QDNN: DNN with Quantum Neural Network Layers*

Chen Zhao[1,2] and Xiao-Shan Gao[1,2]

[1] *KLMM, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China*
[2] *University of Chinese Academy of Sciences, Beijing 100049, China*

Abstract

The deep neural network (DNN) became the most important and powerful machine learning method in recent years. In this paper, we introduce a general quantum DNN, which consists of fully quantum structured layers with better representation power than the classical DNN and still keeps the advantages of the classical DNN such as the non-linear activation, the multi-layer structure, and the efficient backpropagation training algorithm. We prove that the quantum structured layer can not be simulated efficiently by classical computers unless universal quantum computing can be classically simulated efficiently and hence our quantum DNN has more representation power than the classical DNN. Moreover, our quantum DNN can be used on near-term noisy intermediate scale quantum (NISQ) processors. A numerical experiment for image classification based on quantum DNN is given, where high accurate rate is achieved.

Keywords. Quantum DNN, quantum neural network layer, hybrid quantum-classical algorithm, NISQ computing.

1 Introduction

Quantum computers use the principles of quantum mechanics for computing, which are more powerful than classical computers in many computing problems [1, 2]. Noisy intermediate scale quantum (NISQ) [3] devices will be the only quantum devices that can be used in near-term, where we can only use a limited number of qubits without error correcting. So developing NISQ algorithms is a new challenge.

In this paper, we will focus on quantum machine learning. Many quantum machine learning algorithms, such as qSVM, qPCA and quantum Boltzmann machine, have been developed [4–10], and these algorithms were shown to be more efficient than their classical versions. Recently, several NISQ quantum machine learning algorithms, such as QuGAN, QCBM and quantum kernel methods, have been proposed [11–15]. However, these algorithms did not aim to build quantum deep neural networks.

*Partially supported by a NSFC grant no. 11688101 and a NKRDP grant no. 2018YFA0306702.

In recent years, deep neural network [16] became the most important and powerful method in machine learning, which was widely applied in computer vision [17], natural language processing [18], and many other fields. The basic unit of DNN is the perception, which is an affine transform together with an activation function. The non-linearity of the activation function and the depth give the DNN more representation power. Approaches have been proposed to build classical DNNs on quantum computers [19–21]. They achieved quantum speed-up under certain assumptions. But the structure of classical DNNs is still used, since only some operations are speeded up by quantum algorithms, for instance, to speedup the inner product using the swap test [20].

In this paper, we introduce the first quantum analog to classical DNN, which consists of fully quantum structured layers with better representation power than the classical DNN and still keeps the advantages of the classical DNN such as the non-linear activation, the multi-layer structure, and the efficient backpropagation training algorithm.

The main contribution of this paper is to introduce the concept of quantum neural network layer (QNNL) as a quantum analog to the classic neural network layer in DNN. As all quantum gates are unitary and hence linear, the main difficulty of building a QNNL is introducing non-linearity. We solved this problem by encoding the input vector to a quantum state non-linearly with a PQC. A QNNL is a quantum circuit which is totally different from the classical neural network layer. A quantum DNN (QDNN) can be easily built with QNNLs, since the input and output of a QNNL are classical values.

The advantage of introducing QNNLs is that we can access vectors of exponential dimensional Hilbert spaces with only polynomial resources on a quantum computer. We proved that this model can not be classically simulated efficiently unless universal quantum computing can be classically simulated efficiently. So QDNNs have more representation power than classical DNNs. We also give training algorithms of QDNNs which are similar to backpropagation (BP) algorithm. Moreover, QNNLs use the hybrid quantum-classical scheme. Hence, a QDNN with a reasonable size can be trained efficiently on NISQ processors. Finally, a numerical experiment for an image recognition is given using QDNNs, where high accurate rate is achieved.

We finally remark that all tasks using DNN can be turned into quantum algorithms with more representation powers by replacing the DNN by QDNN.

2 Hybrid quantum-classical algorithm

The hybrid quantum-classical algorithm scheme [22] consists of a quantum part and a classical part. In the quantum part, one uses parametric quantum circuits (PQCs) to prepare quantum states using quantum processors. In the classical part, one uses classical computers for optimizing the parameters of the PQCs in the quantum parts.

