Strategy Learning with Multilayer Connectionist Representations [1]

Charles W. Anderson

Department of Computer Science
Colorado State University
Fort Collins, CO 80523

anderson@cs.colostate.edu
http://www.cs.colostate.edu/˜anderson

Abstract
Results are presented that demonstrate the learning and fine-tuning of search strategies using connectionist mechanisms. Previous studies of strategy learning within the symbolic, production-rule formalism have not addressed fine-tuning behavior. Here a two-layer connectionist system is presented that develops its search from a weak to a task-specific strategy and fine-tunes its performance. The system is applied to a simulated, real-time, balance-control task. We compare the performance of one-layer and two-layer networks, showing that the ability of the two-layer network to discover new features and thus enhance the original representation is critical to solving the balancing task.

1. Introduction

A *strategy* is a method for guiding the search for a solution to a problem, where a solution typically consists of a sequence of actions. Most research on the learning of strategies has used symbolic forms of representation, such as production rules, which have been useful in modeling some of the conscious steps that humans appear to follow in acquiring strategies (Klahr, Langley, & Neches, 1987). Not included in these studies is the minute adjustment of strategies exhibited by experts performing real-time control tasks (Anzai, 1987). Such fine-tuning is often relegated to numerical representations, with the assumption that parameter adjustment is useful only after a good representation and the fundamental steps for a successful strategy have been found.

In this report, a numerical connectionist learning system is presented that develops a weak search strategy into a fine-tuned, task-specific strategy. The learning system is an extension of the system defined by Barto, Sutton, and Anderson (1983) and applied by them to a difficult pole-balancing control task. This task, described below, is also used here. The extension of Barto et al.'s system is to *multilayer networks*. With multilayer networks, the learning system can enhance its representation by learning new features that are required by or that facilitate the search for the task's solution. Detailed results are presented for a two-layer network.

The objective of this initial stage of experimentation is not to outperform other methods of solving these tasks, but rather to achieve a greater understanding of the utility of connectionist representations as tools for the learning of problem-solving strategies and skills. In particular, we are interested in the ability of connectionist systems to develop their representations by learning useful new features.

2. The Pole-Balancing Task

The pole-balancing task is often used as an example of the inherently unstable, multiple-output, dynamic systems present in many balancing situations, like two-legged walking and the aiming of a rocket thruster. It has been used to demonstrate modern and classical control-engineering techniques (Cannon, 1967; Cheok and Loh, 1987).

Control-engineering techniques involve detailed analyses of the system to be controlled. When the lack of knowledge about a task precludes such analyses, a control system must adapt as information

[1] Originally appeared in *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA, 1987, pp. 103–114, and also as GTE Laboratories Technical Report TR87-509.3.

is gained through experience with the task. To investigate the use of learning methods for such cases, we assume that very little is known about the pole system, including its dynamics. The pole system is viewed as a black box generating as output the pole's state and a performance evaluation and that accepts as input a control action.

The pole-balancing task involves a pole hinged to the top of a wheeled cart that travels along a track, as shown in Figure 1. The pole is constrained to move within the vertical plane. The state at time t is

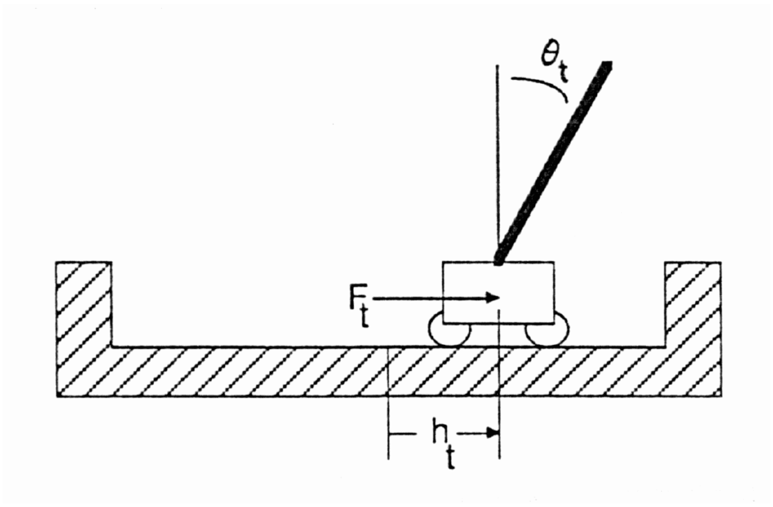


Figure 1: The Pole-Balancing Environment

specified by four real-valued variables:

$$\begin{aligned}
h_t &= \text{the horizontal position of the cart, relative to the track, in meters,}\\
\dot{h}_t &= \text{the horizontal velocity of the cart, in meters/second,}\\
\theta_t &= \text{the angle between the pole and vertical, clockwise being positive, in degrees,}\\
\dot{\theta}_t &= \text{the angular velocity of the pole, in degrees/second.}
\end{aligned}$$

