Hybrid quantum-classical convolutional neural networks

Junhua Liu[1,2], Kwan Hui Lim[1,2], Kristin L. Wood[3], Wei Huang[4], Chu Guo[5*], and He-Liang Huang[6,7,8*]

[1]*Information Systems Technology and Design, Singapore University of Technology and Design, Singapore 487372, Singapore;*
[2]*Forth AI, Singapore 487372, Singapore;*
[3]*College of Engineering, Design and Computing, University of Colorado Denver, Colorado 80208, USA;*
[4]*Guangxi Key Laboratory of Optoelectronic Information Processing, Guilin University of Electronic Technology, Guilin 541004, China;*
[5]*Key Laboratory of Low-Dimensional Quantum Structures and Quantum Control of Ministry of Education, Department of Physics and Synergetic Innovation Center for Quantum Effects and Applications, Hunan Normal University, Changsha 410081, China;*
[6]*Hefei National Laboratory for Physical Sciences at Microscale and Department of Modern Physics, University of Science and Technology of China, Hefei 230026, China;*
[7]*Shanghai Branch, CAS Centre for Excellence and Synergetic Innovation Centre in Quantum Information and Quantum Physics, University of Science and Technology of China, Hefei 201315, China;*
[8]*Henan Key Laboratory of Quantum Information and Cryptography, Zhengzhou 450000, China*

Deep learning has been shown to be able to recognize data patterns better than humans in specific circumstances or contexts. In parallel, quantum computing has demonstrated to be able to output complex wave functions with a few number of gate operations, which could generate distributions that are hard for a classical computer to produce. Here we propose a hybrid quantum-classical convolutional neural network (QCCNN), inspired by convolutional neural networks (CNNs) but adapted to quantum computing to enhance the feature mapping process. QCCNN is friendly to currently noisy intermediate-scale quantum computers, in terms of both number of qubits as well as circuit's depths, while retaining important features of classical CNN, such as nonlinearity and scalability. We also present a framework to automatically compute the gradients of hybrid quantum-classical loss functions which could be directly applied to other hybrid quantum-classical algorithms. We demonstrate the potential of this architecture by applying it to a Tetris dataset, and show that QCCNN can accomplish classification tasks with learning accuracy surpassing that of classical CNN with the same structure.

quantum computing, quantum machine learning, hybrid quantum-classical algorithm, convolutional neural network

PACS number(s): 03.67.-a, 03.67.Ac, 03.67.Lx

1 Introduction

With the rapid progress in quantum computing hardware, we are entering the era of developing quantum software to perform useful computational tasks using the noisy intermediate-scale quantum (NISQ) computers [1, 2]. It has been shown in 2019 that the current 53-qubit quantum computer could already solve random quantum circuit sampling problems more efficiently than the best supercomputers in the world [3]. Theoretically, this success originates from the

*Corresponding authors (Chu Guo, email: guochu604b@gmail.com; He-Liang Huang, email: quanhhl@ustc.edu.cn)

fact that a quantum computer could output wave functions with a polynomial number of quantum gate operations, which could nevertheless generate statistical distributions that are very hard for a classical computer to produce [4-6]. The 53-qubit 20-depth random quantum circuit sampling problem has recently been successfully simulated classically [7], however, simulating larger-scale quantum circuits is still a huge challenge. If a quantum computer could easily produce complex distributions, it is also natural to postulate that it is able to learn patterns from certain data distributions which could be very difficult for classical computers [8]. Quantum machine learning (QML) attempts to utilize this power of quantum computers to achieve computational speedups or better performance for machine learning tasks [8-14], and parameterized quantum circuits (PQCs) offer a promising path for quantum machine learning in the NISQ era [15-17]. Compared with traditional quantum algorithms [12,18-20] such as Shor's algorithm [18,21,22], PQC-based quantum machine learning algorithms are naturally robust to noise [23] and only require shallow quantum circuits, which is highly desirable for near-term noisy intermediate scale quantum devices. Compared with classical neural networks, PQC-based quantum machine learning algorithms have two major potential advantages, stronger expressive power [24-29] and stronger computing power [30], originating from the superposition principle of quantum mechanics. Those advantages and the potential applications of PQCs on near-term quantum devices have stimulated a large number of related algorithms and experiments, including variational quantum eigensolvers (VQE) [31], the quantum approximate optimization algorithm (QAOA) [32], quantum generative adversarial networks [30,33-36], and quantum classifiers [28,37]

For QML to solve real world problems, the first step is to translate classical data, which is usually represented as a multi-dimensional array, into a quantum state. A standard way is to use a kernel function to map each element of the array into a single-qubit state, which is often referred to as qubit-encoding. The kernel function could, for example, be chosen as:

$$\theta_j \rightarrow \cos(\theta_j)|0\rangle + \sin(\theta_j)|1\rangle, \tag{1}$$

which maps a real number into a single-qubit quantum state. For an input array of size L, the mapping would result in an L-qubit quantum state, which lives in a Hilbert space of size 2^L. For real world data with a large size, this mapping would soon become impractical for current quantum computers with less than 100 qubits. The same problem also exists in classical deep learning, which is often built from interlacing layers of linear and nonlinear functions. A straightforward way to implement the linear function is the so-called

fully connected layer, which can be represented as a dense matrix connecting each neuron of the output to all the neurons of the input. When the input size is large, this approach would become inefficient due to the large matrix size. The convolutional neural network (CNN) [38,39] is a very popular scheme which tries to solve this problem by replacing the fully connected layer with a convolutional layer. The convolutional layer only connects each neuron of the output to a small region (window) of the input which is referred to as a feature map, thus greatly reducing the number of parameters. CNN has demonstrated itself as one of the most successful tools in the area of computer vision [40-44], and more recently, it also found applications in natural language processing [45-50].

