Kernel CMAC: an Efficient Neural Network for Classification and Regression

Gábor Horváth

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

Abstract: Kernel methods in learning machines have been developed in the last decade as new techniques for solving classification and regression tasks. Kernel methods have many advantages properties regarding their learning and generalization capabilities, but for getting the solution usually the computationally complex quadratic programming is required. To reduce computational complexity a lot of different versions have been developed. These different versions apply different kernel functions, utilize the training data in different ways or apply different criterion functions. This paper deals with a special kernel network, which is based on the CMAC neural network. Cerebellar Model Articulation Controller (CMAC) has some attractive features: fast learning capability and the possibility of efficient digital hardware implementation. Besides these attractive features the modelling and generalization capabilities of a CMAC are rather limited. The paper shows that kernel CMAC – an extended version of the classical CMAC network implemented in a kernel form – combines the advantages of both approaches. Its modelling and generalization capabilities are improved while the limited computational complexity is maintained. The paper shows the architecture of this network and presents the relation between the classical CMAC and the kernel networks. The operation of the proposed architecture is illustrated using some common benchmark problems.

Keywords: kernel networks, input-output system modelling, neural networks, CMAC, generalization error

1 Introduction

Kernel machines like Support Vector Machines (SVMs) [1], Least Squares SVMs (LS-SVMs) [2] and the method of ridge regression [3] have beed developed in the last decade and proved to be efficient new approaches for solving the learning problem from samples. Kernel machines can be applied for linear and nonlinear classification and function approximation, so they can be used for solving problems that can be solved successfully with classical neural networks too. The main feature of kernel methods is that they 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. Althogh the kernel space can be defined through the feature space, usually it is defined directly without fixing the nonlinear transformations of the feature space. This is a significant advantage, as using the kernel trick the complexity of the solution is greatly reduced: while the dimension of the feature space can be very large, even infinite, the dimension of the kernel space is upper bounded by the number of training samples.

Cerebellar Model Articulation Controller (CMAC) [4] – 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 [5]. The CMAC architecture was proposed by Albus in the middle of the seventies [4] and it is considered as a real alternative to MLP and other feed-forward neural networks [6]. Although the properties of CMAC were analysed mainly in the nineties (see eg. [7]-[9]), some interesting features were only recognised in the recent years. These results show that the attractive properties of the CMAC have a price: its modelling capability is inferior to that of an MLP. This is especially true for multivariate cases, as multivariate CMACs can learn to reproduce the training points exactly only if the training data come from a function belonging to the additive function set [7].

The modelling capability can be improved if the complexity of the network is increased. This more complex network was proposed in [8], 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. 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 shortly presented in [10] and a modified training algorithm was proposed for improving the generalization capability. This training algorithm is derived using a regularized loss function, where the regularization term has some weight-smoothing effect.

This paper presents a different interpretation of the CMAC networks and details why this interpretation can help to improve the quality of the network without increasing the complexity even in multidimensional cases. The paper shows that this new interpretation corresponds to a kernel machine with second order B-spline kernel functions. The kernel interpretation may suffer from the same poor generalization capability, however the weight-smoothing regularization can be applied for the kernel CMAC too. This means that using kernel CMAC both the modelling and the generalization capabilities can be improved significantly. Moreover it can be shown that similarly to the original CMAC the kernel versions can also be trained iteratively, which may be important in such applications where real-time on-line adaptation is required.