The cart-pole system was simulated using the equations of motion given in the appendix. Resulting discrete-time variables are labeled $h[t]$, $\dot{h}[t]$, $\theta[t]$, and $\dot{\theta}[t]$.

The goal is to apply a sequence of right and left forces of fixed magnitude to the cart such that the pole is balanced and the cart does not hit the end of the track. A zero magnitude force is not permitted. The state of the cart-pole system must be kept out of certain regions of the state space. There is no unique solution—any trajectory through the state space that does not pass through the regions to be avoided is acceptable. The only information regarding the goal of the task is provided by the failure signal, $r[t]$, which signals either the pole falling past $\pm 12^o$ or the cart hitting the bounds of the track at ± 2.4 m. The failure signal is defined as

$$r[t] = \begin{cases} -1, & \text{if } |\theta[t]| > 12^o \text{ or } |h[t]| > 2.4 \text{ m;}\\ 0, & \text{otherwise.} \end{cases}$$

Because the pole is restricted to a small angular range about the vertical position ($\pm 12^o$), successful state trajectories can be determined by actions generated by a linear function of the state variables. The formation of a useful evaluation function of states, however, requires a nonlinear function. As shown below, the multilayer system is able to learn a nonlinear evaluation function.

The state of the cart-pole system was presented to the learning system as scaled versions of the state variables:

$$\begin{aligned} x_1[t] &= \tfrac{1}{4.8}(h[t] + 2.4), \\ x_2[t] &= \tfrac{1}{3}(\dot{h}[t] + 1.5), \\ x_3[t] &= \tfrac{1}{24}(\theta[t] + 12), \\ x_4[t] &= \tfrac{1}{230}(\dot{\theta}[t] + 115). \end{aligned}$$

An additional input, $x_5[t]$, with a constant value of 0.5 is provided. Inputs $x_1[t]$ and $x_3[t]$ range from 0 to 1, while $x_2[t]$ and $x_4[t]$ can exceed 0 and 1. This scaling accomplishes two things. Since the input terms $x_i[t]$ are factors in the equations for modifying the learning system, terms with predominantly larger magnitudes will have a greater influence on learning than will other terms. To remove this bias all input terms are scaled to lie within the same range. Secondly, since the values of the state variables are centered at zero, and since initially (before new features are learned) the learning system generalizes linearly, the correct action for positive θ and $\dot{\theta}$ would transfer to negative θ and $\dot{\theta}$ in the right way—the correct action for negative θ and $\dot{\theta}$ *is* the negative of the correct action for positive θ and $\dot{\theta}$. If the state variables are used directly, these correct generalizations make the task much easier. Shifting the state variables allowed us to test the learning system with an input representation that is less tailored to the task.

3. Specification of the Connectionist Learning System

The learning system consists of two connectionist networks, called the *action network* and the *evaluation network*. The action network generates the system's behavior—it decides which action to apply for a given state of the cart-pole system. The evaluation network learns an *evaluation function* of the pole's states. During learning, both networks are simultaneously modified. Figure 2 shows the networks and defines their computations by equations grouped according to Langley's (1985) three components of strategy-learning systems: behavior generation, credit assignment, and modification. In this section the computations are summarized (see Anderson, 1986, Barto et al., 1983, and Rumelhart, Hinton, & Williams, 1986, for further explanations).

