Reinforcement Learning Architectures

Richard S. Sutton
GTE Laboratories Incorporated
Waltham, MA 02254
sutton@gte.com

Abstract

Reinforcement learning is the learning of a mapping from situations to actions so as to maximize a scalar reward or reinforcement signal. The learner is not told which action to take, as in most forms of learning, but instead must discover which actions yield the highest reward by trying them. In the most interesting and challenging cases, actions affect not only the immediate reward, but also the next situation, and through that all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning. In this paper I present a brief overview of the development of reinforcement learning architectures over the past decade, including reinforcement-comparison, actor-critic, and Q-learning architectures. Finally, I present Dyna, a class of architectures based on reinforcement learning but which go beyond trial-and-error learning to include a learned internal model of the world. By intermixing conventional trial and error with hypothetical trial and error using the world model, Dyna systems can plan and learn optimal behavior very rapidly.

1 Reinforcement Learning

The reinforcement learning problem is summarized in Figure 1. On some short time cycle, a learning agent receives sensory information from its environment and chooses an action to send to the environment. In addition, the learning agent receives a special signal from the environment called the *reward*. Unlike the sensory information, which may be a large feature vector, or the action, which may also have many components, the reward is a single real-valued scalar, a number. The goal of learning is the maximization of the cumulative reward received over time. Reinforcement learning systems can be defined as learning systems designed for and that perform well on this problem. Informally, we define reinforcement learning as learning by trial and error from performance feedback—i.e., from feedback that evaluates the behavior generated by the learning agent but does not indicate correct behavior. The term "reinforcement learning" appears to have been coined by Minsky (1961), and independently in control theory by Waltz and Fu (1965). The idea of course comes originally from animal-learning psychology.

One might object to the problem formulation in Figure 1 on the grounds that all possible goals have been reduced to a scalar reward. Although this appears limiting, in practice it has proved to be a useful

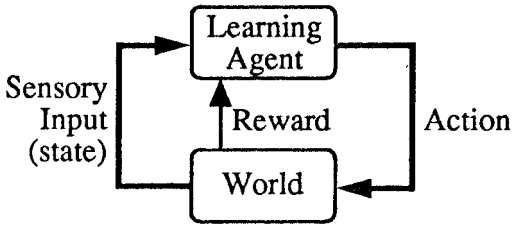

Figure 1. The Reinforcement Learning Problem. The goal is to maximize cumulative reward.

way of structuring the problem. Some examples of goals formulated in this way are:

- Foraging: Reward is positive for finding food objects, negative for energetic motion, slightly negative for standing still.
- Pole-balancing (balancing a pole by applying forces to its base): The reward is zero while the pole is balanced, and then becomes -1 if the pole falls over or if the base moves too far out of bounds.
- Towers of Hanoi: Reward is positive for reaching the goal state.
- Recycling Robot: Reward is positive for dropping soda cans in the recycling bin, negative for bumping into things, more negative for bumping hard into things, most negative for being yelled at or for running down the battery, etc.

- Video Game Playing: One unit of reward for every point scored.

Another reason one might object to the problem formulation in Figure 1 is that the goal of learning is defined solely in terms of something (the reward) arising from the external environment, not from the learning agent itself. Often goals do concern the evolution of the the learning agent's internal state, e.g., its energy reservoirs. Fortunately, it appears that most goals, perhaps all, can be put in the "external reward" form simply by redrawing the boundary between agent and environment. For example, if the goal concerns the agent's energy reservoirs, then these are considered part of the environment; if the goal concerns the positions of the agent's limbs, then these too can be considered part of the environment—the learning agent's boundary is drawn at the interface between the limbs and their control systems. Roughly speaking, things are considered part of the learning agent if they are completely, directly, and with certainty controllable; things are considered part of the environment if they are not. Since the goal is always something over which we have imperfect or uncertain control, it is placed outside the learning agent.

