Partitioning Input Space for Control-Learning

Dean F. Hougen, Maria Gini, and James Slagle
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455
hougen—gini—slagle@cs.umn.edu

Abstract

This paper considers the effect of input-space partitioning on reinforcement learning for control. In many such learning systems, the input space is partitioned by the system designer. However, input-space partitioning could be learned. Our objective is to compare learned and programmed input-space partitionings in terms of the overall system learning speed and proficiency achieved. We present a system for unsupervised control-learning in temporal domains with results for both programmed and learned input-space partitionings. The trailer-backing task is used as an example problem.

1 The Learning System

Many classic control-learning systems, such as Michie and Chambers' BOXES [11] and Barto, Sutton, and Anderson's ASE/ACE system [2], rely on the partitioning of a space of continuous input variables into a fixed number of discrete regions. More recent systems, such as Fuzzy BOXES [18], have blurred the boundaries between the input regions but nonetheless rely on a partitioning of the input space. To use these systems, researchers have typically partitioned the space manually, prior to the application of the learning system. In these cases, the designer must analyze the problem to discover a suitable partitioning, or face a trade-off between a fine partitioning that permits accurate approximation of complex functions or a gross partitioning that allows for rapid learning.

The Self-Organizing Neural Network with Eligibility Traces (SONNET) scheme introduced by Hougen [4] is a general paradigm for the construction of connectionist networks that learn to control systems with a temporal component. In order to form mappings from input parameters to output responses, the system discretizes the input space by learning a partitioning of it, and learns an output response for each resulting discrete input region. SONNET systems have separate subsystems for learning input and output. The input subsystem learns input-space partitionings through self-organization and the output subsystem learns responses through the use of eligibility traces. Both input and output learning make use of topological ordering of the neural elements and associated neighborhoods, as in Kohonen's Self-Organizing Topological Feature Maps [8].

1.1 Topology and neighborhoods

Each SONNET subsystem consists of one or more artificial neural networks. For each network there is a topological ordering of the neural elements that remains constant as the network learns. Each neural element is assigned an integer tuple of the same dimensionality as the network that uniquely defines its coordinates in topology space. The existence of a network topology allows for the definition of a *distance* function for the neural elements. This is typically defined as the Cartesian distance between coordinates in topology space. For the sake of computational efficiency, however, we use the maximum difference between coordinates of the neurons. E.g. for neurons (1,1) and (2,3) in a planar network, the distance between them would be $max(|1-2|, |1-3|)$ or 2.

The distance function is used indirectly through the definition of *neighborhoods*. A neighborhood may have any width from zero (the neighborhood is restricted to the element itself) to the maximum distance between elements in topology space (the entire network is in the neighborhood) and may vary with time, typically starting large and subsequently decreasing. Formally, if U is the set of units u in the network, d the distance function defined on topology space, and W a time dependent function specifying the width of the neighborhood, then the neighborhood N of neuron n at time t is defined as

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

1.2 Input-space partitioning

Input-space partitioning in SONNET systems takes place through the self-organization principles introduced by Kohonen [8]. For this, a SONNET 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. (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. See [10].) In the present application, the input space is two dimensional and two separate one-dimensional input networks of eight neurons each are used.

SONNET 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 neural element has a weight vector consisting of one weight for each dimension of its input network. (In the present application, this vector has a single entry – i.e. each weight "vector" is really a scalar.) Typically 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 difference between numerical values). The element s that has the weight vector most closely matching the input is said to be the "winner" or the "selected" element. 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 element satisfies this equation, then one is selected by any arbitrary method. In the present study, the neuron with the lowest number (topological coordinate) is selected. The weights of the selected neural element and of all other neural elements in its neighborhood are updated to match the input even more closely using

$$\bar{w}^{new} = \bar{w}^{old} + \alpha(t)(\bar{x} - \bar{w}^{old}) \tag{3}$$

where α is a time-dependent function that determines the influence of the input vector. In general, α starts near 1 and decreases with time. In this way, there is competition amongst all the elements in a network to be the element selected and cooperation within a neighborhood as the elements change their weights towards the same target. By repeated presentations of data to the network, the network self-organizes so that closely neighboring elements have similar values for their weights (and therefore respond to similar input) while elements 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 elements from all input networks should allow for the selection of a single corresponding neural element from the output network. Further, this correspondence should be such that adjacent regions of the input space should result in the selection of output neurons adjacent in their topology space. For this reason, it is necessary for the output network to have the same dimensionality as the input space. In the present application, the one-dimensional topological coordinates of neurons in the two input networks are used as indices into a two-dimensional output network. The combination of all weights from all input networks, then, can be seen as 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.