3.1 Generation of Behavior—Search

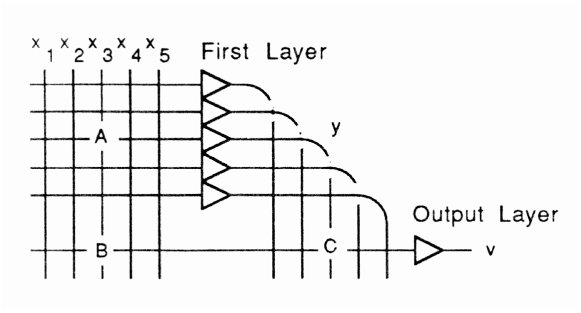
The pole-balancing task differs from those commonly used to demonstrate strategy-learning methods in that it is a *real-time* task (Anzai, 1987). The learning system cannot control the timing of the task's state transitions, therefore the response time of the learning system is critical. If a model of the task is known it can be analyzed or searched off-line, but here we assume that a model (the dynamics) of the cart-pole system are unavailable to the learning system. Standard AI search techniques (breadth-first, depth-first, best-first, etc.) do not perform well for such problems for the following reasons:

- the effects of different actions on the current state cannot be tested,
- an arbitrary amount of time cannot be consumed in choosing an action,
- backtracking cannot be performed.

The system described here demonstrates a type of adaptive probabilistic search that can quickly select an action (due to the parallelism inherent in connectionist networks). The action probabilities are calculated by the action network given the pole's current state.

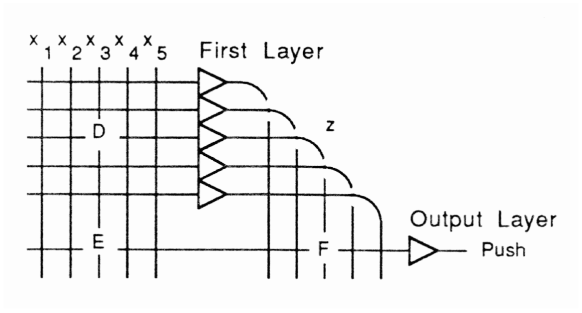
The method of learning used in the connectionist system is not restricted to probabilistically generated behavior. Any other search technique could be used, but it would require additional mechanisms for integrating the search with the heuristic advice generated by the evaluation function. With the approach described here, the updating of action probabilities after each step interfaces naturally with the probabilistic search performed by the action network. Although the search is initially uniformly random over the possible actions, it gradually gains direction with learning as the action probabilities become biased in favor of more successful actions. Thus, the action function, which determines action probabilities based on the state of the task, is the connectionist system's counterpart to symbolic rules for expressing action heuristics.

Evaluation Network



A, B, C are matrices of connection weights. Individual weights are labelled a_{ij}, b_i, c_i.

Action Network



D, E, F are matrices of connection weights. Individual weights are labelled d_{ij}, e_i, f_i.

Output: State Evaluation

$$
\begin{aligned}
y_i[t_1,t_2] &= g(\sum_{j=1}^{5} a_{ij}[t_1]x_j[t_2]),\\
v[t_1,t_2] &= \sum_{i=1}^{5} b_i[t_1]x_i[t_2] + \sum_{i=1}^{5} c_i[t_1]y_i[t_1,t_2],\\
g(s) &= \frac{1}{1+e^{-s}},
\end{aligned}
$$

Double time dependencies are used to avoid instabilities in the updating of weights (Sutton, 1984)

Action Evaluation: Failure Signal Plus Change in State Evaluation

$$
\hat{r}[t+1] = \begin{cases} 0, & \text{if state at time } t+1 \text{ is a start state;}\\ r[t+1] - v[t,t], & \text{if state at time } t+1 \text{ is a failure state;}\\ r[t+1] + \gamma v[t,t+1] - v[t,t] & \text{otherwise,} \end{cases}
$$

Modification

$$
\begin{aligned}
b_i[t+1] &= b_i[t] + \beta\,\hat{r}[t+1]\,x_i[t],\\
c_i[t+1] &= c_i[t] + \beta\,\hat{r}[t+1]\,y_i[t,t],\\
a_{ij}[t+1] &= a_{ij}[t] + \beta_h\,\hat{r}[t+1]\,y_i[t,t]\cdot\\
&\quad (1-y_i[t,t])\,\mathrm{sgn}(c_i[t])\,x_j[t],
\end{aligned}
$$

Parameters

$$0 < \gamma \leq 1; \quad \beta,\, \beta_h > 0$$

Output: Action

$$
\begin{aligned}
z_i[t] &= g(\sum_{j=1}^{5} d_{ij}[t]\,x_j[t]),\\
p[t] &= g(\sum_{i=1}^{5} e_i[t]\,x_i[t] + \sum_{i=1}^{5} f_i[t]\,z_i[t]),\\
q[t] &= \begin{cases} 1, & \text{with probability } p[t];\\ 0, & \text{with probability } 1-p[t] \end{cases}\\
Push[t] &= \begin{cases} 10, & \text{if q[t] = 1;}\\ -10, & \text{if q[t] = 0} \end{cases}
\end{aligned}
$$

