Research on New-CMAC with Differentiability Output and Its Learning Convergence

WANG Shi-tong[1,2], J. F. Baldwin[2], T. P. Martin[2]

[1](*Department of Computer Science, Eastchina Shipbuilding Institute, Zhengjiang* 212003, *China*)

[2](*Advanced Computing Research Center, Bristol University, UK*)

E-mail: zjstwang@public.zj.js.cn

http://www.zj.js.cn

Received April 18, 2000; accepted October 17, 2000

Abstract: In this paper, based on conventional CMAC (cerebellar model architecture controller) neural network and locally weighted regression, the improved New-CMAC with the same amount of memory as that of conventional CMAC is presented, which has the conventional output and its derivative information output and hence is especially appropriate for automatic control. Accordingly, the new learning algorithm is investigated, and its learning convergence is proved.

Key words: CMAC (cerebellar model architecture controller); learning algorithm; differentiability output; locally weighted regression

Cerebellar model architecture controller (CMAC)[1] is a table lookup based neural network, and is getting more and more applications, especially in real time control, due to its attractive properties of learning convergence and speed. A problem is that conventional CMAC cannot provide derivative information of its output. This creates both difficulty and inconvenience in the learning process that needs derivative information. One example is action dependent critic learning[2].

In this paper, we present the New-CMAC scheme that integrates locally weighted regression[3] with the conventional CMAC addressing technique to solve this problem. Derivatives exist everywhere except on the boundaries of quantized regions. With the use of the conventional CMAC addressing technique, data points in the input space are efficiently organized. Only data in the local area are used for regression. This limits the computational complexity efficiently. We will prove that the New-CMAC is a universal approximator and its new learning algorithm has the learning convergence.

This paper is organized as follows. In Section 1, the conventional CMAC is briefly reviewed. In Section 2, the New-CMAC with weighted regression and differentiability output is described. In Section 3, its universal approximation property is proved. In Section 4, the new learning algorithm is described, and then its learning convergence

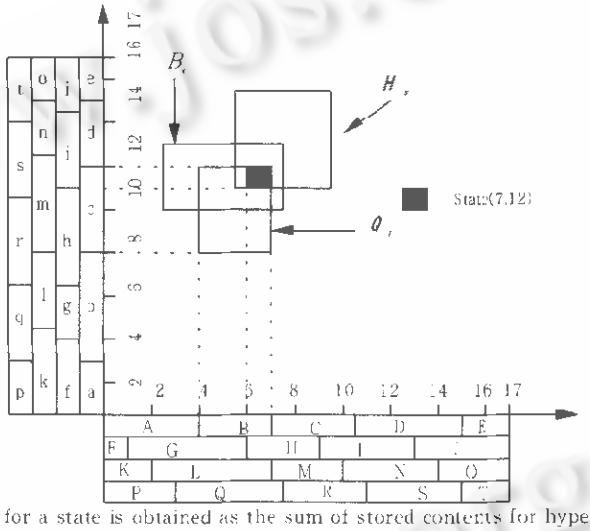
* Supported by the National Natural Science Foundation of China under Grant No. 6983004 (国家自然科学基金); British Royal Society

WANG Shi-tong was born in 1964. He is a professor and doctoral supervisor of the Department of Computer Science, Eastchina Shipbuilding Institute. His research areas include artificial intelligence, neural networks and fuzzy systems. **J. F. Baldwin** is a professor and the head of the Advanced Computing Research Center, University of Bristol, U. K. His research areas include artificial intelligence, neural networks and fuzzy systems. **T. P. Martin** is a Reader (associate professor) of the Advanced Computing Research Center, University of Bristol, U. K. His research areas include artificial intelligence, neural networks and fuzzy systems.

is investigated in Section 5. In Section 6, the conclusion is derived.

