*International Journal of Computer and Information Technology (ISSN: 2279 – 0764)*
*Volume 03 – Issue 02, March 2014*

# A Novel Process Neural Networks Model Based on Quantum Computing

Xiande Liu
School of Computer & Information Technology
Northeast Petroleum University
Daqing, Heilongjiang, China

Panchi Li
School of Computer & Information Technology
Northeast Petroleum University
Daqing, Heilongjiang, China
E-mail: lipanchi {at} vip.sina.com

*Abstract*—**This work is a research on integrating quantum computing with process neural networks. To enhance the approximation and generalization ability of process neural networks (PNN), by studying the quantum implementation of information processing of process neuron, a new designing idea of process neuron, based on the quantum rotation gates and the multi-qubits controlled-Hadamard gates, is proposed in this paper. In the proposed approach, the discrete inputs are represented by the qubits, which, as the control qubits of the controlled-Hadamard gates after being rotated by the quantum rotation gates, control the target qubits for reverse. The model outputs are described by the probability amplitude of state $|1\rangle$ in the target qubits. Then the quantum-inspired process neural networks (QPNN) are designed by applying the quantum-inspired process neurons to the hidden layer and the classical neurons to the output layer. The algorithm of QPNN is derived by employing the principles of quantum computing and the {\it Levenberg-Marquardt} algorithm. Simulation results of a benchmark problem show that, under a certain condition, the QPNN is obviously superior to the classical PNN.**

***Keywords-quantum computation, quantum rotation gates, multi-qubits controller-hadamard gates, quantum-inspired process neuron, quantum-inspired process neural networks***

## I. INTRODUCTION (HEADING 1)

Many neurophysiologic experiments indicate that the information processing character of the biological nerve system mainly includes the following eight aspects: the spatial aggregation, the multi-factor aggregation, the temporal cumulative effect, the activation threshold characteristic, self-adaptability, exciting and restraining characteristics, delay characteristics, conduction and output characteristics [1]. From the definition of the M-P neuron model, we can know that traditional ANN preferably simulates voluminous biological neurons' characteristics such as the spatial weight aggregation, self-adaptability, conduction and output, but it do not fully incorporate temporal cumulative effect because the outputs of ANN depend only on the inputs at the moment regardless of the prior moment. In the process of practical information processing, the memory and output of the biological nerve not only depend on the spatial aggregation of each input information, but also are related to time delay and cumulative effect, or are even related to the multi-factor aggregation. Therefore, the process neural networks proposed by Chinese scholars in the early of this century are a new model of being able to simulate these important information processing characteristics of the biological neurons [2]. General neural networks can only be used to describe the instantaneous mapping relationship between input values and output values. The process neural networks (PNN) can describe the accumulation or aggregation effect of the output towards the input at the time axis. It is the extention of traditional neural networks in the time domain. Process neural networks have a wealth of research content. Because this model is proposed relatively late, there are many issues on algorithm design to be further studied and improved. For the networks training, Refs.[3-5] proposed an orthogonal basis expansion-based algorithm, which simplifies the time-domain aggregation operation by using the orthogonality of basis functions. In the networks performance, Ref.[6] studied some theoretical properties such as continuity, approximation ability and computing power. Refs.[7-9] investigated some application of PNN. The current PNN exists mainly as follows shortcomings. First, PNN can not directly deal with discrete inputs. In PNN, the inputs are time-varying continuous functions, but in many practical problems, the system inputs are discrete data. Secondly, when the input functions are expanded by the orthogonal base functions, the number of orthogonal basis functions is not easy to determine. In theory, the number of orthogonal basis functions is infinite, and a finite number of basis functions will inevitably lead to loss of information. The above shortcomings lead directly to the decline in PNN's approximation ability and generalization ability.

Currently, the integration of quantum computing and neural networks has attracted the attention of international scholars, and has made some interesting theoretical models [10-14]. In order to avoid fitting and orthogonal basis expansion in existing PNN, and effectively enhance the approximation and generalization ability, this paper presents a quantum-inspired process neural networks (QPNN) model. In our approach, the PNN's aggregation operations in time-domain are simulated through applying the evolution of the target qubit in a multi-qubits controlled-Hadamard gates, and the QPNN algorithm is derived from the physical meaning of quantum rotation gates and multi-qubits controlled-Hadamard

This work was supported by the National Natural Science Foundation of China (Grant No. 61170132).

gates. Experimental results show that, compared with existing PNN, QPNN's approximation and generalization ability are obviously improved under a certain condition.

## II. Process Neural Networks

Suppose that the input layer of a PNN has $n$ nodes, the middle layer (process neuron hidden layer) has $p$ nodes, and the output layer has $m$ classical M-P neurons. Its topological structure is shown in Fig.1, where $X(t)=[x_1(t), x_2(t), \dots, x_n(t)]^T$ denote the networks input, $w_{ij}(t)$ denote the connection weight functions, $\theta_1, \theta_2, \cdots, \theta_p$ denote the threshold values in hidden layer, $v_k$ denote the connection weight values in the output layer, and $Y=[y_1, y_2, \dots, y_m]^T$ denote the networks output. The input-output transform relationship of the PNN can be represented as follows

$$y_k = g\Big(\sum_{j=1}^{p} v_{jk} f\Big(\sum_{i=1}^{n}\int_0^T w_{ij}(t)x_i(t)dt - \theta_j\Big)\Big), \tag{1}$$

where $k$=1, 2, … , $m$, [0, T] is an interval of the input process.

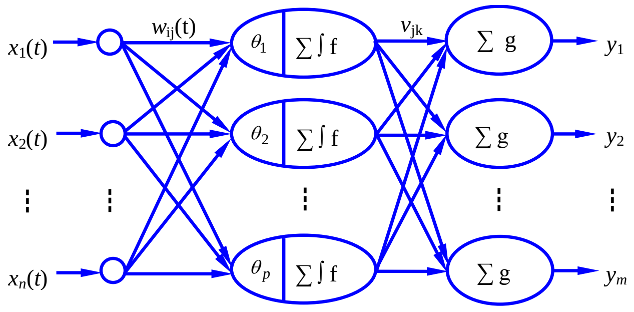

Figure 1. Process neural networks model.

Suppose that the input space of process neural networks is $[0, T]^n$, $b_1(t)$, $b_2(t)$, …, $b_l(t)$, … are a group of standard orthogonal basis functions defined in $[0, T\}]^n$, and $x_i(t)$ can be expressed as the following series form of a group of orthogonal basis functions

$$x_i(t) = \sum_{l=1}^{\infty} a_{il} b_l(t) \approx \sum_{l=1}^{L} a_{il} b_l(t), \tag{2}$$

where $L$ denotes an integer to meet the accuracy requirements, and $a_{il}$ denotes the coefficient of the $l^{th}$ basis function.

Using the a group of same basis functions as the previously mentioned, the connection weight function $w_{ij}(t)$ can be expressed as follows

$$w_{ij}(t) = \sum_{l=1}^{L} w_{ij}^{(l)} b_l(t), \tag{3}$$

where $w_{ij}^l$ is the coefficient of the $l^{th}$ basis function $b_l(t)$, and it is actually a PNN parameter that need to be trained.

