# A Hardware Efficient Implementation of a Boxes Reinforcement Learning System

Yendo Hu and Ronald D. Fellman

*Department of Electrical and Computer Engineering, 0407*
*University of California at San Diego*
*December 21, 1993*

**Abstract**

This paper presents two modifications to the Boxes-ASE/ACE reinforcement learning algorithm to improve implementation efficiency and performance. A state history queue (SHQ) replaces the decay computations associated with each control state, decoupling the dependence of computational demand from the number of control states. A dynamic link table implements CMAC state association to decrease training time, yet minimize the number of control states. Simulations of the link table demonstrated its potential for minimizing control states for unoptimized state-space quantization. Simulations coupling the link table to CMAC state association show a 3-fold reduction in learning time. A hardware implementation of the pole-cart balancer shows the SHQ modification to reduce computation time 12-fold.

## Introduction

The Boxes network, developed by Michie and Chambers [7] and later refined by Barto *et al.* [2], is a reinforcement learning algorithm designed to solve difficult adaptive control problems using associative search elements (ASE) and adaptive critic elements (ACE). The equations of motion of the physical system are not known to the network. Rather, it learns how to respond based upon feedback from past trials. This feedback evaluates system performance from a failure signal occurring when the controlled object reaches an undesired state.

As shown in Figure 1, the ASE acts as a control table that uses the current system state as an address to retrieve a control action for the plant. The resulting action may generate a reinforcement signal, usually negative, that the ASE and ACE receives. On the basis of a first-order linear prediction from past reinforcement signals, the ACE uses this catastrophic reinforcement signal to compute a prediction of the reinforcement signal when the plant produces no actual reinforcement. The ASE updates its control information using both this improved reinforcement signal along with a trace through the previously traversed system states. The effect of the reinforcement signal on a control parameter decreases exponentially with the elapsed time since the system had last entered that state. In [2], a single ASE, along with one ACE, successfully learned to solved the pole balancing problem. Many other researchers have also used a pole-cart balancer to benchmark the performance of their algorithms [8] [1].

The ASE assigns a register to hold an output control value for each unique system state. Taken together, these registers form a control table that maps a plant state to a control action. Each register holds the long-term trace that represents both the output action, the trace's sign, and also a confidence level, the trace's magnitude. Thus, high confidence levels are represented by large trace magnitudes. The ASE adjusts only the traces of those states that led to the reinforcement event.

A second value for each state, the short-term memory trace, tracks the contribution of a state towards producing the current system state. For each state, this short-term trace weighs the reinforcement adjustment of the long-term trace value. This mechanism helps the ACE use past states to learn from a system failure in proportion to their contribution to the current outcome.

An ASE contains one control output (O(t)), one reinforcement input ($\hat{r}(t)$), and a decoded input state vector ($I_i(t)$). Each element of $I_i(t)$ represents a unique state within the system. For each $I_i(t)$, the $i^{th}$ element is 1 and all other elements are 0. The ASE output, which controls the system, is just the thresholded long term trace selected by the decoded input state vector as in (1).

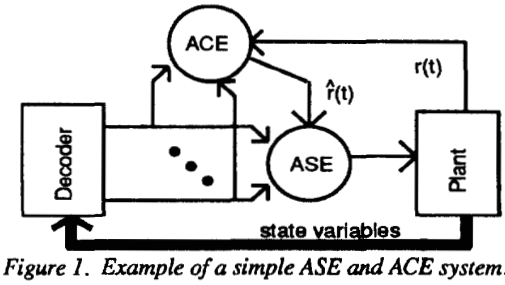


*Figure 1. Example of a simple ASE and ACE system.*

This work was funded by the Microelectronic Information Processing Systems Division of the National Science Foundation, RIA Award #MIP-9008839.

$$O(t) = \theta\left(\sum_{j=1}^{N} I_i(t)\, W_i(t)\right) \quad (1)$$

$\theta(a) = 1$ if $a \geq 0$, else $\theta(a) = -1$.

