QuantumFed: A Federated Learning Framework for Collaborative Quantum Training

Qi Xia

Department of Computer Science
College of William and Mary
Williamsburg, VA 23185, USA
qxia@cs.wm.edu

Qun Li

Department of Computer Science
College of William and Mary
Williamsburg, VA 23185, USA
liqun@cs.wm.edu

Abstract—With the fast development of quantum computing and deep learning, quantum neural networks have attracted great attention recently. By leveraging the power of quantum computing, deep neural networks can potentially overcome computational power limitations in classic machine learning. However, when multiple quantum machines wish to train a global model using the local data on each machine, it may be very difficult to copy the data into one machine and train the model. Therefore, a collaborative quantum neural network framework is necessary. In this article, we borrow the core idea of federated learning to propose QuantumFed, a quantum federated learning framework to have multiple quantum nodes with local quantum data train a mode together. Our experiments show the feasibility and robustness of our framework.

Index Terms—quantum neural networks, federated learning

I. INTRODUCTION

Quantum computing has been greatly developed in recent years. The idea of building a quantum computing based Turing machine was first proposed in early 1980s by Paul Benioff [3]. Many explorations on this area were conducted by Richard Feynman [8], David Deutsch [6], etc. and people started to believe that quantum computing has the capability to beat classic computer in some tasks. In 1994, Shor's algorithm was proposed to factor an integer using a quantum computer in polynomial time, which is exponentially faster than the fastest classic algorithms [19]. Recently, Google AI [1] and USTC [23] claimed quantum supremacy for tasks that are infeasible on any classic computer. In the meantime, deep neural network [14] has been found efficient in many practical tasks such as computer vision [10], [13], [15], [21], natural language processing [4], [5], [7], [20], etc. It uses a hierarchical neural architecture to understand the world and achieves a great success in both industry implementation and academic research.

In recent years, there is a trend of combining deep learning and quantum computing together to reduce the huge computational cost for larger and deeper classic neural networks. For this purpose, the quantum neural network was naturally proposed [11]. A quantum neural network uses the idea of a classical neural network in a quantum way to learn from the training data. By utilizing the main property of qubit superposition and entanglement in quantum mechanics, it tries to improve the computational efficiency and reduce the long training time and heavy computational resources in deep learning [2], [9], [17], [18], [22].

In deep neural network model training, sometimes it is necessary to train a model through multiple machines in a distributed manner, e.g., federated learning [12], [16]. Federated learning is a collaborative way to train a global model where each node has their private local data in classic machine learning. For quantum machine learning, it is natural to use this collaborative approach for collaborative training. In this paper, we learn from this idea and propose a QuantumFed framework. Our contributions are summarized below:

- We propose QuantumFed, a quantum federated learning framework to collaborate multiple quantum nodes with local quantum data together to train a global quantum neural network model.
- We conduct several simulation experiments to show that our QuantumFed framework is capable of collaborating multiple nodes and is robust for noisy data.

II. PRELIMINARIES

A. Quantum Computing Basis

In quantum computing, the qubit is the basic unit to represent the information. A qubit has two basis states $|0\rangle$ and $|1\rangle$ like the classic bit in traditional computer, but it can also be in a superposition, which is a combination of the two basis states: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $\alpha^2 + \beta^2 = 1$. Therefore, a qubit is capable of expressing more information than a classic bit. When observing the qubit, it will collapse to one of the basis states with corresponding probability, and thus we can get a statistically accurate estimation after sufficient times of observations. In addition, the entanglement of qubits allows more qubits to have correlations with each other and n qubits, in this scenario, will have 2^n basis states and can be in a superposition among them, which carries an exponentially increasing amount of information.

In order to perform computations on the qubits, there are several common quantum logic gates: Pauli, Hadamard, Controlled Not. Unlike the AND and OR gate from classic computers, quantum operators are always reversible and will compute an output with the same dimension, and thus can be represented as a unitary. If we represent the input qubits

state as a column vector, for example, $|\psi\rangle = \frac{1}{\sqrt{6}}|00\rangle + \frac{1}{\sqrt{6}}|01\rangle + \frac{1}{\sqrt{3}}|10\rangle + \frac{1}{\sqrt{3}}|11\rangle \rightarrow [\frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}]^T$, the output of operators are corresponding unitary left multiplied states.