Inspired by CNN, we propose a hybrid quantum-classical convolutional neural network (QCCNN). We note that recently, a pure quantum analogy of CNN, named QCNN, was proposed to solve certain quantum many-body problems [51]. Similar to other QML algorithms, QCNN uses as many qubits as the size of the input, which makes it unlikely to be implemented on current quantum computers to solve real world problems. The central idea of QCCNN is to implement the feature map in the convolutional layer with a parametric quantum circuit, and correspondingly, the output of this feature map is a correlational measurement on the output quantum state of the parametric quantum circuit. In the following, we refer to this new structure as a quantum convolutional layer. As a result, the number of qubits required by this approach is only related to the window size of the feature map, which often ranges from 3×3 to 9×9 and is well within the capability of current quantum computers. Moreover, since the output of our quantum convolutional layer is a classical array, it is straightforward to adapt the multi-layer structure as in CNN. Therefore, our QCCNN could utilize all the features of classical CNN, and at the same time, it is able to utilize the power of current NISQ computers. In addition, we propose a framework to automatically compute the gradients of arbitrary hybrid quantum-classical loss functions using a hybrid quantum-classical computer. To demonstrate the representative power of our QCCNN, we apply it for classifying the synthetic Tetris dataset and compare its learning accuracy to classical CNN with the same architecture.

The paper is organized as follows. In sect. 2, we introduce our hybrid quantum-classical Convolutional Neural Network architecture. In sect. 3, we show a framework which can be used to automatically compute the exact gradients of any hybrid quantum-classical neural networks. In sect. 4, we demonstrate our method with the Tetris dataset and show that it can reach a higher accuracy compared to classical CNN. Finally we conclude in sect. 5.
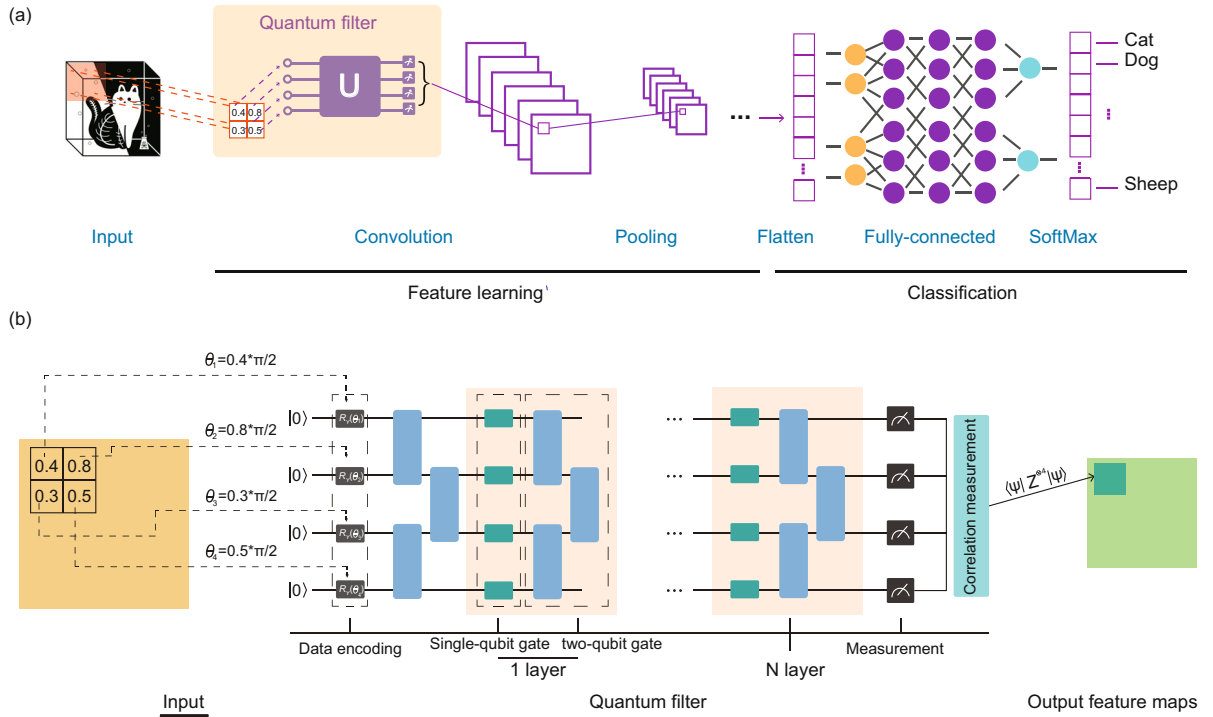
2 Hybrid QCCNN