Substituting the basis function expansions of $x_i(t)$ and $w_{ij}(t)$ into Eq.(1), the input-output transform relationship of the PNN can be represented as

$$y_k = g\Big(\sum_{j=1}^{p} v_{jk} f\Big(\sum_{i=1}^{n}\sum_{l=1}^{L}\sum_{s=1}^{L} a_{is} w_{ij}^l \int_0^T b_l(t)b_s(t)dt - \theta_j\Big)\Big), \tag{4}$$

where $b_1(t)$, $b_2(t)$, … , $b_L(t)$\$ are a group of standard orthogonal basis functions defined in $[0, T]^n$ and satisfy to

$$\int_0^T b_l(t)b_s(t)dt = \begin{cases} 1, & l = s \\ 0, & l \neq s \end{cases}, \tag{5}$$

hence, the input-output relationship of the PNN given by Eq.(4) can be simplified as

$$y_k = g\Big(\sum_{j=1}^{p} v_{jk} f\Big(\sum_{i=1}^{n}\sum_{l=1}^{L} a_{il} w_{ij}^l - \theta_j\Big)\Big). \tag{6}$$

It is clear that the existing PNN can only deal with the samples described by continuous functions instead of the discrete sequences.

## III. Qubits and Quantum Gates

### A. Qubits

In quantum computing, a qubit is described by quantum state wave function $|\phi(t)\rangle$, where notation like $|\,\rangle$ is called the *Dirac notation*, and we will seeing it often in the following paragraphs, as it is the standard notation for states in quantum mechanics. In a time $t \in [0, T]$, the qubit has two possible state such as $|0\rangle$ and $|1\rangle$. The difference between bits and qubits is that a qubit can be in a state other than $|0\rangle$ and $|1\rangle$, it is also possible to form the linear combinations of the states, namely superposition

$$|\phi(t)\rangle = \alpha(t)|0\rangle + \beta(t)|1\rangle, \tag{7}$$

where $\alpha(t)$ and $\beta(t)$ are complex numbers, called probability amplitudes. Hence, the qubit can also be described by probability amplitudes as $[\alpha(t), \beta(t)]^T$.

### B. Quantum Rotation Gate

In the quantum computation, the logic function can be realized by applying a series of unitary transforms to the qubit states, which the effect of the unitary transform is equal to that of the logic gate. Therefore, the quantum services with the logic transformations in a certain interval are called the quantum gates, which are the basis of performing the quantum computation. The definition of a single qubit rotation gate is given by

$$R(\theta(t))=\begin{bmatrix}\cos\theta(t) & -\sin\theta(t)\\ \sin\theta(t) & \cos\theta(t)\end{bmatrix}. \tag{8}$$

Let the quantum state $|\phi(t)\rangle=\begin{bmatrix}\cos(\theta_0(t))\\ \sin(\theta_0(t))\end{bmatrix}$, and $|\phi(t)\rangle$ can be transformed by $R(\theta(t))$ as follows

$$R(\theta(t))|\phi(t)\rangle=\begin{bmatrix}\cos(\theta_0(t)+\theta(t))\\ \sin(\theta_0(t)+\theta(t))\end{bmatrix}. \tag{9}$$

It is obvious that $R(\theta(t))$ shifts the phase of $|\phi(t)\rangle$.

### C. Hadamard Gate

The *Hadamard* gate is defined as $H=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$. This gate turns the computational basis $\{|0\rangle,|1\rangle\}$ into the new basis $\{|+\rangle,|-\rangle\}$, whose states are a superposition of the states of the computational basis $H|0\rangle=(|0\rangle+|1\rangle)/\sqrt{2}=|+\rangle$ and $H|1\rangle=(|0\rangle-|1\rangle)/\sqrt{2}=|-\rangle$. Since $H^2 = I$, $H$ is equal to it own inverse, $H^{-1}=H$. Note that $H$ is Hermitian [15]. Indeed, it is evident from the matrix representation that $(H^T)^*=H$.

### D. Multi-qubits Controlled-Hadamard Gate

In a true quantum system, a single qubit state is often affected by a joint control of multi-qubits. A multi-qubits controlled-Hadamard gate $C^n(H)$ is a kind of control model. The multi-qubits system is also described by the wave function $|x_1x_2\cdots x_n\rangle$. In a ($n$+1)-bits quantum system, when the target bit is simultaneously controlled by $n$ input bits, the dynamic behavior of the system can be described by a multi-qubits controlled-Hadamard gate in Fig.2.

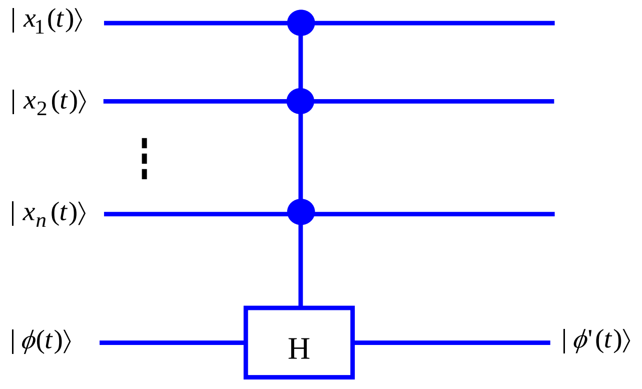

Figure 2. Multi-qubit controlled-Hadamard gate.

In Fig.2, there are n+1 qubits, and H denotes a Hadamard gate. Then we define the controlled operation Cn(H) as follows

$$\begin{aligned}&C^n(H)|x_1(t)x_2(t)\cdots x_n(t)\rangle|\phi(t)\rangle\\ &=|x_1(t)x_2(t)\cdots x_n(t)\rangle H^{x_1(t)x_2(t)\cdots x_n(t)}|\phi(t)\rangle\end{aligned}, \tag{10}$$

where $x_1x_2...x_n$ in the exponent of $H$ means the product of the bits $x_1$, $x_2$, ... , $x_n$. That is, the operator $H$ is applied to last a qubit if the first $n$ qubits are all equal to one, and otherwise, nothing is done.

Suppose that the $|x_i(t)\rangle=a_i(t)|0\rangle+b_i(t)|1\rangle$ are the control qubits, and the $|\phi(t)\rangle=c(t)|0\rangle+d(t)|1\rangle$ is the target qubit. From Eq.(10), the output of $C^n(H)$ is written by the equation

$$\begin{aligned}&C^n(H)|x_1(t)x_2(t)\cdots x_n(t)\rangle|\phi\rangle\\ &=|x_1(t)\rangle\otimes|x_2(t)\rangle\otimes\cdots\otimes|x_n(t)\rangle\otimes|\phi(t)\rangle-\\ &b_1(t)b_2(t)\cdots b_n(t)c(t)|\overbrace{11\cdots1}^{n}0\rangle-\\ &b_1(t)b_2(t)\cdots b_n(t)d(t)|\overbrace{11\cdots1}^{n}1\rangle+\\ &\sqrt{0.5}b_1(t)b_2(t)\cdots b_n(t)(c(t)+d(t))|\overbrace{11\cdots1}^{n}0\rangle+\\ &\sqrt{0.5}b_1(t)b_2(t)\cdots b_n(t)(c(t)-d(t))|\overbrace{11\cdots1}^{n}1\rangle\end{aligned}. \tag{11}$$

We say that a state of a composite system having the property that it can't be written as a product of states of its component systems is an entangled state. For reasons which nobody fully understands, entangled states play a crucial role in quantum computation and quantum information. It is observed from Eq.(11) that the output of $C^n(H)$ is in the entangled state of $n$+1 qubits, and the probability of the target qubit state $|\phi'(t)\rangle$, in which $|1\rangle$ is observed, equals to

$$P(t)=0.5(b_1(t)b_2(t)\cdots b_n(t))^2(c(t)^2+d(t)^2-2c(t)d(t))+d(t)^2. \tag{12}$$

At this time, after the joint control of the $n$ input bits, the target bit $|\phi'(t)\rangle$ can be defined as

$$|\phi'(t)\rangle=\sqrt{1-P(t)}|0\rangle+\sqrt{P(t)}|1\rangle. \tag{13}$$

## IV. QUANTUM-INSPIRED PROCESS NEURAL NETWORKS MODEL

### A. Quantum-inspired Process Neuron Model

In this section, we first propose a quantum-inspired process neuron model, as illustrated in Fig.3. This model consists of quantum rotation gates and multi-qubits controlled-Hadamard gates. The input is the wave functions $|x_i(t)\rangle$ defined in time domain interval [0, T], the output is the spatial and temporal aggregation results $|y\rangle$ in [0, T], and the control parameters are the rotation angles $\bar{\theta}_i(t)$, *i*=1, 2, … , *n*.

