ISSN 0005–1144
ATKAAF 42(3–4), 151–157 (2001)

Mato Baotić, Ivan Petrović, Nedjeljko Perić

Convex Optimization in Training of CMAC Neural Networks

UDK 004.032.26
IFAC IA 2.8.3;4.0

Original scientific paper

Simplicity of structure and learning algorithm play an important role in the real-time application of neural networks. The Cerebellar Model Articulation Controller (CMAC) neural network, with associative memory type of organization and Hebbian learning rule, satisfies these two conditions. But, Hebbian rule gives poor performance during off-line identification, which is used as a preparation phase for on-line implementation.

In this paper we show that optimal CMAC network parameters can be found via convex optimization technique. For standard ℓ_2 approximation this is equivalent to the solution of Quadratic Program (QP), while for ℓ_1 or ℓ_∞ approximation solving Linear Program (LP) suffices. In both cases physical constraints on parameter values can be included in an easy and straightforward way.

Key words: CMAC neural network, identification, convex optimization, quadratic program, linear program

1 INTRODUCTION

Due to the fact that Cerebellar Model Articulation Controller (CMAC) neural network is universal approximator [5, 4], it can be successfully used in identification of nonlinear processes [11]. Combined with the on-line algorithm for network weights adaptation, CMAC network can be used for identification of slowly varying non-linear processes. Using an instantaneous linearization technique [12] on a trained CMAC neural network it is possible to extract, on-line, the best linear model of identified process. Extracted linear model can then be used, for instance, to derive predictive control law.

One open issue in real time application of any type of neural network is the question of initialization of network weights. In order to have a »nice« behaviour of CMAC network during the startup period it is necessary to use some preidentification procedure. Computationally fast Hebbian algorithm used for on-line CMAC learning has some drawbacks when it comes to this off-line training.

Since the network training is basically optimization technique, and CMAC network is for a given set of inputs linear mapping, it is possible to use convex optimization procedure to obtain optimal network weights. Reliability, and simplicity of such algorithms, plus existence of a very large number of professional and shareware solvers, are just some of the positive points of convex optimization approach.

In Section 2 short description of CMAC neural network structure is given. Characteristic features of CMAC network, which enable improvement of standard lookup table technique performance, are explained. In Section 3, we describe training of CMAC neural network, first with standard Hebbian learning rule, and then learning based on convex optimization procedure. Comparison of different types of parameters training is presented in Section 4. At the end, some conclusions are made in Section 5.

2 STRUCTURE OF CMAC NEURAL NETWORK

As depicted in Figure 1, CMAC is an associative memory type of neural network, which for the description of nonlinear function

$$y = f_N(\mathbf{x}, \mathbf{w}), \tag{1}$$

uses two mappings

$$f : \mathcal{X} \to \mathcal{A}, \tag{2}$$

$$g : \mathcal{A} \to \mathcal{Y}, \tag{3}$$

where $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ is input vector, $\mathbf{a} \in \mathcal{A} \subseteq \mathbb{R}^M$ is association vector, $\mathbf{w} \in \mathcal{W} \subseteq \mathbb{R}^M$ is vector of network weights (parameters), n is number of inputs, and M is number of weights. For simplicity of description we use single-output CMAC network $y \in \mathcal{Y} \subseteq \mathbb{R}$, but results can be easily extended to the multi-output case.

In simple terms, CMAC network is perceptron-like lookup table technique. At first, input vector $\mathbf{x}$ is transformed, in a convenient way, into association vector $\mathbf{a}$. In order to fit continuous values of $\mathbf{x}$ into finite number of M distinctive association vector elements, it is necessary to quantize input va-

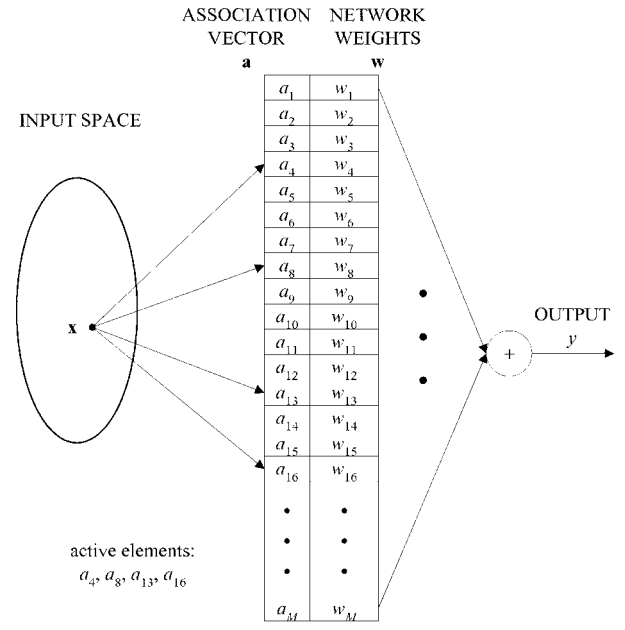

Fig. 1 Structure of CMAC neural network. For any input vector **x**, *just a small fraction (in example G = 4) of association vector elements are active, i.e. differs from zero. Weighted sum of active elements productes output*