B. Quantum Neural Network

There are lots of explorations of implementing deep neural networks in a quantum way. In this article, we adopt a widely used quantum deep neural network architecture as Figure 1. Assume in layer l, the input is a state ρ^{l-1} of m_{l-1} qubits
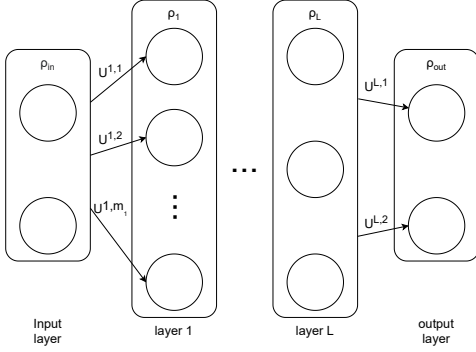

Fig. 1. An architecture example of the quantum neural network.

and this layer will give an output of m_l qubits, then the l-the layer transition map $\mathcal{E}^l$ is given by:

$$\mathcal{E}^l(\rho^{l-1}) = \text{tr}_{l-1}(U^l(\rho^{l-1} \otimes |0\cdots0\rangle_l\langle0\cdots0|)U^{l\dagger}) \quad (1)$$

U^l is the $2^{m_{l-1}+m_l} \times 2^{m_{l-1}+m_l}$ dimensional perceptron unitary of layer l. A partial trace operation is performed to get the output state of layer l. For simplicity, we apply U^l by sequentially applying m_l independent perceptron unitaries $U^{l,j}$ that act on m_{l-1} input qubits and j-th qubit in layer l, that is, $U^l = \prod_{j=m_l}^{1} U^{l,j}$. Note that $U^{l,j}$ here are acting on the current layer, which means $U^{l,j}$ is actually $U^{l,j} \otimes \mathbb{I}^l_{1,\cdots j-1,j+1,\cdots m_l}$. In this way, we can feedforward the input state layer by layer to get an output state:

$$\rho^{out} = \mathcal{E}^{out}(\mathcal{E}^L(\cdots\mathcal{E}^2(\mathcal{E}^1(\rho^{in}))\cdots)) \quad (2)$$

The hyper parameters of the quantum deep neural network are the unitaries, so as long as we have the network structure and unitaries, we can describe a model.

In order to represent the input and output data in a quantum way, for data that is stored by classic bits, we need to first transform the data to qubit representation. One way to do this is that we can use a d-qubits state $|\psi\rangle_d$ to represent a superposition of 2^d basis states in Hilbert space $\mathcal{H}^{2^d}$, that is, $|\psi\rangle_d = \sum_{i=1}^{2^d} \alpha_i|z_i\rangle$ where α_i is the complex amplitude and z_i is a basis state in $\mathcal{H}^{2^d}$.

C. Federated Learning

Federated learning is a special kind of collaborative distributed learning in the machine learning area to train a global model across multiple computational devices or nodes who keep their private local training data. In federated learning, the central server only keeps a global model and does not

keep data. Each node queries a global model at some times, performs local updates on their local data, and uploads the update information to the central server. During one iteration, the central server receives all the computation results and updates the global model by aggregating local updates. In a classic neural network environment, the global model parameter at time t is w_t. Assume there are m participating nodes at time t and node i performs local update to get updated model $w_t^i = \text{LocalUpdate}(w_t)$. The LocalUpdate function is the classic gradient descent for one or more steps on local data. Then after the central server receives all the local updated models, it aggregates those models and updates the global model by $w_{t+1} = \text{GlobalUpdate}(w_t^1, w_t^2, \cdots, w_t^m)$. In practice, we usually simply use a weighted average function as GlobalUpdate to aggregate the uploaded models, where the weight is usually the data volume of each node.

III. QuantumFed Framework

In this section, we will detail our QuantumFed framework.