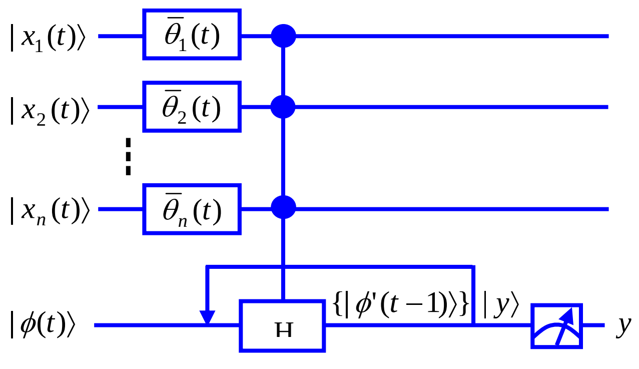

Figure 3. Quantum-inspired process neuron model.

Let the $0<t_1<t_2<\cdots<t_q=T$ represent the sampling time points, then the $|x_i(t)\rangle$ in [0, T] can be written in discrete form as follows

$$|x_i(t_r)\rangle=\cos\theta_i(t_r)|0\rangle+\sin\theta_i(t_r)|1\rangle,\ r=1,2,\cdots,q\,. \qquad (14)$$

Suppose $|\phi(t_1)\rangle=|0\rangle$. according to the definition of quantum rotation gates and multi-qubits controlled-Hadamard gates, the $|\phi'(t_1)\rangle$ is given by

$$|\phi'(t_1)\rangle=\cos\varphi(t_1)|0\rangle+\sin\varphi(t_1)|1\rangle\,, \qquad (15)$$

where $\varphi(t_1)=\arcsin(\sqrt{0.5}\prod_{i=1}^{n}\sin(\theta_i(t_1)+\bar{\theta}_i(t_1)))$.

Let $t=t_r$, $r=2,3,\cdots,q$, from $|\phi(t)\rangle=|\phi'(t-1)\rangle$, the $|\phi'(t_r)\rangle$ can be derived by

$$|\phi'(t_r)\rangle=\cos\varphi(t_r)|0\rangle+\sin\varphi(t_r)|1\rangle\,, \qquad (16)$$

where $\varphi(t_r)=\arcsin\left(\sqrt{\begin{array}{l}(\cos2\varphi(t_{r-1})-\sin2\varphi(t_{r-1}))0.5\times\\ \prod_{i=1}^{n}\sin^2(\theta_i(t_r)+\bar{\theta}_i(t_r))+\sin^2\varphi(t_{r-1})\end{array}}\right)$.

The aggregate results of quantum-inspired process neuron in [0,T] is finally derived by

$$|y\rangle=|\phi'(t_q)\rangle=\cos\varphi(t_q)|0\rangle+\sin\varphi(t_q)|1\rangle\,. \qquad (17)$$

In this paper, we define the output of the quantum-inspired process neuron as the probability amplitude of the corresponding state, in which $|1\rangle$ is observed. Thus, the actual output of the quantum-inspired process neuron is rewritten as follows

$$y=\sqrt{(\cos2\varphi(t_{q-1})-\sin2\varphi(t_{q-1}))0.5\prod_{i=1}^{n}\sin^2(\theta_i(t_q)+\bar{\theta}_i(t_q))+\sin^2\varphi(t_{q-1})}\,. \qquad (18)$$

## B. *Quantum-inspired Process Neural Networks Model*

In this paper, the QPNN model is shown in Fig.4, where the hidden layer consists of quantum-inspired process neurons, and the output layer consists of classical neurons. The $|x_1(t)\rangle,|x_2(t)\rangle,\cdots,|x_n(t)\rangle$ denote the inputs, the $\theta_{ij}(t)$ denote the rotation angles of quantum rotation gates, the $h_1,h_2,\cdots,h_p$ denote the hidden outputs, the $w_{jk}$ denote the connection weights in output layer, and the $y_1$, $y_2$, … , $y_m$ denote the networks outputs. The *Sigmoid* functions are used as the activation function in output layer. Suppose $|x_i(t)\rangle=\cos\theta_i(t)|0\rangle+\sin\theta_i(t)|1\rangle$, the [0,T] denotes the time-domain aggregate interval, and the $0=t_1<t_2<\cdots<t_q=T$ denote the discrete sampling time points, set $|\phi_j(t_1)\rangle=|0\rangle$, $j=1,2,\cdots,p$.

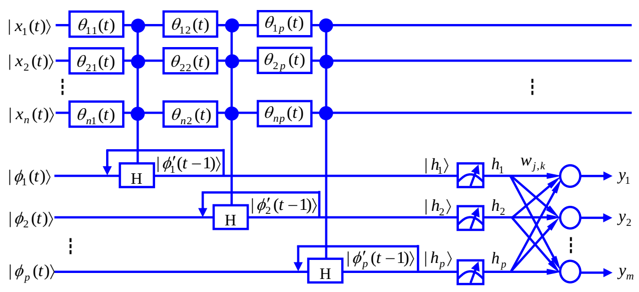

Figure 4. Quantum-inspired process neural networks model.

Let $\bar{h}_{jr}=0.5\prod_{i=1}^{n}\sin(\theta_i(t_r)+\theta_{ij}(t_r))$. According to Eq.(18), in interval [0, $t_r$], the aggregate results of the $j$th quantum-inspired process neuron in hidden layer can be written as

$$\begin{cases}h_j(t_1)=\bar{h}_{j1}\\ h_j(t_r)=\sqrt{\bar{h}_{jr}^2h_{j,r-1}+h_j^2(t_{r-1})}\end{cases}, \qquad (19)$$

where $h_{j,r-1}=1-2h_j^2(t_{r-1})-2h_j(t_{r-1})\sqrt{1-h_j^2(t_{r-1})}$.

The $j$th output in hidden layer (namely, the aggregate results in [0,T]) is given by

$$h_j=h_j(t_q)\,. \qquad (20)$$

The $k$th output in output layer can be written as

$$y_k=\frac{1}{1+\exp(-\sum_{j=1}^{p}w_{jk}h_j)}\,, \qquad (21)$$

where $i = 1, 2, \cdots, n$, $j = 1, 2, \cdots, p$, $k = 1, 2, \cdots, m$.

## V. Quantum-inspired Process Neural Networks Algorithm

### A. *Pretreatment of Input and Output Samples*

Let the sampling time points $0 = t_1 < t_2 < \cdots < t_q = T$. For the $l^{th}$ continuous function sample in *n*-dimensional Euclidean space $\bar{X}^l(t) = [\bar{x}_1^l(t), \bar{x}_2^l(t), \cdots, \bar{x}_n^l(t)]^T$, we discretize $\bar{X}^l(t)$ into the following form

$$\bar{X}^l(t_r) = [\bar{x}_1^l(t_r), \bar{x}_2^l(t_r), \cdots, \bar{x}_n^l(t_r)]^T, \tag{22}$$

where $r = 1, 2, \cdots, q$, $l = 1, 2, \cdots, L$, $L$ denotes the total number of samples.

