Copyright © IFAC Adaptive Systems in Control
and Signal Processing, Glasgow, UK, 1989

# CONVERGENCE PROPERTIES OF ASSOCIATIVE MEMORY STORAGE FOR LEARNING CONTROL SYSTEMS

**P. C. Parks**[1] **and J. Militzer**[2]

*Institut für Regelungstechnik: Fachgebiet Regelsystemtheorie, Technische Hochschule, Darmstadt, D-6100 Darmstadt, FRG*

Abstract. First, the cerebellar model articulation controller (CMAC), invented in the early 1970s by J S Albus, and the associative memory system (AMS), developed for learning control systems by H Tolle, E Ersü and J Militzer in the early 1980s, are briefly described. The underlying mathematics of the AMS learning or training algorithm is then given with a geometrical interpretation from which its convergence properties may be deduced. These are illustrated for some simple cases.

The original algorithm devised by Albus is very simple to compute but is slow to converge, and the second part of the paper investigates various methods of speeding up the algorithm. From an application of these new algorithms to test cases one is strongly recommended for further evaluation.

The results reported here are of relevance also to the topical and rapidly growing field of neural computing.

## INTRODUCTION

A special associative storage and retrieval algorithm with a "generalising" property, called the "Cerebellar Model Articulation Controller" ("C.M.A.C."), was first developed in the early 1970s by J S Albus as a simple model of the cerebellar cortex of mammals and as a potential controller for robots. The structure of the CMAC used earlier ideas embodied in the "Perceptron" of F Rosenblatt (Rosenblatt, 1961)

Further work on the CMAC concept to adapt it for real-time learning control applications was carried out in the early 1980s by H Tolle, E Ersü and J Militzer and led to the so-called "Associative Memory System" ("A.M.S."). Recent advances in microelectronics and computing have made these earlier ideas of learning and adaption much more feasible and it is not surprising to find that some working systems have now been developed (Ersü, Militzer, 1984). A typical learning control loop uses two AMS units as shown in Fig 1. The AMS units are used to build a model of the process as well as to store optimal control strategies.

It is rather surprising to find that the basic convergence properties of the CMAC algorithm were not investigated by Albus himself - indeed in Albus (1975) (p.299) he admitted that "at the present time there exists no formal proof of convergence of the procedure". This lack of a proof continued until 1989 when the present authors published a comprehensive investigation (Parks and Militzer, 1989). In the present paper we shall first summarise and illustrate these results, omitting the heavy matrix algebra involved. We shall then present some new results concerning improvements to the original CMAC and AMS learning algorithm. One of these improved algorithms is recommended for further evaluation in AMS units.

## ASSOCIATIVE MEMORY SYSTEMS (AMS)

Learning control systems, which operate by deducing future control actions from past information and experience, need by their very nature to store large quantities of past data. This can be accomplished by a "look-up" table, but at first sight the memory storage requirements appear to be insuperable. It was the objective of J S Albus, and those who followed him, to reduce the storage requirements dramatically by assuming that most input-output mappings of interest are continuous functions and that consequently the correct output or response at some point of the input space can be approximated by "generalising" the responses at closely neighbouring points, for example by linear interpolation. Thus the AMS algorithm spreads the magnitude of a given piece of data to be stored, between a fixed number, $\rho$, of different memory locations containing certain numbers called "weights".

[1]Deutsche Forschungsgemeinschaft Guest Professor 1986/87, Deutscher Akademischer Austauschdienst Guest Professor 1988; Home address: Applied and Computational Mathematics Group, Royal Military College of Science, Shrivenham, Swindon, SN6 8LA, England.

[2]Scientific Assistant, Present address: ISRA Systemtechnik GmbH, Mornegweg Str. 45A, D-6100 Darmstadt, Federal Republic of Germany.

Albus and his followers devised and used a learning or training procedure in which the individual weights are gradually adjusted to their correct values in an iterative procedure or algorithm. This procedure is based on the learning device invented by F Rosenblatt in the late 1950s called the "Perceptron", and can be traced back further to the book of D O Hebb (Hebb, 1949) which has given rise to the term "Hebbian learning".

The learning process works as follows (see also Fig. 2):-

(i) at the kth step of the algorithm $\rho$ weights are selected for adjustment based on the input information;

(ii) the existing values of the $\rho$ weights are added together and then divided by $\rho$ to form their average;

(iii) this average is subtracted from the value $\hat{r}_k$ which it is desired to store;

(iv) the difference is added as a correction to each of the $\rho$ chosen weights so as to update them.

Mathematically we may write (see also Fig 2)

$$\begin{bmatrix} x_{1'} \\ x_{2'} \\ \cdot \\ \cdot \\ \cdot \\ x_{\rho'} \end{bmatrix}_k = \begin{bmatrix} x_{1'} \\ x_{2'} \\ \cdot \\ \cdot \\ \cdot \\ x_{\rho'} \end{bmatrix}_{k-1} + \hat{r}_k \begin{bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix} - \frac{1}{\rho} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_{1'} \\ x_{2'} \\ \cdot \\ \cdot \\ \cdot \\ x_{\rho'} \end{bmatrix}_{k-1} \quad (2.1)$$

Here $x_{1'}, x_{2'}, \dots, x_{\rho'}$ represent the values of the $\rho$ weights chosen from a total number of p weights ($\rho \ll p$). The suffixes k-1 and k denote the $\rho$-vectors formed by the $x_{i'}$, at respectively the (k-1)th and kth steps of the learning algorithm.