2 Overview of Reinforcement Learning Architectures

In this section we review the major steps in the development of reinforcement learning architectures over the last decade. These steps are illustrated by the four architectures shown in Figures 2-5.

All reinforcement learning involves the learning of a mapping from a representation of a situation or *state* to an appropriate *action* (or a probability distribution over actions) for that situation. This mapping is called the *policy*; it specifies what the agent will do in each situation at its current stage of learning. The simplest reinforcement learner that one might imagine, then, would consist only of a policy and a way of adjusting it based on reward, as shown in Figure 2. Such architectures, in which the policy is the only modifiable data structure (and, indeed, the only structure at all) are here called *policy-only* architectures.

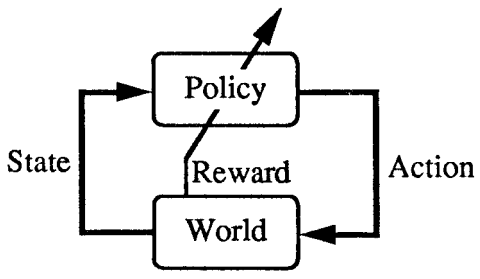

Figure 2. The Policy-Only Architecture.

The policy can be implemented in any of a number of ways. It can be a connectionist neural network, or a symbolic learning structure such as a decision tree or lisp program, or a conventional set of statistics such as is used by maximum-likelihood or nearest-neighbor techniques. Any of these methods can be used—with varying advantages and disadvantages—to implement this and the other modifiable structures shown in Figures 2-5.

The learning algorithm for the policy must be of a slightly unusual type. Standard supervised learning methods such as backpropagation are not sufficient here, but must be modified, at least slightly, to take into account the fact that a "target" action is not directly available. Instead, a form of correlation must be done between the reward received and the actions taken by the agent, all with respect to the sensory input. Actions correlated with high reward have their probability of being repeated increased, while those correlated with low reward have their probability decreased. Examples of such algorithms for policy-only architectures can be found in (Farley & Clark, 1954; Widrow, Gupta & Maitra, 1973; Barto & Sutton, 1981; Barto & Anandan, 1985).

Policy-only architectures really only work well when it is clear a priori what constitutes a high reward and what constitutes a low one—for example, if all high rewards are positive and all low rewards are negative. Often, however, rewards are not distributed around a baseline of zero, but around some other, unknown value. Worse yet, the baseline may change from state to state. A low reward value in one state may be the highest attainable in another. To handle such variations, a baseline value must be learned that is a function of the state; the actual reward is then compared with the current state's baseline. This is what is done in *reinforcement-comparison* architectures (Figure 3). As a baseline, these architectures usually use a *prediction* of the reward. The prediction error—the difference between predicted and actual reward—is used both as an enhanced, zero-balanced reward sig-

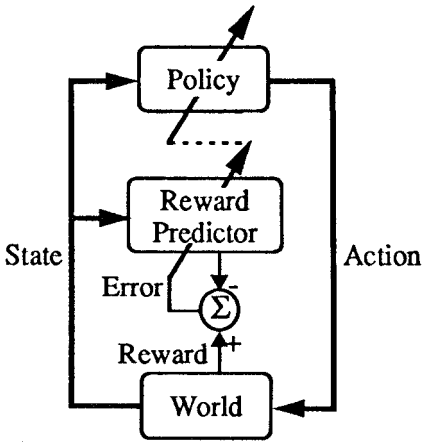

Figure 3. Reinforcement-Comparison Architecture.

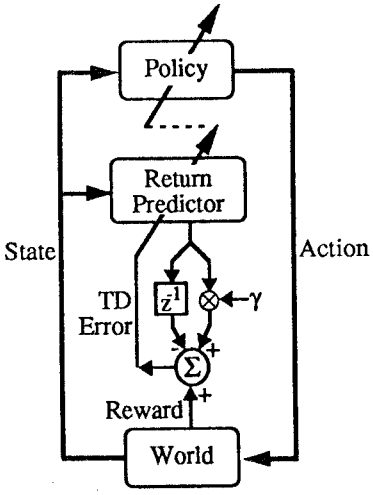

Figure 4. Adaptive-Heuristic-Critic Architecture. The symbol z^{-1} indicates a one-time-step delay, and the symbol $\otimes$ indicates multiplication.

