Quantum neural networks with deep residual learning

Yanying Liang[1,2], Wei Peng[2], Zhu-Jun Zheng[1,3], Olli Silvén[2], Guoying Zhao[2]
[1] *School of Mathematics, South China University of Technology, Guangzhou 510641, China*
[2] *Center for Machine Vision and Signal Analysis, University of Oulu, Finland*
[3] *Laboratory of Quantum Science and Engineering,*
South China University of Technology, Guangzhou 510641, China

Inspired by the success of neural networks in the classical machine learning tasks, there has been tremendous effort to develop quantum neural networks (QNNs), especially for quantum data or tasks that are inherently quantum in nature. Currently, with the imminent advent of quantum computing processors to evade the computational and thermodynamic limitation of classical computations, designing an efficient quantum neural network becomes a valuable task in quantum machine learning. In this paper, a novel quantum neural network with deep residual learning (ResQNN) is proposed. Specifically, a multiple layer quantum perceptron with residual connection is provided. Our ResQNN is able to learn an unknown unitary and get remarkable performance. Besides, the model can be trained with an end-to-end fashion, as analogue of the backpropagation in the classical neural networks. To explore the effectiveness of our ResQNN , we perform extensive experiments on the quantum data under the setting of both clean and noisy training data. The experimental results show the robustness and superiority of our ResQNN, when compared to current remarkable work, which is from *Nature communications, 2020.* Moreover, when training with higher proportion of noisy data, the superiority of our ResQNN model can be even significant, which implies the generalization ability and the remarkable tolerance for noisy data of the proposed method.

I. INTRODUCTION

Artificial neural networks (ANN) stands for one of the most prosperous computational paradigms [1]. Early neural networks can be traced back to the McCulloch-Pitts neurons in 1943 [2]. Over the past few decades, taking advantage of a number of technical factors, such as new and scalable software platforms [3–7] and powerful special-purpose computational hardware [8, 9], the development of machine learning techniques based on neural networks makes progress and breakthrough. Currently, neural networks achieves significant success and have wide applications in various machine learning fields like pattern recognition, video analysis, medical diagnosis, and robot control [10–13]. One notable reason is the increasingly passion for exploring new neural architectures, including manual ways by expert knowledge and automatic ways by Auto machine learning (AutoML) [14–16]. Particularly, in 2016, deep residual networks (ResNets) are proposed with extremely deep architectures showing compelling accuracy and nice convergence behaviors [14, 17]. Their result won the 1st place on the ILSVRC 2015 classification task for ImageNet classification and ResNet (and its variants) achieves revolutionary success in many research and industry applications.

However, one of the main issues for the current breakthrough of deep neural network are the extravagant computational expense and the physical limitation. With the development of the current quantum technologies and devices, which are believed to have the potential to evade the limitation of the classical computation, thus there is a huge passion to construct deep neural networks on the quantum processors. Quantum computing comes with possible advantages due to the properties of quantum mechanics, such as quantum entanglement, quantum superposition and massive parallelism [18, 19].Therefore, inspired by the success of Neural Networks (NN) in the classical machine learning tasks, there has been tremendous effort to develop quantum neural networks (QNNs), especially for quantum data or tasks that are inherently quantum in nature [20–30].

Most of the research works have done in this field involved a translation of classical neural network components into the language of quantum physics. Earlier quantum neural networks [31, 32] can even trace back to two decades ago. Nowadays, a promising progress can be witnessed in the development of the quantum neural networks. Nevertheless, it is hard to provide an analogue to the back propagation, which works as one of the crucial components for the end-to-end training for the classical neural networks. Besides, most of current works deal with quantum states with ideally clean situation, while it is not always the case in reality. Designing a quantum networks for noisy quantum states is very challenging. On one hand, it is quite hard to identify and characterize the noise sources, on the other hand denoising noisy quantum features will most likely affect the quantum feature negatively for clean parts. Work in [33] builds an efficient quantum feedforward neural network with remarkable ability to learn an unknown unitary and striking robustness to noisy training data. This work is important for noisy intermediate-scale quantum (NISQ) devices due to the reduction in the number of coherent qubits. We are interested in it. The cost function in this work is chosen as the fidelity of quantum states. Nonetheless, we find that the cost function begins decreasing when the proportion of noisy data exceeds 50 percent and the final cost function arrives below 0.25 when all the training pairs are noisy. So we wonder whether the performance of the cost function can be improved or not.

To address the issues mentioned above, in this paper, we design a novel quantum neural network with deep residual learning (ResQNN). Specifically, a multiple layer quantum perceptron with residual connection is proposed. The model can be trained with an end-to-end fashion, as analogue of the backpropagation in classical neural networks. Compared to previous work [33], our method outperforms it for learning an unknown unitary transformation. Besides, the training efficiency can be further improved. Our method also has stronger robustness to noisy training data. One thing worth mentioning is that, when the noisy level is increasing, our network can outperform the state of the art with a larger margin. The major contributions of this paper can be summarized as follows:

- Motivated by the deep residual network and the quantum neural network in [33] , we propose a brand-new quantum neural network with deep residual learning, which can be trained with an end-to-end fashion.
- The proposed quantum model can be more efficient for learning an unknown unitary transformation and more robust to the noisy data.
- Extensive experiments on both clean and noisy data provide the effectiveness of the proposed model. Especially, the superiority can be even significant when the noise levels are higher.

The remainder of this paper is organized as following. Section 2 reviews the related contributions about quantum neural networks and residual scheme. In Section 3, we briefly introduce the basic concepts of quantum qubits and the operators we mainly use in the paper, as well as the principles of swap test and the mechanism of deep residual learning. Section 4 gives the model of quantum neural network with deep residual learning, including its architecture and training algorithm both on classical computer and quantum computer. To validate the improved performance, Section 5 provides the experimental simulation and corresponding analysis. Finally a discussion in Section 6 is drawn.

II. RELATED WORK

A. Quantum Neural Networks

Quantum neural networks have strong potential to be superior to the classical neural network after combining neural computing with the mechanics in quantum computing, especially for quantum data. Specifically, these research achievements mainly focus on some aspects: solving central tasks in quantum learning [22, 29, 33]; enhancing the problem of machine learning [23, 24, 31]; and efficient classification of quantum data [26–28]. Among them, those papers about quantum learning for an unknown unitary transformation impress us a lot. In detail, Bisio and Chiribella [29] addressed this task and found optimal strategy to store and retrieve an unknown unitary transformation on a quantum memory. Soon after, Sedlák *et al.* [30] designed an optimal protocol of unitary channels, which generalizes the results in [29]. Moreover, Beer *et al.* [33] proposed a quantum neural network with remarkable generalisation behaviour for the task of learning an unknown unitary quantum transformation. Since in mathematical theory of artificial neural networks, the universal approximation theorem [34] tells us that a neural network composed of classical perceptrons can approximate any function. So it is natural to think about this ability for quantum neural network.

On the other hand, NISQ technology wins a fruitful discussion by academia and the industry, as a useful tool to explore multi-body quantum physics. Quantum computers with 50-100 qubits may be superior to today's classical computers when performing tasks, however the size of quantum circuits which can have a reliable execution will be restricted by the noise in quantum gates [35]. New allocation algorithms for different qubits with corresponding connectivity and error rate were established in [36–38], which concern the qubits mapping problems. Besides the task of learning an unknown unitary quantum transformation mentioned above, Beer *et al.* [33] also focused on the inevitable noise with the NISQ devices and designed a robust quantum neural network for approximate depolarising noise. However, the fidelity in their work decreased quickly with the number of noisy data increasing and the final value was less than 0.25 when the training pairs were all noisy, which is the problem addressed by our work.

B. Residual scheme in Neural Networks

Proposed in 2012, AlexNet [39] became one of the most famous neural network architecture in deep learning era. This was treated as the first time that deep neural network was more successful than traditional, handcrafted feature learning on the ImageNet [40]. Since then, researchers make many efforts to make the network deeper, as deeper architecture could potentially extract more important semantic information. But deeper networks are more difficult to train, due to the notorious vanishing/exploding gradient problem. Residual scheme in ResNet [14] is one of the most successful strategy of improving current Neural Networks. ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance. Basically, this scheme reformulates the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. Based on this, a residual neural network builds on constructs known from pyramidal cells in the cerebral cortex, utilizing skip connections, or shortcuts to jump over some layers. This simple but efficient strategy largely improves the performance of current neural architectures in many fields.

For instance, in image classification tasks, many variants of Resnets [14, 41–43] are proposed and get the state-of-the-art performance. This scheme is even introduced into graph convolutional networks. For instance, works from [16, 44, 45] also endow graph convolutional network with the residual connections, capturing a better representations for skeleton graphs. Nevertheless, as far as we know, there is no work introducing the Residual scheme in the filed of Quantum Neural Networks, since it is non-trivial to introduce it into quantum computing and provide an analogue to the back propagation. In this paper, we will make the first attempt to do this and present a ResQNN which can be trained with an end-to-end fashion.

III. PRELIMINARIES

A. Qubits and quantum gates

Analogous to the role bit is the smallest unit of classical computing, qubit is the smallest unit in quantum computing. The notation $|\cdot\rangle$ is called a ket which is used to indicate that the object is a column vector. The complex conjugate transpose of $|\cdot\rangle$ is $\langle\cdot|$, which is called a bra [46]. A two-level quantum system in a two-dimensional Hilbert space $\mathbb{C}^2$ with basis $\{|0\rangle, |1\rangle\}$ is a single qubit, which can be written from the superposition principles:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\alpha|^2 + |\beta|^2 = 1, \alpha, \beta \in \mathbb{C}.$$

In order to train our quantum algorithm on a quantum computer, we need to introduce some elementary quantum gates. The first interesting quantum gate is the Hadamard gate H, which acts on a single qubit. It maps the basis state $|0\rangle$ and $|1\rangle$ into $(|0\rangle + |1\rangle)/\sqrt{2}$ and $(|0\rangle - |1\rangle)/\sqrt{2}$, respectively. The representation of Hadamard matrix is $H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$. Since $H \cdot H^\dagger = Id$ where Id is the identity matrix in single qubit, then H is a unitary matrix. Swap gate is another useful two-qubit quantum gate used in this paper, which swaps two qubits. The corresponding matrix of swap gate is $SWAP = \begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}$ with a basis of $|00\rangle$, $|01\rangle$,$|10\rangle$ and $|11\rangle$.