The following equations recursively relate two internally stored variables: the long term trace, $w_i(t)$, and the short term trace, $e_i(t)$, to each input, i:

$$w_i(t) = w_i(t-1) + \alpha r(t-1) e_i(t-1) \quad (2)$$

$$e_i(t) = \delta e_i(t-1) + (1-\delta) I_i(t-1) O(t-1) \quad (3)$$

where $r(t)$ = reinforcement signal,
$\delta$ = ASE trace decay rate
$\alpha$ = positive constant determining rate of change

The ACE contains one modified reinforcement output, one reinforcement input, and a set of inputs equaling the number of outputs from the front end decoder. Two internally stored variables, $\bar{x}_i(t)$, the time decay factor, and $v_i(t)$, the state predictor, recursively update their values each cycle from the current system state. The change in the system predictor, p(t), provides feedback to update the variables, $v_i(t)$ and $\hat{r}(t)$. The following equations describe the operation of the ACE:

$$\bar{x}_i(t) = \lambda \bar{x}_i(t-1) + (1-\lambda) I_i(t-1) \quad (4)$$

$$v_i(t) = v_i(t-1) + \beta \hat{r}(t-1) \bar{x}_i(t-1) \quad (5)$$

$$p(t) = \sum_{i=1}^{N} (v_i(t) \times I_i(t)) \quad (6)$$

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1) \quad (7)$$

where $\lambda$ = ACE trace decay rate,
N = number of possible input states
$\gamma$ = discount factor,
$\beta$ = positive constant determining rate of change

During operation of the pole-cart balancer, the system first quantizes each of four input variables: pole angular velocity, cart position, cart velocity, and the reinforcement feedback. The quantized system parameters then pass through the decoder and activate the unit state vector representing the current system state. The ASE/ACE network learned to balance the stick within an average of 70 trials [2].

### State History Queue

The short-term trace or eligibility function, $e_i(t)$, in the ASE, and the state trace, $x_i(t)$, in the ACE, decay exponentially, and, thus, only carry significant values over a limited time period. All computations involving these variables are influential only during this period. The state history queuing (SHQ) scheme takes advantage of this characteristic to reduce both memory and computation time.

Rather than store a decay value for each system state, the SHQ scheme keeps only a truncated list of all recently traversed input states. Weighting these states by position in the history queue generates an equivalent decay value. By restricting the length of this list, the SHQ scheme, in effect, sets a threshold to truncate insignificant decay values. It ignores weight adjustments of those states whose last visits occurred beyond a time period specified by the list length. Thus for each network update, only variables associated with the states listed in the queue are adjusted. This limited updating decouples computation time from the size of the input state space. It not only eliminates unnecessary memory accesses and decay computations to these discarded states.

The state history queue is just a shift register that keeps track of the past inputs states (Figure 2). The register depth, H, defines the length of the queue and sets the time threshold, beyond which, incremental weight adjustments are assumed insignificant, and ignored. The ASE's state history queue records the decoded input address and the generated outputs.

An exponential decay function is approximated by assigning linearly decreasing decay scalars to each register within the queue. The approximated decay function for each input state is just the sum of all decay scalars of registers holding the input state. This decay function replaces the decay variables $e_i(t)$ and $\bar{x}_i(t)$ in the original algorithm. A linear function to represent the set of constant decay scalars further reduces the complexity of the algorithm. When a system dwells in the same input state long enough to fill the queue, the SHQ step response approximates the first two terms in a Taylor expansion of the exponential response given by (3):

$$e(t) = (1-\delta^{-t})\, u(t) \approx tH\kappa - (t^2 - t)\frac{\kappa}{2} \quad (8)$$

where $\delta$ = trace decay rate, u(t) = step function
H = length of the state history queue,
$\kappa$ = decay scalar rate of change.

The simulations in the following sub-sections confirm that this approximation introduces only a small difference to the overall system performance.

