An Integrated Connectionist Approach to Reinforcement Learning for Robotic Control: The Advantages of Indexed Partitioning

Dean F. Hougen HOUGEN@CS.UMN.EDU
Maria Gini GINI@CS.UMN.EDU
James Slagle SLAGLE@CS.UMN.EDU
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 USA

Abstract

We explore the use of a connectionist-learning system designed to allow the application of reinforcement learning to robot control. In particular, we compare direct and indexed partitioning methods and find indexed partitioning has advantages in time and space complexity, learning speed (measured in trials), and success rate. We make these comparisons based on extensive simulations and runs on a real robot learning on-line.

1. Introduction

Reinforcement-learning methods provide great potential for robots to improve their performance through "trial and error" interactions with their environments. Unfortunately, reinforcement-learning methods run up against two obstacles when we apply them to robots: (1) Reinforcement-learning methods work best on discrete states whereas robotic environments are often continuous. (2) Reinforcement-learning schemes require a large number of *trials* to improve significantly, which is impractical for robotic systems.

To overcome these obstacles, we have proposed an integrated connectionist approach that uses laterally-connected networks (Hougen et al., 1997). This approach is general enough to encompass two different input space partitioning methods, which we call *direct* and *indexed*. In this paper, we address the question of which of these input space partitioning methods is superior and we present, for the first time, results with a real robot.

2. Related Work

To overcome obstacle one, three basic approaches have been used. The first approach is to discretize the input by hand (Woodcock et al., 1991). This requires either that the system designer have extensive knowledge of the environment in which the robot will act—whereas unsupervised learning is most appropriate when such knowledge is not available—or that the partitioning be uniform and fine—which may lead to many unnecessary partitions and slower learning. The second approach is to modify the model to accept continuous input. This has produced longer learning times and/or poorer performance (Anderson, 1989; Maričić, 1991). The third approach is for the system to learn a partitioning.

Partitionings may be learned by successively dividing existing partition regions (Salganicoff et al., 1996) but if memory is limited, as in robotic applications, a poor choice of parameters may mean that some regions are partitioned excessively and memory is exhausted. Another alternative is to learn by adjusting the borders of a fixed number of partitioning regions (Kretchmar & Anderson, 1997; Rosen et al., 1992). We use a neighborhood-based, border-adjusting procedure. The fact that our procedure is neighborhood-based gives it the advantage that it can be integrated with a neighborhood-based approach to obstacle two. Our work is also the first to consider an indexed partitioning method, which we show to have distinct advantages over direct partitioning methods.

To overcome obstacle two, the approach has been to use a generalization scheme to find relevant similarities between environment states. Generalization schemes have typically borrowed from methods used previously in supervised learning (Kretchmar & Anderson, 1997). We use an unsupervised, neighborhood-based generalization scheme that can be integrated with our input partitioning method.

3. The Learning System

Rather than trying to surmount the obstacles separately, we take an integrated approach to both. We call this approach the Integrated Connectionist Approach to Reinforcement Learning (ICARL[1]) for control. To overcome obstacle one, we have the system learn a partitioning of the input space using a self-organizing network that extracts regularities from patterns of input data. This is the *input network*.

Partitioning the input space into a finite number of classes allows us to use mechanisms from traditional reinforcement learning. However, we still make use of the fact that the underlying input is continuous to overcome obstacle two. Because we know that the classes to which we have assigned the input have come from underlying continuous variables, we can determine which classes are "near" to one another in the continuous input space. Using this knowledge we allow an experience which involves one class to aid in determining responses to other classes that are "close by." This is done in the *output network*.

The final step in our integrated approach is the choice of method used. For both the input and output networks we use laterally connected networks similar to Self-Organizing Topological Feature Maps (Kohonen, 1989). This parsimonious approach to both obstacles not only requires fewer total mechanisms in the system but allows for changes to the entire system to be made more easily. For example, to reduce memory requirements, all networks can be shrunk proportionally.

3.1 Topology and Neighborhoods

Both the input and output networks make use of a topological ordering of their neurons. Let d_N be the dimension of a network. Each neuron is associated with a d_N-tuple that uniquely defines its coordinates in topology space. The existence of a network topology allows for the definition of a *distance* function for the neurons; we use the maximum difference between neuron coordinates.

