PAMUKKALE ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
PAMUKKALE UNIVERSITY ENGINEERING COLLEGE
MÜHENDİSLİK BİLİMLERİ DERGİSİ
JOURNAL OF ENGINEERING SCIENCES
YIL : 2001
CİLT : 7
SAYI : 3
SAYFA : 323-329

A COMBINE ALGORITHM FOR A CMAC NETWORK

Selahattin SAYIL
Pamukkale Üniversitesi, Teknik Eğitim Fakültesi, Elektronik Eğitimi Bölümü, Kınıklı/Denizli

Geliş Tarihi : 15.03.2001

ABSTRACT

The performance of a CMAC neural network depends on the training algorithms and the selection of input points. Papers have been published that explain CMAC algorithms but little work has been done to improve existing algorithms. In this paper, the existing algorithms are first explained and then compared using computational results and the algorithm properties. Improvements are made to the recommended Maximum Error Algorithm by using a "Combine Algorithm" approach. In this method, CMAC network is first trained by using Neighborhood Training Algorithm and then trained by Maximum Error Algorithm for fine-tuning of CMAC network. Faster initial convergence is achieved for the recommended Maximum Error Algorithm. This approach may reduce the training time and accelerate the initial learning which is very important in many control applications.

Key Words : CMAC training, CMAC algorithms, Maximum error algorithm

CMAC NÖRAL DEVRESİ İÇİN KOMBİNE BİR ALGORİTMA

ÖZET

Bir CMAC nöral devresinin başarısı, öğrenme algoritmasına ve giriş vektörlerin seçimine bağlıdır. CMAC algoritmasını geliştirme konusunda yeterli bir çalışma yapılmamıştır. Oysa, bir algoritmanın zayıf yönleri diğer algoritmanın güçlü yönleriyle birleştirilebilir. Bu makalede önce algoritmaların açıklanması yapılmış olup bunu takiben algoritma özellikleri ve bilgisayar simulasyonları kullanılarak karşılaştırımı yapılmıştır. 'Maksimum Hata Algoritması'nın sonuç olarak en iyi performansı göstermiştir. Geliştirim, önerilen Maksimum Hata Algoritması üzerine olmuş ve önerilen 'Kombine Algoritma' metodu CMAC'in başlangıçta fonksiyonu çok hızlı kavramasını sağlamıştır. Yeni bulunan bu teknikle kontrol sistemlerinde çok büyük öneme sahip olan fonksiyonu hızlı kavrama oranı arttırılabilmektedir.

Anahtar Kelimeler : CMAC nöral devresi, CMAC algoritması, Maksimum hata algoritması

1. INTRODUCTION

The CMAC (Cerebellar Modular Articulation Controller) is a mathematical formalism developed by (Albus, 1975a; Albus, 1975b) to model the information processing characteristics of the cerebellum. The CMAC as a controller computes control values by referring a memory look-up table where those control values are stored. It is believed that the biological organisms use some form of memory driven control systems. Memory table basically stores the relation between input and output or the control function. After memory table is formed, the output values are calculated just by averaging the contents of memory elements addressed by the input vector.

In comparison to the other neural networks, CMAC has the advantage of very fast learning and it has the unique property of quickly training certain areas of memory without affecting the whole memory

structure due to local training property of CMAC. The CMAC has an advantage in training speed and this property is very important in real time applications. For example, in an adaptive flight control system, the CMAC can learn as the flights are performed. The same benefits may be gained in other adaptive control systems (Cembrano et al., 1999).

2. CMAC STRUCTURE

In CMAC, learning is iterative and 'teacher' supplies important information about the desired function to be learned and CMAC adapts itself using this feedback information (Figure 1). This is called supervised learning. The input selects the active weights of CMAC depending on its value. Then active weights are summed and averaged to calculate an output. This output is compared with the desired output, and CMAC uses the difference to adjust its weights to get the desired response.