Our specific implementation of the SHQ is given by (9) and (10). They replace equations (2) and (4), respectively.

$$w[SHQ.A[h]]_{new} = w[SHQ.A[h]]_{old} + \alpha\, R(t)\, (h+1)(SHQ.O[h])\kappa_{ase} \quad (9)$$

| | Address | Input State | Output | Decay Scalar |
|---|---|---|---|---|
| | $H_{ase}$-1 | A(t-[H-1]) | O(t-[H-1]) | k |
| ASE | . | . | . | . |
| State History Queue | . | . | . | . |
| | 1 | A(t-1) | O(t-1) | k(H-1) |
| | 0 | A(t) | O(t) | kH |

*Figure 2. The state history queue structure for the ASE.*

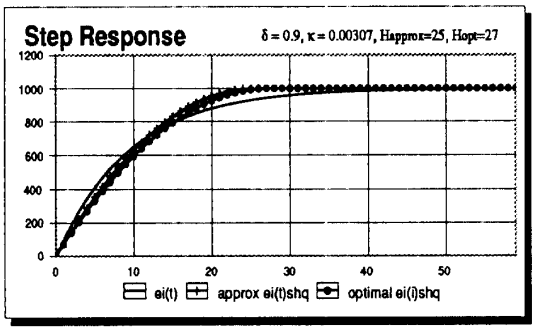


*Figure 3. Step responses of $e_i(t)$ & SHQ decay function.*

where: $\alpha$ =rate of change constant in ASE/ACE
h = pointer in SHQ, $0 \le h < H_{ase} - 1$
$\kappa_{ase}$ = SHQ weight,
w = control weights (addressable memory)

$$v[SHQ.A[h]]_{new} =$$

$$v[SHQ.A[h]]_{old} + \beta\,[R(t) + \gamma\, p(t) - p(t-1)]\kappa_{ace} \qquad (10)$$

where: h = pointer in SHQ, $0 \le h < H_{ace} - 1$,
$\gamma$ = discount factor (a constant)
v = prediction values (addressable memory)

*Matching the SHQ to the Eligibility Function*

Two parameters control the effective time decay in the SHQ scheme: $\kappa$, the decay slope, and H, the register length. Matching end points of the SHQ response with the eligibility function's step response, and matching values after one time constant, one can derive the simple guidelines given by (11) to determine $\kappa$ and H.

$$H = \frac{1 + e^{-1}}{(1 - e^{-1})\,|\ln\delta|} \approx \frac{2.54}{|\ln\delta|} \qquad \kappa \approx \frac{2}{H^2} \qquad (11)$$

The step response plot, Figure 3 plots the step responses of both the SHQ and the original eligibility function. For this example, the mean squared error between these two curves is 8%. The simulations shown on the next page confirm that the pole-cart adaptive control system using the SHQ performs at least as well as the original algorithm.

*Effectiveness of the SHQ Scheme*

We present two figures-of-merit to measure the effectiveness of these suggested improvements: TF, time efficiency factor, and MF, memory efficiency factor. A larger figure-of-merit indicates greater improvement.

$$TF = \frac{\text{time to address memory without modification}}{\text{time to address memory with modification}}$$

$$MF = \frac{\text{memory used without modification}}{\text{memory used with modification}} \qquad (12)$$

The original system without the SHQ would need to update all elements in the e, $\bar{x}$, w, and v arrays each cycle; whereas the modified network with the SHQ only need update some elements within these arrays. Figure 4 gives the number of multiplication and addition operations required to update each array.

