Continuous CMAC-QRLS and Its Systolic Array

TING QIN, HAITAO ZHANG, ZONGHAI CHEN, WEI XIANG

Department of Automation, University of Science and Technology of China, Hefei 230027, P.R.China

e-mail:qinting@ustc.edu

Abstract: Conventionally, least mean square rule (LMS) is used to update the weights of CMAC. The algorithm of CMAC-RLS which applies recursive least square algorithm (RLS) to update the weights of CMAC has proved to be a good tool for modeling on line. Based on QR decomposition, a simplified algorithm of CMAC-RLS named CMAC-QRLS is brought forward next and its corresponding systolic array is also designed. Combining with B-splines, we further devise the systolic array of continuous CMAC-QRLS. The simulation results reveal the good performance of this proposed algorithm.

Key words: CMAC-QRLS, QR decomposition, Systolic array, B-splines

1. Introduction

Three decades ago, the cerebellar model articulation controller (CMAC) was first proposed by Albus [1,2]. In the nineties of last century the continuous CMAC combined with Gaussian[3] or B-splines[4] receptive field functions was introduced. This kind of continuous CMAC can further improve the accuracy of function approximation and provide the values of function derivatives. It was also since the nineties of last century that the convergence property of CMAC has been investigated and these theoretical research results[6-8] simultaneously have been promoting the application of CMAC. Because CMAC possesses good local generalization property, moreover, it is capable of fast learning and easy to be realized in hardware[9] as well as in software, it has been successfully applied in many fields, such as robotic control[10-13], pattern recognition[14], and signal processing[15], etc.

Traditionally, CMAC uses LMS rule to update its weights. Recently, Qin[16] introduced the CMAC-RLS algorithm which can guarantee the learning algorithm to converge in just one epoch when a new provided sample is trained. In order to make this algorithm more suitable for implementing on line, a simplified algorithm based on QR decomposition and named CMAC-QRLS will be introduced next. CMAC-QRLS not only reduces the computation time and memory storage, but also improves the numerical stability. And it can also be easily implemented in pipelined systolic array. Because B-splines are very appropriate for interpolation, the systolic array of continuous CMAC is further devised combined with B-splines.

2. Mechanism of CMAC-RLS

2.1. CMAC Technique

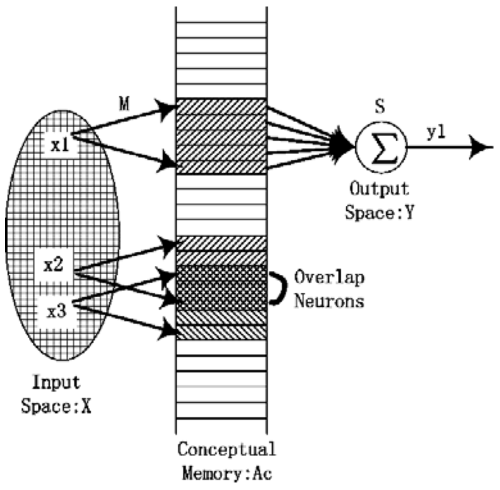

Figure.1. Basic skeleton of CMAC

A schematic sketch of CMAC without hash-coding is illustrated in Figure.1. It can be regarded including two mappings:

$$X \xrightarrow{M} A_c \tag{1}$$

$$A_c \xrightarrow{S} Y \tag{2}$$

where X , A_c and Y are input space, the space of conceptual memory and output space, respectively.

First, the input vector is quantized and then activates g association neurons in A_c by (1) (In Figure.1, $g = 5$).Then, the weights attached to the activated neurons of A_c are summed to produce the output. g is generally called the generalization factor. It represents the neurons activated by one input.

If two inputs are close, there will be overlap between their corresponding activated neurons, such as x_2 and x_3. On the contrary, the neurons activated by input vectors far away do not overlap, such as x_1 and x_2 .

Given a set of sample data (x_k, y_k) , where x_k is the input and y_k is the desired output. We represent the neurons that x_k activates by a vector of characteristic function $a_{ik}(1 \le i \le n)$, where

$$a_{ik} = \begin{cases} 1 & \text{the } i\text{th neuron in } A_c \text{ is activated by } x_k \\ 0 & \text{otherwize} \end{cases} \tag{3}$$

and n is the size of conceptual memory.

The vector of characteristic function is also called indicator[6] and denoted as

$$\varphi_k = [a_{1k}, a_{2k}, \cdots, a_{nk}] \tag{4}$$

Form the matrix $A_k = \begin{bmatrix}\varphi_1^T & \varphi_2^T & \cdots & \varphi_k^T\end{bmatrix}_{k\times n}^T$, $\quad Y_k = \begin{bmatrix}y_1 & y_2 & \cdots & y_k\end{bmatrix}_{k\times 1}^T$ (5)

where k is the number of training samples. The goal of CMAC learning is to find a weight vector θ_k such that

$$A_k \theta_k = Y_k \tag{6}$$

where $\theta_k^T = [w_{1k}, w_{2k}, \cdots, w_{nk}]$ and represents the contents in A_c after the k th set of sample data has been presented and the calculating is over.

2.2. CMAC-RLS Algorithm

CMAC-RLS can converge in just one epoch and has proved to be a useful tool for modeling on line. It can be descried as follows.