The algorithm (2.1) may be rewritten in a neater vector form as

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mu_k \hat{\mathbf{a}}_k$$
$$\mu_k = \sqrt{\rho}\hat{r}_k - \hat{\mathbf{a}}_k^T \mathbf{x}_{k-1} \quad , \qquad (2.2)$$

which we shall call the "Albus learning algorithm". Here $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$ are p-vectors containing all the p weights while $\hat{\mathbf{a}}_k$ is a unit vector having entries of $1/\sqrt{\rho}$ in the $\rho$ places corresponding to the integers $1', 2', 3', \dots, \rho'$ appearing in (2.1), and zeros elsewhere.

The Albus learning algorithm (2.2) has an important geometrical interpretation (which was crucial to the development of the convergence theory in Parks and Militzer (1989)). This is that the point $\mathbf{x}_k$ is the foot of the perpendicular dropped from the point $\mathbf{x}_{k-1}$ to the (p-1)-dimensional hyperplane defined in p-dimensional space by the equation

$$\hat{\mathbf{a}}_k^T \mathbf{x} = \sqrt{\rho}\, \hat{r}_k \qquad (2.3)$$

It is customary in practice to start the algorithm with all the p weights set to zero so that $\mathbf{x}_o = \mathbf{0}$.

## CONVERGENCE PROPERTIES OF THE ALBUS LEARNING ALGORITHM

We first note that the objective of the algorithm is to find, if possible, a column p-vector $\mathbf{x}^*$, say, such that

$$\hat{\mathbf{a}}_i^T \mathbf{x}^* = \sqrt{\rho}\hat{r}_i \qquad (3.1)$$

for each i=1,2,3,...,N. If such an $\mathbf{x}^*$ does not exist then an approximate solution should be found.

A more familiar form of the equations (3.1) is the linear matrix equation

$$\mathbf{A}\, \mathbf{x}^* = \mathbf{h} \qquad (3.2)$$

where Nxp matrix $\mathbf{A}$ is composed of the row p-vectors $\hat{\mathbf{a}}_i^T$ and $\mathbf{h}$ is the Nx1 column vector with elements $\sqrt{\rho}\, r_i$.

In discussing convergence of the Albus algorithm the question of consistency or inconsistency of the equations (3.1) or (3.2) is very important, and in the case of inconsistency, which is quite likely to arise due to measurement noise distorting the values of $\hat{r}_k$, the form of training procedure is also crucial. Here we distinguish two forms of training:

(i) "cyclic training", in which the N equations are considered one by one in a particular cyclic order which is repeated many times, and

(ii) "random training", in which the equations are chosen at random.

These two procedures thus involve a cyclic or a random choice of the integer k appearing in the Albus algorithm (2.2) from the set of integers 1,2,3,...,N.

In Parks, Militzer (1989) it is shown that provided the equations (3.1) or (3.2) are consistent then the weights vector $\mathbf{x}_k$ in the algorithm (2.2) tends to a single fixed point $\mathbf{x}^*$ in the p-dimensional space as $k \to \infty$, which may or may not depend on the starting point of the algorithm $\mathbf{x}_o$ (which is usually but not always zero). If $\mathbf{x}_o \neq \mathbf{0}$ $\mathbf{x}^*$ will be independent of $\mathbf{x}_o$ if the unit vectors $\hat{\mathbf{a}}_i$ used as the algorithm proceeds "span" the vector $\mathbf{x}_o$, i.e. if $\mathbf{x}_o$ can be expressed as a linear combination of these $\hat{\mathbf{a}}_i$. Otherwise $\mathbf{x}^*$ will depend on that part of $\mathbf{x}_o$ which lies in the orthogonal complement of the space spanned by the $\hat{\mathbf{a}}_i$. Simple examples are given in Figs. 3(a) and 3(b).

If the equations (3.1) and (3.2) are inconsistent then the behaviour of the $\mathbf{x}_k$ as $k \to \infty$ is more complicated and depends on the type of training employed. With cyclic training $\mathbf{x}_k$ ends up describing an asymptotically stable "limit cycle" of N points which are approached by $\mathbf{x}_k$ as $k \to \infty$ as shown in Fig 3(c). On the other hand if random training is employed $\mathbf{x}_k$ ends up describing a random motion confined, however, within what we have called a "minimal capture zone". A "capture zone" may first be defined as the union of N sets of points, each set lying in one of the N hyperplanes defined by (2.3) and such that it projects orthogonally into or onto each of the other N-1 sets. A "minimal capture zone" is the smallest capture zone that can exist without losing the mutual orthogonal projection property. Fig 3(d) illustrates this concept and convergence to such a minimal capture zone.

The convergence behaviour of the Albus algorithm is set out in detail in Table 1 and the various possibilities have been illustrated in Fig 3 for simple cases in which p=2. Fuller details with proofs based on matrix algebra will be found in Parks and Militzer (1989).

## A COMPARISON OF 5 LEARNING ALGORITHMS

In general, the main input to an AMS or CMAC unit consists of a real vector $\mathbf{s}$ (the "stimulus"), $\mathbf{s} \epsilon R^n$, which is first mapped as the internal "associative" vector $\hat{\mathbf{a}}$. However, for the following study it is sufficient to define the stimulus as a scalar integer variable s. The first mapping is given by:

$$\hat{\mathbf{a}} = \hat{\mathbf{a}}(s) = (\hat{a}_1, \ldots, \hat{a}_p)^T$$
$$\hat{a}_i = \begin{cases} \frac{1}{\sqrt{\rho}} & \text{for } s+1 \le i \le s+\rho \\ 0 & \text{otherwise} \end{cases} \qquad (4.1)$$

(A description of the general n-dimensional case is given for example in Ersü, Militzer (1982)).

