<table>
<thead>
<tr>
<th>Title</th>
<th>B-spline recurrent neural network and its application to modelling of non-linear dynamic systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Chan, CW; Cheung, KC; Jin, H; Zhang, HY</td>
</tr>
<tr>
<td>Issued Date</td>
<td>1998</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10722/46636">http://hdl.handle.net/10722/46636</a></td>
</tr>
<tr>
<td>Rights</td>
<td>©1998 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.; This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.</td>
</tr>
</tbody>
</table>
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 feedforward 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 BSRNNs.

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 \( x(t) \in \mathbb{R}^n \) 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 \( j \)th 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 \( y(t) \) (Fig. 1) is,

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

(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 \((w_1,\ldots,w_q)\) 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]:

\[ s_{i,p}(x) = \frac{x - \tau_i}{\tau_{i+p} - \tau_i} s_{i,p+1}(x) + \frac{\tau_{i+p} - x}{\tau_{i+p+1} - \tau_i} s_{i+1,p}(x) \]

(2)

\[ s_{i,0}(x) = \begin{cases} 1 & \tau_i \leq x < \tau_{i+1} \\ 0 & \text{otherwise} \end{cases} \]

for \( i=1,\ldots,n \)

(3)

where \( \tau_i \) 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:

\[ \begin{align*}
    x(t+1) &= f(x(t), x(t-1), \ldots, x(t-n+1)); \\
    u(t), u(t-1), \ldots, u(t-m+1))
\end{align*} \]

(4)

Model II:

\[ \begin{align*}
    x_{n-r}(t+1) &= f_{n-r}(x_{t}, x_{t-1}, \ldots, x_{t-n+r+1}); \\
    u(t), u(t-1), \ldots, u(t-m+1))
\end{align*} \]

(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:

\[ \begin{align*}
    \hat{x}(t+1) &= \text{BSNN}(x(t), x(t-1), \ldots, x(t-n+1)); \\
    u(t), u(t-1), \ldots, u(t-m+1))
\end{align*} \]

(6)

BSRNN(Fig. 2):

\[ \begin{align*}
    \hat{x}(t+1) &= \text{BSRNN}(\hat{x}(t), \hat{x}(t-1), \ldots, \hat{x}(t-n+1)); \\
    u(t), u(t-1), \ldots, u(t-m+1))
\end{align*} \]

(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 errors between the target sequence and the outputs of the network,

$$E = \sum_{i=1}^{n} (y(i) - \hat{y}(i))^2$$  \hspace{1cm} (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 \(w_i, i = 1, \ldots, q\). A typical learning rule is the so-called dynamic back propagation algorithm[15] and the weights are adjusted by,

$$w_{new} = w_{old} - \eta \frac{\partial E}{\partial w_k}$$  \hspace{1cm} (9)

where \(\eta\) 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:

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

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

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

$$\begin{bmatrix}
\frac{\partial s_k(x(1))}{\partial w_k} & \ldots & \frac{\partial s_k(x(m))}{\partial w_k} \\
\frac{\partial s_q(x(1))}{\partial w_k} & \ldots & \frac{\partial s_q(x(m))}{\partial w_k}
\end{bmatrix} = \begin{bmatrix}
s_k(x(1)) & \ldots & s_k(x(m)) \\
s_q(x(1)) & \ldots & s_q(x(m))
\end{bmatrix}$$

$$+ \sum_{j=1}^{q} w_j \begin{bmatrix}
\frac{\partial s_j(x(1))}{\partial w_k} & \ldots & \frac{\partial s_j(x(m))}{\partial w_k} \\
\frac{\partial s_q(x(1))}{\partial w_k} & \ldots & \frac{\partial s_q(x(m))}{\partial w_k}
\end{bmatrix}$$

(12)

where the determination of \(\frac{\partial s_j(x(m))}{\partial w_k}\) is very complex.

As the 2\(^{nd}\) 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:

$$x(k) = 0.5 \sin(x(k-1)) - 0.1 x(k-2) + u(k)$$  \hspace{1cm} (13)

$$y(k) = 100 x(k)$$  \hspace{1cm} (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 \(N_1 = 1\); (b) inner knots \(N_2 = 2\). It shows that consideration of the 2\(^{nd}\) term in (12) has no obvious advantage.

