Improving the generalization capability of the binary CMAC

Tamás Szabó, Gábor Horváth
Technical University of Budapest, Department of Measurement and Information Systems,
H-1521. Budapest, Műegyetem rkp. 9, Bldg. R. I./113. Hungary,
Tel.: +36 1 463 2057, Fax.: +36 1 463 4112, E-mails: [szabo, horvath]@mit.bme.hu

Abstract: *This paper deals with some important questions of the binary CMAC neural networks. CMAC - 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. Although the CMAC architecture was proposed in the middle of the seventies quite a lot open questions have been left even for today. Among them the most important ones are its modeling and generalization capabilities. While some essential questions of its modeling capability were addressed in the literature no detailed analysis of its generalization properties can be found. This paper shows that the CMAC may have significant generalization error, even in one-dimensional case, where the network can learn any training data set exactly. The paper shows that this generalization error is caused mainly by the training rule of the network. It derives a general expression of the generalization error and proposes a modified training algorithm that helps to reduce this error significantly.*

1 Introduction and motivations

Cerebellar Model Articulation Controller (CMAC) networks play an important role in non-linear function approximation and system modeling. The main advantages of CMAC against the MLP, RBF, etc. networks are its extremely fast learning and the possibility of low-cost digital implementation. This latter property originates from its multiplier less structure. The CMAC network can be considered as an associative memory, which performs two subsequent mappings. The first one - which is a non-linear mapping - projects an input space point ($\mathbf{u}$) into a binary association vector ($\mathbf{a}$). The association vectors always have C active elements, which means that C bits of an association vector are ones and the others are zeros. C is an important parameter of the CMAC network and it is much less than the length of the association vector. As the value of C affects the generalization property of the CMAC it is often called generalization parameter. The CMAC uses quantized input, 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. 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.

The first mapping should have the following characteristics: It should map two neighboring input points into such association vectors, where only a few elements - i.e. few bits - are different. 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 neighborhood determined by the parameter C - should not have any common bits. This mapping is responsible for the non-linear property of the whole system.

The second mapping calculates the output of the network as a scalar product of the association vector ($\mathbf{a}$) and a weight vector ($\mathbf{w}$): $y(\mathbf{u}) = \mathbf{a}(\mathbf{u})^T\mathbf{w}$. The two mappings are implemented in a two-layer network architecture. The first mapping can be implemented by a fixed combinational network which contains no adjustable elements; the trainable elements, the weight values which can be updated using the simple LMS rule, are used in the second layer [1].

CMAC is often considered as a real alternative to the most popular feed-forward networks because its capability is close to that of the more complex RBF and MLP networks. However, contrary to the MLP and RBF networks, there are no rigorous analysis concerning the approximation capability of the CMAC. In general, approximation capability can be regarded from two different points of view. Firstly it is a question if a network can learn the training points exactly, secondly it is the question of generalization, i.e. the capability of the network to give reasonable outputs for inputs not encountered in the training process. These two different capabilities can be characterized by the modeling error and the generalization error, respectively:

*This work was supported by the Hungarian Fund for Scientific Research (OTKA) under contract T021003

- *modeling error* is the error of the net at the training points, that is the difference between the desired values and the responses of the CMAC,
- *generalization error* is the error of the approximation for inputs not used during training. Generalization error describes how much information the network managed to retrieve from the training set.

