Proceedings of the American Control Conference
Philadelphia, Pennsylvania • June 1998

B-Spline Recurrent Neural Network and Its Application to Modelling of Non-linear Dynamic Systems

C. W. Chan*, K. C. Cheung*, Hong Jin†, H. Y. Zhang†

* Department of Mechanical Engineering, University of Hong Kong
Pokfulam road, Hong Kong, Email: mechan@hkucc.hku.hk
† Department of Automatic Control, Beijing University of Aeronautics and Astronautics
Beijing 100083, P. R. China, Email: buaa301@mimi.cnc.ac.cn
The authors are currently with the Department of Mechanical Engineering, University of Hong Kong

Abstract: A new recurrent neural network based on B-spline function approximation is presented. The network can be easily trained and its training converges more quickly than that for other recurrent neural networks. Moreover, an adaptive weight updating algorithm for the recurrent network is proposed. It can speed up the training process of the network greatly and its learning speed is more quickly than existing algorithms, e.g., back-propagation algorithm. Examples are presented comparing the adaptive weight updating algorithm and the constant learning rate method, and illustrating its application to modelling of nonlinear dynamic system.

Keywords: Recurrent Neural Network, B-Spline network, Adaptive Learning Algorithm, State Estimation, System Modelling.

1. Introduction

Recurrent neural networks(*RNN*) have proved to be valuable tools and have been studied extensively in modelling and control of non-linear dynamic systems, e.g., the designs of non-linear dynamic system identifier and controller[1][2][3]. When the state variables of non-linear dynamic system are not accessible, the identification of the system and the system states[4](i.e., observer's problem), is substantially more complex than in the case where the states are accessible.

The critical issues in the application of recurrent networks are the choice of the network architecture, i.e., number of network elements, and the location of feedback loops, or *RNN* based on Gaussian radial basis function[5] with multilayer feedforward neural network, etc.[1][6] and the development of suitable training procedures. A single or multiple hidden layer networks with feedback loop induces the dynamic complexity while the number of hidden layer elements are associated with the degree of non-linearity. In addition, the training of recurrent network is of much more complex than the training of feedforward networks.

Another problem associated with recurrent networks is that the convergence of learning can be very slow and training errors are not always guaranteed to reduce to previously defined tolerances. By using constructive training methods, it is possible that the neural network could possibly build itself little by little, and speed up the whole training process. So a good architecture and quickly learning algorithm are important for us to apply *RNN* better.

In this paper, a new recurrent neural network architecture is proposed, which is based on B-Spline Neural Network(*BSNN*)[7] and denoted by *BSRNN*(B-Spline Recurrent Neural Network). Basis spline function has several advantages. For example, it has a good ability of approximation; it only needs local adjustment of weights for every input; it needs less computation and storage than other basis functions(e.g., Gaussian function and Berstein function etc.); moreover, the derivative of basis spline function can be readily obtained. Further, the training of the *BSNN* is more quickly than other networks(e.g., multilayer feedforword network) and has been applied for guidance[8], fault detection and isolation[9], Kalman filter initialisation[10]. Here we discuss the recurrent networks based on *BSNN*. Our aim is to model nonlinear dynamic systems better and faster by applying the basis spline function for training recurrent networks. It is, to our knowledge, that few works have focused on *BSRNN*s.

Other parts of this paper are arranged as follows:

In 2nd and 3rd sections, we briefly discuss *BSNN* and *BSRNN* respectively, and give the *BSRNN* models and their architectures for two kinds of non-linear systems. In 4th and 5th sections, we discuss the learning algorithm of weights and the determination of the amounts of weights to update. In 6th section, an adaptive weight updating algorithm is proposed. This adaptive updating method can increase the learning speed greatly, make the learning process converge quickly and overcome the ad hoc choice of learning rate in weight updating. In 7th section, an example is given to show the application of *BSRNN* to the modelling of nonlinear dynamic systems. Finally, we give the conclusion about *BSRNN*.

