*Int. J. Appl. Math. and Comp. Sci., 1999, Vol.9, No.3, 571–598*

# CMAC AND ITS EXTENSIONS FOR EFFICIENT SYSTEM MODELLING

TAMÁS SZABÓ*, GÁBOR HORVÁTH*

This paper deals with the family of CMAC neural networks. The most important properties of this family are the extremely fast learning capability and a special architecture that makes effective digital hardware implementation possible. The paper gives an overview of the classical binary CMAC, shows the limitations of its modelling capability, gives a critical survey of its different extensions and suggests two further modifications. The aim of these modifications is to improve the modelling capability while maintaining the possibility of an effective realization. The basic element of the first suggested hardware structure is a new matrix-vector multiplier which is based on a canonical signed digit (CSD) number representation and a distributed arithmetic. In the other version, a hierarchical network structure and a special sequential training method are proposed which can constitute a trade-off between the approximation error and generalization. The proposed versions (among them a dynamic extension of the originally static CMAC) are suitable for embedded applications where the low cost and relatively high speed operation are the most important requirements.

**Keywords:** CMAC, neural networks, B-splines, hardware implementation.

## 1. Introduction

Neural networks have an important role in both modelling non-linear static or dynamic systems and diagnosis. Static systems can be modelled by feedforward static networks like MLP, RBF, etc. For dynamic system modelling networks can be used that reveal a temporal behaviour. This behaviour can be obtained by various methods, e.g. by using a local storage at the neuron level where every neuron performs a dynamic operation, or by adding global memories to the static networks. These extended networks are usually based on static feedforward networkstructures too. This means that in most cases the main features of a network, like its modelling capability, learning speed, special problems of implementations, etc. depend on the features of the static network in both the static and dynamic cases.

Among static networks, MLP's (Multi-Layer Perceptrons) are the most popular ones. Their universal approximation capability is proved, there are many training algorithms, and based on static MLP's various dynamic feedforward and feedback structures have been developed.

* Department of Measurement and Information Systems, Technical University of Budapest, Műegyetem rkp. 9, Budapest, Hungary, H–1521, e-mail: {szabo,horvath}@mit.bme.hu.

The most serious drawback of MLP's and MLP-based dynamic networks is their rather slow training procedure, which may be an obstacle to apply them to real-time adaptive modelling problems. To increase the learning speed, modified training algorithms have been developed and applied, e.g. instead of the standard backpropagation method, more sophisticated algorithms such as Levenberg-Marquardt, RLS and others can be applied. However, there is a further way to increase the learning speed: to use networks that have only a single trainable layer. There are some networks with a single trainable layer that have the same or approximately the same capabilities as multi-layer perceptrons. CMAC's (Cerebellar Model Articulation Controller) and RBF's (Radial Basis Function Network) are among such networks (Albus, 1975; Broomhead and Lowe, 1988). CMAC networks have further advantages. It is easy to implement such a network: as we will see later, it does not need multipliers at all. Similarly to the MLP, the CMAC is a static network. However, it can also be extended to form dynamic behaviours.

This paper deals with the CMAC network family. It presents the basic structure of this network, summarises the most important advantageous features and the drawbacks of the classical CMAC. It gives a critical review of various versions that have been developed and published in the last twenty years, and suggests some further modifications that have advantageous properties. The aim of these modifications is to improve the modelling capability of the network, or at least to form structures where some bounds can be determined for the modelling error and at the same time to maintain its fast training and the possibility of easy implementation. So special emphasis will be put on the question of hardware realizations, i.e. all versions will be ranked according to the ease of digital low-cost implementation. Although the proposed versions require multiplications, a new efficient multiplier structure is also proposed, so both the classical multiplierless structure and the new versions are suitable for low-cost embedded systems.

The paper is organised as follows. The classical CMAC and its main features will be reviewed in Section 2. Various extensions and modifications will be overviewed in Section 3. Section 4 details the suggested new modification and its properties. In addition, at the end of this section some dynamic extensions will be briefly presented, where most of the advantageous features of the static network can be maintained, while Section 5 deals with special questions of hardware realizations.

## 2. Classical CMAC and Its Main Features

The idea of the CMAC (Cerebellar Model Articulation Controller) neural network goes back to the mid 1970s (Albus, 1975). The CMAC has many excellent properties which make it a real alternative to backpropagated multilayer networks (MLP's). Perhaps the most important feature is its exceptional learning characteristic: the speed of learning can be much higher than that of a corresponding MLP using the backpropagation training algorithm. A further important feature is that the classical binary CMAC can be implemented without using multipliers, so it is especially suited to digital hardware implementation, an ideal network architecture for embedded applications where a relatively high operating speed, a small size and a small implementation cost are the most important requirements.

The binary CMAC network can be considered as an associative memory which performs two subsequent mappings. The first one (which is a non-linear mapping) projects an input space point ($\boldsymbol{u}$) into a binary association vector ($\boldsymbol{a}$). The association vectors always have $C$ active elements, which means that $C$ bits of an association vector are ones and the other are zeros. $C$ is an important parameter of the CMAC network and it is much less than the length of the association vector. The CMAC uses a quantized input, so the number of possible different input data is finite. There is a one-to-one mapping between the discrete input data and the association vectors, i.e. each possible input point has a unique association vector representation.

The first mapping should have the following characteristics:

- It should map two neighbouring input points into association vectors in which only a few elements, i.e. few bits, are different.
- As the distance between two input points increases, the number of common active bits in the corresponding association vectors decreases. The input points far enough from each other (further than the neighbourhood determined by the parameter $C$) should not have any common bits.
- This mapping is responsible for the non-linear property of the whole system.

The second mapping calculates the output of the network as a dot product of the association vector ($\boldsymbol{a}$) and a weight vector ($\boldsymbol{w}$):

$$y = \boldsymbol{a}^T(\boldsymbol{u})\boldsymbol{w} \tag{1}$$

As the association vector is a sparse binary vector with $C$ active elements, every active element will select a weight from the weight vector and the scalar product means nothing but the sum of $C$ selected weights.

In practical applications the two mappings are implemented by a two-layer network. The first layer is responsible for mapping the input points to the association vectors; this mapping is fixed (can be wired in hardware). The second layer is a linear combiner, where the weights can be modified during training. This is the only trainable layer of the CMAC networks; the weights can be updated using a simple LMS rule.

To implement the first mapping, the input variables are divided into overlapping regions where every region is subdivided into quantization intervals. This quantization determines the resolution of the network and the shift positions of the overlapping regions. A given value of an input variable activates all the regions where the input is within a quantization interval of a region. To ensure $C$ active bits in the association vectors, $C$ overlapping regions have to be selected by every quantization interval. Figure 1 shows some overlapping regions in a two-dimensional case when $C = 4$. If the distance between two input points is no more then $C$ quantization intervals, then there are surely common bits in their association vectors. As in this case a local generalization takes place, the parameter $C$ is often called the generalization parameter. This local generalization means that the training of a point does not have any or has only a weak effect on training a different point if these points are far enough from each other. A consequence of this property is the incremental learning capability, the feature that cannot be found in backpropagation nets. Global generalization does not exist in this network.

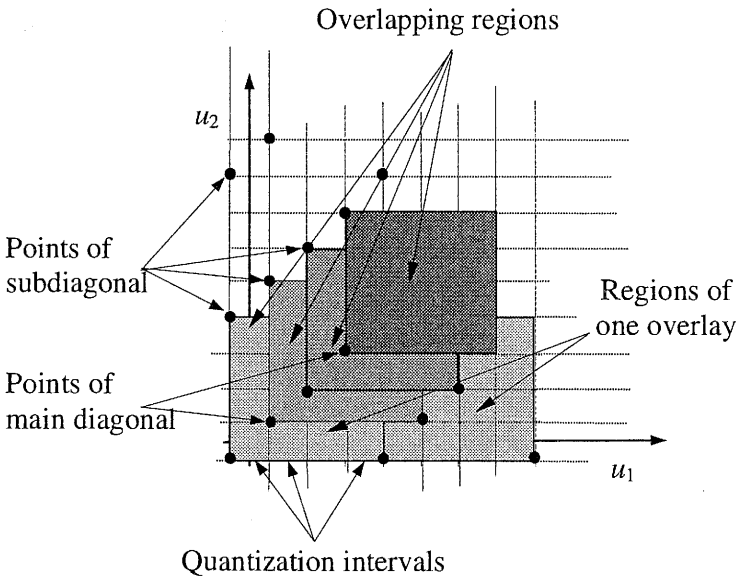


Fig. 1. The overlapping regions of the CMAC.

The operation of the CMAC can be regarded in a different way as well. The output value for an input point can be considered as a weighted sum of selected basis functions. Every element in the association vector corresponds to a basis function. Each basis function has the so-called receptive field. The shaded regions in Fig. 1 are receptive fields of different basis functions. If an input point is within a receptive field, the corresponding basis function is selected.

The basis functions can be grouped into overlays. One overlay contains basis functions with non-overlapping supports, but the union of the supports covers the whole input space. The different overlays have the same structure; they consist of similar basis functions in shifted positions. The positions of the overlays and the basis functions of one overlay can be represented by definite points. In the original Albus scheme the overlay-representing points are on the main diagonal of the input space, while the basis function positions are represented by the subdiagonal points as is shown in Fig. 1 (black dots). The overlay representing points can be described as displacement vectors whose elements are the co-ordinates of the definite points. Every input datum will select $C$ basis functions, each of them on a different overlay, so in an overlay one and only one basis function will be active for every input point.

