UNH_CMAC Version 2.1
The University of New Hampshire Implementation of the Cerebellar Model Arithmetic Computer - CMAC

August 31, 1994
(last modified July, 1996)

W. Thomas Miller and Filson H. Glanz
Robotics Laboratory
University of New Hampshire
Durham, New Hampshire 03824

THE ALBUS CMAC

In this section we review the basic operating principles of the CMAC neural network as proposed by Albus [2,4-7]. We then present a mathematical description of the conventional CMAC in a form intended to facilitate software implementation. The notation used is somewhat different than in the original Albus papers, but the set of equations presented is numerically equivalent to the original Albus algorithm.

General Operation of the Albus CMAC

The operation of the Albus CMAC [2,4-7,11] can most easily be described in terms of a large set of overlapping, multi-dimensional receptive fields with finite boundaries. Any input vector falls within the range of some of the local receptive fields (the excited receptive fields), and falls outside of the range of most of the receptive fields. The response of the CMAC neural network to a given input is the average of the responses of the receptive fields excited by that input, and is not affected by the other receptive fields. Similarly, neural network training for a given input vector affects the adjustable parameters of the excited receptive fields, but does not affect the parameters of the remaining majority of receptive fields.

The organization of the receptive fields of a typical Albus CMAC neural network with a two-dimensional input space is as follows. The total collection of receptive fields is divided into C subsets, referred to in this document as “layers” (the layers represent parallel N-dimensional hyperspaces for a network with N inputs). The receptive fields in each of the layers have rectangular boundaries and are organized so as to span the input space without overlap. Any input vector excites one receptive field from each layer, for a total of C exited receptive fields for any input. Each of the layers of receptive fields is identical in organization, but each layer is offset relative to the others in the input hyperspace. The width of the receptive fields produces input generalization, while the offset of the adjacent layers of receptive fields produces input quantization. The ratio of the width of each receptive field (input generalization) to the offset between adjacent layers of receptive fields (input quantization) must be equal to C for all dimensions of the input space. The integer parameter C is referred to as the generalization parameter.

This organization of the receptive fields guarantees that only a fixed number, C, of receptive fields is excited by any input. However, the total number of receptive fields required to span the input space can still be large for many practical problems. On the other hand, it is unlikely that the entire input state space of a large system would be visited in solving a specific problem. Thus it is not necessary to store unique information for each receptive field. Following this logic, most implementations of the Albus CMAC include some form of pseudo-random hashing, so

that only information about receptive fields that have been excited during previous training is actually stored.

Each receptive field in the Albus CMAC is assumed to be an on-off type of entity. If a receptive field is excited, its response is equal to the magnitude of a single adjustable weight specific to that receptive field. If a receptive field is not excited, its response is zero. The CMAC output is thus the average of the adjustable weights of the excited receptive fields. If nearby points in the input space excite the same receptive fields, they produce the same output value. The output only changes when the input crosses one of the receptive field boundaries. The Albus CMAC neural network thus produces piece-wise constant outputs.

The implementation of the Albus CMAC logically proceeds as follows:

1. Identify the C receptive fields excited by the input.
2. Find the C adjustable weights for those receptive fields in a pool of stored weights.
3. Compute the average of the C adjustable weights.

The Albus CMAC Computation

Consider a classic Albus CMAC neural network with a real valued input vector

$$\underline{S} = < s_1 , s_2 , \dots , s_N > \qquad (1)$$

in an N-dimensional input space. Assume that the generalization parameter (the number of simultaneously excited receptive fields for each input) is C. The first step of the CMAC computation is to form a normalized integer input vector $\underline{S}'$ by dividing each component s_j of the input vector by an appropriate quantization parameter Δ_j:

$$\underline{S}' = < s'_1 , s'_2 , \dots , s'_N > = < \text{int}(\frac{s_1}{\Delta_1}) , \text{int}(\frac{s_2}{\Delta_2}) , \dots , \text{int}(\frac{s_N}{\Delta_N}) > \qquad (2)$$

The width of each receptive field along the jth axis is equal to C * Δ_j in the original input space, and is equal to C along all axes in the normalized input space.

The next step of the CMAC computation is to form the vector addresses $\underline{A}_i$ of the C receptive fields which contain the input point $\underline{S}'$:

$$\underline{A}_i = < s'_1 - ((s'_1 - i)\%C) , s'_2 - ((s'_2 - i)\%C) , \dots , s'_N - ((s'_N - i)\%C) >$$
$$= < a_{i1}, a_{i2}, \dots, a_{iN} > \qquad i = 1, 2, \dots , C \qquad (3)$$

where % represents the modulus operator, and the index i references the C parallel layers of receptive fields. $\underline{A}_i$ is the normalized N-dimensional address of one corner of the hypercubic region spanned by the single excited receptive field in layer i. Due to the properties of the modulus operator, the receptive field address components in the above equation are only valid for s'_j-i positive. A similar expression can be easily formulated, however, for s'_j-i negative.

Since the total number of receptive fields in a space of dimension N can be quite large, the receptive field addresses $\underline{A}_i$ are typically considered as virtual rather than physical addresses. The next step of the CMAC computation is then to form the scalar physical addresses A'_i of the actual adjustable weights to be used in the output computation:

$$A'_i = h(a_{i1} , a_{i2} , \dots , a_{iN}) \qquad (4)$$

In this equation, h(...) represents any pseudo-random hashing function which operates on the components a_{ij} of the virtual addresses of the receptive fields, producing uniformly distributed scalar addresses in the physical weight memory of size M.

Finally, the CMAC scalar output y(S) is just the average of the addressed weights:

$$y(\underline{S}) = \frac{1}{C} \sum_{i=1}^{C} W[A'_i] \qquad (5)$$

A vector CMAC output is produced by simply considering the weight memory locations to contain vector rather than scalar values, and by performing a vector rather than scalar average in the above equation. The weight memory W can contain integer or real values, depending on the desired implementation.

Network training is typically based on observed training data pairs S and $y_d(\underline{S})$, where $y_d(\underline{S})$ is the desired network output in response to the vector input S. The memory training adjustment ΔW is given by:

$$\Delta W = \beta * (y_d(\underline{S}) - y(\underline{S})) \qquad (6)$$

where the same value ΔW is added to each of the C memory locations $W[A'_i]$ accessed in the computation of y(S). This is equivalent to the well known LMS adaptation rule for linear adaptive elements. β is a constant training gain. If β is 1.0, the weights are adjusted to force the network output y(S) to be exactly equal to the training target $y_d(\underline{S})$. If β is 0.5, the network output is adjusted to fall halfway between the old output value and the training target. If β is 0.0, the weights are not changed.

EXTENSIONS TO THE ALBUS CMAC NEURAL NETWORK

In this section we discuss extensions to the conventional Albus CMAC neural network. When appropriate, modifications to the equations of the previous section are presented in a form intended to facilitate software implementation. When all of these extensions are implemented, the algorithms and learning system performance can be quite different than for the conventional Albus CMAC. However, the extensions are faithful to the original learning system concepts of Albus, and thus are still appropriately called CMAC algorithms. Note that some of the frequently described limitations of CMAC (such as only learning integer mappings) are in fact characteristics specific to the original Albus algorithm, and are neither properties of nor limitations of the general CMAC concept.

