Using CMAC for mobile robot motion control

Kristóf Gáti, Gábor Horváth

Budapest University of Technology and Economics,
Department of Measurement and Information Systems
Magyar tudósok krt. 2. Budapest, Hungary H-1117.
http://www.mit.bme.hu
{gatikr, horvath}@mit.bme.hu

Abstract. Cerebellar Model Articulation Controller (CMAC) has some attractive features: fast learning capability and the possibility of efficient digital hardware implementation. These features makes it a good choice for different control applications, like the one presented in this paper. The problem is to navigate a mobile robot (e.g a car) from an initial state to a fixed goal state. The approach applied is backpropagation through time (BPTT). Besides the attractive features of CMAC it has a serious drawback: its memory complexity may be very large. To reduce memory requirement different variants of CMACs were developed. In this paper several variants are used for solving the navigation problem to see if using a network with reduced memory size can solve the problem efficiently. Only those solutions are described in detail that solve the problem in an acceptable level. All of these variants of the CMAC require higher-order basis functions, as for BPTT continuous input-output mapping of the applied neural network is required.

Keywords: CMAC, recurrent neural network, control, BPTT

1 Introduction

The Cerebellar Model Articulation Controller is a special neural network architecture originally proposed by James S. Albus [**?**]. The network has some attractive features like fast convergence, local approximation capability and the possibility of efficient digital hardware implementation. Because of these features the CMAC is often used in control applications [**?**] among other areas like image and signal processing, pattern recognition and modeling. This paper deals with a navigation problem. It presents a solution for mobil robot motion control, implemented with a CMAC network. This is a highly nonlinear problem, which is hard to solve with classical control methods. There are many articles about this problem e. g. [**?**]. The question is if the advantageous properties of CMAC can be utilized in this complex navigation problem. To answer this question it should also be noted that despite the attractive features CMAC has some drawbacks. The most serious one is that its memory complexity may be huge, and that concerning its function approximation capability it may be inferior to an MLP.

Many solutions were suggested on both problems. Hash-coding [?],[?],[?] kernel CMAC [?],[?], fuzzy CMAC [?] and SOP-CMAC [?] are some ways for reducing memory complexity. Weight-smoothing [?] and higher-order CMACs [?],[?] are proposed for improving function approximation capability.

The paper is organized as follows. In Section 2 the basic principle of BPTT is summarized, in Section 3 the classical CMAC is presented, in Section 4 the extensions and variants of the CMAC are presented, while in Section 5 the partial derivatives are determined. Section 6 describes the system and the training in details. The results may be found in Section 7, and conclusions are drawn in Section 8.

2 Backpropagation through time (BPTT)

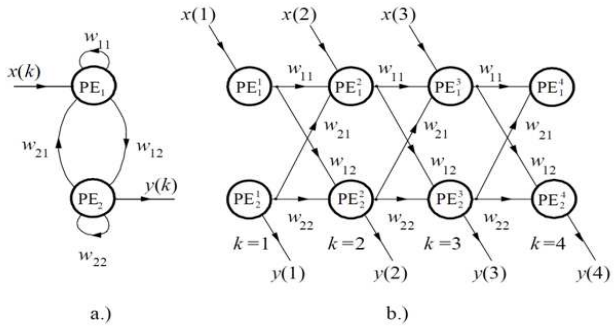


Fig. 1: Simple network with feedback **a.)** and its unfolded equivalent **b.)**

BPTT is an approach proposed for training recurrent neural networks [?]. As a recurrent net is a dynamic network, it needs special training algorithms where the temporal behaviour of the network must be taken into consideration. The basic idea of BPTT is that the recurrent network is unfolded in time - as it can be seen in Fig. **??**.[?] - resulting in a many-stage static one, where this static network basically can be trained using classical backpropagation algorithm. As the operation of the recurrent network is considered in discrete time steps, the number of stages equals to the time steps required for the operation of the network i.e. for determining the number of stages of the static network the time window of the operation of the recurrent network must be fixed. One constraint must be noticed. As the number of weights in the unfolded static network is increased, where in the static network more weights are used instead of a single weight of the original network, these weights must be modified simultaneously and with the same amount, as they represent the same physical weight in different time steps.