As every selected basis function will be multiplied by a weight value, the size of the weight memory is equal to the total number of basis functions or to the length of the association vector. As an element of the association vector can be considered as the value of a basis function for a given input, the output of the binary basis function is one if an input is in its receptive field and zero elsewhere:

$$a_i(\boldsymbol{u}) = \begin{cases} 1 & \text{if } \boldsymbol{u} \text{ is in the receptive field} \\ & \text{of the } i\text{-th basis function} \\ 0 & \text{otherwise} \end{cases}$$

The application of such functions means that if the basis functions are selected, the sum of the corresponding weights will form the network's output value independently of the exact position of the input point in the receptive fields of the basis functions:

$$y(\boldsymbol{u}) = \sum_{i:\, a_i(\boldsymbol{u})=1} w_i \tag{2}$$

The output of the network can be obtained without the need for any multiplication. It is also easy to train the network. As the output layer is the only trainable layer of the CMAC and as this output layer performs a linear operation, the simple LMS algorithm can be applied:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + 2\mu\varepsilon(k)\boldsymbol{a}(k) \tag{3}$$

Here $k$ is the discrete-time index, $\varepsilon(k)$ is the error of the network and $\boldsymbol{a}(k)$ is the association vector at step $k$. As $\boldsymbol{a}(k)$ is a sparse binary vector, only those weights will be modified during training which take part in the formulation of the output value.

The learning rate $\mu$ affects the convergence and the speed of the convergence of the training process. However, the training speed depends not only on $\mu$, but also on the sequence of the training points. If more training points activate at least partly the same basis functions, there will be some learning interference which can slow down the convergence. Interference can happen because a given weight will be adjusted not only for a single training point, but also for many neighbouring ones. This means that after adjusting the selected weights for a given training point some of these weights will be adjusted again when such further training points are used, which are in the neighbourhood of the first one. To reduce the learning interference, only a relatively small learning rate can be applied and many training cycles must be used. A better possibility is to use such training points that just lie outside of the neighbourhood of the previous training points. This technique is called the Neighbourhood Sequential Training (Thompson and Kwon, 1995). In this case the training points are positioned at the nodes of a lattice, where the distance of the nodes is exactly $C$. Using this training technique and applying the optimal learning rate only a single cycle is enough to learn all the training points exactly. The response of the network for all the other input values can be determined easily. The network will perform a linear interpolation between the training points, so by using rather few training points (the points at the nodes defined just before) very quick learning and a relatively good answer can be obtained. (For example, if in a two-dimensional problem the input components are quantized to 10 bits and $C = 32$, the number of possible discrete input values is $2^{20}$, while the number of training points according to the Neighbourhood Sequential Training is about $2^{10}$. So only one thousandth of all the possible points are used to train the network.) Linear interpolation gives a good answer if the function to be learnt is 'smooth enough.' If further points inbetween the lattice nodes must be learnt, these points can be trained, using a much smaller learning rate, after having finished the Neighbourhood Sequential Training phase. In this second training phase learning interference cannot be avoided, so it needs more than one training cycle, but because we have to adjust only the response of the previously trained network, this second phase takes usually a relatively short time, too.

In practical applications it may happen that the training points are not arranged according to the nodes of the lattice, so this sequential training cannot be applied and training takes a much longer time. However, even in such cases the training time is much shorter than that of the corresponding MLP. The relatively fast learning is due to the linear output layer, the required simple operations (no multiplications) and the compact support of the basis functions.

When deciding upon the selection of a network architecture for a given application, however, not only the advantages but also the drawbacks have to be taken into consideration. The most serious drawbacks of the CMAC are its enormous weight space (when a network with a multi-dimensional input must be realized) and its limited modelling capability.

The necessary weight memory can be very large in the case of a multi-input network. For example, if in a ten-dimensional problem all input components are quantized to 10 bits and $C = 32$, the size of the weight memory is approximately $2^{55}$ words, which is absolutely impossible to implement today. To reduce the size of the weight memory hash coding, a random compression technique can be used (Albus, 1975). The basic idea is to project the elements of the original association vector onto a smaller, compressed association one. If compression is applied, the first layer implements two mappings: one from an input point to an association vector and the other from the association vector to the compressed one. All the elements of this compressed vector will select a weight in the weight memory. The mappings of the binary CMAC including compression are shown in Fig. 2. Although compression is almost necessary when the dimension of the input data is larger than three, we will not consider it in the sequel. (It can be shown (Ellison, 1991) that if the length of the compressed vector is not too small — at least $100 \div 1000C$, the effect of hash coding can be neglected.)

The weight space can be reduced using hash coding and because the cost of memory decreases rapidly, the size problem of the weight space is not so serious today as it was a few years ago. However, its limited modelling ability may have more serious consequences. When compared with the MLP, a multi-input CMAC is not a universal approximator. As is shown (Brown *et al.*, 1993), a binary CMAC could model exactly only multidimensional additive functions. This limited modelling capability comes from some consistency equations which impose constraints on given neighbouring training data. These equations are fulfilled only if the multidimensional

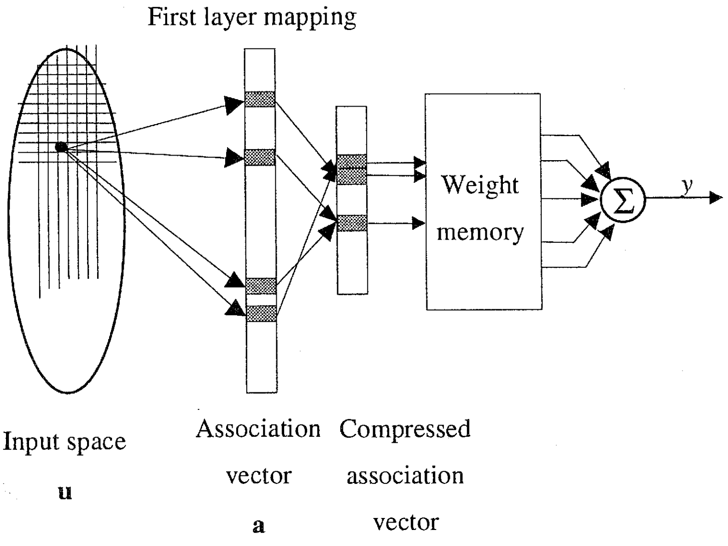


Fig. 2. The mapping of the CMAC.

mapping can be formed as a sum of one-dimensional ones, e.g. if the multi-input single-output non-linear mapping can be described as a sum of univariate functions. This is an inherent feature of the binary CMAC and the reason behind this property is the nature of the fixed non-linear mapping of the first layer and the use of rectangular basis functions.

## 3. Some CMAC Modifications to Improve the Modelling Capability

To overcome the limited modelling capability, some modifications or extensions of the original CMAC have been proposed. These proposals can be grouped into three main directions:

- to change the positions of the basis functions,
- to change (increase) the number of basis functions, or
- to use general basis functions instead of binary (rectangular) ones.

In the original Albus scheme, the positions of the overlays of the basis functions were determined by the main diagonal lattice nodes as was shown in Fig. 1. The positions of the $C$ different overlays are determined by the corresponding displacement vectors. Although it can be shown (Brown *et al.*, 1993) that the modelling capability of CMAC's with different displacement vectors are not the same, there is no binary CMAC (independently of the displacement vectors) that is a universal approximator.

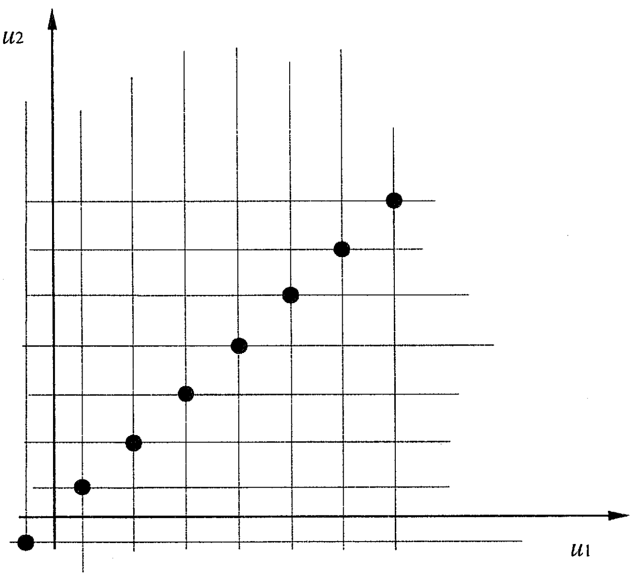
The only difference is the case of $C = 1$, which is a simple look-up table without any generalization capability. The displacement parameters of the network (and the generalization parameter, $C$) have effects on the distribution of the approximation error over the input space (Brown and Harris, 1994), so their selection can be an important step to design a CMAC for a given application.

Increasing the number of basis functions can also help to reduce the approximation error. In Albus' scheme $C$ overlays of basis functions are used independently of the dimension of the problem. Lane *et al.* (1992) proposed some modifications where more basis functions were used than in the original version. The new basis functions are also arranged in overlays. The number of overlays can be increased from $C$ (this is the original Albus scheme) up to the maximum of $C^N$, which is called a fully covered overlay arrangement. In the fully covered case every lattice point determines a different overlay (see Fig. 3 for $C = 8$). As can be seen, there is a unique basis function for each input quantization cell, so this network is a universal approximator: all the training points positioned at the lattice points can be learnt without any error. The price of a better approximation capability is a great number of the required basis functions: the number of basis functions (the size of the weight vector) increases exponentially with the input dimension, making implementation of the network much more difficult, especially in multi-dimensional cases.

