ELSEVIER

Computers and Chemical Engineering 24 (2000) 859–862

Computers & Chemical Engineering

www.elsevier.com/locate/compchemeng

PID neural networks for time-delay systems

Huailin Shu [a,*], Youguo Pi [b]

[a] *Department of Electrical and Mechanical Engineering, Guangzhou University, Guangzhou 510091, PR China*
[b] *Automation Engineering R & M Center, GuangDong Academy of Sciences, Guangzhou, GuangDong 510070, PR China*

Abstract

PID neural network (PIDNN) is a new kind of networks. It consists of three layers and its hidden layer's units are proportional (P), integral (I) and derivative (D) neurons. PIDNN's weights are adjusted by the back-propagation algorithms and it perform a perfect function in process control. In this paper, we introduce PIDNN structure and algorithm and give examples in which PIDNN is used to control time-delay systems. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: PID neural network; Neurocontrol; Time-delay system

1. Introduction

There are a lot of time-delay systems in industry processes but it is difficult to design the controllers for them because the time-delay property. These systems generally have larger overhead, longer adjusting time and are not stable. In classical control theory the Smith method can be used to construct controllers if the transfer function of the system has been known. But, the transfer function of a practical system is not easy to measure or to complete.

As is well known conventional PID controllers have many advantages so that they are most widely used in various fields of the industry, especially in the processes of chemical industry. Although PID controllers have strong abilities they are not suitable for the control of long time-delay systems, in which the P, I, and D parameters are difficult to chose.

Artificial neural networks can perform adaptive control through learning processes. But there are some problems, which should be solved in practice. The main problems are the slow learning speed, the long weight convergence time and uncertain property.

PID neural network (PIDNN) is a new kind of networks. It utilizes the advantages of both PID control and neural structure. It consists of proportional (P), integral (I) and derivative (D) neurons and its weights are adjusted by the back-propagation algorithms. It can control different systems through quick learning process and has perfect performances (Shu, 1997, 1998a,b,c; Shu & Li, 1998; Shu, 1999a,b.

The rest of the paper is organized as follows. Section 2 presents the structure of PIDNN. Section 3 specifies the algorithm of PIDNN. System simulation examples are introduced in Section 4, including the performance behavior comparing between PIDNN and conventional PID controllers. Finally, the conclusion is given in Section 5.

2. Structure of PIDNN

PIDNN consists of a 2-3-1 structure. It has three layers, which are input-layer, hidden-layer and output-layer. The input-layer has two neurons, the hidden-layer has three and the output-layer has only one. The neurons in the net are proportional (P) neuron, integral (I) neuron and derivative (D) neuron, respectively.

The input-layer has two P neurons, one receives system setting input and another connects system output. The hidden-layer has three different neurons, the first is P neuron, the second is I neuron and the third is D neuron. The output- layer only has one neuron which completes the control output duty. The network structure and the control system are shown in Fig. 1.

* Corresponding author.
E-mail addresses: hlshu@guangzu.edu.cn (H. Shu), youguopi@a.gis.sti.gd.cn (Y. Pi).

PII: S0098-1354(00)00340-9

3. Algorithm of PIDNN

3.1. Input layer

In input layer two neurons are P-neurons and their input–output functions are as follows:

$$x_i(k)=\begin{cases}1 & u_i(k)>1\\ u_i(k) & -1\le u_i(k)\le 1\\ -1 & u_i(k)<-1\end{cases} \quad (1)$$

where $i=1, 2$. k is computer sample time.

3.2. Hidden layer

In hidden layer the neurons inputs are

$$u'_j(k)=\sum_{i=1}^{2} w_{ij}\cdot x_i(k) \quad (2)$$

where $j=1, 2, 3$ and w_{ij} are the net weight from input layer to hidden layer.