2.1 Hybrid quantum-classical scheme based on PQCs

PQCs are quantum circuits with parametric gates. In general, a PQC is of the form

$$U(\vec{\theta}) = \prod_{j=1}^{l} U_j(\theta_j)$$

where $\vec{\theta} = (\theta_1, \ldots, \theta_l)$ are the parameters, each $U_j(\theta_j)$ is a rotation gate $U_j(\theta_j) = \exp(-i\frac{\theta_j}{2}H_j)$, and H_j is a 1-qubit or a 2-qubits gate such that $H_j^2 = I$. For example, in this paper we will use the Pauli gates X, Y, Z, and the CNOT gate.

In practical tasks such as VQE [23] and quantum machine learning [12], we want to find a quantum state $|\psi\rangle$ with certain desired properties. This can be done with the following three steps based on the hybrid quantum-classical scheme. First, we need to choose an appropriate ansatz, that is, designing the circuit structure of a PQC $U(\vec{\theta})$. All parameters $\vec{\theta}$ are initialized randomly. Then we apply this PQC to a fixed initial state $|\varphi_0\rangle$, for instance $|0\rangle$. Second, by measuring the final state $|\psi\rangle = U(\vec{\theta})\,|\varphi_0\rangle$ repeatedly, we can estimate the expected value $L = \langle\psi| H |\psi\rangle$ for an Hamiltonian H. H will be designed differently in different tasks. In many tasks, the ground state of H is our goal. To achieve this goal, in the final step, we optimize the loss function L, by updating parameters $\vec{\theta}$ on classical computers.

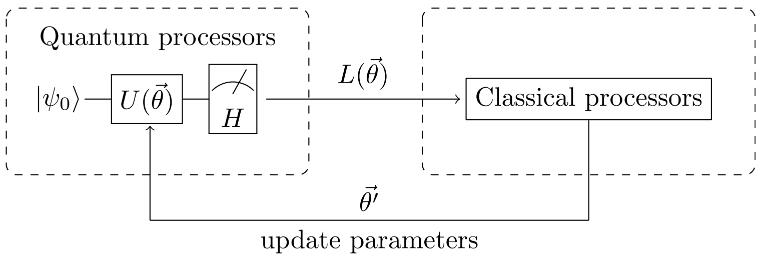


Figure 1: Hybrid quantum-classical scheme.

In summary, a *hybrid quantum-classical scheme*, as shown in Figure 1, consists of a PQC $U(\theta)$ and a loss function of the form $L = \langle\psi| H |\psi\rangle$ together with a classical algorithm for updating parameters, where H is a Hamiltonian.

2.2 Optimization in hybrid quantum-classical scheme

There are many methods for optimizing the loss function for a hybrid quantum-classical scheme based on PQCs. Some are gradient-based [24] and some are gradient-free [25]. We will focus on gradient-based algorithms in this paper.

The gradient $\nabla_{\vec{\theta}} L$ can be estimated by shifting parameters of PQCs without changing the circuit structure. The detail of the gradient estimation algorithm can be found in Appendix A. Once the gradient is obtained, we can use the gradient descent method to update the parameters.

3 DNNs with quantum neural network layers

In this section, we will introduce the concepts of quantum neural network layer (QNNL) and quantum DNN, and give a training algorithm for the quantum DNN.

3.1 QNNL and QDNN

A DNN consists of a large number of *neural network layers*, and each neural network layer is a non-linear function $f_{\overrightarrow{W},\vec{b}}(\vec{x}) : \mathbb{R}^n \to \mathbb{R}^m$ with parameters $\{\overrightarrow{W}, \vec{b}\}$. In the classical DNN, $f_{\overrightarrow{W},\vec{b}}$ takes the form of $\sigma \circ L_{\overrightarrow{W},\vec{b}}$, where $L_{\overrightarrow{W},\vec{b}}$ is an affine transform and σ is a non-linear activation function. The power of DNNs comes from the non-linearity of the activation function. Without activation functions, DNNs will be nothing more than affine transforms.

However, all quantum gates are unitary matrices and hence linear. So the key point of developing QNNLs is introducing non-linearity.

Suppose that the input data $\vec{x} \in \mathbb{R}^n$ is classical. We introduce non-linearity to our QNNL by encoding the input $\vec{x}$ to a quantum state $|\psi(\vec{x})\rangle$ non-linearly. Concretely, we will use a PQC for this process. Choose a PQC $U(\vec{x})$ with at most $O(n)$ qubits and apply it to an initial state $|\psi_0\rangle$. We obtain a quantum state

$$|\psi(\vec{x})\rangle = U(\vec{x})\,|\varphi_0\rangle \tag{1}$$

encoded from $\vec{x}$. The PQC is naturally non-linear in the parameters. For example, the encoding process

$$|\psi(x)\rangle = \exp(-i\frac{x}{2}X)\,|0\rangle$$

from x to $|\psi(x)\rangle$ is non-linear. Moreover we can compute the gradient of each component of $\vec{x}$ efficiently. This is very important, since we need the gradient of the input in each layer when training the QNNL. The encoding step is the analog to the classical activation step.

