Chapter 5

Theory of the FOX controller

5.1 Introduction

This chapter describes a CMAC-based adaptive controller called FOX[1]. FOX will learn to generate control signals that minimize an "error" function of the system's behavior. FOX uses a gradient descent approach to perform the optimization. As shown in Appendix B, some standard gradient descent training methods on adaptive dynamic systems can require a large amount of time or memory space. In contrast, the FOX algorithm is not much slower than the basic CMAC algorithm (per controller time step).

FOX implements eligibility-based reinforcement learning [10] because it maintains auxiliary values for each CMAC weight which measure the eligibility of that weight for modification. Previous CMAC-based reinforcement learning techniques [67] have been slow, and there is little guidance available for choosing their eligibility update equations [68]. In contrast, the FOX algorithm's eligibility update equations are formally derived in section 5.4.

The description of FOX below is motivated by considering the feedback-error (FBE) controller. The standard FBE control scheme described in Chapter 4 requires a reference trajectory which must be both continuous and coincident with the system's starting state. It has already been seen that if these requirements are not met then over-training results and the system becomes unstable. It would be good to overcome this problem for two reasons. First, being able to guarantee stability in a wider range of cases implies more robust control. Second, the reference trajectory limitations mean that FBE only solves half of the intelligent control problem. To explain this, if the system's starting state is arbitrary (or unknown), then a continuous reference trajectory must be generated to get the system from where it is to the desired state. Generating such a trajectory analytically, say as the path that optimizes some error criterion, is the other half of the problem. Ideally the designer would like to specify a single reference trajectory and then have the system converge to it in some well defined manner. Reference trajectory discontinuities should also be handled this way.

One solution to this problem, output limiting, has already been explored. This chapter will first look at another solution, called "weight eligibility". A hand-waving analysis of this scheme will demonstrate its advantages, then it will be shown how it can be implemented in the CMAC. Then the FOX algorithm will be formally derived by considering how to optimize an error function of the controlled system's inputs and outputs. This derivation is not based on FBE, but it will be shown that the resulting FOX system has many similarities to the FBE-plus-eligibility controller. It will also be shown that straight

[1]What does 'FOX' mean? Well, I thought it was a **F**airly **O**bvious e**X**tension to the CMAC.

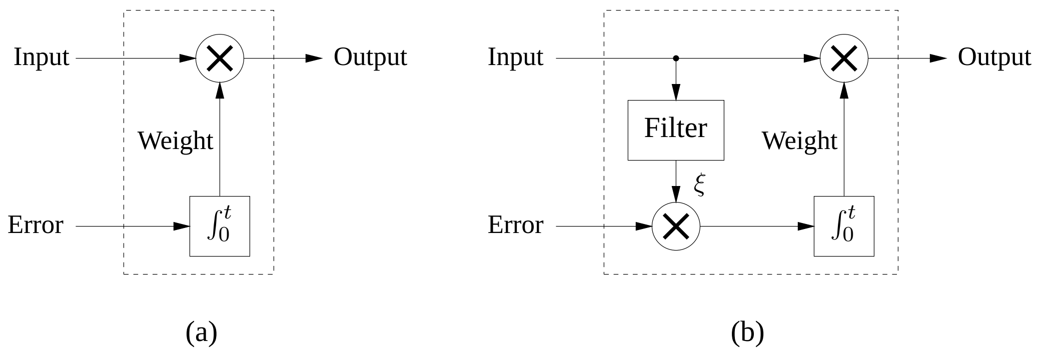


Figure 5.1: *(a) A normal neural network "synapse". (b) A modified synapse which has weight eligibility ξ.*

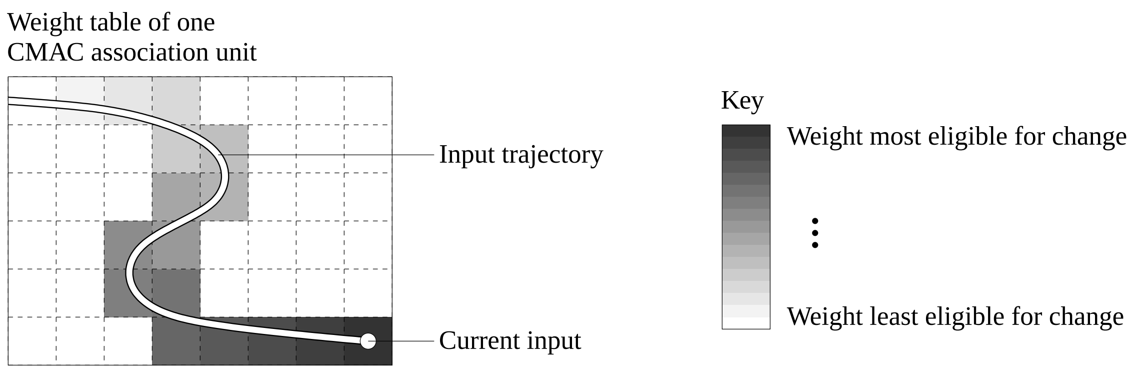


Figure 5.2: *How eligibility is implemented in a two-input binary CMAC.*

FBE is unable to control many kinds of systems. An efficient implementation for the FOX algorithm will be described which avoids having to update every weight's eligibilities after each CMAC mapping.

The derivation of the FOX algorithm is based in part on the author's previous work: [113] for the unfolded linear approximations, and [115] for some of the CMAC-with-eligibility concepts.

5.2 FBE and weight eligibility

Eligibility is an idea that has been used for many years as part of the reinforcement learning paradigm [72]. The basic idea is that each weight in a neural network is given an associated value called its "eligibility". When the weight is used (i.e. when its value makes a contribution to the network output) the eligibility is increased. Thereafter it decays toward zero. Figure 5.1a shows a normal neural network "synapse" in which a weight multiplies some input to get some output. The weight is usually the integrated value of some error signal, which is somehow derived from the global error signals given to the network. Figure 5.1b shows how the synapse is modified for eligibility. The input is filtered to obtain an eligibility signal ξ, which is multiplied by the error and the result integrates the weight. Many different forms of the eligibility filter have been proposed to solve specific problems — see [72] for a review — but it is usually a low pass filter of some description.

Figure 5.2 shows how eligibility is implemented in a two-input binary CMAC. In a binary CMAC all

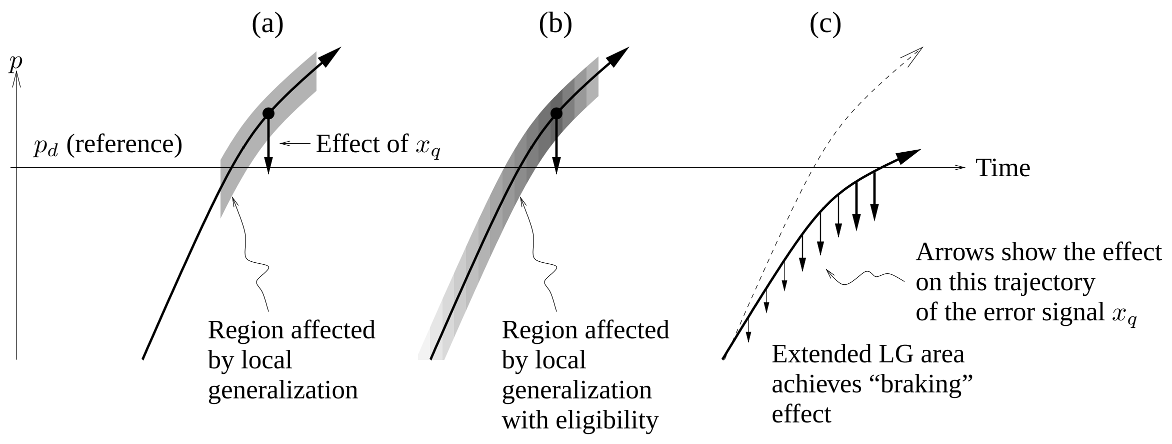

(a)
(b)
(c)
p
p_d (reference)
Effect of x_q
Time
Region affected by local generalization
Region affected by local generalization with eligibility
Arrows show the effect on this trajectory of the error signal x_q
Extended LG area achieves "braking" effect

Figure 5.3: *For the example FBE system, how adding eligibility to the standard CMAC local generalization increases the region over which x_q acts, achieving a "braking" effect which helps to prevent instability.*

the synapse inputs will be either zero or one. All weights along the input trajectory will have some degree of eligibility. Training should change all eligible weights by an amount proportional to their eligibility, i.e. the error signal is only effective at integrating the weight while the eligibility is nonzero (while the input is one and for some time thereafter).

Consider the simple second order FBE example system presented in the previous chapter. Eligibility can improve the system in two ways. First, it can increase the system stability (reduce the oscillations) by increasing the region over which x_q acts, achieving a "braking" effect. Figure 5.3a shows the system state approaching and overshooting the reference. At any point the feedback-error signal x_q influences a small area of the trajectory corresponding to the CMAC local generalization (LG) region. Figure 5.3b shows that this area expands when eligibility is used, because weights influencing the older trajectory have remained eligible for modification by the error signal. The result is that on the next training iteration the braking force arising from x_q is able to be applied earlier, which is more appropriate for reducing the overshoot.

The second improvement is that cause and effect are better associated in the system. This is demonstrated in figure 5.4. Curve 1 shows a position reference with a step change. Curve 2 is a typical FBE response (when output limiting is used). Local generalization allows the system state to slightly anticipate the reference change. Curve 3 shows what can happen if eligibility is used: the system state may be able to greatly anticipate the change, because the training induced by the error signal affects the system further back in time. This helps to reduce the error between the actual and reference trajectories.

