Sliding Window Based Resource-Allocating Network

Haikun Wei[1] , **Xinming Jin**[2] **,** and **Qi Li**[1]

1. Department of Automatic Control, Southeast University, Nanjing, 210096, China

email: hkwei@seu.edu.cn

2. System Integration Division, LSC Group, Lichfield, WS13 8RZ, UK

email:xij@lsc.co.uk

Abstract: In this paper, a sliding window based resource-allocating network (SW-RAN) for non-linear dynamic systems is proposed. SW-RAN is an online constructive-pruning hybrid approach for RBFN design. It adopts the following strategy: adding a new hidden unit if current network performs unsatisfactorily on the latest observation; otherwise, combining or deleting hidden units while adjusting unit centres and spread constants. SW-RAN not only makes the neural network online adapt to the changing dynamics of the plant being modelled, but also maintains a compact network size and guarantees its satisfying generalisation ability. More importantly, by incorporating the idea of sliding window, in which multiple latest samples are used for training, SW-RAN performs good robustness against the changes of learning parameters and is easy to converge. Three benchmark examples demonstrate the effectiveness of the method.

keywords: radial basis function network, sliding window, resource-allocating network, generalisation ability, non-linear system, online modelling

1. Introduction

In addition to its simple structure and strong approximation ability, feed-forward neural network does not require complex theoretical analysis. Because of these advantages, it is becoming more and more widely used in the areas such as non-linear system modelling. As one of its important applications, feed-forward network is often used for online modelling in practical engineering projects and adaptive systems.

Among the large number of feed-forward neural networks, Radial Basis Function Network (RBFN) [1] is one of the most often applied networks in non-linear system modelling. Comparing with other neural networks such as Multi-Layer Perceptron (MLP), RBFN has a very important characteristic named local property. The meaning of local property is to activate one or more hidden units once the neural network produces effective outputs. The local property of RBFN makes it most suitable for online learning. Because when new observations are continuously inputted online, the network performance will not be over affected by either adjusting weights or adding new hidden units. This will help to realise the so-called incremental learning.

There are a number of design methods available for RBFN. Generally, these design methods can be classified into two categories: i) set unit centres of hidden units as random values [2] or get them from sample inputs, such as Orthogonal Least Square (OLS) algorithm [3], Regularized Orthogonal Least Square (ROLS) algorithm [4] and Genetic Algorithm [5]; ii) adjust the positions of unit centres dynamically during the learning procedure, such as dynamic clustering based design method [6], Resource-allocating Network)(RAN) [7, 8], and minimal Radial Basis Function Networks (MRBF) [9, 10, 11, 12]. The common design principle of these methods is to seek a minimal neural network satisfying the accuracy requirement while maintaining its generalisation ability [13, 14].

Unfortunately, all the above design methods except RAN and MRBF adopt off-line learning algorithm. RAN recursively checks all sample input-output pairs during its learning procedure. When a new sample meets the “novelty” requirement, a new unit is allocated. Two criteria are defined for the “novelty”: 1) distance criterion: the distance between the current sample input and its nearest unit centre exceeds a definite value $\delta(t)$; 2) error criterion: the error between the

network output and the sample output exceeds a definite value $e_{\min}$. If these two criteria are both satisfied, a new resource will be allocated, which means a new hidden unit will be added. The unit centre of the new hidden unit is assigned by the input of the current sample. The output weight is set as the error between the current neural network output and the current sample output. The spread constant is assigned by the distance between the current sample input and its nearest unit centre. If the error between the output of the current neural network and the output of the new sample is comparatively small or the distance between the sample input and the existing unit centres is not big enough, a new hidden unit will not be allocated. Under such situations, gradient method or extended Kalman filter will be used to adjust unit centres and weights to further decrease the errors. MRBF improves the RAN method described in [8]. In addition to satisfying distance criterion and error criterion, no new hidden unit will be added until the errors between the output of the current network and previous consecutive samples are all too big. Moreover, a hidden unit will be deleted if it is not activated in multiple consecutive samples.

Although both RAN and MRBF can realise online learning, they have the following obvious disadvantages:

1) The algorithm' robustness to the changes of learning parameters is not satisfying. Take the value of resolution $\delta(t)$ as an example, it is found that when $\delta_{\max}$ and $\delta_{\min}$, especially $\delta_{\min}$, have minor changes, the hidden unit number will change dramatically and even cannot converge. The unsatisfying robustness to the changes of learning parameters is a big barrier of putting this algorithm into practice since it makes the learning parameters difficult to be decided.

2) The network generalisation ability cannot be guaranteed. In RAN, a hidden unit cannot be deleted once it is added. Under the situation of online learning, the network's hidden unit number will increase while the learning period continues, although some hidden units are possibly useless or redundant. In order to satisfy accuracy requirement, the increasing network scale will not only waste system resources, but also deteriorate the generalisation ability of the neural network [13, 14]. MRBF supports the deletion of hidden neurons, but the mechanism of deletion is too simple. Therefore, there are possibilities that some of the redundant hidden neurons will not be removed while some valid hidden neurons are deleted by mistakes.

In order to keep the current advantages of RAN and get rid of its addressed disadvantages, a sliding window based resource-allocating network (SW-RAN) is proposed and its detailed mechanism will be discussed in the following sections. According to the latest error information, SW-RAN can optimise system resources to maintain a compact network structure. This will certify the generalisation ability of the RBFN. SW-RAN optimises resources by online adding new hidden neurons (allocating new resources), online combining redundant hidden neurons (recombining resources), and online removing useless hidden neurons (releasing resources). By adopting the idea of sliding window, SW-RAN appears to have good robustness to the changes of learning parameters and is easier to converge.

