Kernel CMAC with Improved Capability

Gábor Horváth
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Magyar tudósok körútja 2.
H-1521. Budapest, Hungary
E-mail: horvath@mit.bme.hu

***Abstract* – Cerebellar model articulation controller (CMAC) neural network is a real alternative to MLP and RBF and has some advantageous features: its training is fast and its architecture is especially suitable for digital hardware implementation. The price of these attractive features is its rather poor capability. CMAC may have significant approximation and generalization errors. The generalization error can be significantly reduced if a regularization term is applied during training, while the approximation capability can be improved if the complexity of the network is increased. The paper shows that using a kernel interpretation the approximation capability of the network can be improved without increasing the complexity. It also shows, that regularization can be applied in the kernel representation too, so a new version of the CMAC is proposed where both approximation and generalization capabilities are improved significantly.**

I. Introduction

Cerebellar Model Articulation Controller (CMAC) [1], a special feed-forward neural architecture - which belongs to the family of feed-forward networks with a single linear trainable layer - has some attractive features. The most important ones are its extremely fast learning capability and the special architecture that lets effective digital hardware implementation possible [2], [3]. The CMAC architecture was proposed by Albus in the middle of the seventies [1] and it is considered as a real alternative to MLP and other feedforward neural networks [4]. Although the properties of CMAC were analysed mainly in the nineties, some interesting features were only recognised in the last years. These results show that the attractive properties of the CMAC have a price: its modeling capability is inferior to that of an MLP [5]. This is especially true for multivariate cases, as multivariate CMACs can approximate well only a function belonging to the additive function set. A further deficiency of CMAC is that its generalization capability is also inferior to that of an MLP even for univariate cases. The real reason of this property was presented in [6] and a modified training algorithm was proposed for improving the generalization capability. This training algorithm is derived using a regularized loss function, where regularization term has some weight-reconciliation or weight-smoothing effect. The approximation capability can be improved if the complexity of the network is increased. This more complex network was proposed in [7], but as the complexity of the CMAC depends on the dimension of the input data, in multivariate cases the high complexity can be an obstacle of implementation in any way.

CMAC can be interpreted as a network where the output is formed as a sum of weighted basis functions. The original version of the CMAC, the binary CMAC applies rectangular (first-order B-spline) basis functions of fixed positions.

In [8] it was shown, that CMAC can be considered as a kernel machine with second-order B-spline kernel functions where the positions of the kernels depend on the training data. Using this kernel interpretation the complexity of the network is bounded: the number of kernel functions is less than or equal to the number of training points similarly to any other kernel machines. This interpretation implies, that it is easy to construct such a kernel CMAC that can approximate any training data set exactly even in multivariate cases.

This paper presents different versions of the kernel CMAC and shows the relation between the original CMAC and its kernel versions. It shows that weight-smoothing regularization can be applied for the kernel CMAC too, which means that both the approximation and the generalization capability can be improved signify-cantly. The paper also shows that similarly to the original CMAC the kernel versions can also be trained in real time, which may be important in such applications where real-time on-line adaptation is required. The paper is organized as follows. In section II a brief summary of the original CMAC is given. In section III the different kernel versions and the main results are presented, while in section IV some examples are given to illustrate the improved properties of the proposed kernel CMACs.

II A short overview of the CMAC

CMAC is an associative memory type network, which performs two subsequent mappings. The first one, which is a non-linear mapping, projects an input space point $\mathbf{u}\in\Re^N$ into an association vector $\mathbf{a}$. The second one calculates the output $y\in\Re$ of the network as a scalar product of the association vector $\mathbf{a}$ and the weight vector $\mathbf{w}$

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

The association vectors are sparse binary vectors, which have only C active elements, C bits of an association vector are ones and the others are zeros. As the association vectors are binary ones, scalar products can be implemented without any multiplication; the scalar product is nothing more than the sum of the weights selected by the active bits of the association vector.

$$y(\mathbf{u}) = \sum_{i:a_i(\mathbf{u})=1} w_i \qquad (2)$$

CMAC uses quantized inputs, so the number of the 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.