3 Classical CMAC

CMAC is a basis function network where finite-support basis functions are used. The basis functions are applied in the input space in predefined positions and the

supports of the basis functions are fixed-size closed intervals - or in multidimensional cases - fixed-size hypercubes.The classical CMAC applies rectangular basis functions that take constant value over the hypercube and zero elsewhere.The hypercube is often called the receptive field of the basis function. The network has two layers. The first layer performs a fixed nonlinear mapping, which implements the basis functions. The network output is calculated in the second layer as a weighted sum of the basis function outputs. Only the weights are trainable in the network.

The fixed first layer creates a binary vector called association vector, which consists of the outputs of the basis functions. If the input point, $\mathbf{x} \in \Re^N$, is in the receptive field of a basis function then the corresponding element in the association vector will be 1, otherwise it will be 0. The width of the receptive field is finite, controlled by the generalization parameter of the CMAC. This is denoted by C.

The basis functions are arranged in overlays. An overlay is a set of basis functions, which covers the full input space, without any gap and overlap. Hence the number of overlays equals to the number of the activated basis functions. In case of an N dimensional problem this is C^N.

The number of required basis function is $(R + C - 1)^N$, here R means the size of the input space. This number can be enormous in a real world application, for example if $R = 1024$ and $N = 10$, then the required number of basis functions is $\sim 2^{100}$ which cannot be implemented. To reduce the number of basis functions Albus proposed a way of using only C overlays, however even with this reduction the network could need extremely large weight-memory that is rather hard or even impossible to implement [**?**].

The second layer of the CMAC calculates the output $y \in R$ of the network as a scalar product of the association vector $\mathbf{a}$ and the weight vector $\mathbf{w}$:

$$y(\mathbf{x}) = \mathbf{a}(\mathbf{x})^T \mathbf{w} = \sum_{i:a_i=1} w_i \tag{1}$$

Because of the binary basis functions the product can be replaced by the sum of weights corresponding to the activated basis functions.

The weights can be trained using the LMS rule in Eq. **??**.

$$\Delta w_i = \mu(y_d - y), i : a_i = 1 \tag{2}$$

where y_d is the desired output of a training data point, and μ is the learning rate.

4 Variants of the CMAC

4.1 Higher-order CMAC

For BPTT training the binary (rectangular) basis functions are not adequate, because BPTT training needs the derivative of the basis functions. Lane et al.

proposed the CMAC with B-Spline basis functions in [?]. The B-Splines are especially well suited for the CMAC with finite-support basis functions as the B-Splines are non-zero only in a finite and closed interval.

Further advantages are the improved performance and the possibilty of training continuous functions. The main disadvantage is the loss of multiplication-free structure, as the association vector is not binary anymore. There are other type of basis functions for example Gaussian, see [?].

4.2 Kernel CMAC

CMAC can be interpreted as a kernel machine [?], where instead of using directly the basis functions, we use the so called kernel functions that are constructed easily from the basis functions. In a Kernel CMAC(KCMAC) the memory complexity is upper bounded by the number of training points independently of the dimension of the input space and the number of basis functions [?],[?].