In this paper, three benchmark examples are used to demonstrate the efficiency of SW-RAN. The example of static function approximation shows that SW-RAN can realise desired target function with minimal network structure. The example of time-varying non-linear function identification demonstrates that SW-RAN can online adapt to the time-varying dynamic plant. The Mackey-Glass chaotic time series prediction exemplifies SW-RAN's generalisation ability and its robustness to the variations of learning parameters.

This paper is organised as follows. A brief discussion of current RBFN design methods and related concepts are presented in Section 1. In Section 2, the basic idea of SW-RAN is introduced by analysing current RBFN online modelling methods. SW-RAN is discussed in detail in Section 3, which includes the mechanism of sliding window and the operations of adding new hidden units, adjusting network parameters, and pruning and combining hidden units. Section 4 demonstrates

the effectiveness of SW-RAN by simulating three typical examples. Finally, a brief summary of SW-RAN is provided and further research is discussed.

2. Introduction of RBFN

Considering a MISO RBFN with structure $m-h-1$ (Fig. 1), m is the number of network inputs and h is the number of hidden units. Since it is straightforward to get results of MIMO RBFNs based on MISO RBFNs, only will MISO RBFNs be discussed in this section. Assume that the active function of the *i-th* hidden unit is $\phi_i(x)$, which can be Gaussian radial basis function or polynomials in order to accelerate learning procedure [7]. If Gaussian radial basis function is adopted, then

$$\phi_i(x) = e^{-\frac{\|x-c_i\|^2}{r_i^2}} \tag{1}$$

If polynomials is used, then

$$\phi_i(x) = \begin{cases} \left(1-\dfrac{\|x-c_i\|}{qr_i^2}\right)^2, & \|x-c_i\| < qr_i^2 \\ 0, & other \end{cases} \tag{2}$$

In the above equation, $x=[x_1, x_2, ..., x_m]^T \in R^m$ are inputs of the RBFN. $c_i \in R^m$ is the unit centre of the *i-th* hidden unit, and r_i is the spread constant (or called width) of the unit centre. Set $q=2.67$ in equation (2), then the input-output model of the RBFN is:

$$f(x) = \sum_{i=1}^{h} w_i \phi_i(x) + b \tag{3}$$

Where $f(x) \in R$ is the output of the network; $w=[w_1, w_2, ..., w_h]$ is the weight vector while w_i is the weight between the *i-th* hidden unit and its related output; b is the output bias.

It has already been proved that the above RBFN model can approximate any continuous function in compact sets with any accuracy [15, 16, 17]. This includes the NAMARX model commonly used in non-linear dynamic system identification and control:

$$y(k) = f\left(y(k-1), ..., y(k-m_y), u(k-1), ..., u(k-m_u), k\right) \tag{4}$$

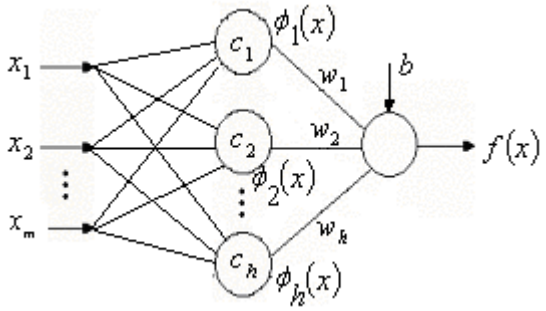

Fig.1 Structure of RBFN

Considering the situation of RBFN online learning, suppose that there are h hidden units in the current network, where $h \geq 0$. Assume a new observation (x_n, y_n) is obtained. Generally, there is a bias e_n between the network output $f(x_n)$ and the teacher's output y_n, which is $e_n = y_n - f(x_n)$. To maintain the bias e_n at a reasonable level, parameters including structure parameters and weights in RBFNs need to be adjusted. The adjustable parameters include: the hidden unit number, the position of unit centre c_i, the spread constant r_i, the output weight w, and

the output bias b.

Since both RAN and MRBF only use the latest single sample for on-line training, the trained RBFN is easy to be trapped into local minimums and its performance is seriously affected by bad samples [22]. Moreover, during the adjustment procedure, some unit centres will possibly overlap completely with the existing unit centres and some other centres will possibly move out of the normal working area. These will greatly waste system resources and influence the network's generalisation ability. So, a reasonable online modelling method is to use multiple latest samples and adjust network parameters dynamically in the learning process. That is, when the current network has a big bias on the latest samples, a new hidden unit is generated; otherwise, unit centres, weights and bias are adjusted to decrease the bias; when two or more centres overlap completely, they are combined; when some of the hidden units move out of the working area, they are pruned. This is the basic idea of SW-RAN.

3. Proposed online modelling method

3.1 Sliding window

The sliding window is a first-in-first-out (FIFO) queue with a fixed length L. All elements in the queue are samples obtained online. These samples are arranged according to the time when they enter the window, where (x_L, y_L) represents the latest sample and (x_1, y_1) is the earliest one. A sliding window with a length L can be represented as

$$Window = [(x_1, y_1), (x_2, y_2), ..., (x_L, y_L)] \tag{5}$$

When a new sample (x_n, y_n) is inputted, it becomes the newest sample in the window. If the number of samples in the window exceeds L, the earliest one will slide out of window. All samples in the window will be used for network training.