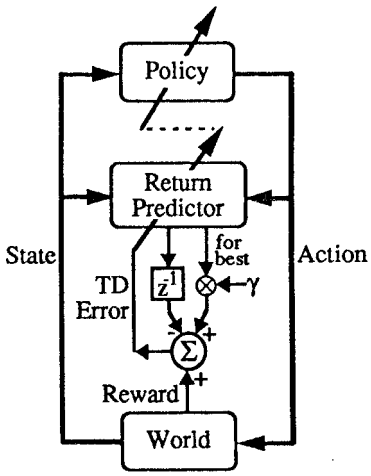

Figure 5. Q-learning Architecture. The line labeled "for best" is the prediction of return for the best action; the other output from the return predictor is the prediction of return for the action actually taken.

nal for adjusting the policy *and* as an ordinary error for learning the reward predictions. A variety of reinforcement-comparison algorithms have been explored and compared (Barto, Sutton & Brouwer, 1981; Sutton, 1984; Williams, 1992).

Reinforcement comparison architectures are effective at optimizing immediate rewards, but not at optimizing total reward in the long run. The problem is that actions have two kinds of consequences—they affect the next reward *and* they affect the next state, but reinforcement-comparison architectures only take the first of these into account. Suppose an action produces high immediate reward but deposits the environment in a state from which only low reward can be obtained? In order to optimize long-term reward, these delayed affects of action must be taken into account.

The *adaptive heuristic critic (AHC)* architecture, shown in Figure 4, was designed to take such delayed effects into account. The predictor of immediate reward has been replaced with a predictor of *return*, a measure of long-term cumulative reward. For any state x, the return is formally defined as the expected value of the sum of all future rewards, discounted by their delay, given that the systen starts in x:

$$return(x) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| x_0 = x\right\},$$

where γ, $0 \leq \gamma \leq 1$, is the discount rate determining how fast one's concern for delayed reward falls off with length of the delay.[1] The "return predictor" box in Figure 4 comes to predict this return by virtue of the circuit shown below it for calculating a temporal-difference error (Sutton, 1988; Tesauro, 1992). In all other respects, the learning algorithm inside this box could be exactly the same as that used in the "reward predictor" box of Figure 3. The AHC architecture has been used in a variety of learning control tasks (Sutton, 1984; Barto, Sutton & Anderson, 1983; Anderson, 1987; Barto, Sutton & Watkins, 1989).

Finally, Figure 5 shows the most recent reinforcement learning architecture, *Q-learning* (Watkins, 1989; Watkins & Dayan, 1992). The primary innovation here is that the predicted return is now a function of action as well as state. Formally, the return for a state x and action a is defined as

$$return(x,a) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| x_0 = x, a_0 = a\right\}. \quad (1)$$

Two kinds of return are always predicted for the current state. One is the best predicted return for the state—the predicted return for the action with the highest predicted return. The other is the predicted return for the action actually selected. These two predictions are then combined according to the same circuit as used in the AHC architecture. In Q-learning, the policy could be a separate modifiable data structure as suggested by Figure 5, but most often it is simply a dependent function of the return predictions. For example, the policy may be simply to pick the action that obtains the maximal return prediction.

3 Dyna Architectures

Reinforcement learning architectures are effective at trial-and-error learning, but no more. They can not do any of the things that are considered "cognitive," such as reasoning or planning. They do not learn an internal model of the world's dynamics, of what-causes-what, but only of what-to-do (policy) and

[1] This is analogous to the discount rate in economics—a dollar today is worth more than a dollar tomorrow.