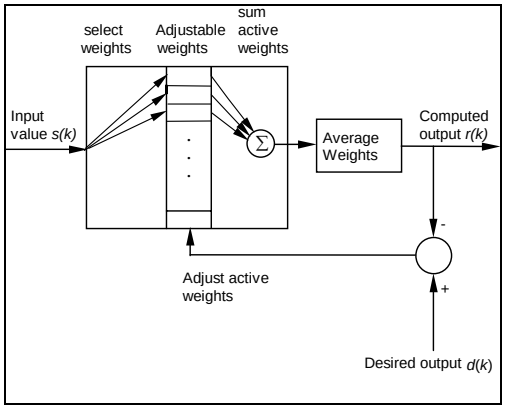

Figure 1. Supervised learning in the CMAC structure

Albus based CMAC on the perceptron by Rosenblatt and the mappings (Rosenblatt, 1961).

$$S \rightarrow A \rightarrow P \qquad (1)$$

Where S is sensory input vector, A is the binary association matrix, and P is the output response.

The first mapping is a non-linear transformation that maps the input variable s, into a binary vector that indeed represents the location of chosen weights by the corresponding input. The generalization factor, ρ, is a design parameter and determines the number of active cells with "ones" in the association vector. In the second mapping of the CMAC, the output values are formed first by summing of all the weights of the cells in the association vector which have been excited by a particular input and then taking their average.

In the training phase, weights are adjusted in such a way that the output of the CMAC approaches the desired output. Training of weights takes place in a supervised learning environment. Given the k-th training sample the error between the desired output $d(k)$ and the calculated output $r(k)$, is $e(k) = d(k) - r(k)$. This error is then used as a correction to each of the memory cells excited by the k-th input. For each excited memory element, the correction of weight is $\Delta W = \gamma\, e(k)$, where, γ is the learning factor.

Miller et al. (1982) identifies the CMAC training algorithm as the well-known Least Mean Squares (LMS) algorithm :

$$W(k) = W(k-1) + \gamma\{d(k) - \frac{1}{\rho} a^T(k) W(k-1)\} a(k) \qquad (2)$$

where $a(k)$ is the binary association vector, and ρ is the generalization parameter.

3. COMPARISON OF CMAC TRAINING ALGORITHMS

3. 1. Selection of Training Input Points

Parks and Militzer (1992) discusses Cyclic and Random Training methods for the CMAC algorithms. In Cyclic (Sequential) Training, a cycle is defined for the input training points and during the training this cycle is repeated until a desired performance is reached. If the training points are in the same neighborhood of the previous input points, good output convergence may not be obtained. Random Training method could be used to prevent the cross-talk (interference) between the input training points. A pseudo-random number generator could be used to generate uniformly distributed random numbers at each training session. In this manner, learning interference is greatly minimized because of the reduction of repetitive learning at the same neighborhood.

3. 2. Comparison of Training Algorithms

The following algorithms were investigated :

1. Albus Instantenous Learning (Nonbatch) Algorithm.
2. Batch Learning Algorithm.
3. Maximum Error Algorithm.
4. Gram-Schmidt Procedure.

5. Partially Optimized Step Length Algorithm.
6. Moving Average Algorithm.

In CMAC, weight updates can be made after each presentation (Non-batch Algorithm) or after a batch of presentations (Batch Algorithm). Instantaneous Error Correction Algorithm developed by Albus (Albus, 1975b) updates the network weight vector after each training pair is presented to the network. The rate of convergence of the algorithm depends directly on the orthogonality (no interference between patterns) of the training patterns (Brown and Harris, 1994).

A new learning algorithm, Gram-Schmidt Procedure are derived which produce weight changes perpendicular to the previous L weight changes (L = user defined parameter) and so the learning is orthogonal for the last L training examples (Parks and Militzer, 1992; Brown and Harris, 1994).

The Moving Average Algorithm uses a counter vector c, to indicate how often a weight has been updated and the update is forwarded to less frequently changed weight elements for a particular input (Parks and Militzer, 1992).

The Maximum Error Training Algorithm, developed by Parks and Militzer (1992), is based on the algorithm proposed by (Albus, 1975b). The algorithm takes L training input points consisting of the current training input point and L-1 past input points and selects the one with the largest output error.

Partially Optimized Step Length Algorithm tries to minimize the sum of errors for the last L training patterns rather than minimizing the square of error for one input. The details of these algorithms can be found in (Albus, 1975a-b; Brown and Harris, 1992; Brown and Harris, 1994 and Parks and Militzer, 1992).

