CMAC: RECONSIDERING AN OLD NEURAL NETWORK

Gábor Horváth

Budapest University of Technology and Economics,
Department of Measurement and Information Systems
Magyar tudósok krt. 2. Budapest, Hungary H-1117.

Abstract: Cerebellar Model Articulation Controller (CMAC) has some attractive features: fast learning capability and the possibility of efficient digital hardware implementation. Although CMAC was proposed many years ago, several open questions have been left even for today. Among them the most important ones are about its modelling and generalization capabilities. The limits of its modelling capability were addressed in the literature and recently a detailed analysis of its generalization property was given. The results show that there are differences between the one-dimensional and the multidimensional versions of CMAC. The modelling capability of a multidimensional network is inferior to that of the one-dimensional one. This paper discusses the reasons of this difference and suggests a new kernel-based interpretation of CMAC. It shows that a one-dimensional binary CMAC can be considered as an SVM with second order B-spline kernel function. Applying this approach the paper shows that one-dimensional and multidimensional CMACs can be constructed with similar modelling capability.

Keywords: Neural Network, Support Vector Machines, CMAC, Ridge regression

1. INTRODUCTION

The paper deals with CMAC, a special neural architecture (Albus, 1975). CMAC has some attractive features. The most important ones are its extremely fast learning capability (Thompson and Kwon, 1995) and the special architecture that lets effective digital hardware implementation possible (Miller *et al.*, 1991; Horváth and Deák, 1993; Ker, *et al*., 1995). CMAC architecture was proposed in the middle of the seventies and it has been mentioned as a real alternative of MLP (Miller *et al.*, 1990). The price of the advantageous properties is that its modelling capability is inferior to that of an MLP (Commuri *et al*., 1995). This especially true for multidimensional case, as a multidimensional CMAC can approximate well only a function belonging to the additive function set (Brown, *et al.*, 1993). A further deficiency of CMAC is that its generalization capability is also inferior to that of an MLP even for one-dimensional cases. This drawback was discussed in (Szabó and Horváth, 2002), where the real reason of this property was discovered and a modified training algorithm was proposed for improving the generalization capability.

However, the other disadvantage, the limited modelling capability in multidimensional cases cannot be improved using the proposed modification. This paper will deal with this question. It will show that the limited modelling capability comes from the architecture of the multidimensional CMAC. The architecture can be modified in such a way that the modified version will have improved modelling capability (Lane *at al.*, 1992). However, this modification would increase the network complexity. This increased complexity is unacceptable from the viewpoint of realization if the input dimension is (much) larger than two.

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

This paper shows that an alternative interpretation can also be given, when the network is considered as an SVM (support vector machine) with second-order B-spline kernel functions, where the positions of the kernel functions depend on the training data. This interpretation can be applied for higher-order CMACs (Chiang Ching-Tsan and Lin Chun-Sin, 1996), (Ker *et al*., 1995) with higher order basis functions (k^{th} order B-splines). In these cases CMACs correspond to SVMs with properly chosen higher–order ($(2k\text{-}1)^{\text{th}}$ order) B-spline kernels.

2. A SHORT OVERVIEW OF THE CMAC

CMAC is an associative memory type neural network, which performs two subsequent mappings. The first one - which is a non-linear mapping - projects an input space point **u** into an association vector **a**. The second mapping calculates the output of the network as a scalar product of the association vector **a** and the weight vector **w**:

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

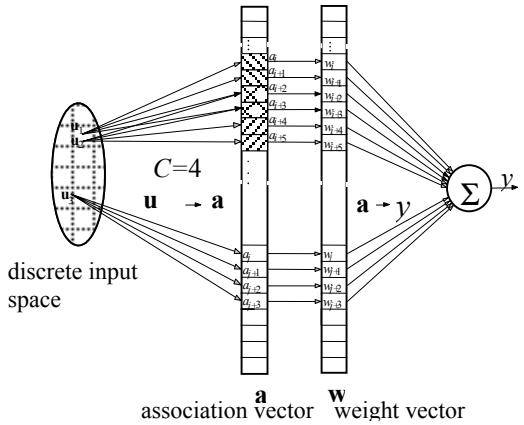


Fig. 1. The mappings of a CMAC

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. In this case scalar products can be implemented without multiplication; the scalar product is nothing more than the sum of the weights selected by the active bits of the association vector.