After encoding the input data, we apply a linear transform as the analog of linear transform in the classical DNNs. This part is natural on quantum computers, because all quantum gates are linear. We use another PQC $V(\overrightarrow{W})$ with parameters $\overrightarrow{W}$ for this step. We assume that the number of parameters in $V(\overrightarrow{W})$ is $O(\mathrm{poly}(n))$.

Finally, the output of a QNNL will be computed as follow. We choose m of fixed Hamiltonians H_j (which means we will not change them during training), $j = 1, \ldots, m$, and output

$$\vec{y} = \begin{pmatrix} y_1 + b_1 \\ \vdots \\ y_m + b_m \end{pmatrix}, \quad y_j = \langle\psi(\vec{x})|\, V^\dagger(\overrightarrow{W}) H_j V(\overrightarrow{W})\, |\psi(\vec{x})\rangle\,, b_j \in \mathbb{R}. \tag{2}$$

Here, the bias term $\vec{b} = (b_1, \ldots, b_m)$ is an analog of bias in classical DNNs. Also, each y_j is a hybrid quantum-classical scheme with PQC U and Hamiltonian $V^\dagger(\overrightarrow{W})H_jV(\overrightarrow{W})$.

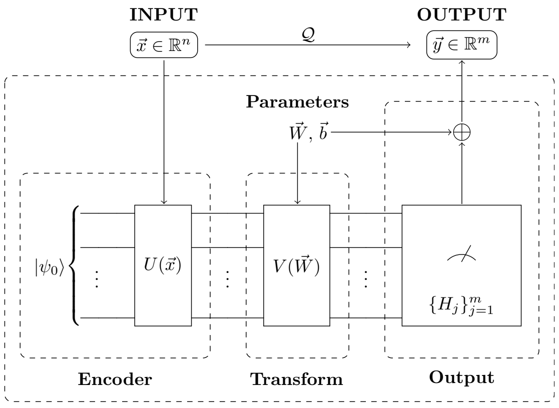


Figure 2: The structure of a QNNL $\mathcal{Q}$

To compute the output efficiently, we assume that the expectation value of each of these Hamiltonians can be computed in $O(\text{poly}(n, \frac{1}{\varepsilon}))$, where ε is the precision. It is easy to show that all Hamiltonianss of the following form satisfy this assumption

$$H = \sum_{i=1}^{O(\text{poly}(n))} H_i,$$

where H_i are tensor products of Pauli matrices or k-local Hamiltonians.

In summary, a *QNNL* is a function

$$\mathcal{Q}_{\overrightarrow{W},\vec{b}}(\vec{x}) : \mathbb{R}^n \to \mathbb{R}^m$$

defined by (1) and (2), and shown in Figure 2. Note that a QNNL is a function with classic input and output, and can be determined by a tuple

$$\mathcal{Q} = (U, V, [H_j]_{j=1,\ldots,m})$$

with parameters $(\overrightarrow{W}, \vec{b})$. Notice that the QNNLs activate before affine transforms while classical DNNLs activate after affine transforms. But this difference can be ignored when we consider multi-layers.

Since the input and output of QNNLs are classical, these QNNLs can be naturally embedded in classical DNNs. A DNN consists of the composition of multiple compatible QNNLs and classical DNN layers is called *quantum DNN (QDNN)*:

$$QDNN = \mathcal{L}_{1,\overrightarrow{W}_1,\vec{b_1}} \circ \cdots \circ \mathcal{L}_{l,\overrightarrow{W}_l,\vec{b_l}}$$

where $\mathcal{L}_{i,\overrightarrow{W_i},\vec{b_i}}$ is a classical or quantum layer from $\mathbb{R}^{n_{i-1}}$ to $\mathbb{R}^{n_i}$ for $i = 1, \ldots, l$ and $\{\overrightarrow{W_i}, \vec{b_i}, i = 1, \ldots, l\}$ are the parameters of the QDNN.