The response r(s;k) of the memory unit at point s after k training steps is obtained as the average of the active weights, which are determined by the non-zero elements of $\hat{\mathbf{a}}(s)$, giving the expression

$$r(s;k) = \frac{1}{\sqrt{\rho}} \hat{\mathbf{a}}^T(s) . \mathbf{x}_k . \qquad (4.2)$$

The following five algorithms were investigated:

Albus learning algorithm ("AL"). This is the basic learning algorithm as already defined in Eqn. (2.2).

Moving average training ("AV"). In this algorithm, the value of a weight $x_i$ is calculated as the average of the corrections, that it would receive in the basic algorithm.

For calculation of the averages, a "counter" vector $\mathbf{c} = (c_1, \ldots, c_p)^T$ of same dimension is assigned to the weight vector $\mathbf{x}$. Initial value is $\mathbf{c}_o = \mathbf{0}$.

The counters are updated to indicate how often each weight has been modified by the end of the kth training step, so that

$$\mathbf{c}_k = \mathbf{c}_{k-1} + \sqrt{\rho}\, \hat{\mathbf{a}}_k \qquad (4.3)$$

as $\sqrt{\rho}\, \hat{\mathbf{a}}_k$ contains ones in the positions of the active weights and zeros elsewhere.

The weights themselves are adjusted as follows:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mu_k \mathbf{b}_k$$
$$\text{with } \mu_k = \sqrt{\rho}\, \hat{r}_k - \hat{\mathbf{a}}_k^T \mathbf{x}_{k-1},$$
$$\mathbf{b}_k = (b_{1k}, \ldots, b_{pk})^T, \qquad (4.4)$$
$$b_{ik} = \begin{cases} \frac{\hat{a}_{ik}}{c_{ik}} & \text{if } a_{ik} \neq 0, \\ 0 & \text{otherwise} \end{cases}$$

Note that: (i) random noise with zero mean, which frequently disturbs the training data $\hat{r}_j$ in practical applications, can be filtered out; (ii) if the Eqns (3.2) are inconsistent, convergence is towards a single point, and not a limit cycle as is obtained with algorithm AL.

Partially optimised steplength in the last perpendicular direction ("OS"). This algorithm applies corrections to the weight vector in the perpendicular direction $\hat{\mathbf{a}}_k$, as the basic scheme AL does. However, the steplength $\mu_k$ is not designed to reach exactly the kth hyperplane $\hat{\mathbf{a}}_k^T \mathbf{x} = \sqrt{\rho}\, \hat{r}_k$, but rather to minimise the sum of squared perpendiculars to the hyperplanes defined by the last $\ell$ pieces of training data so that

$$I_\ell = \sum_{j=k-1+1}^{k} (\hat{\mathbf{a}}_j^T \mathbf{x}_k - \sqrt{\rho}\, \hat{r}_j)^2 \rightarrow \text{minimum}, \qquad (4.5)$$

where the optimization length $\ell$ is a suitably chosen integer with $1 \le \ell \le k$. The perpendicular distances are proportional to the response errors of the AMS.

The weights are adjusted according to:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mu_k \hat{\mathbf{a}}_k$$
$$\mu_k = \frac{\sum_{j=k-\ell+1}^{k} (\sqrt{\rho}\, \hat{r}_j - \hat{\mathbf{a}}_j^T \mathbf{x}_{k-1})\, \hat{\mathbf{a}}_j^T \hat{\mathbf{a}}_k}{\sum_{j=k-\ell+1}^{k} (\hat{\mathbf{a}}_j^T \hat{\mathbf{a}}_k)^2} \qquad (4.6)$$

Note that the basic algorithm AL is a special case of this procedure with $\ell$=1.

Training at the point with maximum error ("ME"). This method applies the normal training procedure at that point in input space $\hat{\mathbf{s}}_j$, for which the desired response $\hat{r}_j$ is currently reproduced with the largest error among a given number, $\ell$ say, of the latest pieces of data. The search length $\ell$ is suitably chosen with $1 \le \ell \le k$.

The correction is then applied for the point $\hat{\mathbf{s}}_m$, say, where the largest error occurs, so that

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mu_k \hat{\mathbf{a}}_m$$
$$\mu_k = \sqrt{\rho}\, \hat{r}_m - \hat{\mathbf{a}}_m^T \mathbf{x}_{k-1} \qquad (4.9)$$

Note that for $\ell$=1 this procedure is also identical to algorithm AL.

Partial Gram-Schmidt procedure ("GS"). The algorithm is designed to store new data without affecting the responses to a given number of former training points $\hat{s}_{k-\ell+1}, \ldots, \hat{s}_{k-1}$. A Gram-Schmidt orthogonalization is used to determine a correction direction $\hat{\mathbf{d}}_k$ which lies in the hyperplane spanned by the last $\ell$ vectors $\hat{\mathbf{a}}_{k-\ell+1}, \ldots, \hat{\mathbf{a}}_k$ and is orthogonal, if possible, to all of these vectors except the last one $\hat{\mathbf{a}}_k$. The new weight vector $\mathbf{x}_k$ is then given by the point of intersection between $\hat{\mathbf{d}}_k$ and the hyperplane $\hat{\mathbf{a}}_k^T \mathbf{x} = \sqrt{\rho}\, \hat{r}_k$. The "basis dimension" $\ell$ is suitably chosen with $1 \le \ell \le k$.