lues. For this purpose, receptive fields are organized in layers, as shown in Figure 2. Every input triggers the same number of receptive fields. This number, called *generalization parameter G*, can modify inherent local approximation capabilities of CMAC network. Larger values of G ensure that vectors that are close in $\mathcal{X}$ map mostly the same elements of $\mathcal{A}$. Since output of CMAC network is calculated as a projection of association vector **a** onto a vector of adjustable weights **w**

$$y = \mathbf{a}^T \mathbf{w} = \sum_{i=1}^{M} a_i w_i , \tag{4}$$

this is the same as saying that similar inputs will produce similar outputs.

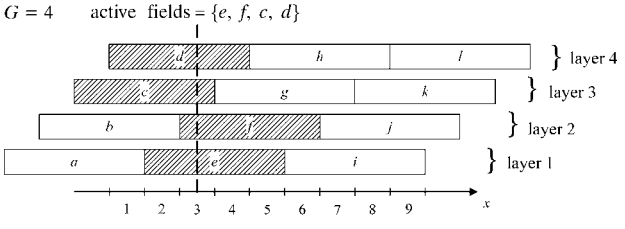

Fig. 2 Organization of CMAC neural network receptive fields for one dimension case

However, lookup table techniques have some common problems:

– Local learning procedure leads to the poor performance in case when higher level of generalization is needed.
– Quantization of input signals adds additional noise, thus reducing the achieved quality of function approximation.
– Size of table grows exponentionally with the number of inputs.

As mentioned before, parameter G deals with the first problem (local learning). Although the second problem (quantization) can not be eliminated, its effects can be reduced. Originally Albus [1] used $\mathcal{A} = \{0, 1\}^n$. This type of association space, with rectangular shape of receptive field functions, gives discontinuous (staircase) function approximation without analytical derivatives. Since in modern identification/control applications, not only good function approximation is needed, but also sensitivity of output to the changes of input (i.e. derivatives), staircase approximation is not satisfactory. This lead to the development of higher order CMAC neural networks [7] which use continuous receptive field functions, and thus can give arbitrary degree of continuous function derivatives.

Evaluation of higher order receptive field functions takes more time, but, the loss of calculation speed is more than compensated by the improvement of CMAC approximation capabilities. One possible solution for receptive field functions are B-Splines, which, for one-dimensional input, can be calculated with the recursive procedure [7]

$$B_j^s(x) = \left[\frac{x - \lambda_{j-s}}{\lambda_{j-1} - \lambda_{j-s}}\right] B_{j-1}^{s-1}(x) + \left[\frac{\lambda_j - x}{\lambda_j - \lambda_{j-s+1}}\right] B_j^{s-1}(x), \tag{5}$$

$$B_j^1(x) = \begin{cases} 1 & \text{for } x \in \left[\lambda_{j-1}, \lambda_j\right] \\ 0 & \text{otherwise} \end{cases}, \quad j = 1, \ldots N,$$

where λ_j is border of j-th quantization interval, N is total number of quantization intervals, and s is order of spline. In multi-dimensional input case calculation is extended as follows [4]

$$R_{j_1,\ldots,j_n}(\mathbf{x}) = \frac{B_{1,j_1}^s(x_1) \cdots B_{n,j_n}^s(x_n)}{\sum_{j_n=1}^{N_n} \cdots \sum_{j_1=1}^{N_1} \prod_{i=1}^{n} B_{i,j_i}^s(x_i)}, \tag{6}$$

where $R_{j_1,\ldots,j_n}(\mathbf{x})$ is multi-dimensional receptive field function, and $B_{i,j_i}^s(x_i)$ is receptive field function for the i-th dimension. Two dimensional example is presented in Figure 3. Functions defined by (5) and (6) satisfy three significant properties: positivity (only G receptive fields per single dimension have value greater than zero), compact support (other $N_i - G$ receptive fields have zero values) and nor-