To better describe our design of QCCNN (see Figure 1(a)), we first briefly outline some basic features of CNN. CNN consists of interlaced convolutional layers and pooling layers, and ends with a fully connected layer. The primary purpose of the convolution layer is to extract features from the input data using a feature map (or filter), which is the most computational intensive step of CNN. After the convolutional layer, it is common to add a pooling layer to reduce the dimensionality of the data and prevent overfitting. A single filter maps small regions (windows) of the input to single neurons of the output, and is parameterized by an array P which has the same shape as the window. The windows are often chosen as follows. Assuming the input is a two-dimensional array A of size $v \times h$, and the predefined window size is $m \times n$, then the first window is located at the upper left conner of A, namely $A_{1:m,1:n}$ (here $a : b$ denotes the range from a to b). Then the mapping is done by the linear function

$$A_{1:m,1:n} \rightarrow \sum_{1 \leq i \leq m, 1 \leq j \leq n} A_{i,j} P_{i,j}. \qquad (2)$$

Then the next window slides to the right with a stride value s, which is often chosen to be 1, until it reaches the right edge. After that it hops down to the (left) beginning of the image with the same stride value s and repeats the process until the entire image is traversed. As a result, after the evolution, the output will be a two-dimensional array of size $\frac{v-m+1}{s} \times \frac{h-n+1}{s}$. Moreover, in general one could have several filters in the same layer and the input could be a three-dimensional array. For example, for a three-dimensional array of size $v \times h \times d$, and if we have k filters with size $m \times n$, assuming the stride s, then the output array would have the shape $\frac{v-m+1}{s} \times \frac{h-n+1}{s} \times (dk)$. Generally after a convolutional layer, the output would become thinner but longer. In some situations, one would like to prevent the data to become thinner, by adding zeros around the edges of input, which is referred to as padding.

In our quantum convolutional layer, the filter is redesigned to make use of the parametric quantum circuit, which is shown in Figure 1(b), and we refer to it as a quantum filter. A quantum filter takes windows of shape $m \times n$, maps them into quantum states $|\psi^i(A_{p:(p+m-1),q:(q+n-1)})\rangle$ of $N = mn$ qubits

Figure 1 (Color online) (a) Hybrid QCCNN. The input demonstrated here is a two-dimensional array, which is sent to a quantum convolutional layer of 6 filters. Each filter takes a 2×2 window, translating it into a separable 4-qubit quantum state, and evolves this state with a parametric quantum circuit. For images encoded as three-dimensional arrays, the filter only works on the first two dimensions. After that a correlational measurement is made on the output quantum state and a scalar is obtained. Gathering the scalar outputs, the final output of the quantum convolutional layer is a 3-dimensional array. Then a pooling layer is used to reduce the dimensionality of the data. This process could be repeated and finally ends with a fully connected layer. (b) Details of our design of parametric quantum circuit, which is made of interlaced single-qubit layer and two-qubit layers. The single-qubit layer consists of R_y gates, each containing one tunable parameter. The two-qubit layer consists of CNOT gates on nearest-neighbour pairs of qubits.

using eq. (1), and then evolves the quantum state with the parametric quantum circuit $C(\boldsymbol{\theta})$, such that the output quantum state $|\psi^o\rangle$ is

$$|\psi^o\rangle = C(\boldsymbol{\theta})|\psi^i(A_{p:(p+m-1),q:(q+n-1)})\rangle. \qquad (3)$$

After the evolution, we take the expectation value of the observable $Z^{\otimes N}$, thus the feature map can be written as:

$$A_{p:(p+m-1),q:(q+n-1)} \rightarrow \langle\psi^o|Z^{\otimes N}|\psi^o\rangle. \qquad (4)$$

Eq. (4) is nonlinear and thus in our quantum convolutional layer, we do not need an additional nonlinear function such as ReLU to explicitly bring in nonlinearity. It is also clear from eq. (4) that in our approach, the minimal number of qubits required is equal to the window size. For the next window of our quantum convolutional layer, these qubits could be reused. Thus, the quantum convolutional layer is experimentally friendly and suitable for NISQ scenarios because only a few qubits are needed and no additional usage of qRAM is required. And the quantum correlational measurement has the potential to better capture the cross-correlation inside each window. In our architecture, the pooling layers as well as the final fully connected layer are kept in the same way as CNN since they are computationally cheap, and can further induce nonlinearities.

The parametric quantum circuit we use contains interlaced single-qubit layers and two-qubit layers. The total number of gate operations of a parametric quantum circuit, denoted as L, grows only polynomially with the number of qubits N, namely $L \sim \mathrm{poly}(N)$ gates. In our setup, the two-qubit layer consists of CNOT gates while the single qubit layer consists of rotational Y gates (R_y), which is defined as:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}. \qquad (5)$$

Here we have used R_y gate instead of other single-qubit rotational gates since then we could represent the quantum state as well as the parametric quantum circuit using real numbers for simplicity. Each layer of rotational gates is counted as one depth, as a result the total number of parameters in one parametric quantum circuit will be the window size times the circuit depth. For optimization problems, it is usually helpful to provide the gradient of the loss function. The easiest way to approximately compute the gradient in our case is to use the finite difference method, which only requires forward evaluation of the loss function.