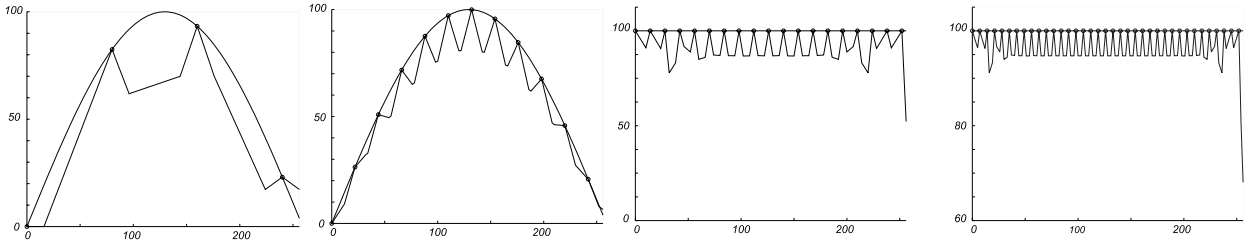
Some papers give general theoretical network-architecture-independent results regarding both questions of the approximation capability [2], [3], [4], [5], [6] and there are a few papers dealing with the special question of the modeling capability of the CMAC networks: [7], [8], [9]. However, according to the knowledge of the authors, there are no general results concerning the generalization capability and generalization error of the CMAC. The network-independent results generally provide too pessimistic upper bound for approximation, so some network-specific results are highly required. The paper deals with this question, it shows that there may be quite significant generalization error even in the case where zero modeling error can be reached. Further, it gives a general mathematical expression of the error and shows that the very reason of this poor generalization capability mainly comes from the improper selection of the generalization parameter and the training process. Modifying the learning rule the generalization error can be reduced significantly.

The CMAC networks have some special features. An important one is that its modeling capability depends on its input dimension. A 1D CMAC - which means that the inputs are formed from one-dimensional data points - is a universal approximator in that respect, that it can learn all training points exactly. In higher dimensional cases this is not true. CMAC can represent exactly only data points of additive functions [8]. Thus the generalization and the modeling error superpose, which means that the two types of error cannot be separated easily in higher dimensional cases. Because our attention is focused on the generalization error, hereafter we will mainly discuss the 1D CMAC network.

Actually it is very hard to define the generalization error, while there is no information about the function to be approximated elsewhere than just at the training points. This is why we will use the deviation from a suitable chosen interpolating function. The simplest way is if we assume linear interpolation between the training points. On the other hand, if we can give an error limit for the deviation from the linear interpolating, piecewise linear function, than we can apply the results of the spline approximation theory (see [10] and [11]) regarding the approximation error of the functions.

2 The generalization error of the CMAC

The problem of generalization can be shown most easily using a simple example. Let us choose a half period of an $y = sin(x)$ function to be approximated by CMAC. First we have to discretize the input: in this example we will use 256 quantization levels. Using equidistant training points different quality approximations can be obtained depending on the ratio of C and d_{train} as it is shown in Fig. 1. Let $C = 64$ and the distances of the equidistant training points $d_{train} = 80$ (Fig. 1 a)). Hereafter we assume that all weights of the networks are initialized to zero values.



$C = 64$ and $d_{train} = 80$ a)

$C = 32$ and $d_{train} = 22$ b) $\frac{C}{d_{train}} = 1.45$

$C = 32$ and $d_{train} = 14$ c) $\frac{C}{d_{train}} = 2.28$

$C = 16$ and $d_{train} = 7$ d) $\frac{C}{d_{train}} = 2.28$

Figure 1: CMAC output for different C and d_{train} parameter sets

As we expect the CMAC linearly interpolates between the training points when $d_{train} = C$ and can not interpolate well if $d_{train} > C$. It is obvious, because the neighboring points activate such association vectors which will differ from that of the nearest training point exactly by $1, 2, 3...$ active bits, thus a training point can influence the output in its neighborhood of radius C. This means that the CMAC has local generalization property. Fig. 2 shows this, where the similarly shadowed parts denote the basis functions selected by the given training points.

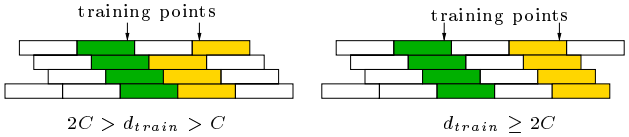


Figure 2: The weights influenced by the training points

If $d_{train} > C$ there will be weights which remain unchanged during training. If $d_{train} \leq 2C$ the weights are selected during training in such a way that for all possible input data at least one modified weight will be selected during recall. If $d_{train} \geq 2C$ and the weights are initialized to zero than there will be such input data between the training points for which the trained network will produce zero output values. It is because these input points will select weights all of which are unchanged (zero). A much more interesting example can be seen in the last part of the figure (Fig. 1 b). The distance of the training points are smaller than C, that is all of the weights are modified during learning. In spite of this, the generalization of the CMAC is really bad (see Fig. 1 c) and d) too).