The following object function is used in network training:

$$E = \sum_{i=1}^{L} \beta_i e_i^2 \tag{6}$$

where e_i is the training error of the *i-th* sample in the window and β_i is the weighting coefficient of errors. Obeying the idea of "decaying factor", the value of β_i is taken as follows: if linear decaying is used, $\beta_i = \frac{2i}{L(L+1)}$; if adopting exponential decaying, then $\beta_i = \frac{1-\mu}{1-\mu^L}\mu^{L-i}$, where μ is the decaying factor. No matter which method is adopted, always have $\sum_{i=1}^{L} \beta_i = 1$. In online modelling, it is often acknowledged that the newer an observation is, the more information it contains. So, in the above decaying method, newer samples have bigger weights while older ones have smaller weights. In the following parts of this paper, linear decaying method will be used.

Sliding window uses not only the newest single sample, but also multiple latest samples. From the distributions of decaying weights, it can be seen that the newest information is the most important. The effect of decaying weights is very similar to the forgetting factor used for process system identification [21]. It is well known that the forgetting factor may be used to identify the time-varying systems, and it has good tracking performance. However, sliding window is different from forgetting factor, because after the presentation of each new training example, other samples remained in sliding window will be repeated for training, which means that each sample will be used for training for L times. Thus the sliding window is a combination of batch learning and sequential learning. It has been proved out that batch learning is more robust than sequential learning [22]. So, comparing with the usage of a single sample, sliding window brings better

robustness to the SW-RAN and makes its learning parameters easy to be tuned. The simulation results in section 4 will validate this assumption.

3.2 Generation of new hidden units

When a new sample (x_n, y_n) is obtained and put into the sliding window, firstly, it needs to decide whether a new hidden unit should be added. According to the idea of resource-allocating networks, if the current network has a large error e_n on the newest sample and the sample input is far away from all existing unit centres in the network (that is, no hidden unit is activated by this new sample), then it is regarded that the sample (x_n, y_n) can not be realised by the current network, which means that it's a "novel" sample complying the following rules:

$$d_n = \|x_n - c_{nearest}\| > \delta_{\min} \tag{7}$$

$$e_n = \|y_n - f(x_n)\| > \varepsilon_{\min} \tag{8}$$

Then, a new hidden unit should be added to eliminate the error. When the new unit is allocated, the network's output should be equal to the sample output y_n. The unit centre of the new hidden unit satisfies:

$$c_{h+1} = x_n \tag{9}$$

The connection weight from the hidden unit to its output is set as the value of e_n, which is the error between the output of the current network and the expected output:

$$w_{h+1} = e_n \tag{10}$$

The width of the hidden unit is taken as

$$r_{h+1} = \kappa d_n \tag{11}$$

where κ is the overlap factor, e_n and d_n are defined in equation (7) and (8).

Equation (7) defines the distance criterion, which means the new sample input is far away from all current unit centres. The error criterion is defined in equation (8), which means there is a great error between the sample output and the expected output of the network. $\varepsilon_{\min}$ is defined as the expected accuracy of network learning. If the error of the current network is greater than $\varepsilon_{\min}$, equation (9), (10), (11) are used to add a new hidden unit and the error will be eliminated immediately; The error which is less than $\varepsilon_{\min}$ will be gradually eliminated by the gradient method which will be introduced in the next section. In RAN and MRBF, distance $\delta_{\min}$ is decayed step by step, that is, it is decayed from a large initial value to a small final value. However, it is found that the maximum and minimum value of $\delta_{\min}$ greatly influence the learning results. In SW-RAN, a fixed value of $\delta_{\min}$ will be taken.

Initially, there is no hidden unit in the network. After the first sample (x_0, y_0) is inputted, let $b = y_0$. In addition, if there is no hidden unit when a new sample (x_n, y_n) is obtained, let $d_n = \delta_{\min}$.

3.3 Online adjustments of network parameters

If the current network performs well on the new sample, or the new sample input activates one or more hidden units, which means the network satisfies $e_n \le \varepsilon_{\min}$ or $d_n \le \delta_{\min}$ for the new sample (x_n, y_n), the error can be eliminated by adjusting unit centres, the spread constants of hidden units, output weights and bias. Normally, the Gradient method [7] or extended Kalman filter [8, 9, 10] are used for such adjustments. In this paper, the gradient method is adopted to modify unit centres and units' width while weights and bias are obtained by solving linear equations.

The gradient functions of the current network function $f(x)$ on unit centre c_i and width r_i separately are:

$$\nabla_{c_i} f(x) = \frac{2w_i}{r_i^2}\phi_i(x)(x - c_i) \tag{12}$$

$$\nabla_{r_i} f(x) = \frac{2w_i}{r_i^3}\phi_i(x)\|x - c_i\|^2 \tag{13}$$

Considering the impact of decaying factor and all samples in the sliding window, modifications of c_i and r_i are separately defined as:

$$\Delta c_i = \eta\beta_i e_i \sum_{j=1}^{L} \nabla_{c_i} f(x_j) \tag{14}$$

$$\Delta r_i = \eta\beta_i e_i \sum_{j=1}^{L} \nabla_{r_i} f(x_j) \tag{15}$$

where $\phi_i(x_j)$ is the output of the i*-th* hidden unit on x_j, w_i is its output weight, η is the learning rate.