Organization of Receptive Fields

The descriptions in the first section apply to the classic Albus CMAC [2,4-7]. Research at UNH [12-14] and elsewhere [14,15] has investigated alternative lattice arrangements for the receptive fields which provide more uniform local generalization in higher dimensional input spaces. The receptive field mapping used in the conventional CMAC implementation has three key features which it is desirable to retain:

1. Each input in the multi-dimensional input state space falls into exactly the same number of receptive fields (C), and this number of overlapping fields is not dependent on the dimensionality N of the space or the total size M of the physical weight memory.

2. Regardless of the operating point in the multi-dimensional space, a change of one quantization level Δ_j in any input parameter causes exactly one active receptive field to become inactive and one new receptive field to become active. This provides for uniform quantization within the space.

3. The receptive fields are arranged in a geometrically regular way, such that the coordinates of the C excited receptive fields for any input can easily be determined in

software, or generated in hardware, without having to compare to independent coordinates stored for each of the very many receptive fields.

The conventional Albus CMAC implementation achieves these properties by offsetting the parallel layers of receptive fields along hyperdiagonals in the input space (the effect of the s'_j-i terms in equation 3). All inputs fall within the same number of receptive fields. However, some inputs fall near the centers of several receptive fields while other inputs fall near the centers of no receptive fields. This results in inhomogeneous and anisotropic generalization within the input state space. Ideally, the distribution of receptive fields should be uniform in the multi-dimensional input space unless prior knowledge of the function to be learned dictates otherwise.

The above three desirable features of the CMAC mapping can be used to place constraints on the possible arrangements of receptive fields, through which new arrangements can be generated. For the following discussion, assume that the N-dimensional input space has been normalized using equation 2 such that the widths of the receptive fields relative to all N components of the input are equal to C (the generalization parameter). The first and third items above suggest that the arrangement should be periodic in the normalized input space, with period C. This is equivalent to assuming that the receptive fields will be arranged in C parallel layers, each with non-overlapping receptive fields, and that only the offsets of each layer relative to the others can be varied when generating new receptive field distributions. In this case, the distribution of receptive fields throughout the space is uniquely defined by the arrangement of C receptive field centers in an N-dimensional hypercube of side C (referred to as the reference hypercube), with one receptive field centered at the corner of the hypercube (coordinate <0,0,...0>).

The individual receptive fields are hypercubes of side C in the normalized input space, and C receptive field centers are located inside of any region bounded by a hypercube of side C. Item 2 in the above list (uniform segmentation of the space) is thus only achieved if the C receptive field centers are spaced uniformly (with integer separation) when projected onto each of the N axes of the reference hypercube. Any arrangement which satisfies these criteria qualifies as a CMAC mapping according to the three items above. However, many of the possible arrangements (such as the conventional CMAC mapping) have locally nonuniform distributions.

Parks and Militzer [15] studied the arrangement of receptive fields in CMAC networks using distance between nearest neighbors as the evaluation criteria, based on the assumption that the most uniform distribution would have the greatest distance between nearest neighbors (given a fixed receptive field density in the space). They further assumed that the receptive field centers were arranged in a lattice defined by a displacement vector

$$\underline{D} = < d_1, d_2, \ldots , d_N > \qquad (7)$$

such that the coordinate of the ith receptive field in the reference hypercube was

$$< (i * d_1) \% C, (i * d_2) \% C, \ldots , (i * d_N) \% C > \qquad i = 1, 2, \ldots , C \qquad (8)$$

In these terms, the conventional CMAC would be defined by a lattice displacement vector of

$$\underline{D}_{Albus} = < 1,1, \ldots 1 > \qquad (9)$$

They performed an exhaustive search of such lattice arrangements for various values of C and N, and developed tables of best displacement vectors according to their nearest neighbor distance criteria.

We have developed a simple heuristic for selecting similar displacement vectors for any values of C and N which provide lattice arrangements equivalent to those found by Parks and

Militzer [14] without performing a search [12-14]. First, we choose the set of integers in the range 1 to C/2 which are not factors of C or integer products of factors of C (the value 1 can be included in the set). These are the candidate values for the displacement vector components (guaranteeing uniform projection of the centers on the axes of the reference hypercube), from which N must be selected. If there are more than N candidate values, we choose N (there are multiple nearly equivalent arrangements). If there are less than N candidates, the best that can be done is to use all of the candidate values with a minimum number of repetitions of any one value. However, the resulting mapping will be diagonalized (locally nonuniform) in some projections to lower dimensional spaces. A better solution in this case is to increase C, in order to achieve at least N candidates for the displacement vector.

For a CMAC with a three-dimensional input and a generalization of 16, the candidate values for the displacement vector are 1, 3, 5, and 7. A typical displacement vector might be <1,3,5> which would produce receptive field locations at <0,0,0>, <1,3,5>, <2,6,10>, and so forth, within the reference cube.

The CMAC computations described in the first section can easily be modified to accommodate the displacement vector approach to specifying receptive field placement. In this case, the virtual addresses of the excited receptive fields are given by

$$\underline{A}_i = < s'_1 - ((s'_1 - i * d_1)\%C) , s'_2 - ((s'_2 - i * d_2)\%C) , \ldots , s'_N - ((s'_N - i * d_N)\%C) >$$
$$= < a_{i1}, a_{i2}, \ldots, a_{iN} > \qquad i = 1, 2, \ldots , C \qquad (10)$$

In place of equation 3. Due to the properties of the modulus operator, the receptive field address components in the above equation are only valid for s'_j-i*d_j positive. A similar expression can be easily formulated, however, for s'_j-i*d_j negative.

It is impossible to quantify the improvement in performance to be gained in the general case by using a relatively uniform lattice of receptive fields, rather than the diagonalized arrangement used by Albus. In our experience, a more uniform arrangement of receptive fields typically provides learning system performance equal to or better than that achieved in the same application using the Albus arrangement (sometimes substantially better), with relatively little increase in computational effort. We typically select C to be the smallest power of 2 which is equal to or greater than 4*N, in which case a good displacement vector is simply the first N odd integers [12].

Receptive Field Sensitivity Functions

We have also investigated CMAC networks with graded, rather than all-or-none, receptive field sensitivity functions [12-14], as have others [16]. In this case, the CMAC output is influenced more by receptive fields for which the input vector is near the center of the active range, and is influenced less by receptive fields for which the input is near the limits of the active range. The CMAC output is then a weighted average of the C addressed adjustable parameters, rather than a simple average as in equation 5. This provides a continuous function approximation (rather than the piece-wise constant function approximation of the conventional CMAC). Any function which is maximum at the center and decreases smoothly to near 0 at the edges is satisfactory (e.g., linear decrease) for generating continuous outputs. Smooth outputs require that the slope of the function also approach 0 near the receptive field edges (e.g., cubic spline, gaussian, etc.). The critical issue is how to form the multi-dimensional receptive field sensitivity function from the one-dimensional primitives.

An obvious choice would be to simply base the receptive field sensitivity function on the radial distance from the center of the receptive field. While there is substantial evidence supporting the use of radial basis functions for general system approximation [17,18], the fixed, relatively sparse distribution of receptive fields inherent in CMAC family networks must be

considered. In the normalized input space defined in the previous section, each CMAC receptive field spans the interior of a hypercube of side C. The distance from the center to the nearest edge (the center of a face of the hypercube) of the receptive field is C/2, while the distance from the center to the farthest edge (a corner of the hypercube) is $N^{1/2}$ C/2. A radial basis function which tapers to a small value at the nearest edge of the receptive field will be very small in most of the corner region, confining the significant response to a limited region of the hypothetical receptive field. On the other hand, if the radial basis function tapers to a small value at the corner, it will have a significant output at the nearest edge of the receptive field, which is counter to the objective of a tapered receptive field.