1 Conventional CMAC

We can view conventional CMAC as a technique with a basis function that has a constant value of a '1' within a restricted area. The area is a square, a cube or a hypercube, depending on the dimension of the input space. The technique can be explained by Fig. 1. Each input state variable is quantized into several discrete regions, named as Aa, Ab, Ba, etc., which are called hypercubes. If the quantization for each variable is shifted by one small interval (called an element), different hypercubes will be obtained. F, G, H, I, J for x_1 and f, g, h, i, j for x_2 are examples of such shifted regions. Fh, Fi, Hg etc. are new hypercubes resulting from the shifted regions. In most CMACs, no hypercube is formed by the combination of different layers such as 'A, B, C, D' and 'f, g, h, i, j'. With this kind of quantization and hypercube composition, each state is covered by N_r different hypercubes, where N_r is the number of elements in a complete block.



The output of CMAC for a state is obtained as the sum of stored contents for hypercubes covering the state. Let $\boldsymbol{x}=(x_1, x_2, \ldots, x_i, \ldots, x_n)^T$ be the state, which is the input $I(\boldsymbol{x})$ to a CMAC. In mathematical description, we can define a vector indicator $I(\boldsymbol{x})$ as

$$I(\boldsymbol{x})=(I_1(\boldsymbol{x}), I_2(\boldsymbol{x}), \ldots, I_j(\boldsymbol{x}), \ldots, I_P(\boldsymbol{x}))^T \tag{1}$$

where $I_j(\boldsymbol{x})=1$ if memory location j is used by one hypercube covering $\boldsymbol{x}$; $I_j(\boldsymbol{x})=0$, otherwise. The vector indicates the memory location used for storing information for $\boldsymbol{x}$. Let $M=[m_1, m_2, \ldots, m_P]^T$ denote the vector of all the centers in physical memory. With I as the memory usage indication vector, information stored for the state $\boldsymbol{x}$, i.e., the output of CMAC, can be expressed as

$$y(\boldsymbol{x})=\mathrm{M}\times\mathrm{I}(\boldsymbol{x})=\sum_{j=1}^{P} m_j I_j(\boldsymbol{x}) \tag{2}$$

In learning, the values of m_i are parameters to be determined. Although the output of CMAC can be expressed by Eq. (2), it is actually retrieved from just a small number (N_r) of memory locations allocated to the hypercubes covering the state.

2 New-CMAC with Weighted Regression and Differentiability Output

Now, we present the new CMAC scheme, called New-CMAC. We combine CMAC with locally weighted regression technique and make New-CMAC to have two outputs: (a) $y(\boldsymbol{x})$ as in Eq. (2); (b) derivative information

$y'(x)$. In many real time controls, it is necessary to know the derivative information. Hence, New-CMAC with differentiability output is more suitable for many practical applications than conventional CMAC.

CMAC addressing technique is adopted for systematically selecting a set of neighboring points. The same method of hypercube decomposition for CMAC is used but the output is computed differently. The New-CMAC scheme intends to have the target function value at the hypercube center stored at the memory allocated to that hypercube. To retrieve information for a given input x, the hypercubes covering x are first identified. The output for x is computed from a local regression model formed by using the data stored for the hypercubes. The construction of the local regression model and the computation of the function value for a given x will be introduced below followed by the often-used learning algorithm.

In discussing information retrieval, we assume that the correct data have been stored. The weighted regression technique assigns different weights for data points at different distances. Given an input point x, the weight for a hypercube depends on the distance from x to the center of the hypercube. A small modification of Eq. (1) will be enough to describe the change. The element $I_j(x)$ will be newly defined as a function of the distance between the input and the center of hypercube j as

$$I_j(x)=\begin{cases}\exp(-|x-C_j|/\sigma_j) & \text{if hypercube } j \text{ covers } x\\ 0 & \text{otherwise}\end{cases} \tag{3}$$

where C_j is the center of hypercube j, σ_j is the given constant. The centers and memory contents (C_j, m_j) for hypercubes that cover x are used to generate the local regression model. C_j and m_j are weighted before the regression is done. Suppose $z_1, z_2, \ldots, z_{N_c}$ indicate the weighted center of N_c involved hypercubes and $y_1, y_2, \ldots, y_{N_c}$ indicate the corresponding weighted memory contents. Let z_j and y_j be for hypercube k, then z_j and y_j are defined as

$$z_j=I_k(x)C_k \tag{4}$$

$$y_j=I_k(x)m_k \tag{5}$$

where $k=j$ or $k()j$. We take $k=j$ below for simplicity. Please note that this does not destroy our analysis, only for denotation simplicity. The locally weighted regression is to determine the vector of regression coefficients for the following equation to achieve the least square error:

$$y=zb \tag{6}$$

where $y=(y_1, y_2, \ldots, y_{N_c})^T$, $z=(z_1, z_2, \ldots, z_{N_c})^T$. Please note: z is an $N_c\times n$ matrix. Thus, b can be solved as

$$b=(z^Tz)^{-1}z^Ty \tag{7}$$

where b is an $n\times 1$ matrix. The effect of weighting is that the error for a distant point is considered less important. With b obtained in Eq. (7), the output $y(x)$ is computed as

$$y=x^Tb \tag{8}$$

The output is continuous and differentiable except at the boundary of a quantized element. The vector b gives the derivative information.

3 New-CMAC as a Universal Approximator

In this section, we will prove that New CMAC is a universal approximator, i.e., it can approximate any given real continuous function on a compact domain to arbitrary accuracy.

Theorem 1[4]. If the basis functions $f_i(x)$ are differentiable in arbitrary order, then their linear combination $\sum p_i f_i(x)$ is a universal approximator, where p_i is a real coefficient.

We will use the above theorem to prove Theorem 2.

Theorem 2. New-CMAC is a universal approximator.

Proof. Suppose $x=(x_1, x_2, \ldots, x_n)^T$, $C_j=(c_{j1}, c_{j2}, \ldots, c_{jn})^T$. In terms of Eq. (4), we have $z_j=(I_j(x)c_{j1}, \ldots,$

$I_j(\boldsymbol{x})c_{jn})^T$. In terms of Eq. (3), we derive that $I_j(x)c_{jk}(k=1,2,\ldots,n)$ is either exponential-type function of $\boldsymbol{x}$ or 0.

Thus, we know that $z=(z_1,z_2,\ldots,z_{N_e})$ is an $n\times N_e$ matrix in which each element is either exponential-type function of $\boldsymbol{x}$ or 0. Obviously, based on the above, we further know $(z^Tz)-1$ is also an $n\times n$ matrix in which each element is either exponential-type function of $\boldsymbol{x}$ or 0. Thus, $\boldsymbol{b}=(z^Tz)^{-1}z^Ty$ is an $n\times 1$ matrix, it can be further expressed as

$$\boldsymbol{b}=(z^Tz)^{-1}z^Ty=\begin{Bmatrix}\sum_{k=1}^{N_e}y_kI_{ik}\exp(-\varphi_{k1}(x,C_1,\ldots,C_{N_e}))\\ \vdots\\ \sum_{k=1}^{N_e}y_kI_{kn}\exp(-\varphi_{kn}(x,C_1,\ldots,C_{N_e}))\end{Bmatrix}=\begin{pmatrix}b_1\\ \vdots\\ b_n\end{pmatrix} \tag{9}$$

where I_{ki} is a constant or 0, φ_{ki} is a polynomial-type function about $x,C_1,\ldots,C_{N_e},i=1,2,\ldots,n;k=1,2,\ldots,N_e$. In terms of Eqs. (8) and (9), we have

$$y(\boldsymbol{x})=\sum_{i=1}^{n}b_ix_i=\sum_{i=1}^{n}\sum_{k=1}^{N_e}x_iy_kI_{ki}\exp(-\varphi_{ki}(x,C_1,\ldots,C_{N_e})) \tag{10}$$

We view $\exp(-\varphi_{ki}(x,C_1,\ldots,C_{N_e}))$ as the basis function. Obviously, $y(x)$ in Eq. (10) is the linear combination of those basis functions. Because $\exp(-\varphi_{ki}(x,C_1,\ldots,C_{N_e}))$ is differential in arbitrary order, in terms of Theorem 1, $y(x)$ in Eq. (10) is a universal approximator, i.e., New-CMAC is a universal approximator.

4 The New Learning Algorithm New-LA of New-CMAC

As it was pointed out previously, an important advantage of New-CMAC over conventional CMAC is that it can give both derivative information $\boldsymbol{b}$ and output $y(\boldsymbol{x})$. However, how do we design a learning algorithm for this New-CMAC?