As we can observe, the generalization error of the LMS trained binary CMAC is quite large if $C \neq k * d_{train}$, $k \in \mathbb{Z}$ ($\mathbb{Z}$ denotes positive integers here) even if a simple constant function should be approximated by a 1D CMAC. We can also see a trend, that if the $\frac{C}{d_{train}}$ quotient is closer to integer, the generalization error is less dominant.

2.1 Mathematical analysis of the generalization error

It can be proved (see e.g. [9]) that during training the weight vector converges to the pseudo-inverse solution:

$$\hat{\mathbf{w}} = \mathbf{A}^{\dagger}\mathbf{d} \qquad (1) \qquad \mathbf{A} = \begin{bmatrix} \text{---} & \mathbf{a}_1^T & \text{---} \\ \text{---} & \mathbf{a}_2^T & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{a}_N^T & \text{---} \end{bmatrix} \qquad (2)$$

where $\mathbf{A}^{\dagger}$ is the Moore-Penrose pseudo-inverse of the matrix built up from the association vectors (Eq. 2), corresponding to the training data (in the equation N denotes the number of training points). Similarly we can form a matrix $\mathbf{T}$ from the association vectors used in the test phase. Let us practically assume, that $\mathbf{T}$ contains all of the possible association vectors (which means that all possible input data are used in the test phase). Thus the response of the optimally trained network is:

$$\mathbf{y}_{test} = \mathbf{T}\hat{\mathbf{w}} = \mathbf{T}\mathbf{A}^{\dagger}\mathbf{d} \qquad (3)$$

Further, we can define a $\mathbf{P}$ matrix, which describes the linear interpolation between the training points: $\mathbf{y}_{LIN} = \mathbf{P}\mathbf{d}$. The general form of $\mathbf{P}$ is shown in Eq. 4

$$\mathbf{P}(i,j) = \begin{cases} \frac{x_{train}(i+1) - x(j)}{x_{train}(i+1) - x_{train}(i)} & if \quad x_{train}(i) \leq x(j) < x_{train}(i+1) \\ 0 & otherwise \end{cases} \qquad (4)$$

That is the deviation from the linear interpolation can be given as:

$$\mathbf{h} = \mathbf{y}_{LIN} - \mathbf{y} = (\mathbf{P} - \mathbf{T}\mathbf{A}^{\dagger})\mathbf{d} = \mathbf{H}\mathbf{d} \qquad (5)$$

Here the values of $\mathbf{H}$ depend on solely the placement of the training points and the length of the support of the basis functions. Thus the error of the network ($\mathbf{h}$) depends on the desired function values $\mathbf{d}$ and the $\mathbf{H}$ matrix, that is the position of the training points and the network's generalization parameter C (the size of the support of the basis functions). A trivial upper limit of this error can be written as:

$$\| \mathbf{h} \|_{\infty} = \| \mathbf{H} \|_{\infty} \| \mathbf{d} \|_{\infty} \qquad (6)$$

One can see that if the placement of the training points and the maxima of the function in the training points are known, than we can give the error of the approximation (the deviation from the piecewise linear curve). Let us see some numeric examples: d_{train} was chosen to 22,10,16 and 8 which yield $\| H \|_{\infty} = 0.7099,\ 1.1876,\ 3.56 * 10^{-14}$ and $7.85 * 10^{-14}$ respectively. These numeric examples show, that if $\frac{C}{d_{train}} \in \mathbb{Z}$, than the deviation from the linear approximation is zero (except for the precision of the matrix inversion).

It is important to see, that this upper bound is sharp. That is one can find a function for which the maximum of the generalization error (the deviation from the piecewise linear approximation) reaches the upper bound. This occurs if vector $\mathbf{d}$ is parallel with any row vector of $\mathbf{H}$. One the other hand this upper bound can be very pessimistic if the angle between $\mathbf{d}$ and the row vectors is small.

