Quantum Machine Intelligence (2021) 3:15
https://doi.org/10.1007/s42484-021-00046-w

RESEARCH ARTICLE

QDNN: deep neural networks with quantum layers

Chen Zhao[1,2] · **Xiao-Shan Gao**[1,2]

Received: 23 November 2020 / Accepted: 19 March 2021

Abstract
In this paper, a quantum extension of classical deep neural network (DNN) is introduced, which is called QDNN and consists of quantum structured layers. It is proved that the QDNN can uniformly approximate any continuous function and has more representation power than the classical DNN. Moreover, the QDNN still keeps the advantages of the classical DNN such as the non-linear activation, the multi-layer structure, and the efficient backpropagation training algorithm. Furthermore, the QDNN uses parameterized quantum circuits (PQCs) as the basic building blocks and hence can be used on near-term noisy intermediate-scale quantum (NISQ) processors. A numerical experiment for an image classification task based on QDNN is given, where a high accuracy rate is achieved.

Keywords Deep neural networks · Quantum machine learning · Hybrid quantum-classical algorithm · NISQ

1 Introduction

Quantum computers use the principles of quantum mechanics for computing, which are more powerful than classical computers in many computing problems (Shor 1994; Grover 1996). Many quantum machine learning algorithms, such as quantum support vector machine, quantum principal component analysis, and quantum Boltzmann machine, were developed (Wiebe et al. 2012; Schuld et al. 2015a; Biamonte et al. 2017; Rebentrost et al. 2014; Lloyd et al. 2014; Amin et al. 2018; Gao et al. 2018), and these algorithms were shown to be more efficient than their classical versions.

In recent years, DNNs (LeCun et al. 2015) became the most important and powerful method in machine learning, which were widely applied in computer vision (Voulodimos et al. 2018), natural language processing (Socher et al. 2012), and many other fields. The basic unit of DNN is the perceptron, which is an affine transformation together with an activation function. The non-linearity of the activation function and the depth give the DNN much representation power (Hornik 1991; Leshno et al. 1993). Approaches were proposed to build classical DNNs on quantum computers (Killoran et al. 2019; Zhao et al. 2019; Kerenidis et al. 2020). They achieved quantum speed-up under certain assumptions. But the structure of classical DNNs is still used, and only some local operations are speeded up by quantum algorithms. For instance, the inner product was speedup using the swap test (Zhao et al. 2019).

Several quantum analogs of DNNs were proposed. In Schuld et al. (2015b) and Cao et al. (2017), quantum analogs of the perceptron are demonstrated. However, methods of building complex DNNs with these quantum perceptrons are not developed. In Wan et al. (2017), a quantum generalization of the feedforward neural network is proposed. However, non-linearity is not introduced in this model. In Tacchino et al. (2020), the non-linearity is introduced with measurement, as a consequence, the training cost is increased. In Steinbrecher et al. (2019), a quantum analog of the DNN which can be run on optical quantum devices was proposed. A quantum analog of deep convolutional neural networks was proposed in Li et al. (2020). In many of these approaches, the inputs or outputs are quantum states, and hence the quantum random access memory (QRAM) (Giovannetti et al. 2008; Aaronson 2015) is used. However, QRAM is difficult to be implemented on noisy intermediate-scale quantum (NISQ) (Preskill 2018) devices. It is known that NISQ will be the only quantum devices that can be used in the near-term, where only a limited number of qubits without error-correcting can be used.

✉ Xiao-Shan Gao
xgao@mmrc.iss.ac.cn

[1] Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China

[2] University of Chinese Academy of Sciences, Beijing 100049, China

Published online: 26 April 2021

Recently, parameterized quantum circuits (PQCs) were widely considered, because PQCs can be efficiently implemented on NISQ devices. Several NISQ quantum machine learning models based on PQCs, such as quantum generative adversarial networks, quantum circuit Born machine, and quantum kernel methods, were proposed (Lloyd and Weedbrook 2018; Dallaire-Demers and Killoran 2018; Liu and Wang 2018; Schuld and Killoran 2019; Havlíček et al. 2019; Benedetti et al. 2019). Several approaches are shown that PQCs have the potential abilities in machine learning tasks including approximating functions (Mitarai et al. 2018), classification (Farhi and Neven 2018; Schuld et al. 2020), and data generating (Liu and Wang 2018; Situ et al. 2020). PQCs are also called quantum neural networks (QNNs) because of its layerwise circuit structure, and QNNs are used for machine learning tasks (Farhi and Neven 2018; Beer et al. 2020). Several different structures of the PQC were proposed (Grant et al. 2018; Liu et al. 2019a; Cong et al. 2019).

In this paper, we introduce the quantum deep neural network (QDNN) which is a composition of multiple quantum neural network layers (QNNLs). We prove that the QDNN can uniformly approximate any continuous function and has more representation power than the classical DNN. Unlike other approaches of quantum analogs of DNNs, our QDNN 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 inputs and the outputs of the QDNN are both classical which makes the QDNN more practical. Because the QNNL is based on PQCs, the QDNN has the potential to be used on NISQ processors. As shown in our experiments, a QDNN with a small number (eight) of qubits can be used in image classification. In summary, QDNN provides a new class of neural networks which can be used in near-term quantum computers and is more powerful than classical DNNs.