1. Observe the current state x and choose an action: $a \leftarrow Policy(x)$.
2. Send the action to the world and observe the resultant next state y and reward r.
3. Apply a reinforcement learning method to the experience x, y, a, and r.
4. Update the world model based on the experience x, y, a, and r.
5. Repeat the following steps k times:
   5.1 Select a hypothetical state x and hypothetical action a.
   5.2 Send x and a to the world model and obtain predictions of next state y and reward r.
   5.3 Apply a reinforcement learning method to the hypothetical experience x, y, a, and r.
6. Go to 1.

Figure 6. Main Loop of the Dyna Algorithm.

how-good-is-it (return predictions). This is an important limitation because potentially much more can be learned in the form of a world model than can be learned by trial and error; the reward signal is just a scalar, while the sensory input signal is a much richer potential source of training information. And what if the goal changes? Typically, a world model can remain relatively intact over goal changes and can assist in achieving the new goal, whereas policy and return predictions must be totally changed.

Dyna architectures are simple extensions of reinforcement learning architectures to include an internal world model (Sutton, 1990; Whitehead, 1989; see also Werbos, 1987). The world model is defined as something that behaves like the world: given a state and an action it is supposed to output a prediction of the resultant reward and next state. If the world's state is observable, then it is straightforward to learn a world model using supervised learning methods and training examples taken from actual interactions with the world. If the world's state is not observable, then it must be inferred from the history of sensory input and action. Although there are a variety of algorithms for doing this, in the general case it is still an open problem (see Whitehead and Ballard, 1991; Chrisman, 1992). Here we assume that the state is observable.

In Dyna architectures the world model is used as a direct replacement for the world in one of the reinforcement learning architectures shown in Figures 2-5. Reinforcement learning continues in the usual way, but, in addition, learning steps are also run using the model in place of the world, using predicted outcomes rather than actual ones. For each real experience with the world, many hypothetical experiences generated by the world model can be processed and learned from as well. The cumulative effect of these hypothetical experiences is that the policy approaches the optimal policy given the current model; a form of planning has been achieved. The overall Dyna algorithm is given in Figure 6.

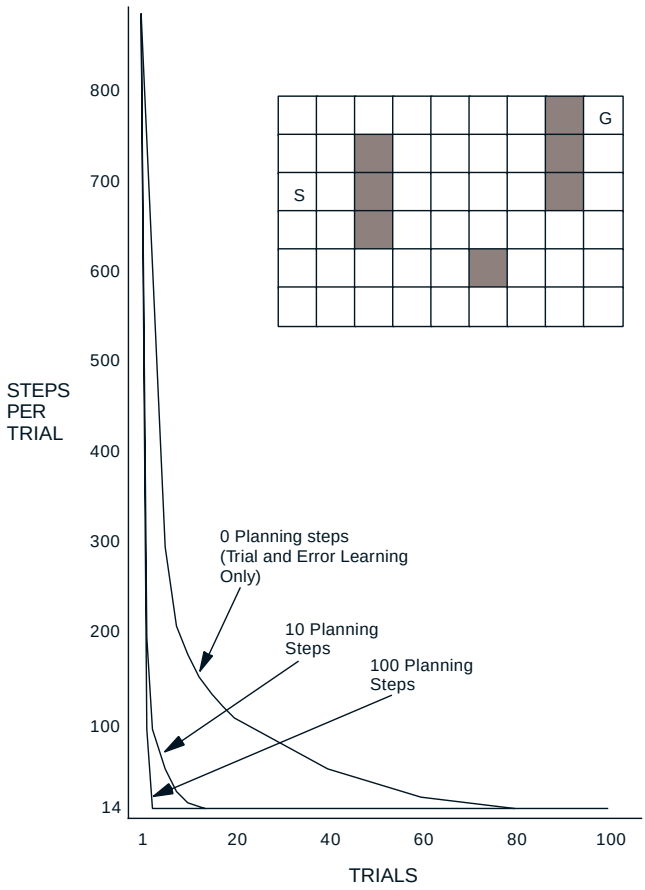

Figure 7. Learning Curves for Dyna-AHC Systems on a Simple Navigation Task. A trial is one trip from the start state "S" to the goal state "G". The more hypothetical experiences ("planning steps") using the world model, the faster an optimal path was found. These data are averages over 100 runs.