A second alternative is to use the distance from the input point to the nearest face of the receptive field as the single parameter in the sensitivity function (rather than the radial distance from the center). This provides a receptive field sensitivity function which is maximum at the center and which has the same value at all points on its boundary. The sensitivity function will be continuous throughout the receptive field, but will have discontinuous slopes along select hyperplanes. We have found this to be the preferred alternative for CMAC neural networks [12].

The CMAC computations described in the previous sections can easily be modified to accommodate non-constant receptive field sensitivity functions. Let a_{ij} represent the jth component of the receptive field virtual address $\underline{A}_i$ in the normalized input space, as given in equation 10. Let s''_j represent the jth component of the real-valued, normalized input vector $\underline{S}''$:

$$\underline{S}'' = < s''_1, s''_2, \ldots, s''_N > = < \frac{s_1}{\Delta_1}, \frac{s_2}{\Delta_2}, \ldots, \frac{s_N}{\Delta_N} > \qquad (11)$$

The corresponding faces of the receptive field occur at $s''_j = a_{ij} - 0.5$ and $s''_j = a_{ij} + C - 0.5$ in the normalized space. For an arbitrary input point, the minimum distance δ_i to any face of receptive field i is then given by:

$$\delta_i = \min(s''_i - a_{ij} + 0.5,\ \ a_{ij} + C - 0.5 - s''_i) \qquad j = 1,2,\ldots,N \qquad (12)$$

In this case, $\delta_i = 0$ corresponds to any point on a face of the receptive field, while $\delta_i = C/2$ corresponds to the single point at the center of the receptive field. δ_i varies linearly along any linear path from any point on a face of the receptive field to the point at the center. Equation 5 can then be modified to give the new CMAC output:

$$y(\underline{S}) = \frac{\sum_{i=1}^{C} f(\delta_i) * W[A'_i]}{\sum_{i=1}^{C} f(\delta_i)} \qquad (13)$$

In equation 13, $f(\delta_i)$ represents the one-dimensional primitive which forms the basis of the receptive field sensitivity function. In practice, $f(\delta_i) = \delta_i$ is a simple and effective choice for the sensitivity function, producing piece-wise planar approximations.

Finally, equation 6 which describes the weight adjustment during training must be replaced by:

$$\Delta W_i = \beta * (y_d(\underline{S}) - y(\underline{S})) * f(\delta_i) * \frac{\sum_{n=1}^{C} f(\delta_n)}{\sum_{n=1}^{C} f^2(\delta_n)} \qquad (14)$$

Note that equation 6 described a weight adjustment ΔW which was added equally to each of the C adjustable weights representing the C excited receptive fields, while in equation 14 the weight adjustment ΔW_i for each receptive field is scaled by the magnitude of the receptive field sensitivity function $f(\delta_i)$. If $f(\delta_i) = 1$ for all δ_i, equations 13 and 14 reduce to equations 5 and 6.

It is obvious from comparing equations 5 and 6 to equations 13 and 14 that the implementation of non-constant receptive field sensitivity functions requires an increase in computational effort. Thus, in many cases the simpler case of $f(\delta_i) = 1$ is preferable, even though it results in piece-wise constant CMAC outputs. When a continuous CMAC output is important, we generally use $f(\delta_i) = \delta_i$, which results in piece-wise planar CMAC outputs. We have also experimented with cubic spline and truncated gaussian functions for $f(\delta_i)$. Paradoxically, learning system performance is usually worse when using these higher order sensitivity functions for input dimensions greater than two. We feel that this results from their much smaller magnitude near the edges of the receptive field, which dominates the receptive field volume in higher dimensional spaces. In applications we thus use either constant or linear sensitivity functions. Note that when using non-constant sensitivity functions, the arrangement of receptive fields is critical to good performance. In such cases, near uniform arrangements of receptive fields always provide substantially better learning system performance [12] than the diagonalized arrangement used by Albus.

Receptive Field Hashing Considerations

As discussed previously, the total number of receptive fields required to span a multi-dimensional space (C times) is often too large for practical implementation in terms of the storage required for the adjustable weights of all possible receptive fields. On the other hand, it is unlikely that the entire input state space of a large 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, most implementations of CMAC neural networks include some form of pseudo-random hashing to transform the vector virtual address $\underline{A}_i$ of an excited receptive field into a scalar address A'_i of the corresponding weight storage.

The major requirement of the hashing function is that the generated addresses A'_i be uniformly distributed in the range M of the available physical memory addresses, even for small changes in $\underline{A}_i$. In software implementations of CMAC, we have primarily used a simple hashing algorithm based on previously generated random number tables:

$$A'_i = \left(\sum_{j=1}^{N} T_j[a_{ij} \,\%\, R_j] \right) \% M \qquad (15)$$

where each T_j represents a table of uniformly distributed random values with R_j total table entries. Here, R_j effectively limits the dynamic range of the jth component of the normalized input vector, due to wrap-around of the table index. This hashing algorithm is numerically efficient and produces a good approximation to uniformly distributed addresses in the range 0 to M-1, as long as the dynamic range of the pseudo-random numbers in the tables is at least M. Note that a single bit change in any component of $\underline{A}_i$ can cause a large change in A'_i, as desired. In our experience, the quality of the hashing produced is sensitive to the quality of the uniform random number generator used to fill the T_j tables, but is not sensitive to the specific seed selected when using a good random number generator.

Hashing collisions are defined by:

$$A'_n = A'_m \quad \text{for} \quad \underline{A}_n \neq \underline{A}_m \qquad (16)$$

This has the effect of introducing learning interference, in the sense that training adjustments to two distinct and possibly distant receptive fields affect the same adjustable weights. In many implementations of CMAC (including that described originally by Albus), hashing collisions are ignored. This is often reasonable since the CMAC outputs are averages over several receptive fields (there is no specific desired response for any single receptive field), and since the goal is often to approximate a target function rather than to reproduce it exactly. In essence, CMAC training involves satisfying coupled equations using the available adjustable weights. Hashing collisions increase the coupling between equations (possibly slowing training convergence), but do not necessarily preclude finding a satisfactory solution. Of course, if hashing collisions are too frequent, the coupled set of training equations can become greatly overdetermined, with no satisfactory solutions.

The probability of no hashing collisions when training a single example of novel data is approximately:

$$P_{\text{no collisions}} = \left(1 - \frac{M_u}{M}\right)^C \qquad (17)$$

where M_u / M represents the fraction of the available weight storage that has been affected by previous training. It is obvious that for reasonable values of the generalization parameter C, the probability of collision-free training is low unless the utilization of available storage is very low. Thus, collision-ignorant hashing is best suited to applications which provide opportunity for repetitive training in order to resolve the additional coupling due to hashing collisions during training. This could involve repetitive off-line training for a pattern recognition application using a fixed training set, or could involve continuous on-line training (and thus retraining) in a control application. The advantage to collision ignorant hashing in applications involving on-line training is that old-information, which is not reinforced in subsequent training and which may no longer be useful, will eventually be completely overwritten and will not tie up storage resources. Complete saturation of the storage capacity will never occur, in the sense that new information can always be learned (although possibly at the expense of previously trained information).