![Fig.4: Performance indexes](image)

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 backpropagation algorithm, i.e., slow learning rate. This slowness arises largely from the lack of suitable methods in selecting the step size \(\eta\). It is that if \(\eta\) is sufficiently small, the overall learning process will be stable[16], but the cost for small \(\eta\) 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 \(Y = [y(1) \ldots y(m)]', \hat{Y}^{(k)} = [y^{(k)}(1) \ldots y^{(k)}(m)]'\), \(\Delta \hat{Y}^{(k)} = Y - \hat{Y}^{(k)}\), \(S = [s(1) \ldots s(m)]'\), where \(k\) denotes the training epoch time, \(s(i) = [s_1 \ldots s_q]'\) is composed of \(q\) B-spline function values of the i-th input vector. An adaptive learning step size is derived as

$$\tilde{\eta} = \frac{||S'\Delta \hat{Y}^{(k-1)}||^2_s}{||S'S||_S}\left(||S'\Delta \hat{Y}^{(k-1)}||^2_s + \frac{\beta}{2}||S||^2_S\right)$$

(15)

where \(||x||^2_S := x'A_x x\). 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 $y = \sin^3 x$.

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 $x = 0.1, 0.2\pi,$ and the initial weights were zero. The condition of stopping learning is: $E < \epsilon$ or $(E(k) - E(k - 1))/E(k - 1) < \epsilon$, where $\epsilon = 1.e^{-6}$.

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 $\eta = 0.01$

<table>
<thead>
<tr>
<th>$N_k$</th>
<th>$N_e$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73</td>
<td>7.8873</td>
</tr>
<tr>
<td>5</td>
<td>216</td>
<td>0.5093</td>
</tr>
<tr>
<td>10</td>
<td>379</td>
<td>0.0245</td>
</tr>
<tr>
<td>20</td>
<td>741</td>
<td>0.0022</td>
</tr>
<tr>
<td>30</td>
<td>1424</td>
<td>2.232e-4</td>
</tr>
<tr>
<td>40</td>
<td>2581</td>
<td>3.600e-5</td>
</tr>
</tbody>
</table>

Table 2: Using the adaptive learning rate (15)

<table>
<thead>
<tr>
<th>$N_k$</th>
<th>$N_e$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>7.8873</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>0.5093</td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td>0.0245</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>0.0022</td>
</tr>
<tr>
<td>30</td>
<td>27</td>
<td>2.232e-4</td>
</tr>
<tr>
<td>40</td>
<td>34</td>
<td>3.5986e-5</td>
</tr>
</tbody>
</table>

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 $\eta(0) = 0.95$, $\epsilon = 0.001, N_k = 5$ for every variable of input, $u(i) = \sin(2\pi/10) + \sin(2\pi/25)$, sample number $m=100$ and 1st-order B-spline function for training.
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

Appendix: Derivation of (15)
The performance index at the k-th iteration can be written as follows:
\[ E(k) = \sum_{n=1}^{m} (y(i) - j(i)^{(k)})^{2} \]  \hspace{1cm} (A1)
For the BSNN, the 2nd term of right hand side in (11) is equal to zero, so (9) can be rewritten as follows
\[ w(k) = w(k - 1) + \eta S'\Delta \hat{y}^{(k-1)} \]  \hspace{1cm} (A2)
Here we have not considered the coefficient 2 in (10). Because \( j(i)^{(k)} = s(i)w(k) \) and noting that
\[ \hat{y}^{(k)}(i) = \hat{y}^{(k-1)}(i) = \eta s(i)'S'y \Delta \hat{y}^{(k-1)} \]
\[ S'\Delta \hat{y}^{(k-1)} = \sum_{i=1}^{m} s(i)\Delta \hat{y}^{(k-1)}(i) \]
and \( S'S = \sum_{i=1}^{m} s(i)s'(i) \), we have
\[ E(k) - E(k - 1) = \sum_{n=1}^{m} [-2(y(i) - \hat{y}^{(k)}(i)) + (\hat{y}^{(k)}(i))^{2} - (\hat{y}^{(k-1)}(i))^{2}] \]
\[ = \sum_{n=1}^{m} [-2(y(i) - \hat{y}^{(k)}(i)) + (\hat{y}^{(k)}(i))^{2} - (\hat{y}^{(k-1)}(i))^{2}] \]
\[ = -2\eta \sum_{m=1}^{m}(\Delta \hat{y}^{(k-1)}(i) - s'(i)s'\Delta \hat{y}^{(k-1)}) \]
\[ + \eta^{2} \sum_{n=1}^{m} s'(i)s'\Delta \hat{y}^{(k-1)} \]
\[ = -2\eta S'\Delta \hat{y}^{(k-1)} - \eta^{2} \sum_{n=1}^{m} s'(i)s'\Delta \hat{y}^{(k-1)} \]
Differentiating this quadratic equation with respect to parameter \( \eta \), (15) can be easily derived.