3.2 Training algorithms of QDNNs

We will use gradient descent to update the parameters. In classical DNNs, the gradient of parameters in each layer is computed by the backpropagation algorithm (BP). Suppose that we have a QDNN. Consider a QNNL $\mathcal{Q}$ with parameters $\vec{W}, \vec{b}$ whose input is $\vec{x}$, output is $\vec{y}$. Refer to (1) and (2) for details. To use the BP algorithm, we need to compute $\frac{\partial \vec{y}}{\partial \vec{W}}$, $\frac{\partial \vec{y}}{\partial \vec{b}}$ and $\frac{\partial \vec{y}}{\partial \vec{x}}$.

$\frac{\partial \vec{y}}{\partial \vec{b}}$ is trivial. And because U, V are PQCs and each component of $\vec{y}$ is an output of a hybrid quantum-classical scheme, both $\frac{\partial \vec{y}}{\partial \vec{W}}$ and $\frac{\partial \vec{y}}{\partial \vec{x}}$ can be estimated by the algorithm in section 2.2. Hence, QDNNs can be trained with the BP algorithm.

4 Representation power of QDNNs

In this section, we will show that QDNNs have more representation power than that of classical DNNs.

According to the definition of QNNLs in (2), each element of the outputs in a QNNL is of the form

$$y_j = b_j + \langle\psi_0| U^\dagger(\vec{x}) V^\dagger(\overrightarrow{W}) H_j V(\overrightarrow{W}) U(\vec{x}) |\psi_0\rangle . \tag{3}$$

In general, estimation of $\langle\psi_0| U^\dagger(\vec{x}) V^\dagger(\overrightarrow{W}) H_j V(\overrightarrow{W}) U(\vec{x}) |\psi_0\rangle$ on a classical computer will be difficult by the following theorem.

Theorem 1. *Estimation (3) with precision $c < \frac{1}{3}$ is* BQP*-hard, where* BQP *is the bounded-error quantum polynomial time complexity class.*

Proof. Consider any language $L \in \mathsf{BQP}$. There exists a polynomial-time Turing machine which takes $x \in \{0,1\}^n$ as input and outputs a polynomial-sized quantum circuit $C(x)$. Moreover, $x \in L$ if and only if the measurement result of $C(x)|0\rangle$ of the first qubit has the probability $\geq \frac{2}{3}$ to be $|1\rangle$.

Because $\{R_x, R_y, R_z, \text{CNOT}\}$ are universal quantum gates, $C(x)$ can be expressed as a polynomial-sized PQC: $U_x(\vec{\theta}) = C(x)$ with proper parameters. Consider $H = Z \otimes I \otimes \cdots \otimes I$, then

$$\langle 0| U_x(\vec{\theta}) H U_x(\vec{\theta}) |0\rangle \leq -\frac{1}{3} \tag{4}$$

if and only if $x \in L$, and

$$\langle 0| U_x(\vec{\theta}) H U_x(\vec{\theta}) |0\rangle \geq \frac{1}{3} \tag{5}$$

if and only if $x \notin L$. $\square$

As it is generally believed that quantum computers can not be simulated efficiently by classical computers, according to Theorem 1, there exist QNNLs which cannot be classically simulated efficiently. Hence, this QNNLs can represent functions which cannot be represented by classical DNNs with polynomial units. Thus, adding QNNLs to DNNs will enhance the representation power of DNNs.

5 Numerical experiments

In this section, we will use QDNN to conduct a numerical experiment for an image classification task. The data comes from the MNIST data set. We built a QDNN with 3 QNNLs. The goal of this QDNN is to recognize the digit in the image is either 0 or 1 as a classifier.

We uses the Julia package Yao.jl as a quantum simulator [26] in our experiments. All data were collected on a desktop PC with Intel CPU i7-4790 and 4GB RAM.

5.1 Details of the model

The data in the MNIST is $28 \times 28 = 784$ dimensional images. This dimension is too large for the current quantum simulator. Hence, we first resize the image to 8×8 pixels. We use three QNNLs in our QDNN, which will be called the input layer, the hidden layer, and the output layer, respectively. Each layer is made of 3 parts: encoder, transform, and output.