2 Kernel Machines

The goal of a kernel machine is to approximate a (nonlinear) function $y_d = f(\mathbf{u})$ using a training data set $\{\mathbf{u}(k), y_d(k)\}_{k=1}^{P}$. A kernel machine can be used for solving classification or regression problems. For classification the function to be approximated is $f \ : \ \Re^N \rightarrow \{\pm 1\}$, while for regression problems a continuous function $f \ : \ \Re^N \rightarrow \Re$ should be approximated. In the kernel machines first the **u** input vectors are projected into a higher dimensional feature space, using a set of nonlinear functions $\boldsymbol{\varphi}(\mathbf{u}) \ : \ \Re^N \rightarrow \Re^M$, then the output is obtained as a linear combination of the projected vectors:

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

where **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

$$y(\mathbf{u}) = \sum_{k=1}^{P} \alpha_k K(\mathbf{u}, \mathbf{u}(k)) + b \qquad (2)$$

where the kernel function is formed as

$$K(\mathbf{u}(k), \mathbf{u}(j)) = \boldsymbol{\varphi}^T(\mathbf{u}(k)) \boldsymbol{\varphi}(\mathbf{u}(j)) \qquad (3)$$

In (2) the α_k coefficients serve as the weight values 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 [1]. The main advantage of a kernel machine is that the kernel function can be defined directly without using the feature space representation. For this purpose the kernel function should fulfill some conditions [11]. Kernel machines can be constructed using constrained optimization, where first a criterion function and some constrainst are defined and where the solution is obtained using Lagrange a multiplicator approach.

Kernel machines have many different versions. The different versions apply different kernel functions or formulate the constrained optimization problem in different ways. Most often Gaussian kernels are used, but polynomial, spline, etc. kernels can also be applied [11]. The complexity of the solution depends on the form of the constraints. Using inequality constraints [1] quadratic programming is required to reach the solution. This approach was introduced by Vapnik and it results in the classical support vector machine (SVM) solution. A less complex solution is obtained if instead of the inequality constraints equality ones are applied. One approach of this version is when quadratic criterion function and equality constraints are used. It is called least squares support vector machine (LS-SVM) [2]. Ridge regression is similar to LS-SVM, although its derivation is slightly different from that of the LS-SVM [3]. In LS-SVM instead of quadratic programming the solution can be obtained using simple matrix inversion. To show the detailes of the kernel machines is beyond the scope of this paper. These detailes can be found in the recently published excellent books [11], [12] and papers.

3 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 $\mathbf{u} \in \Re^N$ into an association vector **a**. The second mapping calculates the output $y \in \Re$ of the network as a scalar product of the association vector **a** and the weight vector **w**:

$$y(\mathbf{u})=\mathbf{a}(\mathbf{u})^{T}\mathbf{w} \tag{4}$$

The association vectors are sparse binary vectors, which have only *C* active elements, *C* bits of the 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.

$$y(\mathbf{u}) = \sum_{i:a_i(\mathbf{u})=1} w_i \tag{5}$$

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 [4].

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 that 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. For input points far enough from each other - further then the neighbourhood determined by the parameter C – the association vectors should not have any common bits.

This mapping is responsible for the non-linear property and the generalization of the whole system. The first layer implements a special encoding of the quantized input data. This layer is fixed. The trainable elements, the weight values that 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, determines 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. If there are r_i discrete values for the i-th input dimension an N-dimensional CMAC needs $M = \prod_{i=1}^{N}(r_i + C - 1)$ weight values. 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 by using basis functions positioned only at the diagonals of the quantized input space. 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 sub-diagonal points. 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, it will be “only” $M = \left\lceil \frac{1}{C^{N-1}} \prod_{i=1}^{N}(r_i + C - 1) \right\rceil$. 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 multivariable CMACs, as reducing the number of basis functions the number of free parameters will also be reduced. Here modelling capability refers to the ability that a network can learn to reproduce exactly the training data: a network with this ability will have no modelling error.

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 are obtained from an additive function [7]. For more general cases there will be modelling error i.e. error at the training points. It should be mentioned that in multivariate cases even this reduced weight memory may be too large, so further complexity reduction may be required. This reduction is achieved by applying a new compressing layer [4], which uses hash-coding. Although hash-coding solves the complexity problem, it can result in collisions of the mapped weight and some unfavourable effects on the convergence of CMAC learning [13], [14]. As it will be seen later the proposed new interpretation solve the complexity problem without the application of hashing, so we will not deal with this effect.