Let

$$\begin{cases} Max_{ir} = \max(\bar{x}_i^1(t_r), \bar{x}_i^2(t_r), \cdots, \bar{x}_i^L(t_r)) \\ Min_{ir} = \min(\bar{x}_i^1(t_r), \bar{x}_i^2(t_r), \cdots, \bar{x}_i^L(t_r)) \end{cases}, \tag{23}$$

$$\theta_i^l(t_r) = \begin{cases} \dfrac{\bar{x}_i^l(t_r) - Min_{ir}}{Max_{ir} - Min_{ir}} \dfrac{\pi}{2}, & if\ Max_{ir} > Min_{ir} \\ \pi/2, & if\ Max_{ir} = Min_{ir} \neq 0 \\ 0, & if\ Max_{ir} = Min_{ir} = 0 \end{cases}. \tag{24}$$

These samples can be converted into the following quantum states

$$\{|X^l(t_r)\rangle\} = [\{x_1^l(t_r)\rangle\}, \{x_2^l(t_r)\rangle\}, \cdots, \{x_n^l(t_r)\rangle\}]^T, \tag{25}$$

where $|x_i^l(t_r)\rangle = \cos(\theta_i^l(t_r))|0\rangle + \sin(\theta_i^l(t_r))|1\rangle$.

Similarly, suppose the $l^{th}$ output sample $\bar{Y}^l = [\bar{y}_1^l, \bar{y}_2^l, \cdots, \bar{y}_m^l]^T$, where $l = 1, 2, \cdots, L$. Let

$$\begin{cases} Max_k = \max(\bar{y}_k^1, \bar{y}_k^2, \cdots, \bar{y}_k^L) \\ Min_k = \min(\bar{y}_k^1, \bar{y}_k^2, \cdots, \bar{y}_k^L) \end{cases}, \tag{26}$$

then, these output samples can be normalized by the following equation

$$y_k^l = \begin{cases} \dfrac{y_k^l - Min_k}{Max_k - Min_k}, & if\ Max_k > Min_k \\ 1, & if\ Max_k = Min_k \neq 0 \\ 0, & if\ Max_k = Min_k = 0 \end{cases}, \tag{27}$$

where $k = 1, 2, \cdots, m$.

### B. *Parameters Adjustment Method*

The QPNN adjustable parameters include the rotation angles of quantum rotation gates in hidden layer, and the weights in output layer. Suppose $\bar{y}_1^l, \bar{y}_2^l, \cdots, \bar{y}_m^l$ denote the normalized desired outputs of the $l^{th}$ sample, and $y_1^l, y_2^l, \cdots, y_m^l$ denote the corresponding actual outputs. The evaluation function is defined as follows

$$E = \max_{1\le l\le L} \max_{1\le k\le m} |e_k^l| = \max_{1\le l\le L} \max_{1\le k\le m} |\bar{y}_k^l - y_k^l|. \tag{28}$$

Let

$$\begin{cases} \bar{h}_{jr}^l = 0.5\prod_{i=1}^n \sin(\theta_i^l(t_r) + \theta_{ij}(t_r)) \\ S_{jr}^l = (\bar{h}_{jr}^l)^2(1 - 2h_j^2(t_{r-1}) - 2h_j(t_{r-1})\sqrt{1 - h_j^2(t_{r-1})}) \end{cases}. \tag{29}$$

According to the gradient descent algorithm, the gradient of the rotation angles of the quantum rotation gates can be calculated as follows

$$\begin{cases} \dfrac{\partial e_k^l}{\partial h_j^l(t_q)} = -y_k^l(1 - y_k^l)w_{jk} \\ \dfrac{\partial h_j^l(t_r)}{\partial h_j^l(t_{r-1})} = \dfrac{(1 - 2(\bar{h}_{jr}^l)^2)h_j^l(t_{r-1})\sqrt{1 - (h_j^l(t_{r-1}))^2} + (\bar{h}_{jr}^l)^2(2(h_j^l(t_{r-1}))^2 - 1)}{h_j^l(t_r)\sqrt{1 - (h_j^l(t_{r-1}))^2}} \\ \dfrac{\partial h_j^l(t_r)}{\partial \theta_{ij}(t_r)} = \dfrac{S_{jr}^l \cot(\theta_i^l(t_r) + \theta_{ij}(t_r))}{h_j^l(t_r)} \end{cases}. \tag{30}$$

Based on the above Eq.(30), we obtain

$$\frac{\partial e_k^l}{\partial \theta_{ij}(t_r)} = \frac{\partial e_k^l}{\partial h_j^l(t_q)} \prod_{s=r+1}^{q} \frac{\partial h_j^l(t_s)}{\partial h_j^l(t_{s-1})} \frac{\partial h_j^l(t_r)}{\partial \theta_{ij}(t_r)}, \tag{31}$$

where $r = 1, 2, \cdots, q$.

The gradient of the connection weights in output layer can be calculated as follows

$$\frac{\partial e_k^l}{\partial w_{jk}} = -y_k^l(1 - y_k^l)h_j^l(t_q). \tag{32}$$

Because the number of parameters is greater and the gradient calculation is more complicated, the standard gradient descent algorithm is not easy to converge. Hence we employ the *Levenberg-Marquardt* algorithm in [15] to adjust the QPNN parameters.

Let **p** denote the parameter vector, **e** denote the error vector, and **J** denote the Jacobian matrix. **p**, **e** and **J** are respectively defined as follows

$$\mathbf{P}^{\mathrm{T}} = [\theta_{11}(t_1), \theta_{11}(t_2), \cdots, \theta_{np}(t_q), w_{11}, w_{12}, \cdots, w_{pm}], \qquad (33)$$

$$\mathbf{e}^{\mathrm{T}}(\mathbf{P}) = [e_1^1, e_2^1, \cdots, e_m^1, e_1^2, e_2^2, \cdots, e_m^2, e_1^L, e_2^L, \cdots, e_m^L], \qquad (34)$$

$$\mathbf{J}(\mathbf{P}) = \begin{bmatrix} \frac{\partial e_1^1}{\partial\theta_{11}(t_1)} & \frac{\partial e_1^1}{\partial\theta_{11}(t_2)} & \cdots & \frac{\partial e_1^1}{\partial\theta_{np}(t_q)} & \frac{\partial e_1^1}{\partial w_{11}} & \frac{\partial e_1^1}{\partial w_{12}} & \cdots & \frac{\partial e_1^1}{\partial w_{pm}} \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ \frac{\partial e_m^1}{\partial\theta_{11}(t_1)} & \frac{\partial e_m^1}{\partial\theta_{11}(t_2)} & \cdots & \frac{\partial e_m^1}{\partial\theta_{np}(t_q)} & \frac{\partial e_m^1}{\partial w_{11}} & \frac{\partial e_m^1}{\partial w_{12}} & \cdots & \frac{\partial e_m^1}{\partial w_{pm}} \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ \frac{\partial e_1^L}{\partial\theta_{11}(t_1)} & \frac{\partial e_1^L}{\partial\theta_{11}(t_2)} & \cdots & \frac{\partial e_1^L}{\partial\theta_{np}(t_q)} & \frac{\partial e_1^L}{\partial w_{11}} & \frac{\partial e_1^L}{\partial w_{12}} & \cdots & \frac{\partial e_1^L}{\partial w_{pm}} \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ \frac{\partial e_m^L}{\partial\theta_{11}(t_1)} & \frac{\partial e_m^L}{\partial\theta_{11}(t_2)} & \cdots & \frac{\partial e_m^L}{\partial\theta_{np}(t_q)} & \frac{\partial e_m^L}{\partial w_{11}} & \frac{\partial e_m^L}{\partial w_{12}} & \cdots & \frac{\partial e_m^L}{\partial w_{pm}} \end{bmatrix} .(35)$$