2. B-Spline neural network

Let $x(t) \in R^n$ be the input vector of the network. By using the operation of tensor product, we can calculate the multivariate basis function vector, i.e., transformed input vector[7], $s(x) = (s_1(x) \ \cdots \ s_q(x))'$ or so-called box spline[11] of x and the jth multivariate B-spline function is denoted by $s_j(x)$. The number q of multivariate B-spline functions is dependent on the order of basis spline function and the number of inner knots of the interval given for every component of x. The space $\{s(x)\}$ is so-called transformed input space. These q multivariate B-spline functions can be considered as net inputs of the hidden layer, and the output of network $\hat{y}(t)$ (Fig.1) is,

$$\hat{y}(t) = \sum_{j=1}^{q} w_j s_j(x(t)) \tag{1}$$

where w_j is the weight value($j=1,\ldots,q$).

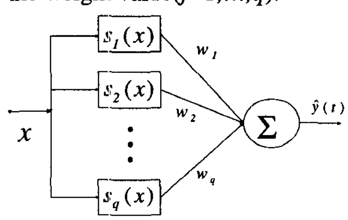


Fig.1: *BSNN*(B-Spline Neural Network)

When x is a univariate variable, equation (1) can be considered as a B-spline curve[12], otherwise, as a B-spline surface[13], where $(w_1,\cdots,w_q)$ can be also considered as control points[14], so the process of adjusting the weights is that of adjusting the control points. The main advantage of the B-spline formulation over other curve fitting(e.g., the Bezier curve) is local control of the curve shape, i.e., the shape of the curve changes only in the vicinity of few changed control points[12]. As in the training of *BSNN*, only a few weights need to be adjusted, much computation time can be saved.

The p-order B-spline function $s_j(x)$ can be determined by using following B-spline recurrence relationship[13]:

$$s_{i,p}(x) = \frac{x-\tau_i}{\tau_{i+p}-\tau_i} s_{i,p-1}(x) + \frac{\tau_{i+p+1}-x}{\tau_{i+p+1}-\tau_{i+1}} s_{i+1,p-1}(x) \tag{2}$$

$$s_{i,0}(x) = \begin{cases} 1 & \tau_i \le x < \tau_{i+1} \\ 0 & otherwise \end{cases} \quad \text{for i=1,}\ldots\text{,n} \tag{3}$$

where τ_i is the i-th knot.

3. B-Spline recurrent neural network

In *BSNN*, the training is undertaken by supervising learning, i.e., the input of network is known and output of network need to be compared to the actual value which is accessible. But for some applications, only output of a system can be obtained. When we need to know the inner state of a system, recurrent neural network is a good choice. In this section, two kinds of recurrent neural network based on *BSNN* are described for two classes of non-linear models. These two models are given as follows:

Model I:

$$x(i+1) = f(x(i), x(i-1), \cdots x(i-n+1); \\ u(i), u(i-1), \cdots, u(i-m+1)) \tag{4}$$

Model II:

$$\begin{cases} x_{n-r}(i+1) = f_{n-r}(x_1(i),\cdots,x_{n-r-1}(i),x_{n-r}(i),\cdots,x_n(i); \\ \qquad u(i),u(i-1),\cdots,u(i-m+1)) \\ \quad \vdots \\ x_n(i+1) = f_n(x_1(i),\cdots,x_{n-r-1}(i),x_{n-r}(i),\cdots,x_n(i); \\ \qquad u(i),u(i-1),\cdots,u(i-m+1)) \end{cases} \tag{5}$$

where r is given.

For these two models, we can use *BSNN* and *BSRNN* to implement. For example of Model I, its *BSNN* and *BSRNN* models can be described by following equations:

BSNN:

$$\hat{x}(i+1) = BSNN(x(i), x(i-1), \cdots x(i-n+1); \\ u(i), u(i-1), \cdots, u(i-m+1)) \tag{6}$$

BSRNN(Fig.2):

$$\hat{x}(i+1) = BSRNN(\hat{x}(i), \hat{x}(i-1), \cdots \hat{x}(i-n+1); \\ u(i), u(i-1), \cdots, u(i-m+1)) \tag{7}$$

(6) and (7) are called series-parallel model and parallel model respectively [1].

