Approximating a Policy Can be Easier Than Approximating a Value Function

Charles W. Anderson
www.cs.colostate.edu/˜anderson

February 10, 2000

Technical Report CS-00-101

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

Approximating a Policy Can be Easier Than Approximating a Value Function

Charles W. Anderson
www.cs.colostate.edu/~anderson

February 10, 2000

Abstract

Value functions can speed the learning of a solution to Markov Decision Problems by providing a prediction of reinforcement against which received reinforcement is compared. Once the learned values relatively reflect the optimal ordering of actions, further learning is not necessary. In fact, further learning can lead to the disruption of the optimal policy if the value function is implemented with a function approximator of limited complexity. This is illustrated here by comparing Q-learning (Watkins, 1989) and a policy-only algorithm (Baxter & Bartlett, 1999), both using a simple neural network as the function approximator. A Markov Decision Problem is shown for which Q-learning oscillates between the optimal policy and a sub-optimal one, while the direct-policy algorithm converges on the optimal policy.

1. Introduction

Reinforcement learning algorithms for solving Markov Decision Problems (MDPs) can be distinguished by whether they directly learn a policy function or a state-action value function from which a policy is determined. Examples of direct-policy algorithms are the REINFORCE algorithm (Williams, 1987) and the recent direct-gradient algorithm (Baxter & Bartlett, 1999). The Q-learning (Watkins, 1989) algorithms are well-known examples of state-action value function algorithms. Other approaches, such as the actor-critic architecture (Barto et al., 1983), develop both a value function and a policy function.

Preference for one algorithm over others can be based on a number of algorithm characteristics. One might prefer algorithms for which proofs of optimality or convergence are known. Proofs of optimality of Q-learning drove many practitioners to apply Q-learning to their problems during the 1990's. Assumptions about the state representation for an MDP limited the applicability of these proofs to real world problems. Function approximators, such as neural networks, are typically used to represent the Q function, violating the optimality proofs and requiring experimentation with function approximator parameters to solve a given MDP. However, proofs of convergence for more general function approximators have recently appeared in the work of Baxter and Bartlett (1999), who prove convergence of their policy-only method, and Sutton et al. (2000), who developed a proof of convergence for a form of the actor-critic architecture.

Another way to develop a preference of one algorithm over others for a given problem is to consider whether the policy function or the value function is easiest to represent for a chosen class of function approximators. Assuming that the function that is easiest to represent is also easiest to learn, a problem for which the policy is easier to represent should be attacked with a policy-only algorithm. If the value function is most easily represented, then one should use a value-only algorithm, or a value and policy algorithm.

This is, of course, a much too simplified perspective. Many factors other than the complexity of the value or policy function affect the efficiency of learning. For example, comparing immediate reinforcement to a baseline, such as that provided by temporal-difference algorithms for learning value functions, can lead to faster learning.

Nonetheless, this article illustrates the advantage that a policy-only algorithm has over a value function algorithm when the same function approximator of limited complexity is used for both. The direct-gradient (Baxter & Bartlett, 1999) and one-step Q-learning (Watkins, 1989) algorithms are implemented using a neural network with a single hidden unit for the function approximator. They are compared on an MDP for which the neural network can represent the optimal policy, but cannot represent the state-value func-

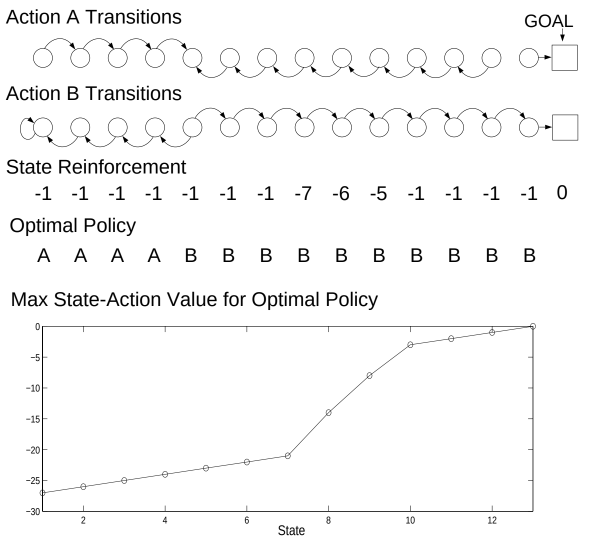

Figure 1. Task

tion for the optimal policy. Results show that Q-learning oscillates between the optimal policy and a sub-optimal policy, while the direct-gradient method converges on the optimal policy.