The input–output functions of the neurons in the hidden layer are different from each other. They are proportional (P) function, integral (I) function and derivative (D) function so that they are named as P-neuron, I-neuron and D-neuron respectively. The function of P-neuron is the same as that of the input layer, as follows:

$$x'_1(k)=\begin{cases}1 & u'_1(k)>1\\ u'_1(k) & -1\le u'_1(k)\le 1\\ -1 & u'_1(k)<-1\end{cases} \quad (3)$$

The function of I-neuron is

$$x'_2(k)=\begin{cases}1 & x'_2(k)>1\\ x'_2(k-1)+u'_2(k) & -1\le x'_2(k)\le 1\\ -1 & x'_2(k)<-1\end{cases} \quad (4)$$

The function of D-neuron is

$$x'_3(k)=\begin{cases}1 & x'_3(k)>1\\ u'_3(k)-u'_3(k-1) & -1\le x'_3(k)\le 1\\ -1 & x'_3(k)<-1\end{cases} \quad (5)$$

3.3. Output layer

The output layer only has one neuron. The input of the neuron is

$$u''(k)=\sum_{j=1}^{3} w_{jo}\cdot x''_{jo}(k) \quad (6)$$

where w_{jo} are net weight from hidden layer to output layer and its input-output function is proportional (P) function as follows:

$$x_o(k)=\begin{cases}1 & u_o(k)>1\\ u_o(k) & -1\le u_o(k)\le 1\\ -1 & u_o(k)<-1\end{cases} \quad (7)$$

3.4. Back-propagation algorithm

The aim of the PIDNN controller is to minimize

$$J=\sum_{h=1}^{n} E_h=\frac{1}{m}\sum_{h=1}^{n}[r(k)-y(k)]^2 \quad (8)$$

where $y(k)$ is system output and $r(k)$ is the system input.

For the aim we have a back-propagation algorithm of PIDNN. We use gradient method to change the weights of PIDNN. After n_0 training and studying steps, the weights from hidden layer to output layer are

$$w'_{jo}(n_0+1)=w'_{jo}(n_0)-\eta_j\frac{\partial J}{\partial w'_{jo}} \quad (9)$$

where η_j is step, and

$$\frac{\partial J}{\partial w'_{jo}}=\frac{\partial J}{\partial E_h}\frac{\partial E_h}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial x''_o}\frac{\partial x''_o}{\partial u''_o}\frac{\partial u''_o}{\partial w'_{jo}} \quad (10)$$

From Eq. (8),

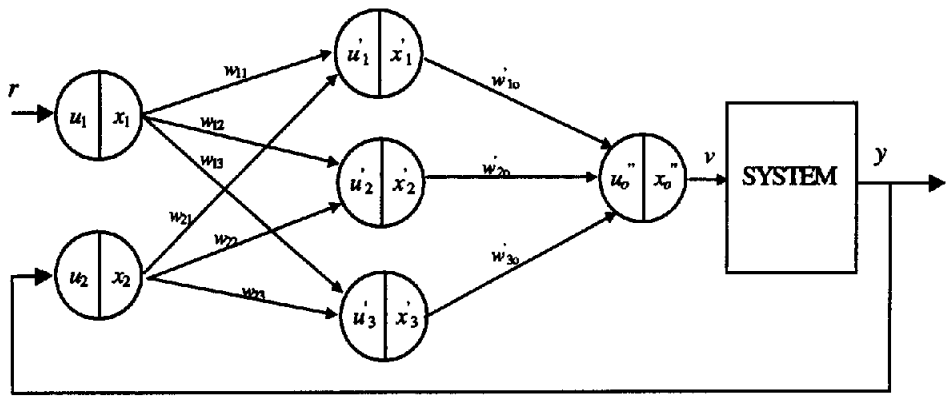

Fig. 1. Structure of PIDNN.

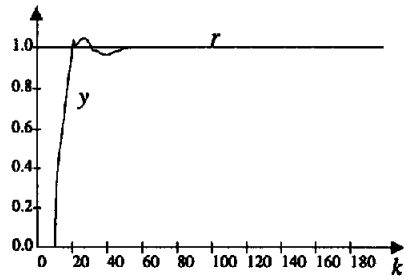

Fig. 2. One-step time-delay system responses.