Another way of avoiding the complexity problem is to decompose a multivariate problem into many one-dimensional ones, so instead of implementing a multidimensional CMAC it is better to implement many simple one-dimensional networks. The resulted hierarchical, tree-structured network - called MS_CMAC [15] - can be trained using time inversion technique [16]. MS_CMAC greatly reduces the complexity of the network, however there are some restrictions in its application as it can be applied only if the training points are positioned at regular grid-points. A further drawback is that most of the simple networks need training even in the recall phase increasing the recall time significantly.

A CMAC – as it has a linear output layer – can be trained by the LMS algorithm:

$$w_i\left(k+1\right)=w_i\left(k\right)+\mu\,\mathbf{a}\left(k\right)e\left(k\right), \tag{6}$$

where $e(k)=y_d(k)-y(k)=y_d(k)-\mathbf{w}^T\mathbf{a}(k)$ is the error at the *k*-th training step. Here $y_d(k)$ is the desired output for the *k*-th training point, $y(k)$ is the network output for the same input, $\mathbf{a}(k)$ = $\mathbf{a}(\mathbf{u}(\mathrm{k}))$ and μ is the learning rate. Training will minimize the quadratic error

$$\min_{\mathbf{w}} J=\frac{1}{2}\sum_{k=1}^{P}e(k)^2 \tag{7}$$

where *P* is the number of training points.

The solution of the training can also be written in a closed form

$$\mathbf{w}^*=\mathbf{A}^{\dagger}\mathbf{y}_d \tag{8}$$

where $\mathbf{A}^{\dagger}=\mathbf{A}^T\left(\mathbf{A}\mathbf{A}^T\right)^{-1}$ is the pseudo inverse of the association matrix formed from the association vectors $\mathbf{a}(i)=\mathbf{a}(\mathbf{u}(i))$, and $\mathbf{y}_d{}^T=[y_d(1)\;\; y_d(2)\;\; \ldots\;\; y_d(P)]$ is the output vector formed from the desired values of all training data. The response of the trained network for a given input $\mathbf{u}$ can be determined using the solution weight vector:

$$y(\mathbf{u})=\mathbf{a}^T(\mathbf{u})\mathbf{w}^*=\mathbf{a}^T(\mathbf{u})\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{y}_d \quad . \tag{9}$$

4 Kernel CMAC

4.1 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 non-linear functions that map the input data points into the feature space, are the rectangular basis functions. The binary basis functions can be regarded as first-order B-spline functions of fixed positions.

To get the kernel representation of the CMAC we should apply (3) for the binary 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 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 interpretation can be applied for higher-order CMACs too [8] with higher order basis functions (*k*-th order B-splines with support of *C*). In these cases CMACs correspond to kernel machines with properly chosen higher–order (2*k*-th order) B-spline kernels.

Kernel machines can be derived through constrained optimisation. The different versions of kernel machines apply different loss functions. Vapnik's SVM for regression applies ε-insensitive loss function [1], while LS-SVM can be obtained if quadratic loss function is used [2]. 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 response of a trained network for a given input can be obtained by (9). 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.

$$\min_{\mathbf{w}} J(\mathbf{w},\mathbf{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{k=1}^{P} e(k)^2 \qquad (10)$$

such that $y_d(k) = \mathbf{w}^T \mathbf{a}(k) + e(k)$. Here there is no bias term, as in the classical CMAC bias term is not used. The problem in this form can be solved by constructing the Lagrangian

$$L(\mathbf{w},\mathbf{e},\alpha) = J(\mathbf{w},\mathbf{e}) - \sum_{k=1}^{P} \alpha_k \left(\mathbf{w}^T \mathbf{a}(k) + e(k) - y_d(k) \right) \tag{11}$$