According to *Levenberg-Marquardt* algorithm, the iterative equation of adjusting QPNN parameters is written as follows

$$\mathbf{P}_{t+1} = \mathbf{P}_t - (\mathbf{J}^{\mathrm{T}}(\mathbf{P}_t)\mathbf{J}(\mathbf{P}_t) + \mu_t \mathbf{I})^{-1} \mathbf{J}^{\mathrm{T}}(\mathbf{P}_t)\mathbf{e}(\mathbf{P}_t), \qquad (36)$$

where $t$ denotes the iterative steps, **I** denotes the unit matrix, $\mu_t$ is a small positive number to ensure the matrix $\mathbf{J}^{\mathrm{T}}(\mathbf{P}_t)\mathbf{J}(\mathbf{P}_t) + \mu_t\mathbf{I}$ invertible.

If the value of the evaluation function $E$ reaches the predefined precision within the preset maximum number of iterative steps, then the execution of the algorithm is stopped, else the algorithm is not stopped until it reaches the predefined maximum number of iterative steps.

## VI. SIMULATIONS

To examine the effectiveness of the proposed QPNN, two example of Time series prediction are used to compare it with the classical PNN (CPNN) in this section. In these experiments, our QPNN has the same structure and parameters as the CPNN, and the same *Levenberg-Marquardt* algorithm is applied in two models. To facilitate comparison, some relevant concepts are defined as follows.

**Approximation error** Suppose $\bar{y}_k^l$ and $y_k^l$ denote the desired output and actual output after training, respectively. The approximation error is defined as

$$E = \max_{1\le l\le L}\max_{1\le k\le m} |\bar{y}_k^l - y_k^l|, \qquad (37)$$

where $L$ denotes the total number of the training samples.

**Average approximation error** Suppose $E_1, E_2, \cdots, E_N$ denotes the approximation error over $N$ training, respectively. The average approximation error is defined as

$$E_{avg} = \frac{1}{N}\sum_{i=1}^{N} E_i . \qquad (38)$$

**Convergence ratio** Suppose $E$ denotes the approximation error after training, and $\varepsilon$ denotes the target error. If $E < \varepsilon$, the network training is considered to have converged. Suppose $N$ denotes the total number of training trials, and $C$ denotes the number of convergent training trials. The convergence ratio is defined as $\lambda = C/N$.

**Iterative steps** In a training trial, the times of adjusting all network parameters are defined as iterative steps.

**Average iterative steps** Suppose $S_1, S_2, \cdots, S_N$ denote the iterative steps over $N$ training trials, respectively. The average iterative steps are defined as

$$S_{avg} = \frac{1}{N}\sum_{i=1}^{N} S_i . \qquad (39)$$

**Average running time** Suppose $T_1, T_2, \cdots, T_N$ denote the running time over $N$ training trials, respectively. The average running time is defined as

$$T_{avg} = \frac{1}{N}\sum_{i=1}^{N} T_i . \qquad (40)$$

### A. *Time series prediction for Mackey–Glass*

Mackey-Glass time series can be generated by the following iterative equation

$$x(t+1) - x(t) = a\frac{x(t-\tau)}{1 + x^{10}(t-\tau)} - bx(t), \qquad (41)$$

where $t$ and $\tau$ are integers, $a = 0.2, b = 0.1, \tau = 17,$ and $x(0) \in (0,1)$.

From the above equation, we may obtain the time sequence $\{x(t)\}_{t=1}^{1000}$. We take the first 800 as the training set, and the remaining 200 as the testing set.

Our prediction scheme is to employ $n$ data adjacent to each other to predict the next one data. Namely, in our model, the sequence length equals to $n$. Therefore, each sample consists of $n$ input values and an output value. Hence, there is only one output node in QPNN and CPNN.

For the number of input nodes of QPNN and CPNN, we employ the following six kinds of settings shown in Table 1.

For each of these settings in Table 1, a single input sample can be described as a matrix.

TABLE I. THE INPUT NODES AND THE SEQUENCE LENGTH SETTING OF QPNNS AND CPNNS

| **Input Nodes** | **Sequence Length** |
|---|---|
| 1 | 32 |
| 2 | 16 |
| 4 | 8 |
| 8 | 4 |
| 16 | 2 |
| 32 | 1 |

In order to fully compare the approximation ability of two models, the number of hidden nodes is respectively set to $10, 15, 20, 25, 30$. The predefined precision is set to 0.05, and the maximum of iterative steps is set to 100. The QPNN rotation angles in hidden layer are initialized to random numbers in $(-\pi/2, \pi/2)$, and the connection weights in output layer are initialized to random numbers in (-1,1). For CPNN, the Lagrange polynomial functions are applied to fitting discrete input data. The Walsh orthogonal basis functions are applied to expand the input functions, and the number of basis functions is set to sequence length. All CPNN weights are initialized to random values in interval (-1,1).

Our experiment scheme is that, for each kind of combination of input nodes and hidden nodes, six QPNNs and CPNNs are respectively run 10 times. Then we use four indicators, such as *the average approximation error*, *the average iterative steps*, *the average running time*, and *the convergence ratio*}, to compare QPNN with CPNN. Training result contrasts are shown in Tables 2-5, where QPNNn_q denotes QPNN with *n* input nodes and *q* sequence length.

Experimental results show that only when the number of input nodes n=1,2,32, the performance of QPNN is inferior to that of CPNN, but when n=4,8,16, the QPNN's performance is superior to CPNN. From the experimental results, we can also see that when the number of input nodes is close to the sequence length, the QPNN is obviously superior to the CPNN.

TABLE II. TRAINING RESULTS OF AVERAGE APPROXIMATION ERROR

| **Model** | **Hidden Nodes** | | | | |
|---|---|---|---|---|---|
| | ***10*** | ***15*** | ***20*** | ***25*** | ***30*** |
| QPNN1_32 | 0.0430 | 0.3170 | 0.1317 | 0.0416 | 0.4101 |
| CPNN1_32 | 0.2459 | 0.2432 | 0.1026 | 0.1747 | 0.1724 |
| QPNN2_16 | 0.0455 | 0.0419 | 0.2266 | 0.2236 | 0.1351 |
| CPNN2_16 | 0.2563 | 0.2568 | 0.1088 | 0.1879 | 0.1848 |
| QPNN4_8 | **0.0426** | **0.0436** | **0.0434** | **0.0425** | **0.0425** |
| CPNN4_8 | 0.2756 | 0.2759 | 0.1163 | 0.1982 | 0.1955 |
| QPNN8_4 | **0.0433** | **0.0443** | **0.0427** | **0.0409** | **0.0441** |
| CPNN8_4 | 0.2885 | 0.2896 | 0.1226 | 0.2082 | 0.2081 |
| QPNN16_2 | **0.0859** | **0.0426** | **0.0440** | **0.0444** | **0.0431** |
| CPNN16_2 | 0.3046 | 0.3053 | 0.1294 | 0.2186 | 0.2180 |
| QPNN32_1 | 0.4746 | 0.4742 | 0.4744 | 0.4742 | 0.4745 |
| CPNN32_1 | 0.3198 | 0.3200 | 0.1359 | 0.2299 | 0.2287 |