If M is the number of basis functions, and P is the number of training samples, then the input-output mapping of a basis function network based on the basis-function representation is:

$$y(\mathbf{x}) = \sum_{j=1}^{M} w_j \varphi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) \tag{3}$$

where $\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), ..., \varphi_M(\mathbf{x})]^T$ is a vector formed from the outputs of the basis functions for the input $\mathbf{x}$ and $\mathbf{w}$ is the weight vector. The same mapping can be described using the kernel function representation as:

$$y\left(\mathbf{x}\right) = \sum_{k=1}^{P} \alpha_k K\left(\mathbf{x}, \mathbf{x}\left(k\right)\right) \tag{4}$$

It is also a weighted sum of nonlinear functions where the $K(\mathbf{x}, \mathbf{x}(k)) = \boldsymbol{\varphi}^T(\mathbf{x})\boldsymbol{\varphi}(\mathbf{x}(k)), k = 1, ..., P$ functions are called kernel functions defined by scalar products, and the α_k coefficients serve as the weight values.

In kernel CMAC the kernel functions are defined as

$$K(\mathbf{x}, \mathbf{x}(k)) = \mathbf{a}^T(\mathbf{x}) \cdot \mathbf{a}(\mathbf{x}(k)) \tag{5}$$

where $\mathbf{a}(\mathbf{x})$ is the association vector for the discrete input $\mathbf{x}$ and the response of a CMAC for a given $\mathbf{x}$ can be written as:

$$y(\mathbf{x}) = \mathbf{a}^T(\mathbf{x})\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{y}_d = \mathbf{a}^T(\mathbf{x})\mathbf{A}^T\boldsymbol{\alpha} = k^T(\mathbf{x})\boldsymbol{\alpha} \tag{6}$$

where

$$\boldsymbol{\alpha} = (\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{y}_d \tag{7}$$

Here $\mathbf{A}$ is a $P \times M$ matrix constructed from the association vectors of the training points, $\mathbf{A} = [\mathbf{a}_1(\mathbf{x}), ..., \mathbf{a}_P(\mathbf{x})]^T$.

In kernel representation the components of $\boldsymbol{\alpha}$ are considered as the weight values, so instead of using M weights, here only P weights will be used. As in multidimensional cases $P << M$ this is a great reduction of memory complexity.

4.3 Fuzzy CMAC

Fuzzy CMAC (FCMAC) is a very similar extension of the CMAC as the KCMAC, nevertheless the mathematical foundation of the two versions is completely different. From the view of FCMAC the creation of the association vector can be considered as the calculation of the strength of a fuzzy rule's IF part, and the calculation of the output can be interpreted as a COG (Center of Gravity) defuzzyfication method.

In FCMAC the first difference from KCMAC, is the selection of the basis function centers and the number of the basis functions [**?**]. These methods have a great diversity [**?**]. The placement of the basis functions is the formulation of the rule's condition part.

The second difference of the FCMAC from the KCMAC is the calculation of the output. While KCMAC uses a weighted sum see Eq. **??**, FCMAC calculates a weighted average of the basis functions:

$$y = \sum_{j=1}^{K} w_j \cdot \phi_j(\mathbf{x}) \Bigg/ \sum_{j=1}^{K} \phi_j(\mathbf{x}) \tag{8}$$

where $\varphi_j(\mathbf{x})$ is the value of the j-th basis function at the input point $\mathbf{x}$, which may be interpreted as the firing strength of the j-th rule. The j-th weight is the value of the j-th rule's consequense part.

5 Calculating the partial derivatives

Because of the BPTT training the derivatives of the network output against its input are required. Basically this means the following:

$$\frac{\partial y}{\partial x_i} = \frac{\partial(\mathbf{w}^T \mathbf{a}(\mathbf{x}))}{\partial x_i} = \mathbf{w}^T \cdot \frac{\partial \mathbf{a}(\mathbf{x})}{\partial x_i} \tag{9}$$

That means that for obatining the derivative of the output the derivative of the basis functions must be calculated. For this reason we must use higher-order CMACs, because the derivative of the binary basis function does not exist. This derivative may be calculated by the equation in Eq. **??** [**?**].

$$B_i^n(t)' = \frac{n}{t_{i+n} - t_i} \cdot B_i^{n-1}(t) - \frac{n}{t_{i+n+1} - t_{i+1}} \cdot B_{i+1}^{n-1}(t) \tag{10}$$

where B_i^n is the i-th B-Spline of order n.