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

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 every bit in the association vector corresponds to a binary basis function with a finite support of C quantization intervals. This means that a bit will be active if the input value is within the support of the corresponding basis function. This support is often called as the receptive field of the basis function. An element of the association vector can be considered as the value of a basis function for a given input, so the output of the binary basis function is one if an input is in its receptive field and zero elsewhere:

$$a_i(\mathbf{u}) = \begin{cases} 1 & \text{if } \mathbf{u} \text{ is in the receptice field} \\ & \text{of the } i\text{th basis function} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

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 mapping 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 higher dimensional cases, the number of basis functions will grow exponentially with the dimension, 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 higher dimensional case the weight memory can be so huge that it cannot be implemented. To avoid this high complexity the number of basis functions must be reduced. In a classical higher–dimensional CMAC this reduction is achieved using basis functions positioned only at the diagonals of the quantized input space as it is shown in Figure 2.

The shaded regions in Figure 2 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.

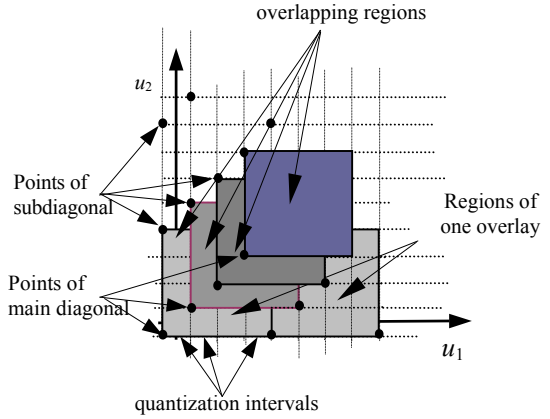


Fig. 2. The basis functions of a two-dimensional CMAC

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 2 (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 higher dimensional cases the number of basis function will not grow exponentially with the 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 modelling capability of the multidimensional CMAC, as reducing the number of basis functions the number of free parameters will also be reduced (Brown and Harris, 1994). To avoid this unwanted effect we can get help from the approach of support vector machines (SVMs) (Vapnik, 1995).

It should be mentioned that in higher-dimensional cases further complexity reduction is required. This reduction is achieved by applying a compressing new layer (Albus, 1975), which uses Hash coding. However, the effect of hashing can be neglected, when its features are selected properly (Ellison, 1991), so we will not deal with this effect.

3. A BRIEF SUMMARY OF SVMs AND LS-SVMs

Support Vector Machines (SVMs) were proposed recently by Vapnik (1995) and are based on Statistical Learning Theory (SLT) and Structural Risk Minimization (SRM) principle. SVMs can be constructed for linear or non-linear classification tasks as well as for linear or non-linear regressions. Here only the regression version will be summarized. An SVM is constructed using training data $\{\mathbf{x}_i, y_i\}_{i=1}^{P}$ similarly to the classical neural networks. The goal of an SVM for regression is to approximate a (non-linear) function (Smola *et al.*, 1998), where the quality of approximation is determined using a loss or cost function. The loss function is representing the cost of the deviation from the target output d_i for each $\mathbf{x}_i$ input. In most cases the ε–insensitive loss function (L_ε) is used, but one can use other (e.g. non–linear) loss functions too. The ε–insensitive loss function is:

$$L_\varepsilon(y) = \begin{cases} 0 & for\ |f(\mathbf{x}) - d| < \varepsilon \\ |f(\mathbf{x}) - d| - \varepsilon & otherwise \end{cases} \qquad (4)$$

Using this loss function the approximation errors smaller than ε are ignored, while the larger ones are punished in a linear way.

Our goal is to give an $y = f(\mathbf{x})$ function, which represents the dependence of the output y on the input $\mathbf{x}$. In the SVM approach first the input vectors are projected into a higher dimensional feature space, using a set of non-linear functions $\boldsymbol{\varphi}(\mathbf{x}) : \Re^N \to \Re^M$. The dimensionality ($M$) of the new feature space is not defined, it follows from the method (it can even be infinite). The function is estimated by projecting the input data to the higher dimensional feature space as follows:

$$y = \sum_{j=0}^{M} w_j \varphi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) \qquad (5)$$

where

$\mathbf{w} = [w_0, w_1, ..., w_M]^T$ and $\boldsymbol{\varphi} = [\varphi_0(\mathbf{x}), \varphi_1(\mathbf{x}), ..., \varphi_M(\mathbf{x})]^T$

