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 problems. Kernel methods have many advantageous 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 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 may be rather limited. The paper shows that kernel CMAC – an extended version of the classical CMAC network implemented in a kernel form – improves that properties of the classical version significantly. Both the modelling and the 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 basic idea of kernel machines is that they apply two consequtive mappings. The first one maps the points of the input space (the input data) into an intermediate space called feature space. The goal of this mapping is to transform the original nonlinear problem into a linear one. More exactly, if a problem in the original representation can only be solved by nonlinear approaches, its transformed version in the feature space can be solved using linear methods (classification or regression). The theoretical background of applying nonlinear mapping to transform a nonlinear problem into a linear one goes back to Cover's theorem on the separability of patterns [4]. Based on this theorem it can be stated that it is more likely to represent a classification problem in a linearly separable way in a high-dimensional space than in a low-dimensional one.

Many neural network architectures apply this idea when first the input vectors are transformed into a higher-dimensional feature space, then in the feature space a linear solution is derived. These networks has two layers: the first one is responsible for the nonlinear dimension-increasing mapping and the second is a simple layer composed of linear neurons. Such networks are the popular radial basis function (RBF) networks, and all other basis function networks, but Cerebellar Model Articulation Controller (CMAC) network can also be interpreted in this way. The drawback of this approach is that in many cases the dimension of the feature space may be extremely large – even infinite. So to look for a solution in the high-dimensional feature space is impractical or in many cases it is practically impossible.

Kernel machines solve the dimensionality-problem by applying a trick, which is called kernel trick. They also apply nonlinear mapping from input space into feature space, however, they do not look for the solution in the feature space, instead the solution is obtained in the kernel space, which is defined easily. The significance of using the kernel trick is that the complexity of the solution is greatly reduced: the dimension of the kernel space is upper bounded by the number of training samples independently of the dimension of the feature space. The mappings of a kernel machine for a classification problem is shown in Fig. 1.

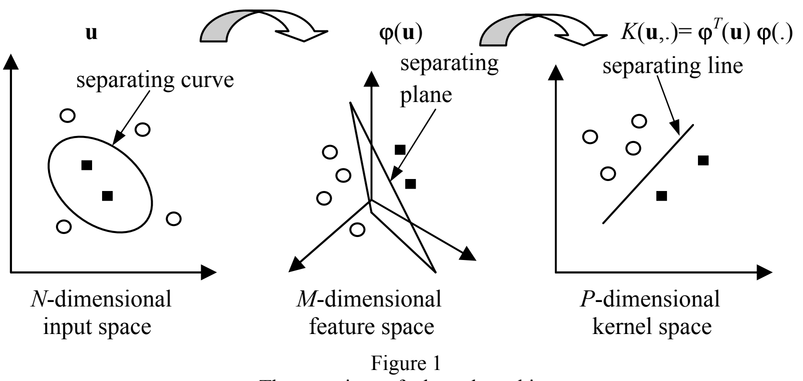

$\mathbf{u}$
$\varphi(\mathbf{u})$
$K(\mathbf{u},.)= \varphi^T(\mathbf{u})\, \varphi(.)$
separating curve
separating plane
separating line
N-dimensional input space
M-dimensional feature space
P-dimensional kernel space

Figure 1
The mappings of a kernel machine

As it can be seen, in the input space only nonlinear separation is possible. In the feature space – where usually $M > N$ – the transformed points are linearly separable. The linear separability is maintained in the kernel space too, while the dimension of the kernel space is upper bounded by P, the number of available data points.

Cerebellar Model Articulation Controller (CMAC) [5] – 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 [6]. The CMAC architecture was proposed by Albus in the middle of the seventies [5] and it is considered as a real alternative to MLP and other feed-forward neural networks [7]. Although the properties of a CMAC were analysed mainly in the nineties (see eg. [8]-[11]), some interesting features were only recognized 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 [8].

The modelling capability can be improved if the complexity of the network is increased. This more complex network was proposed in [9], 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 [11] and a modified training algorithm was proposed for improving the generalization capability. This training algorithm is derived using a regularized [12] 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.

The paper is organized as it follows. Section 2 summarizes some important features of kernel machines. Section 3 gives a short introduction to CMAC networks. This section presents the drawbacks of the classical CMAC. Section 4 shows how a CMAC can be interpreted as a kernel machine and it derives the main results related to the regularized version. Section 5 gives some illustrative examples to show how the proposed network can improve the capability of the network.

2 Kernel Machines

The goal of a kernel machine is to approximate a (nonlinear) function $y_d = f(\mathbf{u})$ ($\mathbf{u} \in \Re^N$, $y_d \in \Re$) using a training data set $\{\mathbf{u}(k), y_d(k)\}_{k=1}^P$. A kernel machine can be used to solve 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 [1]:

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

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

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

where the kernel function is formed as

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

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 [13]. 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 a Lagrange multiplicator approach.