In the comparison, the computational results and algorithm properties have been utilized. The training is performed either in a cyclic or in a random manner. The following two functions are used as target functions for generating the training data:

Case A) An arbitrary mathematical function f (x) which is defined for $-180^\circ \le x \le 180^\circ$:

$$f(x) = 2\left[\sin(x) - \frac{1}{2}\sin(2x) + \frac{1}{3}\sin(3x)\right] .\left[\cos(x) - \frac{1}{2}\cos(2x) + \frac{1}{3}\cos(3x)\right] \qquad (3)$$

Case B) A composite function which has a discontinuous first derivative at $x = (N\text{-}1)/3$ where N=361, and g(x) that is defined for $0^\circ \le x \le 360^\circ$

$$g(x) = \frac{1}{e^3 - 12 + 5}\left[e^{\frac{3x}{N-1}} - 6\left|\frac{3x}{N-1} - 1\right| + 5\right] \qquad (4)$$

Desirable properties of learning algorithms are initial fast learning and long-term convergence. The computational time of each algorithm is also taken into consideration. These criteria formed basis for comparison. The following abbreviations are made:

ALC	:	Albus (nonbatch) Cyclic Training
ALR	:	Albus (nonbatch) Random Training
AVC	:	Moving Average Cyclic Training
AVR	:	Moving Average Random Training
GRC	:	Gram-Schmidt Proc. Cyclic Training
GRR	:	Gram-Schmidt Proc. Random Training
MAC	:	Maximum Error Cyclic Training
MAR	:	Maximum Error Random Training
OSC	:	Optimized Step Cyclic Training
OSR	:	Optimized Step Random Training
BATC	:	Batch Alg. with Cyclic Training
BATR	:	Batch Alg. with Random Training

In both cases, the target functions are the given functions in (3) and (4), and the parameters in the training are: ρ=10 and B=361 (Resolution of 1 degree).

Comparing the two cases, the second function (Case B) is actually harder than the first one in terms of CMAC learning because of its discontinuous first derivative. From the results it can easily be seen that the Albus Nonbatch Algorithm is the least time requiring algorithm for training data and is convenient for on-line learning because of its easy algorithm implementation. But there is a risk if modelling error or measurement noise exists in the network. In that case of mismatch, the weight vector never converges to its optimal value, instead it converges in a domain which surrounds this optimal value, called minimal capture zone (Brown and Harris, 1994).

As it can be seen in Tables 1 and 2, when Cyclic Training is applied, slow initial convergence occurs. This is due to the interference which is inherent in the Cyclic Training nature. The employment of Random Training may improve initial convergence. The drawback in the Random Training is that it may sometimes get stuck since the Random Training process can choose points at which it is already reasonably well trained. At some point CMAC

may not learn anything. The work done by (Ellison, 1988) also explains this fact.

Table 1. Comparison of Training Algorithms for Case A

Algorit.	*L*	*361 pts. =1epoch e(rms)*	*3610 pts. =10 epoch e(rms)*	*18050 pts. =50 epoch e(rms)*	*36100 pts. 100 epoch e(rms)*	*Exec.time / single point t(msec)*
ALC	-	81.78 %	10.20%	0.139%	0.076%	3.30
ALR	-	6.38 %	1.130%	0.198%	0.073%	3.18
AVC	-	4.78 %	0.910%	0.533%	0.460%	5.50
AVR	-	4.61%	0.920%	0.543%	0.492%	3.18
GRC	2	30.76 %	0.185 %	0.062 %	0.016 %	40.41
GRC	10	18.62 %	0.105 %	0.013 %	0.010 %	195.00
GRR	2	11.02 %	0.388 %	0.089 %	0.021 %	42.40
GRR	10	5.88 %	0.112 %	0.048 %	0.015 %	212.13
MAC	10	21.24 %	0.375 %	0.032 %	0.015 %	30.00
MAC	50	3.304 %	0.159 %	0.029 %	0.012 %	126.12
MAR	10	5.701 %	1.159 %	0.100 %	0.061 %	31.81
MAR	50	3.312 %	0.411 %	0.074 %	0.031 %	139.00
OSC	2	67.48 %	1.540 %	0.096 %	0.057 %	10.50
OSC	10	27.04 %	0.405 %	0.145 %	0.090 %	87.42
OSR	2	5.80 %	1.299 %	0.156 %	0.061 %	10.61
OSR	10	5.49 %	1.230 %	0.191 %	0.067 %	88.31
		361 epch	*3610 epoch*	*18050 epoch*		
BATC	-	0.823%	0.176%	0.060%	-	1019
BATR	-	0.928%	0.220%	0.067%	-	1020