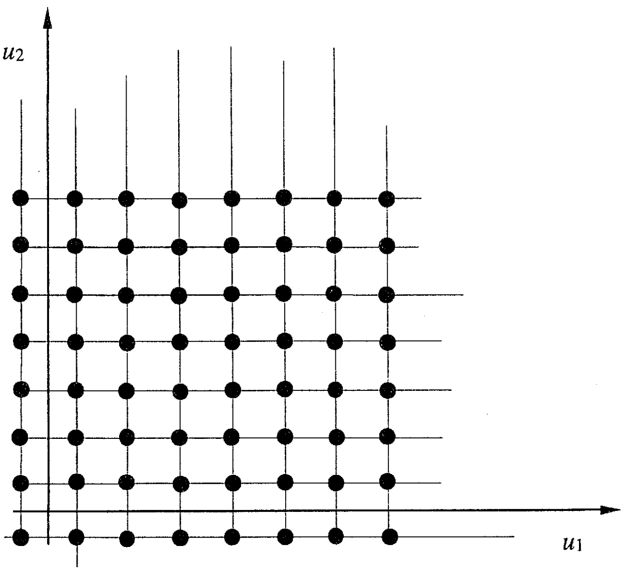
The application of sparsely distributed basis functions is only one reason of the limited modelling ability of the original CMAC. Another reason is the use of binary, rectangular basis functions. To use higher-order basis functions instead of binary ones helps us to improve network capabilities. Recently several papers have dealt with higher-order CMAC's, where the rectangular basis functions were replaced by general ones (Chiang and Lin, 1996; Eldracher *et al.*, 1997; González-Serrano *et al.*, 1998a; 1998b; Lane *et al.*, 1992). For general basis functions higher-order (dilated) B-spline basis functions (Lane *et al.*, 1992) are usually used. Univariate B-spline basis functions are piecewise polynomials, so the main advantage of using higher-order basis functions is that in this case the function and some of its derivatives can be approximated by the network.

Higher-order CMAC's are similar to B-spline networks (Brown and Harris, 1994). However, there are two significant differences: the supports and positions of the basis functions are different. For B-splines there is a close relationship between the order of the spline function ($O$) and the size of its support ($C$): $C = O$. In the case of a higher-order CMAC, when using dilated B-spline functions, the size of the support of the basis function is exactly $C$ which must be an integer multiple of its order: $C = nO$. (Figure 4 shows some univariate B-spline functions of different orders and a dilated second-order basis function when $C = 4$).

As regards the positions of the basis functions, in B-spline networks a fully covered overlay arrangement is always used, as opposed to the sparse distribution of basis functions for the CMAC. Higher-order CMAC's are usually used when not only the function, but also its derivatives have to be approximated. In this case, the CMAC performs a continuous mapping between the input and the output in such a way that the input is really not quantized. The outputs of the first layer of a higher-order CMAC (the elements of the association vector) are the responses of the selected basis



(a)



(b)

Fig. 3. Different overlay arrangements for $C = 8$: the original Albus scheme with $C$ overlays (a) and a fully covered input space with $C^N$ overlays if $N = 2$ (b).

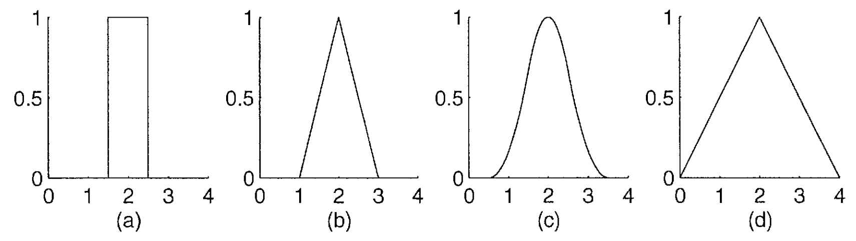


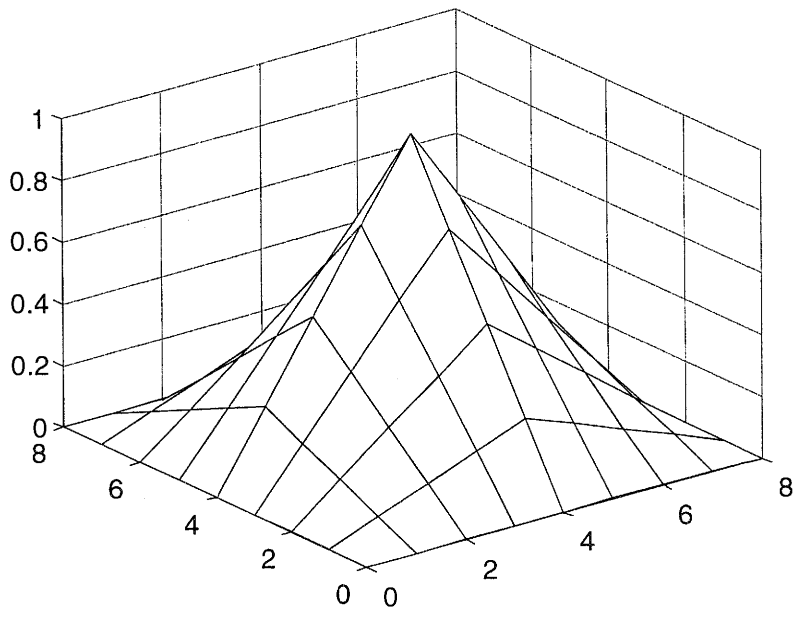
Fig. 4. B-spline functions of different orders.

functions: these responses depend on the relative positions of the input value within the supports of the basis functions. This means that the elements of the association vector are not constants (zeros or ones) as in the binary case, but may get any value of a higher-order basis function. This is the reason why the network cannot be implemented without multipliers when using higher-order basis functions. However, this is not the only drawback of higher-order CMAC's. To get association vectors, the responses of all the selected basis functions for an input point must be determined, i.e. quite a lot of extra computations are needed (Ker *et al.*, 1995).

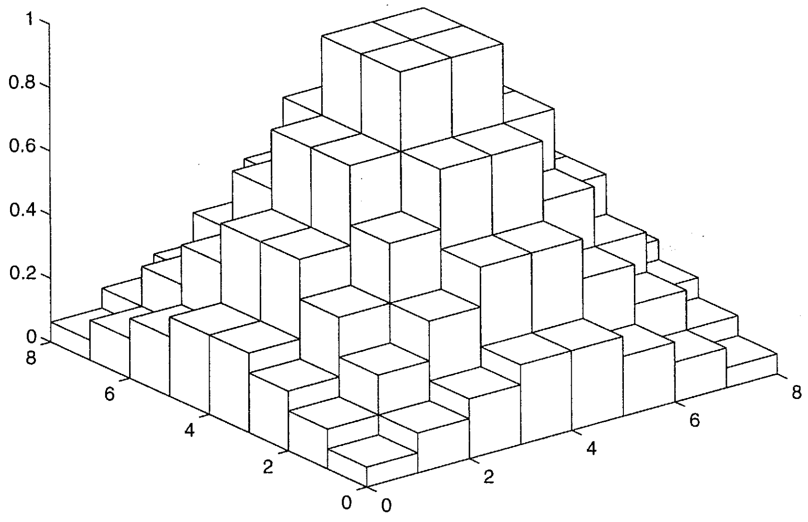
To overcome the difficulties of the basis function computations, discrete, piecewise constant versions of higher-order basis functions can be used where the value of the function is constant within a quantization interval. Figure 5 shows both the continuous and discrete versions of a two-dimensional second-order basis function. Two-dimensional functions can be obtained by the tensor product of one-dimensional ones.

The number of different values of a discrete function is finite. In this case, the quantized input is used again, but this is not a serious restriction as in many applications (like in discrete system modelling and control) the inputs are in digital form. Using discrete basis functions, the calculation of the association vectors is rather simple, however the multiplications to get the output of the whole network cannot be avoided. In the next section, a new multiplier structure will be presented making an efficient realization of higher-order discrete CMAC's possible.

Unfortunately, using higher-order basis functions and the original, sparse Albus overlay scheme, neither the function nor its derivatives can be approximated arbitrarily well. What is more, to the best of our knowledge, for networks with Albus' overlay scheme there is no general result about the approximation error using any order of basis functions. The approximation capability of the network (the errors at the training points and the generalization) highly depends on the parameters of the network (the quantization and the value of $C$). To illustrate this, Fig. 6 shows two different cases for a second-order CMAC. In this example, the function to be approximated was the two-dimensional sinusoid $y = \sin(u_1)\sin(u_2)$. In Fig. 6(a) the input components are quantized to six bits and $C = 8$ is chosen for the generalization parameter. If the training points are defined at the nodes of a regular lattice with the distance of 4, the network can learn the training points exactly, while its generalization is rather poor.