The weights are adjusted as follows:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mu_k \hat{\mathbf{d}}_k$$
$$\mu_k = \frac{\sqrt{\rho}\, \hat{r}_k - \hat{\mathbf{a}}_k^T \mathbf{x}_{k-1}}{\hat{\mathbf{a}}_k^T \hat{\mathbf{d}}_k} \qquad (4.13)$$

Note that (i) For $\ell$=1 this procedure is identical to algorithm AL; (ii) if Eqns. (3.2) are consistent, e.g. n=1 and the training data is not disturbed by noise and all N pieces of data are contained in a sequence of at most $\ell$ consecutive training steps, the algorithm converges at or before the last step of that sequence.

## COMPUTATIONAL RESULTS

The "target functions" used to generate the training data for the numerical experiments are defined for the N discrete argument values s=0,1,...,N-1. The functions are:

1. One period of a sine wave:

$$f_{SINE}(s) = \sin(2\pi \frac{s}{N-1}) \qquad (4.14)$$

2. A composite function which has a discontinuous first derivative if it is defined for a real argument s:

$$f_{COMP}(s) = \frac{1}{e^3-12+5}\left(\exp(\frac{3s}{N-1})-6\left|\frac{3s}{N-1}-1\right|+5\right) \qquad (4.15)$$

3. One period of a square wave:-

$$f_{RECT} = \begin{cases} 1 & \text{for } \frac{s}{N-1} < 0.5 \\ -1 & \text{otherwise} \end{cases} \qquad (4.16)$$

The functions are numbered in order of increasing difficulty of learning. The square wave tests show that the algorithms can cope with discontinuous functions (which should however be an exception in practical applications). The training is performed either in a cyclic or in a random way. In the first case, one cycle consists of the N distinct points in order, starting with some arbitrary number $s_o \epsilon [0,N-1]$ and jumping back through zero if necessary, i.e.:

$$\langle \hat{s}_j \rangle = s_o, s_o+1, \ldots, N-1, 0, 1, \ldots, s_o-1$$

This sequence $\langle \hat{s}_j \rangle$ is then trained repeatedly, with the reproduction error being evaluated only at the end of a pass (k=N,2N,3N,...).

In the second case, a pseudo random number generator is used to produce integers $\hat{s}_j$ uniformly distributed over the interval [0,N-1]. The generator is started at k=0 with an arbitrary initial seed $\phi_o$. The error is evaluated for k=N,2N,3N,... .

The desired responses used in the learning algorithms are either directly the target function values, i.e.

$$\hat{r}_j = f(\hat{s}_j) \qquad (4.17)$$

which is the undisturbed case, or they are disturbed by adding to the target function values some white noise with a Gaussian distribution, zero mean and standard deviation $\sigma$ so that

$$\hat{r}_j = f(\hat{s}_j) + n(j) \qquad (4.18)$$

The disturbances in general introduce inconsistencies in the Eqns. (3.2).

The experiments always start with all weights $x_i$ set equal to zero. In all experiments $p=N+\rho-1$. Fig. 4 gives an example how a memory response looks typically in different sections of the learning process. The accuracy of the memory response after k training steps is expressed by the root mean square error $e_{rms}(k)$:

$$e_{rms}(k) = \frac{\sqrt{\frac{1}{N}\sum_{s=0}^{N-1}(r(s_i k)-f.(s))^2}}{\sqrt{\frac{1}{N}\sum_{s=0}^{N-1}(f.(s))^2}} \; 100\% \qquad (4.19)$$

Learning in the CMAC or AMS has the properties that (i) in general the reproduction error $e_{rms}(k)$ does not necessarily decrease monotonically and (ii) variations of minor experiment parameters such as $s_o$ or $\phi_o$ may result in substantial deviations in the learning process. This behaviour is illustrated by Figs. 5 and 6 for the cases of cyclic and random training, respectively. To obtain reliable experimental results despite these facts, a statistical method was applied. The errors listed in Table 2 are mean values of 25 identical experiments, differing only in the cycle starting point $s_o$ or the initial seed $\phi_o$ for cyclic or random training, respectively. The mean values of the rms error defined by (4.19) are given for three values of k (the number of steps) equal to 50, 500 and 5000.

The memory parameter $\rho$ was chosen such that the error $e_{rms}$ decreases most quickly for all 5 algorithms on the average in a realistic test case (target function $f_{SINE}$, random training, no disturbances).

The mean execution times of the training algorithms $t_{mean}$ in Table 2 are in milliseconds and apply to a DEC PDP11/73 computer with a floating point accelerator. Because the computational burden of all five algorithms does not depend on the target function, the times are listed only for the first function $f_{SINE}$.

## SUMMARY OF PROPERTIES OF THE ALGORITHM

Algorithm GS converges very quickly for undisturbed training data, especially when $\ell$ is large and a high accuracy of the responses is required. The computational effort is enormous, for the training of a single piece of data usually takes more than one second when $\ell=50$.

Algorithm ME shows good convergence for almost all the test problems. The initial convergence rate is especially high, sometimes higher than for the algorithm GS. For the undisturbed case, the best convergence is obtained for the largest values of $\ell$. However, a large $\ell$ is not always beneficial with disturbed training data. The computational effort is moderate, increasing linearly with $\ell$.