The main problem with the above derived mathematical expression is that Eq. 5 requires the inverse association matrix ($\mathbf{A}^{\dagger}$). Although this matrix depends only on the placement of the training points and the

value of C, the inverse must be computed in each case separately and the error or its maxima is not written in a closed form. After some attempts to determine the closed form of this error through the explicit form of the pseudo-inverse a promising solution seems to be found, however, its complexity in well beyond the limits of this paper, thus the details will be presented in a forthcoming paper. Here a special case will be detailed.

2.2 A special case

The root of the generalization problem can be found in the weight update process. If we analyse the effect of the positions of the training points it can be recognized that unless $C = k * d_{train},\ k \in \mathbb{Z}$ there will be some weights which are more frequently updated than the others. For a given case see Fig. 3. The numbers show how many times the corresponding basis functions will be selected by the training points and how many times the corresponding weights will be updated in a training epoch. We assume that a training point is taken once in every epoch, however, the unequal update frequency remains true in all other cases too.

Consider the learning rule again. As the LMS learning rule is used during training the error, $\varepsilon = y - d$ at the training of a given point x_d is distributed evenly between the selected weight values. Thus the increment of all selected weights are the same in an update cycle. This results in that the weights, which are more frequently updated influences much more the output than the others.

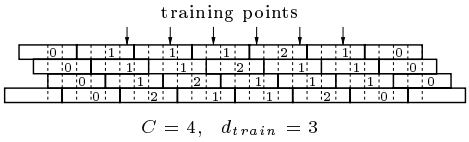


Figure 3: Weight update frequency in a training epoch

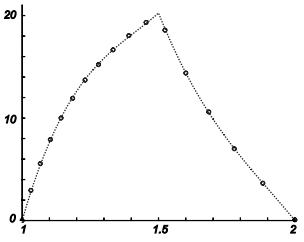


Figure 4: Generalization error vs. $\frac{C}{d_{train}}$

Obviously, this kind of generalization error is the result of the training strategy. This generalization error depends on the ratio of $\frac{C}{d_{train}}$. Fig. 4 shows the results of a simulation series of 16 training processes, where $C = 32$ and $d_{train} = 16...32$, so $\frac{C}{d_{train}}$ varies from 1 to 2. Again a constant function was selected for illustration. In Fig. 4 the absolute maxima of the generalization error is shown as a percent of the constant level.

For this special case when the function to be approximated by CMAC is constant and the training points are equally distributed with d_{train} distance between them we can give a simple analitical form of this generalization error. It can be proved that the weights of the network can be grouped into two sets. The weights in the first set is activated $B_L = \lfloor \frac{C}{d_{train}} \rfloor$ times in every training epoch, while it is $B_H = \lfloor \frac{C}{d_{train}} \rfloor + 1$ for the other set. $\lfloor . \rfloor$ stands for the floor function a largest integer $l < (.)$. We can prove also that the number of the active weights from the two sets are $N_L(x_d)$ and $N_H(x_d)$ respectively, for all the x_d training points. It depends only on C and d_{train}. ($N_L(x_d)$ and $N_H(x_d)$ are not necessarily equal.) Obviously $N_L(x_d) + N_H(x_d) = C$ and one can derive that $N_L(x_d) = \left(c - d + \lfloor \frac{C}{d_{train}} \rfloor d_{train} \right) \lfloor \frac{C}{d_{train}} \rfloor$ and $N_H(x_d) = C - \lfloor \frac{C}{d_{train}} \rfloor d_{train} + \lfloor \frac{C}{d_{train}} \rfloor C - \lfloor \frac{C}{d_{train}} \rfloor^2 d$. While we have assumed a constant function and so $d_i = d$ for all i we can compute the two sort of weight values -from the two sets- in the steady state:

$$w_L = B_L \frac{d_i}{N_L B_L + N_H B_H} \qquad (7) \qquad\qquad w_H = B_H \frac{d_i}{N_L B_L + N_H B_H} \qquad (8)$$

We also can determine $N_L(x_m)$ and $N_H(x_m)$ values for the middle point x_m between two neighbouring training data: $N_H(x_m) = N_H(x_d) - m$ and $N_L(x_m) = N_L(x_d) + m$ where $m = min\{C mod\ d_{train},\ (kd_{train}) mod\ C\}$ and k is the minimal integer that $kd_{train} > C$.