5.3 Introduction to FOX

The basic discrete-time FOX system shown in figure 5.5 will now be considered. This consists of a process F being driven by a CMAC controller. The block F is the controlled system (or "plant") that implements

$$\mathbf{y}_{i+1} \quad = \quad F(\mathbf{y}_i, \mathbf{x}_i) \qquad (5.1)$$

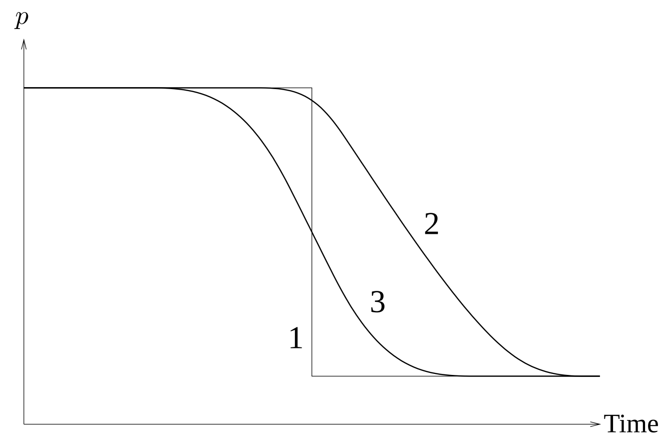


Figure 5.4: *For the example FBE system, how eligibility can give better prediction.*

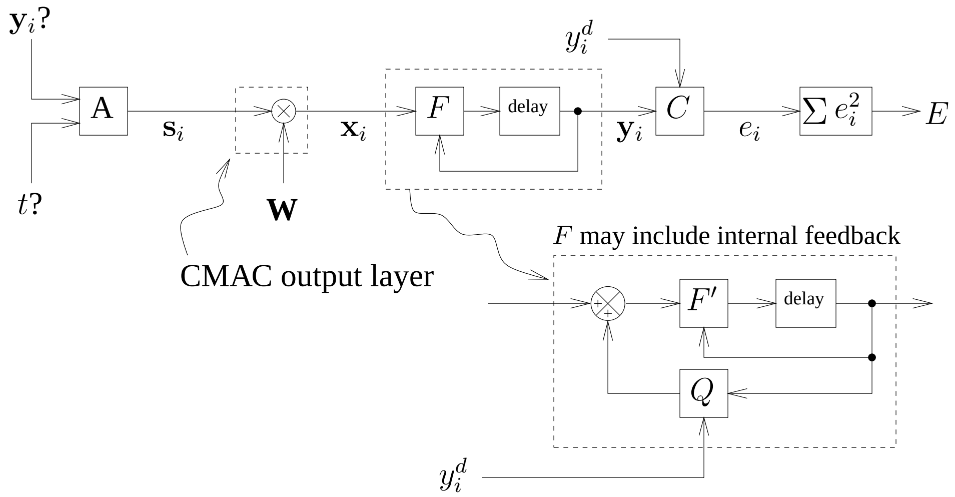


Figure 5.5: *The FOX controller and the controlled system.*

Symbol	Type	Size	Description
$F(\mathbf{y}_i, \mathbf{x}_i)$	f	–	The system function, which takes the current state and control input and generates the next time step's state.
$C(\mathbf{y}_i)$	f	–	The "critic" function, which generates a scalar error value at each time step.
$\mathbf{y}_i$	v	$n_y \times 1$	The system state vector for time i.
$\mathbf{x}_i$	v	$n_x \times 1$	The system control input vector for time i (the output of the CMAC controller).
$\mathbf{s}_i$	v	$n_s \times 1$	A vector of CMAC association-unit "sensors" that encodes the current system state.
W	m	$n_x \times n_s$	The matrix of CMAC weights. Note that $\mathbf{x}_i = W\,\mathbf{s}_i$.
e_i	s	1×1	A scalar error value generated by the critic function for time i.
y_i^d	s	1×1	A position reference (for time i) used in the calculation of e.
E	s	1×1	The total error of the entire system.
w_{pq}	s	1×1	Element (p,q) of the matrix W.
ξ_i^{pq}	v	$n_y \times 1$	The eligibility value for weight w_{pq}.
$\hat{\mathbf{s}}_i^{pq}$	v	$n_x \times 1$	A synonym for $\mathrm{d}\mathbf{x}_i/\mathrm{d}w_{pq}$. $\hat{\mathbf{s}}_i^{pq}$ is all zero except for its p'th element which is equal to the q'th element of $\mathbf{s}_i$.
n_w	s	1×1	Total number of weights, $n_w = n_s\, n_x$.
n_a	s	1×1	Number of association units in the CMAC, i.e. the number of nonzero elements in $\mathbf{s}_i$.
A	m	$n_y \times n_y$	A system matrix for the linear system F^*.
B	m	$n_y \times n_x$	A system matrix for the linear system F^*.
C	m	$1 \times n_y$	The matrix for the simple linear critic function.
α	s	1×1	The main FOX learning rate.

Table 5.1: *Definitions of some symbols in figure 5.6 and elsewhere in this chapter. This is a subset of the symbols defined in the nomenclature section of this thesis. The types are function* (f)*, vector* (v)*, matrix* (m) *and scalar* (s)*.*

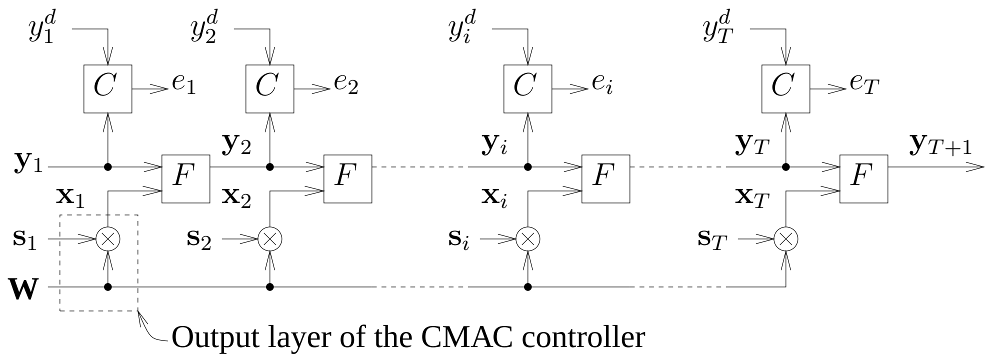


Figure 5.6: *Figure 5.5 unfolded over time.*

As shown in the figure the block F can also incorporate any additional feedback controller Q that the basic system might have (this is equivalent to the feedback controller in the FBE control scheme). The CMAC has been split into two parts in the figure:

1. The block A which represents the input layers (or association unit transformations). At each time step this block reads the system state $\mathbf{y}$ (or perhaps the current time t) and encodes it in the "sensor" vector $\mathbf{s}$.

2. The output layer which multiplies the vector $\mathbf{s}$ with the weight matrix W to get the output $\mathbf{x}$ ($\mathbf{x} = W\,\mathbf{s}$).

At each time step the "critic" block C computes a scalar error value e_i that depends on the current state. The squares of these error values are summed over time to get the global error value E. The goal of the FOX algorithm is to adjust the weights W such that the error E is minimized—in other words, this is an optimal control problem.

Figure 5.6 shows the same system unfolded over time steps $1 \ldots T$ (the symbols used are defined in table 5.1, also see the nomenclature section of this thesis). This is the figure that will be referred to in the following derivations.

Note that the CMAC controller has no internal state variables. Controller state can give extra flexibility, but for the moment it will be assumed that any extra dynamics are incorporated into the F block.

5.3.1 The critic function

The total error E is given by

$$E = \sum_{i=1}^{T} e_i^2 \tag{5.2}$$

The critic function is chosen so that when E is minimized the system achieves some desired behavior. It will be assumed henceforth that the desired behavior is to track a scalar reference position y^d, thus:

$$e_i = C\,\mathbf{y}_i - y_i^d \tag{5.3}$$

Here C is a $1 \times n_y$ matrix that selects whatever element of y corresponds to a position. This style of error function has several limitations. It can not be used if more than one component of $\mathbf{y}$ is required to follow a reference. It also does not limit the control force $\mathbf{x}$. These limitations will be relaxed later when the fundamental FOX algorithm has been developed.

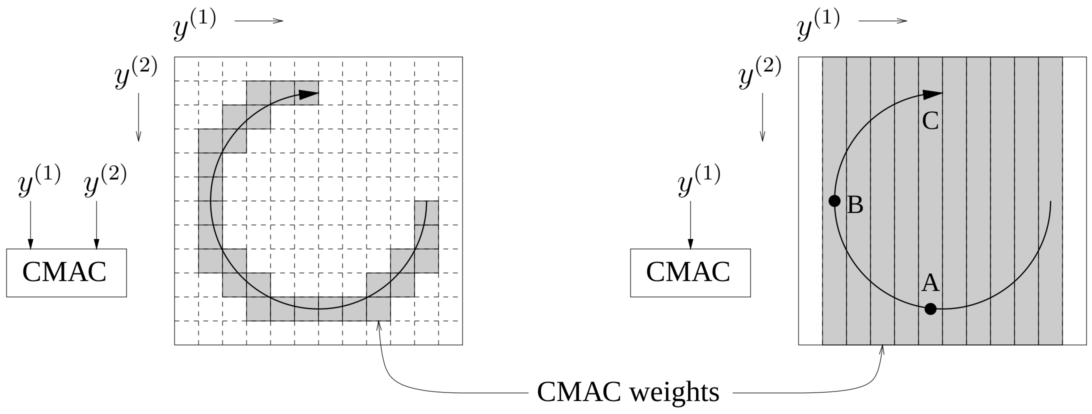


Figure 5.7: *How the sensor vectors can become correlated if the CMAC input does not include enough information.*

5.3.2 The sensor vector and CMAC inputs

For the binary CMAC the sensor vector components are zero or one. In one sense, the ideal set of sensor vectors $\mathbf{s}_i$ would be such that each one selected different columns of the matrix W. This would mean that a different set of weights controls each $\mathbf{x}_i$, so the training process has the greatest amount of flexibility when converging to an optimal control strategy, i.e. independent outputs can be achieved for each time step.

In practice the sensor vectors will usually have overlapping components from one time step to the next, so the training flexibility is reduced, i.e. the sensors $\mathbf{s}$ will not perfectly distinguish the current state from the others. This effect can be an advantage, because the CMAC output will usually be smoother (less varying), thus the CMAC does not learn unjustifiably complex control outputs, which can improve the system's controllability.

In terms of training it makes little difference whether the CMAC input is derived from the current state $\mathbf{y}$, the desired state $\mathbf{y}^d$, or the time t. All that is required is that, for the expected system trajectories, the input is sufficiently variable that the sensor vectors are not overly correlated from one time step to the next. Thus the system designer may select the CMAC input based on other considerations. If the current state $\mathbf{y}$ is used the FOX system is said to be operating in *feedback mode*, and if $\mathbf{y}^d$ or t are used it is said to be operating in *feed-forward mode*.

Figure 5.7 shows what *not* to do. The diagram on the left shows an AU table of a two-input CMAC. At each part of the example trajectory different weights are selected, so that the CMAC output is sufficiently flexible. The diagram on the right shows the AU table when only one of the inputs is used. The two parts of the trajectory between A–B and B–C use the same weight values, so training will not be able to generate independent outputs for them.