A. Cost Function

The cost functions in classic neural networks are usually mean squared error loss or cross-entropy loss. Although these cost functions can still be applied in quantum neural networks, they are not easy to compute with quantum operations. Here like [2], we use fidelity as our cost function to measure the difference between label data states and output states. Fidelity represents the probability that one state will be identified as the other state in one measurement. Let $(\phi_x^{in}, \phi_x^{out}), x = 1, 2, \cdots, N$ be the training data and ρ_x^{out} be the output states that are derived by the current quantum neural network with the input data ϕ_x^{in}, the cost function $\mathcal{C}$ is:

$$\mathcal{C} = \frac{1}{N}\sum_{x=1}^{N}\langle\phi_x^{out}|\rho_x^{out}|\phi_x^{out}\rangle \quad (3)$$

(3) measures the closeness between two states. When the output states are not pure, we can use a generalized fidelity function: $\mathcal{C} = \frac{1}{N}\sum_{x=1}^{N}(tr\sqrt{\phi_x^{out1/2}\rho_x^{out}\phi_x^{out1/2}})^2$. Note that the value of the fidelity cost function is between 0 and 1, while 1 expresses the best performance.

B. Local Update

The local update is performed in each node in the quantum federated learning system. The goal of the local update is to maximize the cost function to 1 based on the local dataset in the given steps (interval length). In quantum neural networks, the analogue of the model weight in classic neural networks is the model perceptron unitary U, and the model update is defined as $U \rightarrow e^{i\epsilon K}U$. Here ϵ is the update step size and K is the update matrix. Therefore, each local step is to maximize the cost function by choosing an appropriate update matrix:

$$K = \arg_K \max(\mathcal{C}(e^{i\epsilon K}U, (\phi_x^{in}, \phi_x^{out})) - \lambda\|K\|_2^2) \quad (4)$$

$\mathcal{C}(U, (\phi_x^{in}, \phi_x^{out}))$ is the fidelity cost of a model with perceptron unitary U and local dataset $(\phi_x^{in}, \phi_x^{out})$. The $-\lambda\|K\|_2$ is

introduced by a Lagrange multiplier λ to bound the norm of update matrix K and $\|\cdot\|_2$ is the matrix L_2-norm.

Specifically, on the node n side, let $(\phi_{n,x}^{in}, \phi_{n,x}^{out})$, $x = 1, 2, \cdots, N_n$ be the local training data in node n, $U_n^{l,j}$ be the perceptron unitary of layer l, perceptron j and $U_n^l = \prod_{j=m_l}^1 U_n^{l,j}$, the output states of data x at layer l ρ_x^l equal to $tr_{l-1}(U_n^l(\rho_x^{l-1} \otimes |0 \cdots 0\rangle_l \langle 0 \cdots 0|)U_n^{l\dagger})$ similar to [2], we can derive the update matrix K_j^l by Proposition 1.

Proposition 1. *Let the cost function be the fidelity defined in (3), we can solve (4) using gradient ascent by the following:*

$$K_j^l = \eta \frac{2^{m_{l-1}}i}{N_n} \sum_{x=1}^{N_n} tr_{rest} M_x^{l,j} \tag{5}$$

tr_{rest} *is over all qubits that are not affected by $U_n^{l,j}$ and $M_x^{l,j}$ is computed by $M_x^{l,j} = [\prod_{\alpha=j}^1 U_n^{l,\alpha}(\rho_x^{l-1} \otimes |0 \cdots 0\rangle_l \langle 0 \cdots 0|) \prod_{\alpha=1}^j U_n^{l,\alpha\dagger}, \prod_{\alpha=j+1}^{m_l} U_n^{l,\alpha\dagger}(\mathbb{I}_{l-1} \otimes \sigma_x^l) \prod_{\alpha=m_l}^{j+1} U_n^{l,\alpha}]$. Here $\mathcal{F}^l$ is the adjoint channel to $\mathcal{E}^l$ and $\sigma_x^l = \mathcal{F}^{l+1}(\cdots \mathcal{F}^{out}(|\phi_{n,x}^{out}\rangle\langle \phi_{n,x}^{out}|))$.*

From Proposition 1, we can derive a closed-form update matrix for each perceptron unitary. This update matrix is an analogue of the gradient in the classic neural network, and the way to derive it is like the back-propagation process. Therefore, we can update the perceptron unitary like gradient descent based on local data and Proposition 1 for each step to maximize the cost function.

