Neurocomputing 134 (2014) 159–164

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Multi-layer quantum neural network controller trained by real-coded genetic algorithm

Kazuhiko Takahashi [a,*], Motoki Kurokawa [b], Masafumi Hashimoto [c]

[a] Information Systems Design, Doshisha University, Kyoto 610-0321, Japan
[b] Graduate School of Doshisha University, Kyoto 610-0321, Japan
[c] Intelligent Information Engineering and Science, Doshisha University, Kyoto 610-0321, Japan

ARTICLE INFO

Article history:
Received 5 June 2012
Received in revised form
31 October 2012
Accepted 31 December 2012
Available online 28 January 2014

Keywords:
Quantum neural network
Qubit neuron
Real-coded genetic algorithm
Control
Identification

ABSTRACT

We investigate a quantum neural network and discuss its application to controlling systems. First, we consider a multi-layer quantum neural network that uses qubit neurons as its information processing unit. Next, we propose a direct neural network controller using the multi-layer quantum neural network. To improve learning performance, instead of applying a back-propagation algorithm for the supervised training of the multi-layer quantum neural network, we apply a real-coded genetic algorithm. To evaluate the capabilities of the direct quantum neural network controller, we conduct computational experiments controlling a discrete-time nonlinear system and a nonholonomic system (a two-wheeled robot). Experimental results confirm the effectiveness of the real-coded genetic algorithm in training a quantum neural network and prove the feasibility and robustness of the direct quantum neural network controller.

1. Introduction

Feynman [1] introduced the possibility of using quantum mechanical systems for reasonable computing. After Deutsch [2] proposed the first quantum computing model, several quantum computing algorithms, such as Shor's factoring algorithm [3] and Grover's search algorithm [4], were proposed. Similarly, interest in artificial neural networks based on quantum theoretical concepts and techniques (hereafter called quantum neural networks) increased after Kak [5] first presented the concept of quantum neural computing. This increase in interest was due to the belief that quantum neural networks may provide a new understanding of certain brain functions and also help solve classically intractable problems [6]. In quantum computing, 'qubits' (an abbreviation for quantum bits) of quantum computers are the counterparts of 'bits' of classical computers. Qubits are used to store the states of circuits during quantum computations. Multi-layer quantum neural networks utilize qubit neurons as information processing units [7,8]. In the qubit neuron model, neuron states are connected to the quantum states and transitions between neuron states are based on operations derived from quantum logic gates. The high-learning ability of multi-layer quantum neural networks with qubit neurons has been demonstrated via numerous basic benchmark tests and applications [9–14]. From the viewpoint of control applications, the high-learning ability of multi-layer quantum neural networks can be used to control a wide class of systems. However, because studies on applying multi-layer quantum neural networks to control systems have not yet been completed, the characteristics of control systems based on quantum neural networks have not yet been clarified. Moreover, these previously mentioned works apply the back-propagation algorithm to train the multi-layer quantum neural network, which minimizes a quadratic cost function using the steepest descent method. When the back-propagation algorithm is used, the convergence of the training procedure for quantum neural networks depends on the initial conditions of the network parameters. Moreover, depending on the search surface of the cost function being optimized, the training procedure occasionally falls into local minima. To address this issue, several approaches used in conventional neural networks, such as the combination of the back-propagation algorithm and a random search algorithm [15], could be used to improve the training of quantum neural networks. However, control applications that use quantum neural networks trained by the back-propagation algorithm need to address the Jacobian problem [16], i.e., the calculation of the derivative of the cost function requires sensitivity information of the output of the target system being controlled with respect to the control input.

We investigate the application of multi-layer quantum neural networks with qubit neurons to controllers. Specifically, we develop a method for designing and evaluating control systems on the basis of quantum neural networks. Several control systems using quantum neural networks can be considered, e.g., a direct

* Corresponding author.
E-mail address: katakaha@mail.doshisha.ac.jp (K. Takahashi).

0925-2312/$ - see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.12.073

controller in which the control input to the target system utilizes the output from the quantum neural network directly, a parallel controller in which the control input consists of the output from the quantum neural network and the output from a conventional controller and a self-tuning controller in which the control input is generated by a conventional controller whose parameters are adjusted by the quantum neural network. We consider a direct controller that uses a multi-layer quantum neural network (hereafter called direct quantum neural network controller), because it has the simplest structure and is suitable for investigating the basic characteristics of quantum neural networks. To improve the learning performance of the quantum neural network and overcome the Jacobian problem, we utilize a real-coded genetic algorithm because it can conduct global optimization and does not require the Jacobian information during the training process of the direct quantum neural network controller. In Section 2, we present multi-layer quantum neural networks that use qubit neurons as its information processing unit and describe the direct quantum neural network controller. In Section 3, we evaluate the feasibility of using direct quantum neural network controllers by conducting computational experiments of controlling a discrete-time system and a nonholonomic robotic system (a two-wheeled robot).