Algorithm OS is well-suited for disturbed smooth test functions. It is able to reach a high accuracy in these cases when $\ell$ is large. In undisturbed cases the convergence is relatively slow and further slowed down by large values of $\ell$. The computational effort is higher than for algorithm ME.

Algorithm AV converges initially faster than algorithm AL when cyclic training is used, but later on, convergence becomes very slow in the undisturbed tests. In the case of disturbed training data, the results obtained with algorithm AV are among the best. The computational effort is low and nearly the same as for AL.

Interesting results can be expected when combining different algorithms, e.g. ME + AV or ME + GS. The final convergence of AV can probably be accelerated by limiting the counters such that $c_i \leq c_{max}$ holds for a predefined value $c_{max}$. These ideas will be investigated further and reported in a future paper.

At the present time we recommend use of the algorithm ME, because it achieves good convergence for a broad spectrum of test cases with moderate computational effort.

## REFERENCES

Albus, J S, (1972). Theoretical and experimental aspects of a cerebellar model, PhD Thesis, University of Maryland, USA.

Albus, J S, (1975). A new approach to manipulator control: the cerebellar model articulation controller (CMAC), ASME Transactions Series G, Journal of Dynamic Systems, Measurement and Control, 97, 220-227.

Albus, J S, (1975). Data storage in the cerebellar model articulation controller (CMAC), AMSE Transactions Series G Journal of Dynamic Systems, Measurement and Control, 97, 228-233.

Ersü, E and Militzer, J, (1982). Software implementation of a neuron-like associative memory system for control applications, Proc. 8th Int. Symposium ISMM, MIMI, Davos, Switzerland.

Ersü, E, (1983). On the application of associative neural network models to technical control problems, Proc. in Life Sciences: Localization and orientation in Biology and Engineering, (Ed.) Varja, Schnitzler, Springer Verlag, Heidelberg.

Ersü, E and Militzer, J, (1984). Real-time implementation of an associative memory-based learning control system for non-linear multivariable processes, 1st Measurement and Control Symposium on Applications of Multivariable System Techniques, Plymouth, UK, 109-119.

Ersü, E and Tolle, H, (1984). A new concept for learning control inspired by brain theory, Proc. 9th World Congress of IFAC, Budapest, Hungary, 7, 245-250.

Greville, T N E and Ben-Israel, A, (1974). Generalised inverses - theory and applications. Wiley Interscience, New York.

Hebb, D, (1949). Organization of behaviour, Wiley, New York.

Kohonen, T, (1988). Self-organisation and associative memory, (2nd edition), Springer Verlag, Heidelberg.

Lancaster, P, (1969). Theory of Matrices, Academic Press, New York, Chap. 2, 70.

Parks, P C and Militzer, J, (1989). Convergence properties of associative memory storage for learning control systems, "Avtomatyka i Telemekhanika" (in Russian), 50 No. 2 (to appear February 1989) English translation: Automation & Remote Control, Plenum Press, New York (to appear).

Rosenblatt, F, (1961). Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, Spartan Books, Washington DC, USA

Rummelhart, D E, McClelland J L et al, (1986). Parallel distributed processing, M.I.T. Press, Cambridge, Mass.

## TABLE 1

Various possibilities for the solution of the matrix equation (3.2) and the consequences for the Albus learning algorithm (2.2)

| Case No. | Relative sizes of p and N | Rank R of A in (3.2) | Rank R' of augmented matrix [**A**.**h**] in (3.2) | Intersection of hyperplanes defined by (2.3) with $k = 1,2,3,\ldots N$ | Convergence properties of Albus Algorithm: Cyclic training | Random training |
|---|---|---|---|---|---|---|
| 1. | $N<p$ | $R=N$ (maximum possible) | $R'=R$ (the only possibility) | Intersection is hyperplane of dimension $p-N$. | Converges to a single point which is unique if the starting point $\mathbf{x}_0$ is spanned by the unit vectors $\hat{\mathbf{a}}_i$, but otherwise is dependent on $\mathbf{x}_0$. | |
| 2. | $N=p$ | $R=N=p$ (maximum possible) | $R'=R$ (the only possibility) | Intersection is hyperplane of dimension 0, i.e. a unique point. | Converges to this unique point from all starting points $\mathbf{x}_0$. | |
| 3. | $N>p$ | $R=p$ | $R'=R$ | Redundant equations present in the set. Intersection is a unique point as in Case 2. | As in Case 2 | |
| 4. | $N>p$ | $R=p$ (maximum possible) | $R'=R+1$ | Equations (3.2) inconsistent. No single hyperplane of intersection exists. | "Limit cycle" exists, dependent on order or cyclic training, but not dependent on $\mathbf{x}_0$ | Points given by algorithm (2.2) eventually lie in a "minimal capture zone". |
| 5. | $N\lesseqgtr p$ | $R<\min(N,p)$ | $R'=R$ | Redundant equations present in the set. Intersection is hyperplane of dimensions $p-R$. | As in Case 1 | |
| 6. | $N\lesseqgtr p$ | $R<\min(N,p)$ | $R'=R+1$ | Equations (3.2) inconsistent. No single hyperplane of intersection exists. | "Limit cycle" exists. Shape dependent on order of cyclic training but independent of $\mathbf{x}_0$. Location independent of $\mathbf{x}_0$ if it is spanned by the $\hat{\mathbf{a}}_i$. | Points given by algorithm (2.2) eventually lie in a "minimal capture zone". |

