As appeared in the Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems, pp. 91-96. Center for Systems Science, Yale University, 1994.

On Step-Size and Bias in Temporal-Difference Learning

Richard S. Sutton
sutton@gte.com

Satinder P. Singh
Brain & Cognitive Sciences Dept. (E10)
Massachusetts Institute of Technology
Cambridge, MA 02139
singh@psyche.mit.edu

Abstract

We present results for three new algorithms for setting the step-size parameters, α and λ, of temporal-difference learning methods such as TD(λ). The overall task is that of learning to predict the outcome of an unknown Markov chain based on repeated observations of its state trajectories. The new algorithms select step-size parameters online in such a way as to eliminate the bias normally inherent in temporal-difference methods. We compare our algorithms with conventional Monte Carlo methods. Monte Carlo methods have a natural way of setting the step size: for each state s they use a step size of $1/n_s$, where n_s is the number of times state s has been visited. We seek and come close to achieving comparable step-size algorithms for TD(λ). One new algorithm uses a $\lambda = 1/n_s$ schedule to achieve the same effect as processing a state backwards with TD(0), but remains completely incremental. Another algorithm uses a λ at each time equal to the estimated transition probability of the current transition. We present empirical results showing improvement in convergence rate over Monte Carlo methods and conventional TD(λ). A limitation of our results at present is that they apply only to tasks whose state trajectories do not contain cycles.

1 Introduction

The defining feature of temporal-difference (TD) learning (Sutton, 1988) is that it involves learning predictions on the basis of other predictions. Suppose you are using observations of an absorbing Markov process to predict where it will absorb as a function of its current state. The process is currently in state s_t and you wish to update your prediction for s_t based on the current trajectory. A conventional Monte Carlo method would wait to see where the process absorbs and then shift the prediction for s_t toward that outcome. A temporal-difference method, on the other hand, would wait just one step, until s_{t+1} was known, and then shift the prediction for s_t toward the prediction for s_{t+1}. The new prediction is used as the *target* for adapting the prediction at s_t. This kind of learning is widely used in reinforcement learning because it can be implemented incrementally and online, because it can learn from incomplete sequences, and because it often converges more rapidly than conventional methods, particularly for nonstationary tasks.

Building predictions upon other predictions brings up special problems relating to step size. First, how does the process start? Initially all predictions may be arbitrary, independent of any data.[1] How can these be used as the targets for learning? If they are used, they would seem to inherently introduce *bias* into the learning procedure. In conventional learning procedures it is possible to eliminate the potential bias due to the initial predictions by an appropriate choice of the step-size parameter. The first time a state is visited, for example, the step-size parameter is 1, causing the old estimate to be completely replaced by the first target—a real, unbiased, data point. In the case of temporal-difference learning it is not this easy, because the first target is itself a prediction and potentially biased. Eliminating the effect of initial bias is a special problem for temporal-difference learning.

A solution may lie in part with the *eligibility trace* (λ) mechanism (Klopf, 1982; Sutton, 1988; Watkins, 1989; Dayan, 1992) used in methods such as TD(λ). These methods shift the prediction for s_t not just toward the prediction for s_{t+1}, but to a mixture of the predictions for s_{t+1}, s_{t+2}, s_{t+3}, etc., up to and including the real final outcome. How should these various predictions best be combined? Should it depend on their bias, their variance, or on how much prior experience there has been with them? For example, initially one might want a mixture that puts all its weight on the final outcome, so as not to be influenced by the bias in the initial predictions. Later, as the predictions become more informed and meaningful, they could be given greater emphasis. In general, a single trajectory may pass through states some of which are highly experienced and some of which are totally naive. How can this be taken in account in combining their predictions into overall targets? If this could be done properly, it might not only reduce bias, but also result in a significantly more efficient learning procedure.