2. The Problem

The Markov Decision Problem consists of 13 states, indexed from 1 to 13, with State 1 being the initial state. Two actions, Actions A and B, are available in each state. Immediate reinforcement ranges between 0 and -7. The problem is diagrammed in Figure 1, which also shows the optimal policy and the maximum of the Q values for each state, given that the optimal policy is followed.

A neural network is used as the function approximator. State is input to the neural network as a single integer between 1 and 13. A neural network with a single, sigmoid, hidden unit and a single linear output unit can represent the optimal policy, as shown below in the results section. However, the problem is designed so that the optimal Q function cannot be represented by the network. An informal argument for this is as follows. One hidden unit is needed to switch the relative values of the Q function between States 4 and 5, so that Action A is preferred for States 1–4 and Action B is preferred for States 5–13. However, at least one additional hidden unit is required to accurately represent the variations in the Q function for States 7–13.

3. Policy-Only Algorithm

The implementation of the direct-gradient algorithm of Baxter and Bartlett (1999) is specified below. The specification is specialized to the particular 13-state MDP and neural network used in the experiments described here.

Let s_k be the state, a_k the action, and r_k the reinforcement at time step k. The network's hidden layer and output layer weights are give by w_k^h and w_k^o, respectively. The input to the network is y_k^i, the output of the hidden layer is y_k^h, and the output of the final layer is y_k^o. Rows of the weight matrices contain the weights for individual units, and the y vectors are column vectors. z_k^h and z_k^o represent the estimated gradients of the reinforcement sum with respect to the network's hidden and output layer weights, and are the same shape as the weight matrices. The function f is the usual sigmoid function

$$f(x) = \frac{1}{1+e^{-x}}.$$

Also, x' represents the vector x augmented by concatenating a constant 1. The isolation of the first column of a matrix x is denoted by $[x]_1$.

The algorithm is initialized by

$$\begin{aligned} w_0^h &= \text{uniform random numbers in } [-0.1, 0.1] \\ w_0^o &= 0 \end{aligned}$$

The following steps are repeated for each of the 100,000 trials:

$$\begin{aligned} z_k^h &= 0, \\ z_k^o &= 0, \\ s_k &= 1. \end{aligned}$$

Then, for each trial, repeat these steps until $s_k = 13$ or 100 steps have taken place:

$$\begin{aligned} y_k^i &= s_k - 6 \\ y_k^h &= f(w_k^h y_k^{i'}) \\ y_k^o &= w_k^o y_k^{h'} \\ a_k &= \begin{cases} A, & \text{with prob } f(y_k^o) \\ B, & \text{otherwise,} \end{cases} \\ \Delta s_k &= \begin{cases} 1, & s_k \le 4, a_k = A; \\ 1, & s_k \ge 5, a_k = B; \\ -1, & s_k \le 4, a_k = B; \\ -1, & s_k \ge 5, a_k = A; \end{cases} \\ s_{k+1} &= \begin{cases} 13, & s_k + \Delta s_k > 13; \\ 1, & s_k + \Delta s_k < 1; \\ s_k + \Delta s_k, & \text{otherwise.} \end{cases} \end{aligned}$$

$$
\begin{aligned}
r_{k+1} &= \begin{cases} -7, & s_{k+1} = 8; \\ -6, & s_{k+1} = 9; \\ -5, & s_{k+1} = 10; \\ -1, & \text{otherwise.} \end{cases} \\
\delta_k &= \begin{cases} 1 - y_k^o, & a_k = A; \\ y_k^o, & a_k = B. \end{cases} \\
z_{k+1}^o &= \beta z_k^o + \delta_k y_k^{h'} \\
z_{k+1}^h &= \beta z_k^h + y_k^h \cdot (1 - y_k^h)[w_k^o]_1 \delta_k y_k^{i'} \\
w_{k+1}^h &= w_k^h + \rho^h r_{k+1} z_{k+1}^h \\
w_{k+1}^o &= w_k^o + \rho^o r_{k+1} z_{k+1}^o
\end{aligned}
$$