5.4 Derivation of the training algorithm

Now the FOX training algorithm will be derived. The purpose of the algorithm is to modify W using gradient descent so that the error E is minimized[2]. This section has a lot of equations, with little

[2]It is instructive to compare this derivation to the dynamic programming solution to the optimal control problem (see Appendix D).

explanation—the next section will say what it all means. The gradient descent process for this system is:

$$w_{pq} \quad \leftarrow \quad w_{pq} - \alpha \frac{\mathrm{d}\, E}{\mathrm{d}\, w_{pq}} \tag{5.4}$$

where w_{pq} is an element of the matrix W and α is a scalar learning rate. This equation gives the modification made to the weight w_{pq} as a result of one iteration of learning. Now, from the chain rule:

$$\frac{\mathrm{d}\, E}{\mathrm{d}\, w_{pq}} \quad = \quad \sum_{i=1}^{T-1} \frac{\mathrm{d}\, E}{\mathrm{d}\, \mathbf{x}_i} \cdot \frac{\mathrm{d}\, \mathbf{x}_i}{\mathrm{d}\, w_{pq}} \tag{5.5}$$

$$\quad = \quad \sum_{i=1}^{T-1} \left[2 \sum_{k=i+1}^{T} e_k \frac{\mathrm{d}\, e_k}{\mathrm{d}\, \mathbf{x}_i} \right] \frac{\mathrm{d}\, \mathbf{x}_i}{\mathrm{d}\, w_{pq}} \tag{5.6}$$

Note that $(T-1)$ is the limit on the outer sum, not T, because $\mathbf{x}_T$ has no effect on e_T. $\mathrm{d}e_k/\mathrm{d}\mathbf{x}_i$ can be calculated using forward propagation, i.e.,

$$\frac{\mathrm{d}\, e_{i+1}}{\mathrm{d}\, \mathbf{x}_i} \quad = \quad \frac{\partial\, e_{i+1}}{\partial\, \mathbf{y}_{i+1}} \cdot \frac{\partial\, \mathbf{y}_{i+1}}{\partial\, \mathbf{x}_i} \tag{5.7}$$

$$\text{and} \qquad \frac{\mathrm{d}\, e_{i+2}}{\mathrm{d}\, \mathbf{x}_i} \quad = \quad \frac{\partial\, e_{i+2}}{\partial\, \mathbf{y}_{i+2}} \cdot \frac{\partial\, \mathbf{y}_{i+2}}{\partial\, \mathbf{y}_{i+1}} \cdot \frac{\partial\, \mathbf{y}_{i+1}}{\partial\, \mathbf{x}_i} \tag{5.8}$$

$$\text{so} \qquad \frac{\mathrm{d}\, e_{i+k}}{\mathrm{d}\, \mathbf{x}_i} \quad = \quad \frac{\partial\, e_{i+k}}{\partial\, \mathbf{y}_{i+k}} \left[\frac{\partial\, \mathbf{y}_{i+k}}{\partial\, \mathbf{y}_{i+k-1}} \cdots \frac{\partial\, \mathbf{y}_{i+2}}{\partial\, \mathbf{y}_{i+1}} \right] \cdot \frac{\partial\, \mathbf{y}_{i+1}}{\partial\, \mathbf{x}_i} \qquad (k > 0) \tag{5.9}$$

Equation 5.6 is transformed into a form suitable for online training (i.e., incremental accumulation of the outer sum) by reversing the order of summation (swapping the variables i and k):

$$\frac{\mathrm{d}\, E}{\mathrm{d}\, w_{pq}} \quad = \quad 2 \sum_{k=2}^{T} e_k \left[\sum_{i=1}^{k-1} \frac{\mathrm{d}\, e_k}{\mathrm{d}\, \mathbf{x}_i} \frac{\mathrm{d}\, \mathbf{x}_i}{\mathrm{d}\, w_{pq}} \right] \tag{5.10}$$

Now for a key step—to determine the meaning of equation 5.10, and to derive a practical algorithm, F is *approximated* by a linear system F^*:

$$\mathbf{y}_{i+1} \quad = \quad A\, \mathbf{y}_i + B\, \mathbf{x}_i \tag{5.11}$$

$$e_i \quad = \quad C\, \mathbf{y}_i - y_i^d \tag{5.12}$$

Of course if F is already linear then this can be an exact model. Combining this with equation 5.9:

$$\frac{\mathrm{d}\, e_{i+k}}{\mathrm{d}\, \mathbf{x}_i} \quad = \quad C\, A^{k-1}\, B \qquad (k > 0) \tag{5.13}$$

thus

$$\frac{\mathrm{d}\, E}{\mathrm{d}\, w_{pq}} \quad = \quad 2 \sum_{k=2}^{T} e_k\, C \left[\sum_{i=1}^{k-1} A^{k-i-1}\, B\, \hat{\mathbf{s}}_i^{pq} \right] \tag{5.14}$$

$$\quad = \quad 2 \sum_{k=2}^{T} e_k\, C\, \xi_k^{pq} \tag{5.15}$$

If $\mathbf{y}_1 = 0$ and the values of $\mathbf{x}_1 \ldots \mathbf{x}_{k-1}$ are known, and $\mathbf{y}_{i+1} = A\,\mathbf{y}_i + B\,\mathbf{x}_i$, then $\mathbf{y}_k$ can be calculated as follows:

$$\begin{aligned}
\mathbf{y}_1 &\triangleq 0 \\
\mathbf{y}_2 &= B\,\mathbf{x}_1 \\
\mathbf{y}_3 &= AB\,\mathbf{x}_1 + B\,\mathbf{x}_2 \\
\mathbf{y}_4 &= A^2B\,\mathbf{x}_1 + AB\,\mathbf{x}_2 + B\,\mathbf{x}_3 \\
&\vdots \\
\mathbf{y}_k &= A^{k-2}B\,\mathbf{x}_1 + A^{k-3}B\,\mathbf{x}_2 + \cdots + AB\,\mathbf{x}_{k-2} + B\,\mathbf{x}_{k-1} \\
&= \sum_{i=1}^{k-1} A^{k-i-1} B\,\mathbf{x}_i
\end{aligned}$$

If $\mathbf{x}_i$ is replaced with $\hat{\mathbf{s}}_i^{pq}$ then $\mathbf{y}_k$ is equal to ξ_k^{pq}.

Table 5.2: *Proof that ξ_k^{pq} is subject to the same dynamics as F^*.*

where

$$\xi_k^{pq} = \sum_{i=1}^{k-1} A^{k-i-1}\,B\,\hat{\mathbf{s}}_i^{pq} \qquad \text{and} \qquad \hat{\mathbf{s}}_i^{pq} = \frac{\partial\,\mathbf{x}_i}{\partial\,w_{pq}} \tag{5.16}$$

$\hat{\mathbf{s}}_i^{pq}$ is all zero except for its p'th element which is equal to the q'th element of $\mathbf{s}_i$. ξ_k^{pq} is the *eligibility* signal. It can be shown that ξ_k^{pq} is a convolution of $\hat{\mathbf{s}}_i^{pq}$ with the impulse response of the system F^*, up to time k (see table 5.2).

In other words, ξ_k^{pq} is subject to the same dynamics as the system's linear approximation, so

$$\xi_1^{pq} = 0 \tag{5.17}$$
$$\xi_{i+1}^{pq} = A\,\xi_i^{pq} + B\,\hat{\mathbf{s}}_i^{pq} \tag{5.18}$$

5.5 Discussion

5.5.1 Algorithm summary

Here is the FOX training algorithm, based on equations 5.4, 5.15 and 5.18:

$$\xi_1^{pq} = 0 \tag{5.19}$$
$$\xi_{i+1}^{pq} = A\,\xi_i^{pq} + B\,\hat{\mathbf{s}}_i^{pq} \tag{5.20}$$
$$w_{pq} \leftarrow w_{pq} - \alpha \sum_{i=2}^{T} e_i\,C\,\xi_i^{pq} \tag{5.21}$$

(note that a factor of 2 has been combined into α). Every CMAC weight w_{pq} requires an associated eligibility vector ξ^{pq}. The *order* of the eligibility model is the size of the matrix A. A zero-order model is a FOX system without any eligibility machinery at all, which is just a straight CMAC (the corresponding zero-order impulse response is just the Dirac delta function $k\delta(t)$, where k is an arbitrary scale factor).

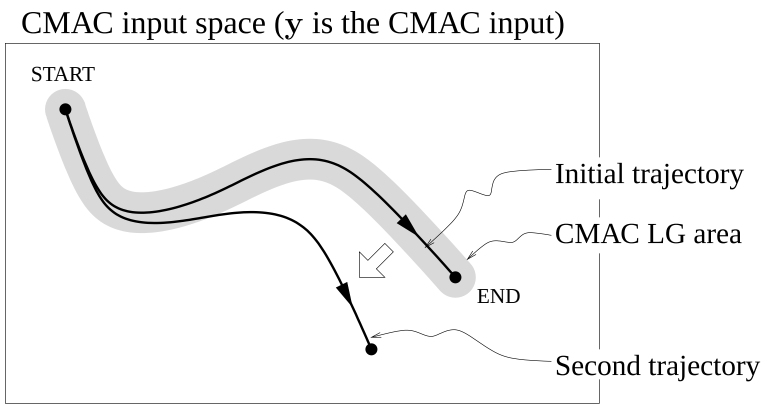


Figure 5.8: *How the* $\mathrm{d}\mathbf{x}/\mathrm{d}\mathbf{y} = 0$ *assumption can be violated in one training iteration.*

Notice that there is a relationship between the two constants α and C : if the magnitude of C is adjusted then α can be changed to compensate. Because of this the convention will be adopted that the magnitude of C is always set to one ($|C| = 1$) and then the resulting α is the main FOX learning rate.

5.5.2 Approximation: $\partial\mathbf{x}/\partial\mathbf{y} = 0$

The derivation of this algorithm has deliberately ignored the influence of the system state $\mathbf{y}$ on the CMAC output $\mathbf{x}$, in other words it has been assumed that $\partial\mathbf{x}/\partial\mathbf{y} = 0$. There are several reasons for this approximation. First, ignoring it makes the FOX algorithm simple. Second, the CMAC input is not necessarily tied to the system state, for example it could be the system time or some other sensor values. Third, the CMAC input to output mapping is discontinuous, so the value of $\partial\mathbf{x}/\partial\mathbf{y}$ is also discontinuous which makes it difficult to use in a training algorithm.