In order to accelerate the learning procedure, a momentum item can be added in the gradient method, thus:

$$c_i(n) = c_i(n-1) + \Delta c_i(n) + \alpha\Delta c_i(n-1) \tag{16}$$

$$r_i(n) = r_i(n-1) + \Delta r_i(n) + \alpha\Delta r_i(n-1) \tag{17}$$

where $\Delta c_i(n-1)$ and $\Delta r_i(n-1)$ are the previous adjustments of c_i and r_i separately, α is the momentum factor, usually within the range 0.1~0.8.

To avoid that r_i becomes too big or too small, in practice, a delimitation operation can be made on r_i and makes it variable only within the range of $[r_{\min}, r_{\max}]$.

If unit centres and width change, weights from hidden units to outputs and bias should be modified accordingly. In fact, if unit centres and width are fixed, it is linear from hidden unit output to network output. So, weights and bias can be calculated by least mean square error algorithm. For the *j-th* sample in the sliding window, assume the input is x_j, $j = 1, 2, \cdots, L$, and the output of the i*-th* hidden unit is $a_{ij} = \phi_i(x_j)$, then the hidden unit output matrix can be defined as $A = [a_{ij}]$, $A \in R^{h \times L}$, where h is the hidden unit number, L is the length of the sliding window. Because of the local property of RBFN, only parts of hidden units are possibly activated by the samples in the sliding window. Obviously, only the related output weights of those activated hidden units should be readjusted, while the other weights should remain unchanged. Whether the i*-th* hidden unit is activated or not can be judged by the following rule:

$$\frac{1}{L}\sum_{j=1}^{L} a_{ij} > a_{\min} \tag{18}$$

Where $a_{\min}$ is a small threshold less than 0.01. If the above rule is satisfied, the i*-th* hidden unit is activated.

Suppose the teacher outputs are $y = [y_1, y_2, ..., y_L]^T$, the output matrix of the M activated hidden units is $\overline{A}$, extend matrix of $\overline{A}$ is $\hat{A} = \begin{bmatrix} \overline{A} \\ 1_L \end{bmatrix}$, where 1_L is a L-dimension line vector of which all elements equal to 1. Let

$$W = \hat{A}^{+} y \tag{19}$$

where $\hat{A}^{+}$ is the pseudo-inverse of $\hat{A}$:

$$\hat{A}^{+} = (\hat{A}\hat{A}^{T})^{-1}\hat{A} \tag{20}$$

Obviously, $W \in R^{1\times(M+1)}$. Thus, the output weights of the M activated hidden units are modified as:

$$w(n) = [W(1), W(2), ..., W(M)] \tag{21}$$

which is the first M elements of W. The bias is updated as:

$$b(n) = W(M+1) \tag{22}$$

Here, only the output weights of the activated hidden units are adjusted. This will not only reduce calculation greatly, but also avoid the overflow problem while solving the inverse matrix. This advantage of SW-RAN is especially important in real applications.

The training process should be stopped if the training reaches a fixed number of epochs, or the error of samples in sliding window satisfies the following inequation:

$$\sqrt{E} < \varepsilon_{\min} \tag{23}$$

To accelerate convergence, the learning procedure can also be stopped when the error does not decrease in two consecutive gradient modifications. This means:

$$\Delta\sqrt{E} < \Delta\varepsilon_{\min} \tag{24}$$

Generally, $\Delta\varepsilon_{\min}$ is less than 0.0001.

3.4 Online combination and deletion of hidden units

When the unit centres and the width values of hidden units are adjusted by the gradient method, two situations could occur: 1) two or more unit centres are near to each other and their width values are nearly the same, then these hidden units should be combined; 2) some of the unit centres move away from the working area and become useless, then these hidden units should be removed. Otherwise, both situations will affect the network performance.

Consider the combination of hidden units first. Suppose that i, j are two hidden units, their unit centres are close to each other because of continuous adjustments. In addition, they are the nearest hidden units of each other. Then,

$$\left\| c_i - c_j \right\| < \Delta c_{\min} \tag{25}$$

$$\left| r_i - r_j \right| < \Delta r_{\min} \tag{26}$$

where both $\Delta c_{\min}$ and $\Delta r_{\min}$ are the combination thresholds, normally taking the value of 0.01. Then, for any input x, outputs of the hidden units i, j satisfy

$$\phi_i(x) \approx \phi_j(x) \tag{27}$$

Thus, the hidden unit j can be combined with the hidden unit i. Obviously, the output weight of the combined unit i is

$$w_i = w_i + w_j \tag{28}$$

The unit centre of the combined unit i is

$$c_i = \frac{1}{2}(c_i + c_j) \tag{29}$$

Then, consider the deletion issue of the useless hidden units. When some of the hidden units move away from the working area due to the adjustments, samples in the working area will no longer activate these hidden units. So, if a hidden unit is not activated in consecutive input samples, it should be removed. That is, when the following inequation is satisfied

$$count > C_{\max} \tag{30}$$

Where $count$ is the accumulated times of the hidden unit not being activated, then the related hidden unit should be pruned. Since the existence of the useless units will not greatly affect the realisation of SW-RAN, $C_{\max}$ can take a relatively large value to prevent useful units being removed.