4. Q-Learning Algorithm

Watkin's (1989) one-step, Q-learning algorithm is defined similarly. The neural network has a single hidden unit as before, but now there are two output units corresponding to the two actions, A and B.

The algorithm is initialized by

$$
\begin{aligned}
w_0^h &= \text{uniform random numbers in } [-0.1, 0.1] \\
w_0^o &= 0
\end{aligned}
$$

The following steps are repeated for each of the 100,000 trials. Set

$$s_k = 1$$

and these steps are repeated until $s_k = 13$ or 100 steps have taken place:

$$
\begin{aligned}
y_k^i &= s_k - 6 \\
y_k^h &= f(w_k^h y_k^{i'}) \\
y_k^o &= w_k^o y_k^{h'} \\
a_k &= \begin{cases} A, & \text{with prob } \frac{e^{\beta [y_k^o]_1}}{e^{\beta [y_k^o]_1} + e^{\beta [y_k^o]_2}} \\ B, & \text{otherwise,} \end{cases} \\
\Delta s_k &= \begin{cases} 1, & s_k \le 4, a_k = A; \\ 1, & s_k \ge 5, a_k = B; \\ -1, & s_k \le 4, a_k = B; \\ -1, & s_k \ge 5, a_k = A; \end{cases} \\
s_{k+1} &= \begin{cases} 13, & s_k + \Delta s_k > 13; \\ 1, & s_k + \Delta s_k < 1; \\ s_k + \Delta s_k, & \text{otherwise.} \end{cases} \\
r_{k+1} &= \begin{cases} -7, & s_{k+1} = 8; \\ -6, & s_{k+1} = 9; \\ -5, & s_{k+1} = 10; \\ -1, & \text{otherwise.} \end{cases} \\
x_{k+1} &= w_k^o f'(w_k^h (s_{k+1} - 6)')
\end{aligned}
$$

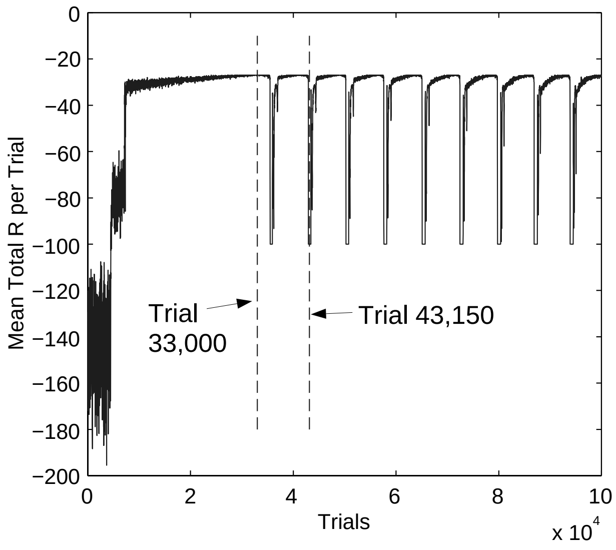

Figure 2. Q-learning: Sum of reinforcement per trial, averaged over bins of 10 trials.

$$
\begin{aligned}
\eta &= \begin{cases} \begin{bmatrix} 1 \\ 0 \end{bmatrix}, a_k = A; \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix}, a_k = B; \end{cases} \\
\delta_k^o &= (r_{k+1} + \max_j([x_{k+1}]_j) - y_k^o)\eta \\
\delta_k^h &= y_k^h \cdot (1 - y_k^h)[w_k^o]_1 \delta_k^o \\
w_{k+1}^h &= w_k^h + \rho^h \delta_k^h y_k^{i'} \\
w_{k+1}^o &= w_k^o + \rho^o \delta_k^o y_k^{h'}
\end{aligned}
$$

5. Results

Parameter values were chosen for each algorithm to obtain the best performance for each. For the Q-learning algorithm, $\rho^h = 0.0001$, $\rho^o = 0.0001$, and $\beta = 10$. For the policy-only algorithm, $\rho^h = 0.01$, $\rho^o = 0.0001$, and $\beta = 0.999$.