The problems of bias and of appropriately mixing predictions into a target for TD methods are closely related. Within existing methods such as TD(λ) they both hinge on the choice of the step-size, α, and the trace factor, λ. Ideally, these parameters ought not simply be fixed or reduced according to a fixed sched-

[1] Of course in practical applications one will want to "load" them with any available prior knowledge. In this paper we instead explore a mathematical perspective in which we strive to eliminate all prior bias. This is done for simplicity only; ultimately the two perspectives should be integrated.

ule, as in prior work. Instead, we seek a way of setting them as a function of state and of a state's experience that will eliminate bias and support rapid convergence. What we seek ideally is something like what already exists for conventional Monte Carlo methods. These methods use, for each state, a step size of $1/n_s$, where n_s is the number of times s has been visited. This removes initial bias, because the step size is 1 on the first visit, and converges nicely, because each outcome that follows s is given equal weight. It is unlikely that there exists any solution for TD methods of quite this simplicity, but some of the new algorithms we present here approach it.

2 The Problem

Consider the problem of predicting the terminal outcome of a Markov chain from knowledge of the current state, on the basis of experience with past state trajectories and outcomes. An individual trajectory is a sequence of states followed by a numerical outcome

$$s_1 \leadsto s_2 \leadsto \cdots \leadsto s_m \leadsto z \tag{1}$$

where $z \in \Re$ and the transitions are generated by a Markov chain.[2] Without loss of generality the initial state s_1 can be taken to be a particular fixed state, and the last transition can be taken to be into a single terminal state, with the expected value of the outcome depending on the final nonterminal state, s_m. For each nonterminal state, s, our task is to maintain an estimate (or prediction) V_s of the expected value of the outcome once state s has been entered:

$$V_s \approx E\{z \mid s\}.$$

In one sense, the ideal solution is to construct a model of the underlying Markov chain and then compute this expected value assuming that the model was exactly correct. We will rule out this *certainty equivalent (CE)* solution because of its prodigious requirements for memory and computation.[3]

The conventional Monte Carlo (MC) predictions (e.g., see Barto & Duff, 1994) are the average of the actual outcomes that have followed visits to the state:

$$V_s = \frac{\sum_{k \in S_s} z_k}{|S_s|} \tag{2}$$

where s is a nonterminal state, S_s is the set of all previously observed sequences that contain s, and $|S_s|$ is the number of such sequences. If we assume that the state sequences do not contain any cycles (state repetitions such that $s_a = s_b$ for $a \neq b$), then it is natural to implement this method incrementally. Each time a sequence completes, then for each state s visited during the sequence we update

$$n'_s = n_s + 1,$$

so that $n'_s = |S_s|$ is the number of times state s has been visited, and

$$\begin{aligned} V'_s &= \frac{n_s}{n'_s} V_s + \frac{1}{n'_s} z \\ &= V_s + \frac{1}{n'_s}(z - V_s) \end{aligned} \tag{3}$$

which has a natural interpretation as an error-correction learning rule with an error of $z - V_s$ and a step size of $\alpha = \frac{1}{n'_s}$. If this update rule is followed, then (2) will hold at each step. Thus, the estimates will be unbiased and their variance will decrease in the usual way according to the law of large numbers.

We call this algorithm the *Monte Carlo (MC)* rule. It is well known (Sutton, 1988) that TD(1), the TD(λ) algorithm with $\lambda = 1$, can achieve an overall effect similar to that of an error correction rule such as (3). In fact, for the case without cycles, TD(1) with a step size of $\alpha = \frac{1}{n'_s}$ results in exactly the same estimates at the end of each state trajectory as the MC rule, as we confirm empirically in Section 5.

3 A TD Analog of the MC Rule

Just as the MC rule maintains each state's estimate as the average of the actual outcomes that follow the state, a TD analog would maintain its estimates as the average of the estimates that follow the state. Thus, if you have visited s twice, passing next to states i and j, where you found estimates v_i and v_j, then $V_s = \frac{1}{2}v_i + \frac{1}{2}v_j$ (we distinguish v_i and v_j, the estimates for i and j *at the time* those states were entered from s, from the current estimates V_i and V_j for those states). The problem with this idea as stated is that the first time a state is entered its estimate is not based on any prior data. Its value is completely biased and "contentless". It is not clear how to average in these contentless initial estimates with later ones. One solution might be to ignore transitions to contentless states, learning nothing from them and not counting them as visits to the source state. This would enable us to keep all the estimates unbiased, but obviously would also throw away useful information. For long sequences, we would be reduced to propagating information back only one state per pass through the sequence (as is done by the original TD(0) algorithm).