Despite this, the approximation is acceptable. Consider figure 5.8 which shows how a CMAC input-space trajectory is modified over one training iteration (assuming that $\mathbf{y}$ is the CMAC input). Training over the initial trajectory modifies the CMAC weights, so that the second trajectory is different because the CMAC outputs $\mathbf{x}$ are different (assuming the starting point is the same). The fact that a new region of the input space is being traversed will change the output $\mathbf{x}$, quite apart from the fact that the $\mathbf{x}$ values in the local generalization (LG) area of the first trajectory have been changed. This is the effect that is ignored by assuming $\partial\mathbf{x}/\partial\mathbf{y} = 0$.

The degree to which the original and modified trajectories differ depends on the learning rate α. If α is small enough that the modified trajectory is still mostly contained within the LG area of the initial trajectory, then the approximation error will be small enough not to matter. Note that the larger the learning rate, the more likely it is that the gradient descent step will actually increase the global error.

5.5.3 Approximation: instantaneous weight update

Equation 5.21 implies that a sum from times $2 \ldots T$ must be computed before the weights can be updated. This would require an auxiliary sum-accumulation variable for each weight. The following approximation will be made to remove the need for these extra variables: at each time step as each term of the sum is computed it will be added directly to the weight. This approximation is very common in neural network training algorithms, including the standard backpropagation algorithm. If α is small enough then this approximation will have a negligible effect on the result.

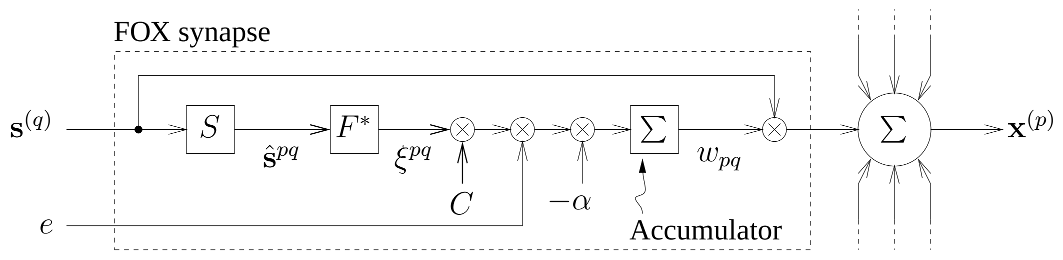


Figure 5.9: *A single "synapse" of the FOX controller, for weight w_{pq}. $\mathbf{s}^{(q)}$ is the q'th element of $\mathbf{s}$, $\mathbf{x}^{(p)}$ is the p'th element of $\mathbf{x}$.*

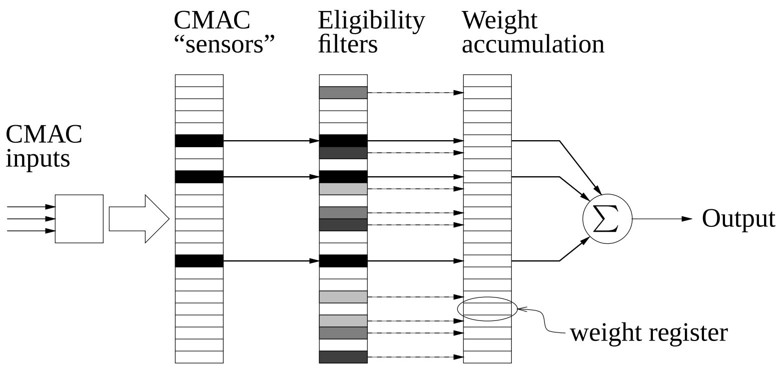


Figure 5.10: *How training is performed, conceptually.*

5.5.4 The synapse mechanism

A schematic of the process used to update a single weight is shown in figure 5.9. In this figure the block S converts the sensor vector element $\mathbf{s}^{(q)}$ into the vector $\hat{\mathbf{s}}^{pq}$ by padding it with zeros. The outputs of the active synapses (those with nonzero $\mathbf{s}^{(q)}$) are summed to produce the CMAC outputs. Compare this with figure 5.1b—it can be seen that the FOX synapse fits the traditional picture of a synapse with eligibility. Figure 5.10 shows how these synapse units are collected together in the CMAC.

5.5.5 Eligibility profile

The eligibility ξ is a vector, but the important quantity in determining the eligibility's effect on the system is the *scalar* value $C\xi$. The response of $C\xi$ to an impulse in the synapse input $\mathbf{s}^{(q)}$ is called the *eligibility profile*. It is a scalar function of time step i:

$$\text{eligibility profile } (i) = C\,A^i\,B \tag{5.22}$$

There are n_x distinct eligibility profiles in the system, one for each component of $\mathbf{x}$. The i'th eligibility profile is the error that is measured (for a zero reference) when the system F^* is excited with an impulse on input $\mathbf{x}^{(i)}$. The eligibility profiles are defined by the matrices A,B and C. They encapsulate all the information about the system dynamics that is required for training.

Many previous eligibility implementations in reinforcement learning algorithms [72] have used an arbitrarily chosen exponential-decay scalar eligibility profile (figure 5.11a). In contrast the FOX eligibility profiles are selected to match the error-impulse-response of the controlled system, and can be a

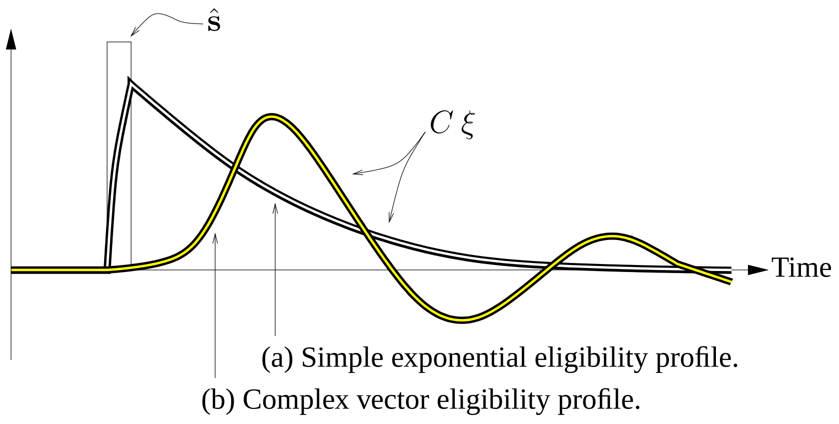

$\hat{\mathrm{s}}$
$C\,\xi$
Time
(a) Simple exponential eligibility profile.
(b) Complex vector eligibility profile.

Figure 5.11: *Some eligibility profiles.*

much more complicated function (figure 5.11b). Thus FOX allows “higher order eligibility” or “vector eligibility”.

Any alternative eligibility filter (F^*,C) that generates a similar profile will work just as well. This fact can be used to simplify F^* for complex systems, and is the basis for design techniques that will be presented later.

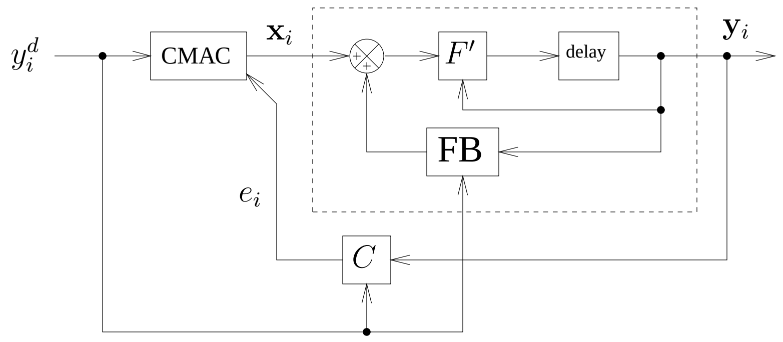
5.6 FOX and FBE

The standard feedback-error controller described in the previous chapter is very similar to a zero-order FOX system. This is demonstrated in figure 5.12 which shows a comparison between the two systems. There are a number of differences, but they are not significant:

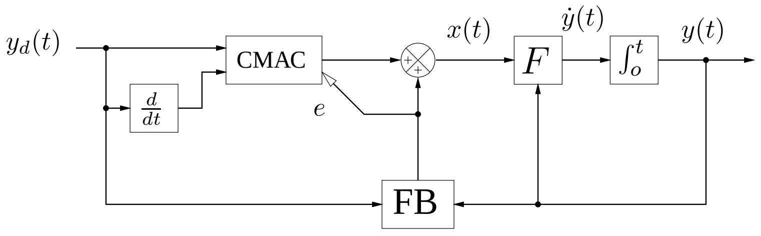
- The FOX system operates in discrete time and the FBE system operates in continuous time. But either system can be easily converted in to the other form.
- The FBE system uses the feedback signal as the error, while the FOX system generates it independently in the function C.
- The FBE CMAC inputs include $\mathrm{d}y_d(t)/\mathrm{d}t$, while the FOX CMAC input only has $\mathbf{y}_i^d$. This allows the FBE system to be trained to follow a $y_d(t)$ trajectory that is different in each iteration while the FOX system’s desired trajectory must be the same in each iteration. But either input set can be used for either system.

Thus there is no real difference between the two systems in terms of the effect that training has, so FBE can be considered to be equivalent to a zero-order FOX. This implies that FBE can only provide successful control when the system F plus its feedback controller have a zero-order impulse response $k\delta(t)$ as described above (this means that the effect of the control effort is immediately observable in the sensors).

This is generally true when FBE is being used to generate forces to control the *accelerations* of a mechanical system (as the effects of applied forces can be immediately observed in the system accelerations). Examples of FBE systems where this is true are robot arm control [87], an automatic car braking system [92], and learning nonlinear thruster dynamics to control an underwater robotic vehicle [131].



