-layer quantum circuit.

5.2 Distributed VQA

We then implement the distributed VQA with 1-layer quantum circuit as shown in Fig. 5. Since the simulation time for deeper circuit is too long, we only consider the 1-layer quantum circuit. We compare the speedup of the distributed VQA with different number of local nodes, $Q = 1,2,4,8$. The speedup is defined as the ratio of the running time of the centralized VQA to the distributed VQA to achieve the same loss value of 0.245. The speedup plot is shown in Fig. 7. It is observed that the distributed VQA has a speedup of 3 times with $Q = 2,4$ local nodes and a speedup of 3.5 times with $Q = 8$ local nodes. However, the loss performance of 1-layer quantum circuit is worse at around the value of 0.24. It is suggested to train the distributed VQA with deeper quantum circuits to achieve better performance.

The source code of our simulation of the distributed VQA is available in the GitHub repository https://github.com/wai-ming-chan/project_distributedQNN.

6 Conclusion and Future Work

In this project, we investigated distributed quantum machine learning and optimization, and provided a practical model to accelerate the training of VQAs. We first reviewed the distributed method of VQAs called Quantum Distributed Optimization (QUDIO) algorithm and then modified the quantum circuit to a smaller size. We implemented the distributed VQA with a 1-layer quantum circuit in pennylane and compared the speedup of the distributed VQA with different numbers of local nodes. We observed that the distributed VQA has a speedup of 3 times with $Q = 2,4$ local nodes. We can see the advantage of the distributed VQA in terms of the speedup, but the simulation of middle to large-scale quantum circuits is still a challenge. The current result of the distributed VQA with 1-layer quantum circuit is still worse than classical neural networks.

For future work, we plan to implement the distributed VQA with deeper quantum circuits to achieve better performance. We also plan to investigate distributed quantum machine learning with different aggregation functions and local model updates or gradients. We will also investigate different model aggregation functions and local model updates or gradients to improve the performance of the distributed VQA.

References

[1] Y. Du, Y. Qian, and D. Tao, "Accelerating variational quantum algorithms with multiple quantum processors," *arXiv preprint arXiv:2106.12819*, 2021.