The structure of the QNNL is similar to that of the QNN. There usually exists only one Hamiltonian in the model of QNN while there exist multiple Hamiltonians in the QNNL and a bias term will be added at the output. The multiple dimensional output of the QNNL makes it possible to build multi-layer structure in QDNN. We use QNNLs as building blocks of the QDNN and we use multiple PQCs which will be trained simultaneously. The universal approximation property of QDNN comes from its multi-layer structure. The QNN can be regarded as a special 1-layer QDNN and has no universal approximation property.

Our paper is organized as follows. We define the QNNL in Section 2.1. The definition of the QDNN and its training algorithms are presented in Section 2.2. In Section 3, we discuss the representation power and potential quantum advantages of the QDNN. In Section 4, a numerical experiment for an image classification task based on QDNN is used to show the effectiveness of QDNN.

2 The 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 transformation 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 transformations.

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

2.1 Quantum neural network layers

We build QNNLs using the hybrid quantum-classical algorithm scheme (McClean et al. 2016), which is widely used in many NISQ quantum algorithms (Liu and Wang 2018; Liu et al. 2019b). As shown in Fig. 1, a hybrid quantum-classical algorithm scheme consists of a quantum part and a classical part. In the quantum part, parameterized quantum circuits (PQCs) are used to prepare quantum states with quantum processors. In the classical part, parameters of the PQCs are optimized using classical computers.

A PQC is a quantum circuit with parametric gates, which 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, when H_j is one of Pauli matrices $\{X, Y, Z\}$, $U_j(\theta_j)$ is the single qubit rotation gates R_X, R_Y, R_Z.

As shown in Fig. 1, once fixed an ansatz circuit $U(\vec{\theta})$ and a Hamiltonian H, we can define the loss function of

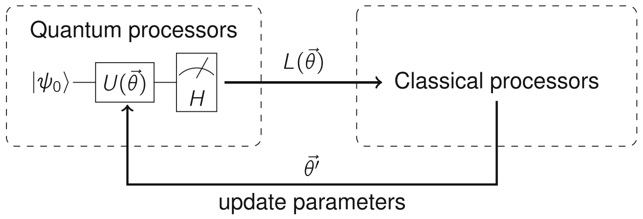

Fig. 1 Hybrid quantum-classical scheme

the form $L = \langle 0|U^\dagger(\vec{\theta})HU(\vec{\theta})|0\rangle$. Then, we can optimize L by updating parameters $\vec{\theta}$ using optimization algorithms (Schuld et al. 2019; Nakanishi et al. 2019). With gradient-based algorithms (Schuld et al. 2019), one can efficiently compute the gradient information $\frac{\partial L}{\partial \theta}$ which is essentially important in our model. Hence, we will focus on gradient-based algorithms in this paper.

Now, we are going to define a QNNL, which consists of 3 parts: the encoder, the transformation, and the output, as shown in Fig. 2.

(1) **The encoder.** For a classical input data $\vec{x} \in \mathbb{R}^n$, we introduce non-linearity to our QNNL by encoding the input $\vec{x}$ to a quantum state $|\psi(\vec{x})\rangle$ non-linearly. Precisely, we choose a PQC $U(\vec{x})$ with at most $O(n)$ qubits and apply it to an initial state $|\psi_0\rangle$ to obtain a quantum state

$$|\psi(\vec{x})\rangle = U(\vec{x})|\psi_0\rangle \tag{1}$$

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

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

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

(2) **The transformation.** After encoding the input data, we apply a linear transformation as the analog of the linear transformation 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 purpose. We assume that the number of parameters in $V(\overrightarrow{W})$ is $O(\text{poly}(n))$.

(3) **The output.** Finally, the output of a QNNL will be computed as follow. We choose m fixed Hamiltonians H_j, $j = 1, \ldots, m$, and output

$$\vec{y} = \begin{pmatrix} y_1 + b_1 \\ \vdots \\ y_m + b_m \end{pmatrix},$$

$$y_j = \langle \psi(\vec{x})|V^\dagger(\overrightarrow{W})H_jV(\overrightarrow{W})|\psi(\vec{x})\rangle, b_j \in \mathbb{R}. \tag{2}$$

Note that the expectation value of a Hamiltonian is a linear function of the density matrix. 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 a PQC $U(\vec{x})V(\overrightarrow{W})$ and Hamiltonians H_j.

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 Hamiltonians 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 Eqs. 1) and 2, and shown in Fig. 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 transformations while classical DNNLs activate after affine transformations. But this difference can be ignored when considering multi-layers.