TABLE III. TRAINING RESULTS OF AVERAGE ITERATIVE STEPS

| **Model** | **Hidden Nodes** | | | | |
|---|---|---|---|---|---|
| | ***10*** | ***15*** | ***20*** | ***25*** | ***30*** |
| QPNN1_32 | 10.000 | 35.900 | 16.900 | 7.0000 | 43.600 |
| CPNN1_32 | 34.770 | 31.344 | 22.423 | 23.281 | 23.095 |
| QPNN2_16 | 7.5000 | 6.8000 | 24.500 | 24.500 | 14.700 |
| CPNN2_16 | 37.109 | 33.777 | 23.789 | 24.822 | 24.373 |
| QPNN4_8 | **6.1000** | **5.3000** | **4.7000** | **4.6000** | **4.6000** |
| CPNN4_8 | 39.203 | 35.325 | 25.590 | 26.389 | 25.643 |
| QPNN8_4 | **6.9000** | **6.3000** | **6.0000** | **5.5000** | **4.9000** |
| CPNN8_4 | 41.606 | 37.405 | 26.742 | 27.766 | 27.108 |
| QPNN16_2 | **34.100** | **15.000** | **11.600** | **10.200** | **8.4000** |
| CPNN16_2 | 43.638 | 39.301 | 28.141 | 29.155 | 28.626 |
| QPNN32_1 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| CPNN32_1 | 45.800 | 41.200 | 29.600 | 30.600 | 30.100 |

TABLE IV. TRAINING RESULTS OF AVERAGE RUNNING TIME ($10^3$s)

| **Model** | **Hidden Nodes** | | | | |
|---|---|---|---|---|---|
| | ***10*** | ***15*** | ***20*** | ***25*** | ***30*** |
| QPNN1_32 | 0.0803 | 0.4095 | 0.3013 | 0.1423 | 1.0611 |
| CPNN1_32 | 0.0178 | 0.0288 | 0.0600 | 0.0537 | 0.0763 |
| QPNN2_16 | 0.0299 | 0.0475 | 0.2310 | 0.2891 | 0.1991 |
| CPNN2_16 | 0.0186 | 0.0305 | 0.0637 | 0.0578 | 0.0814 |
| QPNN4_8 | **0.0177** | **0.0192** | **0.0297** | **0.0401** | **0.0514** |
| CPNN4_8 | 0.0198 | 0.0329 | 0.0676 | 0.0611 | 0.0858 |
| QPNN8_4 | **0.0124** | **0.0196** | **0.0306** | **0.0394** | **0.0454** |
| CPNN8_4 | 0.0209 | 0.0346 | 0.0720 | 0.0642 | 0.0901 |
| QPNN16_2 | **0.0207** | **0.0360** | **0.0392** | **0.0454** | **0.0461** |
| CPNN16_2 | 0.0219 | 0.0364 | 0.0755 | 0.0675 | 0.0947 |
| QPNN32_1 | 0.1222 | 0.1509 | 0.2371 | 0.3722 | 0.4283 |
| CPNN32_1 | 0.0230 | 0.0381 | 0.0793 | 0.0709 | 0.0994 |

TABLE V. TRAINING RESULTS OF COVERGENCE RATIO (%)

| **Model** | **Hidden Nodes** | | | | |
|---|---|---|---|---|---|
| | ***10*** | ***15*** | ***20*** | ***25*** | ***30*** |
| QPNN1_32 | 100 | 70 | 90 | 100 | 60 |
| CPNN1_32 | 80 | 90 | 90 | 80 | 80 |
| QPNN2_16 | 100 | 100 | 80 | 80 | 90 |
| CPNN2_16 | 80 | 80 | 90 | 80 | 80 |
| QPNN4_8 | **100** | **100** | **100** | **100** | **100** |
| CPNN4_8 | 80 | 80 | 90 | 80 | 80 |
| QPNN8_4 | **100** | **100** | **100** | **100** | **100** |
| CPNN8_4 | 80 | 80 | 90 | 80 | 80 |
| QPNN16_2 | **90** | **100** | **100** | **100** | **100** |
| CPNN16_2 | 70 | 80 | 90 | 80 | 80 |
| QPNN32_1 | 0 | 0 | 0 | 0 | 0 |
| CPNN32_1 | 70 | 70 | 90 | 80 | 80 |

Next, we investigate the generalization ability of QPNN. Based on the above experimental results, we only investigate QPNN4_8, QPNN8_4 and QPNN16_2. Our experiment scheme is that three QPNNs and CPNN train 10 times on the training set, and the generalization ability is immediately investigated on the testing set after each training. The average results of the 10 tests are regarded as the evaluation indexes. We first present the following definition of evaluation indexes.

**Average prediction error** Suppose $[\bar{y}_1^l, \bar{y}_2^l, \cdots, \bar{y}_m^l]$ and $[\hat{y}_2^l(t), \hat{y}_2^l(t), \cdots, \hat{y}_m^l(t)]$ denote the desired output of the $l^{th}$

sample and the corresponding prediction output after the $t^{th}$ testing respectively. The average prediction error over $N$ testing is defined as

$$APE = \frac{1}{N}\sum_{t=1}^{N} \max_{1\le l\le L} \max_{1\le k\le m} |\bar{y}_k^l - \hat{y}_k^l(t)|, \qquad (42)$$

where $m$ denotes the dimension of the output space, $L$ denotes the number of the testing samples.

**Average error mean** Suppose $\bar{y}^l = [\bar{y}_1^l, \bar{y}_1^l, \cdots, \bar{y}_m^l]$ and $\hat{y}^l(t) = [\hat{y}_1^l(t), \hat{y}_1^l(t), \cdots, \hat{y}_m^l(t)]$ denote the the desired output of the $l^{th}$ sample and the corresponding prediction output after the $t^{th}$ testing respectively. The average error mean over $N$ testing is defined as

$$AEM = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{L}\sum_{l=1}^{L}|\bar{y}^l - \hat{y}^l(t)|. \qquad (43)$$

**Average prediction variance** Suppose $\bar{y}^l = [\bar{y}_1^l, \bar{y}_1^l, \cdots, \bar{y}_m^l]$ and $\hat{y}^l = [\hat{y}_1^l(t), \hat{y}_1^l(t), \cdots, \hat{y}_m^l(t)]$ denote the desired output of the $l^{th}$ sample and the corresponding prediction output after the $t^{th}$ testing respectively. The average error variance over $N$ testing is defined as

$$APV = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{L-1}\sum_{l=1}^{L}\left(|\bar{y}^l - \hat{y}^l(t)| - \frac{1}{L}\sum_{l=1}^{L}|\bar{y}^l - \hat{y}^l(t)|\right)^2. \qquad (44)$$

The evaluation index contrasts of QPNNs and CPNNs are shown in Table 6. Taking 8 input nodes and 25 hidden nodes for example, and the average prediction result contrasts over 10 testing are illustrated in Fig.5. The experimental results show that the generalization ability of three QPNNs is obviously superior to that of CPNNs.

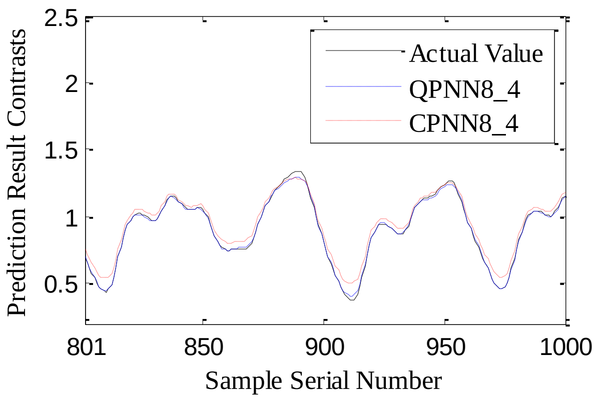

Figure 5. The average prediction result contrasts of QPNN and CPNN.