Modification

$$
\begin{aligned}
e_i[t+1] &= e_i[t] + \rho\,\hat{r}[t+1]\cdot\\
&\quad (q[t]-p[t])\,x_i[t],\\
f_i[t+1] &= f_i[t] + \rho\,\hat{r}[t+1]\cdot\\
&\quad (q[t]-p[t])\,z_i[t],\\
d_{ij}[t+1] &= d_{ij}[t] + \rho_h\,\hat{r}[t+1]\,z_i[t]\cdot\\
&\quad (1-z_i[t])\,\mathrm{sgn}(f_i[t])\cdot\\
&\quad (q[t]-p[t])\,x_j[t]
\end{aligned}
$$

Parameters

$$\rho,\, \rho_h > 0$$

Figure 2: Specification of Evaluation and Action Networks

3.2 Credit Assignment

The failure signal is the only information available to the learning system regarding its performance. The learning system is not limited to learning from failures. When defining a task, any ranking can be placed on the task's states by specifying r. For example, if we want to assume that a vertical pole is desirable, r could be set to 1 for states with an angle near zero.

To assign credit to the individual actions of the action sequence preceding a failure signal, an evaluation function of states is learned. The evaluation network learns this function using a generalization of Samuel's (1959) method called the Adaptive Heuristic Critic (AHC), a member of the class of prediction methods called *temporal difference* methods (Sutton, 1987). The AHC algorithm develops an evaluation function whose value v for a given state is a prediction of future discounted failure signals. Changes in v due to problem-state transitions are combined with the failure signal r to form $\hat{r}$. For all states except those corresponding to failure, r is zero and $\hat{r}$ is just the difference between successive values of v, i.e., predictions of failure. After learning an evaluation function that reliably predicts failure, a positive change in the prediction of failure means the pole entered a state from which failure occurs less often or further in the future than from the previous state. An action is considered desirable if it leads to a positive change in failure prediction and undesirable if it leads to a negative change. Since the action and evaluation networks learn simultaneously, credit and blame is often incorrectly assigned to actions in the initial stages of learning. Modification methods must take this into account.

3.3 Modification

The AHC algorithm specifies how the output layer of the evaluation network is modified. It is a type of *supervised learning*, with $\hat{r}$ playing the role of an error. A positive (or negative) value of $\hat{r}$ results in an increase (or decrease) of the preceding state's evaluation, effectively shifting evaluations to earlier states. The modification procedure for the first layer of the evaluation network is similar to the error back-propagation method used by Rumelhart, Hinton, and Williams (1986). Errors are assigned to the units of the first layer based on $\hat{r}$ and on the influence the units of the first layer have on the output unit.

Supervised-learning methods cannot be used for the action network, because correct actions are not known. Learning must be based on the inaccurate, time-varying, and delayed evaluations of the adaptive evaluation function. The numerical approach to learning with such performance feedback is called *reinforcement learning*. Reinforcement-learning methods have been developed for mathematical learning theory (Bush & Estes, 1959), learning automata (Narendra and Thathachar, 1974), learning control (Mendel and McLaren, 1970), and connectionist systems (Barto, 1985; Barto, Sutton, & Brouwer, 1981).

The reinforcement-learning method used here operates as follows. The output layer is modified in a way that increases the probability of an action that is followed by a positive value of $\hat{r}$, and decreases the probability of an action followed by a negative $\hat{r}$. The change in probability is proportional to the magnitude of $\hat{r}$ and to the difference between the action and the expected value of the action. Thus, the results of unusual actions have more impact on the adjustment of weights than do other actions. Sutton (1984) found this factor to be helpful in learning to discriminate among similar input vectors.

The equations for updating the action network's first layer is again a form of error back-propagation. Once $\hat{r}$ becomes a good evaluation of the previous action, the role of an error is played by the product of $\hat{r}$ and the difference between the action and its expected value. Disregarding the different errors that are back-propagated by the two networks, the methods for modifying the first layers of the two networks are the same.

4. Results

Each experiment consisted of a number of *runs* that differed only in the seed values for the pseudo-random number generator. Performance measurements were averaged across runs. Each run consisted of a number of *trials*, each starting with the cart-pole system set to a state chosen at random, and ending with the appearance of the failure signal. At the start of every run the connection weights were initialized to random values between -0.1 and 0.1.