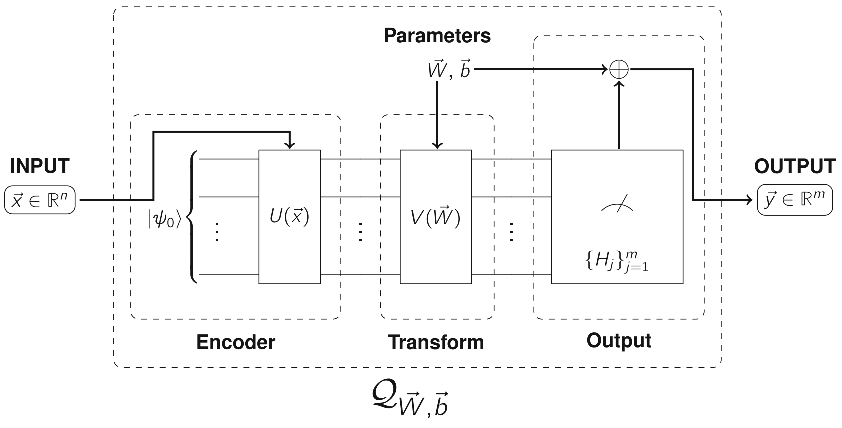

Fig. 2 The structure of a QNNL $\mathcal{Q}_{\vec{W},\vec{b}}$

2.2 QDNN and its training algorithms

Since the input and output of QNNLs are classical values, they can be implemented without the assumption of QRAM. Furthermore, the QNNLs can be naturally embedded in classical DNNs. A neural network consists of the composition of multiple compatible QNNLs and classical DNN layers is called *quantum DNN (QDNN)*:

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

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

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 $\vec{x}$ and output is $\vec{y}$. Refer to Eqs. 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 $\frac{\partial \vec{y}}{\partial \vec{x}}$. Computing $\frac{\partial \vec{y}}{\partial \vec{b}}$ is trivial. 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 shifting parameters (Schuld et al. 2019).

Algorithm 1 Gradient estimation for PQCs.

```
Input: A PQC $U(\vec{\theta})$, an initial state $|\varphi_0\rangle$, a Hamiltonian
    $H$, and an initial value $\vec{x} = (x_1, \ldots, x_m)$
Output: The gradient $\frac{\partial}{\partial x_j} \langle\varphi_0| U^\dagger(\vec{x}) H U(\vec{x}) |\varphi_0\rangle$
Set $x_j := x_j + \frac{\pi}{2}$ for each $j$;
Estimate $\langle H_{j,+}\rangle = \langle\varphi_0| U^\dagger(\vec{x}) H U(\vec{x}) |\varphi_0\rangle$;
Set $x_j := x_j - \pi$ for each $j$;
Estimate $\langle H_{j,-}\rangle = \langle\varphi_0| U^\dagger(\vec{x}) H U(\vec{x}) |\varphi_0\rangle$;
return $\frac{1}{2}\left[\langle H_{j,+}\rangle - \langle H_{j,-}\rangle\right]$
```

We can use Algorithm 2 to estimate the gradient in each quantum layer.

Hence, gradients can be backpropagated through the quantum layer, and QDNNs can be trained with the BP algorithm.

Gradient-based methods are used to train QDNNs. As a consequence, if the circuits in the QNNL reach unitary 2-design, then there exist barren plateaus, which makes the model untrainable (McClean et al. 2018). Thus, the structure of QNNLs should not be randomly chosen. There are several techniques to avoid barren plateaus. For instance, we can use circuits with special structures and local Hamiltonians (Cerezo et al. 2020; Zhao and Gao 2021), or we can use certain initialization strategies and introduce correlations between parameters to obtain large gradients (Grant et al. 2019; Volkoff and Coles 2021).

Algorithm 2 Gradient estimation for QNNLs.

```
Input: A QNNL $\mathcal{Q}(\vec{x}, \vec{W}, \vec{b})$, the current value of
    $\vec{x}, \vec{W}, \vec{b}$ and $\vec{y} = \mathcal{Q}(\vec{x}, \vec{W}, \vec{b})$, the gradient of
    the output $\frac{\partial L}{\partial \vec{y}}$
Output: The gradient $\frac{\partial L}{\partial \vec{x}}, \frac{\partial L}{\partial \vec{W}}, \frac{\partial L}{\partial \vec{b}}$
Initialize $\frac{\partial \vec{y}}{\partial \vec{x}}$ and $\frac{\partial \vec{y}}{\partial \vec{W}}$;
for $x_j$ in $\vec{x}$ do
    Estimate $\frac{\partial \vec{y}}{\partial x_j}$ with Algorithm 1;
    Set the $j$-th component of $\frac{\partial \vec{y}}{\partial \vec{x}}$ to $\frac{\partial \vec{y}}{\partial x_j}$;
end
for $w_k$ in $\vec{W}$ do
    Estimate $\frac{\partial \vec{y}}{\partial w_k}$ with Algorithm 1;
    Set the $k$-th component of $\frac{\partial \vec{y}}{\partial \vec{W}}$ to $\frac{\partial \vec{y}}{\partial w_k}$;
end
Set $\frac{\partial L}{\partial \vec{x}} := \frac{\partial L}{\partial \vec{y}} \frac{\partial \vec{y}}{\partial \vec{x}}$;
Set $\frac{\partial L}{\partial \vec{W}} := \frac{\partial L}{\partial \vec{y}} \frac{\partial \vec{y}}{\partial \vec{W}}$;
Set $\frac{\partial L}{\partial \vec{b}} := \frac{\partial L}{\partial \vec{y}}$;
return $\frac{\partial L}{\partial \vec{x}}, \frac{\partial L}{\partial \vec{W}}, \frac{\partial L}{\partial \vec{b}}$;
```

3 Representation power of QDNNs

In this section, we will consider the representation power of the QDNN. We will show that QDNN can approximate any continuous function similar to the classical DNN. Moreover, if quantum computing can not be classically simulated efficiently, the QDNN has more representation power than the classical DNN.