The initial values of P_k and $\hat{\theta}_k$ are set as

$$P_0 = \beta I_n \text{ and } \hat{\theta}_0 = 0 \tag{7}$$

where β is set as some large constant and I_n is an $n \times n$ unit matrix.

$$\forall k, \quad \pi_{k+1} = \varphi_{k+1} P_k \tag{8}$$

$$L_{k+1} = \pi_{k+1}^T \big/ (1 + \pi_{k+1} \varphi_{k+1}^T) \tag{9}$$

$$P_{k+1} = P_k - L_{k+1} \pi_{k+1} \tag{10}$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + L_{k+1}(y_{k+1} - \varphi_{k+1} \hat{\theta}_k) \tag{11}$$

where $\hat{\theta}_k$ is the estimated vector of θ_k .

The CMAC-RLS learning is used on line following the next steps:

1) Initialize the values of P_k and $\hat{\theta}_k$ by (7);
2) Quantize x_k when a new sample (x_k, y_k) is provided;
3) Compute φ_k by (3) and (4);
4) Update the weight vector by (8),(9),(10),(11);
5) Go to 2) until the end.

Based on the special characteristic of indicator, a simplified CMAC-RLS was also introduced in reference [16] which can be described next:

$$\forall k\,,\quad \pi_{k+1} = \varphi_{k+1}(nzpos_k)P_k(nzpos_k,:) \tag{12}$$

$$L_{k+1} = \pi_{k+1}^T \big/ 1 + \pi_{k+1}(nzpos_k)\varphi_{k+1}^T(nzpos_k) \tag{13}$$

$$P_{k+1} = P_k - L_{k+1}\pi_{k+1} \tag{14}$$

$$\hat{\theta}_{k+1} = \hat{\theta}_k + L_{k+1}(y_{k+1} - \varphi_{k+1}(nzpos_k)\hat{\theta}_k(nzpos_k)) \tag{15}$$

$nzpos_k$ is the set to indicate the positions of those nonzero elements in φ_k.

Then the computation complexity of this simplified CMAC-RLS can be calculated in Table.1:

Table.1. computation complexity of this simplified CMAC-RLS

Equations	multiplications	Divisions	Square Roots
(12)	$g \times n$		
(13)	g	n	
(14)	$n \times n$		
(15)	$g + n$		
Total	$n \times n + g \times n + 2g + n$	n	0

With regard to the storage memory, CMAC-RLS needs at least n^2 memory cells. It is necessary to reduce the computation time and memory storage for more convenient implementation. Thus another simplified algorithm of CMAC-RLS is brought forward in the next section.

3. Continuous CMAC-QRLS and Its Systolic Array

3.1. The QR-RLS Algorithm of CMAC

Let
$$\Phi_k = A_k^T A_k \tag{16}$$

then Φ_k can be expressed in its factored form with Cholesky factorization:

$$\Phi_k = \Phi_k^{1/2}\Phi_k^{T/2} \tag{17}$$

where $\Phi_k^{T/2}$ is the Hermitian transpose of the lower triangular matrix $\Phi_k^{1/2}$.

We also introduce a new vector variable defined by

$$p_k^T = \Phi_k^{T/2}\hat{\theta}_k , \tag{18}$$

and let
$$\delta = \beta^{-1/2} \tag{19}$$

then the QR-RLS[17] algorithm of CMAC is deduced and presented in Table 2:

Table.2. Summary of QR-RLS algorithm of CMAC

1. Initialization:

$$\Phi_0^{1/2} = \delta I_n \,(\delta \text{ is set as a small real constant}),\quad p_0 = \mathbf{0}^T , \tag{20}$$

where $\mathbf{0}$ is the null vector.

2. Recursive Operation

(x_k , y_k) is provided, compute

2.1. Forward computation:

$$\begin{bmatrix} \Phi_{k-1}^{1/2} & \varphi_k^T \\ p_{k-1} & y_k \end{bmatrix} \Theta_k = \begin{bmatrix} \Phi_k^{1/2} & 0 \\ p_k & \xi_k \gamma_k^{1/2} \end{bmatrix} \tag{21}$$

2.2. Back substitution:

$\hat{\theta}_k$ can be calculated through (18) in the way that exploits the upper triangular structure of $\Phi_k^{T/2}$.

3. Terminate

Note: where Θ_k is a unitary rotation that operates on the elements of φ_k^T in the prearray, annihilating them one by one so as to produce a block zero entry in the top block row of the postarray.

$\xi_k = y_k - \varphi_k \hat{\theta}_{k-1}$, $\gamma_k = \dfrac{y_k - \varphi_k \hat{\theta}_k}{y_k - \varphi_k \hat{\theta}_{k-1}}$, and they will not be considered here.

The QR-RLS algorithm of CMAC based on Givens rotation can also be implemented in systolic array[17] which is shown in Figure 2.