To this end, we stress that although in Figure 1 we have focused on classical input data (pictures), our QCCNN architecture could also be adapted to train quantum input data. For example, for an input quantum state on a square lattice of

$n \times n$ qubits, our quantum filter could directly work on each 2×2 sub-lattice without the classical-to-quantum encoding stage, while the rest of the architecture remains the same. As a result, QCCNN could be used, for example, as a variational ansatz for the ground state of quantum many-body systems, or to classify quantum states as in ref. [51].

3 Hybrid auto-differentiation framework

In the following we present a framework to compute the gradient of a hybrid quantum-classical loss function using auto-differentiation, assuming that the "quantum" sub functions involved in the loss function have the form of eq. (4). We starting by showing how to compute the gradient of the quantum feature map, namely eq. (4). There are two cases: (1) The input A is constant and the derivative against A is not required and (2) the input A is the output of previous steps and the derivative against A is required for the following backward propagation process. In the second case, we can add a single-qubit layer of R_y into the parametric circuit, whose parameters correspond to the values of A, and then the problem reduces to the first case. Therefore, it is enough to consider the gradient of the function in eq. (4), which is well-known to be [52]

$$\frac{\partial\langle\psi^o|Z^{\otimes N}|\psi^o\rangle}{\partial\theta_j} = \frac{1}{2}\left(\langle\psi^i|C^\dagger(\boldsymbol{\theta}_j^+)Z^{\otimes N}C(\boldsymbol{\theta}_j^+)|\psi^i\rangle\right.$$
$$\left. - \langle\psi^i|C^\dagger(\boldsymbol{\theta}_j^-)Z^{\otimes N}C(\boldsymbol{\theta}_j^-)|\psi^i\rangle\right), \qquad (6)$$

where $\boldsymbol{\theta}_j^\pm$ means to shift the j-th parameter of $\boldsymbol{\theta}$, θ_j by $\pm\frac{\pi}{2}$ respectively. Now we assume for simplicity that there is a loss function $F(\boldsymbol{\theta})$ which can be written as:

$$F(\boldsymbol{\theta}) = f\left(\langle\psi^o|Z^{\times N}|\psi^o\rangle\right), \qquad (7)$$

where f is some classical function. One may attempt to compute the gradient of $F(\boldsymbol{\theta})$ by computing $F(\boldsymbol{\theta}_j^+)$ and $F(\boldsymbol{\theta}_j^-)$ and then do $\partial F(\boldsymbol{\theta})/\partial\theta_j = (F(\boldsymbol{\theta}_j^+) - F(\boldsymbol{\theta}_j^-))/2$ similar to eq. (6), which however is incorrect in general except a few cases where f is extremely simple. Other than the way to analytically derive the gradient of eq. (7) by hand, a correct and elegant way is to integrate it into the auto-differentiation framework, for which one needs to provide the adjoint function for eq. (4) as a standard practice to extend auto-differentiation to a user-defined function. The adjoint function of eq. (4) is

$$z \rightarrow z \times \frac{\partial\langle\psi^o|Z^{\otimes N}|\psi^o\rangle}{\partial\theta_j}, \qquad (8)$$

which maps an input scalar z to an output array with the same size as $\boldsymbol{\theta}$. Simply speaking, eq. (8) takes as input the gradient z from the outer function f and multiplies it with the gradient of the current function, the output of which is further

passed to its inner function if there exists. Once the correct adjoint functions for each of the elementary functions of the loss function have been defined, the computer will be able to derive the gradient for the loss function using the auto-differentiation framework [53, 54]. The only difference here is that the adjoint function in eq. (8) depends on the output from a quantum computer. Therefore one needs to embed eq. (8) into the back-propagation process, just as one embeds eq. (4) as a sub function into the classical neural network. Then the gradient of a hybrid quantum-classical loss function would be able to be derived automatically using a hybrid quantum-classical computer. In this work the whole process is simulated using a classical computer.

Training our hybrid quantum-classical CNNs works in the same way as a regular neural network. Various gradient-based optimization techniques, such as stochastic, batch, or mini-batch gradient descent algorithms, can be used to optimize the parameters of the hybrid quantum-classical CNNs. Once the model has been trained, it can be then used to predict outputs for given inputs.

4 Numerical results and discussions

We demonstrate the potential of our QCCNN by applying it to the Tetris dataset. We create a Tetris image dataset that consists of 800 gray-scale images with shape 3×3, in which each gray-scale image is a simulated Tetris brick (refer to Figure 2 for some samples). Concretely, the foreground pixels are represented by random floating numbers ranging from 0.7 to 1, whereas the background are small floating numbers ranging from 0 to 0.1. There are 4 classes, namely S, L,