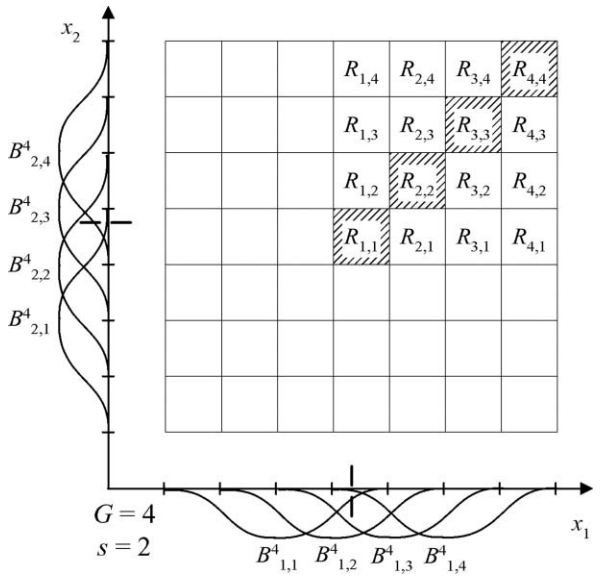

Fig. 3 Multi-dimensional receptive field functions. In majority of CMAC network realizations, only shaded fields are considered as active

malization (sum of all receptive field values is equal to 1), and therefore are appropriate to be used as receptive field functions in CMAC network [4].

From (6) it is obvious where the third problem (exponential growth) arises from. It is true that for a specific value of $\mathbf{x}$ only N^G receptive fields have nonzero values. But, to be able to handle every possible input we would, theoretically, need N^n receptive fields, and consequently same number of network weights. Even for a small scale problems this number can be too big. On the other hand, it is unlikely that the entire input space of certain system would be visited in solving a specific problem. Thus it is only necessary to store information for receptive fields that are excited during training. Following this logic, normally some type of pseudo-random hashing is used to transform virtual address of active receptive field into a scalar address of the corresponding association vector element. Although only N^G elements are non-zero, even this number is to big. So, in practical application, additional cuts are made (see Figure 3) by selecting only G most informative receptive fields [7].

There are many practical aspects which have to be taken into consideration when designing CMAC network structure. We refer reader to the literature [10, 7, 8, 4]. For us it is sufficient to say that: by the choice of number of network weights M, discretization of input, generalization parameter G, and the type of hashing, function f is defined. In that case, from available input $\mathbf{x}(i)$, we can calculate association vector $\mathbf{a}(i)$ that has only G nonzero elements.

3 LEARNING OF CMAC NETWORK

When we speak about neural network learning, we think of some nonlinear fitting of calculated network outputs with the measured values.

3.1 Hebbian learning

For a given set of Q input-output values ($\mathbf{x}(i)$, $y_d(i)$) we want to find suitable values $\mathbf{w}^*$ to satisfy (if possible) the linear equations

$$y_d(i) = \mathbf{a}^T(i)\mathbf{w}^*, \quad i = 1,\ldots,Q. \tag{7}$$

Due to the linear output mapping, very simple Hebbian learning procedure can be used for on-line training of CMAC network [10]. For every input--output pair approximation error is used for adaptation of network weights

$$\mathbf{w}(i) = \mathbf{w}(i-1) + \beta \frac{e_i \mathbf{a}(i)}{\mathbf{a}^T(i)\mathbf{a}(i)}, \tag{8}$$

where e_i is approximation error, $e_i = y_d(i) - y(i)$, and β is training gain. In every step of iteration only those network weights that participate in output calculation are adjusted. Provided $0 < \beta < 2$ convergence of the method to the least squares solution of (7) will be maintained. So, for Hebbian learning, cost function

$$\mathcal{J}(w) = \sum_{i=1}^{Q} e_i^2, \tag{9}$$

is minimized.

Remark 1: Off-line CMAC network training with (8) can be obtained by cyclic repetition of all training points. But, number of iterations needed to reach optinam sloution can vary significantly with the variation of parameter β.

3.2 Quadratic programming

In off-line learning, for a given set of training points, it is enough to calculate association vectors only once, since they do not change during training of weights. If we store those vectors into matrix $\mathbf{A} = [\mathbf{a}(1),\ldots,\mathbf{a}(Q)]^T$, off-line training can be represented as a solution of matrix equation

$$\mathbf{A}\mathbf{w}^* = \mathbf{y}_d, \tag{10}$$

in a sense of ℓ_2 approximation. It is important to emphasize that although solution of the system (10) is analytically given by pseudo-inverse

$$\mathbf{w}^* = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y}_d, \tag{11}$$

it should never be solved in this way. Calculation of pseudo-inverse is computationally highly demanding and time consuming procedure. Not to mention the

fact that very big, sparse matrix $\mathbf{A}(G \ll M)$ usually produces ill conditioned matrix $\mathbf{A}^T\mathbf{A}$, whose inverse is numerically hard to calculate.