3.1 Universal approximation property of QDNNs

The universal approximation theorem ensures that DNNs can approximate any continuous function (Cybenko 1989; Hornik 1991; Leshno et al. 1993; Pinkus 1999). Since the class of QDNNs is an extension of the class of classical DNNs, the universal approximation theorem can be applied to the QDNN trivially. Now, we show that QDNNs with only QNNLs also have the universal approximation property. Consider two cases

– DNN with only QNNLs.
– DNN with QNNLs and affine layers.

In the first case, let us consider a special type of QNNLs which can represent monomials (Mitarai et al. 2018). Consider the circuit

$$U(x) = R_Y(2\arccos(\sqrt{x})) = \begin{pmatrix} x & -\sqrt{1-x^2} \\ \sqrt{1-x^2} & x \end{pmatrix} \quad (3)$$

and the Hamiltonian $H_0 = |0\rangle\langle 0|$. The expectation value of $\langle 0|U^\dagger(x)H_0U(x)|0\rangle$ is the monomial x for $x \in [0,1]$. For multivariable monomial $\mathbf{x} = x_1^{m_1}\cdots x_k^{m_k}$, we use the circuit

$$U(\mathbf{x}) = \left[\otimes_{j_1=1}^{m_1} R_Y(2\arccos(\sqrt{x_1}))\right] \otimes\cdots\otimes \left[\otimes_{j_k=1}^{m_k} R_Y(2\arccos(\sqrt{x_k}))\right] \tag{4}$$

and the Hamiltonian $H_\mathbf{0} = |0\ldots0\rangle\langle 0\ldots 0|$, where $\otimes$ is the tensor product and

$$\otimes_{j_k=1}^{m_k} R_Y(2\arccos(\sqrt{x_k})) = \overbrace{R_Y(2\arccos(\sqrt{x_k}))\otimes\cdots\otimes R_Y(2\arccos(\sqrt{x_k}))}^{m_k}.$$

Similarly, the expectation value of

$\langle 0\ldots0|U^\dagger(\mathbf{x})H_\mathbf{0}U(\mathbf{x})|0\ldots0\rangle$ is $x_1^{m_1}\cdots x_k^{m_k}$

for $x_1,\ldots,x_k \in [0,1]$.

With the above results and Stone-Weierstrass theorem (Stone 1948), we can deduce the following theorem.

Theorem 1 *The QDNN with only QNNLs can uniformly approximate any continuous function*

$$f : [0,1]^k \to \mathbb{R}^l.$$

Now, let us consider the second case. As the affine transformation can map the hypercube $[0,1]^k$ to $[a_1,b_1]\times\cdots\times[a_k,b_k]$ for any $a_k < b_k$. Hence we have the following result.

Corollary 1 *The QDNN with QNNLs and affine layers can uniformly approximate any continuous function*

$$f : D \to \mathbb{R}^l,$$

where D is a compact set in $\mathbb{R}^k$.

Also, the QNNL can be used as a non-linear activation function. For example, we consider a QNNL $\mathcal{Q}_{\mathrm{ac}}$ with the input circuit

$$\otimes_{j=1}^m R_Y(x_j)$$

and the Hamiltonian

$$H_j = I\otimes\cdots\otimes I\otimes|0\rangle\langle 0|\otimes I\otimes\cdots\otimes I,$$

where the projection is on the jth qubit for $j = 1,\ldots,m$. By simple computation, we have

$$\mathcal{Q}_{\mathrm{ac}}\left(\begin{pmatrix} x_1\\ \vdots \\ x_m\end{pmatrix}\right) = \begin{pmatrix}\cos(x_1)\\ \vdots\\ \cos(x_m)\end{pmatrix}. \tag{5}$$

By the universal approximation property (Leshno et al. 1993; Kratsios 2019), neural networks with non-polynomial activation functions can approximate any continuous function $f : \mathbb{R}^k \to \mathbb{R}^l$. Thus, the QDNN with QNNLs and affine layers can approximate any continuous function.

Similar to the classical case, QDNNs with one quantum layer can approximate any continuous function (Hornik 1991). However, if we restrict the number of parameters to be polynomial, then there exist functions which can be approximated with large-depth DNNs and cannot be approximated by small-depth neural networks (Eldan and Shamir 2016; Daniely 2017; Vardi and Shamir 2020), and this is the reason to use multilayer QDNNs.