(a) A zero-order FOX system



(b) A standard feedback-error system

Figure 5.12: *Comparison of (a) a zero-order FOX system and (b) a standard feedback-error system. 'FB' is the "internal" feedback controller for (a) and the feedback controller for (b).*

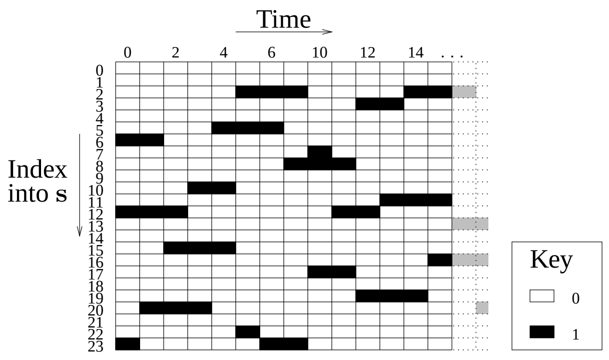


Figure 5.13: *Time evolution example for the vector* $\mathbf{s}$, $n_s = 24$ *and* $n_a = 3$.

It will be shown in the next chapter that if FBE is used to generate forces to control other parameters of a mechanical system (positions or velocities) then over-training and divergence will usually occur, manifested as system oscillations).

5.7 Implementation

A naïve implementation of the training equations is very simple—just update the eligibility state for every weight during each time step. Consider a CMAC with n_w weights ($n_w = n_x n_s$) and n_a association units. To compute the CMAC's output without training (in the conventional way) requires one set of computations per association unit, so the computation required is $\mathcal{O}(n_a)$ per time step. But if eligibilities must be updated as well then one set of computations per *weight* is needed, so the time rises by $\mathcal{O}(n_w)$. A typical CMAC has $n_w \gg n_a$ (e.g. $n_a = 10$ and $n_w = 10,000$), so the naïve approach usually requires too much computation to be practical in an online controller.

The FOX algorithm performs the same computations using a far more efficient approach which eliminates the recalculation of redundant information (in this respect it is like the Fast Fourier Transform algorithm). FOX performs just $\mathcal{O}(n_a)$ extra operations per time step, so the total computation time is still $\mathcal{O}(n_a)$—the same order as the basic CMAC algorithm.

5.7.1 Assumptions

The algorithm described below requires the system F^* to have an impulse response that eventually decays to zero. This is equivalent to requiring that the eigenvalues of A all have a magnitude less than one. This will be called the "decay-to-zero" assumption. The reasons for this assumption will be made clear later. The next chapter will explain how to get around this requirement in a practical system.

5.7.2 Operation

Figure 5.13 shows an example of how the vector $\mathbf{s}$ (the CMAC sensors) changes over time. At any time only n_a (in this case three) vector elements are nonzero. These elements correspond to the activated weights. The activated weight indexes change unpredictably as the CMAC input evolves over time. However, each element of $\mathbf{s}$ spends most of its time at zero, because $n_a \ll n_w$.

Figure 5.14 shows conceptually how an example eligibility filter responds to a typical sensor signal. When the sensor signal is one the eligibility rises, and when it is zero (which is most of the time) the

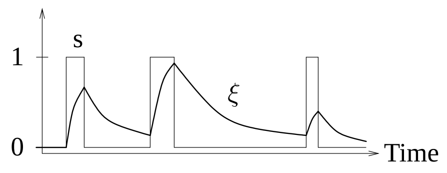


Figure 5.14: *The output (ξ) of a sample eligibility filter being driven by a sensor signal (s).*

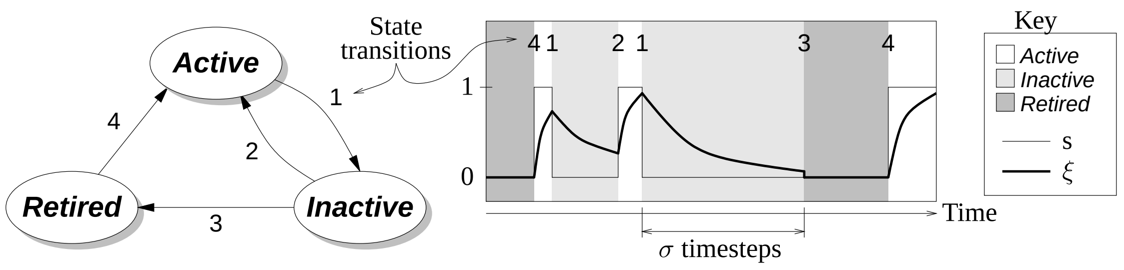


Figure 5.15: *The three states of a FOX weight, and the transitions between them.*

eligibility decays towards zero as required by the decay-to-zero assumption. This behavior inspires division of the weights into the following three categories:

- *Active weights*: where the weight is one of the n_a currently being accessed by the CMAC. There are always n_a active weights.

- *Inactive weights*: where the weight was previously active and its eligibility has yet to decay to zero.

- *Retired weights*: where the weight's eligibility has decayed sufficiently close to zero, so no further weight change will be allowed to take place until this weight becomes active again.

Figure 5.15 shows how a weight makes the transition between these different states. FOX does not have to process the retired weights because their values do not change (their eligibilities are zero and will remain that way) and they do not affect the CMAC output. An active weight turns in to an inactive weight when the weight is no longer being accessed by the CMAC (transition 1 in figure 5.15). An inactive weight turns in to a retired weight after σ time steps have gone past (transition 3 in figure 5.15). The value of σ is chosen so that after σ time steps a decaying eligibility value is small enough to be set to zero. Ways to choose σ will be discussed later. At each new time step a new set of weights are made active. Some of these would have been active on the previous time step, others are transferred from the inactive and retired states as necessary (transitions 2 and 4 respectively in figure 5.15).

The active and inactive weights have nonzero eligibility values, so it seems that ξ will need to be modified for them at each time step according to equation 5.21. However, FOX only keeps the *active* weight values up-to-date, because they are the only ones that contribute to the CMAC output. An inactive weight and its corresponding ξ are not modified until the weight becomes active or retired, whereupon they are updated to their correct values. Active weights must always have their correct value, because they contribute to the CMAC output. Retired weights must also always have their correct value, because no other information about them is stored by FOX. In a typical CMAC most weights will be retired.

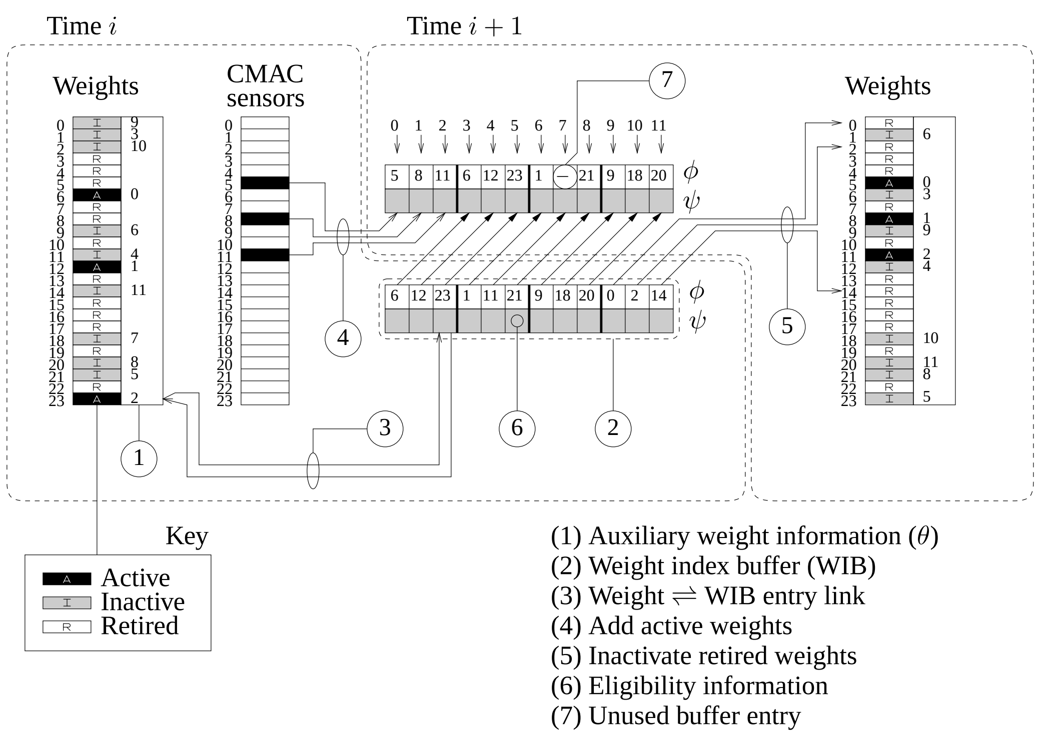

Time i
Time $i+1$
Weights
CMAC sensors
Weights
ϕ
ψ
Key
Active
Inactive
Retired
(1) Auxiliary weight information (θ)
(2) Weight index buffer (WIB)
(3) Weight $\rightleftharpoons$ WIB entry link
(4) Add active weights
(5) Inactivate retired weights
(6) Eligibility information
(7) Unused buffer entry

Figure 5.16: *Operation of the weight index buffer (WIB) for* $n_s = 24$, $n_a = 3$ *and* $\sigma = 4$.

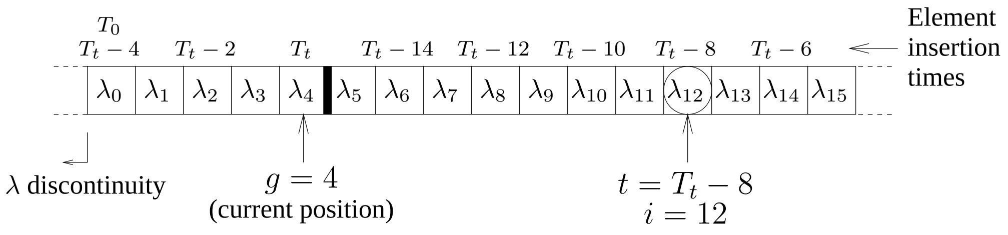
The indexes of the inactive weights (indexes into $\mathbf{s}$) are stored in a shift register called the "weight index buffer" (WIB). Figure 5.16 demonstrates the operation of the WIB from time step i to time step $i+1$. The WIB has room for σ groups of n_a weight indexes. Each WIB entry stores an index into the weight table and some information about the eligibility of that weight. All weights not represented in the WIB are assumed to have zero eligibility. Each weight has an "auxiliary" number which stores that weight's position in the WIB, if any. Thus WIB entries are linked to their corresponding weights and vice versa.

During each time step the WIB is shifted (to the right in figure 5.16). The indexes of the n_a currently active weights are shifted in from the left (5,8 and 11 in figure 5.16). The n_a indexes shifted out from the right correspond to inactive weights that are σ time steps old—these weights are retired (0,2 and 14 in figure 5.16). A particular weight index cannot appear in the WIB more than once. If a weight index shifted in is already in the WIB it will be removed from its old position and that position will be marked "unused" (this happens to index 11 in figure 5.16). This will happen if one or more of the CMAC sensor values remains constant from one time step to the next.