The objective of learning is to determine the function values for the hypercube centers. However, these target values are not explicitly provided. Data available for training could be at any location in the input space. The memory contents must be derived from the available information. Given an $\boldsymbol{x}$, the new learning algorithm New-LA below uses the current memory contents to form the local regression model and then uses the model to evaluate the output for input $\boldsymbol{x}$. The difference between the target value and the evaluated value is used to modify the vector $\boldsymbol{b}$. Note that the new vector $\boldsymbol{b}$ is not memorized, but is used to compute an "estimated" target vector y_i(estimated value at hypercube centers) using Eq. (5). The guessed target values are used to update memory contents.

With the retrieval procedure given in Eqs. (6)~(8), if the target value at $\boldsymbol{x}$ is $y^*(\boldsymbol{x})$, the error for $\boldsymbol{x}$ is $E(\boldsymbol{x})=y^*(\boldsymbol{x})-y(\boldsymbol{x})$. Note that $y^*(\boldsymbol{x})$ is given. The rule for modifying b_k should be

$$\Delta b_k=-(1/2)\beta(\partial E^2(\boldsymbol{x})/\partial b_k)=\beta E(\boldsymbol{x})(\partial y(\boldsymbol{x})/\partial b_k)=\beta x_kE(\boldsymbol{x}) \tag{11}$$

where β is the learning rate $\beta\in(0,1)$. With the change in Eq. (11), the new $y(\boldsymbol{x})$ will be

$$\sum_{k=1}^{n}(b_k+\Delta b_k)x_k=\sum_{k=1}^{n}x_kb_k+\sum_{k=1}^{n}\beta x_kx_kE(\boldsymbol{x})$$

Let $\beta=1/\sum_{i=1}^{n}x_i^2$, then the error at $\boldsymbol{x}$ will be completely corrected. However, we may use a smaller updating rate and hence have the following rule for modifying the vector $\boldsymbol{b}$:

$$b_{k+1}=b_k+\alpha_1E(\boldsymbol{x})x_k/\sum_{i=1}^{n}x_i^2 \tag{12}$$

i.e.,

$$\Delta b_k=\alpha_1(y^*(\boldsymbol{x})-y(\boldsymbol{x}))x_k/\sum_{i=1}^{n}x_i^2 \tag{13}$$

where α_1 is the learning rate, $\alpha_1 \in (0,1)$.

$$\Delta m_k = -(1/2)\alpha_2(\partial E^2(\boldsymbol{x})/\partial m_k) = -\alpha_2 E(\boldsymbol{x})(\partial y(\boldsymbol{x})/\partial m_k) = -\alpha_2 E(\boldsymbol{x}) x_k(\partial/\partial m_k) =$$

$$\alpha_2 E(\boldsymbol{x}) x_k (z^T z)^{-1} z^T \begin{bmatrix} 0 \\ \vdots \\ I_k(\boldsymbol{x}) \\ \vdots \\ 0 \end{bmatrix} \tag{14}$$

where α_2 is the learning rate. Hence, we have the learning rule for m_k:

$$m_{k+1} = m_k + \Delta m_k \tag{15}$$

We summarize the above learning procedure to get the following new learning algorithm New-LA for New-CMAC.

New-LA Algorithm

1. Initialize all memory contents (i.e. m^t) to 0.

2. For a given input $\boldsymbol{x}$ and the target output $y^t(\boldsymbol{x})$, find all hypercubes that cover the input x.

3. Use Eqs. (4) and (5) to obtain the weighted center vectors and the weighted memory contents for all involved hypercubes. They are z_j and y_j, $j = 1, 2, \ldots, N_c$.

4. Compute $\boldsymbol{b} = (z^T z)^{-1} z^T y$ as in Eq. (7).

5. Compute $y(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{b}$ as in Eq. (8).

6. Compute $E(\boldsymbol{x}) = y^t(\boldsymbol{x}) - y(\boldsymbol{x})$.

7. Update b_i by Eq. (12).

8. Update m_k by Eq. (15).

9. Goto Step 2 if the learning result is not satisfactory.

For the above learning algorithm New-LA, a problem occurs, i.e., does it converge? Our answer is certain. Considering Eqs. (12)~(15), we can easily see that if we can prove Eqs. (12) and (14) have the learning convergence, then the above algorithm does converge. In the following section, we will give the detailed analysis.