Collision-free hashing generally involves storing some unique identifier of the virtual receptive field (such as its virtual address $\underline{A}_i$) along with each adjustable weight or weight vector, so that collisions can be detected and thus avoided. While hashing collisions may be eliminated, storing unique identifiers can result in a substantial increase in the amount of storage per receptive field, offsetting the reduction in storage which was the original motivation for hashing. Collision-resistant hashing can provide a compromise by storing a pseudo-random hash tag, derived from the receptive field virtual address $\underline{A}_i$, along with each adjustable weight or weight vector. Collisions can then be detected and avoided reliably (but not certainly) by comparing the stored hash tag with the value derived from the new address. If the tags do not match, the CMAC weight memory is searched sequentially until either a tag match or an unallocated location (blank tag) is found.

In our implementation of collision-resistant hashing, we generate the hash tag using the same pseudo-random generator used for address mapping (equation 15), but with different random tables, and with a different constant k in the final modulus (M in equation 15). Assuming that all detected collisions can be avoided, the probability of no hashing collisions when training a single example of novel data is then approximately:

$$P_{\text{no collisions}} = \left(1 - \frac{1}{k} * \frac{M_u}{M}\right)^C \qquad (18)$$

where k represents the dynamic range of the hash tag. In contrast to the collision-ignorant hashing (equation 17), the probability of collision-free training is high even if the utilization of

available storage is high, assuming a reasonably large value of k. Thus, collision-resistant hashing provides essentially the same performance as collision-free hashing, typically with substantially less storage per receptive field. Collision-resistant hashing is best suited to applications which require long-term retention of previously trained information in the presence of subsequent training of novel data, without reinforcement of previously trained examples (no learning interference). This could involve pattern recognition applications where it may be desirable to train new examples of certain classes as they are encountered, or control applications which require sequential skill learning. The disadvantage to collision-resistant hashing in applications involving on-line training is that old-information, which may no longer be useful, will tie up storage resources indefinitely. This can lead to complete saturation of the storage capacity.

Weight Magnitude Normalization

The output of the CMAC neural network for any input is an average over C adjustable weights. During training, individual weights are adjusted in order to reduce the error in the average. However, there are an infinite number (or in practice, a very large number) of combinations of weight values which will produce the same average. As a result, training provides only indirect control of weight magnitude.

This can be a problem during applications which require continuous on-line training. After the neural network training converges to a low error, residual error and sensor noise can cause continual small adjustments to the weights. These residual adjustments may average to zero over time for the CMAC output and yet not average to zero over time for individual weights. Some weights may drift towards large positive values while others drift towards large negative values, while maintaining good output performance in terms of averages over C weights. These unnecessarily large weights can eventually cause problems, however, in terms of increased error from hashing collisions and weight saturation for weight implementations with limited dynamic range.

The problem can be fixed by placing a penalty on large weight magnitudes during training (similar to the weight decay concept used as part of training for some other neural networks). Since the CMAC output is an average over multiple weights, an appropriate magnitude regularization is to penalize individual weights for varying from the average. The weight adjustment equation (equation 6) is then replaced by

$$\Delta W_i = \beta_1 * (y_d(\underline{S}) - y(\underline{S})) + \beta_2 * (y(\underline{S}) - W[A_i']) \qquad (19)$$

where separate training gains are used to individually emphasize the importance of the supervised learning versus the weight magnitude normalization. We generally select β_2 to be at most equal to β_1 / 4, since good output performance is generally the most important. Note that equation 6 described a single weight adjustment ΔW which was added equally to the C adjustable weights representing the C excited receptive fields, while in equation 19 the weight adjustment ΔW_i is different for each weight. A similar modification can be added to equation 14 for training when using non-constant receptive field sensitivity functions.

Variable Input Quantization

One limitation of the CMAC algorithms described in the previous sections is that quantization and generalization are fixed in the input space by the constants Δ_j and C (Δ_j is the quantization interval and C * Δ_j is the generalization width for input component j). In some problems, it may be desirable to have fine quantization and narrow generalization in some regions of the input space, with coarse quantization and broad generalization in other regions. If these regions can be identified in advance, the s_j / Δ_j terms in equations 2 and 11 can be

replaced by a more general $w_j(s_j)$, where each $w_j()$ is a fixed nonlinear warping function specific to each input.

If appropriate regional variations in quantization and generalization can not be determined in advance, it may be possible to represent input warping functions using models with adaptable parameters, and to adapt the warping functions during neural network training. Our laboratory and others have done preliminary experiments on adaptive input space warping, but no detailed information has yet been published. At least two general approaches have been proposed. In one approach, warping functions at the CMAC inputs are adapted in order to directly reduce the CMAC output error (using backpropagation of the error through the CMAC element). In a second approach, warping functions at the CMAC inputs are adapted in order to reduce the gradient of the estimated variance of the CMAC output error.

An alternative approach to this problem was reported by Moody [19]. He proposed using a multi-resolution CMAC in which the receptive fields in each layer were of different size. The layer with the largest receptive fields was trained first, followed by the layer with the next largest receptive fields, and so forth. In this way, broad generalization could be achieved where appropriate by the initial training of the large receptive fields, and fine details could be learned during the later training of the small receptive fields. Better learning system performance can be obtained by using complete CMACs in parallel, each with a different resolution [13], rather than single layers of different size receptive fields as proposed by Moody. The drawback to the multi-resolution approach (relative to input space warping) is that the smallest receptive fields are placed everywhere in the space, even though there may only be a limited region that actually requires fine quantization. Thus, the memory utilization is unnecessarily high.

THE UNH_CMAC CODE

The file *unh_cmac.c* is a C language implementation of a multiple CMAC driver. Prototypes and constants are defined in *unh_cmac.h*. The code includes multiple (and programmable) designs for the receptive field lattice and the receptive field sensitivity functions. It was developed in the Robotics Laboratory of the Department of Electrical and Computer Engineering at the University of New Hampshire. This C source code is not available for sale from UNH, and can not be resold. You may use it, and give it away freely to others, as long as you, and those you give it to, agree to acknowledge the UNH Robotics Laboratory in any project reports, manuscripts, software manuals, etc., which result from projects which utilize this code.

The code is generic C, but has been tested most thoroughly at UNH using C compilers for Microsoft Windows and Microsoft Windows-NT. The algorithms assume that the *int* data type refers to signed integers with 16 or more bits, and that the *long int* data type refers to signed integers with 32 or more bits. There is no coupling between these restrictions (*int* and *long int* can both be 32 bits, for example). The code makes extensive use of *long int* math, so it runs most efficiently on computer-compiler combinations with direct hardware support for *(long int)+(long int)*, *(long int)*(long int)* and *(long int)/(long int)*.

You should check the following functions for compatibility with your compiler:

malloc() - allocate data region of size specified in bytes.
free() - deallocate data region.
rand() - return random integer in range 0 - 32767.