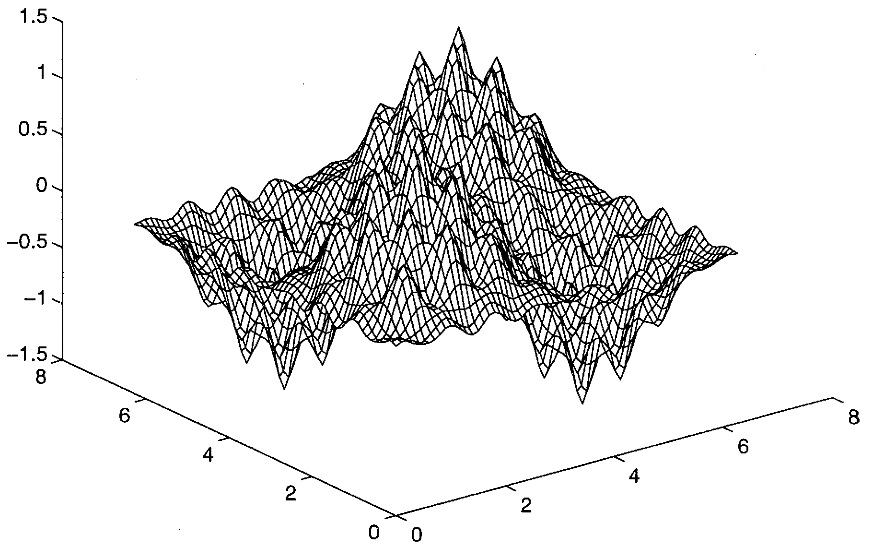


(a)

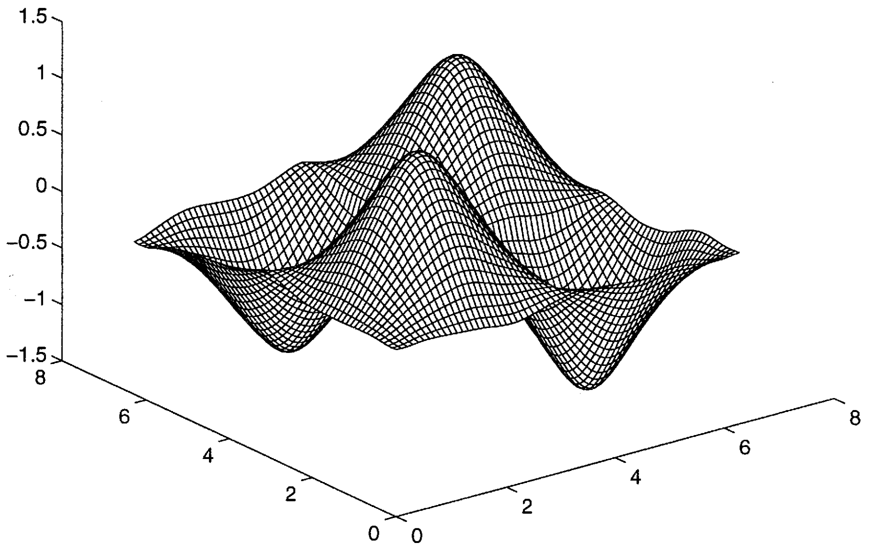


(b)

Fig. 5. Second-order continuous (a) and discrete (b) basis functions.



(a)



(b)

Fig. 6. Responses of a second-order CMAC with different generalization parameters: $C = 8$ (a) and $C = 32$ (b).

A much smoother output can be obtained if we increase the value of the generalization parameter. In Fig. 6(b) the output of the trained network can be seen when $C = 32$ and the other parameters are the same as in the previous case.

A further improvement of the capability of the network can be obtained if higher-order continuous functions and, at the same time, a fully covered overlay arrangement are used. Indeed, this is a second order B-spline network. As for the modelling capability of these networks, Commuri *et al.* (1995) stated that they are universal approximators. This means that the network can learn exactly all the training points that are positioned on the vertices of a hypercube lattice. The error of the network's response, $y = \widehat{f}(\boldsymbol{u})$, between the training points is bounded and can be made arbitrarily small if the parameters of the network are chosen properly, i.e. if the density of the lattice is high enough.

This result can be formulated as follows: if $f(\boldsymbol{u}) : \mathbb{R}^N \to \mathbb{R}^M$ is a continuous function on a compact subset $\Omega$ of $\mathbb{R}^N$ which is partitioned by a hypercube lattice, and $f(\boldsymbol{u}) = \widehat{f}(\boldsymbol{u})$ whenever $\boldsymbol{u}$ is a vertex of the lattice, the approximation error on $\Omega$ is bounded: $\|f(\boldsymbol{u}) - \widehat{f}(\boldsymbol{u})\| \leq \varepsilon$, where $\varepsilon = ML\delta$. Here $L$ is the Lipschitz constant of $f(\cdot)$ on $\Omega$ and $\delta$ is the maximum partition size of $\Omega$. (It must be mentioned that according to the knowledge of the authors this is a special case of general spline approximation results. In spline approximation exact results can be obtained for the approximation error of a function and some of its derivatives, see e.g. (de Boor, 1978; Schumaker, 1981).)

The required density of the lattice (as well as the number of basis functions) depends on the Lipschitz constant of the function to be approximated and the allowable approximation error. Increasing the density of the lattice will decrease the size of the support of the basis functions and decrease the generalization capability of the network. A further drawback of this approach (from the point of view of hardware implementation) is again the application of continuous basis functions, which needs additional computations and requires multiplications.

Figure 7 shows the arrangement of the basis functions as well as the positions of the training points (small black disks and open circles) for the two-dimensional case. The figure shows the four overlays. The black disks on an overlay are positioned exactly on the middle vertices of the support of the corresponding basis functions. These are the only basis functions that are responsible of forming the exact output values for these points, so if the training points are arranged in such a regular form, the response at any training point is determined uniquely by only one basis function.

## 4. A Further Modification with Discrete Basis Functions

In this section, we propose a modified version of the afore-mentioned approach. Here, similarly positioned basis functions are used, but instead of continuous functions, discrete ones are applied again. In the discrete version, the supports of the basis functions are quantized. For an $N$-dimensional case there will be $C^N$ quantization cells in each support. The discrete basis function suitable for this version (Fig. 8 for $C = 8$) is a bit different from the function shown in Fig. 5(b). This small difference

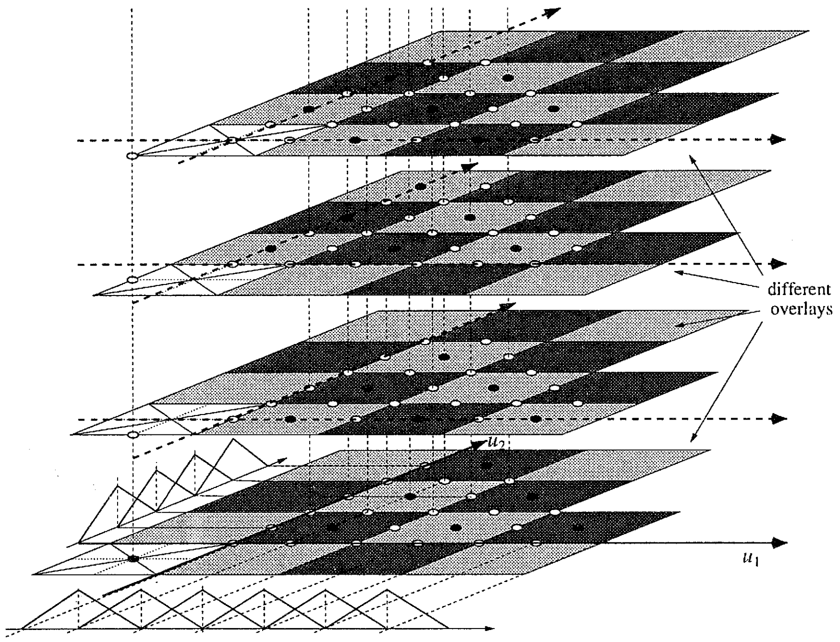


Fig. 7. A higher-order CMAC with fully covered overlay arrangement.

is important because it makes it possible to get a discrete counterpart of the second-order continuous B-spline network with a similar modelling capability. It is also a universal approximator in the sense that it can represent exactly every training datum positioned on a main vertex of the hypercube lattice and that the approximation error between the training points is bounded. The distance between the main vertices of the lattice is $C/2$ quantization intervals for all dimensions.

A further modification is that a hierarchical structure can be formed, which means that the generalization and the approximation error can be controlled independently of each other. The hierarchical structure means that if a network with $C$ generalization parameters results in a too large approximation error instead of increasing the density of the lattice, which would decrease the generalization capability of the network, further similar networks can be applied with smaller generalization parameters. The first network with generalization parameter $C$ can be followed by another one with parameter $C/2$ and a further one with $C/4$ etc. until the required error limit is reached. This approach which we call a hierarchical fully covered CMAC can be a trade-off between generalization and the approximation error. The suggested hierarchical structure is somewhat similar to Moody's proposal (Moody, 1989). However, while he increased the resolution of the network, i.e. when the error could not be reduced below a given limit, he used a next level of the network with a finer input quantization and the same generalization parameter; here the quantization is not

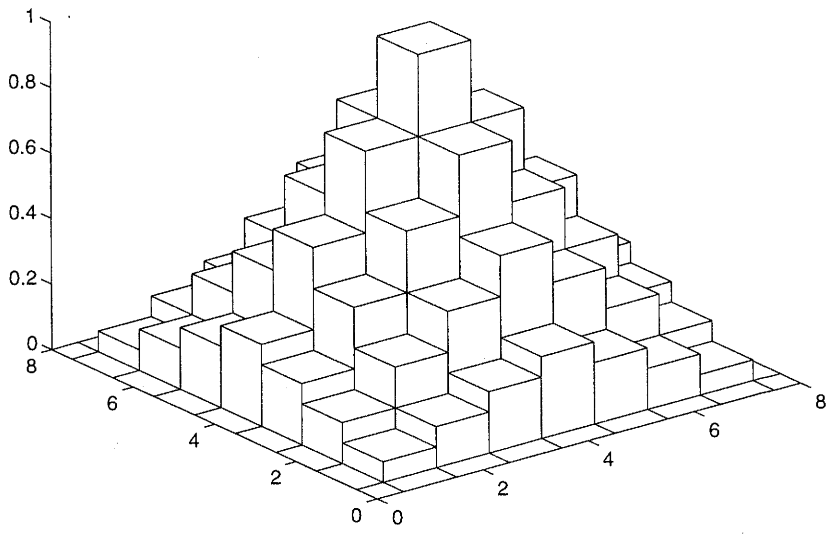


Fig. 8. The modified second-order discrete basis function.

changed, but the generalization parameter is reduced instead step by step.