B. Tensor product, reduced density operator and partial trace

Tensor product is a way to extend the dimension of vector spaces through putting vector space together [46]. The symbol for tensor product is denoted by $\otimes$. Assume A is an m by n matrix, B is a p by q matrix, then $A \otimes B$ is an $m \cdot p$ by $n \cdot q$ matrix:

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1n}B \\ A_{21}B & A_{22}B & \cdots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{m1}B & A_{m2}B & \cdots & A_{mn}B \end{bmatrix}.$$

Another important operator used in this paper is the reduced density operator [46]. It is often used to get the desired subsystems of a composite quantum system. Assume $\mathcal{H_A}$ and $\mathcal{H_B}$ are two Hilbert spaces. The state in $\mathcal{H_A} \otimes \mathcal{H_B}$ is described by density matrix ρ_{AB}. The reduced density operator for $\mathcal{H_A}$ is given by

$$\rho_A = \mathrm{tr}_B(\rho_{AB}),$$

where tr_B is a map of operators called partial trace over $\mathcal{H_B}$. Here partial trace is defined as

$$\mathrm{tr}_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) = |a_1\rangle\langle a_2|\,\mathrm{tr}(|b_1\rangle\langle b_2|),$$

where $|a_1\rangle$ and $\langle a_2|$ are any vectors in $\mathcal{H_A}$, $|b_1\rangle$ and $\langle b_2|$ are any vectors in $\mathcal{H_B}$. For example, $\mathcal{H_A}$ and $\mathcal{H_B}$ are both two-dimensional complex vector space $\mathbb{C}^2$, then

$$\rho_A = \mathrm{tr}_B(|0\rangle_A\langle 0|\otimes|0\rangle_B\langle 0|) = |0\rangle_A\langle 0|\,\mathrm{tr}(|0\rangle_B\langle 0|) = \begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix},$$

$$\rho_B = \mathrm{tr}_A(|0\rangle_A\langle 0|\otimes|1\rangle_B\langle 1|) = |1\rangle_B\langle 1|\,\mathrm{tr}(|0\rangle_A\langle 0|) = \begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}.$$

C. Quantum circuit of swap test

Swap test was firstly proposed by Buhrman *et al.* in 2001 [47]. It is often used to estimate the probabilities or amplitudes of certain desired quantum states. Here we rephrase how it works.

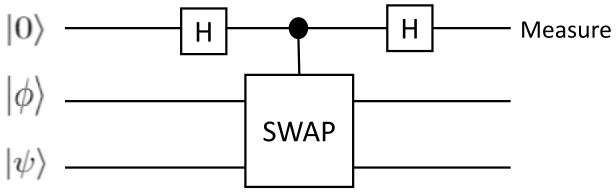


FIG. 1: **Quantum circuit of swap test.**

As shown in Fig.1, the input states are $|\phi\rangle$ and $|\psi\rangle$, and $|0\rangle$ is an additional ancillary qubit in the following process. Here H is the Hadamard gate. *c-SWAP* is the controlled-SWAP, which is controlled by the ancillary qubit.

Step1. Initialize the three qubits in the state $|0\rangle|\phi\rangle|\psi\rangle$.

Step2. Apply the Hadamard gate and end up with the state:

$$\frac{1}{\sqrt{2}}(|0\rangle|\phi\rangle|\psi\rangle + |1\rangle|\phi\rangle|\psi\rangle).$$

Step3. Apply the controlled-SWAP and get the result:

$$\frac{1}{\sqrt{2}}(|0\rangle|\psi\rangle|\phi\rangle + |1\rangle|\psi\rangle|\phi\rangle).$$

Step4. Apply the Hadamard gate the second time and obtain the final state before the measurement:

$$\frac{1}{2}|0\rangle(|\phi\rangle|\psi\rangle + |\psi\rangle|\phi\rangle) + \frac{1}{2}|1\rangle(|\phi\rangle|\psi\rangle - |\psi\rangle|\phi\rangle).$$

Step5. Measure the first qubit of the final state and get 0 with probability p_0:

$$p_0 = \frac{1}{2}(1 + |\langle\phi|\psi\rangle|^2).$$

If $|\phi\rangle$ and $|\psi\rangle$ are orthogonal, then $p_0 = \frac{1}{2}$. If the two states are the same, then the probability p_0 is 1. So the swap test is often used to check how much two quantum states differ or close.

D. Deep residual learning

It is harder to train a deeper neural network. He *et al.* [14] proposed neural networks with deep residual learning framework, which is easier to optimize and obtain high accuracy from increased depth. In a residual block structure, there is a shortcut pathway connecting the input and output of a block structure. Specifically, residual learning chooses to fit the residual mapping F(x):=H(x)-x, rather than approximating the desired underlying mapping H(x) directly. The final mapping of a residual block structure is F(x)+x, which is equivalent to H(x), see Fig.2. As explained in [14], optimizing the residual mapping F(x) is easier than the original mapping H(x), especially when H(x) is identity.

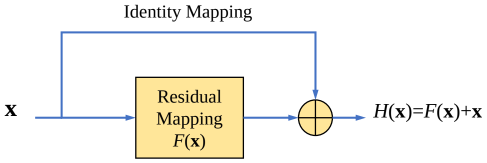


FIG. 2: **Residual block structure.**

IV. THE MODEL OF QUANTUM NEURAL NETWORK WITH DEEP RESIDUAL LEARNING

To our knowledge, there are few researches combining quantum computing with deep residual learning. How can we design a quantum neural network with deep residual learning? Is the performance efficient using deep residual learning in quantum neural network? In the following we will give answers for these two questions. We firstly introduce the architecture of our quantum neural network with deep residual learning (ResQNN) and then explain the training algorithm of our ResQNN as well as an example of this training algorithm. Moreover, we also turn the training algorithm into quantum circuit so that it can be implemented on a quantum computer.

A. The architecture of quantum neural network with deep residual learning

There are several problems to think deeply when designing a ResQNN:

- The input and final mapping of the residual block structure in ResQNN.
- The strategy to apply the identity operation in the residual block structure.
- The trick for combining quantum neural network with the residual block structure.

With the three problems in mind, we design the ResQNN as following. Our ResQNN has L hidden layers. The perceptron nodes of each layer represent single qubits. Denote m_l as the number of nodes in each layer l and we assume $m_{l-1} \le m_l$ for $l = 1, 2, \cdots, L$. Here $l = 0$ represents the input layer and $l = L + 1$ corresponds to output layer. Denote $\rho^{l_{in}}$ and $\rho^{l_{out}}$ represent the input and output state of layer l in our ResQNN. Based on the assumptions and notations above, the residual block structure is designed in Fig.3.

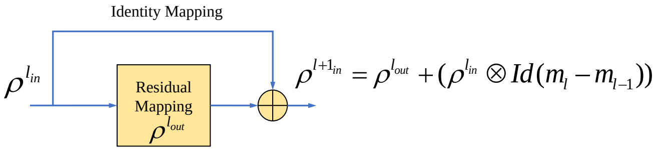

Identity Mapping
$\rho^{l_{in}}$
Residual
Mapping
$\rho^{l_{out}}$
$\rho^{l+1_{in}} = \rho^{l_{out}} + (\rho^{l_{in}} \otimes Id(m_l - m_{l-1}))$

FIG. 3: **Residual block structure of ResQNN.**

Mathematically, we set the input mapping and final mapping of the residual block structure in ResQNN as $\rho^{l_{in}}$ and $\rho^{l+1_{in}}$, respectively. The residual mapping is chosen as $\rho^{l_{out}}$. It is important to note that the output of the former layer is not the exact input of the next layer. As shown in Fig.3, the new input of layer l+1 is the addition of the output state in layer l and the input state in layer l. Here $Id(m_l - m_{l-1})$ represents the identity matrix with $m_l - m_{l-1}$ qubits and the additive operation corresponds to the matrix element-wise addition. We apply tensor product to ρ^l_{in} and $Id(m_l - m_{l-1})$ in order to keep the rule of matrix element-wise addition.

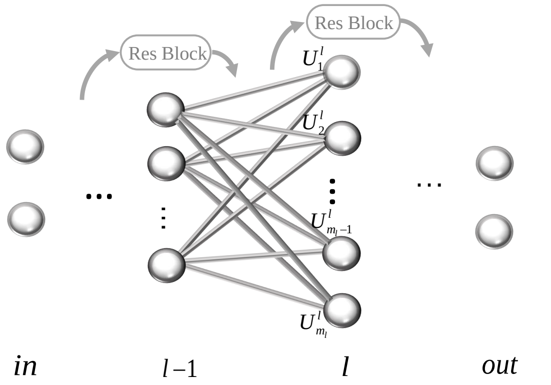


FIG. 4: **The architecture of a multi-layer quantum neural network with deep residual learning**. The "Res block" represents the residual block structure of ResQNN in Fig.3. On one hand, the input state $\rho^{l_{in}}$ of layer l goes through the feedforward neural network from left to right in order to get the output state $\rho^{l_{out}}$ of layer l. On the other hand, we apply the residual block structure to $\rho^{l_{in}}$ and $\rho^{l_{out}}$ in order to obtain the new input state $\rho^{l+1_{in}}$ for next layer $l+1$. The architecture of ResQNN propagates information from input to output and gradually goes through a quantum feedforward neural network.

Next, we go on investigating the strategy to merge the residual block structure and quantum neural network together. The quantum perceptron in our ResQNN is an arbitrary unitary operator with m input qubits and one output qubit. The architecture is presented in Fig.4. Our ResQNN is made up of quantum perceptrons with L hidden layers of qubits. The ResQNN acts on an input state $\rho^{1_{in}}$ of input qubits and obtains a mixed state $\rho^{L+1_{out}}$ for the output qubits based on the layer unitary U^l in the form of a matrix product of quantum perceptrons: $U^l = U^l_{m_l} U^l_{m_l-1} \cdots U^l_1$. Here U^l_j acts on the qubits in layer $l-1$ and l for $j = 1, 2, \cdots, m_l$. At the same time, the residual block structure produces the new input state for layer $l+1$ through acting on the input and output states of layer l so that the layer unitary can directly work. For example, we consider ResQNN with one hidden layer in Fig.5.

Define the layer unitary acting on the qubits in the input layer and the hidden layer $U^1 = U^1_3 U^1_2 U^1_1$, which is in the form of a matrix product of quantum perceptrons. Analogously we define the layer unitary $U^2 = U^2_2 U^2_1$ acting on the qubits in the hidden layer and the qubits in the output layer. In this ResQNN, we apply the quantum perceptrons layer-wise from top to bottom, then the output state $\rho^{1_{out}}$ for the hidden layer is

$$\rho^{1_{out}} = \mathrm{tr}_{in}(U^1(\rho^{1_{in}} \otimes |000\rangle_{hid}\langle 000|)U^{1^\dagger}).$$

Next, we apply residual block structure to $\rho^{1_{in}}$ and $\rho^{1_{out}}$ in order to get the new input state for the output layer:

$$\rho^{2_{in}} = \rho^{1_{out}} + \left(\rho^{1_{in}} \otimes Id(1)\right),$$

where $Id(1)$ means the identity matrix with single qubit. In the third step, we get the final output state of this ResQNN in Fig.5:

$$\rho^{2_{out}} = \mathrm{tr}_{hid}(U^2(\rho^{2_{in}} \otimes |00\rangle_{out}\langle 00|)U^{2^\dagger}).$$

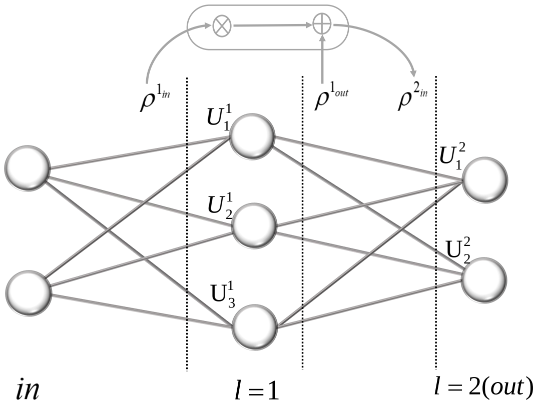


FIG. 5: **The architecture of the ResQNN with one hidden layer.** "⊗" represents the tensor product of $\rho^{1_{in}}$ and $Id(1)$. "⊕" corresponds to the matrix addition of $\rho^{1_{in}} \otimes Id(1)$ and $\rho^{1_{out}}$.