If Fuzzy CMAC is used the calculation of the derivatives is different because of the different output calculation scheme, see Eq. **??**.

$$\frac{\partial y}{\partial x_k} = \frac{\sum_{i=1}^{K} w_i \varphi_i^{'}(\mathbf{x}) \cdot \sum_{j=1}^{K} \varphi_j(\mathbf{x}) - \sum_{i=1}^{K} w_i \varphi_i(\mathbf{x}) \cdot \sum_{j=1}^{K} \varphi_j^{'}(\mathbf{x})}{\left(\sum_{j=1}^{K} \varphi_j(\mathbf{x}) \right)^2} \tag{11}$$

6 The experimental system

The setup of the system can be seen in Fig. **??**. The CMAC used to control the mobile robot gets the position of the robot (x, y), and the angle between its direction and the x axis (θ). In fact, the network uses the sine and cosine of the angle, because the angle itself would cause a discontinuous function, which cannot be learned correctly by neural networks.

The model of the robot is given by Eq. **??**, where the steering wheels angle is generated by the CMAC.

$$\begin{aligned} &\|R\| = \mathrm{sinc}(\gamma_i/2) \cdot v \cdot t, \arg(R) = \theta_i + \gamma_i/2 \\ &x_{i+1} = x_i + \|R\| \cdot \cos(\arg(R)), y_{i+1} = y_i + \|R\| \cdot \sin(\arg(R)) \\ &\theta_{i+1} = \theta_i + \gamma_i \end{aligned} \tag{12}$$

where R is the shift vector, γ_i is the angle of the steering wheel, v is the speed of the car. t is the time of delay between modifications of the angle of the steering wheel.

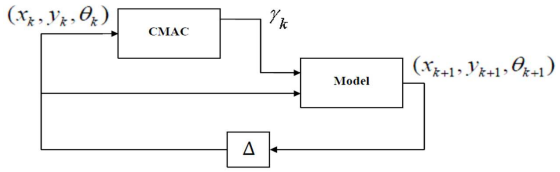


Fig. 2: The set-up of the system

The generation of a given path runs until the error between the desired state and the actual state decrease. It must have been handled if the destination were behind the initial state, because in this situation the first step has increased the error, so without further instructions these routes would not be learned, thus a minimal number of steps was added to the network as constraint for these cases.

If training was done starting from many different initial states, then because of the local approximation capability of CMAC needlessly long paths could arised. For this reason another CMAC was added to the system, which controlled the maximum number of steps to be taken by the robot. See Section 7.1 for the generation of the training set.

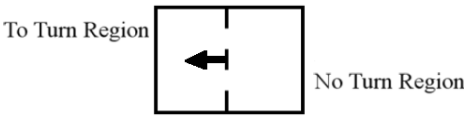


Fig. 3: The division of the input space into regions

The input space was divided into two regions as can be seen on Fig. **??**. The arrow specifies the destination state for the mobile robot. The creation of the No Turn Region was necessary because the paths started from this region could not be learned by the CMAC. The goal for paths, starting from this region, was to reach the dashed line, see Fig. **??**.

After the robot has stopped the weights must be updated during training. The training used backpropagation through time (BPTT).

The weight updating equations for the CMAC in the i-th step are in Eq. **??**.

$$\Delta \mathbf{w}_i = \boldsymbol{\epsilon}_{i+1}^T \mathbf{v}_i \cdot \mathbf{a}(x_i, y_i, \theta_i) \tag{13}$$

Here $\boldsymbol{\epsilon}_i = \left(\frac{\partial \epsilon_x}{\partial x_i} \; \frac{\partial \epsilon_y}{\partial y_i} \; \frac{\partial \epsilon_\theta}{\partial \theta_i} \right)^T$ is the derivative of the error with respect to the states, $\mathbf{v}_i = \left(\frac{\partial x_{i+1}}{\partial \gamma_i} \; \frac{\partial y_{i+1}}{\partial \gamma_i} \; \frac{\partial \theta_{i+1}}{\partial \gamma_i} \right)^T$ is the derivative of the states with respect to the output of the CMAC, the angle of the steering wheel.