## TABLE 2

target function = SINE
cyclic training, no disturbances

| Alg. | ρ | ℓ | e_rms k=50 | e_rms k=500 | e_rms k=5000 | t_mean [ms] |
|---|---|---|---|---|---|---|
| AL | 10 | - | 68.976% | 2.096% | 0.032% | 0.78 |
| AV | 10 | - | 11.908% | 4.080% | 2.552% | 0.88 |
| OS | 10 | 2 | 58.188% | 1.400% | 0.092% | 1.96 |
| OS | 10 | 10 | 28.840% | 1.544% | 0.140% | 6.30 |
| OS | 10 | 50 | 29.396% | 4.072% | 1.920% | 26.00 |
| ME | 10 | 10 | 14.976% | 0.196% | 0.000% | 3.11 |
| ME | 10 | 50 | 3.868% | 0.188% | 0.000% | 11.70 |
| ME | 10 | 250 | 3.868% | 0.188% | 0.000% | 57.69 |
| GS | 10 | 2 | 21.752% | 0.000% | 0.000% | 4.28 |
| GS | 10 | 10 | 10.796% | 0.000% | 0.000% | 40.60 |
| GS | 10 | 50 | 0.000% | 0.000% | 0.000% | 266.20 |
| target function = SINE random training, no disturbances | | | | | | |
| AL | 10 | - | 8.080% | 1.432% | 0.048% | 0.78 |
| AV | 10 | - | 9.652% | 5.204% | 3.912% | 0.88 |
| OS | 10 | 2 | 8.032% | 1.416% | 0.076% | 1.96 |
| OS | 10 | 10 | 8.140% | 1.820% | 0.116% | 6.00 |
| OS | 10 | 50 | 8.364% | 3.204% | 0.516% | 26.12 |
| ME | 10 | 10 | 5.648% | 0.612% | 0.000% | 3.18 |
| ME | 10 | 50 | 3.672% | 0.256% | 0.000% | 11.69 |
| ME | 10 | 250 | 3.672% | 0.212% | 0.000% | 57.74 |
| GS | 10 | 2 | 6.928% | 1.064% | 0.000% | 4.50 |
| GS | 10 | 10 | 3.624% | 0.256% | 0.000% | 44.27 |
| GS | 10 | 50 | 1.660% | 0.000% | 0.000% | 1000.22 |
| target function = SINE cyclic training, disturbed with SIGMA = 0.05 | | | | | | |
| AL | 10 | - | 69.012% | 8.452% | 8.528% | 0.78 |
| AV | 10 | - | 12.188% | 4.292% | 2.740% | 0.89 |
| OS | 10 | 2 | 58.152% | 5.480% | 6.772% | 2.05 |
| OS | 10 | 10 | 28.852% | 3.192% | 3.548% | 6.39 |
| OS | 10 | 50 | 29.380% | 4.740% | 3.280% | 26.10 |
| ME | 10 | 10 | 17.140% | 7.892% | 7.204% | 3.10 |
| ME | 10 | 50 | 7.308% | 7.300% | 7.628% | 11.68 |
| ME | 10 | 250 | 7.308% | 8.436% | 8.824% | 57.70 |
| GS | 10 | 2 | 23.432% | 10.472% | 7.516% | 4.33 |
| GS | 10 | 10 | 12.920% | 10.672% | 7.412% | 40.89 |
| target function = SINE random training, disturbed with SIGMA = 0.05 | | | | | | |
| AL | 10 | - | 9.412% | 8.200% | 7.036% | 0.78 |
| AV | 10 | - | 10.028% | 5.300% | 3.992% | 0.89 |
| OS | 10 | 2 | 9.152% | 7.668% | 6.672% | 1.97 |
| OS | 10 | 10 | 8.816% | 5.144% | 4.420% | 6.03 |
| OS | 10 | 50 | 8.692% | 4.076% | 2.480% | 26.14 |
| ME | 10 | 10 | 8.196% | 7.680% | 7.392% | 3.19 |
| ME | 10 | 50 | 8.012% | 7.628% | 8.008% | 11.69 |
| ME | 10 | 250 | 8.012% | 7.600% | 8.312% | 57.72 |
| GS | 10 | 2 | 9.588% | 9.444% | 8.224% | 4.53 |
| GS | 10 | 10 | 9.084% | 10.208% | 10.336% | 44.35 |
| GS | 10 | 50 | 7.764% | 8.736% | 9.376% | 1028.52 |

target function = COMP
cyclic training, no disturbances

| Alg. | ρ | ℓ | e_rms k=50 | e_rms k=500 | e_rms k=5000 |
|---|---|---|---|---|---|
| AL | 10 | - | 73.408% | 5.084% | 0.368% |
| AV | 10 | - | 12.096% | 5.108% | 3.248% |
| OS | 10 | 2 | 61.304% | 3.396% | 0.216% |
| OS | 10 | 10 | 30.728% | 3.352% | 0.348% |
| OS | 10 | 50 | 31.244% | 9.812% | 4.204% |
| ME | 10 | 10 | 19.160% | 0.436% | 0.000% |
| ME | 10 | 50 | 5.756% | 0.252% | 0.000% |
| ME | 10 | 250 | 5.756% | 0.252% | 0.000% |
| GS | 10 | 2 | 26.720% | 0.132% | 0.000% |
| GS | 10 | 10 | 14.892% | 0.008% | 0.000% |
| GS | 10 | 50 | 0.000% | 0.000% | 0.000% |