From the analysis above, several remarks should be mentioned for easy understanding:

- The unitary operators are arbitrary, and they do not always commute, so the order of the layer unitary is important.
- Since the residual block structure works on the input and output of the hidden layer, we need the ResQNN with as least one hidden layer.
- We do not choose to apply the residual block structure to the last output layer for better performance in experiments.

B. The training algorithm of quantum neural network with deep residual learning

We randomly generalize N pairs training data which are possibly unknown quantum states in the form of $(|\phi^{in}_x\rangle, |\phi^{out}_x\rangle)$ with $x = 1, 2, \cdots, N$. It is also allowed to use enough copies of training pair $(|\phi^{in}_x\rangle, |\phi^{out}_x\rangle)$ of specific x so that we can overcome quantum projection noise when computing the derivative of the cost function. Here for simplicity, we do not allow input states interacting with environment to produce output states (e.g., thermalization). We choose to consider the desired output $|\phi^{out}_x\rangle$ as $|\phi^{out}_x\rangle = V|\phi^{in}_x\rangle$ with V an unknown unitary operation.

The cost function we choose is the same as Ref.[33], which is the fidelity between the output of ResQNN and the desired output averaged over all training data:

$$C(s) = \frac{1}{N}\sum_{x=1}^{N} \langle \phi^{out}_x | \rho^{out}_x(s) | \phi^{out}_x \rangle.$$

If the cost function comes to 1, we judge the ResQNN performs best, otherwise 0 the worst. Since we want to know how close the network output state and the desired output state, and the closer they are, the bigger fidelity is. So our goal is to maximize the cost function in the training process.

For each layer l of ResQNN, denote $\rho_x^{l_{in}}$ as the input state of layer l and $\rho_x^{l_{out}}$ as the output state of layer l with $l = 1, 2, \cdots, L$ and $x = 1, 2, \cdots, N$. The training algorithm for ResQNN is given by the following steps:

I. Initialize:

I.1 Set step $s = 0$.

I.2 Choose all unitary $U_j^l(0)$ randomly, $j = 1, 2, \cdots, m_l$, where m_l is the number of nodes in layer l.

II.For each layer l and each training pair $(|\phi_x^{in}\rangle, |\phi_x^{out}\rangle)$, do the following steps:

II.1 Feedforward:

II.1a Tensor the input state $\rho_x^{l_{in}}(s)$ to the initial state of layer l,

$$\rho_x^{l_{in}}(s) \otimes |0 \cdots 0\rangle_l \langle 0 \cdots 0|.$$

Here $\rho_x^{1_{in}}(s) = |\phi_x^{in}\rangle\langle\phi_x^{in}|$.

II.1b Apply the layer unitary between layer $l-1$ and l,

$$\begin{aligned} U_{apply}^l(s) =& U_{m_l}^l(s) U_{m_l-1}^l(s) \cdots U_1^l(s) (\rho_x^{l_{in}}(s) \otimes \\ & |0 \cdots 0\rangle_l \langle 0 \cdots 0|) {U_1^l}^\dagger(s) \cdots {U_{m_l-1}^l}^\dagger(s) {U_{m_l}^l}^\dagger(s). \end{aligned}$$

II.1c Trace out layer $l-1$ and obtain the output state $\rho_x^{l_{out}}(s)$ of layer l,

$$\rho_x^{l_{out}}(s) = \mathrm{tr}_{l-1}\left(U_{apply}^l(s)\right).$$

II.2 Residual learning:

II.2a Apply the residual block structure in Fig.3 to $\rho_x^{l_{in}}(s)$ and $\rho_x^{l_{out}}(s)$ to obtain the new input state of layer $l+1$,

$$\rho_x^{l+1_{in}}(s) = \rho_x^{l_{out}}(s) + \left(\rho_x^{l_{in}}(s) \otimes Id(m_l - m_{l-1})\right).$$

Here $m_l - m_{l-1}$ is the number of qubits for identity matrix.

II.2b Store $\rho_x^{l+1_{in}}(s)$.

III Update parameters:

III.1 Compute the cost function:

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

III.2 Update the unitary of each perceptron via

$$U_j^l(s+\epsilon) = e^{i\epsilon K_j^l(s)} U_j^l(s).$$

Here $K_j^l(s)$ is the parameters matrices:

$$K_j^l(s) = \eta \frac{2^{m_{l-1}}}{N} \sum_{x=1}^{N} \mathrm{tr}_{rest}\, M_j^l,$$

where the trace is over all qubits of ResQNN which are not affected by U_j^l. η is the learning rate and N is the number of training pairs. Moreover, M_j^l is made up of two parts of the commutator:

$$\begin{aligned} M_j^l(s) =& [U_j^l(s) U_{j-1}^l(s) \cdots U_1^l(s) \left(\rho_x^{l_{in}}(s) \otimes |0 \cdots 0\rangle_l \langle 0 \cdots 0|\right) \\ & {U_1^l}^\dagger(s) \cdots {U_{j-1}^l}^\dagger(s) {U_j^l}^\dagger(s), {U_{j+1}^l}^\dagger(s) \cdots {U_{m_{out}}^{out}}^\dagger(s) \\ & \left(Id(m_{l-1}) \otimes |\phi_x^{out}\rangle\langle\phi_x^{out}|\right) U_{m_{out}}^{out}(s) \cdots U_{j+1}^l(s)]. \end{aligned}$$

III.2 Update $s = s + \epsilon$.

IV Repeat steps **II** and **III** until reaching the maximum of the cost function.

Here one can find that $K_j^l(s)$ and $M_j^l(s)$ have the same formulas in Ref.[33]. Because the residual block structure in ResQNN has impact on the input of hidden layers. However when we calculate the parameter matrices $K_j^l(s)$ in the sense of the definition of derivation, we find that the trick of residual learning does not change $K_j^l(s)$ at all. To clarify more, we consider a simple example for the training algorithm of a three-layer ResQNN in Fig.5, which is presented in the appendix.

C. Training algorithm on a quantum computer

In this subsection, we turn our training algorithm into quantum circuit so that we can implement our algorithm on a quantum computer.

To estimate how close is the network output state ρ_x^{out} to the correct output $|\phi_x^{out}\rangle$, we use the fidelity for quantum states as a cost function: $C(s) = \frac{1}{N}\sum_{x=1}^{N}\langle\phi_x^{out}|\rho_x^{out}(s)|\phi_x^{out}\rangle$. In Subsection 3.3, we have mentioned that swap test is good at estimating how much two quantum states differ or close, so here we will use swap test as a key to design this quantum circuit to compute the cost function. Based on the analytic formula for cost function, at the beginning, we need to obtain the desired output state $|\phi_x^{out}\rangle$ and the output state ρ_x^{out} of ResQNN. The input state of the quantum circuit is $|\phi_x^{in}\rangle$ in a register of m qubits for $x = 1, 2, \cdots, N$. In order to obtain $|\phi_x^{out}\rangle$ and ρ_x^{out} at the same time, we need another register of m qubits. Moreover, an additional ancillary qubit is also needed to conduct the swap test. So in this quantum circuit, we make use of $2m + 1$ qubits in total.

Step1 Prepare two copies of quantum states $|\phi_x^{in}\rangle$ in m qubits with probability $\frac{1}{N}$.

Step2 Apply an unknown unitary operator V on the first m qubits to get the desired output state $V|\phi_x^{in}\rangle = |\phi_x^{out}\rangle$.

Step3 Apply the ResQNN on the last m qubits to get the network output state ρ_x^{out}. As mentioned in Subsection 4.1, we assume each perceptron nodes are single qubits and m_l represents the number of qubits in layer l for $l = 1, 2, \cdots, L$.

Step.3a Tensor m_l qubits in state $|0\rangle$ with the input of layer l,

$$\rho_x^{l_{in}} \otimes |0\cdots 0\rangle_l \langle 0 \cdots 0|.$$

Resources: In this step $m_{l-1} + m_l$ qubits are needed.

Step.3b Apply the unitary operators in layer l,

$$U_{apply}^l = (\prod_{k=1}^{m_l} U_k^l)\rho_x^{l_{in}} \otimes |0\cdots 0\rangle_l \langle 0 \cdots 0| (\prod_{k=1}^{m_l} U_k^l)^\dagger.$$

Resources: We need $m_{l-1} + m_l$ qubits and m_l gates.

Step.3c Take the partial trace over layer $l-1$,

$$\rho_x^{l_{out}} = \mathrm{tr}_{l-1}(U_{apply}^l).$$

Resources: We go from $m_{l-1} + m_l$ qubits to m_l qubits with no gates.

Step.3d Apply the element-wise matrix addition to $\rho_x^{l_{out}}$ and $\rho_x^{l_{in}}$,

$$\rho_x^{l+1_{in}} = \rho_x^{l_{out}} + \left(\rho_x^{l_{in}} \otimes Id(m_l - m_{l-1})\right).$$

Here $Id(m_l - m_{l-1})$ is the identity matrix with $m_l - m_{l-1}$ qubits.
Resources: We need m_l qubits and no gates.

To get $\rho_x^{L+1_{in}}$ from $\rho_x^{1_{in}}$, we need to repeat Steps 3a to 3d a total of L times. Since we do not do the residual learning in the output layer, then we need do Step 3a to 3c to the output layer. We assume $m_{l-1} \leq m_l$ for $l = 1, 2, \cdots, L$, so the total number of qubits needed in ResQNN is $m_{L-1} + m_L$. We also need $\sum_{k=1}^{L} m_k$ perceptrons from Steps 3a to 3d.