In classic federated learning, participating nodes are not required to do only one step gradient descent in each iteration. Therefore, we also assume that the local perceptron unitary can update for several steps. Here we define the number of steps as the interval length I_l. Then the local update algorithm QuanFedNode is described in Algorithm 1.

In QuanFedNode algorithm, there are basically two steps:

- Feedforward step. We apply the input state of the training data to the quantum neural network and feedforward it to every qubit by using the perceptron unitaries.
- Temporary update step. We first compute the unitary update matrix K_j^l for layer l, perceptron j by Proposition 1. Then the local temporary update is derived by $U_n^{l,j} \leftarrow e^{i\epsilon K_j^l} U_n^{l,j}$. Meanwhile, we also compute another update unitary at interval k, $U_{n,k}^{l,j} = e^{i\epsilon \frac{N_n}{N_t} K_j^l}$. N_t is the number of data on all participating nodes in this iteration. $U_{n,k}^{l,j}$ is computed for global update and will be sent to the central server later.

The temporary update will be processed in each participating node with their local training data and repeat for I_l times. Then we can simply send the update unitaries $U_{n,k}^{l,j}$ to the central server.

C. Global Update

Global update is performed on the central server side. It maintains a global model that is updated by each node's local data and update unitaries. The goal of the global update is to maximize the cost function based on the global dataset among

---

Algorithm 1 QuanFedNode (Node n Side)

Input:

Network architecture: there are L layers in the quantum neural network and layer l has m_l quantum perceptrons; a copy of network perceptron unitaries from the global model $U_n^{l,j} = U_t^{l,j}$; training data: $(\phi_{n,x}^{in}, \phi_{n,x}^{out})$, $x = 1, 2, \cdots, N_n$; interval length I_l; total number of data among all participating nodes N_t; learning rate η and update step size ϵ;

Output:

Send update unitaries to the central server.

1: Set the interval index $k = 1$;
2: If $k <= I_l$, continue to the next step, otherwise go to step 6;
3: Feedforward the training data at each layer:
   - For every layer l, apply the current channel $\mathcal{E}^l$ to layer $l - 1$: let $U_n^l = \prod_{j=m_l}^1 U_n^{l,j}$;
   - Let $\rho_x^l = tr_{l-1}(U_n^l(\rho_x^{l-1} \otimes |0 \cdots 0\rangle_l \langle 0 \cdots 0|)U_n^{l\dagger})$ and store ρ_x^l for every layer;
4: Temporarily update the network:
   - Compute the unitary update parameter at layer l, perceptron j, K_j^l by (5);
   - Store update unitary at interval k, $U_{n,k}^{l,j} = e^{i\epsilon \frac{N_n}{N_t} K_j^l}$, and temporarily update the network by $U_n^{l,j} = e^{i\epsilon K_j^l} U_n^{l,j}$;
5: Let $k = k + 1$ and go to step 2;
6: Send all stored update unitaries $U_{n,k}^{l,j}$ to the central server.

---

all quantum nodes. Because the data is stored in each node and the central server is not able to access the private local data, the global update can only be computed based on the update unitaries that are uploaded by each node. We describe the QuanFedPS algorithm in Algorithm 2.

Basically there are three major steps in QuanFedPS algorithm:

- Initialization step. At the beginning of the quantum federated training process, the central server first initializes the model parameters (perceptron unitaries) by randomly assigning the value.
- Node selection step. Like the classic federated learning framework, we need to randomly select N_p nodes out of all N nodes who will participate in the current iteration. This can help improve the randomness of the data distribution and decrease the data heterogeneity. Besides, it can reduce the communication cost by selecting fewer nodes.
- Global update step. After participating nodes complete the local training and send update unitaries back to the central server, the central server updates the global model by applying those update unitaries and finishing the current iteration. We will take the global model update for N_s iterations.