target function = COMP
random training, no disturbances

| Alg. | ρ | ℓ | e_rms k=50 | e_rms k=500 | e_rms k=5000 |
|---|---|---|---|---|---|
| AL | 10 | - | 10.648% | 1.916% | 0.136% |
| AV | 10 | - | 11.152% | 5.528% | 4.004% |
| OS | 10 | 2 | 10.684% | 2.016% | 0.140% |
| OS | 10 | 10 | 11.428% | 2.628% | 0.220% |
| OS | 10 | 50 | 11.640% | 4.440% | 0.804% |
| ME | 10 | 10 | 9.204% | 0.932% | 0.004% |
| ME | 10 | 50 | 6.724% | 0.376% | 0.000% |
| ME | 10 | 250 | 6.724% | 0.300% | 0.000% |
| GS | 10 | 2 | 10.352% | 1.524% | 0.052% |
| GS | 10 | 10 | 7.028% | 0.428% | 0.000% |
| GS | 10 | 50 | 4.444% | 0.020% | 0.000% |
| target function = COMP random training, disturbed with SIGMA = 0.05 | | | | | |
| AL | 10 | - | 13.068% | 10.900% | 9.268% |
| AV | 10 | - | 11.436% | 5.624% | 4.012% |
| OS | 10 | 2 | 12.748% | 10.156% | 8.784% |
| OS | 10 | 10 | 12.480% | 6.736% | 5.824% |
| OS | 10 | 50 | 12.000% | 5.428% | 3.328% |
| ME | 10 | 10 | 12.620% | 10.368% | 9.484% |
| ME | 10 | 50 | 12.592% | 9.576% | 10.004% |
| ME | 10 | 250 | 12.592% | 10.348% | 11.424% |
| GS | 10 | 2 | 14.248% | 12.644% | 10.832% |
| GS | 10 | 10 | 13.176% | 13.428% | 13.620% |
| GS | 10 | 50 | 11.612% | 11.516% | 12.368% |

target function = RECT
cyclic training, no disturbances

| Alg. | ρ | ℓ | e_rms k=50 | e_rms k=500 | e_rms k=5000 |
|---|---|---|---|---|---|
| AL | 10 | - | 68.736% | 15.468% | 3.736% |
| AV | 10 | - | 24.904% | 21.184% | 19.920% |
| OS | 10 | 2 | 59.260% | 14.940% | 4.452% |
| OS | 10 | 10 | 35.884% | 16.404% | 6.684% |
| OS | 10 | 50 | 36.208% | 22.212% | 17.912% |
| ME | 10 | 10 | 24.656% | 7.248% | 0.000% |
| MF | 10 | 50 | 25.108% | 5.856% | 0.000% |
| ME | 10 | 250 | 25.108% | 5.856% | 0.000% |
| GS | 10 | 2 | 29.704% | 1.508% | 0.000% |
| GS | 10 | 10 | 22.800% | 0.848% | 0.000% |
| GS | 10 | 50 | 0.000% | 0.000% | 0.000% |
| target function = RECT random training, no disturbances | | | | | |
| AL | 10 | - | 30.156% | 14.500% | 3.980% |
| AV | 10 | - | 26.296% | 22.432% | 20.904% |
| OS | 10 | 2 | 29.660% | 14.472% | 4.192% |
| OS | 10 | 10 | 27.336% | 15.592% | 5.596% |
| OS | 10 | 50 | 25.616% | 19.028% | 9.572% |
| ME | 10 | 10 | 26.116% | 11.224% | 1.236% |
| ME | 10 | 50 | 24.200% | 7.808% | 0.136% |
| ME | 10 | 250 | 24.200% | 6.844% | 0.004% |
| GS | 10 | 2 | 29.828% | 13.772% | 2.308% |
| GS | 10 | 10 | 25.864% | 8.464% | 0.004% |
| GS | 10 | 50 | 19.744% | 0.792% | 0.000% |
| target function = RECT random training, disturbed with SIGMA = 0.05 | | | | | |
| AL | 10 | - | 30.156% | 15.444% | 6.568% |
| AV | 10 | - | 26.344% | 22.388% | 20.904% |
| OS | 10 | 2 | 29.644% | 15.336% | 6.448% |
| OS | 10 | 10 | 27.304% | 15.976% | 6.440% |
| OS | 10 | 50 | 25.640% | 19.036% | 9.720% |
| ME | 10 | 10 | 27.184% | 11.940% | 5.504% |
| ME | 10 | 50 | 24.716% | 8.824% | 5.564% |
| ME | 10 | 250 | 24.716% | 7.576% | 6.140% |
| GS | 10 | 2 | 29.920% | 15.224% | 6.344% |
| GS | 10 | 10 | 26.220% | 10.728% | 7.212% |