Kernel machines have many different versions. These 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 [13]. The complexity of the solution depends on the form of the constraints. Using inequality constraints quadratic programming is required to reach the solution [1]. This approach was introduced by Boser, Guyon and Vapnik [14] 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]. Both in ridge regression and 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 and papers. See e.g. [13], [15], [16], [17].

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**:

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

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 [5].

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, and the value of C 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 resulted network will be called full-overlay CMAC – , 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 (see Fig. 2).

The shaded regions in Fig. 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.

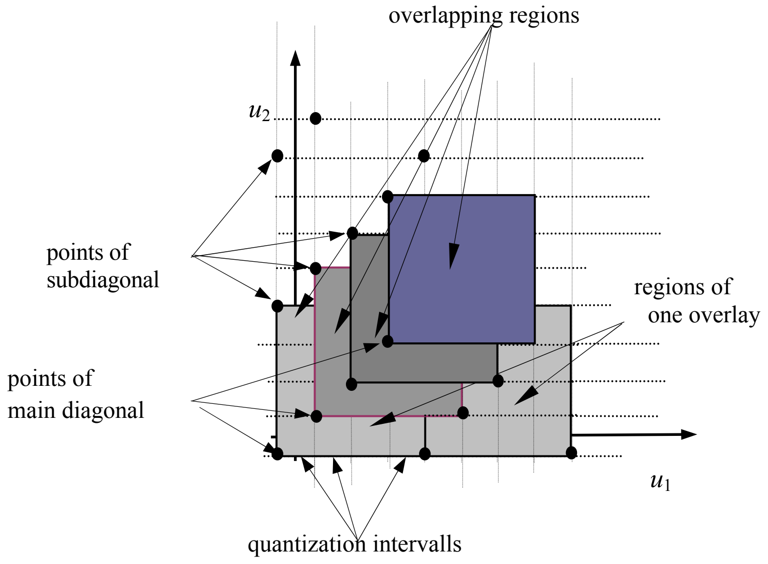


Figure 2
The basis functions of a two variable CMAC

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 [8]. 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. As an example consider a tendimensional binary CMAC where all input components are quantized into 10 bits. In this case the length of the association vector would be approximately 2^{55}. Although this is a greatly reduced value compared to 2^{100}, which is the weight memory size of the full-overlay version, this value is still prohibitively large.

Further 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 weights, and some unfavourable effects on the convergence of CMAC learning [18], [19]. As it will be seen later the proposed new interpretation solves the complexity problem without the application of hash-coding, 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 [20] – can be trained using time inversion technique [21]. 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:

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

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 $\mathbf{y}_d{}^T=[y_d(1)\;\; y_d(2)\;\; \dots\;\; y_d(P)]$ 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 CMACs 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. Fig. 3 shows a first-order B-spline basis function (a), and the corresponding kernel function – a second order B-spline – (b).

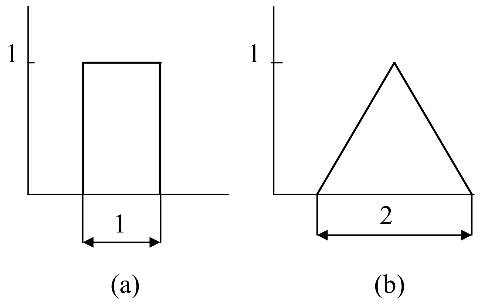
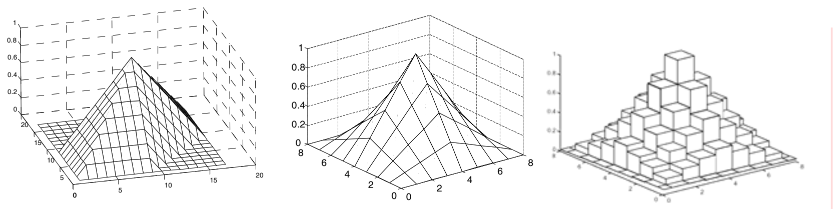


Figure 3
First-order (a) and second-order (b) B-spline function

In kernel representation the number of kernel functions does not depend on the number of basis functions in the feature space; because of the kernel trick it is always upper bounded by the number of training points *P*. This means that there are no reason to reduce the number of basis functions in the feature space: we can apply full-overlay CMAC. The only consequence is that the kernel function will be different and the modelling capability of the resulted network will be improved as a full-overlay CMAC can learn all training points exactly even in multivariate cases. Fig. 4 shows the 2D kernel function for classical CMAC (a), and for a full-overlay CMAC (b). This latter can be obtained as a tensor product of the two one-dimensional second order B-spline functions. Fig. 4(c) shows the quantized version of the 2D full-overlay kernel function. As in a CMAC quantized input data are used, this function is used as a kernel function in the proposed kernel CMAC.

The kernel interpretation can be extended to higher-order CMACs too [9] where higher order basis functions (k-th order B-splines with support of C) are applied. In these cases CMACs correspond to kernel machines with properly chosen higher-order ($2k$-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 can be considered as a special LS-SVM.



(a) (b) (c)

Figure 4

2D kernel functions classical CMAC (a), full-overlay CMAC (b), and its spatially quantized version (c)

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 do construct an LS-SVM network with similar feature space representation. For LS-SVM regression we seek for the solution of the following constrained optimisation.