Luckily, there is a work around. Solution of (10) is equivalent to the following quadratic program

$$\begin{aligned} \min_{\mathbf{z}} \quad & \mathbf{e}^T\mathbf{e} \\ \text{subj. to} \quad & \mathbf{A}\mathbf{w} + \mathbf{e} = \mathbf{y}_d\,, \end{aligned} \tag{12}$$

where $\mathbf{z} \in \mathbb{R}^{M+Q}$ is vector of optimization parameters, which includes network weights and approximation errors, $\mathbf{z} = [w_1, \ldots, w_M, e_1, \ldots, e_Q]^T$.

So, by solving convex optimization problem (12), which can be done numerically with great efficiency [9], we can obtain optimal network weights, and approximation error for all input vectors.

Interesting side effect of convex optimization technique is possibility to include physical constraints of weight values. Namely, in order to obtain solution of original problem (12), when weight w_i is bounded between lower and upper bound

$$L_i \le w_i \le U_i\,, \quad i \in \{1, \ldots . M\}, \tag{13}$$

it is sufficient to include this additional constraint in optimization procedure (12). Number of variables stays the same, and complexity of algorithm does not change. Obviously even more general constraints on network weights can be included. Any constraint of the linear form (equality or inequality) can be just as easily added

$$\mathbf{F}\mathbf{w} \preceq \mathbf{g}, \tag{14}$$

where $\mathbf{F} \in \mathbb{R}^{c \times M}$, $\mathbf{b} \in \mathbb{R}^c$, and c is total number of weight constraints and the symbol $\preceq$ denotes componentwise inequality. This can be useful if network weights are internally coupled (e.g., network weights implemented as a coupled resistor network).

Remark 2: There are many solvers at disposal for solution of QP problems. In our work we used Matlab routines, but other solvers can be used as well. In order to save memory space and additionally improve the speed of computation, it is better to use solvers which can handle sparse matrices [9].

3.3 Linear programming

Question arises: what would change in optimization procedure if we used some other norm? Since, by definition, every norm is a convex function, other norms can be used as well. From a computational point of view two norms are particularly interesting: $\|\mathbf{e}\|_\infty$ and $\|\mathbf{e}\|_1$.

If the goal of optimization is to find such values of CMAC network weights as to minimize $\|\mathbf{e}\|_1$ norm

$$\mathcal{J}_1(\mathbf{w}) = \sum_{i=1}^{Q} |e_i|, \tag{15}$$

calculation of optimal network parameters reduces to the linear program

$$\begin{aligned} \min_{\mathbf{z}} \quad & \mathbf{1}^T\mathbf{e} \\ \text{subj. to} \quad & \mathbf{A}\mathbf{w} + \mathbf{e} \succeq \mathbf{y}_d \\ & \mathbf{A}\mathbf{w} - \mathbf{e} \preceq \mathbf{y}_d\,, \end{aligned} \tag{16}$$

where $\mathbf{z} \in \mathbb{R}^{M+1}$ is vector of optimization parameters, $\mathbf{z} = [w_1, \ldots, w_M, e_1, \ldots, e_Q]^T$, e_i is absolute value of approximation error, and $\mathbf{1}$ is vector consisting of Q ones, $\mathbf{1} = [1, \ldots, 1]^T$.

By solving LP (16) we obtain optimal network weights, and absolute value of approximation error for all input vectors. Since linear program belongs to the class of tractable algorithms [13] we can say that substantial improvement is made in respect to the original problem.

If $\|\mathbf{e}\|_\infty$ norm is chosen as a cost function

$$\mathcal{J}_\infty(\mathbf{w}) = \max |e_i|, \tag{17}$$

optimal CMAC neural network weights are found by solving the linear program

$$\begin{aligned} \min_{\mathbf{z}} \quad & \varepsilon \\ \text{subj. to} \quad & \mathbf{A}\mathbf{w} + \mathbf{1}\varepsilon \succeq \mathbf{y}_d \\ & \mathbf{A}\mathbf{w} - \mathbf{1}\varepsilon \preceq \mathbf{y}_d\,, \end{aligned} \tag{18}$$

where $\mathbf{z} \in \mathbb{R}^{M+1}$ is vector of optimization parameters, $\mathbf{z} = [w_1, \ldots, w_M, \varepsilon]^T$, ε is absolute value of maximal approximation error over complete set of input-output training pairs, and $\mathbf{1}$ is vector consisting of Q ones, $\mathbf{1} = [1, \ldots, 1]^T$. Linear program (18) is, judging by the number of variables, simplest of all optimization techniques presented so far. This type of optimization is necessary, for instance, when we want to have good pointwise function approximation (e.g. bounded tracking error is demanded).