Figure 2 shows how the sum of reinforcement per trial increases from values from -120 to -180 during the initial trials to close to the optimal value of -28 at Trial 33,000, marked by the first vertical dashed line. Then, performance periodically degrades to close to -100, the reinforcement sum that results when the state never progresses beyond State 5; recall that a trial terminates after 100 steps and all states less than State 5 result in -1 reinforcement. Between brief periods of -100 performance, near optimal performance is regained. Trial 43,150 is also marked as a dashed line. Both trials are examined below.

To understand this behavior, the policies, outputs of

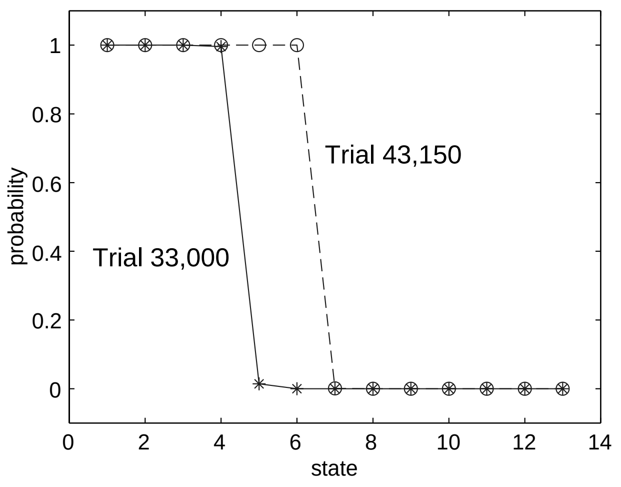

Figure 3. Q-learning: Probability of Action A.

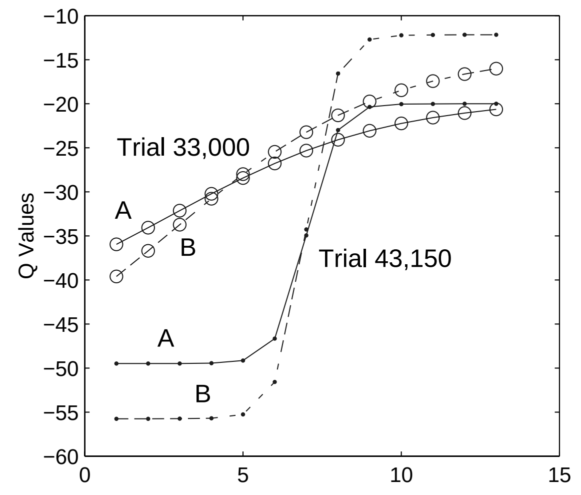

Figure 4. Q-learning: Outputs of the two output units.

the neural network, and the weights of the hidden unit are examined. First, let's look at the policies that result at Trials 33,000 and 43,150. The policies are shown in Figure 3 as the probability of Action A in each state. At Trial 33,000 the policy is extremely close to optimal, but by Trial 43,150 the action chosen for States 5 and 6 have switched.

The switch in chosen action is due to a change in the relative ordering of Q values for those states. This can be seen in Figure 4, a plot of the Q values for the two actions These two sets of Q values suggest the cause for the oscillation. At Trial 33,000, the relative placement of the two Q curves does result in the optimal policy, but the steep reduction in the predicted reinforcement for States 8, 9, and 10 is poorly-approximated.