3.5 Algorithm realisation

Based on the above discussions, the online modelling algorithm of SW-RAN can be realised as follows:

- Initially, there is no hidden unit in the network. Let the bias be the first sample output: $b = y_0$;
- for each online input sample (x_n, y_n):
  - put the sample into the sliding window and move out the oldest sample if it has;
  - count the number of not being activated for each hidden unit, remove those units not being activated for long time;
  - calculate: $\phi_i(x) = e^{-\frac{\|x_n - c_i\|^2}{r_i^2}}$, $f(x_n) = \sum_{i=1}^{h} w_i \phi_i(x_n) + b$, $e_n = y_n - f(x_n)$, $d_n = \|x_n - c_{nearest}\|$;
  - if $d_n > \delta_{\min}$ and $\|e_n\| > \varepsilon_{\min}$, allocate a new hidden unit as follows:

$$c_{h+1} = x_n$$
$$w_{h+1} = y_n - f(x_n)$$
$$r_{h+1} = \kappa d_n$$

  - otherwise, adjust network unit centres, width, weights and bias as follows, until the maximum training epochs is reached, or the training error of samples in the sliding window is less than a given value:

$$\nabla_{c_i} f(x) = \frac{2w_i}{r_i^2}\phi_i(x)(x - c_i)$$
$$\nabla_{r_i} f(x) = \frac{2w_i}{r_i^3}\phi_i(x)\|x - c_i\|^2$$
$$\Delta c_i = \eta \beta_i e_i \sum_{j=1}^{L} \nabla_{c_i} f(x_j)$$
$$\Delta r_i = \eta \beta_i e_i \sum_{j=1}^{L} \nabla_{r_i} f(x_j)$$
$$c_i(n) = c_i(n-1) + \Delta c_i(n) + \alpha \Delta c_i(n-1)$$
$$r_i(n) = r_i(n-1) + \Delta r_i(n) + \alpha \Delta r_i(n-1)$$
$$\hat{A}^+ = (\hat{A}\hat{A}^T)^{-1}\hat{A}$$
$$W = \hat{A}^+ y$$
$$w(n) = [W(1), W(2), ..., W(M)]$$
$$b(n) = W(M+1)$$

Finally, check if there are redundant hidden units; combine them if there are some.

4. Simulation study

4.1 Static function approximation problem

This problem is studied to test whether SW-RAN is able to design a minimal RBFN that exactly matches the target function [9, 18]. The non-linear target function is a two-input-single-output system, which consists of six exponential functions as follows:

$$y(k)=\exp\left[-\frac{(x_1-0.3)^2+(x_2-0.2)^2}{0.01}\right]+\exp\left[-\frac{(x_1-0.7)^2+(x_2-0.2)^2}{0.01}\right]$$
$$+\exp\left[-\frac{(x_1-0.1)^2+(x_2-0.5)^2}{0.02}\right]+\exp\left[-\frac{(x_1-0.9)^2+(x_2-0.5)^2}{0.02}\right] \quad (31)$$
$$+\exp\left[-\frac{(x_1-0.3)^2+(x_2-0.8)^2}{0.01}\right]+\exp\left[-\frac{(x_1-0.7)^2+(x_2-0.8)^2}{0.01}\right]$$

Obviously, the target function can be realised by a RBFN with six hidden units. In order to evolve a minimal RBFN using SW-RAN method, 121 training patterns $((x_1,x_2),y)$ are provided, where (x_1,x_2) is the input, y is the output, $x_i \in \{0,0.1,...,1\}$, $i \in \{1,2\}$.

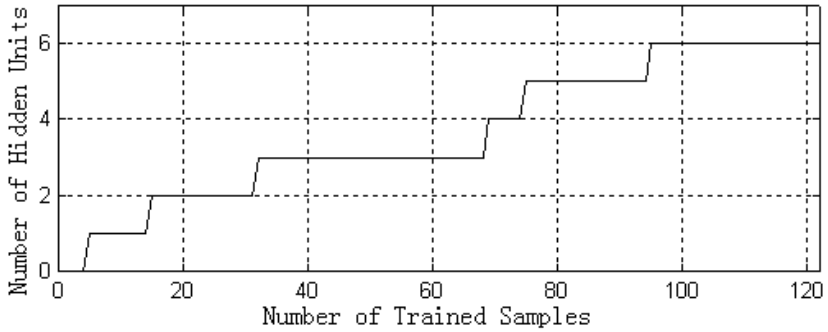

Fig. 2 Evolution of the hidden units

Table 1 True and estimated centres, width and weight

True centre	**(0.3,0.2)**	**(0.7,0.2)**	**(0.1,0.5)**	**(0.9,0.5)**	**(0.3,0.8)**	**(0.7,0.8)**
Estimated centre	(0.301,0.200)	(0.699,0.198)	(0.097,0.500)	(0.900,0.500)	(0.300,0.800)	(0.699,0.802)
True width	0.1	0.1	0.144	0.144	0.1	0.1
Estimated width	0.098	0.099	0.139	0.149	0.099	0.099
True weight	1.0	1.0	1.0	1.0	1.0	1.0
Estimated weight	1.038	1.008	1.013	0.966	1.040	1.004

The learning parameters of SW-RAN are set as follows (some of these parameters will not be used in simulation): $\varepsilon_{\min}$ =0.02, $\delta_{\min}$ =0.3, combination threshold $\Delta c_{\min}$ =0.01, $\Delta r_{\min}$ =0.01, activating threshold $a_{\min}$ =0.01, $C_{\max}$ =400, overlap coefficient κ =1.0, length of sliding window L =60, learning rate η =0.5, momentum rate α =0.5, maximum training epoch MaxEpoch=100, $\Delta\varepsilon_{\min}$ =0.0001, and the maximum and minimum values of width are $r_{\min}$ =0.05, $r_{\max}$ =0.25 separately.