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

$$\begin{cases} \dfrac{\partial L(\mathbf{w},\mathbf{e},\alpha)}{\partial \mathbf{w}} = \mathbf{0} \;\rightarrow\; \mathbf{w} = \sum_{k=1}^{P} \alpha_k \mathbf{a}(k) & \\ \dfrac{\partial L(\mathbf{w},\mathbf{e},\alpha)}{\partial e(k)} = 0 \;\rightarrow\; \alpha_k = \gamma e(k) & k = 1,...,P \\ \dfrac{\partial L(\mathbf{w},\mathbf{e},\alpha)}{\partial \alpha_k} = 0 \;\rightarrow\; \mathbf{w}^T \mathbf{a}(\mathbf{u}(k)) + e(k) - y_d(k) = 0 & k = 1,...,P \end{cases} \tag{12}$$

Using the results of (12) in (11) the Lagrange multipliers can be obtained as a solution of the following linear system

$$\left[\mathbf{K} + \frac{1}{\gamma}\mathbf{I} \right] \boldsymbol{\alpha} = \mathbf{y}_d \tag{13}$$

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

$$\begin{aligned} y(\mathbf{u}) &= \mathbf{a}^T(\mathbf{u})\mathbf{w} = \mathbf{a}^T(\mathbf{u}) \sum_{k=1}^{P} \alpha_k \mathbf{a}(k) = \sum_{i=1}^{P} \alpha_k K(\mathbf{u}, \mathbf{u}(k)) = \mathbf{K}^T(\mathbf{u})\boldsymbol{\alpha} \\ &= \mathbf{a}^T(\mathbf{u})\mathbf{A}^T \left[\mathbf{K} + \frac{1}{\gamma}\mathbf{I} \right]^{-1} \mathbf{y}_d = \mathbf{a}^T(\mathbf{u})\mathbf{A}^T \left[\mathbf{A}\mathbf{A}^T + \frac{1}{\gamma}\mathbf{I} \right]^{-1} \mathbf{y}_d \end{aligned} \tag{14}$$

The obtained kernel machine is an LS-SVM or more exactly a ridge regression solution [3], because of the lack of the bias term. Comparing (9) and (14), it can be seen that the only difference between the classical CMAC and the ridge regression solution is the term $(1/\gamma)\mathbf{I}$, which comes from the modified loss function of (10). However, if the matrix $\mathbf{A}\mathbf{A}^T$ 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 $(\mathbf{A}\mathbf{A}^T)^{-1}$ the regularized inverse $(\mathbf{A}\mathbf{A}^T + \eta\mathbf{I})^{-1}$ is computed, where η is the regularization coefficient. In this case the two forms are equivalent.

4.2 Kernel CMAC with Weight-Smoothing

This kernel representation improves the modelling 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. In the derivation of the kernel machines regularization and the Lagrange multiplier approach are applied. To get a CMAC with better generalization capability a further regularization term can be applied. Smoothing regularization can be obtained if a new term is added to the loss function of (10). The modified optimization problem can be formulated as follows:

$$\min_{\mathbf{w}} J(\mathbf{w},\mathbf{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{k=1}^{P} e_k^2 + \frac{\lambda}{2}\sum_{k=1}^{P}\sum_{i}\left(\frac{y_{d_k}}{C} - w_k(i)\right)^2 \tag{15}$$

$w_k(i)$ 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), we obtain the Lagrangian

$$L(\mathbf{w},\mathbf{e},\alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{k=1}^{P} e_k^2 + \frac{\lambda}{2}\sum_{k=1}^{P}\sum_{i}\left(\frac{y_{d_k}}{C} - w_k(i)\right)^2 - \sum_{k=1}^{P}\alpha_k\left(\mathbf{w}^T\mathbf{a}_k + e_k - y_{d_k}\right) \tag{16}$$

The Lagrange multipliers can be obtained again as a solution of a linear system.