O, and T, each of which represents a type of Tetris bricks. The dataset is further processed by randomly splitting into a training set and a testing set that contain 80% and 20% of the images, respectively. We benchmark our QCCNN against CNN with two particular structures, namely one with a single convolutional layer and another with two convolutional layers. To see the performances with a different number of classes, we create another dataset by only picking the two classes S, T out of the original training and testing data. For the single-layer structure, we use a single (quantum) convolutional layer with 5 filters with no padding, plus a pooling layer also with no padding. For the two-layer structure, we use two (quantum) convolutional layers with 2 and 3 filters respectively, plus a pooling layer in between with padding 1. The window shape for all the layers is 2×2, and the stride value $s = 1$. Therefore the number of qubits fed to the quantum filter is 4. The depth of each parametric quantum circuit is set to 4 to ensure high expressibility, as a result the total number of parameters in a quantum feature map is 16. In comparison, the number of parameters in a classical feature map is 4, which is the same as the window size. The effect of different circuit depths in a quantum feature map will be discussed later. We use the mean square loss as our loss function F^q, defined as:

$$F^q(\boldsymbol{a}) = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left(f^q(\boldsymbol{x}_j) - \boldsymbol{y}_j \right)^2, \tag{9}$$

where $\boldsymbol{a}$ is a list of all the parameters, f^q means the QCCNN which is dependent on $\boldsymbol{a}$ and outputs a vector whose size is equal to the number of labels, $\boldsymbol{x}_j$ represents the j-th input image, y_j is an integer which represents the label corresponding to $\boldsymbol{x}_j$, and $\boldsymbol{y}_j$ is the one-hot vector converted from y_j. N_{train} is
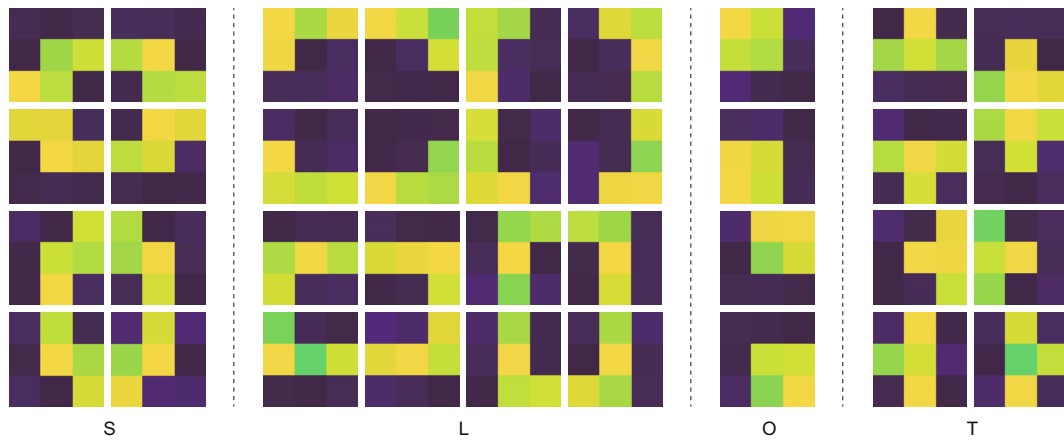
Figure 2 (Color online) Some samples of the Tetris dataset. The dataset contains 1000 gray-scale images with shape 3×3. In the dataset, there are four types of Tetris bricks labeled with S, L, O, T, which have 8, 16, 4 and 8 possible configurations in the gray-scale images, respectively. For each image, the foreground pixels are represented by random floating numbers ranging from 0.7 to 1, whereas the background is small floating numbers ranging from 0 to 0.1. The gray-scale images are plotted using matplotlib.pyplot.imshow as pseudocolor images, so that the pixels with larger numbers are represented with bright colors.

the size of the training set. The accuracy A^q on the testing set is defined as:

$$A^q(\boldsymbol{a}) = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \left(\text{argmax}(f^q(\boldsymbol{x}_j)) == y_j \right), \qquad (10)$$

where the function $\text{argmax}(\boldsymbol{v})$ outputs the position of the largest element in the input vector $\boldsymbol{v}$, and N_{test} is the size of the testing set. The loss function and accuracy in the classical case are defined in the same way, with QCCNN replaced by CNN in f^q. During 1000 iterations, we compute the accuracy on the testing data and store the values of the loss function, which is chosen as mean square loss. In Figure 3(a) and (c), we plot the accuracy and loss values for the 2-label case. While in Figure 3(b) and (d) [55] we plot the accuracy and loss values for the 4-label case. We can see that QCCNN can reach almost 100% accuracy for both the two structures we have used, and it can reach lower loss values for both cases compared to its classical counterpart. Benefiting from the high-dimensional nature of the quantum system, the advantages of QCCNN become more transparent when the number of classes increases from 2 to 4. We can also see that the 4-label case takes more iterations to converge than the 2-label case, and that QCCNN with a two-layer structure converges faster than the single-layer structure, especially in the 4-label case, which indicates that for complex problems, better performance could be achieved by deeper architectures.

To gain deeper insight into our QCCNN, we tune the depth of the parametric quantum circuit (thus changing the number of parameters) to see its effect on the training accuracy.