Another interpretation can also be given to the CMAC. In this interpretation for an N-variate CMAC every bit in the association vector corresponds to a binary basis function with a compact N-dimensional hypercube support. The size of the hypercube is C quantization intervals. This means that a bit will be active if and only if the input value is within the support of the corresponding basis function. This support is often called receptive field of the basis function.

The mapping from the input space into the association vector should have the following characteristics: (i) it should map two neighbouring input points into such association vectors, where only a few elements - i.e. few bits - are different, (ii) as the distance between two input points grows, the number of the common active bits in the corresponding association vectors decreases. The input points far enough from each other - further then the neighbourhood determined by the parameter C - should not have any common bits. This mapping is responsible for the non-linear property and the generalization of the whole system.

The two mappings are implemented in a two-layer network architecture. The first layer implements a special encoding of the quantized input data, this layer is fixed; the trainable elements, the weight values which can be updated using the simple LMS rule, are in the second layer.

The way of encoding, the positions of the basis functions in the first layer, determine the generalization property of the network. In one-dimensional cases every quantization interval will determine a basis function, so the number of basis functions is approximately equal to the number of possible discrete inputs. However, if we follow this rule in multivariate cases, the number of basis functions will grow exponentially with the number of input variables, so the network may become too complex. 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, to the length of the association vector. In multivariate cases the weight memory can be so huge that practically it cannot be implemented. To avoid this high complexity the number of basis functions must be reduced. In a classical multivariate CMAC this reduction is achieved with using basis functions positioned only at the diagonals of the quantized input space as it is shown in Figure 1.

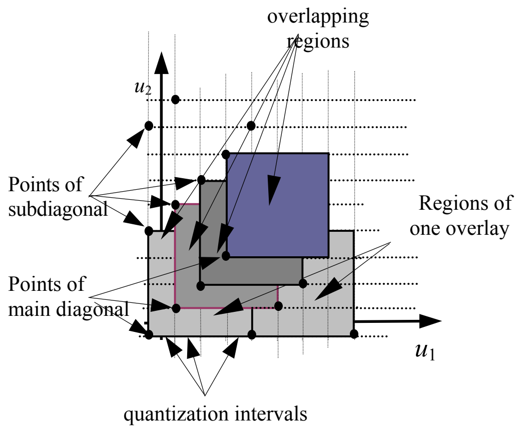

Fig. 1. The basis functions of a two variable CMAC

The shaded regions in Figure 1 are the receptive fields of different basis functions. As it is shown the basis functions are 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. Every input data 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.

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 in the main diagonal of the input space, while the basis-function-positions are represented by the subdiagonal points, as it is shown in Figure 1 (black dots). In the original Albus architecture the number of overlays does not depend on the dimension of the input vectors; it is always C. This means that in multivariate cases the number of basis function will not grow exponentially with the input dimension. This is an advantageous property from the point of view of implementation, however this reduced number of basis functions is the real reason of the inferior modeling capability of the multivariable CMACs, as reducing the number of basis functions the number of free parameters will also be reduced.

It should be mentioned that in multivariate cases further complexity reduction may be required. This reduction is achieved by applying a new compressing layer [1], which uses hash-coding. However, the effect of hashing can be neglected when its features are selected properly [9], further in the proposed kernel CMAC hashing is not required, so we will not deal with this effect.

The consequence of the reduced number of basis functions is that an arbitrary classical binary multivariate CMAC can reproduce exactly the training points only if they obtained from an additive function [5]. For more general cases there will be modeling error i.e. error at the training points.

Modeling error can be reduced only if the number of basis functions is increased. This question was addressed in [8]. General modeling capability (that is a network with capability to learn all training data points without error) can be obtained only if a full-overlay version is used where the number of basis functions increases exponentially with the input dimension; for an N-dimensional problem the number of overlays is C^N.

The generalization error of the binary CMAC was studied in [6]. In this paper the generalization error is defined as the difference between the response of the CMAC and a piecewise linear approximation of the mapping to be learned

$$H = y - y_{lin} . \quad (3)$$

The piecewise linear approximation y_{lin} gives exact responses at the training points and linear interpolation in between. The generalization error depends on C and the positions of the training data points.