5 On the Learning Convergence of New-LA Algorithm

In this section, we will prove that New-LA algorithm has the learning convergence.

Considering algorithm New-LA's Steps 8, 9, 2, 3, 4, 5, in terms of the definitions of z, we know that y is a linear combination of m_k when this algorithm excutes Step 5. Therefore, we can further express $y = A\boldsymbol{m}$, where $A = (a_{ij})_{N_r \times N_c}$ which is bounded, i.e., all elements are bounded in terms of the definitions of z_j, I_j; $\boldsymbol{m} = (m_1, m_2, \ldots, m_{N_c})^T$. Given the real output y^t for input vector $\boldsymbol{x}$, the learning rule is

$$\Delta m_k = \gamma\left(y^t - \sum_{i=1}^{N_c} a_{ki} m_i\right) a_{kk} \tag{16}$$

where γ is the learning rate.

Comparing Eq. (16) with Eq. (14), we have

$$a_{kk} = x_k (z^T z)^{-1} z^T \begin{bmatrix} 0 \\ \vdots \\ I_k(\boldsymbol{x}) \\ \vdots \\ 0 \end{bmatrix}$$

$$y(\boldsymbol{x}) = \sum_{i=1}^{N_c} a_{ki} m_i$$

From the above analysis, we derive that in order to prove that the learning rule Eq. (15) has the learning convergence, we only need to prove that Eq. (16) has the learning convergence.

We define $\boldsymbol{m}_s^{(i)}$ as the vector of weights before the sth sample is presented in the ith iteration of learning; N_s denotes the number of samples. We consider the case that a set of N_s training data is repeatedly presented to the learning rule.

With Eq. (16), we have

$$\boldsymbol{m}_s^{(i)}=\boldsymbol{m}_{s-1}^{(i)}+\Delta\boldsymbol{m}_{s-1}^{(i)}=\boldsymbol{m}_{s-1}^{(i)}+\gamma(y_{s-1}^*-\boldsymbol{A}_{s-1}\boldsymbol{m}_{s-1}^{(i)})(a_{11}^{s-1},\ldots,a_{N_e^{s-1}N_e^{s-1}}^{s-1})^T= \tag{17}$$
$$\boldsymbol{m}_{s-1}^{(i)}+\gamma(y_{s-1}^*-\boldsymbol{A}_{s-1}\boldsymbol{m}_{s-1}^{(i)})R_{s-1}$$

where $R_{s-1}=(a_{11}^{s-1},\ldots,a_{N_e^{s-1}N_e^{s-1}}^{s-1})^T$. With Eq. (17), the difference in the vector $\boldsymbol{m}_s^{(i)}$ between two consecutive iterations i and $i+1$ is calculated as

$$D\boldsymbol{m}_s^{(i)}=\boldsymbol{m}_s^{(i+1)}-\boldsymbol{m}_s^{(i)}=\boldsymbol{m}_{s-1}^{(i+1)}+\Delta\boldsymbol{m}_{s-1}^{(i+1)}-(\boldsymbol{m}_{s-1}^{(i)}+\Delta\boldsymbol{m}_s^{(i)})= \tag{18}$$
$$=D\boldsymbol{m}_{s-1}^{(i)}+\gamma(y_{s-1}^*-\boldsymbol{A}_{s-1}\boldsymbol{m}_{s-1}^{(i+1)})R_{s-1}-\gamma(y_{s-1}^*-A_{s-1}\boldsymbol{m}_s^{(i)})R_{s-1}=(\boldsymbol{E}-\gamma\boldsymbol{A}_{s-1}R_{s-1})D\boldsymbol{m}_{s-1}^{(i)}$$

where $\boldsymbol{E}$ is the identity matrix. We define $D\boldsymbol{m}_0^{(i)}=D\boldsymbol{m}_{N_s}^{(i-1)}$, $I(\boldsymbol{x}_0)=I(\boldsymbol{x}_{N_s})$. For simplicity, we define $E_{s-1}=(\boldsymbol{E}-\gamma\boldsymbol{A}_{s-1}R_{s-1})$, in which each element is a bounded function about $\boldsymbol{x}_s$. Thus, we have

$$D\boldsymbol{m}_s^{(i)}=E_{s-1}E_{s-2}\ldots E_1D\boldsymbol{m}_0^{(i)}=(E_{s-1}E_{s-2}\ldots E_1E_{N_s}\ldots E_s)D\boldsymbol{m}_s^{(i-1)}=(E_{s-1}E_{s-2}\ldots E_1E_{N_s}\ldots E_s)^iD\boldsymbol{m}_s^{(0)} \tag{19}$$