This proposed version combines the advantageous properties of some previously mentioned networks:

- It uses a quantized input space and discrete basis functions, which means that the calculation of the association vector is simple: all the possible values of the association vectors are known in advance if the basis functions are fixed, so they can be stored or built into the network architecture.

- The supports of the basis functions are determined by the generalization parameter ($C$ ) of the network. However, the number of overlays does not depend on $C$. In the two-dimensional case with second-order basis functions there will be only four overlays, so the total number of basis functions is relatively small even if the generalization parameter is quite large. In a hierarchical structure — for the two-dimensional case — the maximum number of overlays is four times as large as the number of the hierarchy levels.

- It is a universal approximator, which means that similarly to the previous case all the training points can be learnt exactly. The positions of the training points are on the vertices of the hypercube lattice which are placed exactly $C/2$ quantization intervals from each other. The response of the network for such a discrete input that lies between the training points can be obtained as a linear interpolation of the responses of the neighbouring training points. The maximal error at these points is bounded, where the bound is determined by the

Lipschitz constant of the function to be approximated and the distance between the training points. For a given error bound and Lipschitz constant, the number of the required hierarchy levels can be determined.

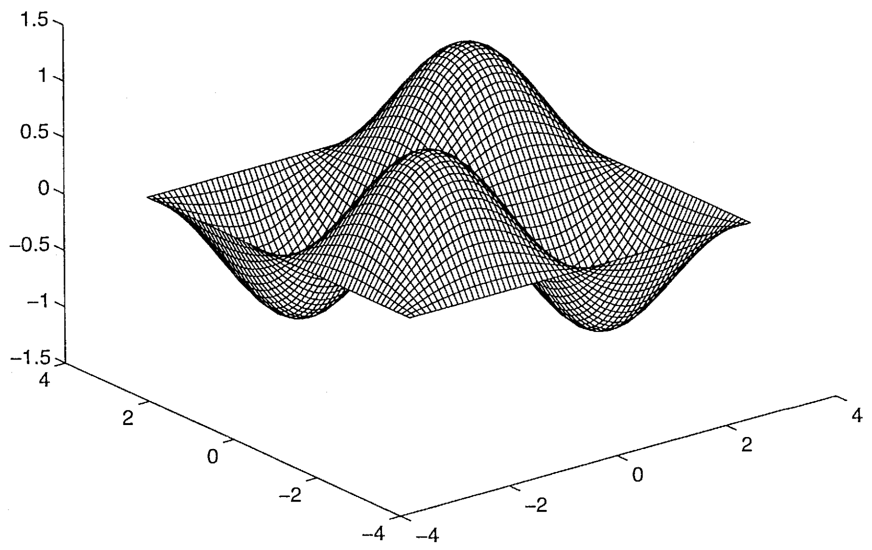
- The network can be trained extremely fast — one training cycle is enough — if similarly to the Neighbourhood Sequential Training method the training points are exactly on the vertices of the lattice of a given hierarchy level. We have to start with a hierarchy level of generalization parameter $C$ and go further to smaller generalization levels if we have further training points exactly at the corresponding lattice points. In many applications we are not in the ideal situation: the training points do not coincide with lattice points and they are distributed randomly over the input space instead. In these cases, the Neighbourhood Sequential Training cannot be applied, however the learning features of the network remain remarkably good. Figure 9 shows the results of two simulations using hierarchical networks. In Fig. 9(a) the training points are in the ideal positions and in Fig. 9(b) they are randomly distributed.
- Although it requires multiplications, it maintains a relatively simple hardware structure, so it is also suitable for low-cost digital implementations.

## 5. Dynamic CMAC Networks

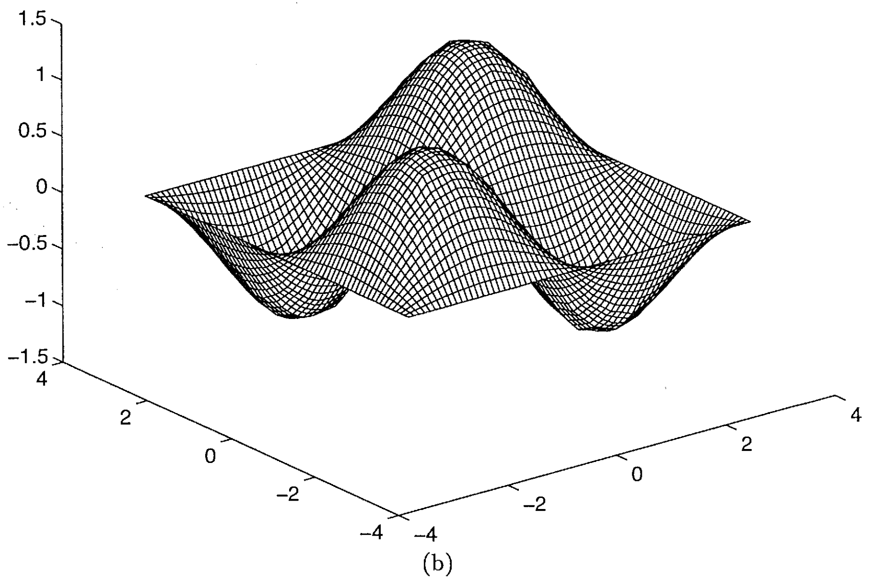
The CMAC is a static network. However, in system modelling and diagnosis in many cases we require a universal modelling device which reveals a dynamic behaviour. To form a dynamic net from the CMAC network some time dependence has to be built into it. A possible solution would be to use tapped delay lines at the inputs, similarly to the MLP-based solutions (Haykin, 1994). By using this approach, the modified network can be used as an alternative to MLP-based dynamic models, while some of the advantageous properties of the CMAC, like faster training and the possibility of simple digital implementation, are maintained. However, using tapped delay lines at the input increases the number of inputs of the CMAC network, which means that the size of the required memory may be too large. For obtaining dynamic CMAC networks such modifications are preferred where the number of the input components is not increased. A dynamic CMAC can be constructed (without increasing the number of the input components) if instead of using tapped delay lines at the input of the network the synaptic weights of the output layer are replaced by filters. If finite impulse response (FIR) filters are used, the result is the so-called FIR-CMAC which is a general NFIR model (Fig. 10). The input-output mapping of an NFIR model is given by

$$y(k) = f\Big(u(k), u(k-1), \ldots, u(k-L)\Big) \tag{4}$$

where $f(\cdot)$ is some nonlinear function. The FIR-CMAC can be considered as a piecewise linear filter or — as it is a cascade connection of a static fixed nonlinear



(a)



(b)

Fig. 9. Responses of the hierarchical network for $C = 16,\ 8, 4$: an ideal training point distribution (a) and randomly distributed training points (b).

mapping and a linear filter bank — it is a special implementation of the Hammerstein model

$$y(k) = \sum_{i=1}^{M} z_i(k) = \sum_{i=1}^{M}\sum_{p=0}^{L} w_i(p)a_i(k-p) = \sum_{i=1}^{M} \boldsymbol{w}_i^T \boldsymbol{a}_i(k) \tag{5}$$

where $\boldsymbol{a}_i(k) = [a_i(k), a_i(k-1), \dots, a_i(k-L)]^T$ is a vector formed from the delayed values of the $i$-th output of the first CMAC layer, $\boldsymbol{w}_i(k) = [w_i(0), w_i(1), \dots, w_i(L)]^T$ is the coefficient vector of the $i$-th filter with $L+1$ components and $z_i(k)$ is the output of the $i$-th FIR filter. The detailed operation, the learning rule and some further dynamic versions were presented in (Horváth and Dunay, 1996).

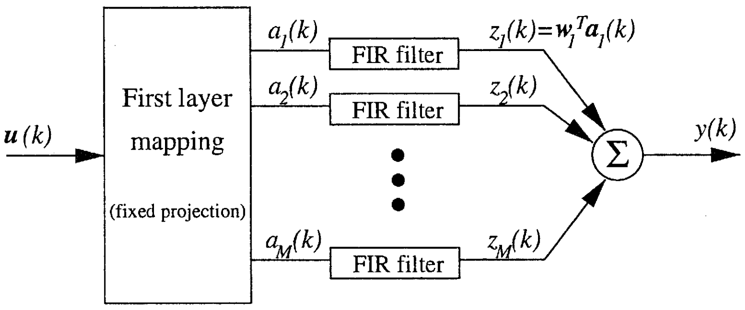


Fig. 10. FIR-CMAC network.