## FIGURES

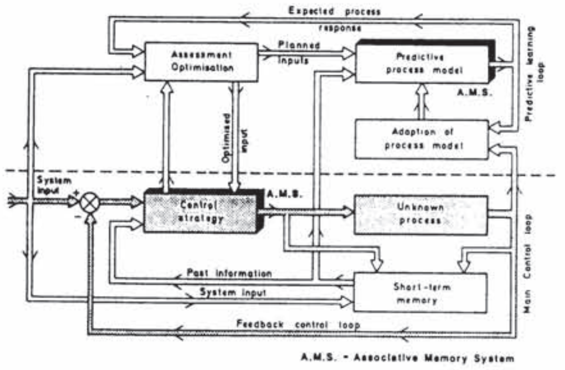


Fig. 1 Learning control loop with two AMS units

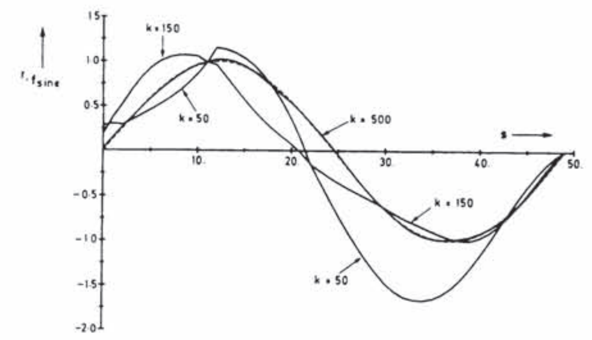


Fig. 4 Convergence of function to its desired form as the number k of Target function (4.14), steps of the algorithm increases. Algorithm AL, cyclic training $s_o=12$, $p = N+\rho-1 = 59$, $\rho=10$. No added noise.

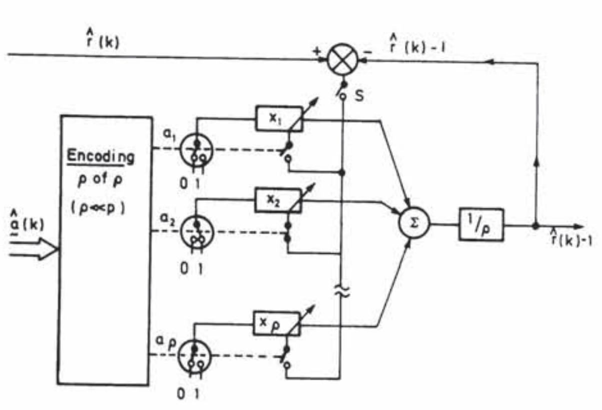


Fig. 2 Weight adjustment process

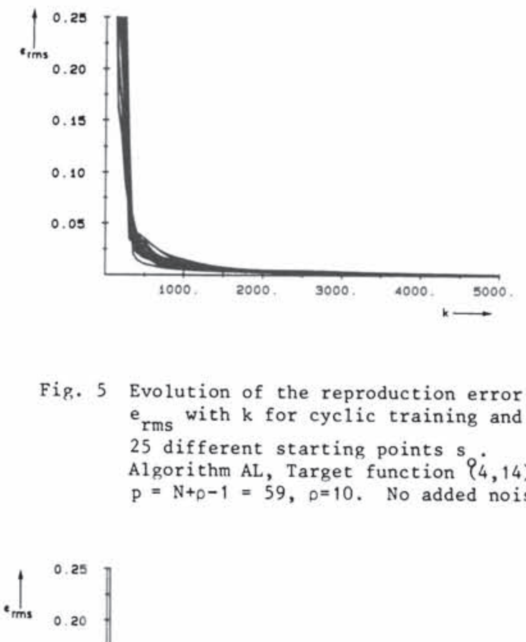


Fig. 5 Evolution of the reproduction error $e_{rms}$ with k for cyclic training and 25 different starting points $s_o$. Algorithm AL, Target function (4,14), $p = N+\rho-1 = 59$, $\rho=10$. No added noise.

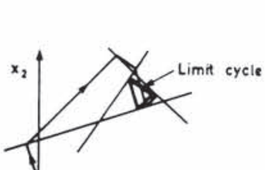


Fig. 3a. Convergence to unique **x*** ($p=N=R=R'=2$)

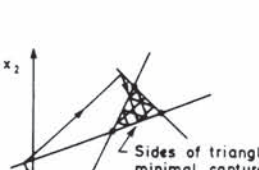
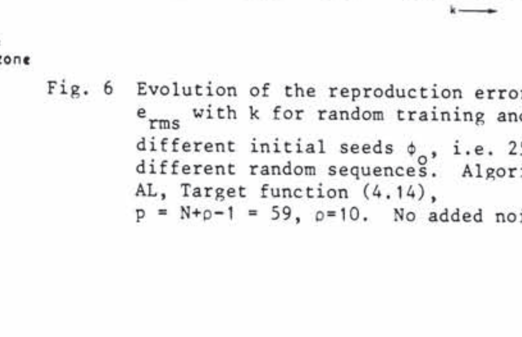
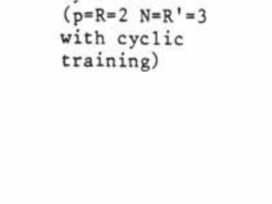


Fig. 3b. **x*** dependent on $\mathbf{x}_o$. ($p=2, N=R=R'=1$)





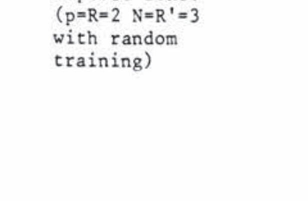


Fig. 6 Evolution of the reproduction error $e_{rms}$ with k for random training and 25 different initial seeds $\phi_o$, i.e. 25 different random sequences. Algorithm AL, Target function (4.14), $p = N+\rho-1 = 59$, $\rho=10$. No added noise.

Fig. 3c. Convergence to stable limit cycle ($p=R=2$ $N=R'=3$ with cyclic training)

Fig. 3d. Convergence to minimal capture zone. ($p=R=2$ $N=R'=3$ with random training)