5.1.1 Input layer

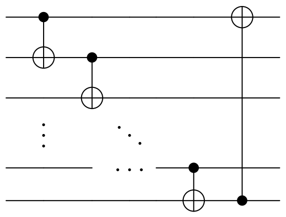
The input layer uses an 8-qubit circuit. The encoder will accept an input vector $x \in \mathbb{R}^{64}$ and use a PQC to map it to a quantum state $|\psi(\vec{x})\rangle = U_{\text{in}}(\vec{x})\,|0\rangle$. The structure of the PQC is like

$$\begin{aligned} U_{\text{in}} = &\, \mathit{Ent} \circ R_Z(\vec{x}_{57:64}) \circ R_X(\vec{x}_{49:56}) \circ R_Z(\vec{x}_{41:48}) \\ & \circ \mathit{Ent} \circ R_Z(\vec{x}_{33:40}) \circ R_X(\vec{x}_{25:32}) \circ R_Z(\vec{x}_{17:24}) \\ & \circ \mathit{Ent} \circ R_Z(\vec{x}_{9:16}) \circ R_X(\vec{x}_{1:8}) \end{aligned}$$

where

$$R_H(\vec{x}_{a:b}) = R_H(x_a) \otimes R_H(x_{a+1}) \otimes \cdots \otimes R_H(x_b), \; H = X, Z. \tag{6}$$

Ent is a CNOT circuit which introduces entanglement to the circuit. In our experiment, *Ent* is of the following form.



The transform in the input layer is similar to the encoder. The structure of the transform is like

$$V_{\text{in}} = (R_Z \circ R_X \circ R_Z \circ \mathit{Ent})^6 \circ R_Z \circ R_X. \tag{7}$$

Also, the transform parameters $\overrightarrow{W}_{\text{in}}$ will be input to V_{in} as the same way as $\vec{x}$ were input to U_{in}.

The output of the input layer is of the form

$$\vec{h}_1 = \begin{pmatrix} \langle\psi(\vec{x},\overrightarrow{W}_{\text{in}})|\, H_{1,X}\, |\psi(\vec{x},\overrightarrow{W}_{\text{in}})\rangle \\ \vdots \\ \langle\psi(\vec{x},\overrightarrow{W}_{\text{in}})|\, H_{8,X}\, |\psi(\vec{x},\overrightarrow{W}_{\text{in}})\rangle \\ \langle\psi(\vec{x},\overrightarrow{W}_{\text{in}})|\, H_{1,Y}\, |\psi(\vec{x},\overrightarrow{W}_{\text{in}})\rangle \\ \vdots \\ \langle\psi(\vec{x},\overrightarrow{W}_{\text{in}})|\, H_{8,Y}\, |\psi(\vec{x},\overrightarrow{W}_{\text{in}})\rangle \\ \langle\psi(\vec{x},\overrightarrow{W}_{\text{in}})|\, H_{1,Z}\, |\psi(\vec{x},\overrightarrow{W}_{\text{in}})\rangle \\ \vdots \\ \langle\psi(\vec{x},\overrightarrow{W}_{\text{in}})|\, H_{8,Z}\, |\psi(\vec{x},\overrightarrow{W}_{\text{in}})\rangle \end{pmatrix} + \vec{b}_{\text{in}} \in \mathbb{R}^{24}, \tag{8}$$

where $|\psi(\vec{x},\overrightarrow{W}_{\text{in}})\rangle = V_{\text{in}}(\overrightarrow{W}_{\text{in}})U_{\text{in}}(\vec{x})\,|0\rangle$ and $H_{j,M}$ denotes the result obtained by applying the operator M on the j-th qubit for $M \in \{X,Y,Z\}$.

5.1.2 Hidden layer

The hidden layer uses 6 qubits. The structure of the hidden layer is almost the same as the input layer, but with less qubit gates.

The encoder is of the form

$$U_h = Ent \circ R_X \circ R_Z \circ Ent \circ R_Z \circ R_X. \tag{9}$$

The transform is

$$V_h = R_X \circ R_Z \circ Ent \circ (R_Z \circ R_X \circ R_Z \circ Ent)^4 \circ R_Z \circ R_X. \tag{10}$$

The output of the hidden layer is

$$\vec{h_2} = \begin{pmatrix} \langle\psi(\vec{h}_1,\overrightarrow{W_h})|\, H_{1,Y}\, |\psi(\vec{h}_1,\overrightarrow{W_h})\rangle \\ \vdots \\ \langle\psi(\vec{h}_1,\overrightarrow{W_h})|\, H_{6,Y}\, |\psi(\vec{h}_1,\overrightarrow{W_h})\rangle \\ \langle\psi(\vec{h}_1,\overrightarrow{W_h})|\, H_{1,Z}\, |\psi(\vec{h}_1,\overrightarrow{W_h})\rangle \\ \vdots \\ \langle\psi(\vec{h}_1,\overrightarrow{W_h})|\, H_{6,Z}\, |\psi(\vec{h}_1,\overrightarrow{W_h})\rangle \end{pmatrix} + \vec{b_h} \in \mathbb{R}^{12}. \tag{11}$$

5.1.3 Output layer

The output layer uses 4 qubits. The structure of the output layer is also similar to the input layer. The only difference is that in the output is classification result.