The distance function is used indirectly through the definition of *neighborhoods*. If U is the set of neurons u in the network, D the distance function defined on topology space, and W a trial-dependent function specifying the width of the neighborhood, then the neighborhood N of neuron n on trial T is defined as

$$N_n(T) = \{u \in U \mid D(n,u) \leq W(T)\} \tag{1}$$

3.2 Input-Space Partitioning

Input-space partitioning in ICARL systems takes place through self-organization. For this, an ICARL system may use a single network of dimensionality equal to that of the input space or may use multiple networks of lower dimension, as long as all dimensions of the input space are covered.[2] In the present application, the input space is two-dimensional. Using a single two-dimensional input network gives *direct partitioning*. Using two separate one-dimensional input networks gives *indexed partitioning*.

Reinforcement-learning systems are trained in a series of *trials*. A trial is divided into discrete time units, and lasts from an initial positioning of the controlled system until a success or failure signal is generated. During a *run* of multiple trials, the learning system progresses from random performance to competence.

Each input neuron has a weight vector consisting of one weight for each dimension of its input network. (For indexed partitioning this vector has a single entry—i.e. each weight "vector" is really a scalar.) These weights are given random values at the start of each run. On each time step a new input vector $\bar{x}$ is given to a network and its values are compared to the corresponding values of the weight vectors using an appropriate similarity measure S (such as the absolute value of their difference). The neuron s that has the weight vector most closely matching the input is *selected*. Formally,

$$\exists s[S(\bar{w}_s, \bar{x}) \leq S(\bar{w}_u, \bar{x}) \mid \forall u \in U, s \in U] \tag{2}$$

where $\bar{w}$ is a neuron's weight vector. If more than one neuron satisfies this equation, then the neuron with the lowest number (topological coordinate) is selected. Once the trial has ended, the input vectors are represented to the input network and for each time step the weights of the selected neuron and of all other neurons in its neighborhood are updated using

$$\bar{w}^{new} = \bar{w}^{old} + \alpha(t)\gamma(t_{final})(\bar{x} - \bar{w}^{old})f' \tag{3}$$

where f' is a feedback signal (+1 failure, 0 success), α and γ are functions that determine the influence of the input vector ($0 \leq \alpha, \gamma \leq 1$), and t_{final} is the total number of time steps in this trial. α starts near 1 and decreases with time to give the neurons independence as they become organized and γ ensures that all trials are weighted equally for purposes of partitioning the space, regardless of their length in time steps. Using Equation 3, there is competition amongst all the neurons in a network to be the neuron selected and cooperation within a neighborhood as the neurons

[1] We had used a name with the acronym SONNET which conflicted with the unrelated work of Nigrin (1990).

[2] While it is possible to map a lower-dimensional self-organizing map onto a higher-dimensional space, the resulting convolutions will have a negative impact on learning the correct output-responses (Martinetz et al., 1989).

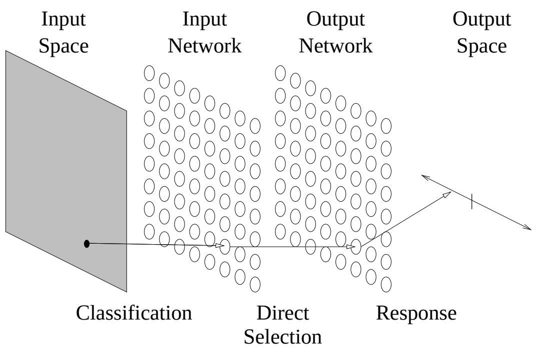

Figure 1. A mapping from input space to output space with a direct partitioning of the input space.