TABLE VI. PREDICTION RESULT CONTRASTS OF QPNNS AND CPNNS

| **Model** | **Hidden Nodes** | | | | | |
|---|---|---|---|---|---|---|
| | ***Index*** | ***10*** | ***15*** | ***20*** | ***25*** | ***30*** |
| QPNN4_8 | *APE* | 0.0509 | 0.0526 | 0.0537 | 0.0524 | 0.0523 |
| | *AEM* | 0.0087 | 0.0093 | 0.0087 | 0.0082 | 0.0089 |
| | *APV* | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |
| CPNN4_8 | *APE* | 0.2845 | 0.2885 | 0.1586 | 0.2101 | 0.2066 |
| | *AEM* | 0.1260 | 0.1351 | 0.0186 | 0.0814 | 0.0820 |
| | *APV* | 0.0159 | 0.0160 | 0.0003 | 0.0108 | 0.0106 |
| QPNN8_4 | *APE* | 0.0518 | 0.0528 | 0.0519 | 0.0502 | 0.0530 |
| | *AEM* | 0.0087 | 0.0095 | 0.0084 | 0.0085 | 0.0085 |
| | *APV* | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |
| CPNN8_4 | *APE* | 0.3003 | 0.3036 | 0.1665 | 0.2197 | 0.2209 |
| | *AEM* | 0.1348 | 0.1432 | 0.0196 | 0.0868 | 0.0862 |
| | *APV* | 0.0168 | 0.0169 | 0.0003 | 0.0112 | 0.0114 |
| QPNN16_2 | *APE* | 0.0962 | 0.0513 | 0.0543 | 0.0522 | 0.0512 |
| | *AEM* | 0.0289 | 0.0112 | 0.0117 | 0.0108 | 0.0115 |
| | *APV* | 0.0017 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |
| CPNN16_2 | *APE* | 0.3166 | 0.3196 | 0.1759 | 0.2315 | 0.2310 |
| | *AEM* | 0.1411 | 0.1497 | 0.0207 | 0.0910 | 0.0908 |
| | *APV* | 0.0177 | 0.0178 | 0.0004 | 0.0119 | 0.0119 |

## *B. Annual average of sunspot prediction*

All month mean and year mean of sunspots from 1749 to 2007 are shown in Table 7.

TABLE VII. PART OF THE SUNSPOT DATA (1749-2007)

| **Year** | **Month Mean** | | | | | | | **Year Mean** |
|---|---|---|---|---|---|---|---|---|
| | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | **…** | ***12*** | |
| 1749 | 58 | 62.6 | 70 | 55.7 | 85 | … | 85.2 | 80.9 |
| 1750 | 73.3 | 75.9 | 89.2 | 88.3 | 90 | … | 75.4 | 83.4 |
| … | … | … | … | … | … | … | … | … |
| 1948 | 109 | 86.1 | 94.8 | 190 | 174 | … | 138 | 136.3 |
| 1949 | 119 | 182 | 158 | 147 | 106 | … | 118 | 134.7 |
| 1950 | 102 | 94.8 | 110 | 113 | 106 | … | 54.1 | 83.9 |
| … | … | … | … | … | … | … | … | … |
| 2007 | 16.8 | 10.7 | 4.5 | 3.4 | 11.7 | … | 10.1 | 7.5 |

Our prediction schemes are to use the month mean in the first $n$ years to predict the $(n+1)^{th}$ year mean. Taking $n$=6 for example, the samples design method is shown in Table 8. From prediction schemes, each of input samples can be described by an $n\times 12$ matrix, and the corresponding output sample is a real number.

TABLE VIII. THE SAMPLES DESIGN METHOD (1749-2007)

| **Serial Number** | **Input Year (Month Mean)** | | | | | | **Prediction (Year Mean)** |
|---|---|---|---|---|---|---|---|
| | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | **…** | |
| 1 | 1749 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 |
| 2 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 |
| 3 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 |
| 4 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 |
| … | … | … | … | … | … | … | … |
| 253 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 |

From prediction schemes, we know that both QPNN and CPNN have $n$ input nodes, one output node, and the discrete sequence length equals to twelve. In this prediction, in order to enhance the objectivity of comparison results, we set the number of input nodes equal to 2, 4, 6, 8, 10, 12 respectively, and the number of hidden nodes equal to 5, 6, … , 20, respectively. The normalized maximum absolute error is set to $10^{-5}$, and the maximum number of iterative steps is set to 100. The QPNN rotation angles in hidden layer are initialized to random numbers in $(-\pi/2, \pi/2)$, and the connection weights in output layer are initialized to random numbers in (-1,1). In CPNN, the Lagrange polynomial functions are applied to fitting discrete input data. The Walsh orthogonal basis functions are applied to expand the input functions, and the number of basis functions is set to 16. All CPNN weights are initialized to random values in interval (-1,1).

For each kind of combination of input nodes and hidden nodes, two models are independently run 10 times, respectively, and then we use three indicators, such as the average approximation error, the average iterative steps, and the convergence times, to compare QPNN with CPNN. In all samples, we use the first 200 years (1749-1948) data to train networks, and the remaining 59 years (1949-2007) data to test the generalization of QPNN and CPNN. Training result contrasts of average approximation error are shown in Table 9, where, in the "QPNNn" and "CPNNn", "n" denotes the number of input nodes.

TABLE IX. TRAINING RESULT CONTRASTS OF AVERAGE APPROXIMATION ERROR

| **Model** | **Hidden Nodes** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | ***6*** | ***8*** | ***10*** | ***12*** | ***14*** | ***16*** | ***18*** | ***20*** |
| QPNN2 | 108.2 | 108.9 | 106.4 | 103.5 | 116.9 | 58.70 | 66.52 | 53.42 |
| CPNN2 | 38.58 | 12.30 | 22.19 | 1.334 | 2.735 | 0.233 | 19.53 | 17.26 |
| QPNN4 | 0.053 | 0.002 | 34.42 | 17.21 | 17.21 | 0.001 | 17.21 | 0.002 |
| CPNN4 | 4.955 | 0.443 | 68.89 | 17.22 | 17.22 | 17.66 | 51.68 | 17.21 |
| QPNN6 | 0.001 | 0.002 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| CPNN6 | 18.86 | 0.787 | 17.50 | 34.68 | 17.21 | 34.64 | 17.35 | 34.42 |
| QPNN8 | 0.001 | 0.001 | 0.001 | 0.001 | 0.002 | 0.001 | 0.0001 | 0.002 |
| CPNN8 | 4.934 | 3.773 | 20.32 | 17.84 | 17.43 | 34.59 | 51.64 | 36.01 |
| QPNN10 | 38.03 | 0.002 | 0.002 | 0.002 | 0.001 | 0.002 | 0.002 | 0.001 |
| CPNN10 | 65.63 | 43.69 | 24.17 | 4.192 | 0.942 | 42.56 | 42.59 | 21.84 |
| QPNN12 | 56.83 | 56.80 | 9.481 | 0.003 | 0.002 | 0.002 | 0.002 | 0.001 |
| CPNN12 | 6.510 | 22.22 | 42.77 | 24.72 | 63.72 | 51.68 | 71.34 | 3.261 |

Experimental results show that only when the number of input nodes n=2, the performance of QPNN is inferior to that of CPNN, but when n=4,6,8,10,12, the QPNN's performance is superior to CPNN. From the experimental results, we can also see that when the number of input nodes is close to the sequence length, the QPNN is obviously superior to the CPNN.