The encoder is

$$U_{\text{out}} = R_Z \circ Ent \circ R_Z \circ R_X. \tag{12}$$

The transform is like

$$V_{\text{out}} = R_X \circ R_Z \circ Ent \circ R_Z \circ R_X \circ R_Z \circ Ent \circ R_Z \circ R_X. \tag{13}$$

The output of the output layer is

$$\vec{y} = \begin{pmatrix} \langle\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})| \left(|0\rangle\langle 0| \otimes I \otimes I \otimes I\right) |\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})\rangle \\ \langle\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})| \left(|1\rangle\langle 1| \otimes I \otimes I \otimes I\right) |\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})\rangle \,. \end{pmatrix} \tag{14}$$

We do not add bias term here, and it will output a vector in $\mathbb{R}^2$. Moreover, if the input $\vec{x}$ is from an image of digit 0, the output $\vec{y}$ should be close to $|0\rangle$, otherwise it should be close to $|1\rangle$ after training.

In conclusion, the settings of these three layers are shown in table 1.

	# of qubits	Input dimension	Output dimension	# of parameters (transform + bias)
Input layer	8	64	24	160 + 24
Hidden layer	6	24	12	96 + 12
Output layer	4	12	2	28 + 0

Table 1: Settings of three layers

Finally, the loss function is defined as

$$L = \frac{1}{|\mathcal{D}|} \sum_{(\vec{x},y)\in\mathcal{D}} \left|\text{DNN}(\vec{x}) - |y\rangle\right|^2, \tag{15}$$

where $\mathcal{D}$ is the training set.

5.2 Experiment results

All parameters were initialized randomly in $(-\pi, \pi)$. We use Adam optimizer [27] to update parameters. We train this QDNN for 400 iterations with batch size of 240. In the first 200 of iterations, the hyper parameters of Adam is set to be $\eta = 0.01, \beta_1 = 0.9, \beta_2 = 0.999$. In the later 200 of iterations, we change η to 0.001.

The values of the loss function on the training set and test set during training is shown in Figure 3. The accurate rate of this QDNN on the test set rise to 99.15% after training.

6 Discussion

We introduced the model of QNNL and built QDNN with QNNLs. We proved that QDNNs have more representation power than classical DNNs. We presented a practical gradient-based training algorithm as the analog of BP algorithms. Because the model is based on hybrid quantum-classical scheme, it can be realized on NISQ processors. As a result, the QDNN has more representation powers than classical DNNs and still keeps most of the advantages of the classical DNNs.

Due to the limited power the classical simulator for quantum computation, only QDNNs with a small number of qubits can be used in practice. As a consequence, we only trained a model for a simple task in our experiment.

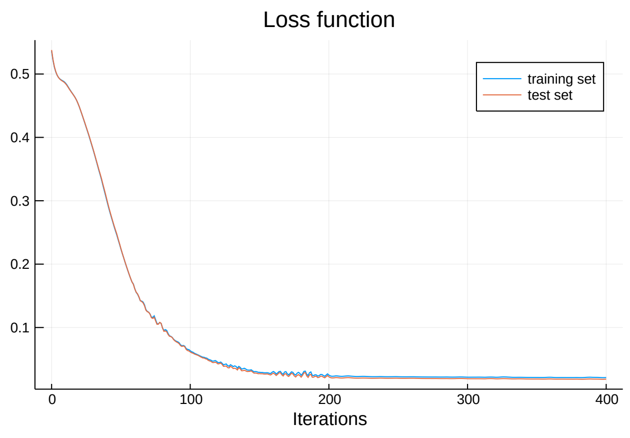


Figure 3: Loss function

If we have more quantum resources in the future, we can access exponential dimensional feature Hilbert spaces [13] with QDNNs and only uses polynomial size of parameters. Hence, we believe that QDNNs will help us to extract features more efficiently in exponential dimensional feature Hilbert spaces. This idea is similar to ideas of kernel methods [28, 29].