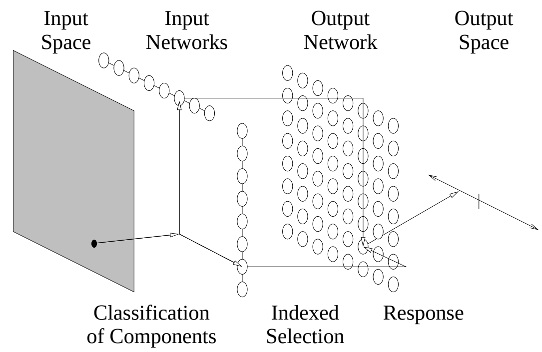

Figure 2. A mapping from input space to output space with an indexed partitioning of the input space.

change their weights towards the same target. By presentations of successive data to the network, the network self-organizes so that closely neighboring neurons have similar values for their weights (and therefore respond to similar input) while neurons that, according to the network topology, are far from one another have very different values for their weights (and therefore respond to very different input).

The selection of neurons from all input networks allows for the selection of a single corresponding neuron from the output network. Further, this correspondence should ensure that adjacent regions of the input space result in the selection of output neurons adjacent in their topology space. For this reason, the output network has the same dimensionality as the input space. For direct partitioning, the two-dimensional topological coordinates of neurons in the single input network can be matched directly with the neurons of the two-dimensional output network (see Figure 1). For indexed partitioning, the one-dimensional topological coordinates of neurons in the two input networks are used as indices into the two-dimensional output network (see Figure 2). This means that for both methods the combination of all weights in the input networks provide a partitioning of the input space. This partitioning gives a discretization of the input space and allows for a single output response to be learned for each of the resulting discrete input regions.

3.3 Output-Response Learning

When a neuron from the output network is selected, it produces an output response based on the value(s) of its weight(s). Each output neuron has one weight for each dimension of the output space. These weights are given random values at the start of each run. Their updates use *eligibility traces*. Eligibility traces have been used to explain brain function (Klopf, 1974) and in reinforcement learning (Singh & Sutton, 1996).

At the start of a trial, all neurons have an eligibility value of zero. When an output neuron is selected and responds (fires) it becomes amenable to change according to the equation

$$e^{new} = e^{old} + I \tag{4}$$

where e is the eligibility of the neuron for adaptation, and I is the initial eligibility value of a neuron that has just fired. This plasticity reduces with time but provides an opportunity for learning based on feedback received by the neuron after its activity. At each time step, the eligibility value of each output neuron decays, regardless of whether that neuron fired on that time step. Eligibility is realized as a trace according to

$$e(t+1) = \delta e(t) \tag{5}$$

where δ is the rate of eligibility decay ($0 \leq \delta \leq 1$). When a success or failure signal is received by the output network, all of the weight vectors of all of the output neurons are updated according to

$$\bar{v}^{new} = \text{sign}(\bar{v}^{old})(|\bar{v}^{old}| + e\,\sigma(T)\,f) \tag{6}$$

where $\bar{v}$ is the weight vector, e is the eligibility for adaptation, σ is a scaling function that changes with the trial number T, and f is the feedback signal (+1 for success, -1 for failure). The scaling function σ is used to allow for large changes to the weights in early training trials and smaller changes in subsequent trials.

As with the input network(s), the output network uses inter-neural cooperation. After the completion of a trial each neuron updates its weight(s) a second time, according to

$$\bar{v}_i = (1 - \beta(T))\bar{v}_i + \beta(T) \sum_{n \in N_i} \frac{\bar{v}_n}{m} \tag{7}$$

where each $\bar{v}$ is a weight vector, N_i is the neighborhood of neuron i, m is the number of neurons in that neighborhood, and β determines the degree to which a neuron's value is "smoothed" with that of its neighbors ($0 \leq \beta \leq 1$). In general, β starts near one and decreases with time. This means that each neuron's value becomes more independent from those of its neighbors as a run progresses.

4. Experiments and Results

Our application is learning to back a truck and trailer rig to a goal. The system has two inputs: The angle of the hitch between the truck and the trailer and the angle between the spine of the trailer and the goal as shown in Figure 3. The output value is thresholded at zero to provide bang-bang steering.

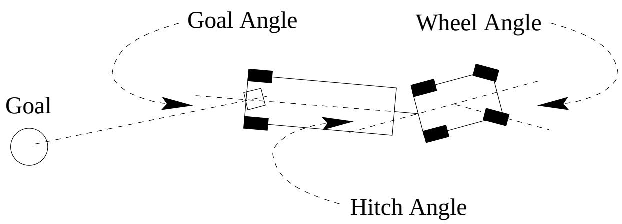

Figure 3. The truck-backing problem.