The maximal value of the relative generalization error for a univariate CMAC when equidistant training data are used can be derived analytically as:

$$h = \frac{H}{D_{\max}} = \min[(C-d),(2d-C)]\frac{1}{r}\left(1+2\frac{r-C}{2b+C-r}\right)D \quad (4)$$

Here d is the distance (measured in quantums) between two neighbouring training inputs, D is the instantaneous amplitude of the function to be learned, $r=\sqrt{C^2-4b^2}$ where $b=C-d$. The generalization error of Eq. (4) is shown in Figure 2 as a function of C/d.

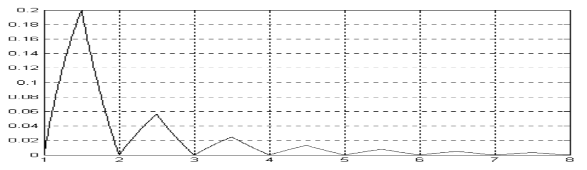

Fig. 2 The maximal relative generalization error as a function of C/d

The figure shows, that the result of a CMAC is exactly the same as the linear interpolation if C/d is integer, i.e. the relative generalization error defined before is zero. In other cases the generalization error can be rather significant. The main cause of the large generalization error is that the weight values take rather different values even if the mapping to be learned is constant.

This error can be reduced if a modified criterion function and – accordingly - a modified training rule is used. The new criterion function has two terms. The first one is a usual squared error term. The second is a weight-smoothing regularization term, which will be responsible for forcing the weights selected by an input data to be as similar as possible. The new criterion function is

$$J(k)=\frac{1}{2}\left(y_d(k)-y(k)\right)^2+\frac{\lambda}{2}\left(\frac{y_d(k)}{C}-w_i(k)\right)^2 . \quad (5)$$

where $y_d(k)$ is the desired output value for the kth training point. The modified training equation is as follows:

$$w_i(k+1)=w_i(k)+\mu(k)e(k)+\lambda\left(\frac{y_d(k)}{C}-w_i(k)\right), \quad (6)$$

where $e(k)=y_d(k)-y(k)$. The first part is the standard LMS rule, and the second part comes from the regularization term. λ is a regularization parameter and the values of μ and λ are responsible for finding a good compromise between the two terms. In Eqs. (5) and (6) $w_i(k)$ denotes the weights that are selected by the active bits $a_i(k)$ of the association vector $\mathbf{a}(k)=\mathbf{a}(\mathbf{u}(k))$ at the kth training step.

III Kernel CMAC

Kernel machines like Support Vector Machines (SVMs) [10], Least Squares SVMs (LS-SVMs) [11] and the method of ridge regression [12] apply a trick, which is called kernel trick. They use a set of nonlinear transformations from the input space to a "feature space". However, it is not necessary to find the solution in the feature space, instead it can be obtained in the kernel space, which is defined easily. All kernel machines can be applied for both classification and regression problems, but here we will deal only with the regression case.

The goal of a kernel machine for regression is to approximate a (nonlinear) function $y_d=f(\mathbf{u})$ using a training data set $\{\mathbf{u}_k, y_{d_k}\}_{k=1}^{P}$. First the input vectors are projected into a higher dimensional feature space, using a set of nonlinear functions $\boldsymbol{\varphi}(\mathbf{u}):\Re^N \to \Re^M$, then the output is obtained as a linear combination of the projected vectors:

$$y(\mathbf{u})=\sum_{j=1}^{M} w_j\varphi_j(\mathbf{u})+b=\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{u})+b \quad (7)$$

where $\mathbf{w}$ is the weight vector and b is a bias term. The dimensionality (M) of the feature space is not defined directly, it follows from the method (it can even be infinite). The kernel trick makes it possible to obtain the solution not in the feature space but in the kernel space

$$y(\mathbf{u})=\sum_{k=1}^{P}\beta_k K(\mathbf{u},\mathbf{u}_k)+b \quad (8)$$

where the kernel function is formed as

$$K(\mathbf{u}_k,\mathbf{u}_j)=\boldsymbol{\varphi}^T(\mathbf{u}_k)\boldsymbol{\varphi}(\mathbf{u}_j) \quad (9)$$

In (8) the β_k coefficients serve as the weights in the kernel space. The number of these coefficients equals to or less (in some cases it may be much less) then the number of training points [10].