That is, we have the CMAC response in every point and so we can give the generalization error. This error takes its highest value in the midpoint between the training points. Thus,

$$h_{gen}(C, d_{train}) = m(w_H - w_L) \qquad (9)$$

We have managed to give the generalization error of the CMAC for this simple case. As one can see on Fig. 1 and Fig. 5 the generalization error seems to be proportional with the function values in the neighboring training points (except for some leakage effect which comes from the sharp changing of the function). This allows us to conclude that the effect of a training point value is practically local while it is theoretically not entirely true for all cases but when $d_{train} = C$. The practical values of **H** matrix support this statement.

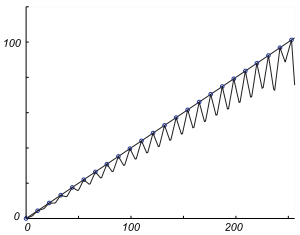


Figure 5: $C = 16$, $d_{train} = 11$

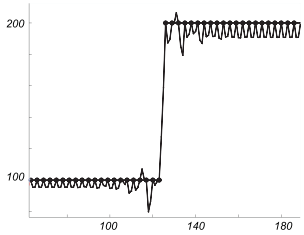


Figure 6: $C = 8$, $d_{train} = 3$

This generalization error formula in Eq. 9 corresponds to the special case of Eq. 5 namely the case of $\mathbf{d} = [1\ 1\ 1\ ...\ 1]^T$. Using the trick to determine the weight update frequency (by establishing the two weight sets) we have managed to bypass the matrix inversion problem for the special case.

On the other hand we can see that the error estimation given by Eq. 9 is too optimistic if the function varies quickly (see Fig. 6). The local increase of the generalization error is caused by the sudden changing of the function. Correct error estimation for this case could be given by using the inverse association matrix.

3 Modified CMAC learning algorithm

As we have seen, using the classical LMS training algorithm the generalization error could be intolerable large. Here we propose a new training approach based on the previously detailed background, which has far better generalization performance. The main problem of the LMS-like training was that the weights of the set are evenly incremented which has an effect that the more frequently changed weights can grow much faster than the others and they will dominate the reproduction of the values of the training points. Thus, in the midpoints between the training points, where the less frequently updated -and so the less dominant- weights form the output of the network the error can be large.

The idea is to forth the weights to be evenly dominant during training. An optimal case would be for a given training point d_j if the corresponding weights were $w_i = \frac{d_j}{C}$ for all w_i, which are selected by the d_j training point. However, this is not possible, while the neighboring training points also affects these weights, because they always have common weights if $d_{train} \leq C$. We can easily construct an algorithm, which helps us to get more proper weight distribution:

$$w_i^{new} = w_i^{old} + \mu(\frac{d_j}{C} - w_i^{old}) \tag{10}$$

One can realize that Eq. 10 is nothing else than a simple exponential averaging. Unfortunately this training algorithm has a serious drawback. The last data in the exponential average have larger influence than the previous ones (the effect of a training point will be exponentially decreased by training further points). This means that this kind of algorithm will be sensitive to the order (the sequence) of the training points. Usually it is an unwanted effect. However, this effect can be reduced if the training points presented randomly and the learning rate (μ) is sufficiently small.

We have seen that building some averaging into the training rule the generalization property of the CMAC network can be significantly improved. However, this kind of training algorithm gives correct result only if the function to be approximated is constant. Apart from this simple case the output of the network will not be entirely error free. This will happen because the fast alterations in the function will be smoothed by the averaging. Hereafter we modify the averaging type learning rule to avoid both the training point sequence dependency and the smoothing effect.