Step4 Do the *c-SWAP* trick used in [33] on ρ_x^{out} and $|\phi_x^{out}\rangle$:

Step4a Suppose $m = 1$. Do the swap test in Fig.1 for the input pure state $|\phi\rangle$ with a mixed state ρ and get the final state δ_{SWAP} before measurement :

$$\begin{aligned}\delta_{SWAP} =& \frac{1}{4}(|0\rangle + |1\rangle)[(\langle 0| + \langle 1|) \otimes (|\phi\rangle\langle\phi| \otimes \rho) + (\langle 0| - \langle 1|) \\ & \otimes (|\phi\rangle\langle\phi| \otimes \rho) SWAP] + \frac{1}{4}(|0\rangle - |1\rangle)[(\langle 0| + \langle 1|) \otimes \\ & (SWAP(|\phi\rangle\langle\phi| \otimes \rho)) + (\langle 0| - \langle 1|) \otimes (SWAP(|\phi\rangle\langle\phi| \\ & \otimes \rho) SWAP)].\end{aligned}$$

Here $SWAP$ represents the swap test in Fig.1.

Step4b Measuring the first control qubit in computational basis, we get 0 and 1 with probability p_0 and p_1:

$$p_0 = \mathrm{tr}(|0\rangle\langle 0| \otimes Id(1) \otimes Id(1) \times \delta_{SWAP}),$$

$$p_1 = \mathrm{tr}(|1\rangle\langle 1| \otimes Id(1) \otimes Id(1) \times \delta_{SWAP}).$$

After tedious computation, we have

$$p_0 = \frac{1}{2} + \frac{1}{2}\mathrm{tr}(SWAP|\phi\rangle\langle\phi| \otimes \rho),$$

$$p_1 = \frac{1}{2} - \frac{1}{2}\mathrm{tr}(SWAP|\phi\rangle\langle\phi| \otimes \rho).$$

Since two-qubit swap gate can be represented as $\sum_{j,k=1}^{2} |j,k\rangle\langle k,j|$, then we can further work out

$$\begin{aligned}p_0 =& \frac{1}{2} + \frac{1}{2}\sum_{j,k=1}^{2} \langle\phi|j\rangle \langle j|\rho|k\rangle \langle k|\phi\rangle \\ =& \frac{1}{2} + \frac{1}{2}F(|\phi\rangle, \rho),\end{aligned}$$

$$\begin{aligned}p_1 =& \frac{1}{2} - \frac{1}{2}\sum_{j,k=1}^{2} \langle\phi|j\rangle \langle j|\rho|k\rangle \langle k|\phi\rangle \\ =& \frac{1}{2} - \frac{1}{2}F(|\phi\rangle, \rho).\end{aligned}$$

In general, we repeat this measurement N times to reduce the fluctuations in the binomial probability distribution. Denote $Num(0)$ be the number of times getting 0, and $Num(1)$ be the number of times getting 1, then $\frac{Num(0)}{N} = p_0 + \gamma p_0$, $\frac{Num(1)}{N} = p_1 + \gamma p_1$, where $\gamma p_i = \sqrt{\frac{p_i(1-p_i)}{N}}$ is the fluctuation.

Step4c When we require m ($m > 1$) qubits in this system, we can replace the two-qubit swap gate into an 2m-qubit swap gate $SWAP(m)$ in the quantum circuit:

$$\begin{aligned}SWAP(m) = & \sum_{j_1,\cdots,j_m;k_1,\cdots,k_m} |j_1, \cdots, j_m; k_1, \cdots, k_m\rangle \\ & \times \langle k_1, \cdots, k_m; j_1, \cdots, j_m|.\end{aligned}$$

Step5 Repeat the former four steps a total of M times for a fixed x to estimate the fidelity of $|\phi_x^{out}\rangle$ and ρ_x^{out} for better accuracy.

Therefore, we can estimate the cost function from the analysis above, which is shown in Fig.6. For N training pairs $(|\phi_x^{in}\rangle, |\phi_x^{out}\rangle)$ with $x = 1, 2, \cdots, N$, employing this quantum circuit N times and computing the expectation

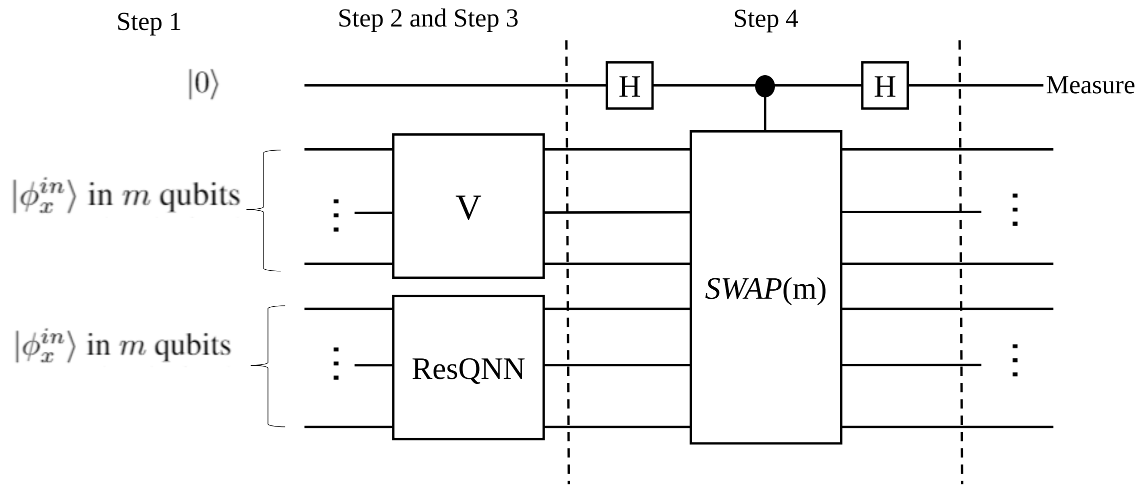

Step 1
Step 2 and Step 3
Step 4
$|0\rangle$
H
H
Measure
$|\phi_x^{in}\rangle$ in m qubits
$|\phi_x^{in}\rangle$ in m qubits
V
ResQNN
SWAP(m)

FIG. 6: **Quantum circuit for calculating cost function.**

value of the fidelity of $|\phi_x^{out}\rangle$ and ρ_x^{out} over x, we get the cost function $C = \frac{1}{N}\sum_{x=1}^{N}\langle\phi_x^{out}|\rho_x^{out}|\phi_x^{out}\rangle$. Compared with Ref.[33], the total number of gates and perceptrons used is $N \times M(\sum_{k=1}^{L} m_k + 3)$ and the number of qubits is no more than $2 \times m_L + m + 1$, which are both no increasing. However, we get better performance in next experiments.

After computing the cost function, we should further work out the derivative of cost function $\frac{dC}{ds}$ on a quantum computer. The method for this part is based on the supplementary of [33], but we rephrase it in an easier and more clear way. We focus on the parameter matrices $K(s)$ with $U(s+\epsilon) = e^{i\epsilon K(s)}U(s)$. We can parametrise $K(s)$ in the form of

$$K(s) = \sum K_{\alpha_1,\cdots,\alpha_{m_{l-1}},\beta}(s)(\sigma^{\alpha_1}\otimes\cdots\otimes\sigma^{\alpha_{m_{l-1}}}\otimes\sigma^{\beta}),$$

with α_i being the qubit in layer $l-1$ and β corresponding to the current qubit in layer l. Therefore we compute $\frac{dC}{dy}$ instead of $\frac{dC}{ds}$ where y is the vector of all the parameters. For example, a single three-qubit perceptron U with $K = \sum K_{\alpha_1,\alpha_2,\beta}(\sigma^{\alpha_1}\otimes\sigma^{\alpha_2}\otimes\sigma^{\beta})$ can produce the vector with 64 entries

$$y = \begin{bmatrix} K_{000} \\ K_{001} \\ K_{002} \\ K_{003} \\ K_{010} \\ \vdots \end{bmatrix}.$$

Now we are aiming at working out

$$\frac{dC}{dy} = \begin{bmatrix} \frac{\partial C}{\partial y_1} \\ \frac{\partial C}{\partial y_2} \\ \vdots \\ \frac{\partial C}{\partial y_\mu} \\ \vdots \\ \frac{\partial C}{\partial y_Q} \end{bmatrix}, \qquad (1)$$

with $\mu = 1, 2, \cdots, Q$ and $Q = num(perc) \times 64$. Here $num(perc)$ is the number of perceptrons. Since $\frac{\partial C}{\partial y_\mu} \approx \frac{C(y+\epsilon_\mu)-C(y)}{\epsilon}$ with

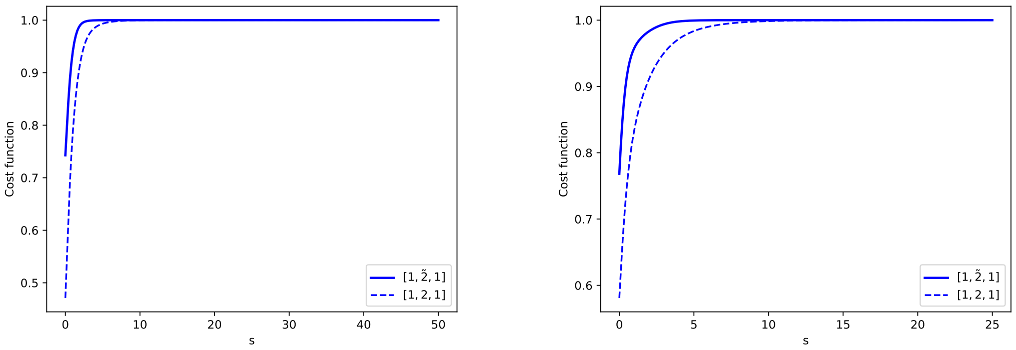
$$\epsilon_\mu = \begin{bmatrix} 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix},$$

where ϵ is the μ-th entry. So supposing $C(y)$ is known, we can get $\frac{dC}{dy}$ directly based on Eq.(1).

Then applying the gradient ascent step, for a small enough positive real number γ, we have $y_{new} = y_{old} + \gamma\frac{dC}{dy}$. Taylor theorem and (1) further lead us to

$$\begin{aligned} C(y_{new}) =& C(y_{old} + \gamma\frac{dC}{dy}) \\ =& C(y_{old}) + \sum_{\mu=1}^{Q}\gamma\left(\frac{\partial C}{\partial y_\mu}(y_{old})\right)^2 + o(\gamma^2). \end{aligned}$$

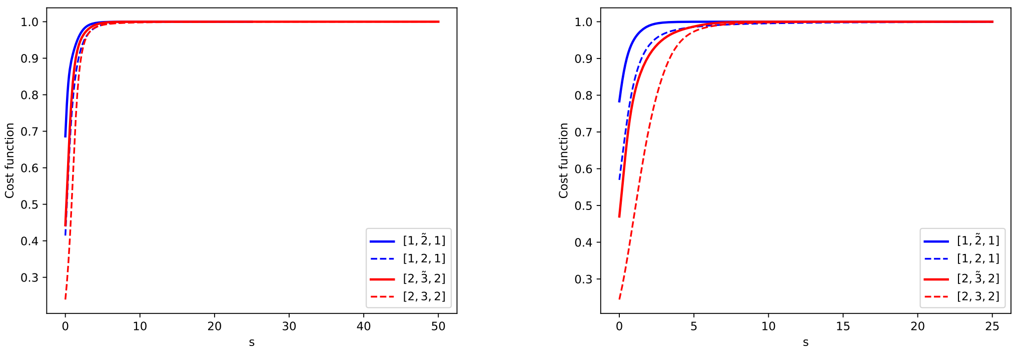
Since γ is small enough, the cost function $C(y_{new})$ can always be bigger than $C(y_{old})$ through updating parameters in the way above.