The key idea to a better solution is to think of working *backwards* along the sequence, from its termination toward the current transition. If one were somehow able to look ahead to the end and update backwards,

[2] All of our algorithms extend trivially to the case in which payoffs occur on every time step and the objective is to predict their expected (possibly discounted) cumulative sum.

[3] This method is approximately $O(N^2)$ in memory and $O(N^3)$ in computation, where N is the number of states. Most of the methods we consider here are only $O(N)$ in memory and computation, though we will stray upwards from there in some cases.

then the estimate at the next state would never be "contentless" because it would always contain at least one real outcome (that from the current sequence). Moreover, one would always be using the information from the completion of the current sequence, so one's estimate will always be more up to date. Can this be done incrementally? Surprisingly, the answer is *Yes*, by the use of an eligibility trace of a particular form.

First we define the backwards computation. Let V'_s denote the estimate after it has been updated by working backwards. For any transition $i \rightsquigarrow j$ we can see that

$$V'_i = \frac{n_i}{n'_i} V_i + \frac{1}{n'_i} V'_j \quad (4)$$

$$= V_i + \frac{1}{n'_i}(V'_j - V_i) \quad (5)$$

where V'_j is defined to be z if j is the terminal state, and of course $n'_i = n_i + 1$. Clearly, this is a TD rule, because the error in (5) is the temporal difference between two successive predictions, $V'_j - V_i$. But it is also of the same form as the MC rule (3) in the way in which it incrementally maintains the sample average. Here we maintain the sample average of the (backwards computed) estimates at the following states. This is a backwards computation because the TD targets are the primed estimates, the estimates after they have been updated. Thus, to implement (5) one would have to work backwards, first computing V'_j and then V'_i, and so on.

Let us examine the backwards rule (5) further, as it is in some sense an ideal TD rule. A simple inductive proof shows that all the estimates are unbiased because they are averages (in the sense of (4)) of other unbiased estimates. Note that (5) clearly does achieve some of our goals. For example, the predictions formed by this rule will be independent of the initial estimates. If this is the first time that state i has been visited, then n'_i will be 1, and the rule reduces to $V'_i = V'_j$. The effect is as if the λ of TD(λ) were held at 1 while passing through never experienced states. When we do reach states with some experience (or a final outcome) that experience is passed all the way back through all the inexperienced states, updating them all fully in one pass.

Now we return to the question of implementing (5) in a forward and incremental way, through the use of eligibility traces. Without loss of generality we can simplify notation by numbering the states in the sequence in numerical order

$$1 \rightsquigarrow 2 \rightsquigarrow 3 \rightsquigarrow \cdots \rightsquigarrow m \rightsquigarrow z. \quad (6)$$

Then we can expand our ideal TD rule (5) as

$$\begin{aligned}
V'_i &= V_i + \frac{1}{n'_i}(V'_{i+1} - V_i) \\
&= V_i + \frac{1}{n'_i}\left[V_{i+1} + \frac{1}{n'_{i+1}}(V'_{i+2} - V_{i+1}) - V_i\right] \\
&= V_i + \frac{1}{n'_i}(V_{i+1} - V_i) + \frac{1}{n'_i}\frac{1}{n'_{i+1}}(V'_{i+2} - V_{i+1}) \\
&= V_i + \frac{1}{n'_i}(V_{i+1} - V_i) + \frac{1}{n'_i}\frac{1}{n'_{i+1}}(V_{i+2} - V_{i+1}) \\
&\qquad + \frac{1}{n'_i}\frac{1}{n'_{i+1}}\frac{1}{n'_{i+2}}(V'_{i+3} - V_{i+2}) \\
&= V_i + \sum_{j=i}^{m}(V_{j+1} - V_j)\prod_{k=i}^{j}\frac{1}{n'_k} \quad (7)
\end{aligned}$$