Table 2. Comparison of Training Algorithms For Case B

Algorit.	*L*	*361 pts. =1epoch e(rms)*	*3610 pts. =10 ep. e(rms)*	*18050 pts. =50 ep. e(rms)*	*36100 pt. =100 ep. e(rms)*	*Exec.time / single point t(msec)*
ALC	-	82.54 %	13.61%	0.323%	0.165%	3.30
ALR	-	7.50 %	1.23%	0.207%	0.110%	3.18
AVC	-	3.53 %	1.10%	0.669%	0.566%	7.90
AVR	-	3.62 %	1.22%	0.685%	0.592%	7.91
GRC	2	17.76 %	0.12 %	0.022 %	0.009 %	40.41
GRC	10	8.62 %	0.08 %	0.017 %	0.007 %	195.00
GRR	2	7.32 %	1.19 %	0.129 %	0.027 %	42.40
GRR	10	5.88 %	0.41 %	0.042 %	0.015 %	212.13
MAC	10	21.97 %	0.48 %	0.072 %	0.035 %	30.00
MAC	50	12.62 %	0.46 %	0.061 %	0.018 %	126.12
MAR	10	6.71 %	0.98 %	0.174 %	0.068 %	31.81
MAR	50	4.13 %	0.62 %	0.078 %	0.055 %	139.00
OSC	2	67.94 %	2.21 %	0.196 %	0.110 %	10.50
OSC	10	27.09 %	0.97 %	0.345 %	0.170 %	87.42
OSR	2	8.16 %	1.29 %	0.177 %	0.061 %	10.61
OSR	10	6.99 %	1.23 %	0.220 %	0.101 %	88.31
		361 epch	*3610 epoch*	*18050 epoch*		
BATC	-	1.66%	0.303%	0.069%	-	1019
BATR	-	2.23%	0.490%	0.077%	-	1023

Initial convergence in the Moving Average Algorithm is better than Albus Algorithm when Cyclic Training is used, but later on convergence rate gets slower and slower. The computational effort is the second best after Albus Nonbatch Algorithm.

The Optimized Step Algorithm with Cyclic Training shows a slow convergence because the transformed input patterns are parallel to each other. Since the weight vectors are optimized with respect to last L pieces of data, inputs are still correlated. With Random Training the initial convergence improves but long term convergence still remains the same. The best result is recorded with small L value. Large values may slow down the algorithm.

Gram-Schmidt procedure gives the fastest convergence, but it takes almost a quarter of a second just to store one single point. Larger L values, gives more accuracy and faster convergence but also requires tremendous amount of computation time.

The Maximum Error Algorithm has a good convergence property. Initial learning rate is especially high and could be made higher using Neighborhood Training technique. The long-term

convergence is the best after Gram-Schmidt procedure. It has a moderate computation time compared to others. Choosing *L* larger increases the accuracy but requires much computation time.

Batch training doesn't look appropriate for our purposes. The requirement that all the information about the training data points in one pass should be known limits the usage of this technique.

The recommended algorithm in this paper is the Maximum Error Algorithm because of its moderate computation time, fast initial and long term convergence. The works of (Ellison, 1988) and also Parks and Militzer, (1992) also agrees with this result.

4. COMBINE ALGORITHM ; NEIGHBORHOOD TRAINED MAXIMUM ERROR ALGORITHM

The Maximum Error Algorithm has been the best algorithm in terms of its moderate computation time and good convergence properties. But can there be any improvement on Maximum Error Algorithm? From Tables 1-2 and discussions before, we conclude that the initial convergence may be improved by "Neighborhood training".

To avoid the interference resulting from the CMAC's generalization property, (Thompson and Sungyyu, 1995) suggested the Neighborhood Training method. Ling later used this form of training in a control system since it minimizes learning interference (Ling and Fischer, 1995). In this method, input training points that lie outside the previous input training point are selected so that no interference will occur between the training points. After applying the Albus Nonbatch algorithm or other algorithms (all gives the same result), the result is appreciable because of the speed advantage of this method. Data for each memory element are adjusted only one time (with gain = 1) during training. Only one pass using the neighborhood method finishes the procedure.