(a)500 training rounds

(b)250 training rounds

FIG. 7: **Numerical results of $[1, \tilde{2}, 1]$ and $[1, 2, 1]$ with 10 training pairs for different training rounds.** Here $\lambda = 1$ and $\epsilon = 0.1$.



(a)500 training rounds

(b)250 training rounds

FIG. 8: **Numerical results of $[2, \tilde{3}, 2]$, $[2, 3, 2]$, $[1, \tilde{2}, 1]$ and $[1, 2, 1]$ with 10 training pairs for different training rounds.** Here $\lambda = 1$ and $\epsilon = 0.1$.

V. EXPERIMENTS

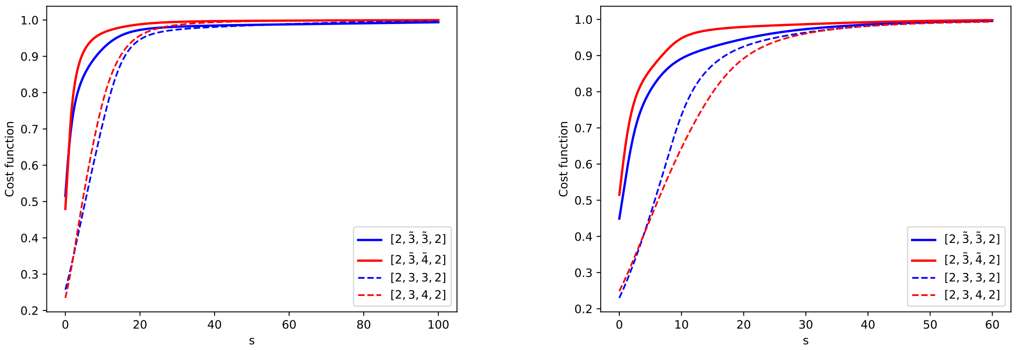
In this section, we conduct comprehensive experiments to evaluate the proposed ResQNN. Firstly, we start from the elementary tests to prove the effectiveness of our basic quantum computing module. Then, we further explore its ability by extending it with more quantum neural layers. Finally, we generalize the experiments into corrupted training data for testing the robustness of our ResQNN. Here, we compare to the current advanced method in [33] with corresponding settings.

For convenience, we apply a 1-dimensional list of natural numbers to refer to the number of perceptrons in the corresponding layer. Specially, if there is a residual block structure shown in Fig.3 that acts on the hidden layers, we plus a tilde on the top of the natural numbers. For example, a 1-2-1 quantum neural network in [33] can be denoted as $[1, 2, 1]$, and a 1-2-1 quantum neural network with our residual block structure in this paper will be written as $[1, \tilde{2}, 1]$.

A. Elementary tests

We consider the ResQNN $[1, \tilde{2}, 1]$ and $[2, \tilde{3}, 2]$ with $\lambda = 1$ and $\epsilon = 0.1$ for the elementary tests. In this two simulations, the number of training pairs is randomly set to be 10. As shown in Fig.7(a), when the training rounds is 500, roughly speaking, the solid line is higher than the dashed line and both lines converge to 1 as the training rounds increasing. To get more details about the improvement, we shrink the training rounds to 250 in Fig.7(b), we find that the solid line exceeds the cost of 0.95 at training step 15 while the dashed line at training step 21. The solid line has faster convergence.

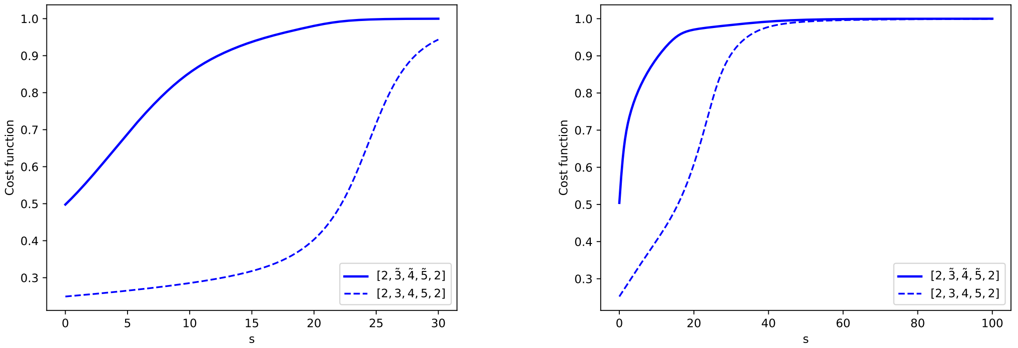
We change ResQNN into $[2, \tilde{3}, 2]$ to explore the difference with $[1, \tilde{2}, 1]$. For convenient comparison, we run them at the same time for different training rounds in Fig.8. We find that the solid lines are still higher than



(a)1000 training rounds

(b)600 training rounds

FIG. 9: **Numerical results of** $[2,\tilde{3},\tilde{3},2]$**,** $[2,\tilde{3},\tilde{4},2]$**,** $[2,3,3,2]$ **and** $[2,3,4,2]$ **with 5 training pairs for different training rounds.** Here $\lambda = 4$ and $\epsilon = 0.1$.



(a)100 training pairs, 300 training rounds

(b)5 training pairs, 1000 training rounds

FIG. 10: **Numerical results of** $[2,\tilde{3},\tilde{4},\tilde{5},2]$ **and** $[2,3,4,5,2]$ **for small (big) training pairs and big (small) training rounds.** Here $\lambda = 4$ and $\epsilon = 0.1$.

the dashed ones in the same color and the solid lines has faster convergence. It is also interesting to observe that the cost of blue lines is bigger than the one of red lines in the same line type, which is more intuitive in Fig.8(b). Moreover, the area between the blue solid line and the blue dashed one for $s \in [0, 25]$ in Fig.8(b) is smaller than the area between the red solid and red dashed lines, which indicates the improvement of our ResQNN may be more obvious for the network with more perceptrons.

B. Big networks

In this subsection, we consider ResQNN with deeper structure to test the advantage of deep residual learning. We firstly select ResQNN $[2,\tilde{3},\tilde{3},2]$ and $[2,\tilde{3},\tilde{4},2]$ for 5 training pairs with $\lambda = 4$, $\epsilon = 0.1$. We present the simulation results in Fig.9 with different training rounds.

In Fig.9(a), we train them for 1000 training rounds. As for the blue and red lines in the same line-type, the values of cost function for $[2,\tilde{3},\tilde{4},2]$ is bigger than the ones for $[2,\tilde{3},\tilde{3},2]$ due to the increase of the number of quantum perceptrons. Compared the solid and dashed lines in the same color, the advantage of our ResQNN is apparent with bigger value and faster convergence of the cost function. When the training rounds come to 600 in Fig.9(b), the improvement of our ResQNN is more intuitive. It should be mentioned that there is an unstable result in Fig.9(b) that the blue dashed line is higher than the red dashed line for some region of s. This result is caused by the randomness of the generated training pairs for quantum neural networks.

We then try ResQNN $[2,\tilde{3},\tilde{4},\tilde{5},2]$ with more complicated quantum neural network structure. We test its performance with small (big) training pairs and big (small) training rounds, which is shown in Fig.10. When the number of training pairs is set to be 100 with training rounds 300 in Fig.10(a), we find that the solid line is increasing with decreasing slope and finally it converges to 1 as the training rounds increasing to 300. However the

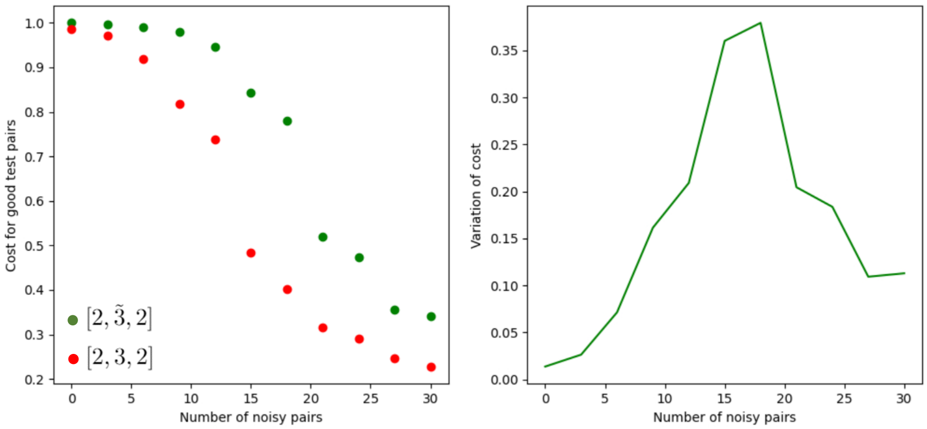


FIG. 11: **Behaviors of $[2,\tilde{3},2]$ and $[2,3,2]$ to noisy training data for 50 training rounds and 30 training pairs.** Here $\lambda = 1$ and $\epsilon = 0.1$. The step-size between two adjacent dots is 3. We also plot the variance of cost between the green dots and red dots on the right and the variance is always positive.