The proposed dynamic CMAC networks maintain the most significant advantageous features of CMAC networks and, as the first layer of the CMAC is not changed, the dynamic extensions can be applied for all versions of the previously mentioned networks.

As regards the complexity of the extended networks, only the size of the weight memory is increased. However, the required weight memory size increases only proportionally to the length of the filters $(L+1)$ when compared with the exponential increase in the memory size if a tapped delay line is used at the input.

## 6. Hardware Implementation

Although CMAC's are especially suited for digital hardware implementation, there are only a few papers (e.g. Miller *et al.*, 1991) which give reports about hardware versions of CMAC. The most recent ones are from (Ker *et al.*, 1995; 1997) who fabricated a VLSI CMAC chip which has been used as an embedded controller for colour correction in image reproduction systems. These realizations were developed for binary CMAC's or for higher-order CMAC's with continuous basis functions. In this section, we present some details of the hardware realization of the newly proposed versions using discrete higher-order basis functions.

In the previous sections, we suggested to use two different modified versions of CMAC. In the first case, the overlay arrangement and so the general architecture of the first layer of the original Albus scheme have been maintained. The most important feature of this version was the use of discrete higher-order basis functions. As was shown, the application of such basis functions results in a quite good approximation capability, while — as we will show in this section — using discrete basis functions, we can create an efficient hardware structure in spite of the need for multipliers. In the other proposed case, discrete basis functions are used again, but with a slight modification (cf. Figs. 5(b) and 8). This modification ensures that using a special fully covered overlay arrangement, a good approximation capability can be obtained in spite of the relatively sparse distribution of the basis functions.

## 6.1. Implementation of CMAC with Discrete Higher-Order Basis Functions

The main points of the implementation of a CMAC are the weight address mapping (the task of the first layer) and the computation of the dot product of the linear output layer.

**The address mapping method.** The first layer is responsible for the mapping from the quantized inputs to the association vectors. From the point of view of a realization, this mapping means the calculation of selected weight memory addresses. For every discrete input vector a proper number of weights must be selected, i.e. the addresses of the selected weights must be determined.

As in the first proposed version addressing is not changed, the classical main diagonal overlay arrangement is used and we will adopt the memory addressing method of Ker *et al.* (1997) which is an excellent solution for this task. Thus a direct memory address mapping method is applied where the whole weight memory is divided into $C$ logical memory banks. The main difficulty of the implementation of addressing is to find a way of efficient physical memory utilization and at the same time to give a possibility for parallel operation, i.e. to obtain the $C$ addresses (and the $C$ weights) in parallel. The essential point of the idea can be illustrated in a two-dimensional example shown in Fig. 11.

The weights are stored in $C$ memory banks where each bank contains memory blocks. The sparsely positioned rows of the original address space are pushed into such blocks. The blocks corresponding to the rows with indices 1, $C$, $2C$, $3C$, etc. are stored in the first memory bank with indices 2, $C+1$, $2C+1$, $3C+1$, etc. in the second bank, and so on. This results in a solution where for a given input point a weight memory cell is selected from each of the $C$ banks, i.e. the $C$ weights are retrieved in parallel. The necessary computational algorithm for this address mapping strategy is extensively detailed and optimized in the cited paper. The main advantages of this approach are that it is suitable for arbitrary dimensions and that by applying some week assumptions ($C$ must be an integer power of two and all the input components must be quantized to the same number of bits) this structure can be significantly simplified resulting in a simple combinational network.

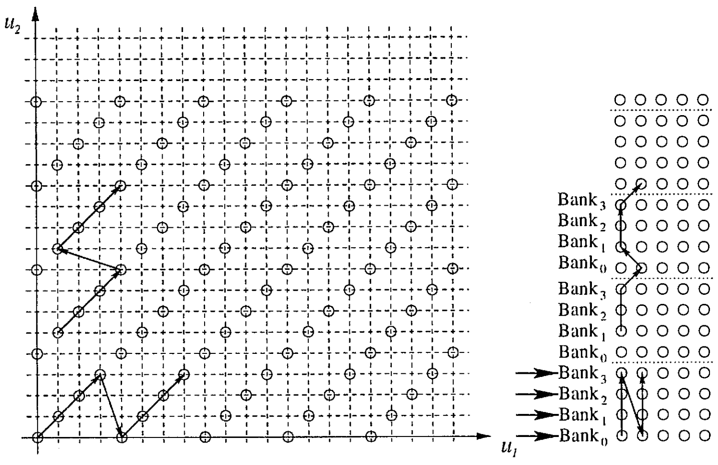


Fig. 11. Direct adressing and the illustration how to form memory banks and remove redundant memory cells (Ker *et al.*, 1995).

**Implementation of the output layer.** In the case of a binary CMAC, to get the answer of the network, it is enough to sum up the selected weights. In higher-order cases, the weight values have to be multiplied by the basis function values before summation, so we have to determine the responses of the selected basis functions for each discrete input value. The response of a selected basis function depends on the relative position of the input value within the receptive field of the basis function.

The dot product at the output layer needs $C$ multiplications. Our idea is based on the obsrevation that if instead of using $C$ general multipliers we apply a special matrix-vector multiplier, the output value can be calculated more efficiently. This special structure is derived from a bit-serial multiplier that can be applied if one of the multiplicands is constant, where the constant coefficient is built into the multiplier structure. For a general neural network, this special multiplier architecture can be applied only if its weights are fixed, so the network is pretrained and further on-line training is not required. Some papers deal with such applications (Szabó *et al.*, 1998; Szabó and Horváth, 1999). However, if on-line training is necessary, the weights of the neural network are not fixed. If the weights are not fixed for using this special multiplier structure, the other multiplicand must be constant.

For a higher-order CMAC with discrete basis functions, the number of different responses of a selected basis function is rather limited. Thus these fixed values can be built into the multiplier structure. This makes it possible to use the special multiplier structure even in the case of an on-line trainable CMAC network. In turn, this results in an efficient, hardware-cost-optimized network structure.

The multiplier operates in a bit-serial mode. This increases the computational time, but decreases the necessary hardware and the number of interconnections. However, if all multiplications can be performed in parallel, the globally parallel architecture can compensate for the speed loss caused by locally serial computation/communication.

To construct the special matrix-vector multiplier, we have to determine all the possible different combinations of the constant values of the selected $C$ discrete basis functions. It can be shown (Szabó and Horváth, 1999) that in a CMAC with the main diagonal addressing mode the values of the active (selected) basis functions can form only $P = C^{N-1}$ different vectors. These vectors must be multiplied by the selected weight vector to get the output value. If $N$ and $C$ are not too large, the number of different constant vectors will also be rather limited, and from all these vectors a constant matrix can formed. In the case of an efficient realization of the multiplication of a constant matrix by variable weight vectors it may be worth of implementing this matrix-vector multiplication. The result of this multiplication is a vector whose only one element represents the output value of the network for a given input. This element can be selected according to the relative position of the current input value in the receptive fields of the selected basis functions.

In a single-input CMAC only one association vector-pattern $\boldsymbol{a}_p$ can be formed from the responses of the selected basis functions. The structure and length of this vector-pattern depend on the order of the basis functions and the value of parameter $C$. In multi-input CMACs there will be more different association vector-patterns depending on the number of the input components, the order of the basis function and the generalization parameter of the network too. From these association vector-patterns a constant matrix $\boldsymbol{A}_p$ can be formed. It is an $m \times n$ constant matrix, where $m$ is the number of different association vector-patterns (in our case $m = C^N$) and $n$ is the length of the association vector-pattern (in our case $n = C$). For obtaining the output value of the network, the weight vector will be multiplied by this matrix. For this constant matrix an optimized multiplier structure can be constructed in advance, so by using this approach an efficient hardware structure can be obtained.

Here we confine the examinations to at most several- (two- or three-) dimensional problems, because otherwise the memory requirements of the CMAC would be extremely large. If the dimension of the input space is higher, we propose to use a *tree of CMACs* structure or to implement conventional multipliers and LUTs (Look-Up Tables) with basis function values. In a two-dimensional case the $C$ active basis functions can form $C$ different combinations of the active association vector elements ($C$ different association vector-patterns). Figure 12 shows these association vector-patterns for a second-order CMAC if Albus' addressing scheme is used and $C = 4$.

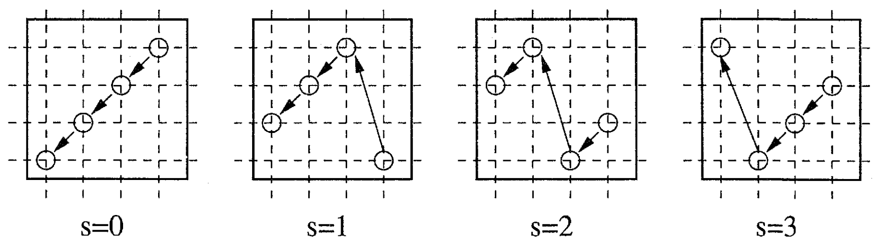


Fig. 12. Possible local association vectors for the two-dimensional case if $C = 4$.