Concretely, for the two-layer QCCNN, we change the circuit depth to be $1, 2, 4$ (the number of parameters in each quantum feature map is thus $4, 8, 16$), and the training accuracies are show in Figure 4(a) with gray lines from lighter to darker. For comparison, in Figure 4(a) we also show the simulation results of the corresponding two-layer CNN in green dotted line and a modified two-layer CNN in green solid line for which the window size of the first layer is enlarged to 3×3 such that we could have more parameters. We can see that the QCCNN with 1-depth quantum circuit could already reach a higher accuracy than CNN (they contain exactly the same number of parameters), and that QCCNN performs better with larger circuit depth, which is reasonable since it needs enough depth to fully entangle all the qubits. The performance of CNN could also be better when the number of parameters increase since it could explore a larger space in the first layer (space of dimension 9 instead of 4), which however is still slightly worse than QCCNN with circuit depths 2 and 4. Since current quantum devices subject to noises, we also study the effects of noisy quantum circuits on the training performance. Concretely, in Figure 4(b) we show the training accuracy for a perfect (yellow dashed line) and noisy (green dashed line) one-layer QCCNN, where in the noisy quantum circuits we have considered the decoherence time T_1, T_2 as well as the measurement errors. We can see that the performance is quite resilient to such errors, and a training accuracy 1 could still be reached in the noisy case.

Till now we have mainly focused on the training accuracy of QCCNN, and we have shown that QCCNN could poten-
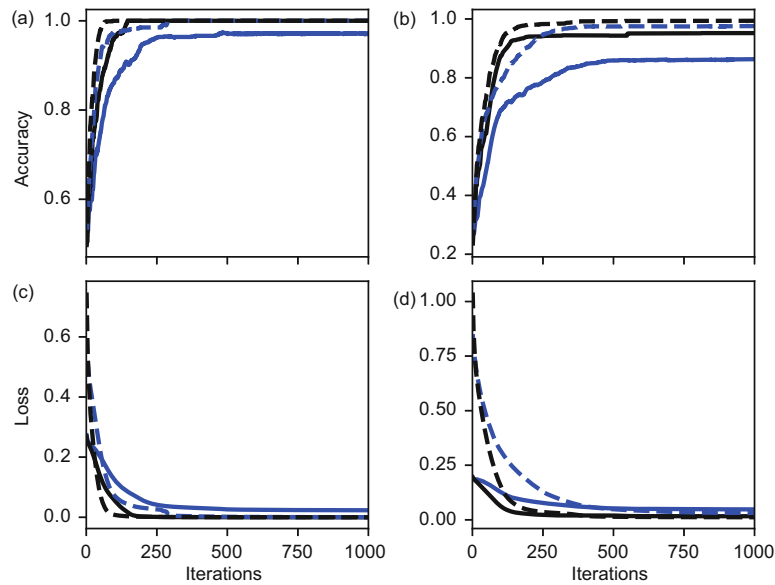
Figure 3   (Color online) Accuracy and loss as a function of the number of iterations. In all the figures, the blue line represents the result of one-layer CNN and the black line represents the result of two-layer CNN, the blue dashed line represents the result of one-layer QCCNN and the black dashed line represents the result of two-layer QCCNN. We have used the optimizer ADAM [55] and an initial learning rate of 0.01 and 1000 iterations. The results are averaged over 10 random simulations. (a) Accuracy in case of 2 classes; (b) accuracy in case of 4 classes; (c) loss in case of 2 classes; (d) loss in case of 4 classes.
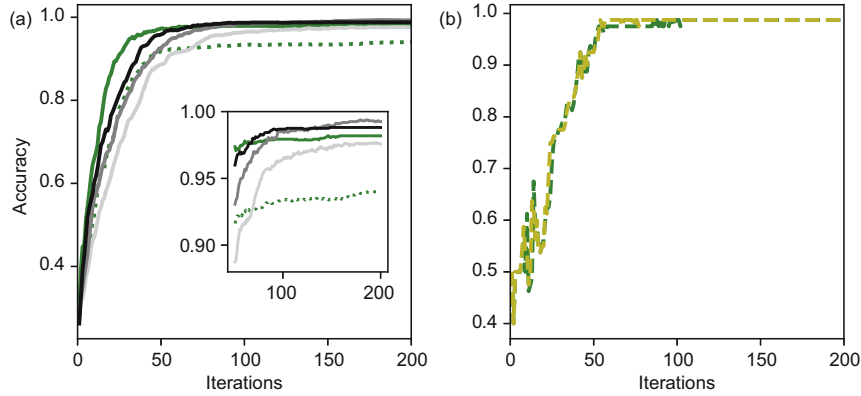
Figure 4   (Color online) (a) Accuracy as a function of the number of iterations for the two-layer QCCNN with quantum feature maps of depths 1, 2, 4, which are shown in gray lines from lighter to darker, and for the two-layer CNN which is shown in dotted green line. The solid green line represents the accuracies computed with a modified two-layer CNN, in which the first convolutional layer uses a window size 3×3 with padding 1, followed by a pooling layer also with a window size 3×3 and padding 1. The rest of the modified two-layer CNN keeps the same as before. The inset shows the accuracies starting from the 50-th iteration. The accuracies are averaged over 10 random initializations and four output labels are used. (b) Accuracy as a function of the number of iterations for the noisy (green dashed line) and perfect (yellow dashed line) one-layer QCCNN with quantum feature map of depth 2. Here 2 output labels are used, and a single initialization with random seed 100 is simulated due to the simulation efficiency in case of noisy quantum circuits and limited computational resources. We have assumed a single-qubit gate operation time of 100 ns, a CNOT gate operation time of 300 ns, and a measurement time of 1000 ns. The decoherence time T_1 and T_2 for the 4 qubits are randomly chosen from normal distributions with mean 50 and 70 µs, respectively. We have used the ADAM optimizer with an initial learn rate 0.05 and 200 iterations.