Define $F_s=(E_{s-1}E_{s-2}\ldots E_1E_{N_s}\ldots E_s)$, then $F_s^i=(E_{s-1}E_{s-2}\ldots E_1E_{N_s}\ldots E_s)^i$. We can see that when γ is sufficiently small, each element in E_k will be less than 1, therefore each element in $\boldsymbol{F}_s^i$ is also less than 1. Please note that $D\boldsymbol{m}_s^{(0)}$ is the accumulation of Δm, that is

$$D\boldsymbol{m}_s^{(0)}=\Delta\boldsymbol{m}_s^{(0)}+\Delta\boldsymbol{m}_{s+1}^{(0)}+\ldots+\Delta\boldsymbol{m}_{N_s}^{(0)}+\Delta\boldsymbol{m}_1^{(0)}+\ldots+\Delta\boldsymbol{m}_{s-1}^{(0)} \tag{20}$$

Theorem 3. The learning rule Eq. (16) (or Eq. (15)) has the learning convergence if the number of iterations approaches infinity and the learning rate approaches 0.

Proof. Please see appendix.

Now, let us consider the learning rule Eq. (12) or Eq. (13). In terms of Step 5 of algorithm New-LA. It is obvious that $y(\boldsymbol{x})$ is the combination of b_i. Therefore, we can easily apply the above proof process to prove that the learning rule Eq. (12) or Eq. (13) has the learning convergence.

Theorem 4. The learning rule Eq. (12) or Eq. (13) has the learning convergence if the number of iterations approaches infinity and the learning rate approaches 0.

Now let's examine the effects of different New-CMAC structures on the performance of the learning scheme and compare New-CMAC with conventional CMAC. We take the function $y^*=f(x_1,x_2)=\sin x_1\sin x_2$, and define two error measurements as follows:

$\text{error}_y=\sum|f(x_1,x_2)-\hat{f}(x_1,x_2)|$, for all samples,

$\text{error}_{d-x_i}=\sum\left|\frac{\partial f}{\partial x_i}-\frac{\partial\hat{f}}{\partial x_i}\right|$, for all samples,

where error_y, error_{d-x_i} denote the sum of all absolute errors over a set of sampled points for the function values and for the derivatives, $\hat{f}(x_1,x_2)$ denotes the computed output from CMAC model.

Table 1 lists the error measurements for the learning performance of different-sized structures. Each structure is denoted by a label such as 9e8b, which represents a structure with 8 blocks for each variable and 9 elements for each complete block. The use of longer memory size results in more accurate results.

Table 2 lists the error measurements for new-CMAC and conventional CMAC.

Table 1 Comparison between different structures

Structure	Memory	$Error_y$(training)	$Error_y$(test)	$Error_{d-x_1}$(training)	$Error_{d-x_2}$
5e8b	320	23.1	19.16	118.93	115.59
7e8b	448	17.1	12.09	88.94	94.83
9e8b	576	14.1	9.71	81.52	85.97
9e6b	324	25.6	15.48	109.75	127.03
9e10b	900	8.9	4.09	56.48	74.17

Table 2 Comparison between New-CMAC and Conventional CMAC structures

Scheme	Structure	$Error_y$(training)	$Error_y$(test)	$Error_{d-x_1}$(training)	$Error_{d-x_2}$
Conventional	5e8b	34.30	28.76	—	
CMAC	9e8b	27.54	19.28	—	—
	9e10b	16.90	14.93	—	—
	5e8b	23.10	19.16	118.93	115.59
New-CMAC	9e8b	14.10	9.71	81.52	85.97
	9e10b	8.90	4.09	56.48	74.17

6 Conclusion

In this paper, we present a novel New-CMAC scheme. Compared with other CMAC schemes, its major advantage is that it can provide both the output and its derivative information with the same memory of conventional CMAC. Our theoretical analysis shows that New-CMAC is a universal approximator and its new learning rules have the learning convergence. This New CMAC is especially suitable for real-time controls which need derivative information.