Interest in the development of useful features by the first-layer units of a network led to a series of experiments using one-layer and two-layer networks. To compare results, the parameters of the modification equations were roughly optimized for each kind of network by manually searching for values resulting in best performance. After testing approximately 30 sets of values, the best were used in 10 runs of 500,000 time steps in duration. For the one-layer networks, the best parameter values were found to be $\rho = 0.5$, $\beta = 0.05$, and $\gamma = 0.9$. For the two-layer networks, the values were $\rho = 1.0$, $\rho_h = 0.2$, $\beta = 0.2$, $\beta_h = 0.05$, and $\gamma = 0.9$.

The results of the pole-balancing experiments are summarized in Figure 3, where the logarithm of the number of steps per trial is plotted versus the number of trials. The curves are smoothed by averaging the steps per trial into bins of 100 trials. As a baseline, the performance of a non-learning strategy of randomly selecting actions was measured over 500,000 steps to be an average of about 12 steps per trial, with an average of 40,000 total failures.

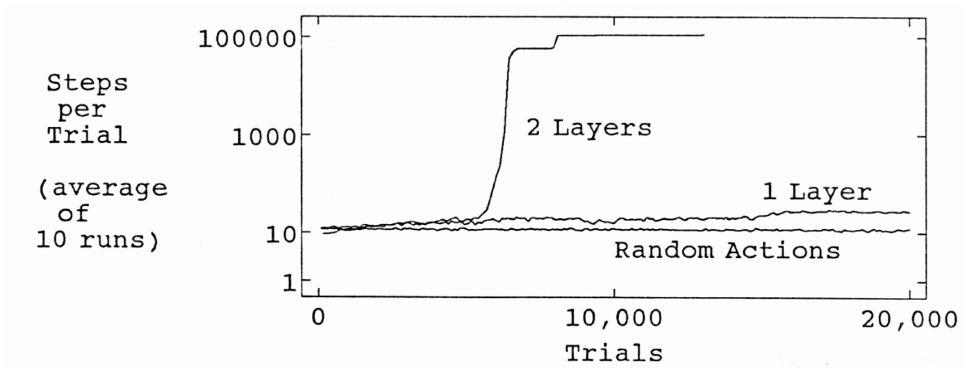


Figure 3: Learning Curves: Logarithm of Steps per Trial vs. Trials

The one-layer system did not perform significantly better than the random strategy, averaging 27,000 failures per run and about 16 steps per trial at the end of the runs. On the other hand, the two-layer system solved the pole-balancing task, achieving an average of 120,000 steps per trial, or about 24 minutes of simulated time. Balancing time is limited by the termination of all runs after 500,000 steps. An average of 8,000 failures occurred.

5. Discussion

The two-layer system can only begin to perform better than the one-layer system after helpful features are formed. This accounts for the slow growth in the two-layer system's learning curve during the first 5,000 trials. Once the features are formed, performance quickly surpasses that of the one-layer networks by at least two orders of magnitude.

What is actually learned by the networks in these experiments? A common problem with connectionist representations is the difficulty of understanding how a network's input values, connection weights, and structure interact in determining the network's output. Figure 4 is a graphical representation of what has been learned by the networks. Since the networks implement functions of four variable inputs, a projection in two dimensions is displayed. The evaluation and action-probability functions are plotted versus θ and $\dot{\theta}$ for three different values of h and $\dot{h}$.

The learned evaluation function is very helpful in solving this task: it forms a diagonal ridge with low points at $+\theta, +\dot{\theta}$ and $-\theta, -\dot{\theta}$, the regions of the state space in which the pole is likely to fall. The surface also varies with h and $\dot{h}$ in the appropriate manner.

For most states, the action function generates probabilities very close to either 0 or 1, representing pushes left or right, with a relatively narrow transition zone where action selection is less deterministic. The transition zone is analogous to the diagonal switching curve of the bang-bang controller that is time-optimal for a linearized model of the cart-pole system. The surface varies appropriately for different values of h and $\dot{h}$: a shift of the switching curve towards negative θ for positive h results in the balancing of the pole to the left of vertical when the cart is towards the right end of the track, producing a preponderance of left pushes which bring the cart back to the middle. The reverse argument applies to behavior towards the left end of the track.