5.7.3 Correcting inactive weights

If a weight is made inactive at time i_1 and it becomes active or retired at time i_2, its value (and the corresponding eligibility value) must be corrected to account for the interval $i_1 \ldots i_2$ during which it was not modified. This can be done efficiently for the linear F^*, as ξ will have a sum-of-exponentials decay profile. The eligibility update equation is:

$$\xi_{i+1}^{pq} \quad = \quad A\,\xi_i^{pq} + B\,\hat{\mathbf{s}}_i^{pq} \tag{5.23}$$



T_0
$T_t - 4$
$T_t - 2$
T_t
$T_t - 14$
$T_t - 12$
$T_t - 10$
$T_t - 8$
$T_t - 6$
Element insertion times
λ_0 λ_1 λ_2 λ_3 λ_4 λ_5 λ_6 λ_7 λ_8 λ_9 λ_{10} λ_{11} λ_{12} λ_{13} λ_{14} λ_{15}
λ discontinuity
$g = 4$
(current position)
$t = T_t - 8$
$i = 12$

Figure 5.17: *An example trace buffer (WIB) with* $\delta = 16$.

If weight w_{pq} goes inactive at time step i_1 then $\hat{s}_i^{pq} = 0$ for $i \geq i_1$, so:

$$\xi_{i+1}^{pq} = A\, \xi_i^{pq}, \qquad i \geq i_1 \tag{5.24}$$

thus

$$\xi_{i_1+i}^{pq} = A^i\, \xi_{i_1}^{pq}, \qquad i \geq 0 \tag{5.25}$$

If w_{pq} is made inactive (or "frozen") at time i_1, to find its true value at time i_2 the following must be computed:

$$\begin{aligned} w_{pq} &\leftarrow w_{pq} - \alpha \sum_{i=i_1}^{i_2} e_i\, C\, \xi_i^{pq} && (5.26)\\ &= w_{pq} - \alpha \left[\sum_{i=i_1}^{i_2} e_i\, C\, A^{i-i_1} \right] \xi_{i_1}^{pq} && (5.27)\\ &= w_{pq} - \alpha\, (\lambda_{i_2} - \lambda_{i_1-1})\, A^{-i_1}\, \xi_{i_1}^{pq} && (5.28)\\ \text{where} \quad \lambda_j &= \sum_{i=1}^{j} e_i\, C\, A^i && (5.29) \end{aligned}$$

Thus if the value λ is accumulated for each time step, any inactive weight can be activated or retired with a fast calculation (equation 5.28). Values of λ must be kept for at least the last σ time steps. Note that only *one* calculation of λ has to be made for the entire system, not one per weight.

There is one remaining complication: the exponentially decreasing A^i factor in equation 5.29 causes λ to quickly converge to a steady value. As time increases, equation 5.28 multiplies an exponentially increasing value (A^{-i_1}) by an exponentially decreasing difference $(\lambda_{i_2} - \lambda_{i_1})$. The result will quickly lose numerical precision.

This can be rectified by periodically resetting λ to zero (as well as resetting the factor A^i to the identity matrix) and then accounting for the discontinuities. This idea is implemented in the "trace" algorithms shown in table 5.3. The TRACERESET procedure is first called to reset some internal variables. Then the TRACENEXT procedure is called for each time step to supply a sequence of scalar error values $e_1, e_2, \ldots e_i$. The TRACEDELTA function allows computation of the following sum without loss of numerical precision:

$$\Delta_t = \sum_{i=t}^{T_t} e_i\, C\, A^{i-t} \tag{5.30}$$

THE TRACE ALGORITHM

Parameters

n_y — Eligibility vector size (integer ≥ 1).
A, C — Matrices, of size $n_y \times n_y$ and $1 \times n_y$ respectively.
δ — Buffer size (integer ≥ 2). This equals $\sigma + 2$

Internal state variables

$\lambda_0 \ldots \lambda_{\delta-1}$ — The buffer, an array of δ vectors (size $1 \times n_y$).
T_t — Current time step.
g — The position of the current time step in the buffer.
T_0 — The time step at λ_0.
Λ — The accumulated value of: $e\,CA^i$ (size $1 \times n_y$).
Γ — The current value of: CA^g (size $1 \times n_y$).

ALGORITHM: TRACERESET — Reset to time 0.

```
⇒   for $i \leftarrow 0 \ldots (\delta - 1)$ : $\lambda_i \leftarrow 0$
    $T_t \leftarrow -1$,   $g \leftarrow -1$                    —next time / buffer position is 0
    $T_0 \leftarrow 0$,   $\Lambda \leftarrow 0$
    $\Gamma \leftarrow C$                                  —Get $\Gamma = CA^0$
```

ALGORITHM: TRACENEXT — Set the next error value e, this is called first for time 0.
Input: e (scalar)

```
⇒   $T_t \leftarrow T_t + 1$,   $g \leftarrow g + 1$
    if $g \geq \delta$ then : $g \leftarrow 0$
    if $g = 0$ then :   $T_0 \leftarrow T_t$,   $\Lambda \leftarrow 0$,   $\Gamma \leftarrow C$
    $\Lambda \leftarrow \Lambda + e\,\Gamma$,   $\lambda_g \leftarrow \Lambda$,   $\Gamma \leftarrow \Gamma\,A$     —Get $\Gamma = CA^g$ for the next step.
```

ALGORITHM: TRACEDELTA — Compute Δ_t, assuming $0 \leq t \leq T_t$ and $t \geq T_t - \delta + 2$.
Input: t (scalar)
Output: Δ_t (vector, size $1 \times n_y$)

```
⇒   ensure that $t \leq T_t$ and $t \geq 0$
    $i \leftarrow t - T_0$                                 —$i$ is the buffer position for time $t$
    if $i < 0$ then
        $i \leftarrow i + \delta$
        ensure that $i > g$                             —now $i$ is in the range $0 \ldots \delta - 1$
    ensure that $i \neq g + 1$
    if $i \leq g$ then
        if $i > 0$ then
            $\Delta_t \leftarrow (\lambda_g - \lambda_{i-1})A^{-i}$
        else
            $\Delta_t \leftarrow \lambda_g\,A^{-i}$
    else
        $\Delta_t \leftarrow (\lambda_{\delta-1} - \lambda_{i-1})A^{-i} + \lambda_g\,A^{T_0 - t}$
```

Table 5.3: *The "trace" algorithms for computing weight updates.*

where T_t is the "current" time, in terms of which the weight update equation is:

$$w_{pq} \quad \leftarrow \quad w_{pq} - \alpha \, \Delta_{i_1} \, \xi_{i_1}^{pq} \tag{5.31}$$

The trace algorithm maintains a buffer of δ past λ values (where $\delta = \sigma + 2$), as shown in figure 5.17. Here g is the index of the λ value for the "current" time T_t. Each call to TRACENEXT increases g and deposits a new λ value (g wraps around to zero at the end of the buffer). The buffer size limits the value of $T_t - t$ to at most $\delta - 2$, or σ. Inspection of the TRACENEXT procedure shows that the λ values are set as follows:

$$\text{for } 0 \le i \le g: \qquad \lambda_i \quad = \quad \sum_{j=0}^{i} e_{T_0+j} \, C \, A^j \tag{5.32}$$

$$\text{for } i > g: \qquad \lambda_i \quad = \quad \sum_{j=0}^{i} e_{T_0-\delta+j} \, C \, A^j \tag{5.33}$$

$$\tag{5.34}$$

Precision can be maintained because A^i never gets too small, as the exponent i never exceeds $\delta - 1$: each time the position g wraps around to zero the exponent on A^i is reset to zero. To calculate equation 5.30 the TRACEDELTA function combines λ terms with the appropriate scaling factors to compensate for the effect of the λ_0–$\lambda_{\delta-1}$ discontinuity. It is easy to show[3] that the TRACEDELTA function gives equation 5.30 for all valid values of t.

With this trace algorithm there is at most one λ-reset discontinuity between any valid time t and the current time T_t. A more general algorithm would allow the λ-reset interval to be selected independently of the buffer length δ. However, this introduces a lot of extra complexity (as more discontinuities must be compensated for) for little gain.

There remains the question of how σ (and therefore δ) should be chosen. This is analysed fully in Appendix F. The guideline is that A^σ should be "small but not too small". σ should be large enough such that an eligibility ξ decaying for σ time steps is small enough that its weight can be retired. σ should be small enough that the λ values stored in the buffer retain enough variation so that their differences do not lose too much numerical precision.

A complicating factor is that A is a matrix, not a scalar. It turns out that the determining factor for numerical precision is the eigenvalue of A with the smallest magnitude—call it q (if A is a scalar then $q = A$). The maximum number of decimal digits of precision that can be lost when calculating Δ_t is

$$\text{maximum decimal precision lost} \quad = \quad -\sigma \frac{\ln q}{\ln 10} \tag{5.35}$$

A proof of this is presented in Appendix F. A practical compromise is to select σ smaller than

$$\text{maximum } \sigma \quad \approx \quad -2 \frac{\ln 10}{\ln q} \approx \frac{-4.605}{\ln q} \tag{5.36}$$

This results in at most two lost digits of decimal precision, and (if the eigenvalues of A are all close enough to q) the eligibility will decay to at most approximately 1% of its starting value.

5.7.4 Summary of the algorithm

The FOX algorithm for a single output ($n_x = 1$) is given in table 5.4. The WIB is stored in the ϕ and ψ arrays. ϕ stores the indexes of the active and inactive weights. Each entry of ψ stores the eligibility

[3] The proof will be left as an exercise for the reader.