1.3 Output-response learning

When a neural element from the output network is selected as described above in Section 1.2, it produces an output response for the SONNET system, based on the value(s) of its weight(s). These responses are given as control signals to the system for which control is being learned. Each output neural element has one weight for each dimension of the output space. These weights, like the weights of input neural elements, are typically given random values at the start of each run. Their updates, however, use a function known as the *eligibility trace*. The concept of the eligibility trace as part of a theory of brain function was presented by Klopf [7]. As discussed by Singh and Sutton [16], eligibility traces have since been used in a variety of reinforcement learning systems.

The use of an eligibility trace allows for response learning in domains in which system performance is temporally dependent on network responses and corresponding evaluations (such as terminal success and/or failure signals) are available. This eliminates the need to have a "teacher" that knows the correct control signal at each time step and thereby allows for unsupervised learning to occur.

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 element after its activity. At each time step, the eligibility value of each output neuron decays, regardless of whether or not 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 success or failure occurs, it is likely that more recent control signals are more responsible than earlier control signals and, due to the eligibility trace, are correspondingly rewarded or punished to a greater extent. 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

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

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 neural element updates its weight(s) a second time, this time based on the weight values of the other elements in its neighborhood, 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 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. In general, β decreases with time. This means that each neuron's value becomes more independent from those of its neighbors as time passes.

2 Experiments

To investigate the possible trade-offs between learned and programmed input-space partitioning, we applied systems of these two types to the same problem. For learned input-space partitionings we used the SONNET system described above. For programmed partitionings we used a similar system known as the Rapid Output Learning Neural Network with Eligibility Traces (ROLNNET), introduced by Hougen et al [5]. In this system, the input space is partitioned by the system designer and remains fixed throughout response learning. Otherwise this system functions in the same way as the SONNET system.

The problem to which both systems were applied was the trailer-backing problem. For a full description of the version of the trailer-backing problem used, see [5]. In both the SONNET and the ROLNNET systems, the output network consisted of an 8 x 8 grid of neural elements with one weight each. This output weight was thresholded to give *bang-bang* control of the steering wheels. (Bang-bang control was necessitated by the hardware used previously [5], but neither SONNET nor ROLNNET is intrinsically linked to it, as will be shown in future work.) In the SONNET system, each of dimensions of the input space (one for the hitch angle between the car and the trailer and one for the angle between the spine of the trailer and the goal) was learned independently by a different linear (one-dimensional) input network of eight neural elements. The topological ordering of the selected elements from the two input networks was then used to index into the network of output elements. In the ROLNNET system, the input space was divided into eight equally sized regions in each dimension, for a total partitioning of sixty-four regions. Each input region, then, was set to correspond with a single output element such that nearest neighbors (according to the neighborhood relation described above) are assigned adjacent regions of the input space.

Four cases that varied in the range of initial car-trailer positions and in the presentation of new positions to the learning system were studied. For each case the system was trained in a series of runs of 100 trials each.

Case 1: the car-trailer rig was started with the back of the trailer from four to seven feet from the target, with an angle to the target between $-60°$ and $+60°$, and with a hitch angle between $-15°$ and $+15°$. New initial positions were chosen randomly at the start of each trial (see below) with a uniform distribution over the entire range.

Case 2: this is identical to case 1, except that the rig was started with the back of the trailer from three to six feet from the target and with an angle to the target between $-45°$ and $+45°$.

Case 3: resembled case 1, differing only in when new initial positions were presented. In case 3, new initial positions were given only when success had been achieved with the previous position; when the system failed to reach the target, the same initial position would be repeated.

Case 4: resembled case 2, differing from it in the same way case 3 differed from case 1.

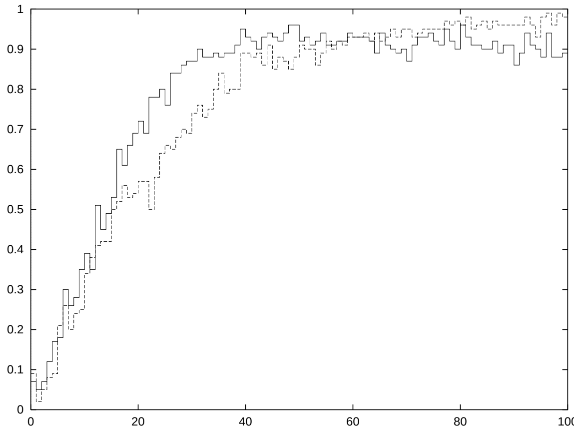
A total of 100 runs were made in simulation for each of four cases, using different random seeds for both the initial starting positions for each trial and for the random initial values of the neural weights. The results are presented in four graphs, one for each of the test cases investigated. Each graph shows the average performance of the SONNET (dashed line) and ROLNNET (solid line) systems, averaged over 100 runs of 100 trials each. The horizontal axis on each graph gives the trial number while the vertical axis gives the success rate.