3.2 Quantum advantages

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

$$y_j = b_j + \langle\psi_0|U^\dagger(\vec{x})V^\dagger(\overrightarrow{W})H_jV(\overrightarrow{W})U(\vec{x})|\psi_0\rangle. \tag{6}$$

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

Theorem 2 *Estimation (6) with precision $c < \frac{1}{3}$ is BQP-hard, where BQP is the bounded-error quantum polynomial time complexity class.*

Proof Consider any language $L \in$ 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 $\{R_X, R_Z, \mathrm{CNOT}\}$ 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

$$\langle 0|U_x(\vec{\theta})HU_x(\vec{\theta})|0\rangle \leq -\frac{1}{3} \tag{7}$$

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

$$\langle 0|U_x(\vec{\theta})HU_x(\vec{\theta})|0\rangle \geq \frac{1}{3} \tag{8}$$

if and only if $x \notin L$. □

Given inputs, computing the outputs of classical DNNs is polynomial time. Hence, functions represented by classical DNNs are characterized by the complexity class Ppoly. On the other hand, computing the outputs of QDNNs is BQP-hard in general according to Theorem 2. The functions represented by QDNNs are characterized by a complexity class that has a lower bound BQP/poly. Here, BQP/poly is the problems which can be solved by polynomial sized quantum circuits with bounded error probability (Aaronson et al. 2005). Under the hypothesis that quantum computers cannot be simulated efficiently by classical computers, which is generally believed, there exists a function represented by a QDNN which cannot be computed by classical circuits of polynomial size. Hence, QDNNs have more representation power than DNNs.

4 Experimental results

We will use QDNNs 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.

4.1 Experiment details

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.

In the experiments, we use trainable QDNNs by adopting local Hamiltonians and small depth structure. We set the ansatz circuit to be the one in Fig. 3, which is similar to the hardware efficient ansatz (Kandala et al. 2017) and has small depth. Because of the small depth of the ansatz and the local Hamiltonian, the QDNN is trainable (Cerezo et al. 2020). Also, the small depth makes the model possible to be run on NISQ devices. The hyperparameters D_T and D_E will be chosen depending on the problem to be solved.

4.1.1 Input layer

The input layer uses an 8-qubit circuit which accepts an input vector $x \in \mathbb{R}^{64}$ and outputs a vector $\vec{h}_1 \in \mathbb{R}^{24}$. The structure in Fig. 3a is used, where $D_E = 2$, $D_T = 5$. We denote $V_{\text{in}}(\overrightarrow{W}_{\text{in}})$ for $\overrightarrow{W}_{\text{in}} \in \mathbb{R}^{136}$ to be the transformation circuit in this layer.

Table 1 Settings of three layers

	# of qubits	Input dimension	Output dimension	# of parameters (transformation + bias)
Input layer	8	64	24	136 + 24
Hidden layer	6	24	12	84 + 12
Output layer	4	12	2	32 + 0

The output of the input layer is of the form

$$\vec{h}_1 = \begin{pmatrix} \langle\psi(\vec{x}, \overrightarrow{W}_{\text{in}})|H_{1,X}|\psi(\vec{x}, \overrightarrow{W}_{\text{in}})\rangle \\ \vdots \\ \langle\psi(\vec{x}, \overrightarrow{W}_{\text{in}})|H_{8,X}|\psi(\vec{x}, \overrightarrow{W}_{\text{in}})\rangle \\ \langle\psi(\vec{x}, \overrightarrow{W}_{\text{in}})|H_{1,Y}|\psi(\vec{x}, \overrightarrow{W}_{\text{in}})\rangle \\ \vdots \\ \langle\psi(\vec{x}, \overrightarrow{W}_{\text{in}})|H_{8,Y}|\psi(\vec{x}, \overrightarrow{W}_{\text{in}})\rangle \\ \langle\psi(\vec{x}, \overrightarrow{W}_{\text{in}})|H_{1,Z}|\psi(\vec{x}, \overrightarrow{W}_{\text{in}})\rangle \\ \vdots \\ \langle\psi(\vec{x}, \overrightarrow{W}_{\text{in}})|H_{8,Z}|\psi(\vec{x}, \overrightarrow{W}_{\text{in}})\rangle \end{pmatrix} + \vec{b}_{\text{in}} \in \mathbb{R}^{24}, \quad (9)$$

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

4.1.2 Hidden layer

The hidden layer uses 6 qubits. It accepts an vector $\vec{h}_1 \in \mathbb{R}^{24}$ and outputs a vector $\vec{h}_2 \in \mathbb{R}^{12}$. The structure shown in Fig. 3b is used, with $D_E = 1$, $D_T = 4$. Because there are 30 parameters in the encoder, we set the last column of R_Z gates to be $R_Z(0)$. Similar to the input layer, the output of the hidden layer is

$$\vec{h}_2 = \begin{pmatrix} \langle\psi(\vec{h}_1, \overrightarrow{W_h})|H_{1,Y}|\psi(\vec{h}_1, \overrightarrow{W_h})\rangle \\ \vdots \\ \langle\psi(\vec{h}_1, \overrightarrow{W_h})|H_{6,Y}|\psi(\vec{h}_1, \overrightarrow{W_h})\rangle \\ \langle\psi(\vec{h}_1, \overrightarrow{W_h})|H_{1,Z}|\psi(\vec{h}_1, \overrightarrow{W_h})\rangle \\ \vdots \\ \langle\psi(\vec{h}_1, \overrightarrow{W_h})|H_{6,Z}|\psi(\vec{h}_1, \overrightarrow{W_h})\rangle \end{pmatrix} + \vec{b}_h \in \mathbb{R}^{12}. \quad (10)$$