tially achieve a higher classification accuracy than CNN. We note that for the same window size N, the quantum feature map requires $O(Nd)$ gate operations for a parametric quantum circuit of depth d, while the classical feature map requires $O(N)$ Floating-point arithmetics (vector inner product). As a result from the point of computational efficiency QCCNN would not be advantageous. We could also use amplitude encoding instead of qubit encoding in principle, such that a parametric quantum circuit of N qubits could process 2^N features in a single quantum feature map. However if naively implemented this approach would have the input problem that the encoding stage itself has a complexity of $O(2^N)$, which is already the same as the classical feature map (our quantum feature map does not have the output problem since it outputs a single expectation value). If a better problem could be found which is free of such input problem, one could probably achieve quantum advantage in terms of computational time with our QCCNN framework.

5   Conclusion

In summary, we present a hybrid QCCNN which could be used to solve real world problems with current quantum computers. QCCNN avoids the input and output problems of quantum machine learning algorithms when applied to real-world problems, which is one of the major challenges for current quantum machine learning algorithms. As a quantum machine learning algorithm inspired by classical CNN, QCCNN keeps the features of CNN such as the nonlinear-

ity, locality of the convolutional layer, as well as extensibility to deep structures. Similar to CNN, our QCCNN architecture provides a framework for developing various hybrid quantum-classical machine learning applications on near-term quantum devices, such as the discriminative model and generative model. Moreover, the generalized feature map with a parametric quantum circuit is able to explore the correlations of neighbouring data points in an exponentially large linear space, hopefully allowing our algorithm to capture the patterns in the dataset more precisely with a quantum computer.

In addition, we also present a framework to automatically compute the gradients of hybrid quantum-classical loss functions, which could be a useful tool for developing complicated hybrid quantum-classical variational algorithms in the future. We demonstrate our approach on the Tetris dataset and show the potential of our approach to reach better learning precision for classification problems.

Note added. During the preparation of this work, we notice a similar work that uses qRAM [56] and another work that uses non-parametric random quantum circuits for feature mapping [57], which were carried out independently.

1 J. Preskill, Quantum **2**, 79 (2018).

2 H. L. Huang, D. Wu, D. Fan, and X. Zhu, Sci. China Inf. Sci. **63**, 180501 (2020).

3 F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. Mc-Clean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, Nature **574**, 505 (2019), arXiv: 1910.11333.

4 A. W. Harrow, and A. Montanaro, Nature **549**, 203 (2017), arXiv: 1809.07442.

5 C. Guo, Y. Liu, M. Xiong, S. Xue, X. Fu, A. Huang, X. Qiang, P. Xu, J. Liu, S. Zheng, H. L. Huang, M. Deng, D. Poletti, W. S. Bao, and J. Wu, Phys. Rev. Lett. **123**, 190501 (2019), arXiv: 1905.08394.

6 C. Guo, Y. Zhao, and H. L. Huang, Phys. Rev. Lett. **126**, 070502 (2021), arXiv: 2011.02621.

7 F. Pan, and P. Zhang, arXiv: 2103.03074.

8 J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Nature **549**, 195 (2017), arXiv: 1611.09347.

9 S. Lloyd, M. Mohseni, and P. Rebentrost, Nat. Phys. **10**, 631 (2014), arXiv: 1307.0401.

10 S. Lloyd, S. Garnerone, and P. Zanardi, Nat. Commun. **7**, 10138 (2016).

11 P. Rebentrost, M. Mohseni, and S. Lloyd, Phys. Rev. Lett. **113**, 130503 (2014), arXiv: 1307.0471.

12 H. L. Huang, X. L. Wang, P. P. Rohde, Y. H. Luo, Y. W. Zhao, C. Liu, L. Li, N. L. Liu, C. Y. Lu, and J. W. Pan, Optica **5**, 193 (2018), arXiv: 1801.06316.

13 H. L. Huang, Y. W. Zhao, T. Li, F. G. Li, Y. T. Du, X. Q. Fu, S. Zhang, X. Wang, and W. S. Bao, Front. Phys. **12**, 120305 (2017), arXiv: 1612.02886.

14 C. Ding, T.-Y. Bao, and H.-L. Huang, arXiv: 1906.08902.

15 J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, New J. Phys. **18**, 023023 (2016), arXiv: 1509.04279.

16 M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, Quantum Sci. Technol. **5**, 019601 (2020).

17 Y. Liu, D. Wang, S. Xue, A. Huang, X. Fu, X. Qiang, P. Xu, H. L. Huang, M. Deng, C. Guo, X. Yang, and J. Wu, Phys. Rev. A **101**, 052316 (2020), arXiv: 1912.07286.