We constructed a baseline reinforcement-learning system for comparison. The baseline system does not learn the input space partitioning (the input space is divided into equal-sized regions in each dimension) and each output neuron learns its response individually. The baseline system, therefore, uses Equations 2, 4, 5, and 6 but not 1, 3, or 7. We applied ICARL and baseline systems in simulation and on a real robot.

4.1 Simulation Experiments

Two simulation cases were examined to provide variation in the details of the task to be learned. These two simulation cases were backing the rig starting from relatively small initial angles, and backing the rig starting from relatively large initial angles. In both cases, the starting point of the rear of the trailer was 6 feet from the goal.[3] For testing on each simulation case, we constructed versions of all three learning systems (baseline, direct-partition-learning ICARL, and indexed-partition-learning ICARL) ranging in size from 2 to 10 divisions per dimension of the input space.

4.1.1 Simulation Case 1

The hitch angle started in the range $\pm 15°$ and the goal angle started in the range $\pm 45°$. New initial conditions were chosen randomly at the start of each trial with a uniform distribution over the entire range of angles. The boundary values for both the hitch and the goal angle were $\pm 90°$; if the value of either angle exceeded these boundaries, failure was signaled.

Table 1 shows the *ultimate success rate* for each size of each learning system. The ultimate success rate is defined to be the average success rate during the last 50 trials of each run, averaged over 100 runs. Each run was 500 trials in length. More detailed results for two system sizes are shown in Figure 4. Systems sizes of 5 and 8 are shown as representative examples. Complete results are given elsewhere (Hougen, 1998).

Table 1. Ultimate success rates for the three versions of the learning system on simulation case 1.

Size	Baseline System	Direct ICARL	Indexed ICARL
2	4.24%	45.40%	2.94%
3	0.06%	42.16%	1.90%
4	30.58%	80.11%	47.26%
5	6.78%	91.12%	77.66%
6	66.12%	92.78%	93.00%
7	88.60%	94.08%	95.94%
8	92.10%	93.92%	95.84%
9	91.34%	90.40%	97.24%
10	92.82%	86.26%	96.32%

4.1.2 Simulation Case 2

The hitch angle started in the range $\pm 65°$ and the goal angle started in the range $\pm 180°$. New initial conditions were chosen randomly at the start of each trial with a uniform distribution over the entire range of angles. The boundary values for the hitch angle were $\pm 90°$. There were no boundary values for the goal angle; the goal angle was measured from $-180°$ to $+180°$ with these two values considered identical. However, because the system was no longer sure to reach either a success or a failure state in a finite number of time steps (it was now possible for the rig to back roughly straight in any direction, or in a circle, indefinitely), we added another failure condition. Failure was signaled if the rig backed for more than one thousand time steps (chosen because successful trials may be completed in a few hundred time steps, given the rig parameters).

Table 2 shows the ultimate success rate for each size of each learning system. More detailed results for two system sizes (5 and 8) are shown in Figure 5 and complete results are given elsewhere (Hougen, 1998). Again, each run was 500 trials in length and 100 runs are averaged together.

[3] Both the simulated and real rigs have a truck length of 7 in. and a trailer length of 11 in.