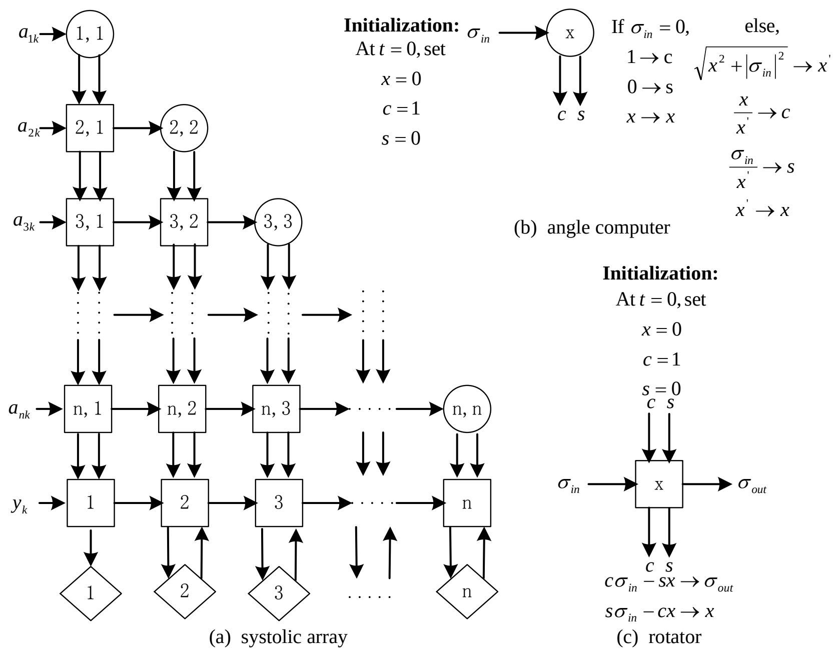

Figure.2. Systolic array of QR-RLS algorithm of CMAC

σ_{in} : cell input signal, σ_{out} : cell output signal,

c , s : cell computation signal.

The systolic array is controlled by a single clock and consists of three types of processing cells arranged in the form of a triangular section:

- Diagonal cells depicted as circles compute square roots and divisions, as described in Figure.2(b). This kind of cell is also called angle computer.
- Internal cells depicted as squares perform only additions and multiplications as described in Figure.2(c). This kind of cell is also called rotator.
- Hemline cells depicted as diamonds store the weights calculated by back substitution.

For simplicity, let us define that

$$\Omega = \begin{bmatrix} \Phi_{k-1}^{1/2} \\ p_{k-1} \end{bmatrix} \tag{22}$$

The upper memory cells correspond Ω one by one. That is, $\boxed{i, j}$ stores the element who is in the i th row and the j th column of $\Phi_{k-1}^{1/2}$ and $\boxed{i}$ stores the i th element of p_{k-1}. It should be noted that the upper triangular elements of $\Phi_{k-1}^{1/2}$ are all zeros and do not participate in any computation, so they are not considered.

The detailed procedure for the operation of the systolic array can be referred to reference [17], and it is only described here by code in Table.3.

Table.3. Code of QR-RLS algorithm of CMAC

```
1. Initialization:
$For\ j = 1,2,\cdots,n\ Loop$
$\Omega_{j,j} = \delta$
    $For\ i = j+1, j+2,\cdots,n+1\ Loop$
    $\Omega_{i,j} = 0$
    $End\ of\ Loop\ i$
$End\ of\ Loop\ j$
2. Recursive Operation
  $(x_k, y_k)$ is provided, compute
  2.1. Forward Computation
$For\ i = 1,2,\cdots,n\ Loop$
$\sigma_i = a_{ik}$
$End\ of\ Loop\ i$
$\sigma_{n+1} = y_k$
$For\ j = 1,2,\cdots,n\ Loop$
$d = \sqrt{\sigma_j^2 + \Omega_{j,j}^2}$
$c = \Omega_{j,j} / d,\ s = \sigma_j / d$
    $For\ i = j, j+1,\cdots,n+1\ Loop$
    $\Omega_{i,j} = c\Omega_{i,j} + s\sigma_i$
    $\sigma_i = -s\Omega_{i,j} + c\sigma_i$
    $End\ of\ Loop\ i$
$End\ of\ Loop\ j$
2.2. Back Substitution
$w_n = \Omega_{n+1,n} / \Omega_{n,n}$
$T = n - g$
$For\ j = n-1, n-2,\cdots,T+1\ Loop$
$temp = 0$
    $For\ i = j+1, j+2,\cdots,n\ Loop$
    $temp = temp + \Omega_{i,j} w_i$
    $End\ of\ Loop\ i$
$w_j = (\Omega_{n+1,j} - temp) / \Omega_{j,j}$
$End\ of\ Loop\ j$
$For\ j = T, T-1,\cdots,1,\ Loop$
$temp = 0$
    $For\ i = j+1, j+2,\cdots, j+g-1,\ Loop$
    $temp = temp + \Omega_{i,j} w_i$
    $End\ of\ Loop\ i$
$w_j = (\Omega_{n+1,j} - temp) / \Omega_{j,j}$
$End\ of\ Loop\ j$
3. Terminate
```

3.2. CMAC-QRLS and Its Systolic Array

The QR-RLS Algorithm of CMAC can be simplified step by step based on the particular characteristic of CMAC that only g nonzero elements exist in the indicator of CMAC:

Table.4. Code of CMAC-QRLS

1. Initialization:

```
$T = n - g$
$For\ j = 1,2,\cdots,T\ Loop$
$\Omega_{1,j} = \delta$
    $For\ i = 2,3,\cdots,g+1\ Loop$
    $\Omega_{i,j} = 0$
    $End\ of\ Loop\ i$
$End\ of\ Loop\ j$
$For\ j = T+1, T+2,\cdots,n\ Loop$
$\Omega_{j-T,j} = \delta$
    $For\ i = j-T+1, j-T+2,\cdots,g+1\ Loop$
    $\Omega_{i,j} = 0$
    $End\ of\ Loop\ i$
$End\ of\ Loop\ j$
```