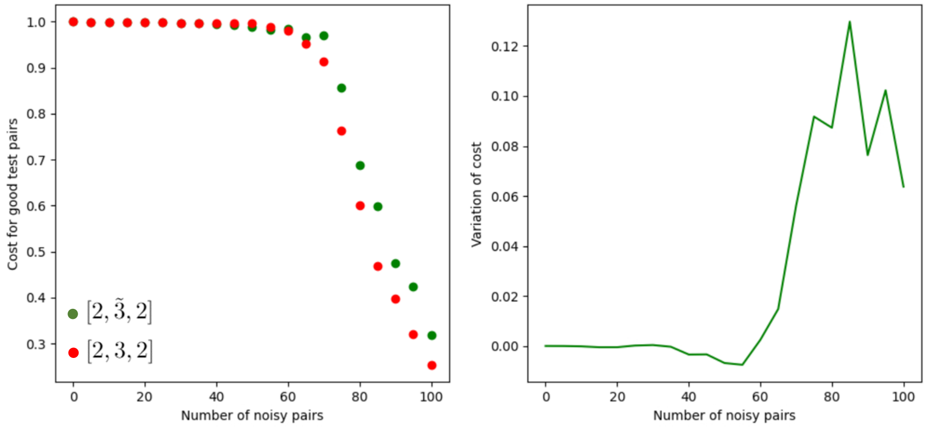


FIG. 12: **Behaviors of $[2,\tilde{3},2]$ and $[2,3,2]$ to noisy training data for 300 training rounds and 100 training pairs.** Here $\lambda = 1$ and $\epsilon = 0.1$. The step-size between two adjacent dots is 5. We also plot the variance of cost between the green dots and red dots on the right.The recorded unstable points are (35,-0.0012), (40,-0.0046), (45,-0.0042), (50,-0.0076), (55,-0.0076), which can be found in both two sub-figures.

slope of the bottom dashed line is increasing and does not converge at all during 300 training rounds. In Fig.10(b), we set the small training pairs to be 5 and big training rounds to be 1000, which is in the same condition of Fig.9(a). So comparing the performance of these three ResQNN $[2,\tilde{3},\tilde{4},2]$, $[2,\tilde{3},\tilde{3},2]$ and $[2,\tilde{3},\tilde{4},\tilde{5},2]$ in 1000 training rounds, the solid lines are always higher than their corresponding dashed lines and the area between the solid and dashed lines with $s \in [0, 100]$ for $[2,\tilde{3},\tilde{4},\tilde{5},2]$ is the biggest. This illustrates that our ResQNN may have better improvement for deeper quantum neural networks.

C. Generalization: the robustness to noisy data

In this subsection, we study the task of examining the robustness of ResQNN to noisy quantum data. Here the noisy quantum data means that the training pairs are in the form of $(|\phi_x^{in}\rangle, |\theta_x^{out}\rangle)$, where the desired output $|\theta_x^{out}\rangle$ has no direct relation to $|\phi_x^{in}\rangle$. The good training pairs introduced before are in the form of $(|\phi_x^{in}\rangle, V|\phi_x^{in}\rangle)$ with an unknown unitary V. Like the rule in [33], we firstly generate N good training pairs and then destroy n of them by replacing them with noisy training pairs. Every time the replaced subset is chosen randomly. The cost function is assessed for all good test pairs.

We choose ResQNN $[2,\tilde{3},2]$ with $\lambda = 1$ and $\epsilon = 0.1$ as an example and their behaviours under approximate depolarizing noise are presented in Fig.11 and Fig.12. In the left sub-figures of Fig.11 and Fig.12, the green dots are the results of our ResQNN $[2, \tilde{3}, 2]$ and the red ones are the corresponding results in [33]. And in the right sub-figures, we also plot the variation of the cost function between the green dots and the red ones. The x-axis of the left sub-figure represents how many good training pairs are replaced by noisy pairs.

When the number of the training rounds and training pairs are small, such as 50 training rounds and 30 training pairs in Fig.11, the value of cost for good pairs decreases as the number of noisy pairs increase and the variation of the cost value is always positive. This shows the superiority of our ResQNN for noisy training data with small training rounds and small training pairs.

When the number of training rounds and training pairs are big, such as 300 training rounds and 100 training pairs in Fig.12, we find if the number of noisy pairs are small, such as less than 35 in Fig.12, our ResQNN and the quantum neural network in [33] both have strong robustness to noisy quantum data. There is an unstable region $[35, 55]$ for the number of noisy pairs that the variation is negative, which can be seen as a comparable results. Nevertheless, as the number of noisy pairs exceeding 60, the value of cost is always bigger than the ones in [33] and the final value comes more than 0.3. So our ResQNN has significant improvement about the robustness to noisy training data, especially for the higher proportion of noisy training pairs.

Up to now, we have gone through the experiments for ResQNN with or without noisy and obtained the improvement of our ResQNN compared with the QNN in [33]. From the perspective of theory, in fact, the residual block structure gives a new input state $\rho^{l+1_{in}}$ instead of $\rho^{l_{out}}$. Since $\rho^{l+1_{in}} = \rho^{l_{out}} + \big(\rho^{l_{in}} \otimes Id(m_l - m_{l-1})\big)$ with $l = 1, 2, \cdots, L$, then for each training pair $(|\phi_x^{in}\rangle, |\phi_x^{out}\rangle)$ with $x = 1, 2, \cdots, N$, the final output state $\rho_x^{L+1_{out}}$ of ResQNN can be abstractly described as

$$\rho_x^{L+1_{out}} = \hat{\rho_x}^{L+1_{out}} + \tau_x,$$

where $\hat{\rho_x}^{L+1_{out}}$ is the final output of QNN in [33] and τ_x is a Hermitian matrix. Based on the fact that any Hermitian matrix can be diagonalized by an unitary matrix, there exists an unitary matrix U such that

$$\tau_x = U \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \lambda_m \end{bmatrix} U^\dagger,$$

with $\lambda_1, \cdots, \lambda_m$ being the non-negative eigenvalues of τ_x. So $\langle \phi_x^{out} | \tau_x | \phi_x^{out} \rangle \geqslant 0$, and naturally we have

$$\langle \phi_x^{out} | \rho_x^{L+1_{out}} | \phi_x^{out} \rangle \geqslant \langle \phi_x^{out} | \hat{\rho_x}^{L+1_{out}} | \phi_x^{out} \rangle,$$

which indicates that the values of cost function for ResQNN is bigger than the ones in [33] with both clean data and noisy data. This theoretical analysis gives good agreement with the results in experiments.

VI. DISCUSSION

We have developed a quantum neural network with deep residual learning. Our work is useful for NISQ devices. As illustrated in [33], their feedforward quantum neural network reduces the number of coherent qubits which are needed to store the intermediate states. What we have to pay for is to evaluate the network many times to compute the derivative of the cost function. But it is worthy it. Since many NISQ devices can repeat the executions of quantum circuit easily and quickly while increasing coherent qubits might be a challenging problem in the near term. In our paper, we keep the previous feedforward quantum neural network and add a residual block structure for the hidden layers. As a result, the advantages of the former is reserved and the performance of experiments is substantially improved. So our ResQNN has the more efficient ability to learn an unknown unitary and stronger robustness for corrupted training data, which could be an important reference for the development of NISQ architectures. For future work, we wish to design a data encoding rule for classical data so that this efficient quantum neural network can deal with meaningful problems in classical machine learning.

Acknowledgments This work is supported by the Guangdong Basic and Applied Basic Research Foundation under Grant No.2020B1515310016, Key Research and Development Project of Guangdong Province under Grant No.2020B0303300001, the Academy of Finland for ICT 2023 project (grant 328115) and project MiGA (grant 316765) and Infotech Oulu. This work is also supported by the Chinese Scholarship Council.

[1] M. A. Nielsen, *Neural networks and deep learning.* Determination press San Francisco, CA, 2015, vol. 2018.

[2] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[3] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, “Theano: A cpu and gpu math compiler in python,” in *Proc. 9th Python in Science Conf*, vol. 1, 2010, pp. 3–10.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the 22nd ACM international conference*

on Multimedia, 2014, pp. 675–678.

[5] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Autograd: Effortless gradients in numpy," in *ICML 2015 AutoML Workshop*, vol. 238, 2015, p. 5.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," *In NISP Workshop*, 2017.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[8] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[9] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[10] C. M. Bishop *et al.*, *Neural networks for pattern recognition.* Oxford university press, 1995.

[11] E. Nishani and B. Çiço, "Computer vision approaches based on deep learning and neural networks: Deep neural networks for video analysis of human pose estimation," in *2017 6th Mediterranean Conference on Embedded Computing (MECO).* IEEE, 2017, pp. 1–4.

[12] F. Amato, A. López, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel, "Artificial neural networks in medical diagnosis," 2013.

[13] T. M. Mitchell and S. B. Thrun, "Explanation-based neural network learning for robot control," in *Advances in neural information processing systems*, 1993, pp. 287–294.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[16] W. Peng, X. Hong, H. Chen, and G. Zhao, "Learning graph convolutional network for skeleton-based human action recognition by neural searching," *AAAI*, 2020.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision.* Springer, 2016, pp. 630–645.

[18] 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.

[19] T. F. Rønnow, Z. Wang, J. Job, S. Boixo, S. V. Isakov, D. Wecker, J. M. Martinis, D. A. Lidar, and M. Troyer, "Defining and detecting quantum speedup," *science*, vol. 345, no. 6195, pp. 420–424, 2014.

[20] M. Schuld, I. Sinayskiy, and F. Petruccione, "The quest for a quantum neural network," *Quantum Information Processing*, vol. 13, no. 11, pp. 2567–2586, 2014.

[21] V. Dunjko and H. J. Briegel, "Machine learning & artificial intelligence in the quantum domain: a review of recent progress," *Reports on Progress in Physics*, vol. 81, no. 7, p. 074001, 2018.

[22] M. Sasaki and A. Carlini, "Quantum learning and universal quantum matching machine," *Physical Review A*, vol. 66, no. 2, p. 022303, 2002.

[23] V. Dunjko, J. M. Taylor, and H. J. Briegel, "Quantum-enhanced machine learning," *Physical review letters*, vol. 117, no. 13, p. 130501, 2016.

[24] U. Alvarez-Rodriguez, L. Lamata, P. Escandell-Montero, J. D. Martín-Guerrero, and E. Solano, "Supervised quantum learning without measurements," *Scientific reports*, vol. 7, no. 1, pp. 1–9, 2017.

[25] G. Verdon, J. Pye, and M. Broughton, "A universal training algorithm for quantum deep learning," *arXiv preprint arXiv:1806.09729*, 2018.

[26] G. Sentís, A. Monràs, R. Muñoz-Tapia, J. Calsamiglia, and E. Bagan, "Unsupervised classification of quantum data," *Physical Review X*, vol. 9, no. 4, p. 041029, 2019.

[27] J. Zhao, Y.-H. Zhang, C.-P. Shao, Y.-C. Wu, G.-C. Guo, and G.-P. Guo, "Building quantum neural networks based on a swap test," *Physical Review A*, vol. 100, no. 1, p. 012334, 2019.

[28] P. Li and B. Wang, "Quantum neural networks model based on swap test and phase estimation," *Neural Networks*, vol. 130, pp. 152–164, 2020.

[29] A. Bisio, G. Chiribella, G. M. D'Ariano, S. Facchini, and P. Perinotti, "Optimal quantum learning of a unitary transformation," *Physical Review A*, vol. 81, no. 3, p. 032324, 2010.

[30] M. Sedlák, A. Bisio, and M. Ziman, "Optimal probabilistic storage and retrieval of unitary channels," *Physical Review Letters*, vol. 122, no. 17, p. 170502, 2019.

[31] G. Purushothaman and N. B. Karayiannis, "Quantum neural networks (qnns): inherently fuzzy feedforward neural networks," *IEEE Transactions on neural networks*, vol. 8, no. 3, pp. 679–693, 1997.