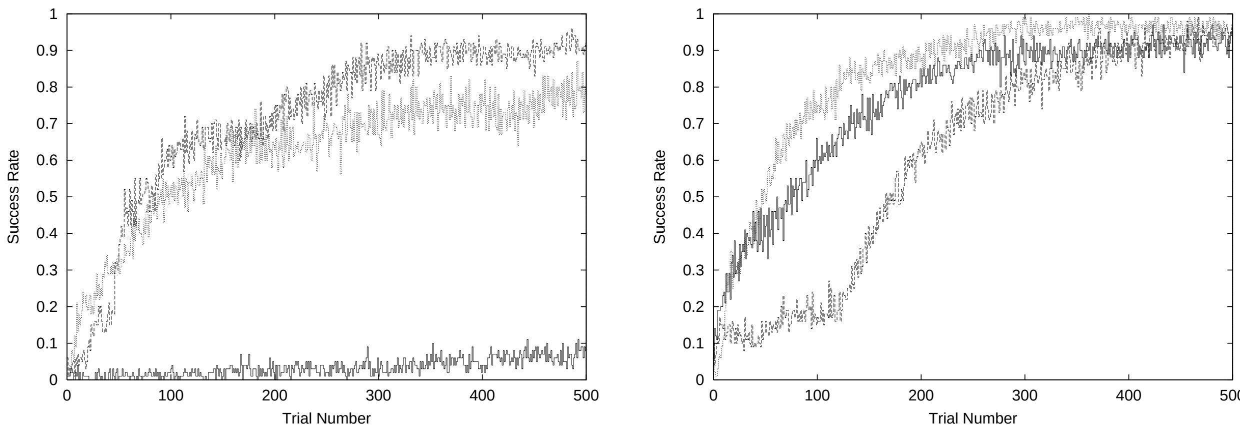

Figure 4. Results for simulation case 1: Relatively small initial angles. Average success rate for 100 runs of 500 trials each. Solid (dark) line: baseline system. Dashed (medium) line: Direct-partitioning ICARL system. Dotted (light) line: Indexed-partitioning ICARL system. Left: Five neurons per dimension. Right: Eight neurons per dimension

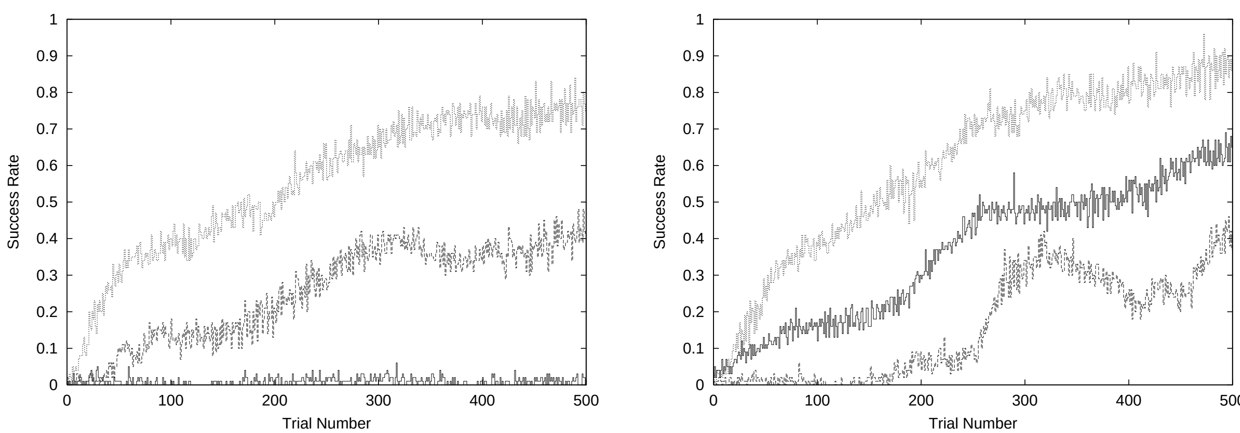

Figure 5. Results for simulation case 2: Relatively large initial angles. Average success rate for 100 runs of 500 trials each. Solid (dark) line: baseline system. Dashed (medium) line: Direct-partitioning ICARL system. Dotted (light) line: Indexed-partitioning ICARL system. Left: Five neurons per dimension. Right: Eight neurons per dimension

Table 2. Ultimate success rates for the three versions of the learning system on simulation case 2.

Size	Baseline System	Direct ICARL	Indexed ICARL
2	0.20%	4.90%	3.14%
3	0.00%	13.80%	33.18%
4	0.56%	32.76%	67.82%
5	1.12%	40.00%	75.36%
6	35.56%	44.46%	79.78%
7	62.58%	63.40%	80.84%
8	62.44%	34.16%	86.24%
9	59.20%	26.58%	83.70%
10	40.58%	9.60%	79.36%

4.2 Real-World Experiments

The real-world experiments are done on the small robotic platform known as the Self-Positioning Trailer-Backing Mini-Robot (SPTBMin) shown in Figure 6. Details of this robot are provided elsewhere (Hougen et al., 1999). Only one case of the application domain was studied using SPTBMin—one similar to simulation case 1. This case was chosen as the hardware could not track the goal through greater than 360° rotations, as is necessary for simulation case 2.