References

[1] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. Ieee, 1994.

[2] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[3] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[4] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Physical review letters*, 109(5):050505, 2012.

[5] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.

[6] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[7] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[8] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631, 2014.

[9] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[10] Xun Gao, Zhengyu Zhang, and Luming Duan. A quantum machine learning algorithm based on generative models. *Science advances*, 4(12):eaat9004, 2018.

[11] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical review letters*, 121(4):040502, 2018.

[12] Jin-Guo Liu and Lei Wang. Differentiable learning of quantum circuit born machines. *Physical Review A*, 98(6):062324, 2018.

[13] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Physical review letters*, 122(4):040504, 2019.

[14] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209, 2019.

[15] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[17] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[18] Richard Socher, Yoshua Bengio, and Christopher D Manning. Deep learning for nlp (without magic). In *Tutorial Abstracts of ACL 2012*, pages 5–5. Association for Computational Linguistics, 2012.

[19] Nathan Killoran, Thomas R Bromley, Juan Miguel Arrazola, Maria Schuld, Nicolás Quesada, and Seth Lloyd. Continuous-variable quantum neural networks. *Physical Review Research*, 1(3):033063, 2019.

[20] Jian Zhao, Yuan-Hang Zhang, Chang-Peng Shao, Yu-Chun Wu, Guang-Can Guo, and Guo-Ping Guo. Building quantum neural networks based on a swap test. *Phys. Rev. A*, 100:012334, Jul 2019.

[21] Iordanis Kerenidis, Jonas Landman, and Anupam Prakash. Quantum algorithms for deep convolutional neural networks. In *International Conference on Learning Representations*, 2020.

[22] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[23] Jin-Guo Liu, Yi-Hong Zhang, Yuan Wan, and Lei Wang. Variational quantum eigensolver with fewer qubits. *Phys. Rev. Research*, 1:023025, Sep 2019.

[24] Jun Li, Xiaodong Yang, Xinhua Peng, and Chang-Pu Sun. Hybrid quantum-classical approach to quantum optimal control. *Physical review letters*, 118(15):150503, 2017.

[25] Ken M Nakanishi, Keisuke Fujii, and Synge Todo. Sequential minimal optimization for quantum-classical hybrid algorithms. *arXiv preprint arXiv:1903.12166*, 2019.

[26] Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang. Yao.jl: Extensible, efficient framework for quantum algorithm design. *arXiv preprint arXiv:1912.10877*, 2019.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[29] Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. *Journal of machine learning research*, 3(Feb):1083–1106, 2003.

A Gradient Estimation of PQCs

Without the loss of generality, suppose that a PQC has the form

$$U(\vec{\theta}) = U_j(\theta_j)\dots U_1(\theta_1).$$

Given a Hamiltonian M, the expection value is defined by the following equation,

$$L(\vec{\theta}) = \langle\psi_0| U^\dagger(\vec{\theta}) M U(\vec{\theta}) |\psi_0\rangle .$$

The goal in hybrid quantum-classical computing is usually optimizing L. We can use gradient descent for this problem. Thus, we need estimate

$$\nabla_{\vec{\theta}} L = \begin{pmatrix} \frac{\partial L}{\partial \theta_1} \\ \vdots \\ \frac{\partial L}{\partial \theta_l} \end{pmatrix}.$$

Notice that $U_j(\theta_j)$ has the form $U_j(\theta_j) = \exp(i\frac{\theta_j}{2}H_j)$, where $H_j^2 = I$. Hence, we have

$$U_j(\theta_j) = \cos(\frac{\theta_j}{2})I - i\sin(\frac{\theta_j}{2})H_j.$$

We denote

$$U_{a:b} = U_b(\theta_b)\dots U_a(\theta_a),\ a<b,$$

and

$$|\psi_a\rangle = U_{1:a}|\psi_0\rangle, \quad M_b = U_{b:l}^\dagger M U_{b:l}$$