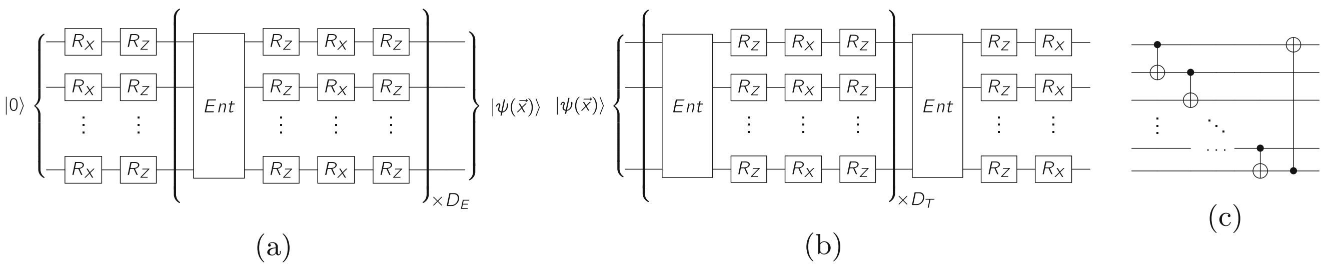

Fig. 3 Ansatz circuits in each QNNL. **a** The ansatz circuit of the encoder. **b** The ansatz circuit of the transformation. c. The structure of gate *Ent*

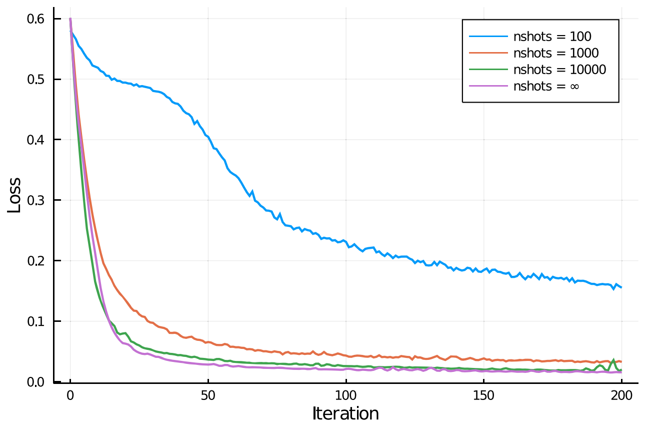

Fig. 4 Loss function

4.1.3 Output layer

The output layer uses 4 qubits. We use the structure in Fig. 3c with $D_E = 1, D_T = 2$. Because there are 20 parameters in the encoder, we set the last column of R_Z and R_X gates to be $R_Z(0)$ and $R_X(0)$. The output of the output layer is

$$\vec{y} = \begin{pmatrix} \langle\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})|(|0\rangle\langle 0| \otimes I \otimes I \otimes I)|\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})\rangle \\ \langle\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})|(|1\rangle\langle 1| \otimes I \otimes I \otimes I)|\psi(\vec{h}_2, \overrightarrow{W}_{\text{out}})\rangle. \end{pmatrix} \tag{11}$$

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

In conclusion, the settings of these three layers are shown in Table 1. Finally, the loss function is defined as

$$L = \frac{1}{|\mathcal{D}|} \sum_{(\vec{x},y)\in\mathcal{D}} |\text{QDNN}(\vec{x}) - \vec{y}|^2 , \tag{12}$$

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

4.2 Experiments result

We used the Julia package `Yao.jl` (Luo et al. 2019) as a quantum simulator in our experiments. All data were collected on a desktop PC with Intel CPU i7-4790 and 4GB RAM.

All parameters were initialized randomly in $(-\pi, \pi)$. We use Adam optimizer (Kingma and Ba 2014) to update parameters. We train this QDNN for 200 iterations with batch size of 240. We set the number of samples when evaluating the expectation value of Hamiltonian each time to 100, 1000, 1000 and ∞. The hyper parameters of Adam is set to be $\eta = 0.01, \beta_1 = 0.9, \beta_2 = 0.999$.

The values of the loss function on the training set during training is shown in Fig˙ 4. The loss function and accurate rate of this QDNN on both training set and test set after training are shown in Table 2. It shows that when the number of samples reaches 1000, we can train the QDNN with high performance.

5 Discussion

We introduce 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. As a result, the QDNN still keeps most of the advantages of the classical DNNs. Because the model is based on the hybrid quantum-classical scheme, it has the potential to be realized on NISQ processors.

Since we use a classical simulator on a desktop PC for quantum computation, only QDNNs with a small number of qubits can be used and only simple examples can be demonstrated. Quantum hardware is developing fast. Google achieved quantum supremacy by using a superconducting quantum processor with 53 qubits (Arute et al. 2019). From Table 1, only 8 qubits are used in our experiments described in the preceding section, so in principle, our image classification experiment can be implemented in Google's quantum processor. However, due to the lack of access to a real quantum computer, we are not able to give simulation tests on how the QDNN works with noises. With quantum computing resources, we can access exponential dimensional feature Hilbert spaces (Schuld and Killoran 2019) with QDNNs and only use polynomial-size parameters. Hence, we believe that QDNNs will help us to extract features more efficiently than DNNs.