The output of the higher-order CMAC will be the dot product of this association vector-patterns and the vector composed of the active weights: $y = \boldsymbol{a}_p^T \boldsymbol{w}$. However, if instead of this dot product the matrix-vector product $\boldsymbol{y} = \boldsymbol{A}_p \boldsymbol{w}$ is computed, the required scalar output $y$ has to be selected from the vector $\boldsymbol{y}$. This can be done using the relative position of the input value within the support of the selected basis functions. In a two-dimensional case, the selection formula is simple: $s = \mathrm{mod}_C(u_0 - u_1)$, where $s$ is the selecting index of the output vector. The block scheme of the system is given in Fig. 13.

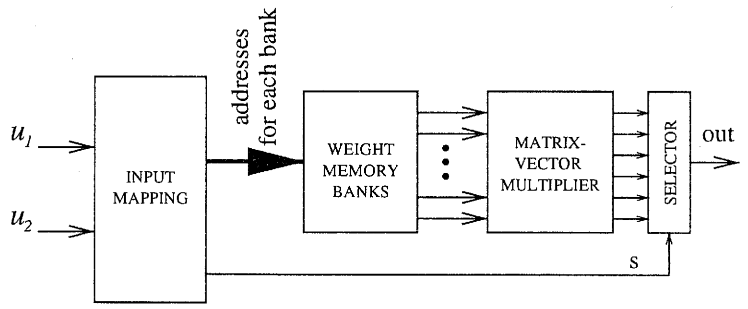


Fig. 13. The block diagram of the higher order CMAC system.

**Implementation of the matrix-vector multiplier.** The idea of efficient implementation of the matrix-vector multiplier is based on the 'do not implement unless it is necessary' rule. This means that given some (e.g. binary) representation of the elements of the coefficient matrix, where there are zero bits in this representation, the corresponding parts of the serial multiplier are needless to implement. Further, if there are similar bit patterns, it is enough to implement the pattern and the corresponding part of the multiplier once in the whole matrix-vector multiplier.

Such an optimization method for vector-vector (inner product) multiplication was originally presented in (Fehér, 1993). In this system, the constant coefficients are not stored explicitly but they are 'built' into the structure. This kind of approach is 'much more distributed' than those in the literature of Distributed Arithmetic. A two-step version of this approach for matrix-vector multiplication was introduced in (Szabó *et al.*, 1998). At first, the partial vector-vector products of a matrix-vector product are reduced independently, then more reduction can be achieved by merging their identical parts. Our most recent paper (Szabó and Horváth, 1999) deals with a generalization of the original method. This algorithm seeks for optimal reduction in the whole matrix-vector product.

An obvious implementation of the matrix-vector product is the application of $m \times n$ serial-parallel multipliers followed by adder trees. Here $m$ and $n$ are the size parameters of the matrix again. On closer examination, such a multiplier is composed of simple one-bit adders and bit-delay elements (D flip-flops). Moreover, it can be found that its complexity depends on the number of non-zero elements in the binary representation of the coefficient matrix. At this point significant hardware cost reduction can be achieved by using Canonical Signed Digit (CSD) encoding (Hartley and Parhi, 1995). In CSD coding, as opposed to the standard binary representation where only 0's and 1's are used, 0, +1, and -1 digits are applied. In this case, the maximum number of non-zero elements in a CSD encoded number is $B_c/2$ where $B_c$ is the word length. Moreover, the average number of nonzero elements in the number representation is $B_c/3$. (The dynamic range of a CSD number also extends.) As can easily be seen, the application of $-1$ does not mean any extra cost in the hardware realization because the hardware costs of a bit-serial adder or substractor are the same.

Another idea is to reduce the occurrence of identical bit patterns in the CSD representation of the coefficients. For this reduction the distributive property of multiplication is exploited. The reduced version can be obtained using an iterative optimization algorithm which is similar to the steepest-descent method. Clearly, it is not guaranteed that the algorithm will find a global optimum — as finding the global optimum is an NP-complete problem — but a significant hardware cost reduction can be achieved. The convergence of the algorithm is guaranteed in a finite number of steps (Szabó and Horváth, 1999). This method reduces the hardware cost of the implementation of a matrix-vector multiplier significantly, and full-precision results are obtained. The approach is very suitable for FPGA and ASIC implementations.

Applying the proposed method for implementing the output layer multiplications of higher-order CMACs, the hardware complexity can be significantly reduced.

### 6.1.1. Implementation of the Hierarchical Fully Covered CMAC

Our next task is to find an efficient implementation for the hierarchical fully covered CMAC architecture. Here we are looking for similar solutions, where the active weights are retrieved in a parallel way. We restrict our study to two-dimensional problems again. In these cases, an input point activates four basis functions, which means that we have to implement four memory banks. In order to select only one weight from each bank for an input point, we have to organize the address space as follows (see Fig. 14): the four memory banks are denoted by $A, B, C, D$ and $Ai, Bj, \ldots$ denote the $i$-th and $j$-th memory addresses within banks $A$ and $B$, respectively. The consecutive memory addresses $A1, A2, A3, \ldots$ for bank $A$ and $B1, B2, B3, \ldots$ for bank $B$ are assigned to the weights which correspond to the basis functions represented by open circles in the figure. Here the representing point of a basis function is in its centre. So along both input components every second basis function is related to the same memory bank.

The address of each memory bank can be determined directly from the input data. First, segments of the addresses are calculated — every input component will determine an address segment. Then the address of a weight value in a memory bank can be formed as a concatenation of the segments. In a two-dimensional case, the two address segments can be computed as $u_i/2C$ and $(u_i + C)/2C$, $i = 1, 2$. Here $u_i$ means the value of the $i$-th component (the co-ordinate of the $i$-th element of the input vector). As we have two input co-ordinates, $u_2$ and $u_1$, we can form the four

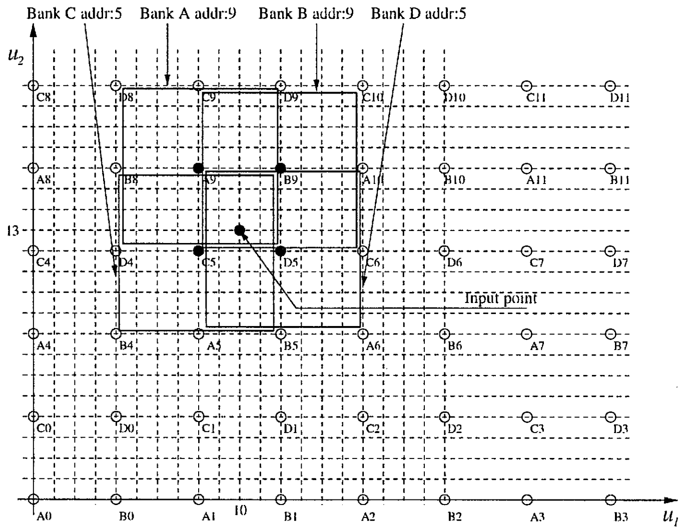


Fig. 14. An illustration of the addressing formula for a hierarchical fully-covered CMAC.

memory bank addresses as

$$\frac{u_2 + C}{2C} \;\&\&\; \frac{u_1 + C}{2C} \tag{6}$$

$$\frac{u_2 + C}{2C} \;\&\&\; \frac{u_1}{2C} \tag{7}$$

$$\frac{u_2}{2C} \;\&\&\; \frac{u_1 + C}{2C} \tag{8}$$

$$\frac{u_2}{2C} \;\&\&\; \frac{u_1}{2C} \tag{9}$$

where $\&\&$ stands for the bit-wise concatenation operation. We assume that $C$ is a power of two again. In the example of Fig. 14 ($C = 4$), the addresses are as follows:

$$A = \left[\frac{17}{8}\right] \&\& \left[\frac{14}{8}\right] = 1001_{\mathrm{BIN}} = 9$$

$$B = \left[\frac{17}{8}\right] \&\& \left[\frac{10}{8}\right] = 1001_{\mathrm{BIN}} = 9$$

$$C = \left[\frac{13}{8}\right] \&\& \left[\frac{14}{8}\right] = 0101_{\mathrm{BIN}} = 5$$

$$D = \left[\frac{13}{8}\right] \&\& \left[\frac{10}{8}\right] = 0101_{\mathrm{BIN}} = 5$$

Here $[\cdot]$ is the integer part function. Both the concatenation and determination of the integer part are very simple, and if the value of $C$ is an integer power of two, division by $2C$ is also a trivial operation when using either bit-serial or bit-parallel computation. In a bit-parallel environment integer divisions, the integer part operation and the bit concatenation do not require any additional hardware.

So far active basis functions have been selected. In the next step the basis function values have to be determined. This means that we have to determine the relative position of the current input point within each basis function receptive field. First, the co-ordinates of the centre of an active basis function must be obtained. This can be determined from the previously described address segments as

$$\left[\frac{u_i}{2C}\right] * 2C - C \tag{10}$$

and

$$\left[\frac{u_i + C}{2C}\right] * 2C \tag{11}$$

In our examples the selected basis function centres are: $A9 = (8,16), B9 = (12,16), C5 = (8,12)$ and $D5 = (12,12)$. To get the relative co-ordinates, these values must be substracted from the input point co-ordinates which are $P(10,13)$ in the example of the figure. According to the previous values, the relative co-ordinates will be $Ar = (2,-3), Br = (-2,-3), Cr = (2,1)$ and $Dr = (-2,1)$. These relative co-ordinate values (or these values with an offset C to avoid negative numbers) can be bit-wise concatenated and used as the addresses for the basis function values.