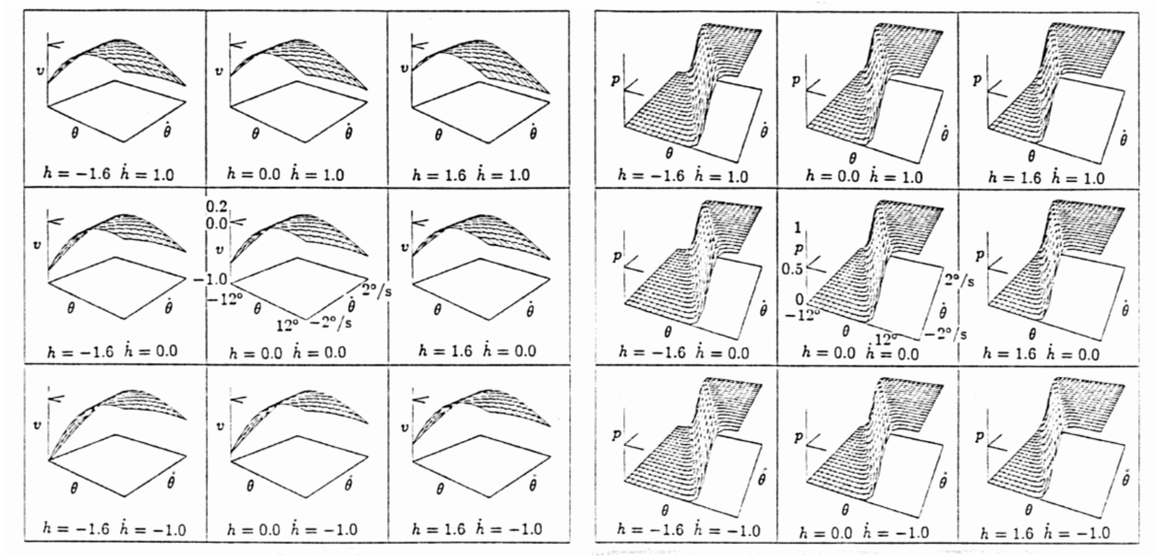


Figure 4: Evaluation and Action Functions as Projections Onto the $\theta, \dot{\theta}$ Plane.

A similar display of the output of first-layer units shows features that have been learned. Figure 5 shows the output of one unit from each network. Other units either learned similar features or features with relatively constant values over all states. The evaluation-network feature (on the left in Figure 5) acquired a positive influence on the evaluation. In combination with the negative effect of θ and $\dot{\theta}$

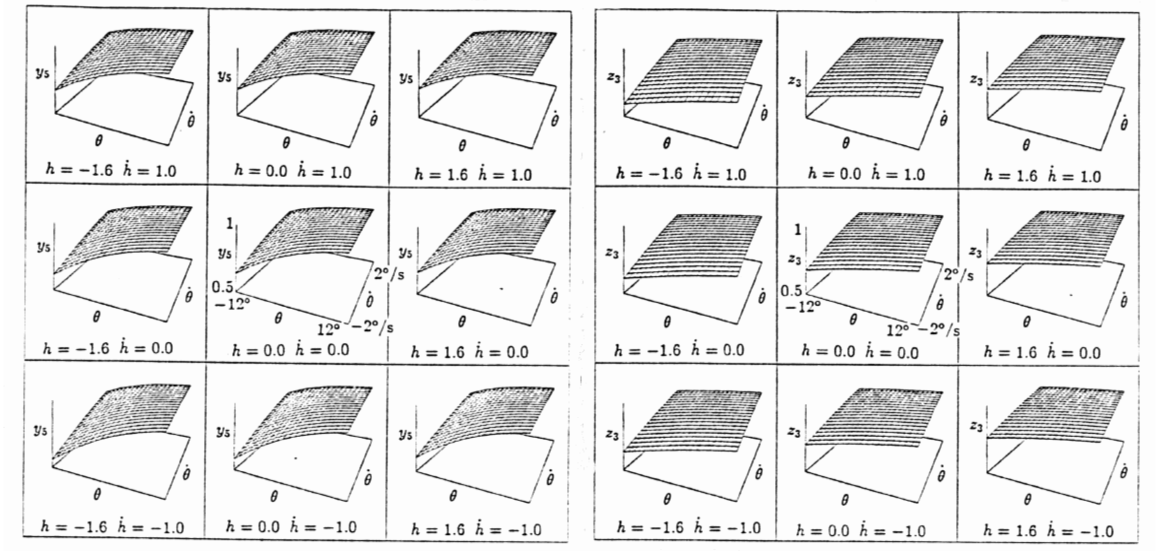


Figure 5: New Features Learned by Hidden Units in Evaluation and Action Networks

acquired by the output unit this results in the formation of the ridge in the evaluation function. Without the new feature, a peak cannot be formed. The right half of Figure 5 is a feature from the action network, but by the end of the run it did not have a significant effect on the action probabilities. Perhaps it was helpful in initially learning a good state-to-action mapping, but later lost its usefulness when the weights connecting the network's inputs to the output unit began to duplicate its effect, made possible by the linearity of the successful action function.