It is assumed that $\varphi_0(\mathbf{x}) \equiv 1$, therefore w_0 represents a bias term b. The goal of approximation is to find the parameter vector $\mathbf{w}$ in the representation of Eq. (5). In this optimization problem we have constraints given in Eq. (6) (it comes from the ε–insensitive loss function $L_\varepsilon(f(\mathbf{x}_i, y_i))$ and a further constraint of $\|\mathbf{w}\|^2 \le c_0$ to keep $\mathbf{w}$ as short as possible (c_0 is a constant). To deal with training points outside the ε boundary, the $\{\xi_i\}_{i=1}^{P}$ and $\{\xi_i'\}_{i=1}^{P}$ slack variables are introduced:

$$\begin{aligned} d_i - \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) &\le \varepsilon + \xi_i, \\ \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) - d_i &\le \varepsilon + \xi_i', \\ \xi_i &\ge 0, \\ \xi_i' &\ge 0, \end{aligned} \qquad i = 1,2,...,P \qquad (6)$$

The slack variables are introduced to describe the penalty for the training points lying outside the ε boundary. The measure of this cost is determined by the loss function. This is solved by minimizing the following objective function:

$$F(\mathbf{w}, \xi, \xi') = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \rho\left(\sum_{i=1}^{P}(\xi_i + \xi_i')\right) \qquad (7)$$

The first term stands for the minimization of $\|\mathbf{w}\|$, while the ρ constant is the trade–off parameter between this and the minimization of training data errors. This constrained optimization can be defined as a Lagrangian function. This is often called as the primal problem:

$$J(\mathbf{w},\xi,\xi',\alpha,\alpha',\gamma,\gamma') = \rho\sum_{i=1}^{P}(\xi_i+\xi_i') + \frac{1}{2}\mathbf{w}^T\mathbf{w} - \\ -\sum_{i=1}^{P}\alpha_i\left[\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) - d_i + \varepsilon + \xi_i\right] - \\ -\sum_{i=1}^{P}\alpha_i'\left[\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) - d_i + \varepsilon + \xi_i'\right] - \sum_{i=1}^{P}(\gamma_i\xi_i + \gamma_i'\xi_i') \quad (8)$$

We have to minimize (8) according to **w** and b and maximize according to the Lagrange multipliers. The primal problem deals with convex cost function and linear constraints, therefore from this constrained optimization problem a dual problem can be constructed, which can be solved more easily. The solution can be obtained using quadratic programming (QP). To do this the Karush–Kuhn–Tucker (KKT) conditions are used (Vapnik, 1995). The dual problem is:

$$Q(\alpha,\alpha') = \sum_{i=1}^{P} d_i(\alpha_i - \alpha_i') - \varepsilon\sum_{i=1}^{P}(\alpha_i + \alpha') \\ -\frac{1}{2}\sum_{i=1}^{P}\sum_{j=1}^{P}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\mathbf{x}_i,\mathbf{x}_j) \quad (9)$$

With constraints:

$$\sum_{i=1}^{P}(\alpha_i - \alpha_i') = 0 \quad 0 \le \alpha_i', \alpha_i \le \rho, \quad i = 1,...,P \quad (10)$$

From the dual problem it can be seen that the solution does not depend directly on the $\boldsymbol{\varphi}$ non-linear function set. Instead a kernel function is used which is formed as: $K(\mathbf{x}_i,\mathbf{x}_j) = \boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}_j)$ The result of the dual problem is the sets of the Lagrange multipliers α_i and α_i'.

Finally the response of the SVM can be determined as the weighted sum of the values of the kernel function.

$$y = \sum_{i=1}^{P}(\alpha_i - \alpha_i')K(\mathbf{x},\mathbf{x}_i) \quad (11)$$

The nonzero multipliers ($(\alpha_i - \alpha_i')$, i=1,...,P) mark their corresponding input data points as support vectors. It can be seen that the response depends only on the support vectors and not on all training data. This property is an interesting and important feature of SVM solutions. It shows that the solution is obtained as a sparse approximation.

For constructing an SVM instead of choosing the non-linear functions, the kernel function should be selected. To be a kernel a function must satisfy the Mercer condition (Vapnik, 1995). The most popular kernel functions are the Gaussian functions. In this case the SVM corresponds to an RBF network where the centre vectors of the Gaussian basis functions are the support vectors. So the structure of a support vector machine and an RBF may be similar although their constructions are quite different.