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

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

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

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 resulted 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 *i*th 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.

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

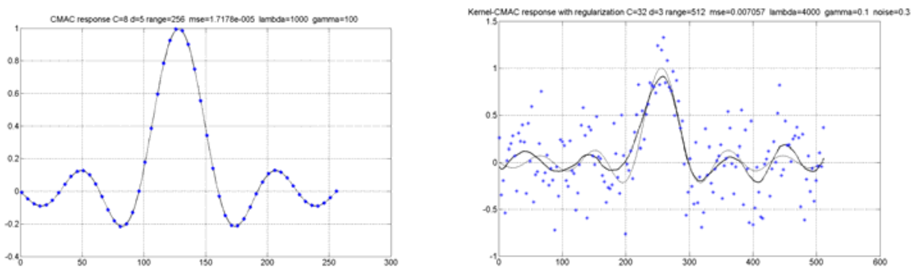
where $\mathbf{K}_{\mathbf{D}} = \mathbf{A}\left(\mathbf{I}+\lambda\mathbf{D}\right)^{-1}\mathbf{A}^T$ and $\mathbf{D} = \sum_{k=1}^{P} diag(\mathbf{a}_k)$.

The response of the network becomes

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

5 Illustrative Experimental Results

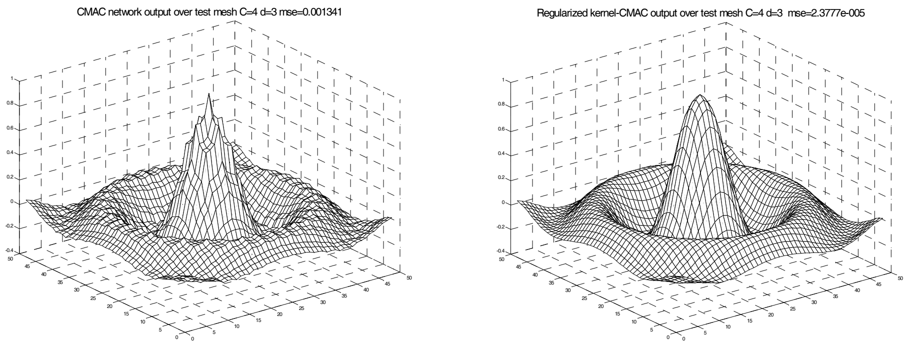
The different kernel versions of the CMAC network were validated by extensive experiments. Here only the results for some simple classification and regression benchmark problems are presented. The function approximation capability of the kernel CMAC is illustrated using the 1D (Fig. 5) and 2D (Fig. 6) *sinc* functions. For classification the two spiral problem (Fig. 7) 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 much better than the response of the classical binary CMAC, the approximation or the classification error is significantly reduced.



(a) (b)

Figure 5

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 6

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

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.

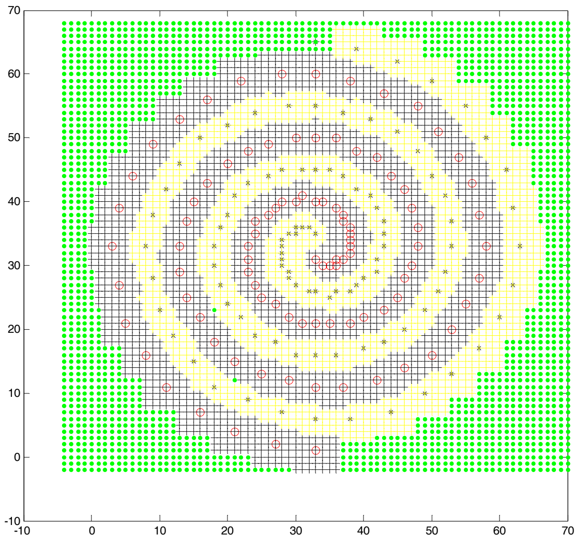


Figure 7

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

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, 1998

[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] T. M. Cover: “Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition” *IEEE Trans. on Electronic Computers*, EC-14, pp. 326-334, 1965

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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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

[11] 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

[12] A. N. Tikhonov and V. Y. Arsenin: “Solution of Ill-posed Problems”, Washington, DC, W. H. Winston, 1977

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

[14] B. E. Boser, I. M. Guyon and V. N Vapnik: “A Training Algorithm for Optimal Margin Claasifiers” Fifth Annual Workhop on Computational Learning Theory, Pittsburg, ACM. pp. 144-152, 1992

[15] B. Schölkopf, C. J. C. Burges and A. J. Smola (eds.): “Advanced in Kernel Methods. Support Vector Learning” The MIT Press, Cambridbe, MA, 1999

[16] V. N. Vapnik: “The Nature of Statistical Learning Theory”, Springer, 1995

[17] R. Herbrich: “Learning Kernel Classifiers, Theory and Algorithms”, The MIT Pres, Cambridge, MA, USA, 2002

[18] 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

[19] 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

[20] 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

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