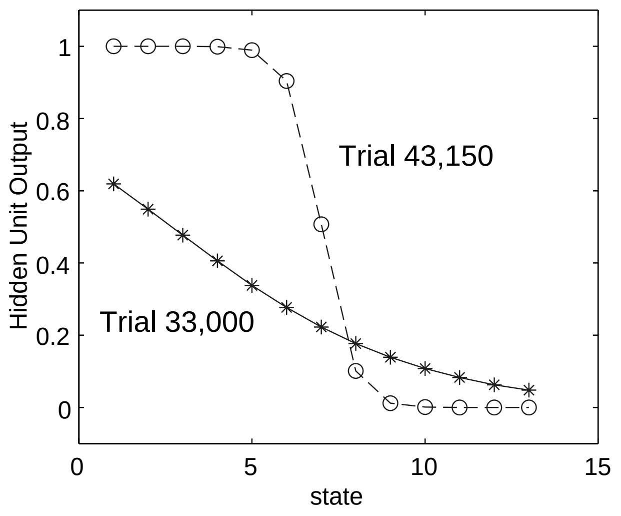

Figure 5. Q-learning: Hidden unit outputs.

By Trial 43,150, the steep reduction for these states is well-approximated, but this has shifted the state at which the chosen action switches and the optimal policy has been lost. Further learning results in the oscillation between these two configurations.

A similar oscillation is seen in the output of the hidden unit, shown in Figure 5. The oscillation is explicit in a plot of the evolution of two weight values in the hidden unit while learning. Figure 6 shows this trajectory. Clearly the weight vector trajectory is periodic.

Now let's examine the behavior of the policy-only, direct-gradient algorithm. The plot in Figure 7 of the sum of reinforcement per trial shows that the algorithm relatively quickly converges close to the optimal sum of -28 and this performance is stable.

The output of the single output unit is shown in Figure 8 and the resulting policy is shown in Figure 9. Both are for the final trial, Trial 100,000. The policy is clearly very near optimal, with probabilities close to 0 and 1. The output of the hidden unit is shown in Figure 10. The evolution of the hidden unit's weight vector is shown in Figure 10.

To show that the Q-learning algorithm is capable of solving this problem with an adequate function approximator, training was repeated with a neural network consisting of two hidden units. Figure 12 shows that the problem is solved—the reinforcement sum is close to the optimal value of -28. The resulting Q values for Trial 100,000 are shown in Figure 13, and the policy is shown in Figure 14. The policy is very close to optimal for all states but State 4, which should be 1 rather than 0.6. Further training does not increase this probability, though additional hidden units proba-

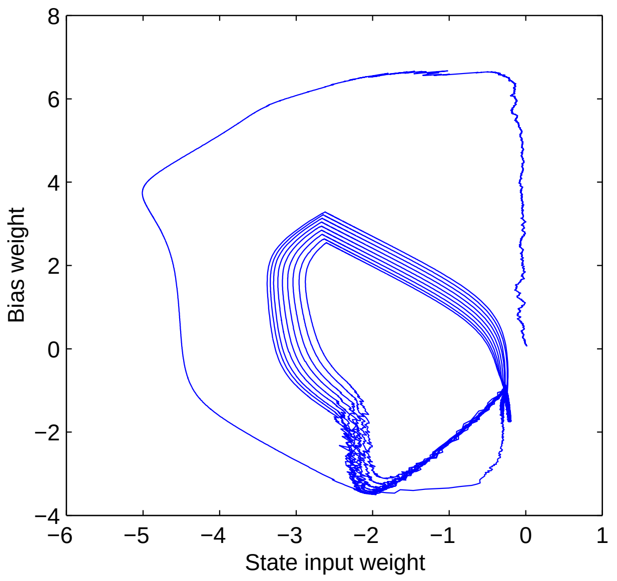

Figure 6. Q-learning: Evolution of hidden unit weight vector.