The main drawback of SVM is its high computational burden because of the required quadratic programming. Recently a least square (LS) version of SVM was proposed by Suykens (2001). LS-SVM can also be applied for both classification and regression. LS-SVM applies quadratic cost function and equality constraints instead of the inequality ones given in Eq. (6). The optimisation problem of LS-SVM is formulated as follows:

$$F(\mathbf{w},\mathrm{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \gamma\frac{1}{2}\left(\sum_{i=1}^{P}e_i^2\right), \quad (12)$$

where

$$d_i = \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i. \quad (13)$$

From these equations one can construct the Lagrangian, which leads to the following overall solution:

$$\begin{bmatrix} 0 & \vec{\mathbf{1}}^T \\ \vec{\mathbf{1}} & \boldsymbol{\Omega} + C^{-1}\mathbf{I} \end{bmatrix}\begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix} \quad (14)$$

Here $\mathbf{d}^T = [d_0, d_1, ..., d_P]$, $\boldsymbol{\alpha}^T = [\alpha_0, \alpha_1, .., \alpha_P]$, $\vec{\mathbf{1}}^{\mathbf{T}} = [1,...,1]$ and $\Omega_{i,j} = K(x_i, x_j)$.

The response of the "network" can be obtained as

$$f(\mathbf{x}) = \sum_{i=1}^{P}\alpha_i K(\mathbf{x},\mathbf{x}_i) + b \quad (15)$$

It can be seen that the response for a given input can be obtained similarly to (11), however, here instead of quadratic programming only a matrix inversion is required to determine the Lagrange multipliers. It can also be seen from (12) that the real problem is to find such a weight vector that minimizes the cost function. An important difference between Vapnik's SVM and LS-SVM is that the latter solution is not sparse; all training points are used for getting the solution. To get sparse solution, however, many different approaches were proposed for example (Suykens *et al.*, 2000), (Valyon and Horváth, 2002).

It can also be recognised that LS-SVM and the method of ridge regression (Saunders *et al.*, 1998) are almost equivalent, as the function to be minimized in ridge regression is:

$$F(\mathbf{w},\mathrm{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \gamma\frac{1}{2}\left(\sum_{i=1}^{P}e_i^2\right) \quad (16)$$

with the constraint of

$$d_i = \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + e_i \quad (17)$$

This means that the approximation error will be:

$$e_i = d_i - \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) \quad (18)$$

Again the Lagrangian can be constructed which leads to the solution:

$$\boldsymbol{\alpha} = \left(\boldsymbol{\Omega} + \frac{1}{\gamma}\mathbf{I}\right)^{-1}\mathbf{d} \quad (19)$$

and

$$f(\mathbf{x}) = \sum_{i=1}^{P}\alpha_i K(\mathbf{x},\mathbf{x}_i) \quad (20)$$

The only difference between LS-SVM and ridge regression is that in ridge regression bias is not used. Further it can be shown easily that the solution of ridge regression and a regularized LS solution are equivalent. The classical LS solution can be obtained if (18) is used in (16). In this case the optimal weight vector is obtained from the minimization of the regularized LS problem.

$$\mathbf{w} = \left(\mathbf{\Omega} + \frac{1}{\gamma} \mathbf{I} \right)^{-1} \sum_{i=1}^{P} d_i \boldsymbol{\varphi}(\mathbf{x}_i) \qquad (21)$$

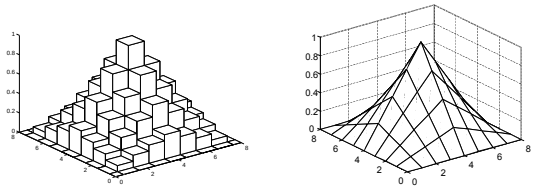
where $\Omega_{i,j} = K(x_i, x_j)$ as before.

The basic difference between the two approaches is that in the LS solution the weight vector **w,** which is used in the *M*-dimensional feature space, is obtained directly. Using the Lagrange method the direct results are the Lagrange multipliers. The weight vector **w** can be obtained indirectly from $\boldsymbol{\alpha}$. An important feature is that while the dimension of the feature space may be arbitrarily large, even infinite, the dimension of $\boldsymbol{\alpha}$ is *P*, which is the number of the training samples. A more important feature is that using the Lagrange approach the solution (20) is expressed as a linear combination of different position kernel functions, where the centre parameters of the kernel functions are determined by the training points. Using the kernel representation to get the response of the network we do not need to determine the representation in the feature space. The two solutions can be called primal and dual solutions. Primal solution can be preferred when the dimension of the feature space is not too high and where there are no difficulties with the implementation of the $\boldsymbol{\varphi} = [\varphi_0(\mathbf{x}), \varphi_1(\mathbf{x}), ..., \varphi_M(\mathbf{x})]^T$ non-linear functions. On the other hand ridge regression or LS-SVM approach should be preferred if the dimension of the feature space is huge or if the sparse representation has special advantages.