In the rest of the paper, we refer to the quadratic program given by (12) simple as QP. Similarly LP corresponds to the (16), while LP_∞ corresponds to the (18).

There is a vast number of solvers available for solution of LP problems. In our work we used Matlab routines, but other solvers can be used as well. Standard algorithm for solving LPs is well known *simplex* method developed by Danzig back in 1953. Most solvers now implement interior-point methods which garantee to reach solution in polynomial time [2, 13]. All methods give global minimum with provable lower bound. Again, ability of potentially used solver to handle sparse matrices is desirable feature.

Remark 3: If QP and/or LP solvers need a feasible starting point, one such can be easily generated by putting all elements of $\mathbf{z}$ which correspond to the weights (w_i) to ze-ro and all elements which correspond to the errors (e_i, ε) to $\max|\mathbf{y}_d|$. Simple check shows that this point is indeed feasible for QP given by (12), LP_1 given by (16) and LP_∞ given by (18).

Remark 4: In order to achieve certain level of maximal absolute error, additional constraints of the form $e_i \le \varepsilon$, $i = 1, \ldots, Q$, can be included in QP and/or LP_1. Value ε in such a case has to be provided by the user. Caution is needed when demanding some value of ε. While original problem always has solution, this newly imposed requirements can make the problem infeasible. Majority of solvers in such case gives solution which violates constraints the least.

4 TEST RESULTS

To test algorithms for off-line identification with CMAC neural network described in previous section, two case studies were used: linear process, and batch distillation.

4.1 Linear process

Consider off-line identification of the following linear process

$$G(z) = \frac{0.2z^{-1} + 0.1z^{-2}}{1 - 1.5z^{-1} + 0.6z^{-2}}.$$

Sampling time was set to 0.5 [s] to get $Q = 500$ input--output pairs, as depicted in Figure 4. CMAC network with generalization parameter $G = 20$, memory size $M = 100$, and spline receptive field functions of order $n = 2$ was used to approximate function. Regression vector used as an input to the CMAC network was composed of two past values of process output and two past values of process input.

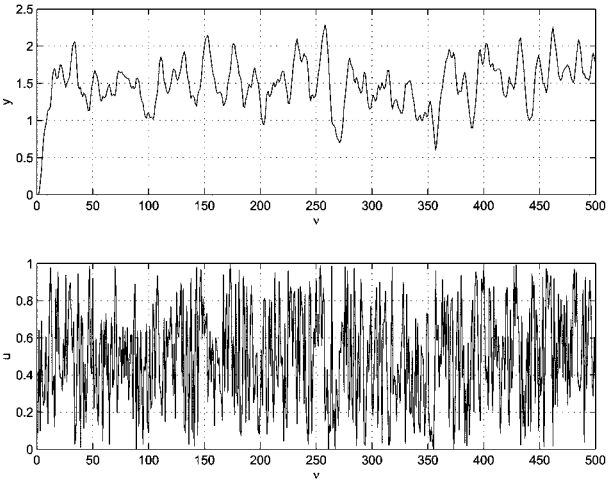

Fig. 4 Input-output signals for linear process

Results of network training are presented in Figure 5 and Figure 6.

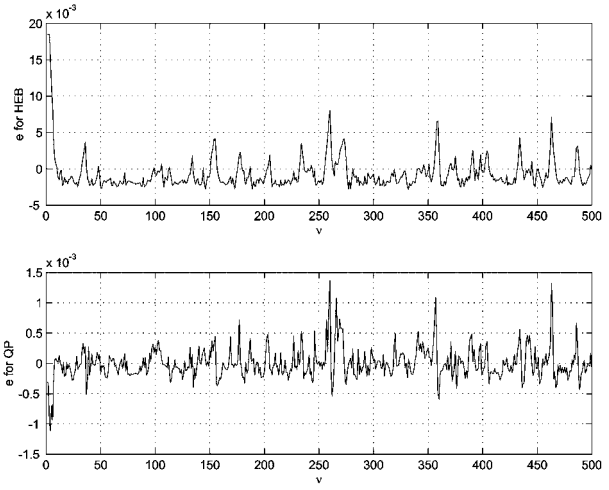

Fig. 5 Linear function approximation error for CMAC network trained with: a) Hebbian algorithm (10 000 iterations with $\beta = 0.6$), b) Quadratic Programming