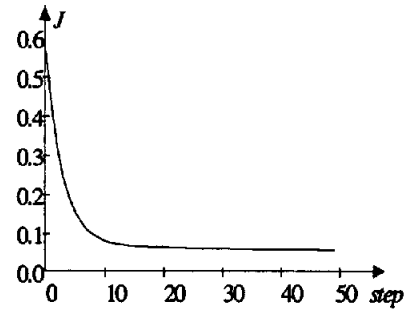

Fig. 3. Constringency curve of aim J of one-step time-delay system.

$$\frac{\partial J}{\partial E_k}\frac{\partial E_k}{\partial y} = -\frac{2}{m}\sum_{k=1}^{m}[r(k)-y(k)] \tag{11}$$

and

$$\frac{\partial v}{\partial x_o''}\frac{\partial x_o''}{u_o''} = 1 \tag{12}$$

From Eq. (6)

$$\frac{\partial u_{jo}''}{\partial w_{jo}'} = x_j'(k) \tag{13}$$

In Eq. (10), $\partial y/\partial v$ cannot be directly decided because we do not know the transfer function of the system. For the solution, we use the following equation

$$\frac{\partial y}{\partial v} \approx \frac{\Delta y}{\Delta v} = \frac{y(k+1)-y(k)}{v(k)-v(k-1)} \tag{14}$$

Thus from Eqs. (11–14), we can write

$$\frac{\partial J}{\partial w_{jo}'} = -\frac{2}{m}\sum_{k=1}^{m}[r(k)-y(k)]\frac{y(k)-y(k-1)}{v(k)-v(k-1)}x_j'(k)$$

$$= -\sum_{k=1}^{m}\delta_j(k)x_j'(k) \tag{15}$$

From input-layer to hidden-layer the connect weights are

$$w_{ij}(n_0+1) = w_{ij}(n_0) - \eta_i\frac{\partial J}{\partial w_{ij}} \tag{16}$$

where η_i is step and

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial E_k}\frac{\partial E_k}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial x_o''}\frac{\partial x_o''}{\partial u_o''}\frac{\partial u_o''}{\partial x_j'}\frac{\partial x_j'}{\partial u_j'}\frac{\partial u_j'}{\partial w_{ij}} \tag{17}$$

where

$$\frac{\partial J}{\partial E_k}\frac{\partial E_k}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial x_o''}\frac{\partial x_o''}{\partial u_o''} = -\sum_{k=1}^{m}\delta_j(k) \tag{18}$$

which come from Eq. (15), and

$$\frac{\partial u_o''}{\partial x_j'} = w_{jo}' \tag{19}$$

from Eq. (6) and

$$\frac{\partial x_j'}{\partial u_j'} \approx \frac{\Delta x_j'}{\Delta u_j'} = \frac{x_j'(k)-x_j'(k-1)}{u_j'(k)-u_j'(k-1)} \tag{20}$$

and

$$\frac{\partial u_j}{\partial w_{ij}} = x_i(k) \tag{21}$$

from Eq. (2). Thus the following equation can be written from Eqs. (18–21).

$$\frac{\partial J}{\partial w_{ij}} = -\sum_{k=1}^{m}\delta_j(k)\frac{x_j'(k)-x_j'(k-1)}{u_j'(k)-u_j'(k-1)}x_i(k) = -\sum_{k=1}^{m}\delta_i(k)x_i(k) \tag{22}$$

4. Examples

4.1. One-step system

A one-step time-delay system is described by the following function.

$$y(k+1) = 0.368y(k) + 0.632v(k-10) \tag{23}$$

and system input is

$$r(k) = 1(k) \tag{24}$$

PIDNN is the controller and let $w_{1j}(0) = +1$, $w_{2j}(0) = -1$, where $j = 1, 2, 3$, and let $w_{jo}'(0) = 0.1$. After 50 learning steps, the system responses are shown in Fig. 2.

From Fig. 2 we know that the system has perfect performance. The dynamic response is quick and there is little over-adjust. The static error is zero. The system is stable. The PIDNN can complete the one-step time-delay system control duty.

The constringency curve of aim J, Eq. (8), of this one-step time-delay system is shown in Fig. 3.

From Fig. 3, it is obvious that the PIDNN has monotonous constringency property and the constringency time is very short.