[32] A. A. Ezhov and D. Ventura, "Quantum neural networks," in *Future directions for intelligent systems and information sciences.* Springer, 2000, pp. 213–235.

[33] K. Beer, D. Bondarenko, T. Farrelly, T. J. Osborne, R. Salzmann, D. Scheiermann, and R. Wolf, "Training deep quantum neural networks," *Nature communications*, vol. 11, no. 1, pp. 1–6, 2020.

[34] M. M. Wolf, "Mathematical foundations of supervised learning," 2018.

[35] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[36] W. Finigan, M. Cubeddu, T. Lively, J. Flick, and P. Narang, "Qubit allocation for noisy intermediate-scale quantum computers," *arXiv preprint arXiv:1810.08291*, 2018.

[37] A. Ash-Saki, M. Alam, and S. Ghosh, "Qure: Qubit reallocation in noisy intermediate-scale quantum computers," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[38] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1001–1014.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image

database," in *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 2009, pp. 248–255.

[41] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[42] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, pp. 6571–6583, 2018.

[43] C. Dong, L. Liu, Z. Li, and J. Shang, "Towards adaptive residual network training: A neural-ode perspective," in *International Conference on Machine Learning.* PMLR, 2020, pp. 2616–2626.

[44] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," *arXiv preprint arXiv:1801.07455*, 2018.

[45] W. Peng, J. Shi, Z. Xia, and G. Zhao, "Mix dimension in poincar\'{e} geometry for 3d skeleton-based action recognition," *ACM Multimedia*, 2020.

[46] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[47] H. Buhrman, R. Cleve, J. Watrous, and R. De Wolf, "Quantum fingerprinting," *Physical Review Letters*, vol. 87, no. 16, p. 167902, 2001.

Appendices

This is an example for the training algorithm of ResQNN in Subsection 4.2. Here we present the training algorithm of a three-layer ResQNN in Fig.5, which can be denoted as $[2, \tilde{3}, 2]$.

I. Initialize:

I.1 Set step $s = 0$.

I.2 Choose all unitary $U_j^1(0)$ and $U_q^1(0)$ randomly, $j = 1, 2, 3$, $q = 1, 2$.

II. For each element $(|\phi_x^{in}\rangle, |\phi_x^{out}\rangle)$ from the set of training data, do the following steps:

II.1 Feedforward:

II.1a Tensor the input state $|\phi_x^{in}\rangle$ to the inintial state of the hidden layer,

$$|\Phi_x^{in}\rangle = |\phi_x^{in}\rangle \otimes |000\rangle_{hid}.$$

II.1b Applying the layer unitary $U^1(s) = U_3^1 U_2^1 U_1^1(s)$ to $|\Phi_x^{in}\rangle$ leads to

$$U_{apply}^1(s) = U^1(s)\left(|\Phi_x^{in}\rangle\langle\Phi_x^{in}|\right){U^1}^\dagger(s).$$

II.1c Trace out the input layer and obtain the output state of the hidden layer,

$$\rho_x^{1_{out}}(s) = \mathrm{tr}_{in}(U_{apply}^1(s)).$$

II.2 Residual learning:

II.2a Apply the residual block structure to $\rho_x^{1_{out}}(s)$ to get the new input state of the output layer,

$$\rho_x^{2_{in}}(s) = \rho_x^{1_{out}}(s) + \left(\rho_x^{1_{in}}(s) \otimes Id(1)\right).$$

II.2b Store $\rho_x^{2_{in}}(s)$.

II.3 Repeat Step II.1(feedforward) for $\rho_x^{2_{in}}(s)$, and get the final output of ResQNN $[2, \tilde{3}, 2]$ with $U^2(s) = U_2^1 U_2^2(s)$:

$$\rho_x^{2_{out}}(s) = \mathrm{tr}_{hid}(U^2(s)\left(\rho_x^{2_{in}}(s) \otimes |00\rangle_{out}\langle 00|\right){U^2}^\dagger(s)).$$

III. Update the parameters:

III.1 Compute the cost function:

$$C(s) = \frac{1}{N}\sum_{x=1}^{N}\langle\phi_x^{out}|\rho_x^{2_{out}}(s)|\phi_x^{out}\rangle.$$

III.2 Calculate the parameter matrices $K_j^1(s)$ and $K_q^2(s)$ for $j = 1, 2, 3$ and $q = 1, 2$, which will be illustrated later.

III.3 Update each perceptron unitary via

$$U_j^1(s+\epsilon) = e^{i\epsilon K_j^1(s)}U_j^1(s).$$

$$U_q^2(s+\epsilon) = e^{i\epsilon K_q^2(s)}U_q^2(s).$$

III.4 Update $s = s + \epsilon$.

IV. Repeat Step II and Step III until reaching the maximum of the cost function.

In the following, we will derive a formula for $K_j^1(s)$ to update the perceptron unitaries $U_j^1(s)$ with $l = 1, 2$. We assume the unitaries can always act on its current layer, such as $U_2^1(s)$ is actually $U_2^1(s) \otimes Id(2)$ in ResQNN $[2, \tilde{3}, 2]$. The method is similar but actually different from [33] due to the part of deep residual learning.

We begin with a derivative function of the cost function: $\frac{dC}{ds} = \lim\limits_{\epsilon\to 0}\frac{C(s+\epsilon)-c(s)}{\epsilon}$. This derivative function leads us to find an analytical expression of $C(s+\epsilon)$. According to the definition of $C(s+\epsilon)$, we focus on the output state of the updated unitary

$$U^1(s+\epsilon) = e^{i\epsilon K_3^1(s)}U_3^1(s)e^{i\epsilon K_2^1(s)}U_2^1(s)e^{i\epsilon K_1^1(s)}U_1^1(s),$$

$$U^2(s+\epsilon) = e^{i\epsilon K_2^2(s)}U_2^2(s)e^{i\epsilon K_1^2(s)}U_1^2(s).$$

Firstly, we consider the output state of the updated unitary $U^1(s+\epsilon)$ in the hidden layer. For convenience, we omit to write the parameter s in $K_j^1(s)$ and $U_j^1(s)$ in the following with $l = 1, 2$. Then

$$\begin{aligned}
\rho_x^{1_{out}}(s+\epsilon) =& \operatorname{tr}_{in}(e^{i\epsilon K_3^1}U_3^1 e^{i\epsilon K_2^1}U_2^1 e^{i\epsilon K_1^1}U_1^1|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \\
& \times {U_1^1}^\dagger e^{-i\epsilon K_1^1}{U_2^1}^\dagger e^{-i\epsilon K_2^1}{U_3^1}^\dagger e^{-i\epsilon K_3^1}) \\
=&\rho_x^{1_{out}}(s) + i\epsilon \operatorname{tr}_{in}(U_3^1U_2^1K_1^1U_1^1|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \\
& \times {U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger + U_3^1K_2^1U_2^1U_1^1|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \\
& \times {U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger + K_3^1U_3^1U_2^1U_1^1|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \\
& \times {U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger - U_3^1U_2^1U_1^1|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \\
& \times {U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger K_3^1 - U_3^1U_2^1U_1^1|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \\
& {U_1^1}^\dagger{U_2^1}^\dagger K_2^1{U_3^1}^\dagger - U_3^1U_2^1U_1^1|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \\
& \times {U_1^1}^\dagger K_1^1{U_2^1}^\dagger{U_3^1}^\dagger) + o(\epsilon^2) \\
=&\rho_x^{1_{out}}(s) + R(\epsilon) + o(\epsilon^2),
\end{aligned}$$

where the second inequality is due to the Taylor's Formula of the exponential function. We denote the second term in the second inequality as $R(\epsilon)$. Then using the residual block structure to $\rho_x^{1_{out}}(s+\epsilon)$, we obtain the new input state of the output layer:

$$\begin{aligned}
\rho_x^{2_{in}}(s+\epsilon) =&\rho_x^{1_{out}}(s+\epsilon) + \big(|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \otimes Id(1)\big) \\
=&\rho_x^{1_{out}}(s) + \big(|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \otimes Id(1)\big) + R(\epsilon) \\
=&\rho_x^{2_{in}}(s) + R(\epsilon).
\end{aligned}$$

Based on the new input state for the output layer, we then go on computing the updated final output state of ResQNN $[2, \tilde{3}, 2]$ in terms of $U^2(s+\epsilon)$:

$$\begin{aligned}
\rho_x^{2_{out}}(s+\epsilon) =& \operatorname{tr}_{hid}(e^{i\epsilon K_2^2}U_2^2 e^{i\epsilon K_1^2}U_1^2\left(\rho_x^{2_{in}}(s+\epsilon) \otimes |00\rangle_{out}\langle 00|\right) \\
& {U_1^2}^\dagger e^{-i\epsilon K_1^2}{U_2^2}^\dagger e^{-i\epsilon K_2^2}) \\
=& \operatorname{tr}_{hid}(e^{i\epsilon K_2^2}U_2^2 e^{i\epsilon K_1^2}U_1^2\left(\rho_x^{2_{in}}(s) \otimes |00\rangle_{out}\langle 00|\right) \\
& {U_1^2}^\dagger e^{-i\epsilon K_1^2}{U_2^2}^\dagger e^{-i\epsilon K_2^2}) \\
& + \operatorname{tr}_{hid}(e^{i\epsilon K_2^2}U_2^2 e^{i\epsilon K_1^2}U_1^2\left(R(\epsilon) \otimes |00\rangle_{out}\langle 00|\right) \\
& {U_1^2}^\dagger e^{-i\epsilon K_1^2}{U_2^2}^\dagger e^{-i\epsilon K_2^2}) \\
=&\rho_x^{2_{out}}(s) + A_1 + A_2 + o(\epsilon),
\end{aligned}$$

in which

$$\begin{aligned}
A_1 =&i\epsilon \operatorname{tr}_{hid}(U_2^2K_1^2U_1^2(\rho_x^{2_{in}}(s) \otimes |00\rangle_{out}\langle 00|){U_1^2}^\dagger{U_2^2}^\dagger \\
& + K_2^2U_2^2U_1^2(\rho_x^{2_{in}}(s) \otimes |00\rangle_{out}\langle 00|){U_1^2}^\dagger{U_2^2}^\dagger \\
& - U_2^2U_1^2(\rho_x^{2_{in}}(s) \otimes |00\rangle_{out}\langle 00|){U_1^2}^\dagger{U_2^2}^\dagger K_2^2 \\
& - U_2^2U_1^2(\rho_x^{2_{in}}(s) \otimes |00\rangle_{out}\langle 00|){U_1^2}^\dagger K_1^2{U_2^2}^\dagger),
\end{aligned}$$