As with simulation case 1, the hitch angle started in the range $\pm 15°$ and the goal angle started in the range $\pm 45°$. New initial conditions were chosen randomly at the start of each trial with a uniform distribution over the entire range of angles. The boundary values for both the hitch and the goal angle were $\pm 90°$.

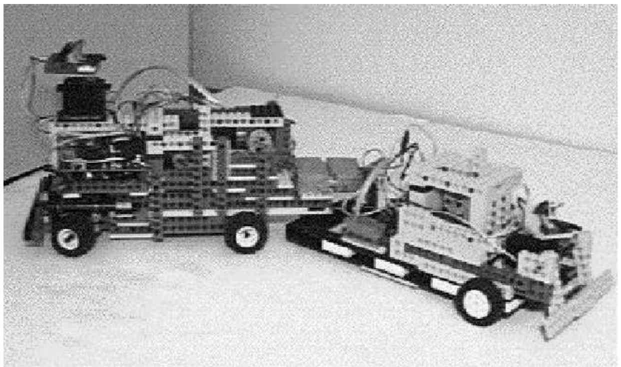

Figure 6. SPTBMin, the robot used in the real-world experiments.

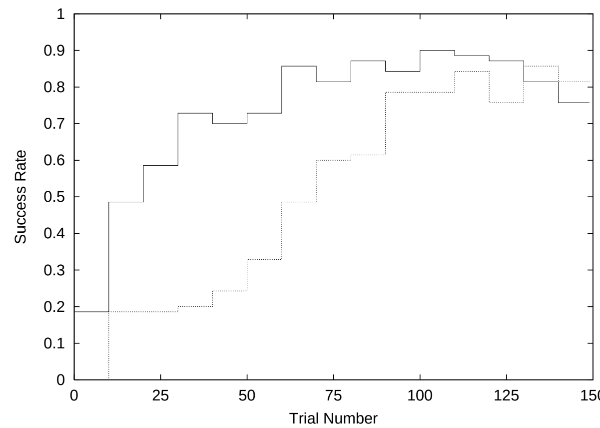

Figure 7. Results for real-world experiments. Average success rate for 7 runs of 150 trials each. Solid (dark) line: baseline system. Dotted (light) line: Indexed-partitioning ICARL system.

Only the baseline and indexed-partitioning ICARL systems were implemented on SPTBMin. This is because the limited on-board memory of the robot is not sufficient to hold both the code and data segments for ICARL systems that learn direct partitionings. Why the direct-partitioning ICARL system required more memory than the others, and the implications of this requirement, are discussed in Section 5. We chose to implement baseline and indexed ICARL systems of size 8, as this size gave good results in terms of learning rate and ultimate success rate for both learning systems. For each implemented system we ran seven runs of 150 trials each.

The results of these experiments are shown in Figure 7. To help make trends more evident in the data, we not only averaged the seven runs of each system together, we also averaged each set of ten trials with itself. That is, if during the first ten trials there were a total of 60 failures and 10 successes between the seven runs, then the average success rate of 14% would be plotted for the first ten trials. Using each set of ten consecutive trials as a basis for comparison, there are no significant differences between the success rates after the first 70 trials (two-tailed, two-sample t test, $p = 0.05$).

5. Discussion

Our goal is to develop a system that can learn on real robotic platforms. For this reason we are primarily concerned with having a system that is (1) able to rapidly reach high levels of performance, (2) able to function well over a range of conditions, (3) robust enough to overcome the problems of noisy input data and uncertain interactions between motor commands and effects in the world, (4) compact enough to fit into available on-board memory, and (5) able to give responses in real time.

While a few authors have used unsupervised learning systems for variations on the truck backing problem (Woodcock et al., 1991; Koza, 1992; Geva et al., 1992), no adequate comparison can be made with our work. For one, all of these authors provided their systems with information about the absolute position of the rig in the world—something not readily available to a real robot. Further, Woodock et al. provide no quantitative results and Geva et al. solve almost the entire problem by hand through task-decomposition (similar to the non-learning method of Jenkins & Yuhas, 1993), leaving almost nothing for a learning system to learn. Koza (1992) provides the closest comparison, using a genetic programming method that requires in excess of 400,000 back-up attempts (similar to our trials) to produce a solution.