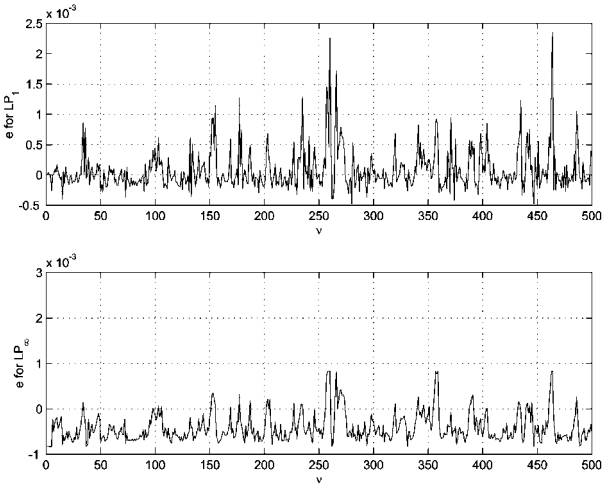

Fig. 6 Linear function approximation error for CMAC network trained with: a) LP_1 algorithm, b) LP_∞ algorithm

As summarized in Table 1, every algorithm outperformed others in the value of the cost function for which it was designed. Beside this expected behaviour, significant improvement over Hebbian learning can be seen for all proposed algorithms.

Table 1 **Summarized results for linear function approximation**

	$\mathcal{J}$	$\mathcal{J}_1$	$\mathcal{J}_\infty$
Hebbian	$3.16 \cdot 10^{-3}$	$8.63 \cdot 10^{-1}$	$3.13 \cdot 10^{-3}$
QP	$3.31 \cdot 10^{-5}$	$1.03 \cdot 10^{-1}$	$1.36 \cdot 10^{-3}$
LP_1	$4.81 \cdot 10^{-5}$	$9.13 \cdot 10^{-2}$	$2.37 \cdot 10^{-3}$
LP_∞	$1.34 \cdot 10^{-4}$	$2.36 \cdot 10^{-1}$	$8.25 \cdot 10^{-4}$

4.2 Batch distillation

Thermodynamical model of batch distillation process is derived under assumption that at every stage (tray) of column vapor and liquid are in equilibrium [6]. These nonlinear relations, together with total mass and component balance through all stages, give mathematical description of distillation process:

$$\mathbf{y}_n = f(\mathbf{x}_n, T_n)_P,$$

$$L = RD,$$

$$\frac{\mathrm{d}(H_0, \mathbf{x}_0)}{\mathrm{d}t} = V(\mathbf{y}_1 - \mathbf{x}_0),$$

$$\frac{\mathrm{d}(H_n, \mathbf{x}_n)}{\mathrm{d}t} = V(\mathbf{y}_{n+1} - \mathbf{y}_n) + L(\mathbf{x}_{n-1} - \mathbf{x}_n),$$

$$\frac{\mathrm{d}(H_{N+1}\mathbf{x}_{N+1})}{\mathrm{d}t} = -V\mathbf{y}_{N+1} + L\mathbf{x}_N,$$

$$\frac{\mathrm{d}H_{N+1}}{\mathrm{d}t} = -D$$

where P [Pa] is pressure, T [K] is temperature, L, V and D [kmol/h] are liquid, vapor, and distillate molar flow respectively, R is reflux rate, n is index of plate, $\mathbf{x}_n$ and $\mathbf{y}_n$ are vectors of liquid and vapor composition respectively, H_n is liquid holdup [mol], and N is total number of column plates. Index $n=0$ corresponds to the top, while $n=N+1$ represents bottom of the column.

Control and output variables are reflux rate $R \in [0,1]$, and concentration of distillate $\mathbf{x}_0$. For identification purpose distillation of a binary mixture aceton-water (250 mL of each component) was used.

As depicted in Figure 7, total number of $Q=695$ input-output pairs was collected, with the sampling rate of 8 [s]. CMAC network with generalization parameter $G=40$, memory size $M=1000$, and spline receptive field functions of order $n=2$ was used to approximate function. Regression vector was composed of three past values of process output and three past values of process input. Results are shown in Figure 8 and Figure 9.