Algorithm 2 QuanFedPS (Central Server Side)

Input:

Network architecture: there are L layers in the quantum neural network and layer l has m_l quantum perceptrons; total number of nodes N and number of selected nodes in each iteration N_p; total synchronization iterations N_s; number of training data on node n N_n; interval length I_l;

Output:

The trained quantum neural network.

1: Initialize the network by randomly choosing all the unitaries $U_t^{l,j}$, set the iteration index $t = 1$;
2: If $t <= N_s$, continue to the next step, otherwise go to step 6;
3: Randomly select N_p nodes from all nodes. Assume the set of selected node indexes is S_n, compute the total number of data among all participating nodes $N_t = \sum_{n \in S_n} N_n$. For each selected node, run QuanFedNode algorithm and get update unitaries $U_{n,k}^{l,j}$;
4: Compute the global update unitaries by applying update unitaries from all selected nodes:

$$U^{l,j} = \prod_{k=I_l}^{1} \prod_{n \in S_n} U_{n,k}^{l,j} \qquad (6)$$

5: Update the global model by $U_{t+1}^{l,j} = U^{l,j} U_t^{l,j}$;
6: Let $t = t + 1$ and go to step 2;
7: Output the trained quantum neural network model.

The design of global update is based on the observation that the order of applying update unitaries almost does not matter and the update unitaries almost surely have multiplicative identity property when $\epsilon \to 0$. Theoretically, we have the following lemma.

Lemma 1. *Assume* $U_1 = e^{i\epsilon K_1}, U_2 = e^{i\epsilon K_2}$ *are two update unitaries and* K_1, K_2 *are bounded by the* L_2*-norm, we have* $\lim_{\epsilon \to 0} U_1 U_2 = e^{i\epsilon(K_1+K_2)}$ *at convergence speed* $O(\epsilon^2)$.

Proof. By Taylor's expansion, we have:

$$U_1 = I + i\epsilon K_1 + O(\epsilon^2)$$
$$U_2 = I + i\epsilon K_2 + O(\epsilon^2)$$
$$e^{i\epsilon(K_1+K_2)} = I + i\epsilon(K_1 + K_2) + O(\epsilon^2)$$

Then we have:

$$U_1 U_2 - e^{i\epsilon(K_1+K_2)} = O(\epsilon^2) \qquad (7)$$

From (7), we can derive $\lim_{\epsilon \to 0} U_1 U_2 = e^{i\epsilon(K_1+K_2)}$ at convergence speed $O(\epsilon^2)$. □

From Lemma 1, when ϵ is small enough, we can rewrite the global update unitaries (6) as:

$$U^{l,j} = \prod_{k=I_l}^{1} e^{i\epsilon K_k^{l,j}}, K_k^{l,j} = \frac{\sum_{n \in S_n} N_n K_{n,k}^{l,j}}{\sum_{n \in S_n} N_n} \qquad (8)$$

Here we define $K_{n,k}^{l,j}$ as the update matrix for node n at step k, layer l, and perceptron j. Note that the update matrix $K_{n,k}^{l,j}$ is derived from (5), and (5) is actually an average of the partial trace of $M_x^{l,j}$ for all the local data. Therefore, if we denote the dataset on node n is D_n, and $D_p = \bigcup_{n \in S_n} D_n$, we have:

$$K_k^{l,j} = \frac{\sum_{n \in S_n} (N_n \eta \frac{2^{m_l-1}i}{N_n} \sum_{x \in D_n} tr_{rest} M_x^{l,j})}{\sum_{n \in S_n} N_n}$$
$$= \eta \frac{2^{m_l-1}i}{\sum_{n \in S_n} N_n} \sum_{x \in D_p} tr_{rest} M_x^{l,j}$$

This is equivalent to compute for a local update on the union dataset of the data on all participating nodes in this iteration when $k = 1$. So when the interval length is set to 1, the QuantumFed framework is exactly the same as training on a single quantum machine with all data. However, when the interval length is greater than 1, things become much more complicated. We will discuss this problem in the next subsection.