For further comparison with existing methods, we applied a method for directly using continuous input that had been previously used for learning pole-balancing (Anderson, 1989). On simulation case 1, average results for ten runs of 100,000 trials each produced a maximum success rate of less than 50% between 70,000 and 75,000 trials. While no attempt was made to tune the parameters for this learning system, this result, and that of Koza (1992), suggest just how difficult this problem is.

More may be learned by comparing our systems with each other and with the baseline system introduced in Section 4. The baseline system is similar to the basic system from which other researchers have started (e.g. Barto et al. 1983).

At small sizes the baseline system was unable to learn satisfactory responses on either simulation case but at larger sizes it was able to achieve good performance on simulation case 1 and moderate performance on simulation case 2 (the harder case). In contrast, the direct-partition-learning ICARL system learned quite well on simulation case 1 at moderate sizes and had at least some success on simulation case 2 at those sizes but fell off in performance in both simulation cases as the sizes grew larger. Finally, the indexed-partitioning ICARL system did moderately to quite well in both simulation cases and at all sizes except the very smallest (2 or 3 divisions per dimension of the input space).

One reason that the direct-partition-learning ICARL system had difficulties as its size was increased, is that there is an important consideration that needs to be taken into account—adjacency relations. We need to ensure that neighboring output units correspond to adjacent input regions, at least as the system begins to stabilize, so that the sharing of response lessons is reasonable. Yet the input partitioning regions themselves are being learned and the Self-Organizing Maps are not initially organized to reflect the topology of the input space. As the SOMs learn this topology they may develop twists or folds. Such twists or folds happen early in the learning of the space and generally disappear if given enough time (Kohonen, 1989).

One advantage of learning an indexed partitioning over learning a direct partitioning is that the indexed system is less subject to both twisting and folding. With an indexed partitioning of a two-dimensional input space there are two one-dimensional input maps and the number of units in each map is equal to the number of partitions in one dimension. With the single two-dimensional input map for a direct partitioning the number of units in the input map is equal to the *square* of the number of partitions in one dimension. This means many more units need to untangle themselves before the adjacency consideration is met.

The differences in the systems in terms of their memory and computational requirements are summarized in Table 3. The memory requirements include the number of input weights (IN), the number of output weights (OUT), and whether a log of the input states needs to be recorded (LOG?). Because we are using rectangular partition regions for the baseline system, we can store only $d_I\eta$ input values, rather than all η^{d_I} weights. If our fixed partition regions were non-rectangular, the baseline system would have to store all η^{d_I} weights. The length of the state log was set at 1,000 time steps for simulation. For use on the real robot, however, the log contained only a subsample of the inputs, saving only every fifth state, and was therefore only one-fifth as long.

The computational requirements include classifying input data (CLASS), adjusting the partition boundaries (PART), and doing cooperative output learning (COOP). The classification of the input data takes place during a trial and thus impacts the real-time performance of the system as the robot moves. The adjustment of the partition boundaries and the cooperative output learning take place at the end of a trial and affect the amount of time the robot remains stationary before the next trial begins. As with the storage of input weights, the time for classification for the baseline learning system is reduced because we have chosen to use rectangular partition regions. If we used non-rectangular partition regions, the computational requirement would be the same as for the direct ICARL system, rather than the indexed ICARL system.

From this table we can see that the direct ICARL system requires the most memory and that the baseline system requires the least. The direct ICARL system also requires the greatest computation during each trial, although the baseline system would require equal computation for non-rectangular partition regions. Similarly, the direct ICARL system requires the most post-trial computation and the baseline system requires the least. For both memory and computation, indexed ICARL systems require slightly more resources than the baseline system, although the only significant difference is in the off-line computation required for cooperative response learning. Indexed ICARL systems make up for these additional demands by providing additional performance.

The runs made on the real robot demonstrate that indexed ICARL systems are capable of learning on board a real robot despite all the noise present in the system.

Table 3. Differences in the three versions of the learning system with respect to memory and computational requirement areas. d_I–dimensions in input space. η–neural units per dimension of input space. $W(T)$–width of neighborhood for response learning; changes with trial number T.