Global Parameters

The code contains several global parameters which must be set at compile time since they determine the size of static storage elements. Most often, the default values shown below are reasonable and need not be changed.

```
/* Set this to the maximum number of independent CMACs you will need */
#define NUM_CMACS 8

/* Set this to the maximum number of input dimensions you will need in any CMAC */
#define MAX_STATE_SIZE 64

/* Set this to the maximum number of output dimensions you will need */
#define MAX_RESPONSE_SIZE 8

/* Set this to the maximum number of layers of receptive fields you will need */
#define MAX_GEN_SIZE 256

/* Set this to the size of the receptive field function look-up table you will use */
#define RF_TABLE_SIZE 128
```

Primary CMAC Procedures

```
int allocate_cmac(int num_state,int *qnt_state,int num_resp,
    int num_cell,int memory,int field_shape, int collide_flag)
```

This procedure is used to allocate a new CMAC with the specifications given in the parameters. The first parameter is the number of dimensions N to the CMAC input vectors. The second parameter is a pointer to a vector of dimension N which defines the quantization parameters (Δ_j in equations 2 and 11). The third parameter is the dimension of the CMAC output vectors. The fourth parameter is the generalization parameter C (the number of overlapping layers of receptive fields). In the version 2.1 code, this value must be equal to an integer power of 2 (1, 2, 4, 8, 16, ...). If not, the code will operate incorrectly. The fifth parameter is the total size M of the CMAC vector memory (the total number of weights is *memory*num_resp*). The sixth parameter sets the design of the CMAC receptive fields, using predefined constants. *ALBUS* creates a conventional Albus CMAC: on-off receptive fields in a hyperdiagonal arrangement. *RECTANGULAR* creates on-off receptive fields in a uniform arrangement. *LINEAR* creates linearly-tapered receptive field sensitivity functions in a uniform arrangement. *SPLINE* creates gaussian-like (cubic-spline approximation) tapered receptive field sensitivity functions in a uniform arrangement. *CUSTOM* creates a CMAC with user defined receptive field sensitivity function and arrangement (set using subsequent calls to *set_cmac_rf_magnitude()* and *set_cmac_rf_displacement()*). The last parameter is set TRUE or FALSE (not 0 or 0) to indicate whether or not to allow hashing collisions (TRUE allows collisions).

The procedure returns an integer value which is greater than 0 if successful. This value is called the CMAC handle and is used to identify the specific CMAC in subsequent calls to other procedures. If the returned handle is 0, an error occurred during allocation.

```
int deallocate_cmac(int cmac_id)
```

This procedure frees all storage associated with the specified CMAC, and makes the handle invalid. The procedure returns TRUE if successful and FALSE if not.

```
int train_cmac(int cmac_id,int *state,int *respns,int beta1,int beta2)
```

This procedure is used to train a previously allocated CMAC. The first parameter is the CMAC handle. The second parameter is a pointer to the vector containing the CMAC training input. The third parameter is a pointer to a vector containing the target (desired) response. The last two parameters set the two training gains of equation 19. However, the training gain parameters in the procedure call are right shift factors (*beta1* = 1 means $\beta_1 = 0.5$, *beta1* = 2 means $\beta_1 = 0.25$, etc.). The procedure returns TRUE if successful and FALSE if not.

```
int adjust_cmac(int cmac_id,int *state,int *drespns,int beta1)
```

This procedure is similar to *train_cmac()* except that the third parameter is a pointer to a vector containing the training error signal, rather than the desired response. In other words, *train_cmac()* is used to train the CMAC by supplying a particular desired response, while *adjust_cmac()* is used to train the CMAC by supplying how you would like the current response to change. The procedure returns TRUE if successful and FALSE if not.

```
int cmac_response(int cmac_id,int *state,int *respns)
```

This procedure returns the CMAC vector response to the input vector specified. The first parameter is the CMAC handle. The second parameter is a pointer to the vector containing the CMAC input. The third parameter is a pointer to a vector to receive the CMAC response. The procedure returns TRUE if successful and FALSE if not.

```
int clear_cmac_weights(int cmac_id)
```

This procedure sets the weights of the specified CMAC to 0. All subsequent CMAC responses will be 0 until further training occurs. The procedure returns TRUE if successful and FALSE if not.

```
int cmac_memory_usage(int cmac_id)
```

This procedure returns the number of CMAC weight vectors which have been modified by training since the last call to *clear_cmac_weights()*.

CMAC Support Procedures

```
int save_cmac(int cmac_id,char *filename)
```

This procedure stores all information about the state of a CMAC in a file. It is most useful for storing previously trained information. The first parameter is the CMAC handle. The second parameter is a pointer to a file name string. The procedure returns TRUE if successful and FALSE if not.

```
int restore_cmac(char *filename)
```

This procedure restores a CMAC from a file created by a prior call to *save_cmac()*. It is most useful for restoring previously trained information. The single parameter is a pointer to a file name string. The procedure returns the handle to a newly allocated CMAC if successful, otherwise it returns 0.

```
int get_cmac(int cmac_id,int index,int *buffer,int count)
```

This procedure returns raw CMAC weight vectors from the physical memory. It is useful for saving trained information or gathering statistics about weight magnitudes, for example. Each weight vector is of size *num_resp* + 1 and includes one integer weight for each component of the output vector, plus one extra integer value used in address hashing. The first parameter is the CMAC handle. The second parameter is an index (0 to M-1) into the physical weight vector memory (an A' address). The third parameter is a pointer to a vector of size *count*(num_resp + 1)* which receives the CMAC weights. The fourth parameter is the count of contiguous weight vectors to transfer. The procedure returns the number of weight vectors actually transferred.

```
int put_cmac(int cmac_id,int index,int *buffer,int count)
```

This procedure stores raw CMAC weight vectors in the physical memory. It is useful for restoring previously trained information or directly modifying/corrupting weights, for example. Each weight vector is of size *num_resp* + 1 and includes one integer weight for each component of the output vector, plus one extra integer value used in address hashing. The trailing hashing value should never be modified by application code (the correct way to overwrite a weight vector is to use *get_cmac()* to get the weights and hashing value, then change the weights, and finally use *put_cmac()* to write the new weights with the original hashing value). The first parameter is the CMAC handle. The second parameter is an index (0 to M-1) into the physical weight vector memory (an A' address). The third parameter is a pointer to a vector of size *count*(num_resp + 1)* which contains the CMAC weights. The fourth parameter is the count of contiguous weight vectors to transfer. The procedure returns the number of weight vectors actually transferred.

```
int set_cmac_rf_displacement(int cmac_id,int *buffer)
```

This procedure sets the receptive field mapping displacement vector (D in equation 7) for use by a *CUSTOM* CMAC. The first parameter is the CMAC handle. The second parameter is a pointer to a vector of size N which contains the displacement vector. The procedure returns TRUE if successful and FALSE if not.

```
int set_cmac_rf_magnitude(int cmac_id,int *buffer)
```

This procedure sets the receptive field sensitivity function look-up table for use by a *CUSTOM* CMAC. The first parameter is the CMAC handle. The second parameter is a pointer to a vector of size RF_TABLE_SIZE which contains the table. The procedure returns TRUE if successful and FALSE if not.

```
int map_cmac_input(int cmac_id,int *state,int *weights[], int *rfmags)
```

This procedure returns internal CMAC mapping information. The first parameter is the CMAC handle. The second parameter is a pointer to the vector containing the CMAC input. The third parameter is a pointer to a vector of size C which in turn receives pointers to the weight vectors mapped to by the input vector (the A'_i in the equations of the previous sections). The fourth parameter is also a pointer to a vector of size C which receives the magnitudes of the receptive field sensitivity functions for the mapped receptive fields (the $f(\delta_i)$ in equations 13 and 14). The procedure returns TRUE if successful and FALSE if not.