and

$$\begin{aligned}
A_2 =&i\epsilon \operatorname{tr}_{in,hid}(U_2^2U_1^2U_3^1U_2^1K_1^1U_1^1(|\Phi_x^{in}\rangle\langle\Phi_x^{in}|\otimes \\
& |00000\rangle_{hid,out}\langle 00000|){U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger{U_1^2}^\dagger{U_2^2}^\dagger \\
& + U_2^2U_1^2U_3^1K_2^1U_2^1U_1^1(|\Phi_x^{in}\rangle\langle\Phi_x^{in}|\otimes \\
& |00000\rangle_{hid,out}\langle 00000|){U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger{U_1^2}^\dagger{U_2^2}^\dagger \\
& + U_2^2U_1^2K_3^1U_3^1U_2^1U_1^1(|\Phi_x^{in}\rangle\langle\Phi_x^{in}|\otimes \\
& |00000\rangle_{hid,out}\langle 00000|){U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger{U_1^2}^\dagger{U_2^2}^\dagger \\
& - U_2^2U_1^2U_3^1U_2^1U_1^1(|\Phi_x^{in}\rangle\langle\Phi_x^{in}|\otimes \\
& |00000\rangle_{hid,out}\langle 00000|){U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger K_3^1{U_1^2}^\dagger{U_2^2}^\dagger) \\
& - U_2^2U_1^2U_3^1U_2^1U_1^1(|\Phi_x^{in}\rangle\langle\Phi_x^{in}|\otimes \\
& |00000\rangle_{hid,out}\langle 00000|){U_1^1}^\dagger{U_2^1}^\dagger K_2^1{U_3^1}^\dagger{U_1^2}^\dagger{U_2^2}^\dagger) \\
& - U_2^2U_1^2U_3^1U_2^1U_1^1(|\Phi_x^{in}\rangle\langle\Phi_x^{in}|\otimes \\
& |00000\rangle_{hid,out}\langle 00000|){U_1^1}^\dagger K_1^1{U_2^1}^\dagger{U_3^1}^\dagger{U_1^2}^\dagger{U_2^2}^\dagger).
\end{aligned}$$

So according to the properties of the trace operator in quantum information theory, the derivative function of the cost function can be calculated as

$$\begin{aligned}
\frac{dC}{ds} =& \lim_{\epsilon\to 0}\frac{C(s+\epsilon) - c(s)}{\epsilon} \\
=& \lim_{\epsilon\to 0}\frac{\frac{1}{N}\sum_{x=1}^N \langle\phi_x^{out}|\rho_x^{2_{out}}(s+\epsilon)|\phi_x^{out}\rangle}{\epsilon} \\
& - \lim_{\epsilon\to 0}\frac{\frac{1}{N}\sum_{x=1}^N \langle\phi_x^{out}|\rho_x^{2_{out}}(s)|\phi_x^{out}\rangle}{\epsilon} \\
=&\frac{i}{N}\sum_{x=1}^N A_1 + \frac{i}{N}\sum_{x=1}^N A_2 \\
=&\frac{i}{N}\sum_{x=1}^N \operatorname{tr}(M_1^2K_1^2 + M_2^2K_2^2) + \\
& \frac{i}{N}\sum_{x=1}^N \operatorname{tr}(M_1^1K_1^1 + M_2^1K_2^1 + M_3^1K_3^1),
\end{aligned}$$

where

$$\begin{aligned}
M_1^1 =&[U_1^1((|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \otimes |000\rangle_{hid}\langle 000|){U_1^1}^\dagger, \\
& {U_2^1}^\dagger{U_3^1}^\dagger{U_1^2}^\dagger{U_2^2}^\dagger(Id(2) \otimes |\phi_x^{out}\rangle\langle\phi_x^{out}|)U_2^1U_3^1U_1^2U_2^2],
\end{aligned}$$

$$\begin{aligned}
M_2^1 =&[U_2^1U_1^1((|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \otimes |000\rangle_{hid}\langle 000|){U_1^1}^\dagger{U_2^1}^\dagger, \\
& {U_3^1}^\dagger{U_1^2}^\dagger{U_2^2}^\dagger(Id(2) \otimes |\phi_x^{out}\rangle\langle\phi_x^{out}|)U_3^1U_1^2U_2^2],
\end{aligned}$$

$$\begin{aligned}
M_3^1 =&[U_3^1U_2^1U_1^1((|\Phi_x^{in}\rangle\langle\Phi_x^{in}| \otimes |000\rangle_{hid}\langle 000|){U_1^1}^\dagger{U_2^1}^\dagger{U_3^1}^\dagger, \\
& {U_1^2}^\dagger{U_2^2}^\dagger(Id(2) \otimes |\phi_x^{out}\rangle\langle\phi_x^{out}|)U_1^2U_2^2],
\end{aligned}$$

$$\begin{aligned}
M_1^2 =&[U_1^2((\rho_x^{2_{in}} \otimes |00\rangle_{out}\langle 00|){U_1^2}^\dagger, \\
& {U_2^2}^\dagger(Id(3) \otimes |\phi_x^{out}\rangle\langle\phi_x^{out}|)U_2^2],
\end{aligned}$$

$$
\begin{aligned}
M_2^2 =&[U_2^2U_1^2((\rho_x^{2_{in}} \otimes |00\rangle_{out}\langle 00|)U_1^{2^\dagger}U_2^{2^\dagger},\\
&Id(3)\otimes|\phi_x^{out}\rangle\langle\phi_x^{out}|].
\end{aligned}
$$

Here the mathematical commutator operator is in unitary group, which reads $[a,b]=a\times b-b\times a$ for arbitrary unitary matrix a and b. The ”×“ means the multiplication of matrix. This symbol is omitted above.

Playing the similar trick in [33], we next continue investigating the formula for K_j^1 and K_q^2 with $j=1,2,3$ and $q=1,2$. Since unitary $U_j^1(s)$ in the quantum perceptron works on three qubits, then we can parameter K_j^1 as

$$
K_j^1(s)=\sum_{\alpha_1,\alpha_2,\beta}{K_j^1}_{\alpha_1,\alpha_2,\beta}(s)(\sigma^{\alpha_1}\otimes\sigma^{\alpha_2}\otimes\sigma^{\beta}),
$$

where σ is Pauli matrix in single qubit. α_1,α_2 represents the qubits in the input layer and β represents the current qubit of unitary U_j^1 in the hidden layer. Our goal is to reach the maximum of the cost function, however $\frac{dC}{ds}$ can be regarded as a linear function of variable ϵ with the maximum value infinity. So we make use of a Lagrange multiplier λ which is a real number to find a finite solution. When $j=1$, the analysis above leads us to solve a maximization problem in the following:

$$
\begin{aligned}
&\max_{K_1^1,\alpha_1,\alpha_2,\beta}(\frac{dC}{ds}-\lambda\sum_{\alpha_1,\alpha_2,\beta}{K_1^1}_{\alpha_1,\alpha_2,\beta}{}^2)\\
&=\max_{K_1^1,\alpha_1,\alpha_2,\beta}(\frac{i}{N}\sum_{x=1}^{N}\mathrm{tr}(M_1^2K_1^2+M_2^2K_2^2+M_3^1K_3^1\\
&+M_2^1K_2^1+M_1^1\sum_{\alpha_1,\alpha_2,\beta}{K_1^1}_{\alpha_1,\alpha_2,\beta}(\sigma^{\alpha_1}\otimes\sigma^{\alpha_2}\otimes\sigma^{\beta}))\\
&-\lambda\sum_{\alpha_1,\alpha_2,\beta}{K_1^1}_{\alpha_1,\alpha_2,\beta}{}^2)\\
&=\max_{K_1^1,\alpha_1,\alpha_2,\beta}(\frac{i}{N}\sum_{x=1}^{N}\mathrm{tr}_{\alpha_1,\alpha_2,\beta}(\mathrm{tr}_{rest}(M_1^2K_1^2+M_2^2K_2^2\\
&+M_3^1K_3^1+M_2^1K_2^1)+\mathrm{tr}_{rest}(M_1^1)\sum_{\alpha_1,\alpha_2,\beta}{K_1^1}_{\alpha_1,\alpha_2,\beta}(\sigma^{\alpha_1}\\
&\otimes\sigma^{\alpha_2}\otimes\sigma^{\beta}))-\lambda\sum_{\alpha_1,\alpha_2,\beta}{K_1^1}_{\alpha_1,\alpha_2,\beta}{}^2)
\end{aligned}
$$

Taking the derivative of ${K_j^1}_{\alpha_1,\alpha_2,\beta}$ to be zero yields

$$
{K_1^1}_{\alpha_1,\alpha_2,\beta}=\frac{i}{2\lambda N}\sum_{x=1}^{N}\mathrm{tr}_{\alpha_1,\alpha_2,\beta}\left(\mathrm{tr}_{rest}(M_1^1)(\sigma^{\alpha_1}\otimes\sigma^{\alpha_2}\otimes\sigma^{\beta})\right).
$$

This above equation further leads to the matrix:

$$
\begin{aligned}
K_1^1=&\sum_{\alpha_1,\alpha_2,\beta}{K_1^1}_{\alpha_1,\alpha_2,\beta}\\
=&\frac{i}{2\lambda N}\sum_{x=1}^{N}\sum_{\alpha_1,\alpha_2,\beta}\mathrm{tr}_{\alpha_1,\alpha_2,\beta}\left(\mathrm{tr}_{rest}(M_1^1)(\sigma^{\alpha_1}\otimes\sigma^{\alpha_2}\otimes\sigma^{\beta})\right)\\
&\times(\sigma^{\alpha_1}\otimes\sigma^{\alpha_2}\otimes\sigma^{\beta})\\
=&\frac{4i}{\lambda N}\sum_{x=1}^{N}\mathrm{tr}_{rest}(M_1^1).
\end{aligned}
$$

Analogously, we find out the formulas for K_2^1 and K_3^1: $K_2^1=\frac{4i}{\lambda N}\sum_{x=1}^{N}\mathrm{tr}_{rest}(M_2^1)$; $K_3^1=\frac{4i}{\lambda N}\sum_{x=1}^{N}\mathrm{tr}_{rest}(M_3^1)$.

As $K_q^2(s)$ with $q=1,2$ can be parameterized as $K_q^2(s)=\sum_{\alpha_1,\alpha_2,\alpha_3,\beta}{K_q^2}_{\alpha_1,\alpha_2,\alpha_3,\beta}(s)(\sigma^{\alpha_1}\otimes\sigma^{\alpha_2}\otimes\sigma^{\alpha_3}\otimes\sigma^{\beta})$, in ResQNN $[2,\tilde{3},2]$, in which $\alpha_1,\alpha_2,\alpha_3$ represents the qubits in the hidden layer and β represents the current qubit of unitary U_q^2 in the output layer. After experiencing the similar process for K_j^1, we can have the formula for K_q^2:

$$
K_q^2=\frac{8i}{\lambda N}\sum_{x=1}^{N}\mathrm{tr}_{rest}(M_q^2).
$$

Up to now, we have presented a comprehensive example $[2,\tilde{3},2]$ of the training algorithm, which is complex but skillful.