We use the remaining 59 years (1949-2007) data to test the generalization ability of QPNN and CPNN. From the above experimental results, when the number of input nodes *n*=6 and *n*=8, the QPNN shows the best performance, hence, we only investigate QPNN8 and CPNN8. Our experiment scheme is that, for each value of hidden nodes, two PNNs are respectively done 10 training on the training set, and are immediately investigated the generalization ability on the testing set after each training. The average prediction error over 10 predictions is shown in Fig.6. When the number of hidden nodes equal to fifteen, the prediction result contrasts are shown in Fig.7. Comparison results show that the generalization ability of two QPNNs is obviously superior to that of CPNN.

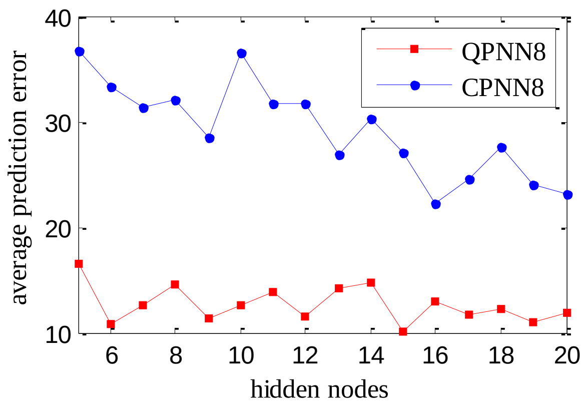

Figure 6. The average prediction error contrasts of QPNN and CPNN.

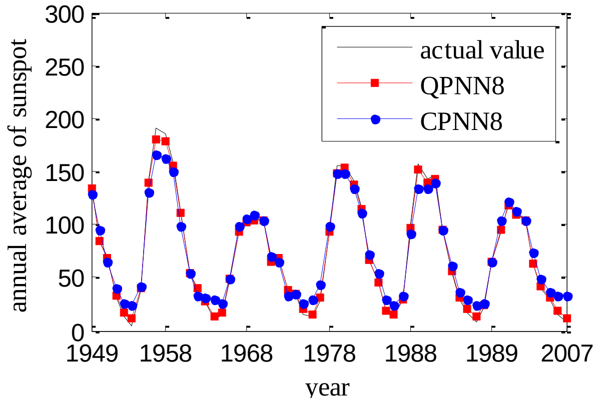

Figure 7. The prediction result contrasts of QPNN and CPNN.

## C. *The experiment results analysis*

These experimental results can be explained as follows. For processing of input information, QPNN and CPNN take two different approaches. QPNN directly receives a discrete input sequence. In QPNN, using quantum information processing mechanism, the input is circularly mapped to the output of quantum controlled-Hadamard gates in hidden layer. As the controlled-Hadamard gate's output is in the entangled state of multi-qubits, therefore, this mapping is highly nonlinear, which make QPNN have the stronger approximation ability. In addition, QPNN's each input sample can be described as a matrix with *n* rows and *q* columns. It is clear from QPNN's algorithm that, for the different combination of *n* and *q*, the output of quantum-inspired neuron in hidden layer is also different. In fact, the number of discrete points *q* denotes the *depth* of pattern memory, and the number of input nodes *n* denotes the *breadth* of pattern memory. When the *depth* and the *breadth* are appropriately matched, the QPNN shows excellent performance. For the CPNN, because it is not directly deal with discrete input, and need to transform a discrete sample into a continuous function, therefore, there must be fitting errors. In subsequent orthogonal basis expansion, there must be truncation errors. Hence, in the CPNN information processing, there exists inevitably the loss of sample characteristics, which affects its approximation ability and generalization ability.

## VII. CONCLUSIONS

This paper proposes quantum-inspired process neural networks model based on the principle of quantum computing. The architecture of the proposed model includes three layers, where the hidden layer consists of quantum-inspired neurons and the output layer consists of classical neurons. An obvious difference from classical PNN is that each dimension of a single input sample consists of a discrete sequence rather that a continuous function. The activation function of hidden layer is redesigned according to the principle of quantum computing. The *Levenberg-Marquardt* algorithm is employed for learning. With application of the information processing mechanism of quantum controlled-Hadamard gates, proposed model can effectively obtain the sample characteristics by way of *breadth* and *depth*. The experimental results reveal that a greater difference between input nodes and sequence length leads to a lower performance of proposed model than that of classical PNN, on the contrary, it obviously enhance approximation and generalization ability of proposed model when input nodes is close to sequence length. The following issues of the proposed model, such as continuity, computational complexity, and improvement of learning algorithm, are subject of further research.

## REFERENCES

[1] A.C. Tosi, “Locally Recurrent Globally Feed-forward Networks, A Critical Review of Architectures”, IEEE Transactions on Neural Networks, vol. 7, pp.229-239, 1997.

[2] X.G. He, J.Z. Liang, “Process Neural Networks”, In Proceedings of the Congress on Intelligent Information Processing, pp.15-23, 2000.

[3] X.G. He, J.Z. Liang, “Some Theoretical Issues on Process Neural Networks”, Chinese Engineering Science, vol. 2, pp.40-44, 2000.

[4] X.G. He, J.Z. Liang, “Learning for Process Neural Networks and Its Applications”, Chinese Engineering Science, vol. 3, pp.31-45, 2001.

[5] S.S Zhong, Y. Li, G. Ding, “Continuous Wavelet Process Neural Networks and Its Application”, Neural Network World, vol. 17, pp.483-495, 2007.

[6] S.H. Xu, X.G. He, “Some Theoretical Issues on Continuous Process Neural Networks” , Acta Electronica Sinic, vol. 34, pp.1838-1841, 2006.

[7] G. Ding, S.S. Zhong, “Aircraft Engine Lubricating Oil Monitoring by Process Neural Networks”, Neural Network World, vol. 16, pp.15-24, 2006.

[8] G. Ding, S.S. Zhong, “Time Series Prediction by Parallel Feedforward Process Neural Networks with Time-varied Input and Output Functions”, Neural Network World, vol. 12, pp.137-147, 2005.

[9] T. Ye, X.F. Zhu, “The bridge relating process neural networks and traditional neural networks”, Neurocomputing, vol. 74, pp.906-915, 2011.

[10] M. Michiharu, S. Masaya, M. Hiromi, “Qubit Neuron According to Quantum Circuit for XOR Problem”, Applied Mathematics and Computation, vol. 185, pp.1015-1025, 2007.

[11] P.C. Li, K.P. Song, E.L. Yang, “Model and Algorithm of Neural Networks with Quantum Gated Nodes”, Neural Network World, vol. 20, pp.189-206, 2010.

[12] P.C. Li, G.Y.Shi, “Sequence input-based quantum neural networks model and algorithm”, Pattern Recognition and Artificial Intelligence, vol. 26, pp. 247-253, 2013.

[13] C.Y.i Liu, C. Chen, C. Chang, et al, “Single-hidden-layer feed-forward quantum neural network based on Grover learning”, Neural Networks, vol. 45, pp.144-150, 2013.

[14] J. Adenilton, D. WilsonR, B. Teresa, “Classical and superposed learning for quantum weightless neural networks”, Neurocomputing, vol. 75, pp. 52-60, 2012.

[15] T.H. Martin, B.D. Howard, H.B. Mark, “Neural Network Design”, PWS Publishing Company, New York,1996.