**Operations Per Update**

| | Without SHQ | | With SHQ | | |
|---|---|---|---|---|---|
| | Multiply Operations | Addition Operations | Multiply Operations | Addition Operations | Shifting Operations |
| e | 3M | 1M | 0 | 0 | H |
| X | 2M | 1M | 3H | 2H | 0 |
| W | 2M | 1M | 0 | 0 | H |
| V | M+2 | M+2 | H+2 | H+2 | 0 |

Definition of Variables:
M = total number of possible inputs states
H = Length of State History Queue

*Figure 4. Comparative effectiveness of the SHQ.*

From equations (2) and (9), the time efficiency factor becomes:

$$TF_{shq} = \frac{(4M+2)\,T_a + (M+2)\,T_m}{2T_s H + (3H+2)\,T_a + (4H+2)\,T_m} \qquad (13)$$

where $T_s$ = time to perform one shift operation,
$T_a$ = Time to perform one addition operation
$T_m$ = Time to perform one multiply operation.

The memory needed for the original network must be large enough to hold the four arrays: e, $\bar{x}$, w, and v, resulting in a total memory demand of 4M. The SHQ eliminates the two arrays, e and $\bar{x}$, but adds the state history queue of length H; the SHQ will need 2H memory registers. The memory efficiency factor is thus:

$$MF_{shq} = \frac{4M}{2H + 2M} \qquad (14)$$

*SHQ Simulation*

Computer simulations of this learning system, both with and without our modifications, measured the improvement in speed, performance, and hardware demand. The original adaptive network of [2] offered a reference for comparison. All programs are written in Objective C and ran under the Unix operating system. We took advantage of the NextStep interface library to generate a graphical user interface that animated real-time learning of the complete system. A hardware implementation using a 2 MHz, 8-bit microcontroller (MC68HC11) demonstrated the practical feasibility of such a system.

The simulation recreated the environment and system of [2] as faithfully as possible. The simulator used Euler's method to solve the differential motion equation, using a time step of 20 ms. We measured performance by observing the number of trails verses elapsed time between failures.

*Effectiveness of the SHQ*

We evaluated the SHQ's effects on learning performance of the modified ASE/ACE network. The SHQ replaced the short-term trace, or eligibility function, in the ASE, and the state trace in the ACE according to (9) and (10), respectively. For $\delta = 0.9$, equation (12) set ASE SHQ length, $H_{ase}$,= 25 and $\kappa_{ase} = 0.00307$. Similarly, for $\lambda = 0.8$, $H_{ace}$ and $\kappa_{ace}$ equaled 13 and 0.0109. We computed the average learning curves over ten random seeds, shown in Figure 5. Without SHQs, our simulator closely matched the performance reported in [2].

The faster learning rate of the SHQ arose as a side effect of the SHQ's truncating the decay impulse response and from the limited number of noise seeds for the simulation runs. The truncation eliminated from consideration those states whose effect on the current outcome was probabilistically irrelevant. For the pole-cart simulation, the SHQ's time and memory efficiency gains were:

$$TF_{shq} = \frac{650T_a + 1298T_m}{50T_s + 77T_a + 102T_m}$$

$$MF_{shq} = \frac{648}{50 + 324} = 1.7326 \qquad (15)$$

**Link Table Dynamic Memory Allocation**

The amount of control memory is proportional to the number of possible decoded states, and is exponentially related to the number of inputs and their quantization levels. To minimize the size of the state space, [2] manually optimized the quantization levels for his specific system. The quantization levels used were non-uniform and their number varied among the input parameters.

In general, manual state optimization is not feasible. We would prefer a system that allows a large state space, but sparsely allocates physical memory only to states that the system actually uses. Otherwise, memory demand may become unreasonably large were physical memory allocated to every possible address. Furthermore, most complex systems do not access all possible states, many states are unstable or not physically possible.

The link table only allocates physical addresses for states that have been accessed. It dynamically assigns new physical memory to each new state. However, each read or write operation requires comparing the input address to all addresses stored in the link table. For associative operation, the system accesses all addresses within a given distance from the input address. If no cached state exists within a given distance of the input state, the system adds it.