AN OUTLINE OF APPLICATIONS AND THEORY LITERATURE

Over the years there have been a large number of attempts to use CMAC for real applications, in simulated applications, and in demonstration projects. Below is a listing of as

many of those applications as we have found. There are others that we know of but have been unable to obtain, and there have undoubtedly been some oversights (our apologies!). These are organized by application with a listing of the numbers from the papers in the Bibliography at the end of the chapter. Headings for theory papers and hardware papers are also included.

Control (including robotics)

- Kinematics [10, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]
- Dynamics [30, 33, 34, 36, 41, 42, 43, 44, 45]
- Unstable plant [34]
- Time delay in plant [46]
- Adaptive critic [47, 48, 49, 50]
- Walking (biped and quadruped) [35, 36, 38, 43, 44, 51]
- Dynamic programming [52]
- Chemical systems [9, 53, 54, 55, 56, 57]
- Optimal [52, 58, 59]
- Mobile [60]
- Manipulator/robot [5, 10, 30, 31, 32, 33, 45, 61, 62, 63, 64, 65, 66]
- Fuzzy [27, 28, 67, 68, 69]

Manufacturing/CIM/tool fault [39, 70, 71, 72]

Pattern recognition

- Nearest neighbor methods [73]
- Character recognition [23, 74]
- Handwriting recognition [74]

Signal processing [23, 75, 76, 77]

Biomedical [2, 68, 77, 78, 79, 80, 81]

Others

- Physics detectors [82]
- Geophysical [83, 84, 85]
- Ultrasonics [86]
- Color correction [87]

Theory [12, 14, 15, 16, 25, 27, 28, 29, 57, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Hardware implementation [87, 101, 102, 103]

BIBLIOGRAPHY

(Created August, 1994.)

[1] F. Rosenblatt, Principles of neurodynamics. New York, NY: Spartan, 1962.

[2] J. S. Albus,"Theoretical and experimental aspects of a cerebellar model," PhD. Dissertation, University of Maryland, 1972.

[3] B. Widrow," Generalization and information storage in networks of adaline 'neurons'," Self-Organizing Systems, Ed. M. C. Yovits, Washington, DC: Spartan, 1962. pp. 435-461.

[4] J. S. Albus, "Data storage in the cerebellar model articulation controller," J. Dynamic Systems," Measurement and Control, pp. 228-233, 1975.

[5] J. S. Albus, "A new approach to manipulator control: the cerebellar model articulation controller (CMAC)," Trans. ASME, pp. 220-227, 1975.

[6] J. S. Albus, "Mechanisms of planning and problem solving in the brain," Mathematical Biosciences, Vol. 45, pp. pp. 247-293, 1979.

[7] J. S. Albus, Brains, behavior, and robotics. Peterborough, N.H.: Byte Books/McGraw-Hill, 1981.

[8] E. Ersu, "A learning Mechanism for an Associative Storage System," IEEE Intl'l Conference on Cybernetics and Society, Atlanta, GA, pp. 26-28, 1981.

[9] H. Tolle and E. Ersu, Neurocontrol. Berlin Heidelberg: Springer-Verlag, 1992.

[10] W. T. Miller, "A Nonlinear Learning Controller for Robotic Manipulators," Proc. of the SPIE : Intelligent Robots and Computer Vision, Vol. 726, pp. 416-423, 1986.

[11] W. T. Miller, F. H. Glanz, and L. G. Kraft, "CMAC: an associative neural network alternative to backpropagation," IEEE Proceedings, Vol. 78, pp. 1561-1567, 1990.

[12] W. T. Miller, E. An, F. H. Glanz, and M. J. Carter, "The design of CMAC neural networks for control," Adaptive and Learning Systems, New Haven, CT, Vol. 1, pp. 140-145, 1990.

[13] P.-C. E. An,"An improved multi-dimensional CMAC neural network: receptive field function and placement," PhD Dissertation, University of New Hampshire, 1991.

[14] P.-C. E. An, W. T. Miller, and P. C. Parks, "Design Improvements in Associative Memories for CMAC," ICANN '91, Helsinke, Fin., Vol. 2, pp. 1207-1210, 1991.

[15] P. C. Parks and J. Militzer, "Improved allocation of weights for associative memory storage in learning control systems," IFAC Design Methods of Control Systems, Zurich, Switzerland, pp. 507-512, 1991.

[16] S. H. Lane, D. A. Handelman, and J. J. Gelfand, "The theory and development of higher-order CMAC neural networks," IEEE Control Systems Magazine, pp. 23-30, April, 1992.

[17] T. Poggio and F. Girosi, A theory for approximation and learning, MIT AI Lab, Rept. no. No. 1140, July, 1989.

[18] J. Moody and C. Darken, "Learning with localized receptive fields," Connectionists Models Summer School, pp. 1988.

[19] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," Neural Computation, Vol. 1, pp. 281-294, 1989.

[20] A. V. Oppenheim and R. W. Schaferr, Discrete-Time Signal Processing. Englewood Cliffs, N.J.: Prentice Hall, 1989.

[21] D. Peterson and D. Middleton, "Sampling and reconstruction of wave-number limited functions in N-dimensional Euclidean spaces," Information and Control, Vol. 5, pp. 279-323, 1962.

[22] D. E. Dudgeon and R. M. Mersereau, Multidimensional Digital Signal Processing. Prentice-Hall, Inc., 1984.

[23] F. H. Glanz, W. T. Miller, and L. G. Kraft, "An overview of the CMAC neural network," IEEE Conference on Neural Networks for Ocean Engineering, Washington, DC, pp. 301-308, 1991.

[24] F. H. Glanz and J. Yang, "Experimental parameter studies for the CMAC neural network," IJCNN-91, Seattle, WA, pp. 1991.

[25] F. H. Glanz, "CMAC mechanism and behavior," Submitted to IEEE Transactions on Neural Networks, 1994.

[26] X. J. Yang,"Experimental parameter studies for the CMAC neural network," MS Thesis, University of New Hampshire, 1993.

[27] M. Brown and C. Harris, Neurofuzzy Adaptive Modelling and Control. Hemel Hempstead, UK: Prentice Hall, 1994.

[28] M. Brown,"Neurofuzzy adaptive modelling and control," PhD. Dissertation, Southhampton University, 1993.