$$\boldsymbol{\alpha} = \left(\mathbf{K}_{\mathbf{D}} + \frac{1}{\gamma}\mathbf{I}\right)^{-1}\left(\mathbf{I} - \frac{\lambda}{C}\mathbf{K}_{\mathbf{D}}\right)\mathbf{y}_d \tag{17}$$

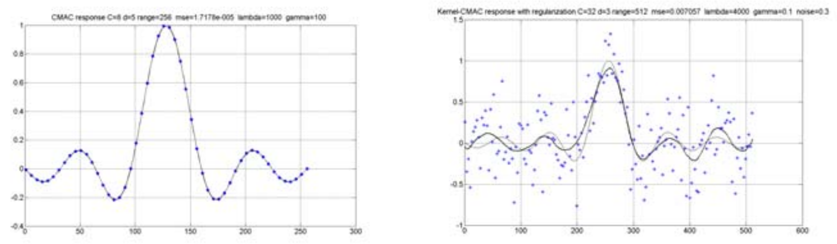
where $\mathbf{K}_{\mathbf{D}} = \mathbf{A}(\mathbf{I}+\lambda\mathbf{D})^{-1}\mathbf{A}^T$ and $\mathbf{D} = \sum_{k=1}^{P} diag(\mathbf{a}_k)$.

The response of the network becomes

$$y(\mathbf{u}) = \mathbf{a}^T(\mathbf{u})(\mathbf{I}+\lambda\mathbf{D})^{-1}\mathbf{A}^T\left[\boldsymbol{\alpha} + \frac{\lambda}{C}\mathbf{y}_d\right]. \tag{18}$$

5 Illustrative Experimental Results

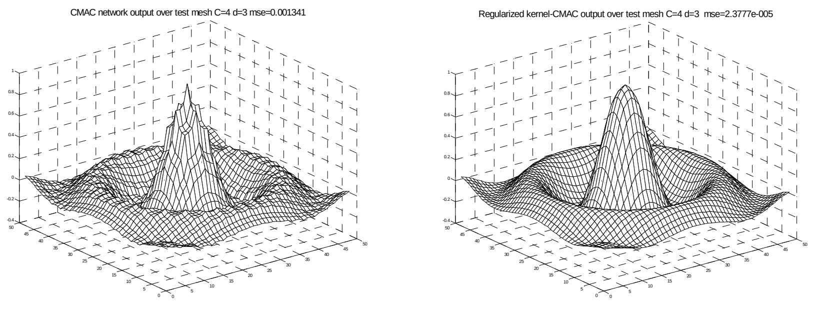
The different kernel versions of the CMAC network were validated by extensive experiments. Here only the results for the simple classification and regression benchmark problems are presented. The function approximation capability of the kernel CMAC is illustrated using the 1D (Figure 1) and 2D (Figure 2) *sinc* functions. For classification the two spiral problem (Figure 3) is solved. This is a benchmark task, which is rather difficult for a classical MLP. These experiments show that the response of the reguralized kernel CMAC is smoother than the response of the classical binary CMAC.



(a) (b)

Figure 1

The response of the kernel CMAC with weight-smoothing regularization using noiseless (a), and noisy (b) training data. C =8, λ=10^3



(a) (b)

Figure 2

The response of the kernel CMAC without (a), and with weight-smoothing regularization. C =32, λ=10^3.