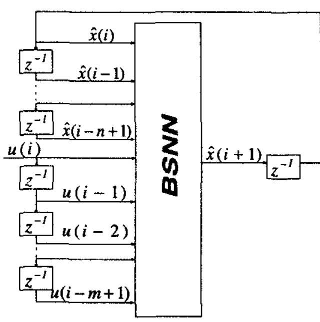


Fig.2: ***BSRNN*** of Model I

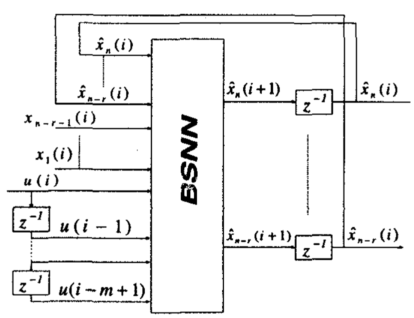


Fig.3: ***BSRNN*** of Model II

Fig.3 gives the *BSRNN* of Model II. In these two *BSRNNs*, the output of network is fed back as an input. In this case, the transformed input vector is dependent on the weight values of the previous step, which influences the updating of the weights. This issue will be discussed next.

4. Learning algorithm for *BSRNN*

In training *BSRNN*, the steepest descent gradient algorithm can be used to adjust weights of network. For example, in Model I, a natural performance criterion for a recurrent network is the sum of the squared of errors between the target sequence and the outputs of the network,

$$E = \sum_{i=1}^{m} (y(i) - \hat{y}(i))^2 \qquad (8)$$

where *y(i)* is the real output of system(4) at time i, $\hat{y}(i)$ is the output of the *BSRNN* for estimating *y(i)*.

The least-squares approximation to *y(i)* is to minimise *E* by finding the optimal weight values $\{w_i, i = 1, \cdots, q\}$. A typical learning rule is the so-called dynamic back propagation algorithm[15] and the weights are adjusted by,

$$w^{new} = w^{old} - \eta \cdot \partial E / \partial w \big|_{w^{old}} \qquad (9)$$

where η is the learning rate or step size. Weight adjustments can be performed at each time or in a batch mode.

5. Determination of the partial derivatives with respect to the weights

The partial derivative of (8) with respect to w_k can be written as follows:

$$\frac{\partial E}{\partial w_k} = -2\sum_{i=1}^{m} (y(i) - \hat{y}(i)) \frac{\partial \hat{y}(i)}{\partial w_k} \qquad (10)$$

where $k=1,\ldots,q$. From (1),

$$\frac{\partial \hat{y}(i)}{\partial w_k} = s_k(x(i)) + \sum_{j=1}^{q} w_j \frac{\partial s_j(x(i))}{\partial w_k} \qquad (11)$$

Because the transformed input vector of the *BSRNN* is dependent on the weights of the network, in contrast to that of the *BSNN* or other non-recurrent radial basis function neural networks, the second term in the right hand side of (11) is not equal to zero. It needs to know how large the effect caused by this factor is when calculating (10). In matrix formation, (11) becomes

$$\left(\frac{\partial \hat{y}(1)}{\partial w} \cdots \frac{\partial \hat{y}(m)}{\partial w} \right) = \begin{bmatrix} s_1(x(1)) & \cdots & s_1(x(m)) \\ \cdots & \cdots & \cdots \\ s_q(x(1)) & \cdots & s_q(x(m)) \end{bmatrix} + \sum_{j=1}^{q} w_j \begin{bmatrix} \frac{\partial s_j(x(1))}{\partial w_1} & \cdots & \frac{\partial s_j(x(m))}{\partial w_1} \\ \cdots & \cdots & \cdots \\ \frac{\partial s_j(x(1))}{\partial w_q} & \cdots & \frac{\partial s_j(x(m))}{\partial w_q} \end{bmatrix} \qquad (12)$$

where the determination of $\partial s_j(x(i))/\partial w_k$ is very complex. As the 2nd term in the right hand side of equation (12) requires a heavy calculation load, Considerable saving in computing time can be made if this term is ignored. It is shown in the following example that ignoring this term affects only the rate of convergence of the estimates of the weights.

Example 1: Consider a model as follows:

$$x(k) = 0.5\sin(x(k-1)) - 0.1x(k-2) + u(k) \qquad (13)$$

$$y(k) = 100x(k) \qquad (14)$$

where *u(k)* is zero-mean white noise with the standard deviation of $\sigma = 0.01$. Given $x(0) = 0, x(1) = 1$. Fig.4 gives curves of errors performance with or without considering the second term of (12). (a) the number of inner knots $N_k = 1$; (b) inner knots $N_k = 2$. It shows that consideration of the 2nd term in (12) has no obvious advantage.