The total number of input points to be trained depends on the resolution of the bins and ρ. For sin (x) function with $0 \le x \le 360$, a resolution of 1 degree and ρ = 10, the total number of Neighborhood Training points is Ntrainp = 360/10 + 1 = 37. Training at about 10 % of input points gives an rms error of 0.3%. The training points for this example are chosen as 1, 11, 21,..., 351 with no interference with each other. For an arbitrary function (Thompson and Sungyyu, 1995) used in simulations the calculated rms error is found 3.2 % after training only 37 pts. This result can be compared with the cyclic training with 180 training points (Figure 2).

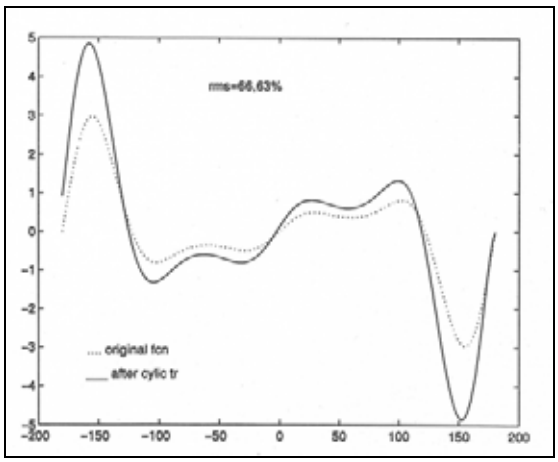

Figure 2. Neighborhood training at 37 points vs. Cyclic training at 180 points for function f(x)

In the proposed approach, the training is continued with the Maximum Error Algorithm for fine-tuning of CMAC. During simulations, input points for the Maximum Error Algorithm are chosen in a cyclic order. Instead of using only one pass of neighborhood at 37 points, a second pass of Neighborhood Training are also utilized in the midpoints. ie. for case b, mid-points correspond to the half of the previous neighborhoods, i.e., 6, 16, 26, ..., 356 respectively. In doing so, the points that had the least interference with the previously trained ones are used. After the second pass, the rms error for case a was 1.45 % and the rms for case b was only 0.1148 % (Figure 3). For both functions, L value is chosen to be 10.

From Table 3, it can be concluded that the initial convergence improves appreciably. ie, for only after one pass of Neighborhood Algorithm 1.18 % rms error is obtained vs. 21.24 % for Cyclic Trained Maximum Error Algorithm and 5.7 % for random trained Maximum Error Algorithm. After 10 epoch, rms error drops down to 0.23 % instead of 0.37 % and 1.15 % corresponding to the Cyclic and Random Training, respectively. The initial convergence rate was better for the second function. After first pass of Neighborhood Training, the rms error becomes 0.145 %. The rms error value drops down to 0.034 % after the first epoch of the Maximum Error algorithm. Long term convergence also well improves from 0.035 % to 0.017 %. In conclusion the combination of these two algorithms

improves CMAC neural network convergence considerably.

Table 3. Recommended Algorithm (L = 10)

Neighborhood Training Algorithm			Maximum Error Algorithm with Neighborhood Training (L=10)				
Case	1st pass e(rms)	2nd pass e(rms)	361 pts. =1 epoch e(rms)	3610 pts. =10 epoch e(rms)	18050 pts. =50 epoch e(rms)	36100 pts 100 epoch e(rms)	Mean Exec. time t (msec)
A	3.20%	1.45%	1.180%	0.230%	0.040%	0.016%	30
B	0.14%	0.11%	0.065%	0.034%	0.024%	0.017%	30