Now, we have the address values representing the relative positions of the current input point within the support of the selected basis functions. Using these addresses, the basis function values can be reached. At this point we could use the matrix-vector multiplier described in the previous section. However, because we have only four active basis functions in a two-dimensional case, this is not worth the trouble. Thus, here we propose to use conventional multipliers plus an LUT for the basis evaluation. That is, the relative basis function addresses can address LUT's directly.

Figure 15 shows the block diagram for the modified CMAC hardware. There are many blocks, denoted by dashed boxes, which do not require any hardware to implement when bit-parallel computations are used. Using a bit-serial arithmetic, these parts are also very simple to implement (they require simple masking, shift and add operations). A further advantage is that this result can be extended to higher input dimension cases too.

## 6.2. Implementation of the Dynamic CMAC Versions

The real difference between the static and dynamic versions is the use of filters instead of the weight values. The input layer and the addressing scheme of the dynamic version remain the same as that of the corresponding static version. However, instead of every memory bank, $L+1$ memory banks have to be used, so the total number of memory banks is $(L+1)C$ or $4(L+1)$ depending on the selected static CMAC

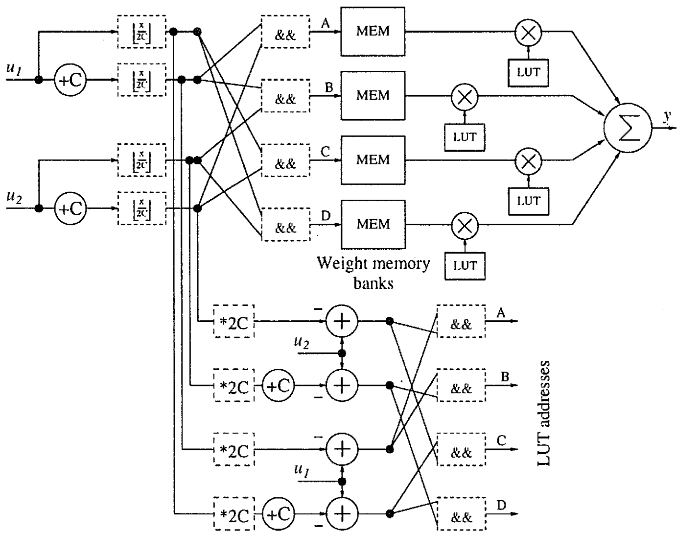


Fig. 15. Block scheme of the hierarchical fully-covered CMAC hardware.

version if $L + 1$ is the number of filter coefficients in every filter of the output layer. An open question is if an efficient matrix-vector multiplier can be applied for the dynamic versions too. This will be studied in the near future.

### 6.3. Implementation of Higher-Order B-Spline Networks

The proposed addressing method (Fig. 15) can be easily generalized to higher-order B-spline networks. The main difference between the second- and higher-order B-spline approximation, beyond the distinct shape of the basis function, is that the higher the order of the spline functions, the more overlays of the basis functions have to be used. If the basis function order is $O$, then the supports of the basis functions in different overlays are shifted by the $C/O$ quantization interval, where similarly to the previous cases the support of the basis functions in all the dimensions are divided into $C$ quantization intervals. From this it follows directly that one input point will activate $O^N$ basis functions in an $N$-dimensional space. Thus we have to select $O$ possible address segments for each dimension and combine them for address segment concatenation. If we consider the practically most important cubic spline case ($O = 4$), we have to select $4^2 = 16$ basis functions for a two-dimensional space. We can use the address segments computed from $u_i/4C, (u_i + C)/4C, (u_i + 2C)/4C$ and $(u_i + 3C)/4C$. However, we have to note that both the number of activated basis functions and the necessary weight memory space increase dramatically with the input space dimension and the order of approximation. Thus hereafter the usage of association vector compression (e.g. hash coding) and a hierarchical network structure should be considered, or at least the fully parallel network computation should be given up. The detailed analysis of these problems, however, is beyond the scope of this paper and requires further investigations.

## 7. Conclusions

In this paper, many different versions of the CMAC network as well as their most important properties were presented. The CMAC family is formed from networks which are characterized by a very interesting architecture (they have a single trainable layer), and have very attractive features such as fast training, an easy-to-implement structure, etc. Besides those advantageous features, these networks have some serious drawbacks, the most important of them being their limited modelling capability. The main aim of the paper was to show that using some extensions or modifications, some versions can be obtained which reveal an improved modelling capability, or at least versions where some bound can be determined for the modelling error while maintaining the possibility of an efficient digital hardware realization. For that purpose, the use of discrete higher-order basis functions and a special multiplier structure have been proposed. The networks can be used in systems modelling and, because of their good properties, they are especially suited for real-time applications in low-cost embedded systems.

## Acknowledgement

This work was partly sponsored by the Hungarian Fund for Scientific Research (OTKA) under contract T 021003.

## References

Albus J.S. (1975): *A new approach to manipulator control: The cerebellar model articulation controller (CMAC).* — Trans. ASME, J. Dyn. Syst., Meas. Contr., Vol.97, No.3, pp.220–227.

Broomhead D.S. and Lowe D. (1988): *Multivariable functional interpolation and adaptive networks.* — Complex Syst., Vol.2, pp.312–355.

Brown M., Harris C.J. and Parks P.C. (1993): *The interpolation capabilities of the binary CMAC.* — Neur. Netw., Vol.6, No.3, pp.429–440.

Brown M. and Harris C. (1994): *Neurofuzzy Adaptive Modelling and Control.* — New York: Prentice Hall.

Chiang Ching-Tsan and Lin Chun-Sin (1996): *CMAC with general basis functions.* — Neur. Netw., Vol.9, No.7, pp.1199–1211.

Commuri S., Lewis F.L. and Jagannathan S. (1995): *Discrete-time CMAC neural networks for control applications.* — Proc. 34-th Conf. *Decision and Control*, New Orleans, LA, Vol.2, pp.2420–2426.

De Boor C. (1978): *A Practical Guide to Splines.* — New York: Springer-Verlag.

Eldracher M., Staller A. and Pompl R. (1997): *Adaptive encoding strongly improves function approximation with CMAC.* — Neur. Comp., Vol.9, pp.403–417.

Ellison D. (1991): *On the convergence of the multidimensional Albus perceptron.* — Int. J. Robot. Res., Vol.10, No.4, pp.338–357.

Fehér B. (1993): *Efficient synthesis of distributed vector multipliers.* — Microprocess. Microprogramm., Vol.38, pp.345–350.

González-Serrano F.J., Artés-Rodríguez A. and Figueiras-Vidal A.R. (1998a): *Generalizing CMAC Architecture and Training.* — IEEE Trans. Neur. Netw., Vol.9, No.6, pp.1509–1514.

González-Serrano F.J., Artés-Rodríguez A. and Figueiras-Vidal A.R. (1998b): *The generalized CMAC.* — Proc. IEEE Int. Symp. *Circuits and Systems*, ISCAS'96. Atlanta, GA, pp.594–597.

Hartley R.I. and Parhi K.K. (1995): *Digit-Serial Computation.* — Boston: Kluwer Academic Publishers.

Haykin S. (1994): *Neural Networks: A Comprehensive Foundation.* — New York: Macmillan College.

Horváth G. and Dunay R. (1996): *Modelling of non-linear dynamic systems by using neural networks.* — Proc. IEEE Int. Symp. *Industrial Electronics*, Vol.1, pp.92–97.

Horváth G., Dunay R. and Pataki B. (1996): *Recurrent CMAC: A powerful neural network for system identification.* — Proc. IEEE Conf. *Instrumentation and Measurement Technology*, Brussels, Belgium, Vol.2, pp.992–997.

Ker J-S., Kuo Y-H. and Liu B-D. (1995): *Hardware realization of higher-order CMAC model for color calibration.* — Proc. Int. Conf. *Neural Networks*, ICNN'95 Australia, Adalaide, pp.1656–1661.

Ker J-S., Kuo Y-H., Wen R-C. and Liu B-D. (1997): *Hardware implementation of CMAC neural network with reduced storage requirement.* — IEEE Trans. Neur. Netw., Vol.8, No.6, pp.1545–1556.

Lane S.H., Handelman D.A. and Gelfand J.J. (1992): *Theory and development of higher-order CMAC neural networks.* — IEEE Contr. Syst., pp.23–30.

Miller W.T.III., Box B.A., Whitney E.C. and Glynn J.M. (1991): *Design and implementation of a high speed CMAC neural network using programmable CMOS logic cell array,* In: Advances in Neural Information Processing System (R.P. Lippmann and D.S. Touretzky, Eds.). — Morgan Kaufman, Vol.3, pp.1022–1027.

Moody J. (1989): *Fast learning in multi-resolution hierarchies.* In: Advances in Neural Information Processing System (D.S. Touretzky, Ed.), Morgan Kaufman, Vol.1, pp.29–39.

Schumaker L. (1981): *Spline Functions. Basic Theory.* — New York: John Wiley.

Szabó T., Fehér B. and. Horváth G. (1998): *Neural network implementation using distributed arithmetic.* — Proc. Int. Conf. *Knowledge-Based Intelligent Electronic Systems*, KES '98, Adelaide, Australia, Vol.3, pp.510–518.

Szabó T. and Horváth G. (1999): *Dedicated digital neural network implementations.* — (in preparation).

Thompson D.E. and Kwon S. (1995): *Neighborhood sequential and random training techniques for CMAC.* — IEEE Trans. Neur. Netw., Vol.6, No.1, pp.196–202.

Received: 6 January 1999
Revised: 20 April 1999