Table 2 Training results

	Training set		Test set	
nshots	Loss	Accurate rate	Loss	Accurate rate
100	0.15542687722068718	90.36%	0.15639276595744667	90.17%
1000	0.03284751330438213	98.32%	0.0294211153664303	98.91%
10000	0.020227478326095588	98.92%	0.017112455910165428	99.39%
∞	0.015732338671740852	98.92%	0.013040602800738285	99.57%

Acknowledgements We thank Xiu-Zhe Luo and Jin-Guo Liu for helping us in Julia programming.

Funding This work is partially supported by an NSFC grant no. 11688101 and an NKRDP grant no. 2018YFA0306702.

Code availability All codes and data are available on http://github.com/ChenZhao44/QDNN.jl.

Declarations

Conflict of interest The authors declare no competing interests.

References

Aaronson S, Kuperberg G, Granade C (2005) The complexity zoo

Aaronson S (2015) Read the fine print. Nat Phys 11(4):291–293

Amin MH, Andriyash E, Rolfe J, Kulchytskyy B, Melko R (2018) Quantum boltzmann machine. Phys Rev X 8(2):021,050

Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, Biswas R, Boixo S, Brandao FG, Buell DA et al (2019) Quantum supremacy using a programmable superconducting processor. Nature 574:505–510

Beer K, Bondarenko D, Farrelly T, Osborne TJ, Salzmann R, Scheiermann D, Wolf R (2020) Training deep quantum neural networks. Nat Commun 11(1):1–6

Benedetti M, Lloyd E, Sack S, Fiorentini M (2019) Parameterized quantum circuits as machine learning models. Quantum Sci Technol 4(4):043,001

Biamonte J, Wittek P, Pancotti N, Rebentrost P, Wiebe N, Lloyd S (2017) Quantum machine learning. Nature 549(7671):195–202

Cao Y, Guerreschi GG, Aspuru-Guzik A (2017) Quantum neuron: an elementary building block for machine learning on quantum computers. arXiv:1711.11240

Cerezo M, Sone A, Volkoff T, Cincio L, Coles PJ (2020) Cost-function-dependent barren plateaus in shallow quantum neural networks. arXiv:2001.00550

Cong I, Choi S, Lukin MD (2019) Quantum convolutional neural networks. Nat Phys 15(12):1273–1278

Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2(4):303–314

Dallaire-Demers PL, Killoran N (2018) Quantum generative adversarial networks. Physical Rev A 98(1):012,324

Daniely A (2017) Depth separation for neural networks. In: Kale S, Shamir O (ed) Proceedings of the 2017 conference on learning theory, proceedings of machine learning research, vol 65. PMLR, Amsterdam, pp 690–696. http://proceedings.mlr.press/v65/daniely17a.html

Eldan R, Shamir O (2016) The power of depth for feedforward neural networks. In: Feldman V, Rakhlin A, Shamir O (eds) 29th Annual conference on learning theory, proceedings of machine learning research, vol 49. pp 907–940. PMLR, Columbia University, New York, New York, USA. http://proceedings.mlr.press/v49/eldan16.html

Farhi E, Neven H (2018) Classification with quantum neural networks on near term processors. arXiv:1802.06002

Gao X, Zhang Z, Duan L (2018) A quantum machine learning algorithm based on generative models, vol 4

Giovannetti V, Lloyd S, Maccone L (2008) Quantum random access memory. Phys Rev Lett 100(16):160,501

Grant E, Benedetti M, Cao S, Hallam A, Lockhart J, Stojevic V, Green AG, Severini S (2018) Hierarchical quantum classifiers. npj Quantum Inf 4(1):1–8

Grant E, Wossnig L, Ostaszewski M, Benedetti M (2019) An initialization strategy for addressing barren plateaus in parametrized quantum circuits. Quantum 3:214

Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of 28th annual ACM symposium on theory of computing, STOC '96. ACM, New York, pp 212-219. https://doi.org/10.1145/237814.237866

Havlíček V, Córcoles AD, Temme K, Harrow AW, Kandala A, Chow JM, Gambetta JM (2019) Supervised learning with quantum-enhanced feature spaces. Nature 567(7747):209

Hornik K (1991) Approximation capabilities of multilayer feed-forward networks. Neural Netw 4(2):251–257. https://doi.org/10.1016/0893-6080(91)90009-T, http://www.sciencedirect.com/science/article/pii/089360809190009T

Kandala A, Mezzacapo A, Temme K, Takita M, Brink M, Chow JM, Gambetta JM (2017) Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature 549(7671):242–246

Kerenidis I, Landman J, Prakash A (2020) Quantum algorithms for deep convolutional neural networks. In: International conference on learning representations. https://openreview.net/forum?id=Hygab1rKDS

Killoran N, Bromley TR, Arrazola JM, Schuld M, Quesada N, Lloyd S (2019) Continuous-variable quantum neural networks. Phys Rev Res 1(3):033,063

Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv:1412.6980

Kratsios A (2019) The universal approximation property: characterizations, existence, and a canonical topology for deep-learning. arXiv:1910.03344

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444

Leshno M, Lin VY, Pinkus A, Schocken S (1993) Multilayer feed-forward networks with a nonpolynomial activation function can approximate any function. Neural Netw 6(6):861–867. https://doi.org/10.1016/S0893-6080(05)80131-5. http://www.sciencedirect.com/science/article/pii/S0893608005801315