THE FOX ALGORITHM

Parameters

```
        All CMAC parameters (see table 3.1), but assume $n_x = 1$.
        $\sigma$ : The size of the internal buffer
```

Internal state variables

```
        $\theta[0] \ldots \theta[n_w - 1]$ — An array of $n_w$ auxiliary numbers, one for each weight.
        $\phi[0] \ldots \phi[n_a\sigma - 1]$ — An array of $n_a\sigma$ weight indexes (scalar).
        $\psi[0] \ldots \psi[n_a\sigma - 1]$ — An array of $n_a\sigma$ eligibility vectors (size $n_y \times 1$).
        $h$ — The current buffer head position (scalar).
        $T_f$ — The current time step (scalar).
```

ALGORITHM: FOXRESET — Reset to time 0.

```
⇒       if (this is not the first time FOXRESET has been called) then:
                $T_f \leftarrow T_f + 1, \quad h \leftarrow h + n_a$
                if $h \geq n_a\sigma$ then : $h \leftarrow 0$
                for $i \leftarrow 0 \ldots (n_a\sigma - 1)$ : if $\phi[i] \neq -1$ then : FOXCORRECTWEIGHT ($\phi[i]$,1)
        $T_f \leftarrow -1, \quad h \leftarrow -n_a$
        for $i \leftarrow 0 \ldots (n_w - 1)$ : $\theta[i] = -1$
        for $i \leftarrow 0 \ldots (n_a\sigma - 1)$ : $\phi[i] = -1$
        TRACERESET()
```

ALGORITHM: FOXMAP — Map CMAC inputs to output, keep track of weights.
Inputs: $y_1 \ldots y_{n_y}$ (scalars)
Output: x_1 (scalar)

```
⇒       CMACQUANTIZEANDASSOCIATE ($y_1 \ldots y_{n_y}$)    —sets $\mu_1 \ldots \mu_{n_a}$
        $T_f \leftarrow T_f + 1, \quad h \leftarrow h + n_a$
        if $h \geq n_a\sigma$ then : $h \leftarrow 0$
        for $i \leftarrow 0 \ldots (n_a - 1)$: if $\phi[h+i] \neq -1$ then : FOXCORRECTWEIGHT ($\phi[h+i]$,1)
        for $i \leftarrow 0 \ldots (n_a - 1)$
                FOXCORRECTWEIGHT ($\mu_{i+1}$,0)            —sets $\epsilon$
                $\psi[h+i] \leftarrow B + \epsilon$                 —driving part of $F^*$ filter
                $\phi[h+i] \leftarrow \mu_{i+1}$
                $\theta[\mu_{i+1}] \leftarrow h + i$
        $x_1 = \sum_{i=1}^{n_a} W_1[\mu_i]$
```

ALGORITHM: FOXCORRECTWEIGHT — Compute the correct value for weight i.
Inputs: i, old (scalars)
Output: ϵ (vector of size $n_y \times 1$)

```
⇒       if $\theta[i] < 0$ then : $\epsilon = 0$ and exit          —do nothing if weight not in buffer
        $\phi[\theta[i]] \leftarrow -1$                           —take weight out of buffer
        bpos $\leftarrow \lfloor \theta[i]/n_a \rfloor$                     —find buffer timestep position
        hd $\leftarrow \lfloor h/n_a \rfloor$
        $t \leftarrow T_f +$ bpos $-$ hd                       —find time $t$ when weight last active
        if (old $= 1$ and bpos $\geq$ hd) or (old $= 0$ and bpos $>$ hd) then : $t \leftarrow t - \sigma$
        TRACEDELTA ($t$)                               —sets $\Delta_t$
        $W_1[i] \leftarrow W_1[i] + \Delta_t \, \psi[\theta[i]]$
        if old=0 then : $\epsilon = A^{T_f - t} \, \psi[\theta[i]]$ else $\epsilon = 0$
```

Table 5.4: *The FOX algorithm for a single output.*

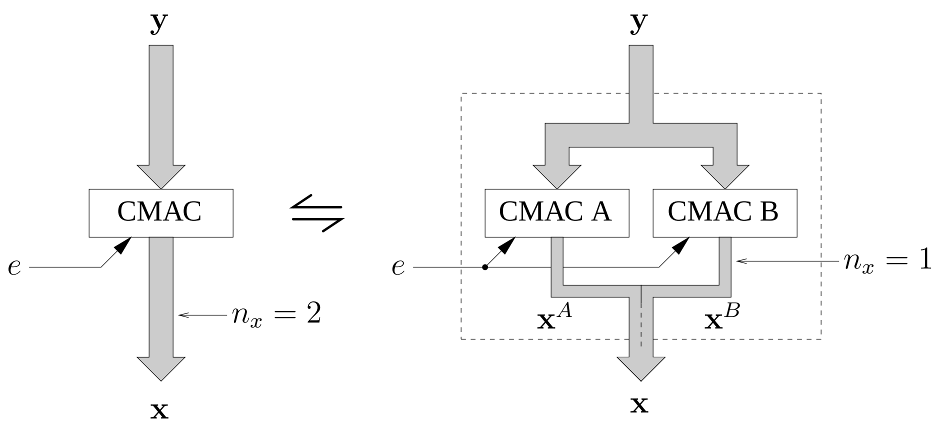


Figure 5.18: *A simple way to extend the FOX algorithm to* $n_x = 2$.

vector that the corresponding WIB weight had when it was last active. If $\phi[i] = -1$ then that WIB entry is unused. θ stores the weight auxiliary information. If $\theta[i] = -1$ then that weight is retired. This algorithm requires variant-2 CMAC hashing. This is because if two or more indexes in the WIB can point to the same weight then the eligibility computations become more complicated. This is how the FOX algorithm is used:

1. Call FOXRESET to reset the system time to 0.
2. For each system time step:
   (a) Call FOXMAP to map the input vector y to the output x.
   (b) Use the output x in the system and measure e, the resulting output error.
   (c) Call TRACENEXT ($\alpha\, e/n_a$).

The error is divided by n_a in the call to TRACENEXT to compensate for the number of AUs in the CMAC. At most n_a new weights can be made active per time step, so at most n_a weights will need to be retired per time step. Thus the total computation per time step is $\mathcal{O}(n_a)$. The FOX and TRACE algorithms can be sped up in several ways:

- In TRACEDELTA the matrix exponents (A^i) need not be calculated each time. Instead the values of A^i for $i = -\delta \ldots \delta$ can be buffered.
- In FOXCORRECTWEIGHT, if the weight to be updated was active in the last timestep (i.e. $T_t - t = 1$), as most weights usually are, then it can be updated quickly by

$$W_1[i] \leftarrow W_1[i] + (\text{last\_error\_passed\_to\_}\textsc{TraceNext})\; C\, \psi[\theta[i]] \tag{5.37}$$

These improvements are present in the FOX C++ source code (Appendix H).

Extending the FOX algorithm to $n_x > 1$ is relatively easy. One particularly simple way is shown in figure 5.18. Here a single CMAC with two outputs ($n_x = 2$) is replaced by two CMAC's with one output. The same error signal is fed to CMAC-A and CMAC-B, but the internal eligibility filters are different (reflecting the different effects of $\mathbf{x}^A$ and $\mathbf{x}^B$ on the system). Using two independent CMACs in this way is wasteful because computations common to both are duplicated. To gain full efficiency both CMACs must be combined into one algorithm. The details are left as an exercise for the reader.

5.8 More flexible error functions

Thus far the following error function has been assumed:

$$E = \sum_{i=1}^{T} \left(C\,\mathbf{y}_i - y_i^d \right)^2 \tag{5.38}$$

This will be called the *standard* error function. It has two limitations in practical systems. First, there is no constraint on the control signal $\mathbf{x}$. This can result in overtraining, as seen in Chapter 4 and Chapter 6. Second, the form of equation 5.38 does not allow multiple elements of the vector $\mathbf{y}$ to meet independent targets. These limitations will now be addressed.

5.8.1 Adding $\mathbf{x}$ constraints (output limiting)

As shown in the previous chapter, it will usually be necessary to limit the magnitude of $\mathbf{x}$ somehow in a practical system. Doing so means that the minimum-error trajectory from any starting point off the desired trajectory is fully specified. Here is an addition to the standard error function which directly constrains the magnitude of $\mathbf{x}$:

$$E = \underbrace{\sum_{i=1}^{T} \left(C\,\mathbf{y}_i - y_i^d \right)^2}_{\text{part 1}} + \underbrace{\frac{\beta}{2} \sum_{i=1}^{T} \mathbf{x}_i^T\, \mathbf{x}_i}_{\text{part 2}} \tag{5.39}$$

This will be called the *output limiting* error function. The weight gradient can be found using the chain rule, as before:

$$\begin{aligned}
\frac{\mathrm{d}\,E}{\mathrm{d}\,w_{pq}} &= \left(2 \sum_{k=2}^{T} e_k\, C\, \xi_k^{pq} \right) + \sum_{i=1}^{T-1} \frac{\mathrm{d}\,E}{\mathrm{d}\,\mathbf{x}_i} \cdot \frac{\mathrm{d}\,\mathbf{x}_i}{\mathrm{d}\,w_{pq}} && (5.40)\\
&= \left(2 \sum_{k=2}^{T} e_k\, C\, \xi_k^{pq} \right) + \beta \sum_{i=1}^{T-1} \mathbf{x}_i^T\, \hat{\mathbf{s}}_i^{pq} && (5.41)\\
&= \left(2 \sum_{k=2}^{T} e_k\, C\, \xi_k^{pq} \right) + \beta \sum_{i=1}^{T-1} \mathbf{x}_i^{(p)}\, \mathbf{s}_i^{(q)} && (5.42)
\end{aligned}$$

The bracketed term on the left is the standard eligibility equation arising from part 1 of the error function (i.e. equation 5.15). The term on the right arises from part 2. This results in the following weight update rule:

$$w_{pq} \leftarrow w_{pq} - \alpha \sum_{i=2}^{T} e_i\, C\, \xi_i^{pq} - \left(\beta \sum_{i=1}^{T-1} \mathbf{x}_i^{(p)}\, \mathbf{s}_i^{(q)} \right) \tag{5.43}$$

The bracketed term on the right is the addition to the FOX algorithm caused by part 2 of equation 5.39. It is equivalent to normal CMAC target training with a target of zero. Thus training with output limiting is implemented in the following way:

1. Call FOXRESET to reset the system time to 0.

2. For each system time step:

(a) Call FOXMAP to map the input vector y to the output x.

(b) Use the output x in the system and measure e, the resulting output error.

(c) Call TRACENEXT ($\alpha\, e/n_a$).