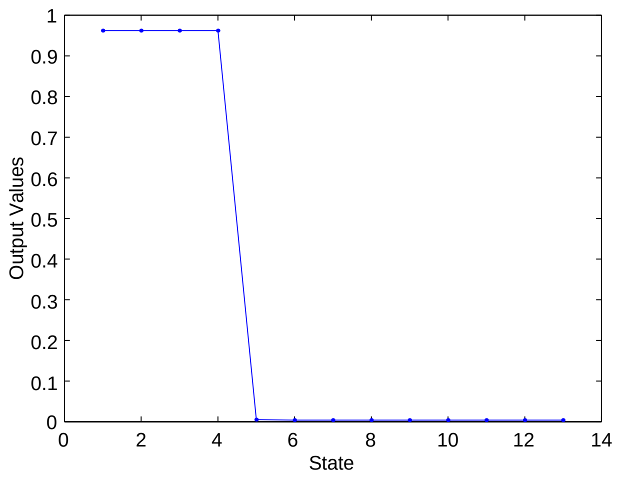

Figure 8. Policy-only: Output of output unit at Trial 100,000.

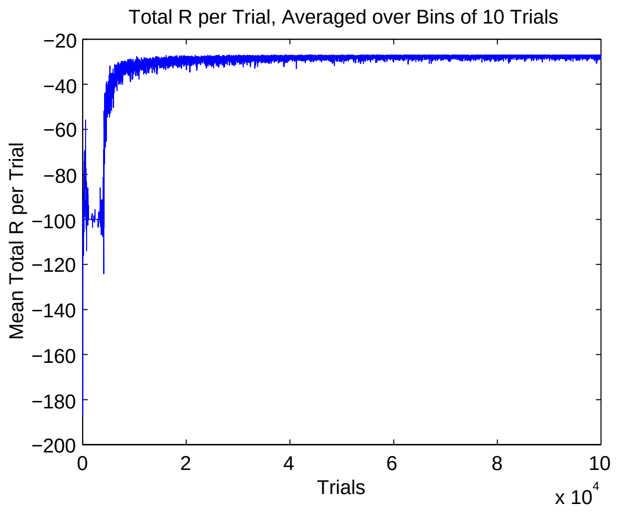

Figure 7. Policy-only: Sum of reinforcement per trial averaged in bins of 10 trials.

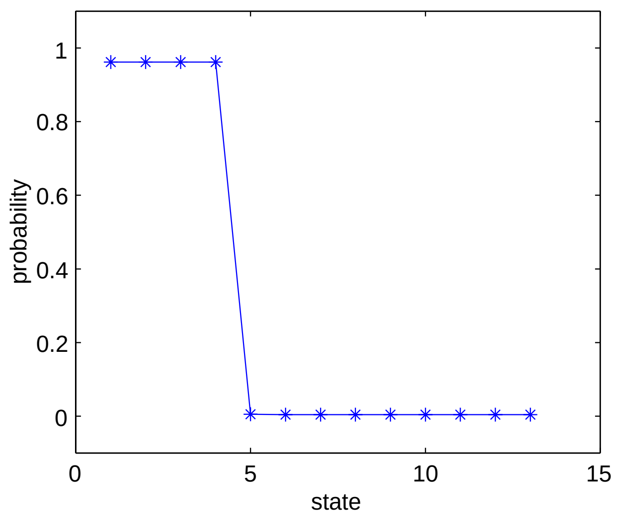

Figure 9. Policy-only: Probability of Action A at Trial 100,000.

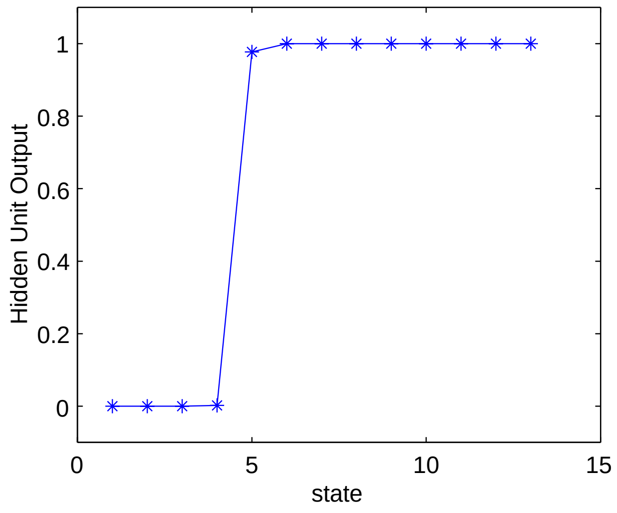

Figure 10. Policy-only: Hidden unit output at Trial 100,000.