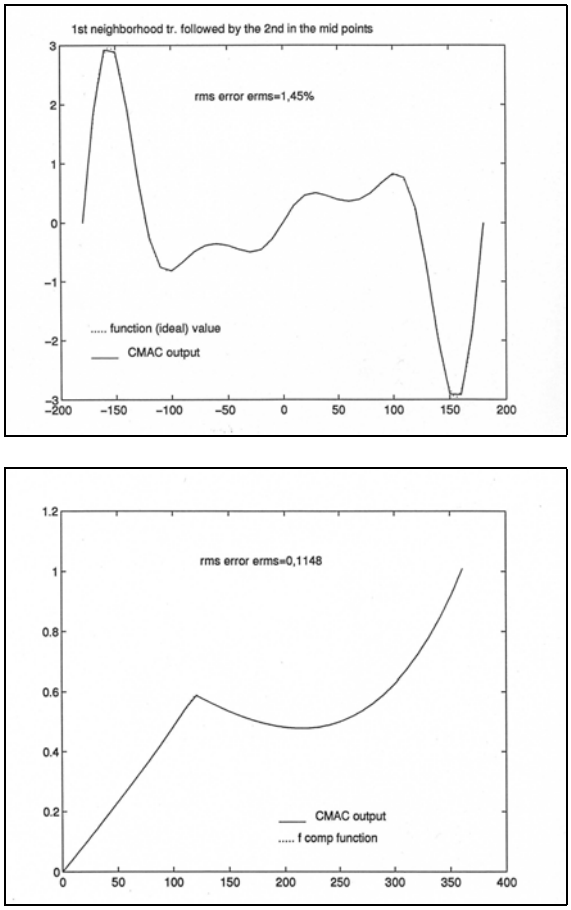

Figure 3. Function f(x) (top figure) and g(x) after 2-pass Neighborhood training

5. CONCLUSION

After explaining the CMAC with illustrative examples, the possible algorithms and training methods are analyzed thoroughly for the CMAC network. Throughout the paper the importance is stressed on the selection of the input training points. For that reason beside the Cyclic Training, Random Training is also taken into consideration in comparing different algorithms, and good results are obtained for Optimized Step Algorithm performance when Random Training is employed. Among the

algorithms that were in comparison, Maximum Error Algorithm has been recommended because of its moderate computation time, fast initial and long-term convergence. Improvements are then examined for the recommended Maximum Error algorithm and a combine algorithm approach has been developed. In this scheme, it is thought that prior to CMAC training using the recommended algorithm, a new form of training called Neighborhood training technique can be meaningful to apply if faster initial convergence is required.

The employment of the combine algorithm technique yielded faster initial convergence for both cases a and b, and also resulted in better long term convergence for case b. However, the acceleration in the convergence may depend on shape and complexity of a function as well as other network parameters.

In comparison to the other neural networks, CMAC, has the advantage of very fast learning (Horst, 1993). By using the combine approach, much faster initial convergence is achieved for the CMAC, which is very important for control applications.

6. REFERENCES

Albus, J. S. 1975a. "A New Approach to Manipulator Contol: The Cerebellar Model Articulation Controller (CMAC)". Transactions of the ASME, (97), 220-227

Albus, J. S. 1975b. "Data Storage in the Cerbeller Model Articulation Controller (CMAC)", Transactions of the ASME, (97), 228-233.

Brown, M. and Harris, C. J. 1992. “Least Mean Square Learning in Associative Memory Networks", Proc. IEEE Int. Symposium on Intelligent Control, Glasgow, UK

Brown, M. and Harris, C. J. 1994. Neurofuzzy Adaptive Modelling and Control, Prentice Hall, Hertfordshire, UK.

Cembrano, G., Wells, G., Sarda, J., Ruggeri, A. 1999. "Dynamic Control of a Robot Arm Using CMAC Neural Networks", Control Engineering Practice, 5 (4), 485-492.

Ellison, D. 1988. “On the Convergence of the Albus perceptron”, IMA J. Math. Cont. and Information. (5), 315-331.

Horst, W. P. 1993. A Comparison of Five Neural Networks MS Thesis, Department of EE, Pennsylvania State University, University Park, PA.

Ling, Rong-Ho., Fischer, G. W. 1995. "An Online Arc Welding and Quality Monitor and Process Control System" Int. IEEE/IAS Conf. on Industrial Automation and Control, pp. 22-29.

Miller, W. T., Filson, H. G. and Craft, L. G. 1982. “Application of a General Learning Algorithm to the Control of Robotic Manipulators”: The International Journal of Robotics Research, 6 (2), 123-147.

Parks, P. C. and Militzer, J. 1992. “A Comparison of Five Algorithms for the Training of CMAC Memories for Learning Control Systems, Automatica 28 (5), 1027-1035.

Rosenblatt, F. 1961. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Washington, DC : Spartan Books.

Thompson, D. E. and Sungyyu, K. 1995. “Neighborhood and Random training Techniques For CMAC” IEEE Transactions on Neural Networks, 6 (1), 196-202.