Further research work is how to extend our present results to fuzzy CMAC or other CMAC, etc.

Acknowledgements This research is supported by British Royal Society and the National Natural Science Foundation of China.

References:

[1] Albus, J. A new approach to manipulator control: the cerebellar model articulation controller (CMAC). Journal of Dynamic Systems, Measurement and Control. Transcations ASME, 1975,(97):200~227.

[2] Werbos, P. Neural network for control and system identification. In: Proceedings of the IEEE Conference Decision and Control. 1989.

[3] Fan, J., *et al*. Variable bandwidth and local linear regression smoothers. Journal of Ann. Stat., 1992,20(4):2008~2036.

[4] Wei, Guang, *et al*. Research on global approximation of feedforward neural networks. Journal of Information and Control, 1997,(2):1~9.

[5] Wang, Shi-tong. Neural-Fuzzy Systems and Their Applications. Beijing: The Publishing House of Beijing University of Aeronautics and Astronautics, 1998 (in Chinese).

附中文参考文献:

[5] 王士同.神经模糊系统及其应用.北京:北京航空航天大学出版社,1998.

具有微分输出的神经网络 New-CMAC 及其学习收敛性

王士同[1,2], J.F. Baldwin[2], T.P. Martin[2]

[1](华东船舶工程学院 计算机系,江苏 镇江 212003)

[2](英国 Bristol 大学 高级计算研究中心,英国)

摘要: 基于传统的 CMAC 神经网络和局部加权回归技术,提出了与传统 CMAC(cerebellar model articulation computer)有着同样存储空间量的改进的新 CMAC 网络 New-CMAC,它具有传统的输出和具有其微分信息的输出,因而更适合于自动控制.接着,又提出了其新的学习算法,并研究了其学习收敛性.

关键词: CMAC(cerebellar model articulation computer);学习收敛;模糊泛集合;学习规则

中图法分类号: TP183 **文献标识码**: A

Appendix: Proof of Theorem 3

In terms of Eq. (18), we have

$$m_s^{(i+1)}=Dm_s^{(i)}+m_s^{(i)}=Dm_s^{(i)}+Dm_s^{(i-1)}+m_s^{(i-1)}=Dm_s^{(i)}+Dm_s^{(i-1)}+\ldots+Dm_s^{(1)}+Dm_s^{(0)}+m_s^{(0)}=$$

$$\sum_{k=0}^{i}Dm_s^{(k)}F_s^k+m_s^{(0)}$$

Because $m_s^{(0)}=\Delta m_1^{(0)}+\ldots+\Delta m_{s-1}^{(0)}+m_1^{(0)}$, in terms of Eq. (20), we further have

$$m_s^{(i+1)}=\sum_{k=0}^{i}(\Delta m_1^{(k)}+\Delta m_{s+1}^{(i+1)})+\ldots+\Delta m_{N_s}^{(i+1)}+\Delta m_1^{(0)}+\ldots+\Delta m_{s-1}^{(0)}F_s^k+m_s^{(0)}=$$

$$\sum_{k=0}^{i}\gamma F_s^k[(y_s^*-A_sm_s^{(0)})R_s+(y_{s+1}^*-A_{s+1}m_{s+1}^{(0)})R_{s+1}-\ldots-(y_{N_s}^*-A_{N_s}m_{N_s}^{(0)})R_{N_s}+(y_{s-1}^*-A_1m_1^{(1)})R_{s-1}-\ldots+$$

$$(y_{s-1}^*-A_{s-1}m_{s-1}^{(1)})R_{s-1}]+\Delta m_1^{(0)}+\ldots+\Delta m_{s-1}^{(0)}+m_1^{(0)}=$$