where V_{m+1} is defined to be z, as usual. This is of the form of TD(λ), where λ varies over time according to $\lambda_t = \frac{1}{n'_{s_t}}$. Thus, we can implement the ideal "backwards" rule (5) by the following incremental algorithm. At the beginning of each sequence we set the eligibility trace e_s for each state to 0. Then, on each transition $i \rightsquigarrow j$ we update, in order:

$$n_i := n_i + 1$$

$$e_i := 1$$

$$V_k := V_k + \frac{1}{n_i}(V_j - V_i)e_k \qquad \forall k$$

$$e_k := e_k \frac{1}{n_i} \qquad \forall k.$$

This algorithm achieves exactly (5) for the case in which their are no cycles in the sequence. Let us call this algorithm TD(1/n), because it is like TD(λ) with $\alpha = 1/n_s$ and $\lambda = 1/n_s$. As we confirm empirically in Section 5, this algorithm produces exactly the same estimates (at the end of each state trajectory) as TD(0) applied backwards with a step size of $\alpha = \frac{1}{n_s}$.

One might hope that the estimates of TD(1/n) would be more accurate than those of the MC algorithm. Unfortunately, this is not always the case. For example, consider TD(1/n) receiving two estimates v_1 and v_2 at two times from downstream states. The estimate formed, $\frac{1}{2}v_1 + \frac{1}{2}v_2$, is an excellent one if v_1 and v_2 are from two different downstream states, but what if they are from the same state? In this case, a simple average puts too much weight on v_1, the earlier estimate, because the outcomes that went into forming it also went into forming v_2. What we would really prefer to do is *replace* v_1 with v_2, but to do that requires keeping track of which estimates are due to which downstream states. This is what the algorithm described in the next section does.

4 A Corrected TD Algorithm

We now introduce our second new algorithm, the *Corrected TD (TDC)* algorithm. This algorithm maintains a record of n_{ij}, the number of times the $i \rightsquigarrow j$ transition has occurred, and of v_{ij}, the most recently known value of V_j. The objective of the TDC algorithm is to maintain

$$V_i = \sum_j \frac{n_{ij}}{n_i} v_{ij} \quad (8)$$

If j is the terminal state, then the corresponding v_{ij} is the current estimate of the expected outcome when an outcome is received immediately following state i. Achieving (8) is really a general goal for all TD algorithms. Suppose we have an $i \leadsto j$ transition. V_i should be updated as

$$\begin{aligned} V_i' &= \sum_k \frac{n_{ik}'}{n_i'} v_{ik}' \\ &= \frac{1}{n_i'} \left[\sum_{k \neq j} n_{ik} v_{ik} + n_{ij}' V_j' \right] \end{aligned}$$

because $n_{ik}' = n_{ik}$ and $v_{ik}' = v_{ik}$ for $k \neq j$, and $v_{ij}' = V_j'$. This can be rewritten in a TD-like form as follows

$$\begin{aligned} V_i' &= \frac{1}{n_i'} \left[\frac{n_i}{n_i} \sum_{k \neq j} n_{ik} v_{ik} + n_{ij} v_{ij} - n_{ij} v_{ij} + n_{ij}' V_j' \right] \\ &= \frac{n_i}{n_i'} \sum_k \frac{n_{ik}}{n_i} v_{ik} + \frac{1}{n_i'} \left[n_{ij}' V_j' - n_{ij} v_{ij} \right] \\ &= (1 - \frac{1}{n_i'}) V_i + \frac{1}{n_i'} \left[n_{ij}(V_j' - v_{ij}) + V_j' \right] \\ &= V_i + \frac{1}{n_i'} \left[V_j' - V_i + n_{ij}(V_j' - v_{ij}) \right], \qquad (9) \end{aligned}$$

or

$$\begin{aligned} \Delta V_i' &= V_i' - V_i \\ &= \frac{1}{n_i'} \left[V_j' - V_i + n_{ij}(V_j' - v_{ij}) \right] \\ &= \frac{1}{n_i'} \left[V_j - V_i + \Delta V_j' + n_{ij}(V_j - v_{ij} + \Delta V_j') \right] \\ &= \frac{1}{n_i'} \left[V_j - V_i + n_{ij}(V_j - v_{ij}) + \Delta V_j'(1 + n_{ij}) \right] \\ &= \Delta V_i + \frac{n_{ij}'}{n_i'} \Delta V_j' \qquad (10) \end{aligned}$$

where

$$\Delta V_i = \frac{1}{n_i'} \left[V_j - V_i + n_{ij}(V_j - v_{ij}) \right]. \qquad (11)$$