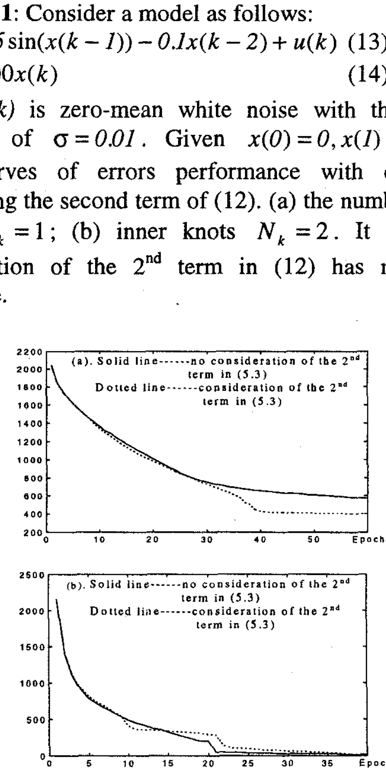


Fig.4: Performance indexes

6. Adaptive Weight Updating Algorithm

The proposed learning algorithm is based on the gradient decent method, and is similar to the backpropagation algorithm. Hence it suffers from the same drawback as that in the back propagation algorithm, i.e., slow learning rate. This slowness arises largely from the lack of suitable methods in selecting the step size η. It is that if η is sufficiently small, the overall learning process will be stable[16], but the cost for small η is a very long time for convergence. But if the step size is chosen too large, there is a possibility of divergence.

In this section, we give an optimal value of learning rate for both *BSNN* and *BSRNN*. Firstly, we discuss the determination of learning rate of *BSNN*.

Let $Y=[y(1) \cdots y(m)]'$, $\hat{Y}^{(k)} = [y^{(k)}(1) \cdots y^{(k)}(m)]'$, $\Delta\hat{Y}^{(k)} = Y - \hat{Y}^{(k)}$, $S = [s(1) \cdots s(m)]'$, where k denotes the training epoch time, $s(i) = [s_{i1} \cdots s_{iq}]'$ is composed of q B-spline function values of the i-th input vector. An adaptive learning step size is derived as

$$\hat{\eta} = \| S'\Delta\hat{Y}^{(k-1)} \|^2 / \| S'\Delta\hat{Y}^{(k-1)} \|^2_{S'S} \qquad (15)$$

where $\|x\|_A^2 := x'Ax$. The derivation is given in the Appendix. As for the *BSRNN*, if the matrix *S* is used as an approximation instead of the right hand side of (12), then (15) also applies. The following example compares the adaptive weight updating algorithm with ordinary constant learning rate algorithm by using *BSNN*.

Example 2: Consider $y = \sin^2 x$.

Since no information on the learning rate is available, we tried different learning rates and find $\eta = 0.01$ is best. Table 1 gives epoch number (N_e) and its final performance index (E) for different number N_k of inner knots and a constant learning rate $\eta = 0.01$ by using 1st-order B-spline function to train the *BSNN*, the samples of variable $x = 0{:}0.1{:}2\pi$, and the initial weights were zero. The condition of stopping learning is: $E < eps$ or $(E(k)-E(k-1))/E(k-1) < eps$, where $eps = 1.e-6$.

Table 2 gives the results under the same condition except the adaptive learning rate given by (15) is used and the initial learning rate is $\eta = 0.01$.

Table 1: Using a constant learning rate $\eta = 0.01$

N_k	N_e	E
1	73	7.8873
5	216	0.5093
10	379	0.0245
20	741	0.0022
30	1424	2.233e-4
40	2581	3.600e-5

Table 2: Using the adaptive learning rate (15)

N_k	N_e	E
1	7	7.8873
5	16	0.5093
10	18	0.0245
20	20	0.0022
30	27	2.2327e-4
40	34	3.5986e-5

Clearly, the learning is much faster when the adaptive learning rate is used.