Fig. 2 displays the evolution of the number of hidden units in SW-RAN during the learning procedure. Table 1 presents the comparisons of true and estimated centres, width and weights. From Table 1, it can be seen that the estimated values are very close to the true values. This simulation proves that SW-RAN can realise a given minimal RBFN quite well and the results are much better than the results presented in reference [9] and [18]. When the learning process finishes, the estimated bias is $b=-0.001$, which is very close to the true value as well.

4.2 Non-linear dynamic system identification

This is a non-linear SISO time-varying discrete system [12,19]:

$$y(k)=\frac{29\beta(k)}{40}\sin\left(\frac{16u(k-1)+8y(k-1)}{\beta(k)(3+4u(k-1)^2+4y(k-1)^2)}\right)+\frac{2}{10}u(k-1)+\frac{2}{10}y(k-1) \tag{32}$$

where $u(k)$ is the system input signal with uniform distributions within $[-1,1]$, $\beta(k)$ is a time-varying parameter, its evolution is as follows:

$$\beta(k)=\begin{cases}1.0, & 0\le k\le 1500\\ 0.9, & 1501\le k\le 2500\\ 0.8, & 2501\le k\le 5000\end{cases} \tag{33}$$

Let $y(0)=0$, generate 5000 training patterns according to the above equation.

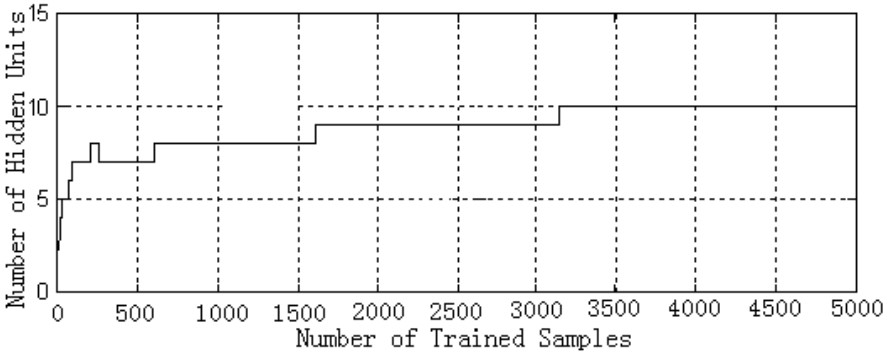

Fig. 3 Evolution of hidden units

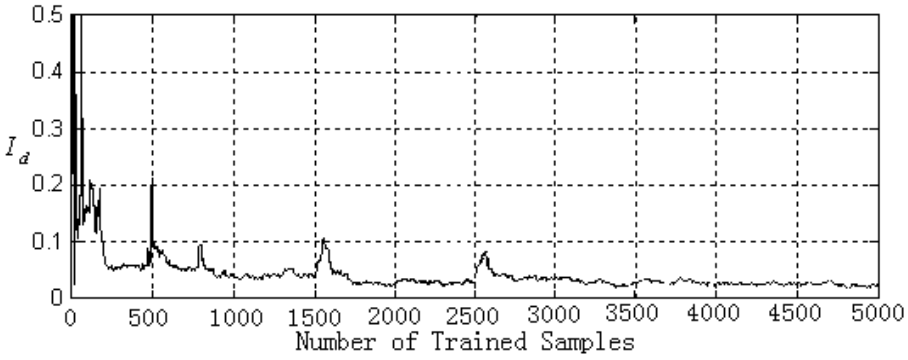

Fig. 4 Evolution of error (I_d)

Set the learning parameters of SW-RAN as follows: $\varepsilon_{\min}$ =0.01, $\delta_{\min}$ =1.5, $\Delta c_{\min}$ =0.01, $\Delta r_{\min}$ =0.01, $a_{\min}$ =0.01, $C_{\max}$ =200, κ =0.9, L =100, η =0.5, α =0.5, MaxEpoch=200, $\Delta\varepsilon_{\min}$ =0.00001, $r_{\min}$ =0.5, $r_{\max}$ =2.5.

For a certain training set, Fig 3 demonstrates the changes of hidden neurons together with $\beta(k)$. If $\beta(k)=1.0$, SW-RAN fits well with the target function with 8 hidden units when $k=600$; when $k=1600$, the number of units increases to 9 while $\beta(k)=0.9$; when $k=3100$ or so, the number of units becomes 10 while $\beta(k)=0.8$. After that, the number of hidden units remains unchanged.

For comparison, the following error criterion is used [12, 19]:

$$I_d(i)=\frac{1}{L}\sum_{p=0}^{L-1}\left|y(i-p)-\hat{y}(i-p)\right| \tag{34}$$

Fig 4 shows the evolution history of error index I_d. It can be seen from Fig 4 that I_d increases

obviously when $\beta(k)$ changes at $k=1500$ and $k=2500$, then they are depressed by SW-RAN very soon.

In Table 2, the identification results of SW-RAN, MRBF and ONSAHL are compared, where I_{dav} is the average value of I_d calculated from $k=1500$ to $k=5000$. The results of MRBF and ONSAHL are taken from the reference [12]. From Fig 3, Fig 4 and Table 2, it can be seen that SW-RAN performs better than MRBF and ONSAHL on learning speed, training error and the final hidden units.

Worthy of noting, because of the randomness of training sets, the training results also have some kind of randomness. During the 10 tests being made, the number of hidden units fluctuates within 10 ~ 13 and I_{dav} within 0291 ~ 0.0407. So, the results are still quite stable.