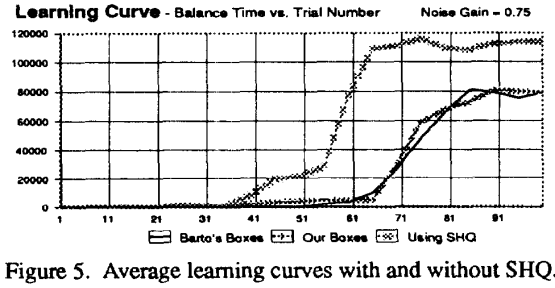


Figure 5. Average learning curves with and without SHQ.

Although searching the link table can significantly lengthen memory access time, several factors mitigate this problem. First, only a relatively small fraction of total system states will end up in the link table. And secondly, each new link table entry can be added in a presorted order, halving the search time, on average. And while sequential search is the simplest means of finding an address, more efficient search algorithms exist [5] that scale logarithmically with table size. Furthermore, the link table can be readily partitioned for parallel searching. Special hardware accelerators can speed access, as we implemented in our hardware system.

For this work, we designed system with sufficient memory such that memory resources were never exhausted. In continuation of this work, we are investigating two approaches to reallocating memory when all physical memory is consumed: least recently used line replacement, and a modified Kohonen Self-Organizing memory.

In general, the number of accessed states accumulate much more slowly than the exponential memory space increase that follows a linear increase in state address length. In particular, as more complex systems require more quantization levels or system parameters, the ratio of possible states to accessed states grows rapidly. As this ratio grows, so does the memory efficiency provided by the link table.

A memory efficiency factor, $MF_{link}$ measures the effectiveness of the link table. This memory efficiency factor, $MF_{link}$, is given as:

$$MF_{link} = \frac{\text{memory used without link table}}{\text{memory used with link table}} = \frac{M}{2M_u} \qquad (16)$$

*Link Table Effectiveness*

We applied the link table scheme to our adaptive control system with the SHQ modifications. Two versions of the system with different input quantization levels were used to measure the link table scheme's contribution to the overall memory savings for the system. The first system optimally quantized the input state space into 162 states, according to [2], such that nearly all states are frequented. The second system doubled the number of quantization intervals for each of the four input variables: cart position, cart velocity, stick angle, and stick angular velocity to 7, 7, 12, and 7 levels, respectively. This resulted in 4,116 possible system states, a 25-fold increase. Figure 6 plots the normalized average number of input states allocated with physical memory verses trial for both systems.

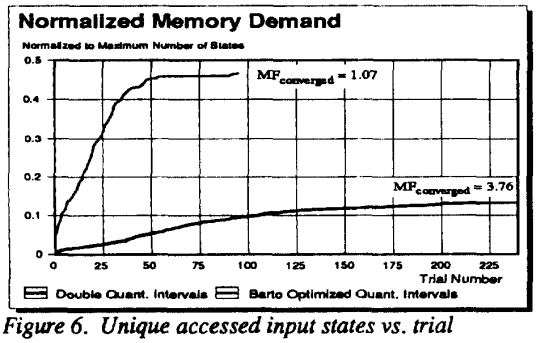


*Figure 6. Unique accessed input states vs. trial number.Thin line-optimized quantization. Thick line - double the quantization intervals.*

For the optimized system, the SHQ showed a 7% degradation in memory demand from the overhead of storing the link table itself. However, the second system realized a 3.76-fold savings in memory. In more complex systems, where it is difficult to optimize quantization, the link table would provide significantly greater memory savings. It also provides an effective and flexible realization of vector quantization by dynamically mapping a large state space to a small number of quantized states.

### CMAC State Association

Significant improvements in learning occur when the system can apply learning from previous experiences to similar but slightly different situations. This association of learning from nearby states to a new state is equivalent to a form of interpolation that fills gaps in knowledge. Non-linear regression and CMAC state association have explicitly been applied to adaptive learning systems [2] [6]. Such associative learning systems incorporate a distance measure in their state addressing to interpolate a response from the values of neighboring states.