[29] M. Brown, C. J. Harris, and P. C. Parks, "The interpolation capabilities of the binary CMAC," Neural Networks, Vol. 6, pp. pp.429-440, 1993.
[30] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft, "Real-time dynamic control of an industrial manipulator using a neural-network-based learning controller," IEEE Trans. Robotics Automat., Vol. 6, pp. 1-9, 1990.
[31] W. T. Miller, "A learning controller for nonrepetitive robotic operations," Proc. of the Workshop on Space Telerobotics, Pasadena, CA, Vol. II, pp. 273-281, 1987.
[32] W. T. Miller, "Sensor Based Control of Robotic Manipulators Using A General Learning Algorithm," IEEE Trans. Robotics Automat., Vol. RA-3, pp. 157-165, 1987.
[33] W. T. Miller, F. H. Glanz, and L. G. Kraft, "Application of a general learning algorithm to the control of robotic manipulators.," Internat. J. Robotics Research, Vol. 6, pp. 84-98, 1987.
[34] W. T. Miller and C. M. Aldrich, "Rapid learning using CMAC neural networks: real time control of an unstable system," 5th. Symposium on Intelligent Control, Philadelphia, PA, pp. 465-470, 1990.
[35] W. T. Miller III, "Real-time neural network control of a biped walking robot," IEEE Transactions on Automatic Control, 1993.
[36] W. T. Miller III, "Real-time control of a biped walking robot," World Conference on Neural Networks, Portland, OR, pp. 1993.
[37] H. Werntges, "Delta rule-based neural networks for inverse kinematics," 1990 International Joint Conference on Neural Networks, San Diego, CA, Vol. 3, pp. 415-420, 1990.
[38] Y. Lin and S.-M. Song, "Kinematic control and coordination of walking machine motion using neural networks," 1991 IEEE International Joint Conference on Neural Networks - IJCNN '91, Singapore, Singapore, pp. 248-253, 1991.
[39] Z. Geng and L. S. Haynes, "Neural network solution for the forward kinematics problem of a stewart platform," Robotics and Computer-Integrated Manufacturing, Vol. 9, pp. 485-495, 1992.
[40] Z. Geng and L. Haynes, "Neural network solution for the forward kinematics problem of a Stewart platform," 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, Vol. 3, pp. 2650-2655, 1991.
[41] R. P. Hewes and W. T. Miller, "Practical Demonstration of a Learning Control System for a Five Axis Industrial Robot," Proc. of the SPIE: Intelligent Robots and Computer Vision- Seventh in a Series, Cambridge, MA, Vol. 1002, pp. 679-685, 1989.
[42] W. T. Miller and R. P. Hewes, "Real time experiments in neural network based learning control during high speed nonrepetitive robotic operations," Proceedings of the Third IEEE International Symposium on Intelligent Control, Arlington, VA, pp. 513-518, 1988.
[43] W. T. Miller, P. J. Latham, and S. M. Scalera, "Bipedal gait adaption for walking with dynamic balance," American Control Conference, Boston, MA, pp. 1990.
[44] W. T. Miller III, "Learning dynamic balance of a biped walking robot," IEEE International Conference on Neural Networks, Orlando, Florida, Vol. 5, pp. 2771-2776, 1994.
[45] A. Eskandarian, N. E. Bedewi, B. Kramer, and A. J. Barbera, "Dynamics modeling of robotic manipulators using an artificial neural network," Journal of Robotic Systems, Vol. 11, pp. 41-56, 1994.
[46] A. V. Sebald and J. Schlenzig, "Minimax design of neural net controllers for highly uncertain plants," IEEE Transactions on Neural Networks, Vol. 5, pp. 73-82, 1994.
[47] C.-S. Lin and H. Kim, "Selection of learning parameters for CMAC-based adaptive critic learning," International Conference on Artificial Neural Networks in Engineering, pp. 153-160, 1992.
[48] C.-S. Lin and H. Kim, "CMAC-based adaptive critic self-learning control," IEEE Transaction on Neural Networks, Vol. 2, pp. 530-533, 1991.
[49] R. O. Shelton and J. K. Peterson, "Controlling a truck with an adaptive critic temporal difference CMAC design," 3rd. Workshop on Neural Networks: Academic/Industrial/NASA/Defense, Auburn, AL, SPIE Vol. 1721, pp. 195-206, 1993.
[50] R. O. Shelton and J. K. Peterson, "Controlling a truck with an adaptive critic CMAC design," Simulation, Vol. 58, pp. 319-326, 1992.
[51] W. T. Miller, P. J. Latham, and S. M. Scalera, "Bipedal gait adaption for walking with dynamic balance," American Control Conference, Boston, MA, pp. 1991.
[52] J. K. Peterson and R. O. Shelton, "Use of CMAC neural architectures in obstacle avoidance," 3rd. Workshop on Neural Networks: Academic/Industrial/NASA/Defense, Alabama, AL, Vol. 1721, pp. 187-194, 1993.
[53] E. Ersu and X. Mao, "Control of pH using a self-organizing control concept with associative memories," Int. IASTED Conf. on Applied Control and Identification, Copenhagen, Denmark, pp. 1983.

[54] E. Ersu and J. Militzer, "Real-time implementation of an associative memory-based learning control scheme for non-linear multivariable processes," 1st Measurements and Control Symposium on Applications of Multivariable Systems Techniques, Plymouth, UK, pp. 109-119, 1984.
[55] E. Ersu eds., On the application of associative neural network models to technical control problems, Springer Verlag, 1984, pp. 90-93.
[56] S. Gehlen and J. Kreuzig, "Learning by interpolating memories for modelling of fermentation processes," Advanced Control of Chemical '91, Toulouse, France, pp. 273-278, 1991.
[57] L. Xu, J.-P. Jiang, and J. Zhu, "Supervised learning control of a nonlinear polymerization reactor using the CMAC neural network for knowledge storage," IEE Proceedings:Control Theory and Application, Vol. 141, pp. 33-38, 1994.
[58] J. K. Peterson, "On-line estimation of optimal control sequences," International Conference on Artificial Neural Networks in Engineering, pp. 579-584, 1992.
[59] R. Carlson, C. Lee, and K. Rothermel, "Real time neural control of an active structure," International Conference on Artificial Neural Networks in Engineering, pp. 623-628, 1992.
[60] T. Fukuda, F. Saito, and F. Arai, "Study on the brachiation type of mobile robot (Heuristic creation of driving input and control using CMAC)," 12th. International Conference on Soil Mechanics and Foundation Engineering, Rio de Janeiro, Br., Vol. 2, pp. 478-483, 1989.
[61] W. T. Miller III, L. G. Kraft, and F. H. Glanz, "Real time comparison of neural network and traditional adaptive controllers," The Yale Conference on Adaptive Control, May 20-22,1992, Yale University, New Haven, CT, pp. 99-104.
[62] H. Kano and K. Takayama, "Learning control of robotic manipulators based on neurological model CMAC," 11th. Triennial World Congress of the International Federation of Automatic Control, Tallinn, USSR, Vol. 5, pp. 249-254, 1991.
[63] Y. Jin, T. Pipe, and A. Winfield, "Stable neural network control for manipulators," International Joint Conference on Neural Networks, Nagoya, Jpn, Vol. 3, pp. 2775-2778, 1993.
[64] Z. Geng and L. S. Haynes, "Dynamic control of a parallel link manipulator using a CMAC neural network," Computers & Electrical Engineering, Vol. 19, pp. 265-276, 1993.
[65] Z. Geng and L. S. Haynes, "Dynamic control of a parallel link manipulator using CMAC neural network," IEEE International Symposium on Intelligent Control, Arlington, VA, pp. 411-416, 1991.
[66] T.-Y. Kuc and K. Nam, "CMAC based iterative learning control of robot manipulators," 28th. IEEE Conference on Decision and Control, Tampa, FL, Vol. 3, pp. 2613-2618, 1989.
[67] G. Calcev, "Self-tuning neurofuzzy controller," IEEE International Symposium on Intelligent Control, Chicago, IL, pp. 577-580, 1993.
[68] J. Nie and D. A. Linkens, "Fuzzified CMAC self-learning controller," Second IEEE International Conference on Fuzzy Systems, San Francisco, CA, pp. 500-505, 1993.
[69] J. Ozawa, I. Hayashi, and N. Wakami, "Formulation of CMAC-fuzzy system," IEEE international Conference on Fuzzy Systems - Fuzz-IEEE, San Diego, CA, pp. 1179-1186, 1992.
[70] H. Park and H. S. Cho, "CMAC-based learning controller for pressure tracking control of hydroforming processes," Winter Annual Meeting of the American Society of Mechanical Engineers, Dallas, TX, pp. 101-106, 1990.
[71] J. Lee and B. M. Kramer, "Analysis of machine degradation using a neural network baded pattern discrimination model," Journal of Manufacturing Systems, Vol. 12, pp. 379-387, 1993.
[72] J. Lee and B. Kramer, "On-line fault monitoring and detection using an integrated learning and reasoning approach," Japan-USA Symposium on Flexible Automation, San Francisco, CA, Vol. 1, pp. 235-242, 1992.
[73] N. Ramesh and I. K. Sethi, "Nearest neighbor classification using CMAC," IEEE international Conference on Neural Networks, Orlando, Florida, Vol. 5, pp. 3061-3066, 1994.
[74] W. T. Miller III, K. F. Arehart, S. M. Scalera, and H. L. Gresham, "On-line hand-printed character recognition using CMAC neural networks," World Conference on Neural Networks, Portland, OR, July 12-15, 1993, pp. IV10-IV13.
[75] F. H. Glanz and W. T. Miller, "Deconvolution using a CMAC neural network," Proc. of the First Annual Meeting of the International Neural Network Society, Boston, MA, pp. 440, 1988.
[76] F. H. Glanz and W. T. Miller, "Deconvolution and nonlinear inverse filtering using a neural network," International Conference on Acoustics and Signal Processing, Glasgow, Scotland, Vol. 4, pp. 2349-2352, 1989.
[77] E. Wilson and J. LaCourse, "Analyzing biological signals with CMAC, a neural network," 1991 IEEE 17th, Annual Northeast Bioengineering Conference, Hartford, CT, pp. 3-4, 1991.