When j is the outcome state, then we define $V_j = v_{ij}$ in (11) and, in (10), we define $\Delta V_j'$ by

$$\Delta V_j' = V_j' - V_j = \frac{1}{n_{ij}} (z - v_{ij}). \qquad (12)$$

This arranges for $V_j' = v_{ij}'$ to be the average of the outcomes received from state i, as in the MC algorithm.

Now we are ready to write a recursive equation like (7). Switch to the notation in which the states in the sequence are in numerical order, $1 \leadsto 2 \leadsto \cdots \leadsto m \leadsto z$. Then we have

$$\begin{aligned} V_i' &= V_i + \Delta V_i + \frac{n_{i,i+1}'}{n_i'} \Delta V_{i+1}' \\ &= V_i + \Delta V_i + \frac{n_{i,i+1}'}{n_i'} \Delta V_{i+1} \\ &\quad + \frac{n_{i,i+1}'}{n_i'} \frac{n_{i+1,i+2}'}{n_{i+1}'} \Delta V_{i+2} + \cdots \\ &\quad + \frac{n_{i,i+1}'}{n_i'} \cdots \frac{n_{m,m+1}'}{n_m'} \Delta V_{m+1} \\ &= V_i + \sum_{j=i}^{m+1} \Delta V_j \prod_{k=i}^{j-1} \frac{n_{k,k+1}'}{n_k'}. \qquad (13) \end{aligned}$$

And there is a corresponding algorithm. Initialize $n_i := n_{ij} := 0, \forall i, j$ and, at the start of each sequence, set $e_i := 0, \forall i$. Then, on each transition, $i \leadsto j$, update, in order:

$$\begin{aligned} n_{ij} &:= n_{ij} + 1; \qquad n_i := n_i + 1 \\ e_i &:= 1 \\ \Delta V_i &:= \frac{1}{n_i} \left[V_j - V_i + (n_{ij} - 1)(V_j - v_{ij}) \right] \\ V_k &:= V_k + \Delta V_i e_k \qquad \forall k \\ e_k &:= e_k \frac{n_{ij}}{n_i} \qquad \forall k. \end{aligned}$$

In addition, for the final transition $i \leadsto j$ of the sequence, where j is the outcome state, update

$$V_k = V_k + \frac{1}{n_{ij}} (z - v_{ij}) e_k \qquad \forall k$$

and then go back and set

$$v_{ij} = V_j$$

for all nonterminal transitions $i \leadsto j$ in the sequence. (This can also be done incrementally, if desired.) This completes the specification of the TDC algorithm.

Note that the TDC algorithm uses significantly longer traces than does TD(1/n). The new traces fall with $n_{k,k+1}'/n_k'$ rather than with $1/n_k'$. This may be a better trace decay to use even if one uses the first rule for $\Delta V_i'$ rather than the corrected one given here. Let us call that algorithm TD(n/n), because it is like TD(1/n) except it uses $\lambda = n_{ij}/n_i$ rather than $\lambda = 1/n_i$.

5 Empirical Results on Acyclic Tasks

To test the new algorithms (and our analyses of them), we applied them to two kinds of randomly constructed test problems: pyramid tasks and random acyclic tasks. Both kinds of problems were constructed so that their state trajectories would contain no cycles.

In the *pyramid* tasks, the states were organized in a pyramid, as shown in Figure 1. All state trajectories started at the top of the pyramid and ended after exiting from the base. At each step, a transition occurred from the current state to one of the four (or less, if there were less than four states in the level below) nearest states in the level below it. The transition probabilities were selected randomly as a uniform, random partition of the unit interval. After reaching the base of the pyramid, the next transition was to the terminal state. The outcome was a deterministic function of the base state, varying linearly from 0 at the far left to 1 at the far right.