4. CMAC AS A SUPPORT VECTOR MACHINE

The two different approaches discussed previously can be applied in the case of the CMAC. The classical binary CMAC uses rectangular basis functions (first-order B-splines) with fixed positions, which map the input data into the feature space. The association vector corresponds to the feature space representation. In one-dimensional case the length of the association vector is rather limited, so CMAC can be implemented efficiently in the feature space. A one-dimensional binary CMAC can also be interpreted as an LS-SVM (or more exactly as a ridge regression solution, because the classical CMAC does not use bias term), where the kernel function is obtained from the fixed-position first-order B-splines: the kernel functions will be second-order B-splines where the centre parameters are the input data points. If the support of the rectangular functions is *C* measured in quantums of the input data, the support of a kernel function is 2*C*. In one-dimensional case the kernel representation of the CMAC has no special advantages. However, in a multidimensional case using the primal representation we have to reduce the number of rectangular basis functions, the dimension of the feature space. Without this reduction the complexity of the multidimensional CMAC would increase exponentially with the input dimension. The reduction of the dimension of the feature space in a classical *N*-dimensional CMAC is achieved by using only *C* overlays of rectangular basis functions instead of C^N ones (and we apply Hash coding) as it was discussed in Section 2. But this reduction is the real reason of the limited modelling capability of the multidimensional CMAC. If we used C^N overlays we would get a multidimensional CMAC with the same modelling capability as a one-dimensional network. In the primal space a CMAC with C^N overlays cannot be implemented because of its high complexity. However, in the dual space we use the kernel function, and the complexity will not increase even if the feature (primal) space is a very-large dimensional one.

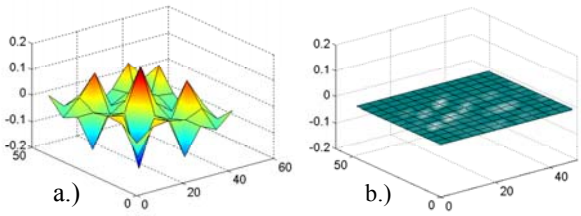
The second-order B-spline kernel function in two-dimensional case is shown in Figure 3a. This is the discretized version of the continuous second-order B-spline (Figure 3b.) according to the quantized input space of the CMAC.



a.) b.)

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

The new interpretations have some important consequences. First of all there will be no significant difference between the one-dimensional and the multidimensional cases. The modelling capability of the multidimensional version will be the same as that of the one-dimensional CMAC without getting so complex network architecture that cannot be implemented. In multidimensional cases the SVM (ridge regression) representation will not be equivalent to the Albus CMAC, as it corresponds to a CMAC with exponentially increasing number of overlays. Figure 4. shows this difference in a simple example. The approximation error in the training points of the 2D *sinc* function is shown for the Albus CMAC in Figure 4a. and for the kernel-based version in Figure 4b. It can be seen that in the latter case all training points can be represented exactly, while the original version are not able to learn exactly all training data.



a.) b.)

Fig. 4. The training error of the 2D CMAC a.) and the kernel network b.) for the 2D *sinc* function.

The kernel representation and the ridge regression version can be constructed not only for binary CMAC but for higher-order networks too. In higher-order CMACs instead of the rectangular basis

functions higher-order B-splines are used. Using a k^{th}-order B-spline in the original CMAC (in the primal space), the corresponding SVM versions will use $(2k+1)^{th}$ order B-splines as SVM kernels.

The improved modelling capability has a price. The kernel representation needs higher-order functions even in the case that corresponds to the binary CMAC. Therefore the network cannot be implemented without using multipliers. This drawback, however, at least partly can be compensated using some effective multiplier architecture (see e.g. Szabó *et al.*, 2000) and an efficient construction where all advantageous of this multiplier structure can be utilised.

5. CONCLUSION

In summary we can conclude that two interpretations of the CMAC networks can be used. The first – the original one – applies rectangular basis functions, where the basis functions implement a mapping from input space into a "feature" space, and the solution is obtained as a linear mapping from this feature space into the output. The SVM approach applies piecewise linear B-spline basis functions as kernel functions and the solution is obtained as a mapping from the kernel space into the output space. The first version is better (the network is rather simple, its training is fast, etc) in one-dimensional case, but the second one can be preferred in multidimensional cases as - although it may need more complex training algorithm - it has better modelling capability.