Table 2 Comparison of identification results of ONSAHL, MRBF and SW-RAN

Performance	**Hidden units**	I_{dav}
ONSAHL	25	0.0586
MRBF	11	0.0326
SW-RAN	10	0.0311

4.3 Prediction of chaotic time series

In this section, the SW-RAN is applied to model and predict future values of a chaotic time series – the Mackey-Glass (MG) data set [20], which has been used as a benchmark in neural networks, fuzzy systems, and hybrid systems. The time series is created by using the MG time-delay differential equation defined as below:

$$\frac{dx(t)}{dt}=\frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)}-0.1x(t) \tag{34}$$

To obtain the values of this time series at integer points, the fourth-order Runge-Kutta method is used to get the numerical solution to the above MG equation. Here, assume the time step is 0.1, $x(t)=0$, $\tau=17$ and $x(0)=1.2$ for $t<0$.

The task is to predict the value $x(t+50)$ from the input vector $[x(t-18)\ x(t-12)\ x(t-6)\ x(t)]$ and test the robustness of SW-RAN on the changes of the learning parameters. The following experiment is conducted: 1500 data points, from t = 201 to 1700, are used as learning data, and 500 data points, from t = 3001 to 3500, are used as testing data.

The parameters used in this experiment are set as follows: $\varepsilon_{\min}$ =0.01, $\delta_{\min}$ =0.15, $\Delta c_{\min}$ =0.01, $\Delta r_{\min}$ =0.01, $a_{\min}$ =0.0001, $C_{\max}$ =400, κ =0.9, L =300, η =0.5, α =0.5, MaxEpoch=200, $\Delta\varepsilon_{\min}$ =0.0001, $r_{\min}$ =0.01, and $r_{\max}$ =2.5.

Fig 5 presents the changing number of the hidden neurons during the learning procedure. Fig 6 records the changing history of the training error and testing error. The training error is calculated according to the samples in the sliding window while the testing error is normalised rooted mean square error (RSME). After being trained on 1500 patterns, the final number of the hidden units is 45 and the normalised RSME on 500 test samples is 0.0427. From Fig 5 and Fig 6, it can be seen that the test only uses no more than 1500 patterns to fit well on MG time series, in addition, with more compact network structure and less testing error. It's obvious that SW-RAN performs better than RAN [7, 8].

As mentioned before, SW-RAN shows good robustness against changing learning parameters. To test the robustness of SW-RAN, the values of main parameters are changed in simulation. The testing results are presented in Table 3. The parameters being changed are $\varepsilon_{\min}$, $\delta_{\min}$, sliding window length L, and maximum training epochs MaxEpoch. The number of hidden units and

normalised RSME for different parameters are also listed in Table 3. The results show that the fluctuations of network structures and testing errors are much more smooth than RAN [7, 8], which proves that SW-RAN has good robustness and makes the parameters easier to tune.

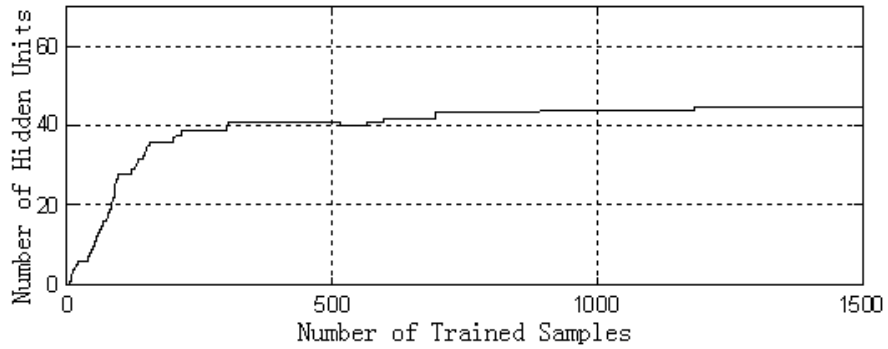

Fig.5 Evolution of hidden units

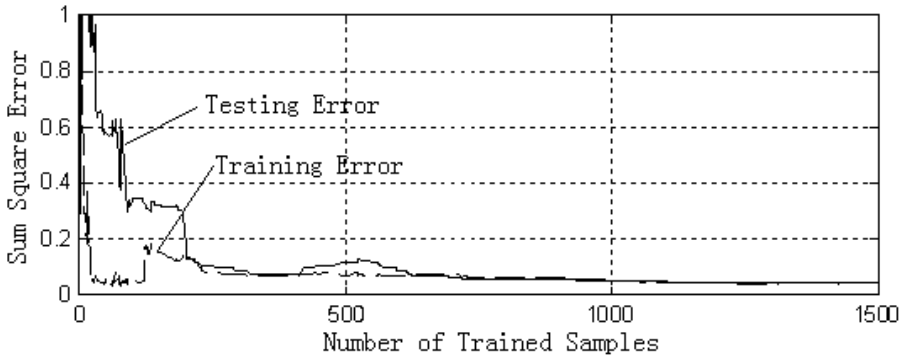

Fig.6 Evolution of training error and normalized RSME

Table 3 Result comparison with different learning parameters

$\varepsilon_{\min}$	$\delta_{\min}$	L	MaxEpoch	Hidden Unit	Normalized RSME
0.010	0.15	300	200	45	0.0427
0.005	0.15	300	200	62	0.0668
0.015	0.15	300	200	46	0.0622
0.010	0.18	300	200	35	0.0540
0.010	0.12	300	200	62	0.0393
0.010	0.15	250	200	46	0.0443
0.010	0.15	350	200	45	0.0430
0.010	0.15	300	100	45	0.0427

5. Discussions and conclusions

In this paper, some problems of current online modelling methods are addressed firstly. Then, an online RBFN modelling method using sliding window called SW-RAN is proposed. The proposed method incorporates sliding window with resource-allocating networks to gain some additional benefits for online modelling. The issues of network structure and parameters' online adjustment, including operations of adding, combining and pruning hidden units, modifications of unit centres, weights and bias, are discussed to online design a compact network that satisfies the accuracy requirement and has a good generalisation ability. Three benchmark examples demonstrate that SW-RAN has the abilities of building minimal models, adapting to time-varying plants, and being robust to the changes of the learning parameters.