2. Recursive Operation: **H2**

(x_k, y_k) is provided, compute

2.1. Forward Computation

```
$EndSign = 0; EndAddress = T$
$For\ i = 1,2,\cdots g, Loop$          }
$\sigma_i = a_{l+i-1,k}$               } H1
$End\ of\ Loop\ i$                     }
$\sigma_{g+1} = y_k$
$For\ j = l, l+1,\cdots,T\ Loop$
$d = \sqrt{\sigma_1^2 + \Omega_{1,i}^2}$
$c = \Phi_{1,i}/d,\ s = \sigma_1/d$
    $For\ i = 1,2,\cdots,g+1\ Loop$
    $\Omega_{i,j} = c\Omega_{i,j} + s\sigma_i$
    $\sigma_i = -s\Omega_{i,j} + c\sigma_i$
    $End\ of\ Loop\ i$
$If\ all(\sigma_{out}) = 0$            }
$EndSign = 1$                          } H3
$EndAddress = j$                       }
$End\ If$                              }
    $For\ i = 1,2,\cdots g-1, Loop$
    $\sigma_i = \sigma_{i+1}$
    $End\ of\ Loop\ i$
$\sigma_g = 0$
$End\ of\ Loop\ j$
$If\ !EndSign$
    $For\ j = T+1, T+2,\cdots,n, Loop$
    $d = \sqrt{\sigma_{j-T}^2 + \Omega_{j-T,j}^2}$
    $c = \Phi_{j-T,j}/d,\ s = \sigma_{j-T}/d$
        $For\ i = j-T, j-T+1,\cdots,g+1\ Loop$
        $\Omega_{i,j} = c\Omega_{i,j} + s\sigma_i$
        $\sigma_i = -s\Omega_{i,j} + c\sigma_i$
        $End\ of\ Loop\ i$

    $End\ of\ Loop\ j$
$End\ of\ If$
```

2.2. Back Substitution

```
$If\ !EndSign$
$w_n = \Omega_{g+1,n} / \Omega_{g,n}$
    $For\ j = n-1, n-2,\cdots,T+1\ Loop$
    $temp = 0$
        $For\ i = j-T+1, j-T+2,\cdots,n\ Loop$
        $temp = temp + \Omega_{i,j} w_{i+T}$
        $End\ of\ Loop\ i$
    $w_j = (\Omega_{g+1,j} - temp) / \Omega_{j-T,j}$
    $End\ of\ Loop\ j$
$End\ If$
$For\ j = EndAddress, EndAddress-1,\cdots,1\ Loop$
$temp = 0$
    $For\ i = 2,3,\cdots,g\ Loop$
    $temp = temp + \Omega_{i,j} w_{i+j-1}$
    $End\ of\ Loop\ i$
$w_j = (\Omega_{g+1,j} - temp) / \Omega_{1,j}$
$End\ of\ Loop\ j$
```

3. Terminate

1. **Address mapping.(H1)**

Suppose that the indicator of x_k is $[0,0,\cdots,a_l\cdots a_{l+g-1},\cdots 0]$, where l is the address of the first nonzero element. Considering the result of unitary transformation in (21), the elements in front of a_l are not required to participate in the computation. So the unitary transformation begins from a_l and simultaneously from the lth column of Ω. In other words, it is not required the computation to begin from the first column and thus we can use the address mapping mechanism of CMAC to locate the beginning address for the forward computation.

2. **Change the triangular structure to a rectangular structure.(H2)**

The elements behind a_{l+g-1} are all zeros. Considering the procedure of unitary transformation, it can be seen that only g elements closely lower from the diagonal of Ω and the elements of the last row of Ω take part in the computation, so we can eliminate the elements unused and change the triangular structure to a rectangular structure of $(g+2)\times n$.

3. **End the forward computation and begin the back substitution.(H3)**

In the process of one epoch, the forward computation can be carried to the last column of Ω. However, when the forward computation reaches one column whose cell output signals are all zeros, it can also be ended. Then the back substitution begins from this column till the first column, i.e., the weights after this column remain unchanged.

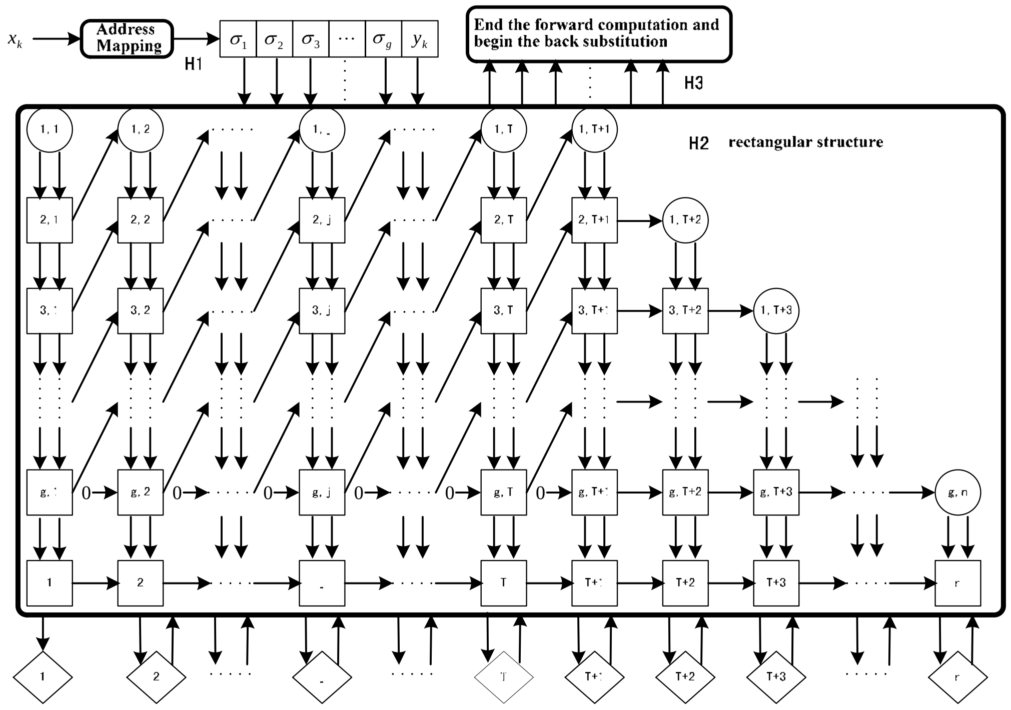
x_k
Address Mapping
H1
σ_1 σ_2 σ_3 $\cdots$ σ_g y_k
End the forward computation and begin the back substitution
H3
H2 rectangular structure