AREA	BASELINE SYSTEM	DIRECT ICARL	INDEXED ICARL
IN	$d_I\eta$	η^{d_I}	$d_I\eta$
OUT	η^{d_I}	η^{d_I}	η^{d_I}
LOG?	NO	YES	YES
CLASS	$O(\eta)$	$O(\eta^2)$	$O(\eta)$
PART	N/A	$O(\eta^{d_I+1})$	$O(\eta)$
COOP	N/A	$O(\eta^{d_I}W(T)^{d_I})$	$O(\eta^{d_I}W(T)^{d_I})$

6. Conclusions

Reinforcement learning allows systems to autonomously improve their performance through interactions with their environment. We have addressed the problem of continuous input by having the system learn its partitioning. We have addressed the problem of long learning times through cooperative response learning. The most original, and also the most successful, learning system developed is the indexed ICARL system. This learning system does consistently well in all experiments run. It benefits both from its components and from their coherent integration. It allows the learning system to be applied to control problems with continuous input spaces without requiring that a good partitioning be known in advance, yet it minimizes the overhead required. By making use of the underlying continuous nature of the input space, it directly addresses the problem of long learning times; it thereby mitigates the structural credit assignment problem found in reinforcement learning systems with many partition regions.

Acknowledgements

Our thanks to Paul Rybski for building and helping to program SPTBMin, to Amy Larson and Esra Kadioglu for helping to collect results using SPTBMin, and to Charles Anderson for providing his code, adapted and used for comparison.

References

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, *9*, 31–37.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-13*, 834–846.

Geva, S., Sitte, J., & Willshire, G. (1992). A one neuron truck backer-upper. *Proceedings of the International Joint Conference on Neural Networks* (pp. 850–856).

Hougen, D. F. (1998). *Connectionist reinforcement learning for control of robotic systems*. Doctoral dissertation, Department of Computer Science and Engineering, University of Minnesota, Minneapolis.

Hougen, D. F., Gini, M., & Slagle, J. (1997). Partitioning input space for reinforcement learning for control. *Proceedings of the IEEE International Conference on Neural Networks* (pp. 755–760).

Hougen, D. F., Rybski, P. E., & Gini, M. (1999). Repeatability of real world training experiments: A case study. *Autonomous Robots*, *6*, 281–292.

Klopf, A. (1974). Brain function and adaptive systems – a heterostatic theory. *Proceedings of the International Conference on Systems, Man and Cybernetics*

Kohonen, T. K. (1989). *Self-organizing and associative memory, Third Edition*. Berlin: Springer-Verlag.

Koza, J. R. (1992). A genetic approach to finding a controller to back up a tractor-trailer truck. *Automatic Control Conference* (pp. 2307–2311).

Kretchmar, R. M., & Anderson, C. W. (1997). Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. *Proceedings of the IEEE International Conference on Neural Networks* (pp. 834–837).

Maričić, B. (1991). Genetically programmed neural network for solving pole-balancing problem. *Proceedings of the 1991 International Conference on Artificial Neural Networks* (pp. 1273–1276).

Martinetz, T. M., Ritter, H. J., & Schulten, K. J. (1989). 3D-neural-net for learning visuomoter-coordination of a robot arm. *Proceedings of the International Joint Conference on Neural Networks* (pp. 351–356). Lawrence Erlbaum.

Nigrin, A. (1990). SONNET: A self-organizing neural network that classifies multiple patterns simultaneously. *Proceedings of the International Joint Conference on Neural Networks* (pp. 313–318).

Rosen, B. E., Goodwin, J. M., & Vidal, J. J. (1992). Process control with adaptive range coding. *Biological Cybernetics*, *66*, 419–428.

Salganicoff, M., Ungar, L. H., & Bajcsy, R. (1996). Active learning for vision-based robot grasping. *Machine Learning*, *23*, 251–278.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22*, 123–158.

Woodcock, N., Hallam, N. J., & Picton, P. D. (1991). Fuzzy BOXES as an alternative to neural networks for difficult control problems. *Applications of Artificial Intelligence in Engineering VI* (pp. 903–919).