4.2. Two-step system

A two-step system is described as the following function.

$$y(k+1) = 1.368y(k) - 0.368y(k-1) + 0.0092v(k-10) + 0.066v(k-11) \quad (25)$$

The system input is Eq. (24) and PIDNN is used as controller. After 50 learning steps, the system responses is shown in Fig. 4.

Fig. 4 proved that PIDNN can adapt different systems through learning process. The control system, also has better performance for the long time-delay two-step object.

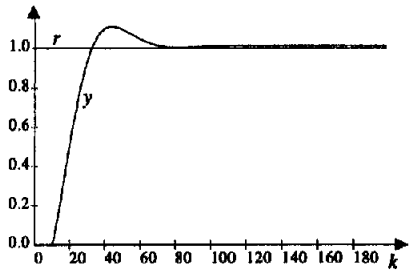

Fig. 4. Two-step time-delay system.

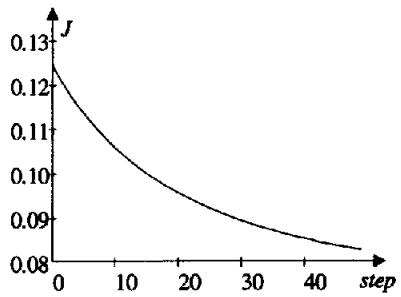

Fig. 5. Constringency curve of aim J of two-step time-delay system.

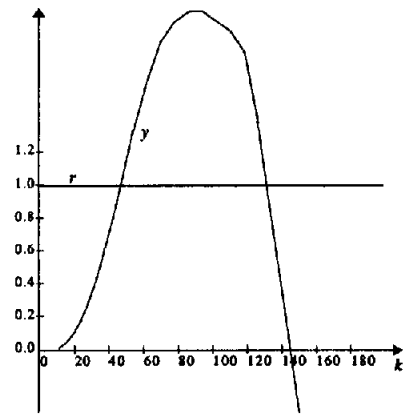

Fig. 6. Conventional PID control two-step time-delay system responses.

The constringency curve of aim J, Eq. (8), of this system is shown in Fig. 5. This monotonous reducing curve proved that PIDNN can finish the learning process quickly.

4.3. Performance of conventional PID controller

As a comparison, we use a conventional PID controller to control the two-step time-delay system as Eq. (25). The PID controller function is Eq. (26) and the system step responses are shown in Fig. 6.

$$u(k) = K_P e(k) + K_I \sum_{i=1}^{k} e(i) + K_D[e(k) - e(k-1)]$$

$$= 0.0115e(k) + 0.00575 \sum_{i=1}^{k} e(i) + 0.05175[e(k) - e(k-1)] \quad (26)$$

The result above tells us that conventional PID controller cannot suit to control the long time-delay system. The system is unstable. The result proved that PIDNN has much better properties than conventional PID controller.

5. Conclusions

PIDNN is a multilayered neural network and its structure is simple. PIDNN has abilities to control different time-delay system and has perfect performance. Using PIDNN we needn't measure or calculate the system parameters. The whole adjusting process is completed through self-learning and adaptive process. PIDNN has short convergence time and quick learning speed and it can be used in practical process.

References

Shu, H. L. (1997). Study on the neural PID network based cascade control system. *Automation & Instrumentation (China), 5*, 5–7.

Shu, H. L. (1998a). Self-study decouple PID neural network control systems. *Computation Technology and Automation (China), 7*, 43–46.

Shu, H.L. (1998b). The analyze of PID neurons and PID neural networks. *Proceedings of '98 Chinese Control Conference.* (pp. 607–613). *vol.* 9. China:Lingbo.

Shu, H. L. (1998c). PID neural network for decoupling control of strong coupling multivariable time-delay systems. *Control Theory and Application (China), 15*(6), 920–924.

Shu, H. L., & Li, Z. (1998). The multivariable decouple control system based on multilayer network with PID neurons. *Process Automation Instrumentation (China), 19*(3), 24–27.

Shu, H. L. (1999a). Analysis of PID neural network multivariable control systems. *Acta Automatica Sinica (China), 25*(1), 105–111.

Shu, H.L. (1999b). PID neural network control for complex systems. *Proceedings of the International Conference on Computational Intelligence for Modeling, Control and Automation.* (166–171). Amsterdam: IOS Press.