Figure.3. Systolic array of CMAC-QRLS

Combine the three simplifying methods above, the code in Table.3 can be rewritten in Table.4.

We call this latter simplified algorithm CMAC-QRLS. The systolic array of CMAC-QRLS is illustrated in Figure.3. For easy explanation, the various module functions of Figure.3 are descried by the corresponding code in Table.4 using the same symbol.

Similarly, the computation complexity of CMAC-QRLS can be calculated through the next table:

Table.5. computation complexity of CMAC-QRLS

	Multiplications	Divisions	Square Roots
Angle computer	$2n$	$2n$	n
Rotator	$4g \times n$		
Back substitution	$\frac{n \times (n-1)}{2}$	n	
Total	$\frac{n^2}{2} + 4g \times n + \frac{3}{2}n$	$3n$	n

Compared with Table.1, we find that the former simplified CMAC-RLS needs almost n^2 multiplications, whereas CMAC-QRLS requires nearly $\frac{n^2}{2}$ multiplications. However, CMAC-QRLS increases the operations of divisions and it also needs the operations of square roots. From these analyses, we can not decide which algorithm runs faster. So in order to compare their practical computation speed, the simulation to approximate same function in Matlab is carried out.

The function is:

$$y = \sin(2\pi x / 360) \qquad x \in [0,360] \tag{23}$$

CMAC uses the parameters of $Q = 360; g = 18$ and is trained using data with the sampling interval of 1. Q is the level of quantization.

Each algorithm is run eight times and the computation time results are listed in Table.6.

Table.6. computation time comparison

	1	2	3	4	5	6	7	8	Average
Simplified CMAC-RLS	24.44	24.11	23.93	23.96	24.00	24.00	24.06	24.10	24.07
CMAC-QRLS	11.22	11.14	11.20	11.18	11.22	11.19	11.13	11.19	11.18

Obviously, CMAC-QRLS runs faster than the previous algorithm. So compared with the former simplified CMAC-RLS algorithm, CMAC-QRLS reduces the computation time. It should be noticed that CMAC-QRLS is much easier to be implemented in hardware structure due to its pipelined systolic array, and therefore if it is realized in high-speed hardware, it may achieve rather faster computation speed compared with its software implementation.

As shown in Figure.3, CMAC-QRLS only needs about $(g+2) \times n$ memory cells. The memory storage is reduced. Moreover, due to the property of QR-RLS[18], the numerical stability is also improved simultaneously.

Considering the analyses above, it can be concluded that CMAC-QRLS exhibits more advantage in fast computation speed, low memory storage, pipelined hardware realization and high numerical stability, so it will be more suitable for implementation.

3.3. Continuous CMAC-QRLS

Since the rectangular shape of CMAC receptive field functions produce discontinuous staircase function approximation, by formulating CMAC with B-splines receptive field functions. Lane[4] has developed continuous CMAC-LMS which can provide the information of both functions and function derivatives.

B-splines can be constructed using deBoor-Cox recursive relation[19].

$$BS_{j,1}(x)=\begin{cases}1, & x\in[x_j,x_{j+1})\\ 0, & \text{otherwise}\end{cases} \tag{24}$$

$$BS_{j,m}(x)=\frac{x-x_j}{x_{j+m-1}-x_j}BS_{j,m-1}(x)+\frac{x_{j+m}-x}{x_{j+m}-x_{j+1}}BS_{j+1,m-1}(x),m>1,\ \ 0\le j\le n \tag{25}$$

and its first-order derivatives can be calculated

$$BS_{i,m}^{(1)}(t)=(k-1)\left[\frac{BS_{i,m-1}(t)}{t_{i+m-1}-t_i}-\frac{BS_{i+1,m-1}(t)}{t_{i+m}-t_{i+1}}\right] \tag{26}$$

So the higher derivative can be calculated recursively by (26)

Give n partitions over the interval $x\in[a,b)$, an m th order spline function $BS_{j,m}(x)$ can be constructed to approximate $f(x)$

$$f(x)\approx\hat{f}(x)=\sum_{j=1}^{n+m-1}w_jBS_{j,m}(x) \tag{27}$$

by using a linear combination of $BS_{j,m}(x)$ weighted by w_j.