18 P. W. Shor, SIAM Rev. **41**, 303 (1999).

19 L. K. Grover, in *A fast quantum mechanical algorithm for database search: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, Philadelphia, 1996, pp. 212-219.

20 G. L. Long, Phys. Rev. A **64**, 022307 (2001), arXiv: quant-ph/0106071.

21 C. Y. Lu, D. E. Browne, T. Yang, and J. W. Pan, Phys. Rev. Lett. **99**, 250504 (2007), arXiv: 0705.1684.

22 H. L. Huang, Q. Zhao, X. Ma, C. Liu, Z. E. Su, X. L. Wang, L. Li, N. L. Liu, B. C. Sanders, C. Y. Lu, and J. W. Pan, Phys. Rev. Lett. **119**, 050503 (2017), arXiv: 1707.00400.

23 L. Gentini, A. Cuccoli, S. Pirandola, P. Verrucchi, and L. Banchi, Phys. Rev. A **102**, 052414 (2020), arXiv: 1912.06744.

24 N. Killoran, T. R. Bromley, J. M. Arrazola, M. Schuld, N. Quesada, and S. Lloyd, Phys. Rev. Res. **1**, 033063 (2019), arXiv: 1806.06871.

25 Y. Du, M. H. Hsieh, T. Liu, and D. Tao, Phys. Rev. Res. **2**, 033125 (2020).

26 G. García-Pérez, M. A. C. Rossi, and S. Maniscalco, npj Quantum Inf. **6**, 1 (2020), arXiv: 1906.07099.

27 J. Tangpanitanon, S. Thanasilp, N. Dangniam, M. A. Lemonde, and D. G. Angelakis, Phys. Rev. Res. **2**, 043364 (2020), arXiv: 2005.11222.

28 M. Schuld, and N. Killoran, Phys. Rev. Lett. **122**, 040504 (2019).

29 S. D. Sarma, D.-L. Deng, and L.-M. Duan, arXiv: 1903.03516.

30 S. Lloyd, and C. Weedbrook, Phys. Rev. Lett. **121**, 040502 (2018), arXiv: 1804.09139.

31 A. Peruzzo, J. McClean, P. Shadbolt, M. H. Yung, X. Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. OBrien, Nat. Commun. **5**, 4213 (2014), arXiv: 1304.3061.

32 E. Farhi, J. Goldstone, and S. Gutmann, arXiv: 1411.4028.

33 P. L. Dallaire-Demers, and N. Killoran, Phys. Rev. A **98**, 012324 (2018), arXiv: 1804.08641.

34 H.-L. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu, R. Yang, T. Liu, M.-H. Hsieh, H. Deng, H. Rong, C.-Z. Peng, C.-Y. Lu, Y.-A. Chen, D. Tao, X. Zhu, and J.-W. Pan, arXiv: 2010.06201.

35 L. Hu, S. H. Wu, W. Cai, Y. Ma, X. Mu, Y. Xu, H. Wang, Y. Song, D. L. Deng, C. L. Zou, and L. Sun, Sci. Adv. **5**, eaav2761 (2019), arXiv: 1808.02893.

36 K. Huang, Z.-A. Wang, C. Song, K. Xu, H. Li, Z. Wang, Q. Guo, Z. Song, Z.-B. Liu, D. Zheng, D.-L. Deng, H. Wang, J.-G. Tian, and H. Fan, arXiv: 2009.12827.

37 V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, Nature **567**, 209 (2019), arXiv: 1804.11326.

38 K. Fukushima, Biol. Cybernetics **36**, 193 (1980).

39 Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Shape, Contour and Grouping in Computer Vision* (Springer, Berlin, Heidelberg, 1999), pp. 319-345.

40 A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., 2012), pp. 1097-1105.

41 O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, Int. J. Comput. Vis. **115**, 211 (2015).

42 K. Simonyan, and A. Zisserman, arXiv: 1409.1556.

43 K. He, X. Zhang, S. Ren, and J. Sun, in *Deep residual learning for image recognition: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, 2016, pp. 770-778.

44 I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, arXiv: 1406.2661.

45 Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, arXiv: 1612.08083.

46 J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, arXiv: 1705.03122.

47 Y. Zhang, and B. Wallace, arXiv: 1510.03820.

48 A. Kirillov, D. Schlesinger, W. Forkel, A. Zelenin, S. Zheng, P. Torr, and C. Rother, arXiv: 1511.05067.

49 S. Song, H. Huang, and T. Ruan, Multimed Tools Appl. **78**, 857 (2019).

50 A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, arXiv: 1804.09541.

51 I. Cong, S. Choi, and M. D. Lukin, Nat. Phys. **15**, 1273 (2019), arXiv: 1810.03787.

52 K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Phys. Rev. A **98**, 032309 (2018), arXiv: 1803.00745.

53 D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Nature **323**, 533 (1986).

54 C. Guo, and D. Poletti, Phys. Rev. E **103**, 013309 (2021).

55 D. P. Kingma, and J. Ba, arXiv: 1412.6980.

56 I. Kerenidis, J. Landman, and A. Prakash, arXiv: 1911.01117.

57 M. Henderson, S. Shakya, S. Pradhan, and T. Cook, Quantum Mach. Intell. **2**, 2 (2020).