2. Quantum neural network controller

2.1. Quantum neural networks with qubit neurons

In quantum computing, a qubit state $|\psi\rangle$ maintains a coherent superposition of states: $|\psi\rangle = a|0\rangle + b|1\rangle$, where $|0\rangle$ corresponds to bit 0 of the classical computers, $|1\rangle$ corresponds to bit 1 and a and b are complex numbers called probability amplitudes that satisfy $|a|^2 + |b|^2 = 1$. Moreover, two quantum logic gates, such as a 1-bit rotation gate and a 2-bit controlled NOT gate, are often utilized to operate a qubit state. Using the phase ϕ, we can express the operations of these gates by the quantum state. Specifically, the rotation gate, which is a phase-shift gate that transforms the phase of quantum states, can be expressed by $f(\phi_1+\phi_2) = f(\phi_1)f(\phi_2)$, where $f(\phi) = e^{i\phi}$ (i) is an imaginary unit). The controlled NOT gate, which is the phase reverse operation defined with respect to the controlled input parameter γ, can be expressed by $f(\pi\gamma/2-\phi)$, where $\gamma=1$ and $\gamma=0$ correspond to the reversal rotation and the non-rotation, respectively. By considering these gates, the state of the j-th qubit neuron model in the m-th sets, z_j^m, is defined as follows:

$$z_j^m = f\left\{\frac{\pi}{2}\delta_j^m - \arg\left[\sum_k f(\theta_{kj}^m) f(z_k^{m-1}) - f(\lambda_j^m)\right]\right\} \tag{1}$$

here z_k^{m-1} is the input from the k-th neuron in the $(m-1)$-th sets, δ_j^m is the reversal parameter corresponding to the controlled NOT gate, θ_{kj}^m is the phase parameters corresponding to the phase of the rotation gate and λ_j^m is a threshold parameter.

A multi-layer quantum neural network is designed by combining the qubit neurons in the layers. First, in the input layer (indicated by the superscript m of I), network inputs x_k in the range [0, 1] are converted into quantum states with phases in the range $[0, \pi/2]$. Next, outputs given by $z_k^I = f(\pi x_k/2)$ are fed into the neurons in the hidden layer (indicated by the superscript m of H). In the hidden and output layers, the outputs of the neurons are represented by Eq. (1). By considering the probability of the state in which $|1\rangle$ is observed from the j-th neuron of the output layer (indicated by the superscript m of O), the output from the network u_{NN_j} is represented follows:

$$u_{NN_j} = |\mathrm{Im}(z_j^O)|^2 \tag{2}$$

2.2. Learning using a real-coded genetic algorithm

The training of the multi-layer quantum neural network is carried out by searching for the optimal parameters θ_{kj}^m, δ_j^m and λ_j^m that minimize the following cost function:

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_p\sum_j \{u_{d_j}(p) - u_{NN_j}(\boldsymbol{w}, p)\}^2 \tag{3}$$

where u_{d_j} is the teaching signal for the j-th neuron of the p-th pattern. Moreover, vector $\boldsymbol{w}$ is composed of the parameters θ_{kj}^m, δ_j^m and λ_j^m ($m=I$, H, and L).

The real-coded genetic algorithm [17] is a powerful algorithm used to solve real parameter optimization problems of multi-modality, parameter dependency and ill-scale. We utilize it to find the vector $\boldsymbol{w}$ that minimizes the cost function $J(\boldsymbol{w})$. Thus, the values of the parameters of $\boldsymbol{w}$ are directly used as the gene parameters of an individual. The real-coded genetic algorithm is composed of a multi-parental crossover and a generation alternation model. For the multi-parental crossover, we use a real-coded ensemble crossover, which is a generalization of the unimodal normal distribution crossover. In particular, to avoid the asymmetry and bias of children distributions, the multi-parental crossover operation is assigned a probability distribution. In the real-coded ensemble crossover, new individuals (children), $\boldsymbol{\omega}_c$, are generated using multi-parental individuals, $\boldsymbol{\omega}_i$ ($i=1,2,\ldots,N+K$, where N is the dimension of the problem, K is defined in the range $[1, N_p-N]$ and N_p is the number of population), as follows:

$$\boldsymbol{\omega}_c = \boldsymbol{\omega}_g + \sum_{i=1}^{N+K} \xi_i(\boldsymbol{\omega}_i - \boldsymbol{\omega}_g) \tag{4}$$

here $\boldsymbol{\omega}_g$ indicates the centre of gravity of the parents, and ξ_i is a stochastic variable that follows the probability distribution $\varphi(0, \sigma_\xi^2)$, where $\sigma_\xi^2 = 1/(N+K)$. The generation alternation model used is the just generation gap model, which replaces parents with children in every generation. The fitness function is defined by the reciprocal of the cost function $J(\boldsymbol{w}_i)$, where i denotes the number of individuals.

2.3. Learning-type direct controller using quantum neural networks

To apply the quantum neural network to control systems, we use gain and shift factors to convert the outputs of the quantum neural network from the range [0, 1] into the range $[u_{min}, u_{max}]$; $u_j = c_0(u_{NN_j} - u_0)$, where u is the control input, c_0 is the gain factor and u_0 is the shift factor.

In general, two types of training are used in control systems: adaptive and learning. Adaptive training is a real-time process in which the plant/quantum neural network achieves the desired outputs within one trial. Learning training is an off-line process in which the plant/quantum neural network approximates the desired outputs after several trials. Because the quantum neural network is trained by the real-coded genetic algorithm in the off-line process, we consider a learning-type controller.

To simplify the design of the direct quantum neural network controller, we consider the following single-input single-output discrete-time plant as the target system to be controlled:

$$y(k+d_p) = F_p[y(k), \ldots, y(k-n_p+1), u(k), \ldots, u(k-m_p-d_p+1)] \tag{5}$$

where y is the plant output, u is the plant input, n_p and m_p are the plant orders, k is the sampling number, d_p is the dead time of the plant and $F_p()$ is a function that expresses the plant dynamics. The plant output $y(k)$ depends on the past plant inputs and outputs. The plant orders determine the period for which the plant output depends on them. This period is typically shorter than the trial period. Our design is based on the following assumptions: the upper limit of the orders and the dead time of the plant are