B-splines receptive function is preferable to other spline functions due to its superior numerical and computational properties. The main properties include:

a) Positive on a bounded support:

$$BS_{j,m}(x)=\begin{cases}>0, & x\in[x_j,x_{j+m})\\ 0, & \text{otherwise}\end{cases} \tag{28}$$

b) Form a partition of unity

$$\sum_{j=0}^{n}BS_{j,m}(x)=1\quad x\in[x_{m-1},x_{n+1}) \tag{29}$$

To efficiently design continuous CMAC-QRLS chip, we have to develop cost-effective architectures for the recurrent B-splines evaluator. For easy explanation, we suppose that the order of B-splines function equals to the generalization factor and let $g = m = 4$. Thus the values of B-splines in CMAC-QRLS can be calculated through the following procedure:

$$\begin{tikzcd}
BS_{j+3,1} \arrow[r, "30"] \arrow[dr, "31"'] & BS_{j+3,2} \arrow[r, "30"] \arrow[dr] & BS_{j+3,3} \arrow[r, "30"] \arrow[dr] & BS_{j+3,4} = \sigma_4 \\
\text{(a)} & BS_{j+2,2} \arrow[r, "25"] \arrow[dr, "31"'] & BS_{j+2,3} \arrow[r, "25"] \arrow[dr] & BS_{j+2,4} = \sigma_3 \\
 & \text{(b)} & BS_{j+1,3} \arrow[r, "25"] \arrow[dr, "31"'] & BS_{j+1,4} = \sigma_2 \\
 & & \text{(c)} & BS_{j,4} = \sigma_1 \\
 & & & \text{(d)}
\end{tikzcd}$$

Figure.4. Computation procedure of B-splines in CMAC-QRLS

It can be seen from Figure.4 that (30), (31) are used in each side and only (25) is used in the internal part.

$$BS_{j,m}(x) = \frac{x - x_j}{x_{j+m-1} - x_j} BS_{j,m-1}(x), m > 1,\ \ 0 \le j \le n \qquad (30)$$

$$BS_{j,m}(x) = \frac{x_{j+m} - x}{x_{j+m} - x_{j+1}} BS_{j+1,m-1}(x), m > 1,\ \ 0 \le j \le n \qquad (31)$$

In order to implement B-splines in hardware, we need to arrange the data of in the registers properly. At first the rightmost register stores 1, and the following procedure corresponding to Figure.4 is depicted in Figure.5.

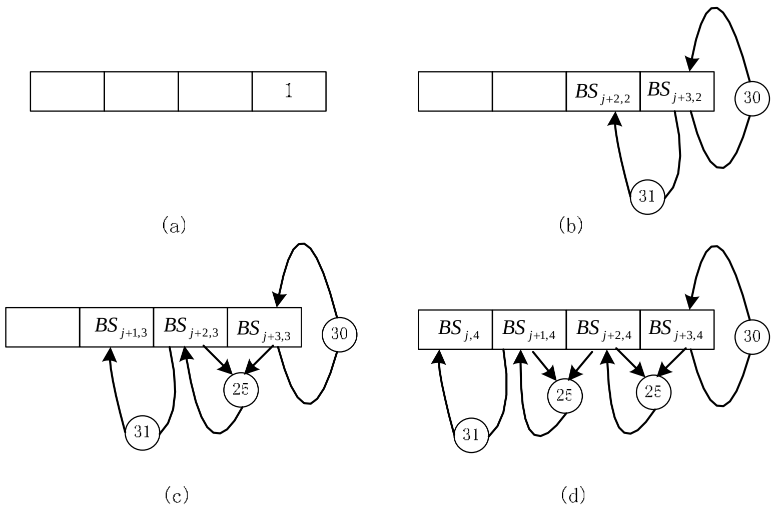

Figure.5. B-splines array in CMAC-QRLS

The indicator of continuous CMAC is calculated using address mapping and B-splines functions, so it is composed of floats instead of only binary elements in that of discrete CMAC. So one advantage of continuous CMAC-QRLS is that it is able to provide the output derivative information with respect to input when it is used to model on line.

4. Simulation Study

To show the effectiveness of the proposed continuous CMAC-QRLS algorithm, two examples are studied for demonstration.

4.1. Example 1: Approximate Sinusoid

As stated in previous sections, continuous CMAC-QRLS is very suitable for modeling. Next we will examine this capability through approximate a sine function:

$$f(x) = \sin(2\pi x / 360) \qquad x \in [0,360] \tag{32}$$

The simulation results in Figure.6 are based on the following choice of design parameters and conditions:

The parameters of continuous CMAC-QRLS are adopted as $Q = 20$ and $g = m = 4$

CMAC is trained using data with the sampling interval of 10, and checked with the sampling interval of 0.2.