7. Example

Example 3: Modelling of ARMA models

Model:

$$x(k+1) = 0.01x(k) + 0.4\sin(3x(k)) + 0.5u(k)$$
$$y(k) = \sin(x(k)) + 0.1x(k-1)$$

Let $x_1(k) = x(k)$, $x_2(k) = x(k-1)$, the architecture of *BSRNN* is shown in Fig.5. Simulation conditions: initial input $x_1(1) = 1$, $x_2(1) = 1$, initial learning rate $\eta(0) = 0.95$, $eps = 0.001$, $N_k = 5$ for every variable of input, $u(i) = \sin(2\pi i/10) + \sin(2\pi i/25)$, sample number m=100 and 1st-order B-spline function for training.

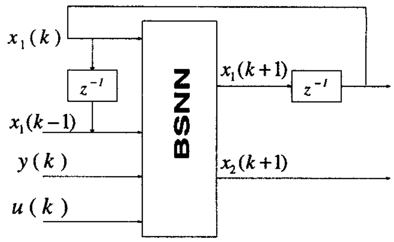
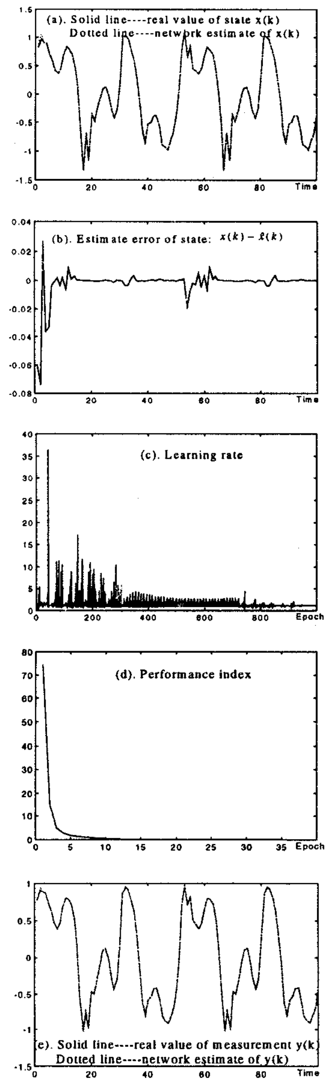


Fig.5: Architecture of ***BSRNN***

Fig.6 gives some simulation results. (a) real state with its estimate of network; (b) estimate error curve of state; (c) learning rate curve; (d) performance index curve; (e) real measurement with its estimate; (f) measurement estimate error curve. When epoch time $N_e = 720$, *E=0.00099942*, its corresponding learning rate is $\eta = 2.84775$.



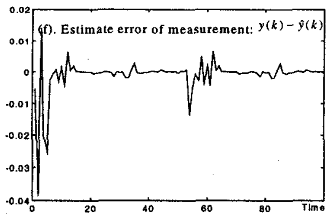


Fig.6

8. Conclusion

Recurrent neural networks based on B-spline neural networks are proposed in this paper. In order to decrease the computational burden of the learning algorithm, an adaptive learning rate of updating weights is used. Simulation results have shown that a great reduction of learning time is achieved.

Acknowledgment

The work was supported by the Research Grants Council of Hong Kong.

Reference

[1]. Narendra K. S. and K. Parthasarathy(1990), Identification and Control of Dynamic Systems Using Neural Networks, IEEE Trans. on Neural Networks, Vol.1, No.1, pp4-27.

[2]. Yip P. P. C. and Y. H. Pao(1994), A Recurrent Network Net Approach to One -Step Ahead Control Problems, IEEE Trans. on Systems, Man, and Cybernetics, Vol.24, No.4, pp678-683.

[3]. Park Y. M., M. S. Choi and K.Y. Lee, An Optimal Tracking Neuro-Controller for Nonlinear Dynamic Systems, IEEE Trans. on Neural Networks, Vol.7, No.5, 1996, 1099-1110.

[4]. Levin A. U. and K. S. Narendra(1996), Control of Nonlinear Dynamical Systems Using Neural Networks---- Part II: Observability, Identification, and Control, IEEE Trans. On Neural Networks, Vol.7, No.1, pp30-42.

[5]. Obradovic D, On-line Training of Recurrent Neural Networks with Continuous Topology Adaptation, IEEE Trans. on Neural Networks, Vol.7, No.1996, 222-228.

[6]. Lee S. W. and H. H. Song, A New Recurrent Neural - Network Architecture for Visual Pattern Recognition, IEEE Trans. on Neural Networks, Vol.8, No.2, 1997, pp331-340.