[78] S. Gehlen, M. Hormel, and S. Bohrer, "A learning control scheme with neuron-like associative memories for the control of biotechnological processes," neural networks, Nimes, France, pp. 1988.
[79] D. Bergantz and H. Barad, "Neural network control of cybernetic limb prostheses," Annual International Conference of the IEEE Engineering in Medicine and Biology Society, New Orleans, LA, Vol. 3, pp. 1486-1487, 1988.
[80] A. V. Sebald, C. A. Sebald, and J. Schlenzig, "Use of neural net control strategies in difficult adaptive control problems closed loop control of drug infusion," 23rd. Annual Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, Vol. 1, pp. 342-345, 1989.
[81] D. J. Wasser, D. W. Hislop, and R. N. Johnson, "Evaluation of a neural network for fault-tolerant, real-time, adaptive control," Images of the Twenty-first Century -11 Annual International Conference of the IEEE Engineering in Medicine and Biology, Seattle, WA, pp. 2027-2028, 1989.
[82] G. Simpson and K. Reinhard, A new approach to event location, UNH - GRO-Comptel Group, Rept. no. COM-TN-UNH-F70-044, June 9, 1988.
[83] A. Hagens and J. H. Doveton, "Application of a simple cerebellar model to geologic surface mapping," Computers & Geosciences, Vol. 17, pp. 561-567, 1991.
[84] G. Simpson and K. Li, Artificial neural networks: solutions to problems in remote sensing, Earth Observation Sciences, Ltd (EOS), Rept. no. EOS-92/00(16000)-RP-001, March 1993.
[85] D. Verrall and G. Simpson, Neural networks for meteosat cloud classification, Earth Observation Sciences, Ltd. (EOS), Rept. no. EOS-92/078-RP-001, Oct. 1992.
[86] Daarla and Zhao, "A learning algorithm for a CMAC-based system and its application to classification of ultrasonic signals," Ultrasonics, Vol. 32, pp. 91-98, 1994.
[87] R.-C. Wen, et al., "A CMAC neural network chip for color correction," IEEE International Conference on Neural Networks, Orlando, Florida, Vol. 3, pp. 1943-1948, 1994.
[88] J. S. Albus, "A theory of cerebellar functions," Mathematical Biosciences, Vol. 10, pp. 25-61, 1971.
[89] E. Ersu and H. Tolle, "A new concept for learning control inspired by brain theory," FAC 9th World Congress,, Budapest, Hungary, pp. 1984.
[90] E. Ersu and H. Tolle, "Hierarchical Learning Control--An Approach with Neuron-Like Associative Memories," IEEE Conference on Neural Information Processing Systems, Denver, CO, pp. 1988.
[91] D. Ellison, "On the convergence of the multidimensional Albus perceptron," The Intermational Journal of Robotics Research, Vol. 10, pp. 338-357, 1991.
[92] Y.-F. Wong, "CMAC learning is governed by a single parameter," IEEE International Conference on Neural Networks, San Francisco, Vol. 1, pp. 1439-1443, 1993.
[93] N. E. Cotter and T. J. Guillerm, "The CMAC and a theorem of Kolmogorov," Neural Networks, Vol. 5, pp. 221-228, 1992.
[94] M. Brown and C. J. Harris, "The modelling abilities of the binary CMAC," IEEE international Conference on Neural Networks, Orlando, Florida, Vol. 3, pp. 1335-1339, 1994.
[95] S. Yao and Z. Bo, "Learning convergence of CMAC in cyclic learning," International Joint Conference on Neural Networks, Nagoya, Jpn, Vol. 3, pp. 2583-2586, 1993.
[96] Y. Jin, A. G. Pipe, and A. Winfield, "Stable neural control of discrete systems," IEEE international Symposium on Intelligent Control, Chicago, IL, pp. 110-115, 1993.
[97] J. Moody and C. Darken, "Speedy alternatives to back propagation," International Neural Network Society First Annual Meeting, Boston, MA, pp. 202, 1988.
[98] N. E. Cotter and O. N. Mian, "A pulsed neural network capable of universal approximation," IEEE Transactions on Neural Networks, Vol. 3, pp. 308-314, 1992.
[99] S. Lane, D. Handelman, and J. J. Gelfand, "Higher-order CMAC neural networks- theory and practice," American Control Conference, Boston, MA, Vol. 2, pp. 1579-1585, 1991.
[100] E. Ersu and H. Tolle eds., Learning control structures with neuron-like associative memory systems, VCH Verlagsgesellschaft mbH, pp. 417-438, 1988.
[101] W. T. Miller, B. A. Box, E. C. Whitney, and J. M. Glynn, "Design and implementation of a high speed CMAC neural network using programmable logic cell arrays." In Advances in Neural Information Processing Systems 3, edited by R.P. Lippmann, J.E. Moody, and D.S. Touretzky. Morgan Kaufmann, San Mateo, CA, pp. 1022-1027, 1991.
[102] B. Yang, "A VLSI Implementation of the CMAC Neural Network." MS Thesis, University of New Hampshire, 1992.

[103] A. Kolez and N. M. Allinson, “Realisation of a modified CMAC architecture using reconfigurable logic devices.” 3rd Workshop on Neural Networks: Academic/Industrial/NASA/Defense, Auburn, AL, SPIE Vol. 1721, pp. 195-206, 1993.