As can be seen from these graphs, there was a trade-off between the SONNET approach and the simplified ROLNNET approach. True to its name, ROLNNET learned output responses rapidly, achieving its maximum proficiency at the trailer-backing task in roughly 40 trials in three of the four test cases and in fewer than 100 trials in all cases. The SONNET system, on the other hand, consistently learned more slowly near the start of the runs in all four cases and reached its maximum proficiency later than the ROLNNET system in three out of the four cases. In these three cases, however, the SONNET system was able to achieve a greater maximum proficiency. This would tend to indicate that the self-organizing system realized a preferable partitioning of the input space in these three cases, while using the same number of divisions in each dimension. This suggests that learning input-space partitionings could have a similar positive result for other problems and other reinforcement learning schemes but this remains to be verified.

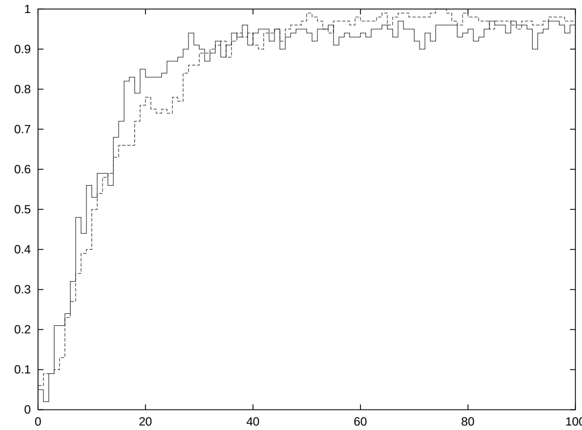
Future work includes extending these comparisons to higher dimensions. ROLNNET has already proven capable of learning with a three-dimensional input space (backing a truck with two trailers, see [6]) and SONNET has likewise proven itself in a four-dimensional input space (pole-balancing, see [4]), but direct comparisons on these tasks is still in progress. In general, we see no reason that these systems cannot be extended to higher dimensions, but doing so brings with it the "curse of dimensionality" [3]. SONNET minimizes this problem by achieving a fine partitioning where needed with a small number of total divisions but is not immune to it altogether. We are, therefore, more interested in finding ways of combining multiple networks of low dimension, rather than building individual networks of high dimension.

3 Related work

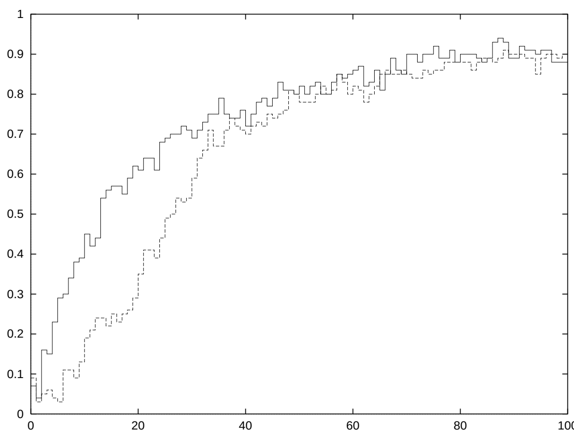
Many researchers have used one version or another of the trailer-backing task as an example problem on which to demonstrate their systems. It is beyond the scope of this paper to consider all such works or



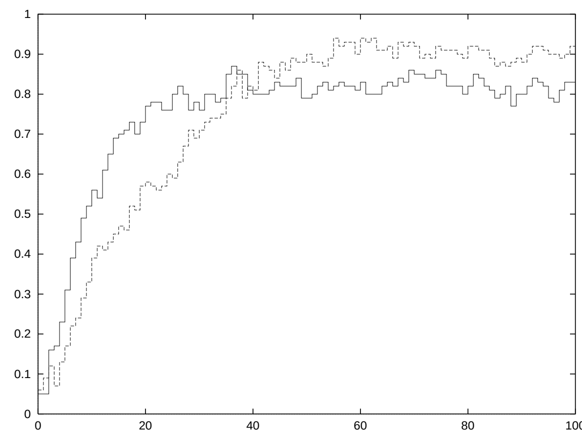
Case 1. Average performance for 100 runs of 100 trials each. Initial position: Rear of trailer 4 to 7 feet from target, angle to target $+/-60°$, angle of hitch $+/-15°$. New random position for each trial.



Case 2. Average performance for 100 runs of 100 trials each. Initial position: Rear of trailer 3 to 6 feet from target, angle to target $+/-45°$, angle of hitch $+/-15°$. New random position for each trial.



Case 3. Average performance for 100 runs of 100 trials each. Initial position: Rear of trailer 4 to 7 feet from target, angle to target $+/-60°$, angle of hitch $+/-15°$. New random position only following success; previous position repeated after failure.