REFERENCES

Albus, J.S. (1975) "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)", *Transaction of the ASME*, Sep. 1975. pp. 220-227.

Brown, M. - Harris, C.J. - Parks, P (1993). "The Interpolation Capability of the Binary CMAC", *Neural Networks*, Vol. 6, pp. 429-440.

Brown, M. and Harris, C.J. (1994) "Neurofuzzy Adaptive Modeling and Control" Prentice Hall, New York, 1994.

Chiang Ching-Tsan and Lin Chun-Sin (1996): CMAC with General Basis Functions. *Neural Networks*, Vol. 9, pp. 1199-1211.

Commuri S., Lewis F.L. and Jagannathan S. (1995): Discrete-time CMAC Neural Networks for Control Applications. *Proc. of the 34th Conference on Decision & Control*, New Orleans, LA, Vol. 2. pp. 2420-2426.

Ellison, D. (1991) "On the Convergence of the Multidimensional Albus Perceptron", *The International Journal of Robotics Research*, Vol. 10, pp. 338-357.

Horváth, G. and Deák, F. (1993) "Hardware Implementation of Neural Networks Using FPGA Elements" *Proc. of The International Conference on Signal Processing Application and Technology*, Santa Clara Vol. II, pp. 60-65.

Ker, J.S. - Kuo, Y.H. - Liu, B.D. (1995) "Hardware Realization of Higher-order CMAC Model for Color Calibration", *Proceedings, of the IEEE International Conference on Neural Networks*, Perth, Vol. 4, pp. 1656-1661.

Lane, S.H. - Handelman, D.A. and Gelfand, J.J (1992) "Theory and Development of Higher-Order CMAC Neural Networks", *IEEE Control Systems*, Apr. 1992. pp. 23-30.

Miller, T.W. III. Glanz, F.H. and Kraft, L.G. (1990) "CMAC: An Associative Neural Network Alternative to Backpropagation" *Proceedings of the IEEE*, Vol. 78, pp. 1561-1567.

Miller, W.T. - Box, B.A. and Whitney E.C. (1991) "Design and Implementation of a High Speed CMAC Neural Network Using Programmable CMOS Logic Cell Arrays", *ANIPS 3,* pp. 1022-1027.

Saunders, C.- Gammerman, A. and Vovk, V. (1998), Ridge Regression Learning Algorithm in Dual Variables. Machine Learning, *Proc of the Fifteenth International Conference on Machine Learning*, pp. 515-521.

Smola, A.J. and Schölkopf, B. (1998) "A Tutorial on Support Vector Regression", *NeuroCOLT2 Technical Report Series NC2–TR–1998–030*, Oct., 1998.

Suykens, J.A.K., Lukas, L. and Vandewalle J. (2000) "Sparse approximation using least squares support vector machines" *Proc. of the IEEE International Symposium on Circuits and Systems ISCAS' 2000*. Vol. II, pp. 757-760.

Suykens, J.A.K. (2001) "Nonlinear Modeling and Support Vector Machines", *IEEE Instrumentation and Measurement Technology Conference*, Budapest, Vol. I, pp. 287-294.

Szabó, T. and Horváth, G. (1999) CMAC and its Extensions for Efficient System Modelling" *Int. Journal of Applied Mathematics and Computations*, Vol. 9, pp. 571-598.

Szabó, T. and Horváth, G. (2002) "CMAC Neural Network with Improved Generalization Capability for System Modelling" *Proc. of the IEEE Conference on Instrumentation and Measurement*, Anchorage, AK. Vol. II, pp. 1603-1608.

Szabó, T., Antoni, L., Horváth, G. and Fehér, B. (2000) "An efficient implementation for a matrix-vector multiplier structure," *in Proc. of IEEE International Joint Conference on Neural Networks, IJCNN 2000*, Vol. II, pp. 49–54.

Thompson D.E. and Kwon S. (1995): Neighbourhood Sequential and Random Training Techniques for CMAC. *IEEE Trans. on Neural Networks*, Vol. 6, pp. 196-202.

Valyon, J. and Horváth, G. (2002) "Reducing the Complexity and Network Size of LS–SVM Solutions" submitted to *IEEE Trans. on Neural Networks*.

Vapnik, V. (1995) "The Nature of Statistical Learning Theory", Springer, New–York.