A. The derivation of the kernel CMAC

The relation between CMAC and kernel machines can be shown if we recognize that the association vector of a CMAC corresponds to the feature space representation of the kernel machines. This means that the nonlinear functions, that map the input data points into the feature space, are the rectangular basis functions. The rectangular functions can be regarded as first-order B-spline functions of fixed positions.

To get the kernel representation of the CMAC we should apply (9) for the rectangular basis function. In univariate cases second-order B-spline kernels can be obtained where the centre parameters are the input training points. In multivariate cases the kernels will be different because of the reduced number of rectangular basis functions. However, we can apply the full-overlay CMAC, (although the dimension of the feature space would be unacceptable large). Because of the kernel trick the dimension of the kernel space will not increase: the number of kernel functions is upper bounded by the number of training points. Thus we can build a kernel version of a multivariate CMAC without reducing the length of the associate vector. This implies that this kernel CMAC can learn any training data set without error independently of the number of input variables. The multivariable kernel functions can be obtained as tensor products of univariate second order B-spline functions. This kernel function in two-dimensional case is shown in Figure 3a. This is the discretized version of the continuous second-order B-spline (Figure 3b.) according to the quantized input space of the CMAC.

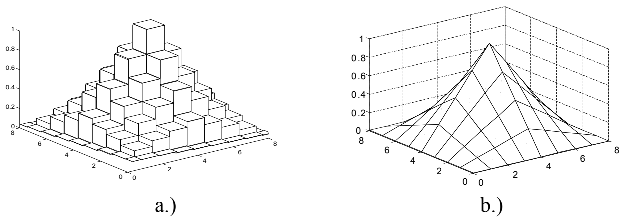
a.) b.)

Fig. 3. Second-order B-spline kernel functions for CMAC with C=4.

Kernel machines can be derived through constrained optimisation. The different versions of kernel machines apply different loss functions. Vapnik SVM for regression applies ε-insensitive loss function [10], while LS-SVM can be obtained if quadratic loss function is used [11]. The classical CMAC uses quadratic loss function too, so we obtain an equivalent kernel representation if in the constrained optimisation also quadratic loss function is used. This means that the kernel CMAC is similar to an LS-SVM. The optimisation task for a CMAC is as follows. We should find the solution of the following optimisation problem:

$$\min_{\mathbf{w}} J = \frac{1}{2}\sum_{k=1}^{P} e_k^2$$

such that (10)

$$y_{d_k} = \mathbf{w}^T \mathbf{a}_k + e_k, \quad k=1,\ldots,P$$

where $\mathbf{a}_k=\mathbf{a}(\mathbf{u}_k)$. As it was given in (1) the response of the network is:

$$y_k = y(\mathbf{u}_k) = \mathbf{w}^T \mathbf{a}_k, \quad k=1,...,P. \tag{11}$$

From (7) and (8) the optimal weight vector can be obtained as

$$\mathbf{w}^* = \mathbf{A}^\dagger \mathbf{y}_d \tag{12}$$

where $\mathbf{A}^\dagger = \mathbf{A}^T\left(\mathbf{A}\mathbf{A}^T\right)^{-1}$ is the pseudo inverse of the association matrix, a matrix built from the association vectors as row vectors, and $\mathbf{y}_d$ is a vector formed from all desired responses.

The response of a trained network for a given input is:

$$y(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})\mathbf{w}^* = \mathbf{a}^T(\mathbf{u})\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{y}_d \tag{13}$$

To see that this form can be interpreted as a kernel solution let construct an LS-SVM network with similar feature space representation. For LS-SVM regression we seek for the solution of the following constrained optimisation.