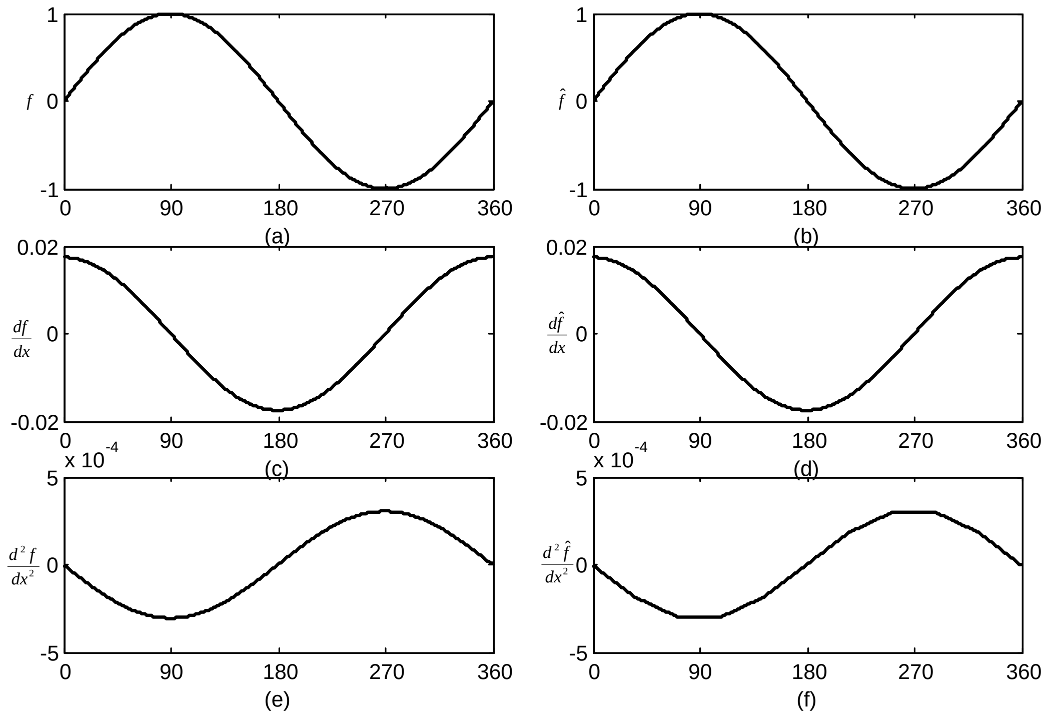

Figure.6 .Approximation ability of continuous CMAC-QRLS for Sinusoid

The errors of function values, first-order and second-order derivatives are summarized in Table.7 respectively.

Table.7. Various errors of continuous CMAC-QRLS

	function values	first-order derivatives	second-order derivatives
RMSE	0.2894e-4	0.7739e-5	1.5137e-6
MAE	3.7270e-4	7.9379e-5	10.240e-6

RMSE: root mean square error; MAE: max absolute error

As illustrated in Figure.6 and Table.7, the approximation results show that continuous CMAC-QRLS can approximate function and provide the information of derivatives well even if it is trained with incomplete training sets. So when it is used to model on line, it can provide us more accurate information than the discrete CMAC.

4.2. Example 2: Approximate Simple Pendulum

Now let us demonstrate the approximation ability of continuous CMAC-QRLS again via an elementary example of a simple pendulum.

The equation of motion of a simple pendulum is given by (see reference [20])

$$ml\ddot{\eta} = -mg\sin\eta - kl\dot{\eta} + \frac{1}{l}u \tag{33}$$

where u is the torque applied to the pendulum, η is the anticlockwise angle between the vertical axis through the pivot point and the rod, g is the gravity acceleration, and the constants k, l and m denote a coefficient of friction, the length of the rod and the mass of the bob, respectively. Set $a = g/l$, $b = k/m$, $c = 1/ml^2$ and define the state variables $x_1 = \eta$, $x_2 = \dot{\eta}$, system (33) can be written as its state equation:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -a\sin x_1 - bx_2 + cu \end{cases} \tag{34}$$

In this case, the parameters of CMAC are $Q = 20$ and $g = m = 4$.

The nominal parameters are $a = c = 10$ and $b = 1$.

The original state is $(0,0)$. The time span of t is $[0,8]$.

The input is varying and $u = 0.5\sin(2\pi t/80)$.

CMAC whose input is time t and output is the angle η is trained using data with sampling period of 0.4 to approximate the running process, and checked with the sampling period of 0.1.

The simulation results are depicted in next figure:

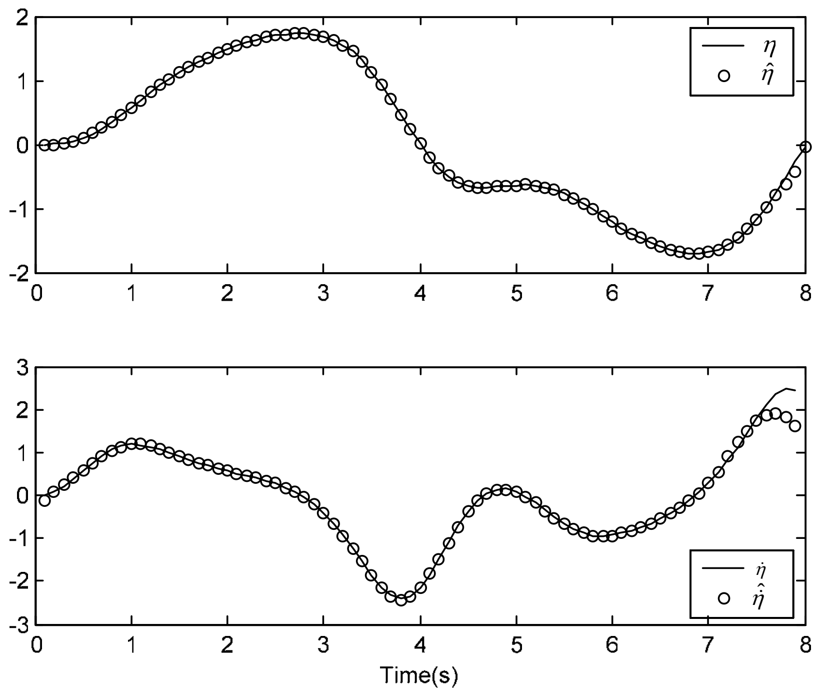

Figure7 .Approximation ability of continuous CMAC-QRLS for Simple Pendulum

In Figure.7, the solid curve is the original state of the pendulum, and the curve denoted by circle represents the state approximated by continuous CMAC-QRLS. Obviously, we even can get the information of angular velocity when continuous CMAC-QRLS is only used to approximate the angle with incomplete training sets. And owing to the more information of continuous CMAC-QRLS, it can be concluded that the angular acceleration may also be obtained and we do not discuss it here. Approximating angular velocity of around eighth second still requires the data after this time, so there exist relatively bigger errors for angular velocity around eighth second.