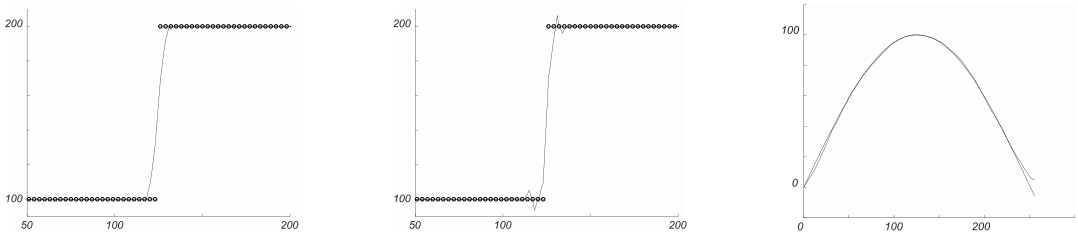
The training point order dependency effect can be avoided by using other averaging methods like simple recursive averaging, approximate recursive averaging or moving averaging. Conventional averaging method can not be used here because it requires the storage all of the training points. This is why some recursive form of the averaging should be used. The main disadvantage of the simple recursive averaging is that it requires division with the current sample number in every step. This is why exponential averaging can be used in adaptive hardware implementations where usually relatively slow adaptation (tracking) required and the simple (hardware) implementation is a must. Theoretically both the approximate recursive- and the moving averaging can be used.

Let us consider the smoothing property of the proposed algorithm again. This effect can be significantly reduced if a combined algorithm is used: the combination of the original LMS rule and the new one. This results in a regularization type algorithm, where the ratio between the LMS and the averaging part of the

learning rule can be determined according to the trade-off between the smoothing effect and the generalization error. Thus the learning rule for e.g. exponential averaging is

$$w_i^{new} = w_i^{old} + \mu\varepsilon_j + \lambda(\frac{d_j}{C} - w_i^{old}) \tag{11}$$

where λ is the regularization coefficient. Fig. 7 shows some representative results using the modified training algorithm. This results should be compared to the ones on Fig. 1 d) and Fig. 6.



$C = 8$ and $d_{train} = 3$ $\mu = 0, \lambda = 1$ a)

$C = 8$ and $d_{train} = 3$ $\mu = 0.5, \lambda = 0.5$ b)

$C = 32$ and $d_{train} = 22$ $\mu = 0.5, \lambda = 0.5$ c)

Figure 7: CMAC trained with the modified learning rule

Conclusion

The paper deals with the generalization error of the CMAC neural network. It shows that in spite of the universal approximator capability of the 1D CMAC rather large generalization error can be found depending on the network generalization parameter and the positions of the training points, even in very simple approximation problems. The paper gives a general mathematical expression of the generalization error and detailed results for special cases. Based on the generalization error analysis it proposes a modified training algorithm that can be combined with the original LMS rule, and that helps to reduce the generalization error significantly. Further tasks to extend the results for more general cases, for multidimensional problems and for higher order basis functions remain for future investigation.

References

[1] J.S. Albus, "A new approach to manipulator control: The Cerebellar Model Articulation Controller," *Transactions of the ASME*, vol. 97, no. 3, pp. 220–227, 1975.

[2] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, 1993.

[3] Andrew R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, may 1993.

[4] Věra Kürková, "Approximation of functions by neural networks," in *Proceedings of NC'98*, 1998, pp. 29–36.

[5] Kurt Hornik, "Some new results on neural network approximatiom," *Neural Networks*, vol. 6, pp. 1069–1072, 1993.

[6] Kurt Hornik, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[7] Neil E. Cotter and Thierry J. Guillerm, "The CMAC and a theorem of Kolmogorov," *Neural Networks*, vol. 5, pp. 221–228, 1992.

[8] Martin Brown, Christopher J. Harris, and Patrick C. Parks, "The interpolation capabilities of the binary CMAC," *Neural Networks*, vol. 6, pp. 429–440, 1993.

[9] Martin Brown and Chris Harris, *Neurofuzzy Adaptive Modelling and Control*, Prentice Hall, 1994.

[10] Carl de Boor, *A practical guide to Splines*, Springer-Verlag, 1978.

[11] Larry L. Schumaker, *Spline functions, Basic Theory*, Wiley & Sons, 1981.