Case 4. Average performance for 100 runs of 100 trials each. Initial position: Rear of trailer 3 to 6 feet from target, angle to target $+/-45°$, angle of hitch $+/-15°$. New random position only following success; previous position repeated after failure.

Figure 1: Simulation data. Dashed lines are SONNET results. Solid lines are ROLNNET results.

even those in which the demonstrated system was a learning system (as opposed to a planning system, etc.). This paper is concerned with the combination of input space partitioning and output response learning for control systems.

Michie and Chambers [11], Barto, Sutton, and Anderson [2], Woodcock, Hallam, and Picton [18], Kong and Kosko [9], and others have introduced systems which use a fixed partitioning of the input space to learn control of various systems. Anderson [1] also explored the difficulty of moving from a system with a user-defined partitioning to a system in which an evaluation of the input must be learned. Anderson uses a layer of neural elements to learn "features" of the input space through a variant of back-propagation. While not a partitioning per se, these features do allow his system to learn to treat related input vectors as similar to one another. Anderson also found a trade-off between learning time and performance.

Simons, Van Brussel, De Schutter, and Verhaert [15] and Moore and Atkeson [12] have introduced learning systems that can adaptively move from coarse to fine partitionings. In our system in contrast, the number of partitionings is fixed; only their extent is changed.

Rosen, Goodwin, and Vidal [14] independently developed a system known as Adaptive Coarse Coding (ACC) that adjusts its input-space partitionings in a way similar to that of SONNET systems. ACC, however, was only used to refine partitionings initialized by the system designer, rather than to learn them from a random initialization. Rosen et al found that such refinements could improve system performance. Differences with SONNET include the lack of inter-neural cooperation in the learning of output in ACC systems.

The systems of Barto et al and Rosen et al also use eligibility traces to learn output responses, as do many other reinforcement learning systems (see [16]). Many of these systems, including these two, also use additional components, such as temporal difference methods [17], to aid in this process. Our system does not currently use these additional methods, although we are considering such combinations.

The use of a topological ordering of neural elements for self-organization has been used by Kohonen [8]. This has been extended to include output learning for control by Ritter, Martinetz, and Schulten [13]. These ideas were brought together with the use of the eligibility trace by Hougen [4] and the present paper is believed to be the first to explicitly consider the trade-off between a completely learned partitioning and one assigned by the system designer.

References

[1] C. W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37, 1989.

[2] A. R. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[3] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[4] D. F. Hougen. Use of an eligibility trace to self-organize output. In *Science of Artificial Neural Networks II, Proc. SPIE*, volume 1966, pages 436–447, 1993.

[5] D. F. Hougen, J. Fischer, M. Gini, and J. Slagle. Fast connectionist learning for trailer backing using a real robot. In *IEEE Int'l Conf. on Robotics and Automation*, pages 1917–1922, 1996.

[6] D. F. Hougen, M. Gini, and J. Slagle. Rapid, unsupervised connectionist learning for backing a robot with two trailers. In *IEEE Int'l Conf. on Robotics and Automation*, 1997.

[7] A. Klopf. Brain function and adaptive systems – a heterostatic theory. In *Proc. of the Int'l Conf. on Systems, Man, and Cybernetics*, 1974.

[8] T. K. Kohonen. *Self-organizing and associative memory*. Springer-Verlag, Berlin, 3rd edition, 1989.

[9] S.-G. Kong and B. Kosko. Adaptive fuzzy systems for backing up a truck-and-trailer. *IEEE Trans. on Neural Networks*, 3(2):211–223, 1992.

[10] T. M. Martinetz, H. J. Ritter, and K. J. Schulten. 3d-neural-net for learning visuomoter-coordination of a robot arm. In *Int'l Joint Conf. on Neural Networks*, volume II, pages 351–356. Erlbaum, 1989.

[11] D. Michie and R. Chambers. Boxes: an experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence*. Oliver and Boyd, Edinburgh, 1968.

[12] A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233, 1995.

[13] H. Ritter, T. Martinetz, and K. Schulten. *Neural computation and self-organizing maps: an introduction*. Addison-Wesley, Reading, MA, 1992.

[14] B. E. Rosen, J. M. Goodwin, and J. J. Vidal. Process control with adaptive range coding. *Biological Cybernetics*, 66:419–428, 1992.

[15] J. Simons, H. V. Brussel, J. D. Schutter, and J. Verhaert. A self-learning automaton with variable resulution for high precision assembly by industrial robots. *IEEE Trans. on Automatic Control*, 27(5):1109–1113, October 1982.

[16] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[17] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[18] N. Woodcock, N. J. Hallam, and P. D.Picton. Fuzzy BOXES as an alternative to neural networks for difficult control problems. *Artificial Inteligence in Engineering*, pages 903–919, 1991.