$$\sum_{k=0}^{i}\gamma F_s^k\Big[\sum_{l=1}^{N_s}y^*R_l-A_sm_s^{(0)}R_s-A_{s+1}m_{s+1}^{(0)}R_{s+1}-A_{N_s}m_{N_s}^{(0)}R_{N_s}-A_1m_1^{(0)}R_1-\ldots-A_{s-1}m_{s-1}^{(0)}R_{s-1}\Big]+m_1^{(0)}+$$

$$((y_{s-1}^*-A_1m_1^{(1)})R_1+\ldots+(y_{s-1}^*-A_{s-1}m_{s-1}^{(1)})R_{s-1})\gamma-$$

$$\sum_{k=0}^{i}\gamma F_s^k\Big[\sum_{l=1}^{N_s}y_l^*R_l-A_sm_s^{(0)}R_s-A_{s+1}m_{s+1}^{(0)}R_{s+1}-A_{N_s}m_{N_s}^{(0)}R_{N_s}-A_1m_1^{(1)}R_1-\ldots-A_{s-1}m_{s-1}^{(1)}R_{s-1}\Big]+m_1^{(0)}+O(\gamma) \tag{21}$$

Let us observe:

$$A_jm_j^{(1)}R_j=A_j(m_{j-1}^{(1)}+\Delta m_{j-1}^{(1)})R_j=A_j(m_1^{(0)}+\Delta m_1^{(0)}-\ldots+\Delta m_{N_s}^{(0)}-\Delta m_1^{(0)}+\ldots+\Delta m_{j-1}^{(0)})R_j=$$

$$A_jm_1^{(0)}R_j+O(\gamma u_j)$$

where u_j is bounded. Similarly, we have

$$A_jm_j^{(0)}R_j=A_j(m_{j-1}^{(0)}+\Delta m_{j-1}^{(0)})R_j=A_j(m_1^{(0)}+\Delta m_1^{(0)}+\ldots+\Delta m_{j-1}^{(0)})R_j=$$

$$A_jm_1^{(0)}R_j+O(\gamma v_j)$$

where v_j is also bounded. Thus, for Eq. (21), we have

$$\lim_{\gamma\to 0,i\to\infty}\lim m_s^{(i+1)}=\lim_{\gamma\to 0,i\to\infty}\lim\Big\{\sum_{k=0}^{i}\gamma F_s^k\Big[\sum_{l=1}^{N_s}(y_l^*-A_lm_1^{(0)})R_l+O(\gamma u_j)+O(\gamma v_j)\Big]+m_1^{(0)}+O(\gamma)\Big\}=$$

$$\lim_{\gamma\to 0,i\to\infty}\lim\Big\{\gamma(E-F_s)^{-1}\Big[\sum_{l=1}^{N_s}(y_l^*-A_lm_1^{(0)})R_l+O(\gamma u_j)+O(\gamma v_j)\Big]+m_1^{(0)}+O(\gamma)\Big\} \tag{22}$$

Because

$$F_s=E_{s-1}E_{s-2}\ldots E_1E_{N_s}\ldots E_s=E-\gamma\sum_{l=1}^{N_s}A_lR_l+O(\gamma^2) \tag{23}$$

Therefore, substituting Eq. (23) into Eq. (22), we obtain

$$\lim_{\gamma\to 0}\lim_{i\to\infty}\boldsymbol{m}_s^{(i+1)}=\lim_{\gamma\to 0}\lim_{i\to\infty}\left\{\left(\sum_{l=1}^{N_s}\boldsymbol{A}_l R_l\right)^{-1}\left[\sum_{l=1}^{N_s}(y_l^*-\boldsymbol{A}_l\boldsymbol{m}_1^{(i)})R_l+O(\gamma u_j)+O(\gamma v_j)\right]+\boldsymbol{m}_1^{(i)}+O(\gamma)\right\}=$$

$$\left(\sum_{l=1}^{N_s}\boldsymbol{A}_l R_l\right)^{-1}\sum_{l=1}^{N_s}y_l^* R_l$$

This means that the learning rule has the learning convergence, and converges to $=\left(\sum_{l=1}^{N_s}\boldsymbol{A}_l R_l\right)^{-1}y_l^* R_l$, therefore, this theorem holds.