$$\min_{\mathbf{w}} J(\mathbf{w},\mathbf{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{k=1}^{P} e_k^2$$

such that (14)

$$y_{d_k} = \mathbf{w}^T \mathbf{a}_k + e_k$$

Here there is no bias term, as in the classical CMAC bias term is not used. The problem in this form can be solved constructing the Lagrangian

$$L(\mathbf{w},\mathbf{e},\alpha) = J(\mathbf{w},\mathbf{e}) - \sum_{k=1}^{P}\alpha_k\left(\mathbf{w}^T\mathbf{a}_k + e_k - y_{d_k}\right) \tag{15}$$

where α_k are the Lagrange multipliers. The conditions for optimality can be given by

$$\begin{cases} \dfrac{\partial L(\mathbf{w},\mathbf{e},\alpha)}{\partial \mathbf{w}} = \mathbf{0} \;\rightarrow\; \mathbf{w} = \sum_{k=1}^{P}\alpha_k\mathbf{a}_k \\ \dfrac{\partial L(\mathbf{w},\mathbf{e},\alpha)}{\partial e_k} = 0 \;\rightarrow\; \alpha_k = \gamma e_k \qquad k=1,...,P \\ \dfrac{\partial L(\mathbf{w},\mathbf{e},\alpha)}{\partial \alpha_k} = 0 \;\rightarrow\; \mathbf{w}^T\mathbf{a}(\mathbf{u}_k) + e_k - y_{d_k} = 0 \quad k=1,...,P \end{cases} \tag{16}$$

Using the results of (16) in (15) the Lagrange multipliers can be obtained as a solution of the following linear system

$$\left[\mathbf{K} + \frac{1}{\gamma}\mathbf{I}\right]\boldsymbol{\alpha} = \mathbf{y}_d \tag{17}$$

Here $\mathbf{K} = \mathbf{A}\mathbf{A}^T$ and $\mathbf{I}$ is a $P \times P$ identity matrix. The response of the network can be obtained as

$$y(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})\mathbf{w} = \mathbf{a}^T(\mathbf{u})\sum_{k=1}^{P}\alpha_k\mathbf{a}_k = \sum_{i=1}^{P}\alpha_k K(\mathbf{u},\mathbf{u}_k) = \mathbf{K}^T(\mathbf{u})\boldsymbol{\alpha}$$

$$= \mathbf{a}^T(\mathbf{u})\mathbf{A}^T\left[\mathbf{K} + \frac{1}{\gamma}\mathbf{I}\right]^{-1}\mathbf{y}_d = \mathbf{a}^T(\mathbf{u})\mathbf{A}^T\left[\mathbf{A}\mathbf{A}^T + \frac{1}{\gamma}\mathbf{I}\right]^{-1}\mathbf{y}_d \tag{18}$$

The obtained kernel machine is an LS-SVM or more exactly a ridge regression solution [12], because of the lack of the bias term. Comparing (13) and (18), it can be seen that the only difference between the classical CMAC and the ridge regression solution is the term (1/γ)**I**, which comes from the modified loss function of (14). However, if the matrix $\mathbf{AA}^T$ is singular or it is near to singular, that may cause numerical stability problems in the inverse calculation, a regularization term must be used: instead of computing $(\mathbf{AA}^T)^{-1}$ the regularized inverse $(\mathbf{AA}^T + \eta\mathbf{I})^{-1}$ is computed, where η is the regularization coefficient. In this case the two forms are equivalent.

B. Kernel CMAC with weight-smoothing

This kernel representation improves the modeling property of the CMAC. As it corresponds to a full-overlay CMAC it can learn all training data exactly. However, the generalization capability is not improved. To get a CMAC with better generalization capability weight smoothing regularization should be applied for the kernel version too. Smoothing regularization can be obtained if a new term is added to the loss function of (10) or (14). Starting from (14) the modified optimization problem can be formulated as follows:

$$\min_{\mathbf{w}} J(\mathbf{w},\mathbf{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{k=1}^{P} e_k^2 + \frac{\lambda}{2}\sum_{k=1}^{P}\sum_{i}\left(\frac{y_{d_k}}{C} - w_k(i)\right)^2 \quad (19)$$