D. Discussions

1) GD vs SGD: The QuanFedNode algorithm that we described in Algorithm 1 uses all the training data in each update step. Therefore, the training process is more like gradient descent (GD) in classic deep learning. An alternative training method is by randomly choosing a mini-batch of training data in each step, which is an analogue of mini-batch stochastic gradient descent (SGD) in classic deep learning. SGD can solve the biased data distribution problem by introducing randomness and reduce the computational cost. In the experiment part, we will compare GD-type and SGD-type quantum federated training.

2) Interval Length > 1: As we discussed before, when $I_l = 1$, the QuantumFed framework is exactly the same as training on a single machine. When $I_l > 1$, because of the local temporary update, the update unitaries that are computed after the first step are based on the temporarily updated model parameters, which are not the same as each other participating nodes in one iteration. One way to understand it is that since we usually choose a small ϵ, the temporary update is a small perturbation of the perceptron unitaries. Assume the perceptron unitary is U_p and the update unitary is $U_u = e^{i\epsilon K_u}$. By Taylor's expansion, the update perturbation is given by:

$$U_u U_p - U_p = (I + i\epsilon K_u + O(\epsilon^2) - I)U_p$$
$$= (i\epsilon K_u + O(\epsilon^2))U_p$$

Therefore, the perturbation is small compared to the perceptron unitary when ϵ is small and K_u is bounded, and intuitively we can consider the local temporary updated model as a same model. We also show it is feasible to use larger interval length in the experiment, where it reduces the synchronization iterations and accelerates the training speed.

3) Why Unitary: When communicating between the central federated server and quantum nodes, we choose to use update unitary matrix as the model parameters for transmission. The

reasons are below. First, in classic federated learning, each node sends the local updated model, or gradient to the central server. In the quantum neural network, the analogue is the local updated network unitary $U_n^{l,j}$ or updated unitary $U_{n,k}^{l,j}$. Second, the central server can simply apply the update unitaries from each worker to update the global model. It is more convenient and reduces the computations in the central server side. Third, unitary is the basic operation in quantum mechanics and it is easier to optimize in the system level.

4) Learning Rate and Step Size: In the QuanFedNode algorithm, there are learning rate η and step size ϵ to control the update and they have different meanings. ϵ is derived from classic quantum neural network update, and η is derived from how we would like to bound the update matrix K. A larger η leads to a tighter bound of K. However, actually we can rewrite the update unitary by:

$$U_{n,k}^{l,j} = e^{i\epsilon \frac{N_n}{N_t} K_j^l} = e^{i\epsilon \eta \frac{2^{m_l-1}i}{N_t} \sum_{x=1}^{N_n} tr_{rest} M_x^{l,j}} \qquad (9)$$

We can actually combine ϵ and η to one parameter. For convenience and easy to understand, we set $\lambda = 1.0$ at all time and fine tune the step size by adjusting ϵ in practice.

IV. EXPERIMENT

In this section, we conduct simulated experiments for our QuantumFed framework.

A. Environment Setup

Here we use a quantum environment simulated by QuTip library [1] (Quantum Toolbox in Python). We set up our experiment environment in the following aspects.

First, in order to get the training data, similar to [2], we first randomly generate a global unitary U_g which is the unitary we would like to approximate. Then we randomly generate the training data input and apply the global unitary to the input to get the corresponding output. We use the randomly generated input and output pair as the clean training data. The same method is applied to generate the test data. In this way, we can generate clean training data $(|\phi_{n,x}^{in}\rangle, U_g|\phi_{n,x}^{in}\rangle)$ on the node n side, and test data $(|\phi_{test,x}^{in}\rangle, U_g|\phi_{test,x}^{in}\rangle)$ on the central server side. In order to show the robustness of the training, we also pollute a proportion of training data with randomly generated input and output to get noisy training data. Second, as for the quantum neural network architecture, because the experiments that we conduct are in a simulated environment using the classic computer and the computational complexity increases exponentially with the width of the network increases, we choose to train small-size quantum neural networks with width that are not greater than 3. In this section, we choose a network of size 2-3-2. Third, in order to simulate the heterogeneous federated learning environment, we put similar training data into the same node. We first gather all the generated training data from all nodes, sort them by their vector representation value, and divide them to each node in order. In this way, we can somehow guarantee that the data on each node is not i.i.d.