For an online modelling method, it's also important to consider the issue of real time realisation. The simulation results in this paper show that SW-RAN can satisfy the real time requirements of most practical applications. In the testing examples of 4.2 and 4.3, a simple test on learning time of

SW-RAN is conducted. The simulation is implemented by VC++ 6.0, the operation system of the PC is Windows XP, and the CPU of the PC is Pentium 2.0GHz. The results present that the average learning time of SW-RAN is less than 0.2 second for every new training pattern, even with worse learning parameters. This is fast enough to meet the real time requirements of most applications. Of course, digital signal processing (DSP) chips may be used to meet faster speed requirement.

However, SW-RAN is supposed to know the order of the model in advance and its only structure parameter need to be adjusted is the number of hidden units. In reality, it is also common that the model order is unknown and needs to be determined online. For example, in the identification of NARMARX model in equation (4), the value of m_u and m_y sometimes also need online determination. How to determine the model order together with the number of hidden units is an issue need to be further investigated.

References

[1] T. Piggio and F. Girosi, “Networks for Approximation and Learning”, Proc. IEEE. 1990, 78(9), 1481-1497.

[2] D. Broomhead and D. Lowe. “Multi-variable functional interpolation and adaptive networks”, Complex Systems. 1988, 2, 269-303.

[3] S. Chen, C. Cowan and P. Grant. “Orthogonal Least Squares Learning Algorithms for Radial Basis Function┬ Networks”, IEEE Trans. Neural Networks, 1991, 2(2), 302-309.

[4] M. Orr. “Regularization in the selection of radial basis function centres”, Neural Computation, 1995, 7, 606-623.

[5] K. Mao and S. Billings. “Algorithms for minimal model structure detection in non-linear dynamic system identification”, Int. J. Control, 1997, 68(2), 311-330.

[6] J. Moody And C. Darken. “Fast Learning in Networks of Locally-Tuned Processing Units”, Neural Computation, 1989, 1, 281-294.

[7] J. Platt. “A resource-allocating network for function inerpolation”, Neural Computation, 1991, 3, 213-225.

[8] V. Kadirkamanathan. “A function estimation approach to sequential learning with neural networks”, Neural Computation, 1993, 5, 954-975.

[9] L. Yingwei, N.Sundararajan and P.Saratchandran. “Identification of time-varying non-linear systems using minimal radial basis function neural networks”, IEE Proc.-Control Theory Appl. 1997, 144(2), 202-208.

[10] L. Yingwei, N.Sundararajan and P.Saratchandran. “A sequential learning scheme for function approximation and using minimal radial basis neural networks”, Neural Computation, 1997, 9(2), 1-18.

[11] L. Yingwei, N.Sundararajan and P.Saratchandran. “Performance evaluation of a sequential minimal radial basis function (RBF) neural networks learning algorithm”, IEEE Trans. Neural Networks, 1998, 9(2), 308-318.

[12] Y. Li, N.Sundararajan and P.Saratchandran. “Analysis of minimal radial basis function neural networks algorithm for real-time identification of non-linear dynamic systems”, IEE Proc.-Control Theory Appl. 2000, 147(4), 476-484.

[13] P. Niyogo and F. Girosi. “On the Relationship between Generalization Error, Hypothesis Complexity, and Sample Complexity for Radial Basis Function”, Neural Computation, 1996, 8, 819-842.

[14] S. Geman, E. Bienenstock and R. Doursat. “Neural Networks and Bias/Variance Dilemma”. Neural Computation, 1992, 4, 1-58.

[15] E. Hartman, J. D. Keeler and J. Kowalski. “Layered neural networks with Gaussian hidden units as universal approximators”, Neural Computation. 1990, 2, 210-215.

[16] J. Park and I. W. Sandberg. “Universal approximation using radial-basis-function”, Neural Computation. 1991, 3, 246-257.

[17] F. Girosi and T. Poggio. “Networks and the best approximation property”, Boil. Cybernet. 1990, 63, 169-176.

[18] C. Chen, W. Chen and F. Cheng. “Hybrid learning algorithm for Gaussian potential network”, IEE Proc. D, 1993, 140, 442-448.

[19] T. Junge and H. Unbehauen, “On-line identification of non-linear time-variant systems using

structurally adaptive radial basis function networks", Proc. Amer. Control Conf., Albuquerque, New Mexico, 1997, 1037-1041.
[20] M. Mackey and L. Glass, "Oscillation and Chaos in Physiological Control systems", Science, 1977, 197, 287- 289
[21] L. Ljung. "System identification: Theory for the User", Prentice-Hall, 1999.
[22] T. Heskes and W. Wiegerinck, "A theoretical comparison of batch-mode, on-line, cyclic, and almost-cyclic learning", IEEE Trans. Neural Networks, 1996, 7(4), 919-925.