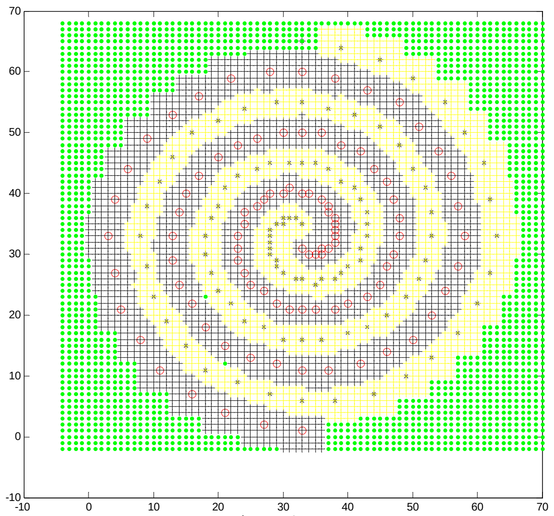


Figure 3

The solution of the two-spiral classification problem using kernel CMAC. C=4

Because of the finite support kernel functions local approximation and relatively low computational complexity are the additional advantages of kernel CMAC. Using this solution the large generalization error can be reduced significantly, so the regularized kernel CMAC is a real alternative of the popular neural network architectures like MLP and RBF even for multivariate cases.

Conclusions

In this paper it was shown that a CMAC network can be interpreted as a kernel machine with B-spline kernel function. This kernel interpretation makes it possible to increase the number of binary basis functions – the number of overlays, as in kernel interpretation the network complexity is upper bounded by the number of training samples, even if the number of binary basis functions of the original network is extremely large. The consequence of the increased number of basis function is that this version will have better modelling capability, and applying a special weight smoothing regularization the generalization capability can also be improved. Kernel CMAC can be applied successfully for both regression and classification problems even when high dimensional input vectors are used. The possibility of adaptive training ensures that the main advantages of the classical CMAC (adaptive operation, fast training, simple digital hardware implementation) can be maintained, although the multiplierless structure is lost.

References

[1] V. Vapnik: "Statistical Learning Theory", Wiley, New York, 1995

[2] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, B. and J. Vandewalle: "Least Squares Support Vector Machines", World Scientific, Singapore, 2002

[3] C. Saunders, A. Gammerman and V. Vovk: "Ridge Regression Learning Algorithm in Dual Variables. Machine Learning", *Proc. of the Fifteenth Int. Conf. on Machine Learning*, pp. 515-521, 1998

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

[5] J. S. Ker, Y. H. Kuo, R. C. Wen and B. D. Liu: "Hardware Implementation of CMAC Neural Network with Reduced Storage Requirement", *IEEE Trans. on Neural Networks*, Vol. 8, pp. 1545-1556, 1997

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

[7] M. Brown, C. J. Harris and P. C. Parks, "The Interpolation Capabilities of the Binary Cmac", *Neural Networks,* Vol. 6, No. 3, pp. 429-440, 1993

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

[9] C. T. Chiang and C. S. Lin: ‘‘Learning Convergence of CMAC Technique’’ *IEEE Trans. on Neural Networks, Vol. 8.* No. 6, pp. 1281-1292, 1996

[10] T. Szabó and G. Horváth: "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

[11] B. Schölkopf and A. Smola: “Learning with Kernels. Support Vector Machines, Regularization, Optimization and Beyond” The MIT Press, Cambridge, MA, 2002

[12] B. Schölkopf, C. J. C Burges and A. J. Smola: “Advances in Kernel Methods. Support Vector Learning” The MIT Press, Cambridge, MA, 1999

[13] L. Zhong, Z. Zhongming and Z. Chongguang: “The Unfavorable Effects of Hash Coding on CMAC Convergence and Compensatory Measure” *IEEE International Conference on Intelligent Processing Systems,* Beijing, China, pp. 419-422, 1997

[14] Z.-Q. Wang, J. L. Schiano and M. Ginsberg: “Hash Coding in CMAC Neural Networks” *Proc. of the IEEE International Conference on Neural Networks,* Washington, USA, Vol. 3, pp. 1698-1703, 1996

[15] J. C. Jan and S. L. Hung: High-Order MS_CMAC Neural Network, *IEEE Trans. on Neural Networks*, Vol. 12, No. 3, 2001, pp. 598-603

[16] J. S. Albus: "Data Storage in the Cerebellar Model Articulation Controller", *J. Dyn Systems, Measurement Contr.* Vol. 97, No. 3, pp. 228-233, 1975