After the value of the weight modification is calculated the error must be backpropagated. This is done by Eq. **??**.

$$\boldsymbol{\epsilon}_i = \mathbf{S}_i \boldsymbol{\epsilon}_{i+1} + \left(\boldsymbol{\epsilon}_{i+1}^T \mathbf{v}_i \right) \cdot \mathbf{q}_i \tag{14}$$

Here the matrix of $\mathbf{S_i}$ is the derivative of the states with respect to the previous states, see Eq. **??**. And $\mathbf{q}_i = \left(\frac{\partial \gamma_i}{\partial x_i} \; \frac{\partial \gamma_i}{\partial y_i} \; \frac{\partial \gamma_i}{\partial \theta_i} \right)^T$ is the derivatives of the network with respect to the actual state.

$$\mathbf{S}_i = \begin{pmatrix} \frac{\partial x_{i+1}}{\partial x_i} & \frac{\partial y_{i+1}}{\partial x_i} & \frac{\partial \theta_{i+1}}{\partial x_i} \\ \frac{\partial x_{i+1}}{\partial y_i} & \frac{\partial y_{i+1}}{\partial y_i} & \frac{\partial \theta_{i+1}}{\partial y_i} \\ \frac{\partial x_{i+1}}{\partial \theta_i} & \frac{\partial y_{i+1}}{\partial \theta_i} & \frac{\partial \theta_{i+1}}{\partial \theta_i} \end{pmatrix} \tag{15}$$

Weight updating is done only if the error is backpropagated. The weights are updated using Eq. **??** and Eq. **??**.

$$\Delta \mathbf{w} = \mu \cdot \sum_{i=1}^{sn} \Delta \mathbf{w}_i \tag{16}$$

Here sn means the number of steps taken by the car, which equals to how many times the CMAC set the angle of the steering wheel.

An example for the error during the training may be seen in Fig. **??**. The dashed line, the dotted line, the dashed-dotted line and the solid line are the errors for the x, the y, the θ axis and their aggregated, respectively. On the right side of the figure the gray colored arrows show the direction of the weight modification and the black arrows show the path of the car. The destination state is at the (100,100) point, and the desired angle is π.

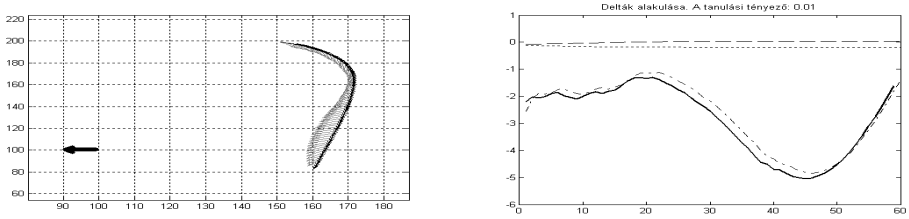


Fig. 4: The path of the robot during training and the errors which was calculated during the backpropagation.

7 Experimental Results

The size of the input space was 300 for all dimensions, where the input components were $x, y, \cos(\theta), \sin(\theta)$.

7.1 Training of the maximum step number

In this case a training sample was an initial state as input and the number of steps of the corresponding trained path as desired output. Every initial state was trained until the distance between the destination and the end state got smaller then 0.3 and the error of the angle got smaller than 5°. Due to the separation of the input space, see Fig. **??**, two CMACs must have been constructed and the training samples corresponding to the different regions were trained separately.

An example from the No Turn Region is drawn in Fig. **??**, where the x, y is plotted, and the angle is $3\pi/4$.