[1]QuTip: https://github.com/qutip/qutip

Fourth, we measure the experiment results using two metrics. First metric is the fidelity cost function that we defined in (3), to show the probability that the output state will be identified as the output label in a measurement. We also adopt another metric mean square error (MSE) that is widely used in classic machine learning as a comparison. The MSE is defined below:

$$\text{MSE} = \frac{1}{N} \sum_{x=1}^{N} \|\rho_x^{out} - |\phi_x^{out}\rangle\langle\phi_x^{out}|\|^2 \qquad (10)$$

We examine our experiments using both metrics on the training and test data respectively to show the performance.

We set $\eta = 1.0$, $\epsilon = 0.1$, $N = 100$, and $N_p = 10$ if not specified.

B. Experiment Results

1) Accuracy: We first show how a 2-3-2 quantum network performs with different interval lengths in Figure 2. Here, the
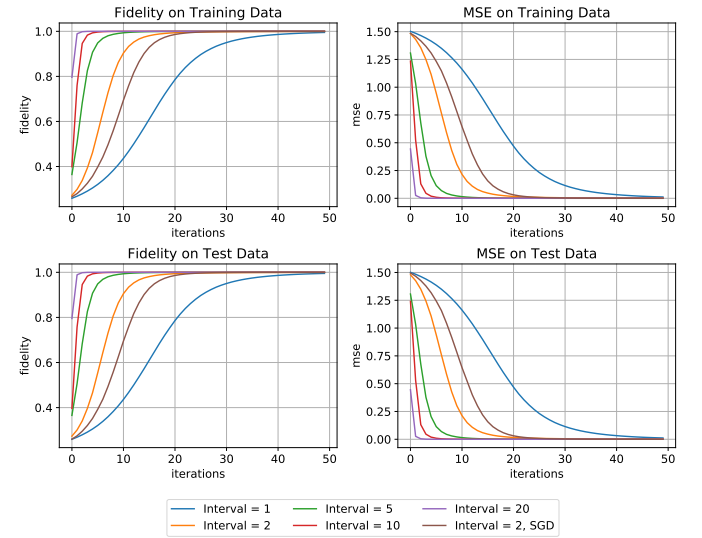

Fig. 2. Experiment results of a 2-3-2 quantum network with different interval lengths.

interval length of 1 case is actually the same as the scenario that is running on a single machine. Therefore, we can see that after 50 iterations, all of them reach fidelity of approximately 1 and MSE of approximately 0 on both training data and test data. This shows that our QuantumFed framework works on collaborating different quantum nodes for training a global model. Besides, we can find that the performance becomes better when we conduct more local steps in each iteration. This is because we have more local training on local data, which learns more information in each iteration. In addition, as a comparison, we also plot the SGD scenario with interval length 2, here we use a mini-batch of 5 for this experiment. We can see that the convergence speed is slower a little bit, which makes sense because we have less data in each iteration, but the final performance is similar. This shows that our framework is feasible for both SGD and GD optimization.

2) Robustness: We then show the performance of a 2-3-2 quantum network with a different ratio of noisy data. We compare the data with 10% noisy data to 90% noisy data on noisy training data and clean test data. The results are in Figure 3. As we can see from the figure, the performance
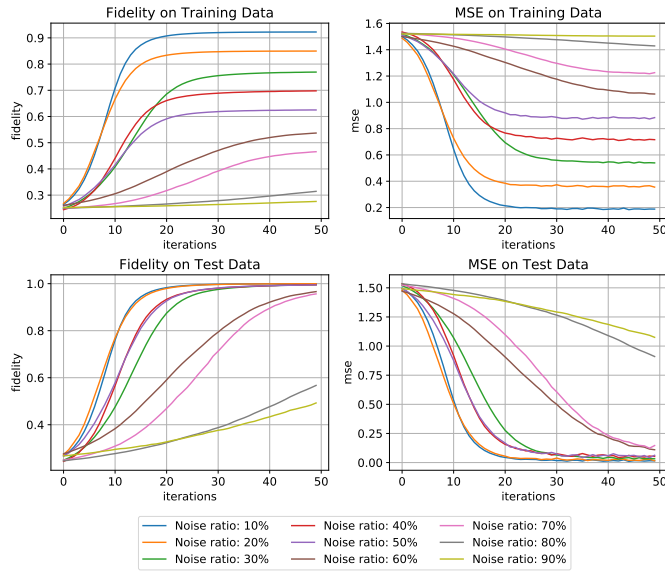