To express a feature in a more understandable form, we can approximate it with a symbolic rule. For example, the evaluation-network feature if Figure 5 can be expressed as:

> *If* the angle and angular velocity are large,
> *then* the pole is falling to the right.

Of course, the phrase "falling to the right" is simply a label created for illustration—the system has no understanding of the concept of "falling" or "to the right".

Similarly, the action function is crudely approximated by the following rules:

> *If* the pole is falling to the right (left),
> *then* push to the right (left).
>
> *If* the pole is a bit to the left (right) of vertical and moving up,
> *then* push to the right (left).

The effect of the learned evaluation function on the assignment of credit can also be approximated by a set of rules. For example:

> *If* the pole moved towards (away from) the vertical position,
> *then* the action is desirable (undesirable).
>
> *If* the pole's angular velocity decreased (increased),
> *then* the action is desirable (undesirable).

These rules do not capture the interdependency of the pole's angle and angular velocity, which requires more complex conditions, such as:

> *If* the state moves towards (away from) the negatively-sloped diagonal in the angle, angular velocity plane,
> *then* the action is desirable (undesirable).

A further increase in the accuracy with which the rules represent the learned functions requires numerically precise conditions to express the placement of the diagonal and its variability with the cart's position and velocity. However, such rules would be harder to understand. Perhaps a form of representation that is a synthesis of symbolic rules and numerical functions could provide the necessary balance when both fine-tuned accuracy and understandiblity are desired. Alternatively, as shown here, numerical representations can be used to govern learning and performance and the functions approximated by translating them to symbolic rules. Automatic translation procedures would be very useful both for programming initial information into a connectionist system and for expressing learned information in an understandable form.

6. Conclusion

The two-layer connectionist learning system reliably learned to balance the pole. Deficiencies in the input representation of the pole's state were overcome by the learning of new features. The weak, initial strategy of random action-selection was transformed into a nearly deterministic choice of the better action for each state. An action is quickly chosen by calculating the output of the action network once for each action. This approach is well-suited to refining the control of a real-time task, and can do so even when given an inappropriate representation and with few clues as to what a good strategy would be. With minor modifications, this learning system has also been applied to the Tower of Hanoi puzzle (Anderson, 1986), demonstrating that it is not limited to numerical control tasks.

A question not addressed here is how well the learned solution transfers to related tasks. Selfridge, Sutton, & Barto (1985) considered transfer in the one-layer system. With two-layers, a different kind of transfer is possible, due to the acquisition of features that are useful for a number of tasks independent of their associations to actions.

A number of alterations remain to be tried that will probably result in faster learning. Relatively minor changes can be made, such as adapting the parameter values during learning or using averages of past input values in updating weights rather than the single input from the previous step as done here. A more major change would be the development of weight-modification rules other than the error back-propagation method. Other numerical optimization techniques or other less-formal methods developed with representation development issues in mind might be found to be useful. A potentially great improvement in performance might result from the addition of a mechanism for learning a model of the problem even before goals are specified. Such a model can be used either to search for actions in an off-line, look-ahead type search, or to form a prediction of future evaluations (Sutton & Pinette, 1985).

Appendix: Cart-pole Simulation

The dynamics of the cart-pole system are given by the following equations of motion. The values of θ, $\dot{\theta}$, and $\ddot{\theta}$ are assumed to be in radians, radians/s, and radians/s^2, respectively, whereas in the body of this report all angular measurements are given in degrees.