In particular, [6] applied CMAC state association to the ASE/ACE system. However, they used an unspecified hash coding scheme which did not address many of the implementation concerns. Their hash function randomly map a sparse system state space to a denser set of physical registers. However in general, hash functions do not generate a uniformly dense mapping of states. Thus to prevent the overlay of several states mapping to the same control register, the density of physical registers is not minimal. When several system states overlay, control can become random and unpredictable. Methods to avoid such overlap, by necessity, require greater addressing computation and less efficient use of physical memory. Our dynamic link table memory allocation scheme minimizes the complexity in computing neighborhood state addresses, improves predictability and reliability, has a widely applicable neighborhood model, and minimizes hardware and physical memory demand. However, it does so at the expense of increased access time.

The input state consists of several fields. Each field represents an input parameter. The control table address may be computed using (17).

$$A = \sum_{i=1}^{N} \left[ \left( \prod_{j=0}^{i-1} L_j \right) X_i(t) \right] \tag{17}$$

where N = number of input variables
$L_i$ = number of quantization levels for input $i$
$X_i(t)$ = quantized value of input $i$ at time $t$

$$0 \le X_i(t) < L_i \text{ where} L_i > 0 \tag{18}$$

The total number of addresses that the input vector may address is then,

$$\text{Total Number of addresses} = \prod_{i=1}^{N} L_i \tag{19}$$

The distance function defines the distance between two states. Equation (20) describes a simple, yet effective distance measure between states X and Y.

$$D = \sum_{j=1}^{N} \rho_j ABS(X_j - Y_j) \tag{20}$$

In our work, we uniformly weighted the state parameters and set the weight of these differences, $\rho_j$, equal to 1. A more accurate distance function might weight the elements' differences, but that would depend upon the control problem.

Associative writing accesses all addresses within a given neighborhood distance threshold, $D_{th}$, and accumulates the data into each location. Reading accumulates the contents of all addresses within $D_{th}$ and thresholds the sum to form a two-level output. This is very similar to Kanerva's Sparse Distributed Memory, but with deterministic (rather than random) allocation of physical memory.

*Performance of State Association*

We observed the effect of this CMAC state association on system learning. The read and write distances were both set to 1 for this simulation. As before, a Gaussian noise signal was added to the accumulated contents of the accessed neighborhood registers only during read cycles. The summed signal was then thresholded. The local regression was introduced only on the weight array, not the eligibility array e, $\bar{x}$, or the prediction array v. The average learning curve of the complete modified system, including SHQ, the link table, and CMAC state association, showed a 4-fold decrease in learning time and a 4-fold improvement in robustness (Figure 7) with the following set of network parameters: $H_{ace}$=10, standard deviation = 0.01, $\kappa_{ace}$ = 0.0182, and association distance = 1. All other parameters remained the same as before. The learning curve of Figure 7 plots the average number of time steps the stick remained balanced for each trial.

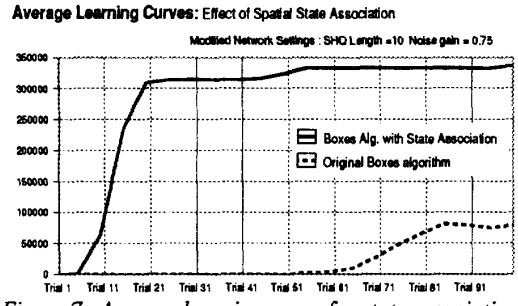


*Figure 7. Average learning curve for state association.*

### Hardware Implementation

A hardware implementation of the Boxes ASE/ACE adaptive controller, with the SHQ modification, controlled a physical pole-cart plant to demonstrate the implementation advantages of the SHQ scheme. The reduction in hardware enable this system to be self-contained as a single portable unit capable of learning to balance the stick in real time.

Except for the link table and CMAC state association, a single, low-cost 8-bit microcontroller, a Motorola 68HC11, implemented the entire ASE/ACE control algorithm. The reduction in computation time resulting from the State History Queue enabled this 2 MHz processor to update weights in real time, a 50 Hz rate. A Xilinx 3090 programmable gate array augmented the 68HC11 to control external memory. A single Xilinx 3090 contained all circuitry required to maintain the link table and the CMAC state association.