From these two examples, we find that adequate approximation results with a broad and uniform distributed incomplete training set can be obtained if the approximated function is smooth and the parameters of continuous CMAC-QRLS are well-chosen. All these demonstrated continuous CMAC-QRLS is a good approximation tool for modeling on line.

5. Conclusion

Least square adaptive algorithms based on the QR decomposition are very promising due to their numerically robust performance. In this paper, CMAC-RLS is simplified based on QR decomposition. Compared with the original CMAC-RLS algorithm, it reduces the memory storage, what's more, the proposed algorithm exhibits low time cost and pipeline structure. Subsequently we design the systolic array of continuous CMAC-QRLS which can provide the further information of derivatives. The simulation results show that CMAC-QRLS appears to be an appropriate algorithmic tool for modeling on line.

Acknowledgement: This work is supported by the national '985'high-level university fund, Hefei key science and technology plan and the youth fund of USTC. The authors also would like to thank the anonymous reviewers for their constructive comments, which helped improve the presentation.

Reference:

1. J.S.Albus, ”A new approach to manipulator control: The cerebellar model articulation controller(CMAC)”, Trans.ASME,J.Dyanmic Syst.,Meas.Contr.,Vol.97,pp.220-227,1975.
2. J.S.Albus,”Data storage in the cerebellar model articulation controller(CMAC)”, Trans.ASME J. Dynamics Syst.Means.Contr.,vol.97,pp.228-233,1975.
3. Chiang-Tsan Chiang and Chun-shin Lin,” CMAC with general basis functions”, Neural Networks, Vol.9, pp.1199-1211, 1996.
4. Stephen H.Lane,David A.Handelman,and Jack J.Gelfand, ”Theory and development of higher-order CMAC neural networks”, IEEE Control Systems Magazine, Vol.12, pp.23-30, 1992.
5. P.C.Parks and J.Militzer, ”Convergence properties of associative memory storage for learning control system.”, Automation and Remote Cotnrol,vol.50,pp254-286,1989
6. Yiu-fai Wong and Athanasios Siders, ”Learning convergence in the cerebellar model articulation controller”, IEEE trans.on nerural networks,vol.3,pp115-121,1992
7. Chun-shin Lin and Ching-Tsan Chiang, ”Learning convergence of CMAC technique”, IEEE trans.on neural networks,vol.8,pp.1281-1292,1997.
8. Chao He, Lixin Xu, Yuhe Zhang, ”Learning Convergence of CMAC Algorithm”, Neural Processing Letters, vol.14, pp.61-74,2001
9. Ker Jar-Shone,Kuo Yau_Hwang,Liu Bin-Da ,”Hardware realization of higher-order CMAC model for color calibration”, Proceedings of IEEE International Conference on Neural Networs, Perth, WA, Austrialia, 11/27/1995-12/01/1995,pp.1656-1661
10. W. T.Miller, Latham, P. J., and Scalera, S. M., "Bipedal Gait Adaptation for Walking with Dynamic Balance", Proceedings of the 1991 American Controls Conference, Boston, MA, vol.2, pp. 1603-1608, June, 1991.
11. W.T.Miller, F.H.Glanz and L.G.Kraft, ”Application of a general learning algorithm to the control of robotic manipulators”, Int.J.of Robotics Research, vol.6,pp.84-98,1987.
12. W.T.Miller, “Real-time application of neural networks for sensor-based control of robots with vision”, IEEE Transactions on Systems, Man and Cybernetics, Vol. 19,pp. 825 -831,1989
13. W.T.Miller,R.P.Hewes,F.H.Glanz,L.G.Kraft, ”Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller”, IEEE trans. on robotics and automation,vol.6,pp.1-9,1990
14. S.Manglevhedakar, ”An adaptive hierarchical model for computer vision”, Thesis, Louisiana State Univ. 1986.
15. A.Kolcz and N.M.Allinson,” Application of the CMAC input encoding scheme in the N-tuple approximation network”, IEE Proc.Comput.digital Techniques,vol.141,pp.177-183,1994
16. Ting Qin, Zonghai Chen, etc , ” A Learning Algorithm of CMAC Based On RLS”, Neural Processing Letters, 19: 49 – 61, 2004.
17. Simon Haykin,”Adaptive Filter Theory”, Prentice Hall ,2002
18. Yang, B., Bohme, J.F, ” Rotation-based RLS algorithms: unified derivations, numerical properties, and

parallel implementations”, IEEE Transactions on Signal Processing, Vol.40, pp.1151-1167, 1992

19. Wan Shengfu, “spline functiond and their application”, Northwestern Polytechnical University Press,1989

20. H. Khalil, Nonlinear Systems, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2002.