$w_k(i)$ is a weight value selected by the ith active bit of $\mathbf{a}_k$, so i runs through the indexes where $a_k(i)=1$. As the equality constraint is the same as in (10) or (14), we obtain the Lagrangian

$$L(\mathbf{w},\mathbf{e},\alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{k=1}^{P} e_k^2 + \frac{\lambda}{2}\sum_{k=1}^{P}\sum_{i}\left(\frac{y_{d_k}}{C} - w_k(i)\right)^2 - \sum_{k=1}^{P}\alpha_k\left(\mathbf{w}^T\mathbf{a}_k + e_k - y_{d_k}\right) \quad (20)$$

that can be written in the following form:

$$L(\mathbf{w},\mathbf{e},\alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{k=1}^{P} e_k^2 - \sum_{k=1}^{P}\alpha_k\left(\mathbf{a}_k^T diag(\mathbf{a}_k)\mathbf{w} + e_k - y_{d_k}\right) + \frac{\lambda}{2}\sum_{k=1}^{P}\frac{y_{d_k}^2}{C} - \lambda\sum_{k=1}^{P}\frac{d_k}{C}\mathbf{a}_k^T diag(\mathbf{a}_k)\mathbf{w} + \frac{\lambda}{2}\sum_{k=1}^{P}\mathbf{w}^T diag(\mathbf{a}_k)\mathbf{w} \quad (21)$$

where $diag(\mathbf{a}_k)$ is a diagonal matrix with $\mathbf{a}_k$ in its main diagonal. The conditions for optimality can be obtained similarly to (16). The Lagrange multipliers can be obtained again as a solution of a linear system.

$$\boldsymbol{\alpha} = \left(\mathbf{K_D} + \frac{1}{\gamma}\mathbf{I}\right)^{-1}\left(\mathbf{I} - \frac{\lambda}{C}\mathbf{K_D}\right)\mathbf{y}_d \quad (22)$$

where $\mathbf{K_D} = \mathbf{A}(\mathbf{I} + \lambda\mathbf{D})^{-1}\mathbf{A}^T$ and $\mathbf{D} = \sum_{k=1}^{P} diag(\mathbf{a}_k)$.

The response of the network becomes

$$y(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})(\mathbf{I} + \lambda\mathbf{D})^{-1}\mathbf{A}^T\left[\boldsymbol{\alpha} + \frac{\lambda}{C}\mathbf{y}_d\right] \quad (23)$$

Similar results can be obtained if we start from (10). In this case the response of the network will be:

$$y(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})(\lambda\mathbf{D})^{-1}\mathbf{A}^T\left[\boldsymbol{\alpha} + \frac{\lambda}{C}\mathbf{y}_d\right]. \quad (24)$$

where $\boldsymbol{\alpha}$ is also different from (22).

$$\boldsymbol{\alpha} = \left(\mathbf{A}(\lambda\mathbf{D})^{-1}\mathbf{A}^T + \mathbf{I}\right)^{-1}\left(\mathbf{I} - \mathbf{A}\mathbf{D}^{-1}\mathbf{A}^T\frac{1}{C}\right)\mathbf{y}_d \quad (25)$$

C. Real-time on-line learning

One of the most important advantages of kernel networks is that the solution can be obtained in kernel space instead of the possibly very high dimensional (even infinite) feature space. In the case of CMAC the dimension of the feature space can be extremely large for multivariate problems especially if full-overlay versions are used. However, another feature of the kernel machines is, that the kernel solutions are obtained using batch methods instead of adaptive techniques. This may be a drawback is many applications. The classical CMAC can be built using both approaches: the analytical solution is obtained using batch method or the networks can be trained from sample-to-sample iteratively. Adaptive methods are useful when one wants to track the changing of the problem. A further advantage of adaptive techniques is that they require less storage than batch methods since data are used only as they arrive and need not be remembered for the future.