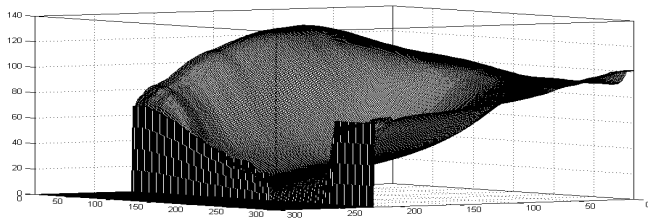


Fig. 5: The maximum number of steps if the angle is fixed.

An FCMAC was trained with $C = 70$ and the center selection method was used from [**?**]. This method selects a training point as new center if the maximum activation value of the existing centers are smaller than a prespecfied threshold. Higher limit causes more basis functions but better performance. This limit was set to 90% of the maximum value of the basis functions, which is quite strong.

3000 training samples were generated for each region, and as it was predictable almost all training samples were selected as basis function center, exactly 2818 were selected by the FCMAC for the No Turn Region.

7.2 Training the driving using a KCMAC

For the driving the input space was not separated as shown in Fig. **??**. The position of the initial states was generated with polar coordinates with the radius $r \in [0, 150]$ and angle $\vartheta \in [0, 2\pi]$. The angle of the initial states was generated with $\theta \in [0, 2\pi]$. 550 initial states were generated. An actual state of the robot was a training point candidate. The training points from these candidates were selected similarly as in the previous section, the limit was set to 60% in this case.

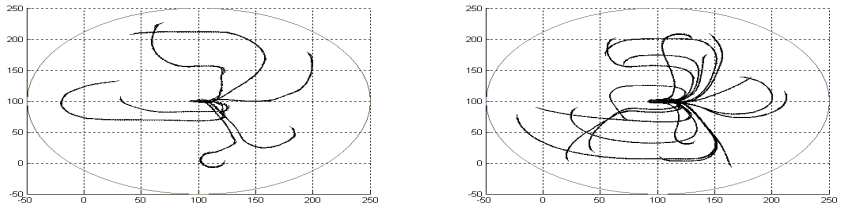
60 epochs were trained with the network, and in every epoch from all initial states the actual path were generated and the error was backpropagated. The learning rate was set for every backpropagation based on the error between the difference of the destination state and the end state of the path. The resulting 3 element vector is compared with the vectors in **??**. If all components are smaller than in Table **??**, the learning rate may be set for the network. The smallest must be selected from these options. For example if the error is $(5.2, 2.7, \pi/30)$ than KCMAC will use 10^{-3} and FCMAC will use 10^{-4} as learnig rates respectively.

The size of the input space was 300 and the generalization parameter was set to $C = 40$ for all dimensions. The kernel functions was set to 6-th order B-Splines. After the training 46780 training points were selected as kernel function center.

This number could be decreased by increasing the generalization parameter or decreasing the limit of the training point selection. Some sample paths may be seen on Fig. **??**.

Table 1: The learning rates corresponding to the error of the last state of a given path

Value of errors	KCMAC	FCMAC
Default	10^{-2}	10^{-2}
$\left(30\ 30\ \pi/4\right)$	10^{-2}	10^{-3}
$\left(12\ 12\ \pi/18\right)$	10^{-3}	10^{-4}
$\left(3\ 3\ \pi/45\right)$	10^{-4}	10^{-5}
$\left(0.6\ 0.6\ \pi/180\right)$	10^{-6}	10^{-7}



(a) KCMAC (b) FCMAC

Fig. 6: Path generated from different initial states controlled by KCMAC and FCMAC

7.3 Training the driving using a FCMAC

When the FCMAC was used the general setting was $C = 70$ for all dimensions, the learning rates are in Table. **??**. The size of the input space, and the basis functions were the same as in the KCMAC. The robot was trained from 550 initial states for 50 epochs. Sample paths may be seen on Fig. **??**. 35672 training points were selected as basis function centers.