[7]. Brown M. and C. Harris(1994), Neurofuzzy Adaptive Modelling and Control, Prentice Hall International (UK) Limited.

[8]. Doyle R. S. and C. J. Harris(1996), Multi-sensor data fusion for helicopter guidance using neuro-fuzzy estimation algorithms, The Aeronautical Journal of the Royal Aeronautical Society, June/July, pp240-251.

[9]. H. Benkhedda and R. J. Patton, B-spline Network Integrated Qualitative and Quantitative Fault Detection, IFAC 13th Triennial World Congress, San Francisco, USA, 1996.

[10]. Roberts J. M., D. J. Mills, D. Charnley and C. J. Harris(1995), Improved Kalman Filter Initialisation using Neurofuzzy Estimation, 1995/6 Research Journal: Image, Speech and Intelligent Systems, The University of Southampton, pp10.1-10.10

[11]. Chui C. K., Multivariate Spline, Society for Industrial and Applied Mathematics, 1988.

[12]. Newman W. M. and R. F. Sproull(1979), Principles of Interactive Computer Graphics, McGraw-Hill, Inc.

[13]. Piegl L. and W. Tiller, The NURBS Book, Springer-Verlag Berlin Heidelberg, 1995.

[14]. Wang C. H., W. Y. Wang, T. T. Lee and P. S. Tseng, Fuzzy B-spline Membership Function (BMF) and Its Applications in Fuzzy-neural Control, IEEE Trans. on Systems, Man, and Cybernetics, Vol.25, No.5, 1995, pp841-851.

[15]. Levin A. U. and K. S. Narendra(1993), Control of Nonlinear Dynamical Systems Using Neural Networks: Controllability and Stabilization, IEEE Trans. On Neural Networks, Vol.4, No.2, pp192-206.

[16]. Narendra K.S. and S. M. Li, Neural Networks in Control System, Neural Networks Theory, Technology, and Applications (Ed. By P. K. Simpson), The Institute of Electrical and Electronics Engineers ,Inc., New York City, New York, 1996.

Appendix: Derivation of (15)

The performance index at the k-th iteration can be written as follows:

$$E(k)=\sum_{i=1}^{m}(y(i)-\hat{y}^{(k)}(i))^2 \qquad \text{(A1)}$$

For the *BSNN*, the 2nd term of right hand side in (11) is equal to zero, so (9) can be rewritten as follows

$$w(k)=w(k-1)+\eta S'\Delta\hat{Y}^{(k-1)} \qquad \text{(A2)}$$

Here we have not considered the coefficient 2 in (10). Because $\hat{y}^{(k)}(i)=s'(i)w(k)$ and noting that

$$\hat{y}^{(k)}(i)-\hat{y}^{(k-1)}(i)=\eta s'(i)S'\Delta\hat{Y}^{(k-1)}$$

$$S'\Delta\hat{Y}^{(k-1)}=\sum_{i=1}^{m}s(i)\Delta\hat{y}^{(k-1)}(i)$$

and $S'S=\sum_{i=1}^{m}s(i)s'(i)$, we have

$$\begin{aligned}E(k)-E(k-1)&=\sum_{i=1}^{m}[-2y(i)(\hat{y}^{(k)}(i)-\hat{y}^{(k-1)}(i))\\&\quad+(\hat{y}^{(k)}(i))^2-(\hat{y}^{(k-1)}(i))^2]\\&=\sum_{i=1}^{m}[-2(y(i)-\hat{y}^{(k-1)}(i))(\hat{y}^{(k)}(i)-\hat{y}^{(k-1)}(i))\\&\quad+(\hat{y}^{(k)}(i)-\hat{y}^{(k-1)}(i))^2] \qquad \text{(A3)}\\&=-2\eta\sum_{i=1}^{m}(\Delta\hat{y}^{(k-1)}(i)\cdot s'(i)S'\Delta\hat{Y}^{(k-1)})\\&\quad+\eta^2\sum_{i=1}^{m}(s'(i)S'\Delta\hat{Y}^{(k-1)})^2\\&=-2\eta\|S'\Delta\hat{Y}^{(k-1)}\|^2+\eta^2\|S'\Delta\hat{Y}^{(k-1)}\|_{S'S}^2\end{aligned}$$

Differentiating this quadratic equation with respect to parameter η, (15) can be easily derived.