The entire network fits onto a 5 $in.^2$ board. All 2 kilobytes of firmware resides within the 68HC11. Computations were performed with 16 bit integer arithmetic and shift-add multiplies. The 256 bytes of internal controller RAM held system variables, parameters, and program stack. Less than 1 Kilobyte of external RAM implemented the SHQ, holding values for long-term traces and the ACE's prediction array. For the 68HC11, $TF_{shq} = 12$. A separate 68HC11 controlled the cart's stepper motors. It kept track of cart position and controlled acceleration. An optical sensor measured stick angle and fed this information to the built-in analog to digital converter in the other 68HC11.

This physical environment presented a non-ideal training environment to challenge the controller. Mechanical vibration from the motors reduced the angle sensor's accuracy in determining the stick's state. The stepper motors limited the cart's positional resolution. Inconsistency during human supervision when re-centering the stick set different initial states for successive trials. These uncertainties introduced excess noise that reduced system learning rate. Without state association, the controller with state history queuing and the dynamic link table successfully learned to balance the stick for more than 10 minutes after 100 human supervised trials. Debugging and measurement of the system with state association is in progress.

### Conclusion

The compatibility of the Boxes-ASE/ACE reinforcement learning algorithm with digital technology, combined with the efficiency enhancements of the State History Queue and Dynamic Link Table, permitted a low-cost, real-time implementation of an entire, self-contained pole-cart balancer using only two low cost 8-bit microcontrollers, a single programmable gate array, 2 KBytes of ROM and 1 KByte of RAM. These enhancements not only kept hardware demand at a practical level, it also improved learning speed.

The State History Queue limits decay computations and memory access only to addresses are significant to the control system. It thereby reduced provided a 12-fold reduction in computation time and a 1.7-fold reduction in memory. A link table dynamically maps a sparse control state space to a small set of physical memory of control states the system actually accessed. CMAC state association provided local regression to fill gaps in control knowledge. It augmented the control algorithm to provide a 3-fold reduction in simulated training time. A single gate array integrated circuit contained all the circuitry to support state association and the link table.

These enhancements demonstrate the practical application of associative reinforcement learning to the hardware implementation of a simple adaptive control problem and provide the foundation for implementation of complex intelligent control applications.

### References

[1] **Anderson, C. W.,** "Learning to control an inverted pendulum using neural networks," *IEEE Control Systems Magazine*, vol. 9, pp. 31-37 April 1989.

[2] **Atkeson, C. G. and Reinkensmeyer, D. J.,** "Using associative content-addressable memories to control robots," in Neural Networks for Control, Miller III, W. T., Sutton, R. S. and Werbos, P. J. eds., Cambridge, MA: The MIT Press. 1990.

[3] **Barto, A. G., Sutton, R. S., and Anderson, C. W.,** "Neuronlike adaptive elements that can solve difficult learning control problems," in *IEEE Trans. Syst., Man, Cybern.*, SMC-13, pp. 834-846, 1983.

[4] **Kanerva, P., Sparse Distributed Memory**, Cambridge, MA: The MIT Press. 1988.

[5] **Knuth, D.,** "Sorting and Searching," in *The Art of Computer Programming*, vol.3, Menlo Park, CA: Addison Wesley, 1973.

[6] **Lin, C. and Kim, H.,** "CMAC-Based Adaptive Critic Self-Learning Control," *IEEE Trans. Neural Nets*, vol.2, No. 5., Sept. 1991.

[7] **Michie, D. and Chambers, R.A.,** "BOXES: An Experiment in Adaptive Control," in **Machine Intelligence 2**, Editors: E. Dale and D. Michie, Edinburgh: Oliver and Boyd, pp. 137-152, 1968.

[8] **Widrow, B.,** "The original adaptive neural net broom-balancer," in *Int. Symp. Circuits and Syst.*, pp. 351-357, May 1987.