The comparison of the methods used in this paper is presented in Table **??**.

Table 2: The comparison of the different CMACs

Type of CMAC	KCMAC	FCMAC
Complexity(Number of weights)	46780	35672
Number of epochs	60	50
Initial states	550	550
Value of C	40	70
Type of basis- kernel functions	6th order B-Spline	6th order B-Spline

8 Conclusions

In this paper different types of CMACs were used for robot motion control. Mainly two types were investigated, the kernel and fuzzy CMAC. Solutions for CMAC with hash-coding and SOP-CMAC was not adequate. Perhaps the parameters were not tuned correctly in the latter cases.

However in the case of FCMAC and KCMAC the results was quite satisfying, still improving the solutions is possible. The number of initial states, the value

of the learning rate may have been chosen better, but the overall performance could not be improved too much.

The robot motion control is a highly nonlinear control problem, which makes it suitable for testing neural networks. The main advantages of the CMAC, the fast convergence and local approximation, were exploited, becuase only 50 epochs were needed for the training which is very fast compared to an MLP where thousands of epochs were required [**?**]. The disadvanteges of the network, like the huge memory requirement, made impossible to implement the task with a classical CMAC.

This work is connected to the scientific program of the "Development of quality-oriented and cooperative R+D+I strategy and functional model at BME" project. This project is supported by the New Hungary Development Plan(Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002).

References

1. Albus, J.S. (1975) "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)", Transaction of the ASME, Sep. 1975. pp. 220-227.
2. L. G. Kraft, W. T. Miller, and D. Dietz, "Development and application of CMAC neural network-based control," in Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches, D. A.White and D. A. Sofge, Eds. New York: Van Nostrand, 1992, pp. 215-232.
3. Z.-Q. Wang, J. L. Schiano and M. Ginsberg: "Hash Coding in CMAC Neural Networks" Proc. of the IEEE International Conference on Neural Networks, Washington, USA, Vol. 3, pp. 1698-1703, 1996
4. G. Horváth and T. Szabó, Kernel CMAC with improved capability, IEEE Trans. Sys. Man Cybernet. B 37 (1): 124-138, 2007.
5. G. Horváth, and K. Gáti, Kernel CMAC with Reduced Memory Complexity, In Proceedings of the 19th international Conference on Artificial Neural Networks, Limassol Vol. I, pp. 698-707, 2009
6. Nie, J. and Linkens, D. A. 1994. FCMAC: a fuzzified cerebellar model articulation controller with self-organizing capacity. Automatica 30, 4 (Apr. 1994), 655-664.
7. K. Mohajeri, M. Zakizadeh, B. Moaveni, M. Teshnehlab "Fuzzy CMAC Structures" Proc. IEEE 2009 Int. Conf. on Fuzzy Systems.
8. C. S. Lin and C. K. Li, "A low-dimensional-CMAC-based neural network," Proceedings of IEEE Conference on Systems, Man and Cybernetics, vol. 2, pp. 1297-1302, 1996.
9. S. H. Lane, D. A. Handelman, and J. J. Gelfand, "Theory and development of higher-order CMAC neural networks," IEEE Contr. Syst. Mag., pp. 23-30, Apr. 1992.
10. C.-T. Chiang and C.-S. Lin, "CMAC with general basis functions," Neural Networks, vol. 9, no. 7, pp. 1199-1211, 1996.
11. D. H. Nguyen and B. Widrow. Neural networks for self-learning control systems. IEEE Control Systems Magazine, pages 18-23, April 1990.
12. D. E. Rumelhart, G. E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in Parallel Distributed Processing: Explorations in the Microstructure of Cognition (D. E. Rumelhart and J. L. McClelland, eds.), Vol. 1, Chapter 8, Cambridge, MA, MIT Press (1986).