Figure 7 shows results for a Dyna architecture based on the AHC architecture, called *Dyna-AHC*. The task is to navigate through the maze shown in Figure 7 from the starting state "S" to the goal state "G". From each state there are four possible actions: UP, DOWN, RIGHT, and LEFT, which change the state accordingly, except where such a movement would take the system into a barrier (shaded state) or outside the maze, in which case the state is not changed. Reward is zero for all transitions except for

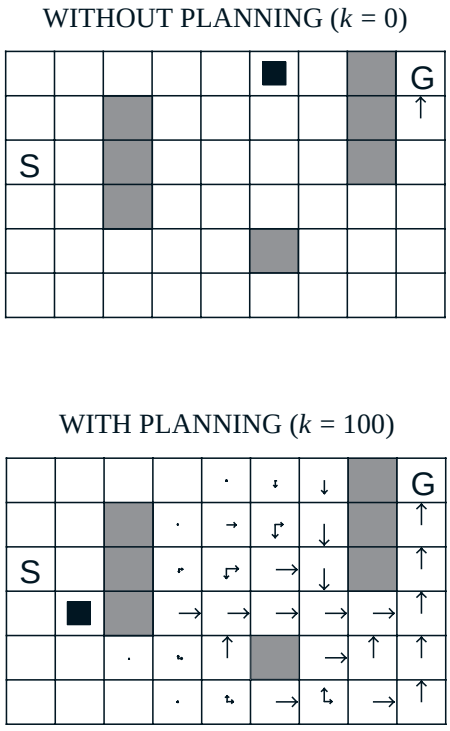

Figure 8. Policies Found by Planning and Non-Planning Dyna-AHC Systems by the Middle of the Second Trial. The black square indicates the current location of the Dyna system. The arrows indicate action probabilities (excess over the smallest) for each direction of movement.

those into the goal state, for which it is +1. The lower left portion of the figure shows learning curves for Dyna-AHC systems with $k = 100$, $k = 10$, and $k = 0$. The $k = 0$ case involves no hypothetical steps; this is a pure trial-and-error reinforcement-learning system. Although the length of path taken from start to goal falls dramatically for this case, it falls much *more* rapidly for the cases including hypothetical experiences, showing the benefit of planning using the learned world model. For $k = 100$, the optimal path was generally found and followed by the fourth trip from start to goal. This is extremely rapid learning.

Figure 8 shows why the Dyna-AHC system solved this problem so much faster than the pure reinforcement learning system. Shown are the policies found by the $k = 0$ and $k = 100$ systems half-way through the second trial. Without planning ($k = 0$), each trial adds only one additional step to the policy, and so only one step (the last) has been learned so far. With planning, the first trial also learned only one step, but here during the second trial an extensive policy has been developed that by the trial's end will reach back almost to the start state. By the end of the third or fourth trial a complete optimal policy will have been found and perfect performance attained.

4 Limitations and Conclusions

The simple illustrations presented here are clearly limited in many ways. The state and action spaces are small and denumerable, permitting tables to be used for all learning processes and making it feasible for the entire state space to be explicitly explored. For large state spaces it is not practical to use tables or to visit all states; instead one must represent a limited amount of experience compactly and generalize from it. Both Dyna architectures are fully compatible with the use of a wide range of learning methods for doing this. For example, Lin (1992) has explored the use of Dyna architectures using backpropagation networks instead of tables, and Tesauro has obtained excellent results in computer backgammon by combining backpropagation with temporal-difference learning.