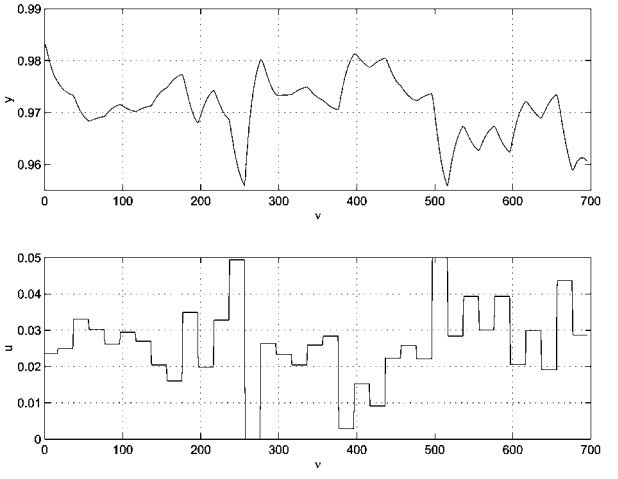

Fig. 7 Input-output signals for batch distillation process

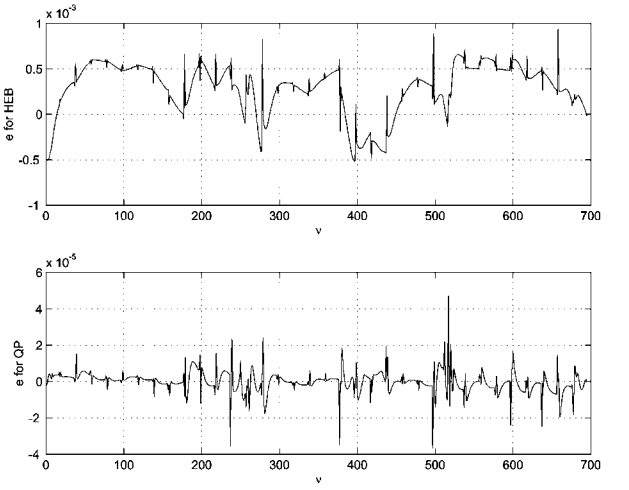

Fig. 8 Batch distillation approximation error for CMAC network trained with: a) Hebbian algorithm (10 000 iterations with $\beta=0.6$), b) Quadratic Programming

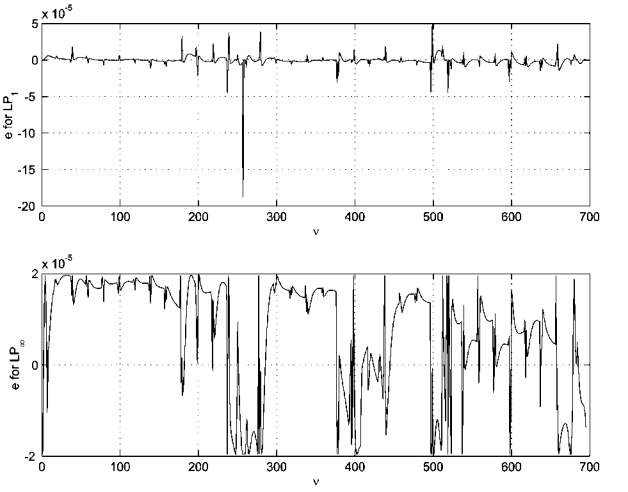

Fig. 9 Batch distillation approximation error for CMAC network trained with: a) LP_1 algorithm, b) LP_∞ algorithm

Table 2 **Summarized results for batch distillation approximation**

	$\mathcal{J}$	$\mathcal{J}_1$	$\mathcal{J}_\infty$
Hebbian	$1.12 \cdot 10^{-4}$	$2.54 \cdot 10^{-1}$	$9.35 \cdot 10^{-4}$
QP	$2.59 \cdot 10^{-8}$	$2.70 \cdot 10^{-3}$	$4.69 \cdot 10^{-5}$
LP_1	$6.20 \cdot 10^{-8}$	$2.41 \cdot 10^{-3}$	$1.88 \cdot 10^{-4}$
LP_∞	$1.40 \cdot 10^{-7}$	$9.02 \cdot 10^{-3}$	$1.96 \cdot 10^{-5}$

Concise results, presented in Table 2, confirm expected performance for all convex optimization algorithms. Similarly to the previous example, here too, better identification results are obtained with all convex optimization procedures than with simple Hebbian learning.

5 CONCLUSION

Since the network training is basically optimization technique, and CMAC network is, for a given set of inputs, linear mapping, it is shown that convex optimization procedure can be used to obtain optimal network weights. In the case of $\|\cdot\|_2$ norm this optimization is equivalent to the solution of Quadratic Program (QP), while for $\|\cdot\|_1$ and $\|\cdot\|_\infty$ norms it leads to the Linear Program (LP). Learning based on solving quadratic program and linear program described in our work shows some advantages over standard Hebbian learning: possibility to optimize network for different cost functions, from standard ℓ_2 to ℓ_1 and ℓ_∞ approximation. Additional constraints on weights (optimizing parameters) are included in an easy and straightforward way. Test on linear and nonlinear processes show very good performance of convex optimization.