Li Y, Zhou RG, Xu R, Luo J, Hu W (2020) A quantum deep convolutional neural network for image recognition. Quantum Sci Technol 5(4):044,003. https://doi.org/10.1088/2058-9565/ab9f93

Liu JG, Wang L (2018) Differentiable learning of quantum circuit born machines. Phys Rev A 98(6):062,324

Liu JG, Zhang YH, Wan Y, Wang L (2019) Variational quantum eigensolver with fewer qubits. Phys Rev Res 1(023):025. https://doi.org/10.1103/PhysRevResearch.1.023025

Liu JG, Zhang YH, Wan Y, Wang L (2019) Variational quantum eigensolver with fewer qubits. Phys Rev Res 1(023):025. https://doi.org/10.1103/PhysRevResearch.1.023025

Lloyd S, Mohseni M, Rebentrost P (2014) Quantum principal component analysis. Nat Phys 10(9):631

Lloyd S, Weedbrook C (2018) Quantum generative adversarial learning. Phys Rev Lett 121(4):040,502

Luo XZ, Liu JG, Zhang P, Wang L (2019) Yao.jl: Extensible, efficient framework for quantum algorithm design. arXiv:1912.10877

McClean JR, Boixo S, Smelyanskiy VN, Babbush R, Neven H (2018) Barren plateaus in quantum neural network training landscapes. Nat Commun 9(1):1–6

McClean JR, Romero J, Babbush R, Aspuru-Guzik A (2016) The theory of variational hybrid quantum-classical algorithms. New J Phys 18(2):023,023

Mitarai K, Negoro M, Kitagawa M, Fujii K (2018) Quantum circuit learning. Phys Rev A 98(032):309. https://doi.org/10.1103/PhysRevA.98.032309

Nakanishi KM, Fujii K, Todo S (2019) Sequential minimal optimization for quantum-classical hybrid algorithms. arXiv:1903.12166

Pinkus A (1999) Approximation theory of the mlp model in neural networks. Acta Numerica 8:143–195. https://doi.org/10.1017/S0962492900002919

Preskill J (2018) Quantum computing in the nisq era and beyond. Quantum 2:79

Rebentrost P, Mohseni M, Lloyd S (2014) Quantum support vector machine for big data classification. Phys Rev Lett 113(13):130,503

Schuld M, Bergholm V, Gogolin C, Izaac J, Killoran N (2019) Evaluating analytic gradients on quantum hardware. Phys Rev A 99(3):032,331

Schuld M, Bocharov A, Svore KM, Wiebe N (2020) Circuit-centric quantum classifiers. Phys Rev A 101(3):032,308

Schuld M, Killoran N (2019) Quantum machine learning in feature hilbert spaces. Phys Rev Lett 122(4):040,504

Schuld M, Sinayskiy I, Petruccione F (2015) An introduction to quantum machine learning. Contemp Phys 56(2):172–185

Schuld M, Sinayskiy I, Petruccione F (2015) Simulating a perceptron on a quantum computer. Phys Lett A 379(7):660–663. https://doi.org/10.1016/j.physleta.2014.11.061, https://www.sciencedirect.com/science/article/pii/S037596011401278X

Shor PW (1994) Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. IEEE, pp 124–134

Situ H, He Z, Wang Y, Li L, Zheng S (2020) Quantum generative adversarial network for generating discrete distribution. Inf Sci

Socher R, Bengio Y, Manning CD (2012) Deep learning for nlp (without magic). In: Tutorial abstracts of ACL 2012. Association for Computational Linguistics, pp 5–5

Steinbrecher GR, Olson JP, Englund D, Carolan J (2019) Quantum optical neural networks. npj Quantum Inf 5(1):1–9

Stone MH (1948) The generalized weierstrass approximation theorem. Math Mag 21(4):167–184. http://www.jstor.org/stable/3029750

Tacchino F, Barkoutsos P, Macchiavello C, Tavernelli I, Gerace D, Bajoni D (2020) Quantum implementation of an artificial feed-forward neural network. Quantum Sci Technol 5(4):044,010. https://doi.org/10.1088/2058-9565/abb8e4

Vardi G, Shamir O (2020) Neural networks with small weights and depth-separation barriers. In: Advances in neural information processing systems, p 33

Volkoff T, Coles PJ (2021) Large gradients via correlation in random parameterized quantum circuits. Quantum Sci Technol 6(2):025,008

Voulodimos A, Doulamis N, Doulamis A, Protopapadakis E (2018) Deep learning for computer vision: a brief review. Comput Intell Neurosci 2018

Wan KH, Dahlsten O, Kristjánsson H., Gardner R, Kim M (2017) Quantum generalisation of feedforward neural networks. npj Quantum Inf 3(1):1–8

Wiebe N, Braun D, Lloyd S (2012) Quantum algorithm for data fitting. Phys Rev Lett 109(5):050,505

Zhao C, Gao XS (2021) Analyzing the barren plateau phenomenon in training quantum neural network with the ZX-calculus. arXiv:1802.06002

Zhao J, Zhang YH, Shao CP, Wu YC, Guo GC, Guo GP (2019) Building quantum neural networks based on a swap test. Phys Rev A 100(012):334. https://doi.org/10.1103/PhysRevA.100.012334