$$\ddot{\theta}_t = \frac{g \sin\theta_t + \cos\theta_t \left[\frac{-Push_t - m_p l \dot{\theta}_t^2 \sin\theta_t}{m_c + m_p} \right]}{l \left[\frac{4}{3} - \frac{m_p \cos^2\theta_t}{m_c + m_p} \right]},$$

$$\ddot{h}_t = \frac{Push_t + m_p l \left[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t \right]}{m_c + m_p},$$

where

$$\begin{array}{rclcl} Push_t & = & \pm 10 \text{ Nt} & = & \text{output of action network}, \\ m_c & = & 1.0 \text{ kg} & = & \text{mass of the cart}, \\ m_p & = & 0.1 \text{ kg} & = & \text{mass of the pole}, \\ l & = & 0.5 \text{ m} & = & \text{distance from center of mass of pole to the pivot}, \\ g & = & 9.8 \text{ m/s}^2 & = & \text{acceleration due to gravity}, \end{array}$$

and sin and cos take angles measured in radians. This system was simulated on a digital computer by numerically approximating the equations of motion using Euler's method with a time step $\tau = 0.02$ seconds and the following discrete-time state equations:

$$\begin{array}{rcl} h[t+1] & = & h[t] + \tau \dot{h}[t], \\ \dot{h}[t+1] & = & \dot{h}[t] + \tau \ddot{h}[t], \\ \theta[t+1] & = & \theta[t] + \tau \dot{\theta}[t], \\ \dot{\theta}[t+1] & = & \dot{\theta}[t] + \tau \ddot{\theta}[t]. \end{array}$$

The sampling rate of the cart-pole system's state and the rate at which control forces are applied are the same as the basic simulation rate, i.e., 50 Hz.

Acknowledgements

Rich Sutton, Oliver Selfridge, and Glenn Iba provided valuable help in improving the clarity and conciseness of this report. This research was supported in part by the Air Force Office of Scientific Research and the Avionics Laboratory (Air Force Wright Aeronautical Laboratories) through contract F33615-83-C-1078.

References

Anderson, C. W. (1982). *Feature generation and selection by a layered network of reinforcement learning elements: some initial experiments.* (Technical Report COINS 82–12). Amherst, MA: University of Massachusetts, Department of Computer and Information Science.

Anderson, C. W. (1986). *Learning and problem solving with multilayered connectionist systems.* Doctoral Dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA.

Anzai, Y. (1987). Doing, understanding, and production systems. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development*, Cambridge, MA: MIT Press.

Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology, 4*, 229–256.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics, 13*, 835–846.

Barto, A. G., Sutton, R. S., & Brouwer, P. S. (1981). Associative search network: a reinforcement learning associative memory. *Biological Cybernetics, 40*, 201–211.

Bush, R. R. & Estes, W. K. (Eds.). (1959). *Studies in mathematical learning theory*, Stanford University Press.

Cannon, R. H., Jr. (1967). *Dynamics of Physical Systems.* McGraw-Hill, Inc.

Cheok, K. C. & Loh, N. K. (1987). A ball-balancing demonstration of optimal and disturbance-accommodating control. *IEEE Control Systems Magazine*, February, 1987, 54–57.

Klahr, D., Langley, P., & Neches, R. (Eds.). (1987). *Production system models of learning and development*, Cambridge, MA: MIT Press.

Langley, P. (1985). Learning to search: from weak methods to domain-specific heuristics. *Cognitive Science, 9*, 217–260.

Mendel, J. M. & McLaren, R. W. (1970). Reinforcement learning control and pattern recognition systems. In J. M. Mendel & K. S. Fu (EDS.), *Adaptive, learning and pattern recognition systems: theory and applications.* Academic Press.

Narendra, K. S. & Thathachar, M. A. L. (1974). Learning automata—a survey. *IEEE Transactions on Systems, Man, and Cybernetics, 4*, 323–334.

Plaut, D. C., Nowlan, S. J., & Hinton, G. E. (1986). *Experiments on learning by back propagation.* (Technical Report CMU-CS-86-126). Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

Rumelhart, D. E., Hinton, G. E., & Williams, R. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & the PDP research group (Eds.), *Parallel distributed processing: explorations in the microstructure of cognition*, Cambridge, MA: Bradford Books.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development, 3*, 210–229.

Selfridge O., Sutton, R. S. & Barto, A. G. (1985). Training and tracking in robotics. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA.

Sutton, R. S. (1984). *Temporal aspects of credit assignment in reinforcement learning.* Doctoral Dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA.

Sutton, R. S. (1987). *Learning to predict by the methods of temporal differences.* (Technical Report TR87-509.1). Waltham, MA: GTE Laboratories Incorporated.

Sutton, R. S. & Pinette, B. (1985). The learning of world models by connectionist networks. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 54–64). Irvine, CA.