Specially simple procedure is LP_∞ algorithm, since it demands the least number of constraints to be declared. Unfortunately, this algorithm, due to the cost function it minimizes, can have obvious offset after training (e.g. Figure 6). Since CMAC neural network stores information locally, it is virtually impossible to devise simple correction rutine, as would be the case for Multi-Layer Perceptron (MLP) neural networks. Still, tractability of convex optimization algorithms is very important advantage. Computational time is small, and grows gracefully with problem size. Global solutions are attained, with non heuristic stopping criteria, and with provable lower bound (and thus provable error with respect to the real solution).

ACKNOWLEDGEMENTS

This research was supported by the Ministry of Science and Technology of the Republic of Croatia, and by the pharmaceutical company Pliva, d.d., Croatia.

REFERENCES

[1] J. S. Albus, **A New Approach to Manipulator Control: Cerebellar Model Articulation Controller (CMAC)**. Trans. ASME – J. Dyn. Syst. Meas. Control, vol. 97, no. 3, pp. 220–227, 1975.

[2] V. Chvatal, **Linear Programming**. W. H. Freeman, New York, 1983.

[3] D. W. Clarke, C. Mohtadi, **Properties of Generalized Predictive Control**. Automatica, vol. 25, no. 6, pp. 859–875, 1989.

[4] S. Commuri, F. L. Lewis, **Control of Unknown Nonlinear Dynamical Systems Using CMAC Neural Networks: Structure, Stability and Pasivity**. In Proc. IEEE Int. Symp. Intell. Control, pp. 123–129, 1995.

[5] N. E. Cottre, T. J. Guillerm, **The CMAC Theorem of Kolmogorov**. Neural Networks, vol. 5, pp. 221–228, 1992.

[6] G. P. Distefano, **Mathematical Modeling and Numerical Integration of Multicomponent Batch Distillation Equations**. AIChE Journal, vol. 14, pp. 190–199, 1968.

[7] S. H. Lane, D. A. Handelman, J. J. Gelfand, **Theory and Development of Higher-Order CMAC Neural Networks**. IEEE Control Systems, vol. 12, no. 2, pp. 23–30, 1992.

[8] C. S. Lin, C. T. Chiang, **Learning Convergence of CMAC Technique**. IEEE Trans. Neural Network, vol. 8, no. 6, pp. 1281–1292, 1997.

[9] D. G. Luenberger, **Linear and Nonlinear Programming**. Addison-Wesley, Reading, Massachusetts, 1984.

[10] P. C. Parks, J. Militzer, **Improved Allocations of Weights for Associative Memory Storage in Learning Control Systems**. Proc. 1st IFAC symposium on Design Methods for Control Systems, pp. 777–782, 1990.

[11] I. Rivals, L. Personnaz, **A Recursive Algorithm Based on the Extended Kalman Filter for the Training of Feedforward neural models**. Neurocomputing, 20, pp. 173–188, 1998.

[12] O. S rensen, **Neural Networks in Control Applications**, Ph.D. Disertation, Department of Control Engineering, Aalborg University, Denmark, 1994.

[13] A. Schrijver, **Theory of Linear and Integer Programming**, John Wiley Sons, New York, 1986.

Konveksna optimizacija u učenju CMAC neuronskih mreža. Jednostavnost građe i algoritama učenja od iznimne su važnosti u primjenama neuronskih mreža u stvarnom vremenu. CMAC neuronska mreža s asocijativnom memorijskom organizacijom i Hebbianovim algoritmom učenja udovoljava ovim zahtjevima. Međutim, Hebbianov algoritam učenja ne daje dobre rezultate pri off-line identifikaciji, koja se koristi kao pripremna faza za on-line identifikaciju.

U ovom se članku pokazuje da se optimalne vrijednosti parametara CMAC neuronske mreže mogu dobiti primjenom tehnika konveksne optimizacije. Za standardnu l_2 aproksimaciju koristi se kvadratno programiranje (QP), a za l_1 i l_∞ aproksimacije linearno programiranje (LP). U oba je slučaja jednostavno uključiti fizikalna ograničenja na vrijednosti parametara u algoritam optimizacije.

Ključne riječi: CMAC neuronske mreže, identifikacija, konveksno optimiranje, kvadratno programiranje, linearno programiranje

AUTHORS' ADDRESSES:

Mato Baotić, Ivan Petrović, Nedjeljko Perić

Department of Control and Computer Engineering in Automation, Faculty of Electrical Engineering and Computing, University of Zagreb, HR-10000 Zagreb, Croatia

e-mail: mato.baotic@fer.hr, ivan.petrovic@fer.hr, nedjeljko.peric@fer.hr

Received: 2001–07–12