Another limitation of the Dyna systems presented here is the trivial form of search control used. Search control in Dyna boils down to the decision of whether to consider hypothetical or real experiences, and of picking the order in which to consider hypothetical experiences. The tasks considered here are so small that search control is unimportant, and thus it was done trivially, but a wide variety of more sophisticated methods could be used (e.g., see Peng & Williams, 1992).

Despite these limitations, the results presented here are significant. They show that the use of an internal model can dramatically speed trial-and-error learning processes even on simple problems. Moreover, they show how planning can be done with world models contructed through learning (see also Sutton, 1990). Finally, they show how the functionality of planning can be obtained in a completely incremental manner, and how a planning process can be freely intermixed with execution and learning. I conclude that it is not necessary to choose between planning, reacting, and learning. These three can be integrated not only into one learning agent, but into a single algorithm, where each appears as a different facet of that algorithm.

Acknowledgments

The author gratefully acknowledges the extensive contributions to these ideas by Andrew Barto, Chris Watkins, Ron Williams, and Steve Whitehead. I also wish to also thank the following people for ideas and discussions: Paul Werbos, Luis Almeida, Glenn Iba, Leslie Kaelbling, John Vittal, Charles Anderson, Bernard Silver, Oliver Selfridge, Judy Franklin, Tom Dean and Chris Matheus.

References

Anderson, C. W. (1987) Strategy learning with multilayer connectionist representations. *Proceedings of the Fourth International Workshop on Machine Learning,* 103–114. Morgan Kaufmann, Irvine, CA.

Barto, A. G., & Anandan, P. (1985) Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics 15*, 360–375.

Barto, A. G., & Sutton. R. S. (1981) Landmark learning: An illustration of associative search. *Biological Cybernetics 42*, 1–8.

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

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

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1989) Learning and sequential decision making. In *Learning and Computational Neuroscience,* M. Gabriel and J.W. Moore (Eds.), 539–602, MIT Press, 1991.

Chrisman, L. (1992) Reinforcement learning with perceptual aliasing: The predictive distinctions approach. AAAI-92.

Farley, B. G., & Clark, W. A. (1954) Simulation of self-organizing systems by digital computer. *I.R.E. Transactions on Inf. Theory 4*, 76–84.

Lin, L. (1992) Self-improving reactive agents based on reinforcement learning, planning, and teaching. it Machine Learning 8, 293–322.

Minsky, M.L. (1961) Steps toward artificial intelligence. *Proceedings I.R.E., 49*, 8-30. Reprinted in E.A. Fiegenbaum & J. Feldman (Eds.), *Computers and Thought*, 406–450, New York: McGraw-Hill.

Peng, J. & Williams, R.J. (1992) Efficient search control in Dyna. Northeastern University Technical Report.

Sutton, R. S. (1984) Temporal credit assignment in reinforcement learning. Doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

Sutton, R.S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning 3*, 9–44.

Sutton, R. S. (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, 216–224, Morgan-Kaufmann.

Tesauro, G. (1992) Practical issues in temporal-difference learning. *Machine Learning 8*, 257–278.

Waltz, M.D. & Fu, K.S. (1965) A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control, AC-10*, 390–398.

Watkins, C. J. C. H. (1989) *Learning with Delayed Rewards*. Ph.D. dissertation, Cambridge University, Psychology Department.

Watkins, C. J. C. H. & Dayan, P. (1992) Q-learning. *Machine Learning 8*, 279–292.

Werbos, P. J. (1987) Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17, No. 1, 7–20.

Whitehead, S. D. (1989) Scaling reinforcement learning systems. Technical Report 304, Dept. of Computer Science, University of Rochester, Rochester, NY 14627.

Widrow, B., Gupta, N. K., & Maitra, S. (1973) Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics 5*, 455–465.

Williams, R. J. (1986) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8*, 229–256.