By calculus,

$$\begin{aligned}\frac{\partial L}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j}\langle\psi_0| U^\dagger(\vec{\theta}) M U(\vec{\theta}) |\psi_0\rangle \\ &= \langle\psi_{j-1}| \frac{\partial U_j^\dagger(\theta_j)}{\partial\theta_j} M_{j+1} U_j(\theta_j) |\psi_{j-1}\rangle + \langle\psi_{j-1}| U^\dagger(\theta_j) M_{j+1} \frac{\partial U_j(\theta_j)}{\partial \theta_j} |\psi_{j-1}\rangle \\ &= \frac{1}{2}\langle\psi_{j-1}| \big(-\sin\frac{\theta_j}{2}I + i\cos\frac{\theta_j}{2}H_j\big) M_{j+1} \big(\cos\frac{\theta_j}{2}I - i\sin\frac{\theta_j}{2}H_j\big)|\psi_{j-1}\rangle \\ &\quad + \frac{1}{2}\langle\psi_{j-1}| \big(\cos\frac{\theta_j}{2}I + i\sin\frac{\theta_j}{2}H_j\big) M_{j+1} \big(-\sin\frac{\theta_j}{2}I - i\cos\frac{\theta_j}{2}H_j\big)|\psi_{j-1}\rangle \\ &= -\cos\frac{\theta_j}{2}\sin\frac{\theta_j}{2}\langle\psi_{j-1}| M_{j+1} |\psi_{j-1}\rangle + \cos\frac{\theta_j}{2}\sin\frac{\theta_j}{2}\langle\psi_{j-1}| H_j M_{j+1} H_j |\psi_{j-1}\rangle \\ &\quad + \frac{i}{2}\big(\cos^2\frac{\theta_j}{2} - \sin^2\frac{\theta_j}{2}\big)\langle\psi_{j-1}| H_j M_{j+1} - M_{j+1} H_j |\psi_{j-1}\rangle .\end{aligned}$$

Denote

$$\begin{aligned} L_{j,+}(\vec{\theta}) &= L(\theta_1,\dots,\theta_{j-1},\theta_j+\frac{\pi}{2},\theta_{j+1},\dots,\theta_l), \\ L_{j,-}(\vec{\theta}) &= L(\theta_1,\dots,\theta_{j-1},\theta_j-\frac{\pi}{2},\theta_{j+1},\dots,\theta_l). \end{aligned}$$

That is, we shift the j-th parameter. Then

$$\begin{aligned}
L_{+,j}(\vec{\theta}) = \langle\psi_{j-1}| & \big[\cos(\frac{\theta_j}{2}+\frac{\pi}{4})I + i\sin(\frac{\theta_j}{2}+\frac{\pi}{4})\big] M_{j+1} \\
& \big[\cos(\frac{\theta_j}{2}+\frac{\pi}{4})I - i\sin(\frac{\theta_j}{2}+\frac{\pi}{4})\big] |\psi_{j-1}\rangle \\
= & \frac{1}{2}\big(\cos\frac{\theta_j}{2} - \sin\frac{\theta_j}{2}\big)^2 \langle\psi_{j-1}| M_{j+1} |\psi_{j-1}\rangle \\
& + \frac{1}{2}\big(\cos\frac{\theta_j}{2} + \sin\frac{\theta_j}{2}\big)^2 \langle\psi_{j-1}| H_j M_{j+1} H_j |\psi_{j-1}\rangle \\
& + \frac{i}{2}\big(\cos^2\frac{\theta_j}{2} - \sin^2\frac{\theta_j}{2}\big) \langle\psi_{j-1}| H_j M_{j+1} - M_{j+1} H_j |\psi_{j-1}\rangle .
\end{aligned}$$

By similar computation,

$$\begin{aligned}
L_{-,j}(\vec{\theta}) = & \frac{1}{2}\big(\cos\frac{\theta_j}{2} + \sin\frac{\theta_j}{2}\big)^2 \langle\psi_{j-1}| M_{j+1} |\psi_{j-1}\rangle \\
& + \frac{1}{2}\big(\cos\frac{\theta_j}{2} - \sin\frac{\theta_j}{2}\big)^2 \langle\psi_{j-1}| H_j M_{j+1} H_j |\psi_{j-1}\rangle \\
& - \frac{i}{2}\big(\cos^2\frac{\theta_j}{2} - \sin^2\frac{\theta_j}{2}\big) \langle\psi_{j-1}| H_j M_{j+1} - M_{j+1} H_j |\psi_{j-1}\rangle .
\end{aligned}$$

Thus, one can simply check that

$$\frac{\partial L}{\partial \theta_j} = \frac{1}{2}\left[L_{j,+}(\vec{\theta}) - L_{j,-}(\vec{\theta})\right].$$

In conclusion, we can estimate gradient of parameters of a PQC by shifting parameters and runing the same circuit.