The kernel CMACs derived above all use batch methods. Adaptive solution can be derived if these equations are written in different forms. The response of any kernel CMAC can be written in a common form:

$$y(\mathbf{u}) = \boldsymbol{\Omega}^T(\mathbf{u})\boldsymbol{\beta} \quad (26)$$

where $\boldsymbol{\Omega}(\mathbf{u})$ is the kernel space representation of an input point $\mathbf{u}$, and $\boldsymbol{\beta}$ is the weight vector in the kernel space. $\boldsymbol{\Omega}(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})\mathbf{A}^T$ for the classical CMAC (Eq. 18), while $\boldsymbol{\Omega}(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})(\mathbf{I} + \lambda\mathbf{D})^{-1}\mathbf{A}^T$ and $\boldsymbol{\Omega}(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})(\lambda\mathbf{D})^{-1}\mathbf{A}^T$ for the kernel CMACs with weight-smoothing (Eqs. 23 and 24, respectively). $\boldsymbol{\beta} = \boldsymbol{\alpha}$ for the classical CMAC and $\boldsymbol{\beta} = \boldsymbol{\alpha} + \frac{\lambda}{C}\mathbf{y}_d$ for the versions with weight-smoothing. Adaptive solution can be obtained if $\boldsymbol{\beta}$ is trained iteratively. Using the simple LMS rule the iterative training equation is:

$$\boldsymbol{\beta}(k+1) = \boldsymbol{\beta}(k) + 2\mu\boldsymbol{\Omega}(k)\mathbf{e}(k) \quad (27)$$

where $\boldsymbol{\Omega}(k)$ is a matrix formed from the kernel space representations of the training input vectors $\boldsymbol{\Omega}^T(\mathbf{u}_i)$, $i=1,\ldots,k$ and $\mathbf{e}(k) = [e_1, e_2, \ldots, e_k]^T$ at the kth training step. It should be mentioned that the size of $\boldsymbol{\Omega}(k)$ and the lengths of $\boldsymbol{\beta}(k)$ and $\mathbf{e}(k)$ are changing during training.

IV. Illustrative Experimental Results

The different kernel versions of the CMAC network were validated by extensive experiments. Here only the results for the simple *sinc* function approximation will be presented. For the tests both noiseless and noisy training data were used.

Figures 4 and 5 show the responses of the binary CMAC and the CMAC with weight-smoothing regularization, respectively. Similar results can be obtained for both the original and the kernel versions. In these experiments the inputs were quantized into 8 bits and were taken from the range [-4π,+4π].

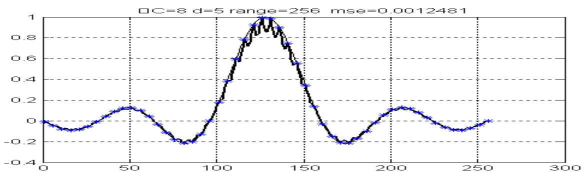

Fig. 4. The response of the CMAC (Eq. 13) with C=8, d=5

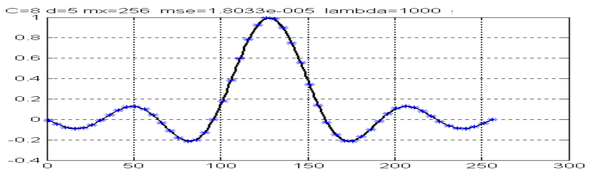

Fig. 5 The response of a weight-smoothing CMAC C=8, d=5, $\lambda=10^3$.

Figure 6, 7 and 8 show responses for noisy training data. In these cases the input range was [-6π,+6π], and the inputs were quantized into 9 bits. Gaussian noise with zero mean and σ = 0.3 was added to the ideal desired outputs.

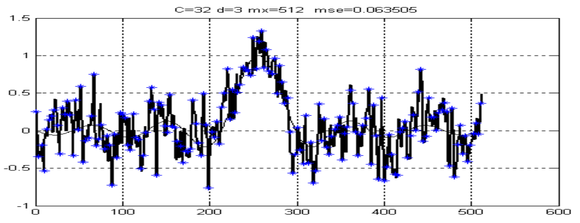

Fig. 6. The response of the CMAC (Eq. 13) with C=32, d=3

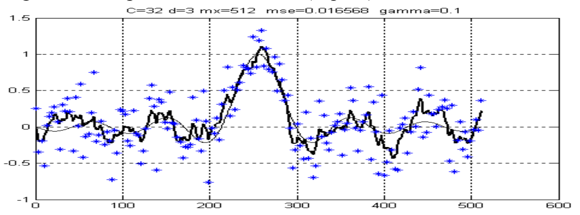

Fig. 7. The response of the ridge regression kernel CMAC without weight-smoothing (Eq. 18) C=32, d=3, γ=0.1

In all figures the solid lines with larger line-width show the response of the network, while with the smaller line-width show the ideal function and the training points are marked with $\ast$. Similar results can be obtained for multivariate problems. Simulation results also show, that on-line training is efficient, the convergence is fast: few (2-3) epochs of the training data are enough to reach the batch solutions within the numerical accuracy of MATLAB if μ is properly selected.