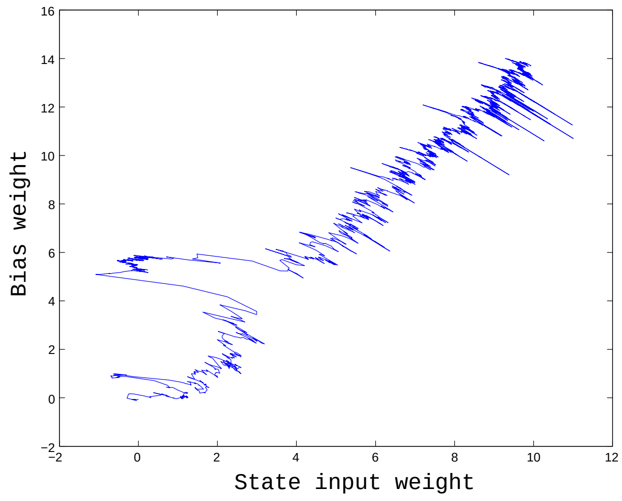

Figure 11. Policy-only: Evolution of hidden unit weight vector.

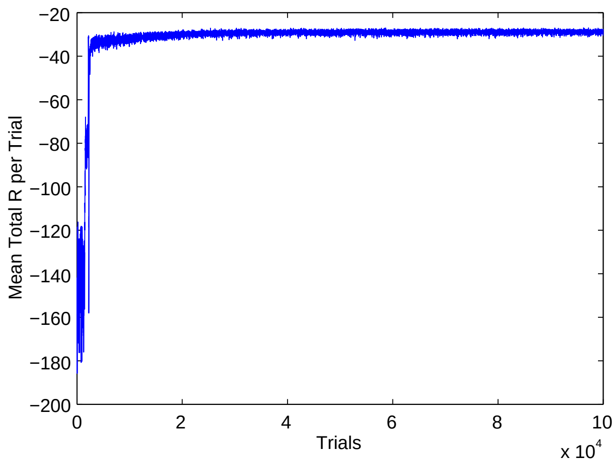

Figure 12. Q-learning with two hidden units: Sum of reinforcement per trial averaged in bins of 10 trials.

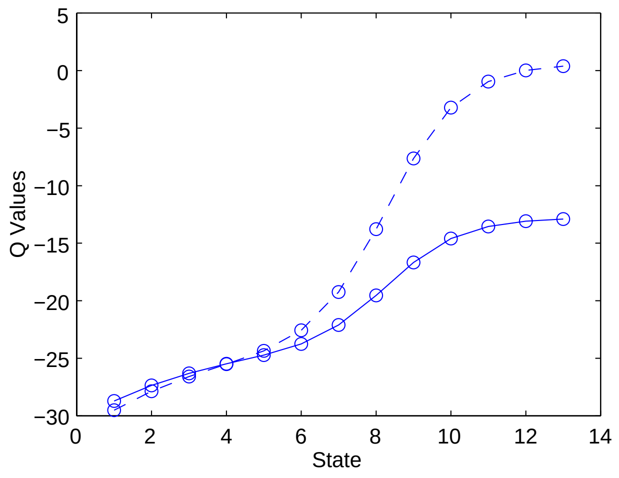

Figure 13. Q-learning with two hidden units: Outputs of output units at Trial 100,000

bly would. Figure 15 shows that the two hidden units learned two functions that are very similar to the two configurations that the single-hidden-unit network oscillated between.

6. Conclusions

This experiment is only an illustration of a case where directly learning a policy has an advantage over trying to accurately learn a state-action value function. The advantage arises by limiting the complexity of the function approximator so that it is not capable of accurately approximating the value function but is capable of approximating the optimal policy.