(d) Call CMACTARGETTRAIN (0, β)

5.8.2 Adding $\mathbf{x}$ derivative constraints

Sometimes limiting the magnitude of $\mathbf{x}$ is not useful because it may be required to grow arbitrarily large to perform some function in the system. In such situations $\mathbf{x}$ can be prevented from driving the system too hard by limiting the magnitude of $\mathrm{d}\mathbf{x}/\mathrm{d}t$ instead. For instance, the following error function does this with the first order numerical derivative of $\mathbf{x}$:

$$E \;=\; \gamma \sum_{i=1}^{T-1} |\mathbf{x}_{i+1} - \mathbf{x}_i|^2 \tag{5.44}$$

But in fact this error is less than perfect for implementation in FOX. Consider:

$$\frac{\mathrm{d}\,E}{\mathrm{d}\,w_{pq}} \;=\; \left[2\gamma \sum_{i=1}^{T-1} \Big(-\mathbf{x}_{k-1} + 2\mathbf{x}_k - \mathbf{x}_{k+1}\Big)^T \hat{\mathbf{s}}_i^{pq}\right] + 2\gamma \mathbf{x}_1^T \tag{5.45}$$

For a time span $i = T_1 \ldots T_2$ in which $\hat{\mathbf{s}}_i^{pq}$ is constant, changing the value of $\mathbf{x}_j$ (where $T_1 < j < T_2$) will not affect the value of $\mathrm{d}E/\mathrm{d}w_{pq}$. What this means is that $\mathrm{d}E/\mathrm{d}w_{pq}$ is only affected by discontinuities in the value of $\hat{\mathbf{s}}_i^{pq}$. This property has been found in practice to limit the effectiveness of the error function in constraining the $\mathbf{x}$ trajectory.

A more active (although somewhat heuristic) alternative is to train the CMAC output at each iteration using the CMAC output of the previous timestep as a target. This will be called *output derivative limiting*. It results in the following FOX algorithm:

1. Call FOXRESET to reset the system time to 0.
2. $\mathbf{x}_{\text{old}} \leftarrow 0$
3. For each system time step:

   (a) Call FOXMAP to map the input vector y to the output x.

   (b) Use the output x in the system and measure e, the resulting output error.

   (c) Call TRACENEXT ($\alpha\, e/n_a$).

   (d) Call CMACTARGETTRAIN ($\mathbf{x}_{\text{old}}$, γ)

   (e) $\mathbf{x}_{\text{old}} \leftarrow \mathbf{x}$

5.8.3 Adding extra $\mathbf{y}$ constraints

Consider the case where there are two seperate $\mathbf{y}$ constraints:

$$E \;=\; \sum_{i=1}^{T} \Big(\underbrace{C_A\, \mathbf{y}_i - y_i^{dA}}_{=\,e_i^A}\Big)^2 + \sum_{i=1}^{T} \Big(\underbrace{C_B\, \mathbf{y}_i - y_i^{dB}}_{=\,e_i^B}\Big)^2 \tag{5.46}$$

This leads to the following weight update equation:

$$w_{pq} \leftarrow w_{pq} - \alpha_A \sum_{i=2}^{T} e_i^A \, C_A \, \xi_i^{pq} - \alpha_B \sum_{i=2}^{T} e_i^B \, C_B \, \xi_i^{pq} \tag{5.47}$$

There are now two different error signals (e^A and e^B) and two different eligibility profiles ($C_A \xi$ and $C_B \xi$). Thus the FOX algorithm's data structures must effectively be duplicated, i.e. two WIB buffers and two ψ eligibility vector buffers are required. The resulting algorithm is structurally similar to table 5.4 (its derivation is relatively simple and is left as an exercise for the reader). As with the multiple-output case this may be easily implemented using two independent CMACs, except that in this case both CMACs must reference the *same* weight table. This argument may be extended to more than two constraints.

5.9 Special training situations

Some situations require special training procedures or special handling of the eligibility values.

5.9.1 Disturbances

If the system F is disturbed then the training process will be degraded. This is because the FOX model of the system does not encompass any mechanism that leads to the disturbance—in other words, it is unexpected. A disturbance can be an external force or noise signal that is applied unexpectedly to the system. It can also be something like a change in the control target, reference trajectory or system dynamics. There are three main ways to deal with disturbances:

1. Ignore them. This will result in some incorrect training, but if the disturbances are infrequent then this may be acceptable.
2. Anticipate the disturbances by feeding the appropriate sensor information to the CMAC. The sensor information must be sufficient to associate the cause of the disturbance with its effect on the system. Ideally this sensor information should anticipate the disturbance by some appropriate amount of time, so that FOX can *predict* it and optimally correct for it.
3. Whenever a disturbance occurrs, reset all the weight eligibilities to zero. This is done because the memory of the past context (encoded in the eligibilities) becomes irrelevant for predicting cause and effect after the disturbance has occurred. To reset all eligibilities to zero the FOXRESET function is called, which in turn will call FOXCORRECTWEIGHT for all weights in the WIB. Detection of controller-imposed disturbances (such as a change in the reference value) is easy. Detection of genuinely unpredictable external disturbances can be difficult, and will usually require some extra controller subsystems.

5.9.2 Switched CMACs

Consider what happens when a CMAC output can be switched in or out, so that sometimes it has no influence on the system. An example is shown in figure 5.19, which shows two CMACs multiplexed together to make a single controller. The eligibility profile should be set to zero when the CMAC output has no effect, because an impulse in the CMAC output would produce no change in the measured error. The normal eligibility profile can be used when the CMAC is switched in. The easiest way to implement this is to switch the eligibility filter driving force on and off as the CMAC is switched in and out. To do this in the FOXMAP algorithm the line

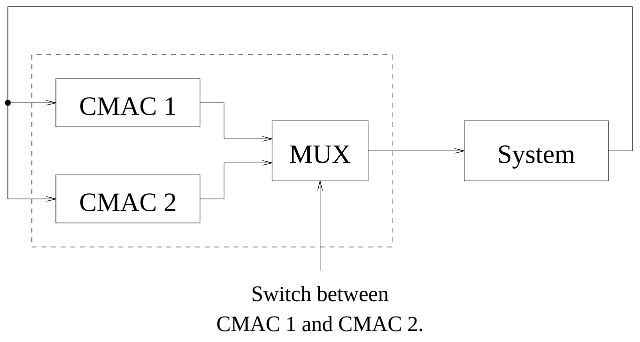


Figure 5.19: *A controller made from two CMACs multiplexed together.*

$$\psi[h+i] \leftarrow B + \epsilon$$

should be replaced with

```
if (CMAC is switched in) then
    $\psi[h+i] \leftarrow B + \epsilon$
else
    $\psi[h+i] \leftarrow \epsilon$
```

5.9.3 Timer control

The CMAC output can be used as a time delay (or threshold) that triggers the transition from one control state into another. In this case the CMAC output only influences the system during the instant that it triggers the state change. Therefore this point should be the only one that has a nonzero eligibility profile. This can be achieved using a modification to the FoxMap algorithm similar to that mentioned above. The (nonzero) eligibility profile will probably have to be determined heuristically due to the nonlinearity of the CMAC output's effect.

5.10 The continuous version

Thus far discrete time systems have been described. The development applies equally to continuous time systems. For example, the continuous eligibility and weight update equations are:

$$\dot{\mathbf{y}}(t) = A\,\mathbf{y}(t) + B\,\mathbf{x}(t) \tag{5.48}$$

$$\xi^{pq}(0) = 0 \tag{5.49}$$

$$\dot{\xi}^{pq}(t) = A\,\xi^{pq}(t) + B\,\hat{\mathbf{s}}^{pq}(t) \tag{5.50}$$

$$w_{pq} \leftarrow w_{pq} - \alpha \int_0^T e(t)\, C\, \xi^{pq}(t)\, \mathrm{d}t \tag{5.51}$$

But note that for equivalent discrete and continuous systems, the A matrices are different. No further consideration will be given to continuous time systems.

5.11 Related methods

Other authors have used similar approaches for incorporating eligibility parameters in to the CMAC.

Lin and Kim [67] implement Barto's adaptive critic [10] architecture using a CMAC instead of a simple lookup table. Each CMAC weight had an associated scalar eligibility value, but no guidance was given for choosing the eligibility dynamics and a naïve eligibility update approach was used. In [68] Lin and Kim provided loose guidelines for selecting scalar eligibility dynamics, but no connection between the eligibility dynamics and the controlled system dynamics was made.

Hu and Fellman [48] implemented a "state history queue" (SHQ) for improving the speed of the simple table lookup scheme used in Barto's adaptive critic [10]. Each "box" in the adaptive critic has an associated eligibility value. Their approach has some similarities to the weight index buffer technique used in FOX (it eliminates computations for temporally insignificant weights), but it does not achieve as high a speed-up, as they process the eligibility values of *all* weights in the SHQ whereas only retired and re-activated weight eligibilities are processed in FOX's WIB.

No other authors have considered the problem of parameter selection or implementation speed for *vector* eligibilities.

5.12 Conclusion

The FOX algorithm achieves optimal control using a CMAC to learn the best control actions. FOX associates an eligibility value with each weight, which allows the information in the error signal to be used to the best effect by altering a much larger group of weights than with the CMAC. In contrast to other reinforcement learning systems, the eligibility values can be vectors (not just scalars) and their update equations can be optimally selected to match the dynamics of the controlled system. FOX maintains auxiliary "trace" information about the error signal, which can be computed cheaply as a substitute for the expensive computation of eligibility values for all weights. FOX's speed is thus virtually independent of the CMAC size or the system size.

The FOX algorithm can be regarded as an extension of the feedback-error control approach. It has been shown that FBE by itself may not be able to control systems where the control actions do not have an immediately observable effect (systems having a non-zero-order impulse response). This will be experimentally verified in the next chapter. With FBE, explicit reference trajectories must be generated to suit different situations. In contrast FOX will automatically find an error-minimizing trajectory that approaches a single reference path in a well defined way. FBE uses the same signal for feedback and error purposes, whereas FOX allows independent choice of an internal feedback controller and error signal. This gives the designer more flexibility.

The FOX algorithm's synapse eligibility make it an even better model of the cerebellum than the standard CMAC, as cerebellar Purkinje cells also have an "eligibility" property (see section 2.7). It is possible that FOX can teach us something about how Purkinje synapses operate. For example, perhaps the eligibility profile of the Purkinje cell synapse correlates somehow with the dynamics of the motor region being modulated by that cell. Perhaps the dynamics of such motor regions are sufficiently damped that they can be successfully modeled by a low order eligibilty profile. However, these arguments (by functional analogy) are not very strong, and there has been no supporting evidence presented to date. In any case, the detailed operation of the biological motor control system is still not fully understood.

The next chapter will describe some design methodologies for FOX-based systems, and will look at how successful FOX is in practice.