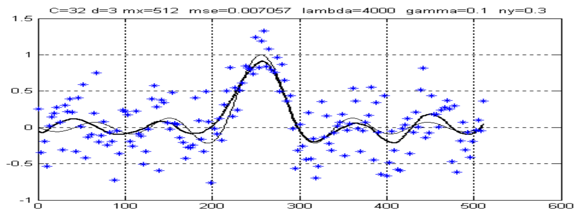

Fig. 8. The response of the ridge regression kernel CMAC with weight-smoothing (Eq. 23) C=32, d=3, γ=0.1, λ=4000

V. Conclusions

In this paper it was shown that CMAC can be interpreted as a kernel machine. An important consequence of this interpretation is that in this way the full-overlay version can be easily implemented even in multivariate cases, improving the modeling capability. The paper also shows that generalization capability can also be mproved as weight-smoothing regularization can be appied for kernel CMAC too. The possibility of adaptive raining ensures that the main advantages of the classical MAC (adaptive operation, fast training, simple digital ardware implementation) can be maintained, although he multiplierless structure is lost.

Acknowledgement

This work was partly sponsored by National Fund for Scientific Research (OTKA) under contract T 046771.

References

[1] Albus, J.S. "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)", *Transaction of the ASME*, pp. 220-227, Sep. 1975.
[2] Thompson D.E. and Kwon S. "Neighbourhood Sequential and Random Training Techniques for CMAC". *IEEE Trans. on Neural Networks*, Vol. 6, pp. 196-202, 1995.
[3] Ker, J.S. - Kuo, Y.H. - Liu, B.D. "Hardware Realization of Higher-order CMAC Model for Color Calibration", *Proc. of the IEEE Int. Conf. on Neural Networks*, Perth, Vol. 4, pp. 1656-1661, 1995.
[4] Miller, T.W. III. Glanz, F.H. and Kraft, L.G. "CMAC: An Associative Neural Network Alternative to Backpropagation" *Proceedings of the IEEE*, Vol. 78, pp. 1561-1567, 1990
[5] Brown, M. and Harris, C.J. "Neurofuzzy Adaptive Modeling and Control" Prentice Hall, New York, 1994.
[6] Szabó, T. and Horváth, G. "Improving the Generalization Capability of the Binary CMAC" *Proc. Int. Joint Conf. on Neural Networks, IJCNN'2000.* Como, Italy, Vol. 3, pp. 85-90, 2000.
[7] Lane, S.H. - Handelman, D.A. and Gelfand, J.J "Theory and Development of Higher-Order CMAC Neural Networks", *IEEE Control Systems*, Vol. Apr. pp. 23-30, 1992.
[8] Horváth, G. "CMAC: Reconsidering an Old Neural Network" *Proc. of the Intelligent Control Systems and Signal Processing, ICONS 2003*, Faro, Portugal. pp. 173-178, 2003.
[9] Ellison, D. "On the Convergence of the Multidimensional Albus Perceptron", *The International Journal of Robotics Research*, Vol. 10, pp. 338-357, 1991.
[10] Vapnik, V. "Statistical Learning Theory", Wiley, New York, 1995.
[11] Suykens, J.A.K., Van Gestel, T, De Brabanter, J., De Moor, B. and Vandewalle, J. "Least Squares Support Vector Machines", World Scientific, Singapore, 2002.
[12] Saunders, C.- Gammerman, A. and Vovk, V. "Ridge Regression Learning Algorithm in Dual Variables. Machine Learning", *Proc. e Fifteenth Int. Conf. on Machine Learning*, pp. 515-521, 1998.