The results shown here do not support any general conclusions about the relative merits of policy-only versus

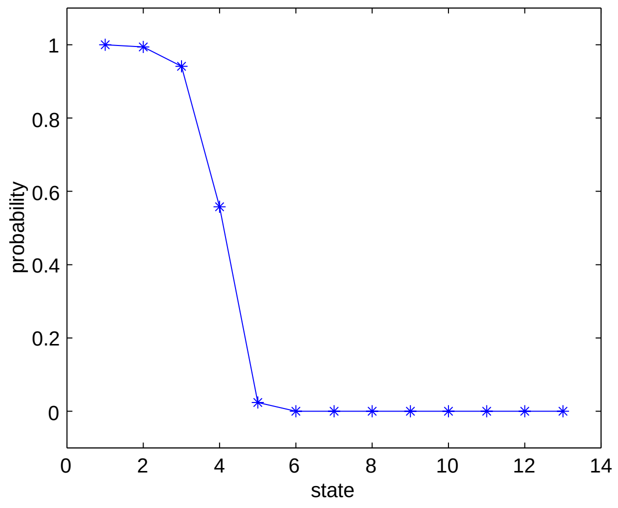

Figure 14. Q-learning with two hidden units: Probability of Action A at Trial 100,000.

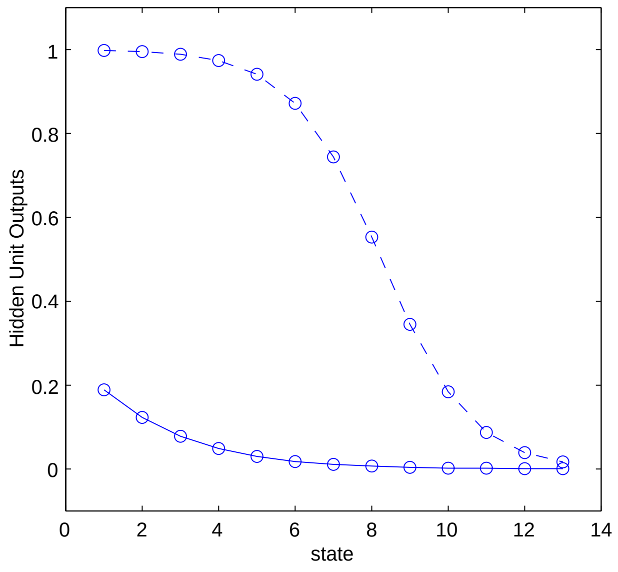

Figure 15. Q-learning with two hidden units: Outputs of the two hidden units at Trial 100,000.

value function algorithms. However, this illustration does suggest that it might be fruitful to examine the relative complexity of policies versus value functions in larger, realistic problems. An optimal value function for a given problem is probably at least as complex as the optimal policy function. The value function not only must represent all of the boundaries in state space which separate regions of different optimal actions; it must also capture the possibly many variations in values within the regions. Since the latter variations have no effect on the resulting policy, dedicating the resources of a function approximator to representing the variations is misdirected and possibly disastrous to the maintenance of a good policy, as shown by the results of the illustration.

Acknowledgements

Discussions with Peter Bartlett and Jonathan Baxter about their direct-gradient algorithm motivated the experiments reported here. Thanks also to the Department of Systems Engineering, Research School of Information Sciences and Engineering, The Australian National University, in Canberra, Australia, for providing computing resources during the author's sabbatical visit.

References

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*, 835–846. Reprinted in J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, 1988.

Baxter, J., & Bartlett, P. (1999). *Direct gradient-based reinforcement learning: I. gradient estimation algorithms* (Technical Report). Computer Sciences Laboratory, Australian National University.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). *Policy gradient methods for reinforcement learning with function approximation* (Technical Report). AT& T Labs.

Watkins, C. (1989). *Learning with delayed rewards.* Doctoral dissertation, Cambridge University Psychology Department, Cambridge, England.

Williams, R. J. (1987). *Reinforcement-learning connectionist systems* (Technical Report NU-CCS-87-3). College of Computer Science, Northeastern University, Boston, MA.