Fig. 3. Experiment results of a 2-3-2 quantum network with different ratios of noisy data.

keeps acceptable when the noise data ratio is no more than 70%, while the final performance is similar when the noise ratio is no more than 50%. This shows the robustness of our QuantumFed framework and it is able to resist with a considerable proportion of noisy data.

V. Conclusion

In this paper, we propose a novel quantum federated learning framework in which multiple quantum nodes collaborate using local quantum data. Several experiments are conducted to show the feasibility and robustness of our QuantumFed framework. With the emergence of quantum computing, the potential of quantum neural networks is enormous. We believe it will be practical to train a deep neural network collaboratively on multiple quantum devices in the near future.

References

[1] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, Oct 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1666-5

[2] K. Beer *et al.*, "Training deep quantum neural networks," *Nature Communications*, vol. 11, p. 808, Feb 2020. [Online]. Available: https://doi.org/10.1038/s41467-020-14454-2

[3] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of Statistical Physics*, vol. 22, pp. 563–591, May 1980. [Online]. Available: https://doi.org/10.1007/BF01011339

[4] Q. Chen *et al.*, "Enhanced LSTM for natural language inference," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1657–1668. [Online]. Available: https://www.aclweb.org/anthology/P17-1152

[5] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 160–167. [Online]. Available: https://doi.org/10.1145/1390156.1390177

[6] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proceedings of the Royal Society of London Series A*, vol. 400, pp. 97–117, Jul. 1985.

[7] J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: http://aclweb.org/anthology/N19-1423

[8] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, pp. 467–488, Jun 1982. [Online]. Available: https://doi.org/10.1007/BF02650179

[9] S. Gupta and R. Zia, "Quantum neural networks," *Journal of Computer and System Sciences*, vol. 63, pp. 355–383, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000001917696

[10] K. He *et al.*, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[11] S. Kak, "On quantum neural computing," *Information Sciences*, vol. 83, pp. 143–160, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002002559400095S

[12] J. Konecný *et al.*, "Federated learning: Strategies for improving communication efficiency," *CoRR*, vol. abs/1610.05492, 2016.

[13] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[15] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[16] H. B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017. [Online]. Available: http://arxiv.org/abs/1602.05629

[17] B. Ricks and D. Ventura, "Training a quantum neural network," in *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16. MIT Press, 2004. [Online]. Available: https://proceedings.neurips.cc/paper/2003/file/505259756244493872b7709a8a01b536-Paper.pdf

[18] M. Schuld, I. Sinayskiy, and F. Petruccione, "The quest for a quantum neural network," *Quantum Information Processing*, vol. 13, pp. 2567–2586, Nov 2014. [Online]. Available: https://doi.org/10.1007/s11128-014-0809-8

[19] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, pp. 303–332, 1999. [Online]. Available: https://doi.org/10.1137/S0036144598347011

[20] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon *et al.*, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[21] A. Voulodimos *et al.*, "Deep learning for computer vision: A brief review," *Computational Intelligence and Neuroscience*, vol. 2018, p. 7068349, Feb 2018. [Online]. Available: https://doi.org/10.1155/2018/7068349

[22] K. H. Wan *et al.*, "Quantum generalisation of feedforward neural networks," *npj Quantum Information*, vol. 3, p. 36, Sep 2017. [Online]. Available: https://doi.org/10.1038/s41534-017-0032-4

[23] H.-S. Zhong *et al.*, "Quantum computational advantage using photons," *Science*, vol. 370, pp. 1460–1463, 2020. [Online]. Available: https://science.sciencemag.org/content/370/6523/1460