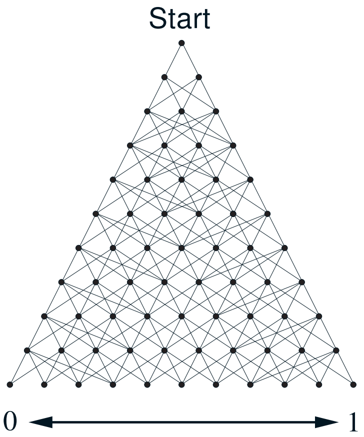

Figure 1: A pyramid task. All state trajectories progressed downward, along one of the transitions shown as solid lines. The transition probabilities were selected randomly. This is a pyramid of height 11, whereas the experiment used a pyramid of height 15.

Figure 1 shows a pyramid of height 11. In our experiment we used a pyramid of height 15. Each trip from peak to base of the pyramid was called a trial, and 1000 trials made up a complete run. The results we show are averages over 50 runs, each with a different randomly constructed pyramid task. After each trial, we measured the predictions made by the various algorithms at each of the 10 states in the 10th level in the pyramid. Figure 2 shows the root-mean-squared error (RMSE) between these predictions and the true expected outcomes, which we computed analytically from knowledge of the task. In computing the RMSE we took into account the probability p_i of visiting each state during a trial, which we also computed analytically. In addition, we excluded states that had never been visited. That is, we used

$$\text{Squared Error} = \frac{\sum_{i \in S} p_i (V_i - V_i^*)^2}{\sum_{i \in S} p_i}$$

where V_i^* is the correct prediction for state i and S is the set of 10th level states that have been visited at least once. This measure of squared error was then averaged over the 50 different runs, and the square root taken, to obtain the RMSE. Figure 2 is a plot of this RMSE after every 10 trials, starting after trial 1.

Results are shown for all the algorithms discussed above, including MC, TD(1/n), TD(n/n), and TDC. The lowest (best) learning curve, labeled "CE" is the performance level of the certainty equivalent predictor (see Section 2). Also shown are results for conventional TD(λ) at λ's of 0 and 1, each with an appropriate

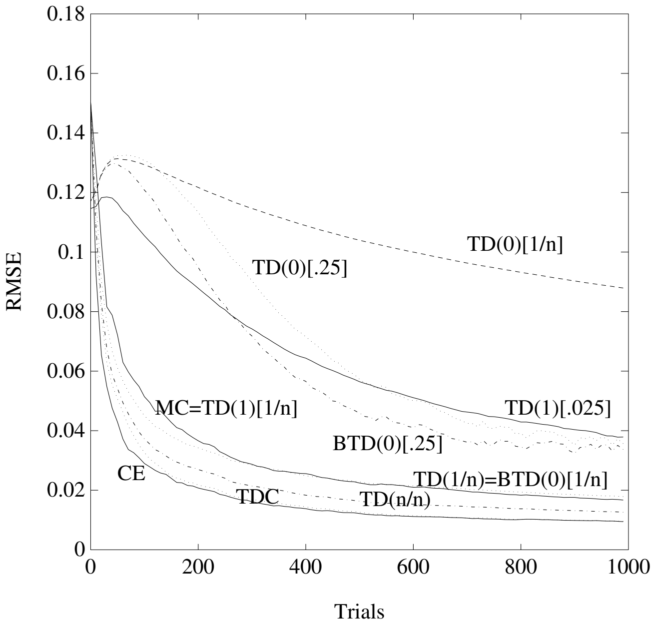

Figure 2: Results on the Pyramid Task. Numbers in brackets are α values. For example, TD(0)[.25] is TD(0) with $\alpha = 0.25$.

α value, and for backwards TD(0), the TD(0) algorithm applied backwards along a trial's state trajectory. The highest (worst) learning curve is for TD(0) with a step size of $\alpha = \frac{1}{n_s}$. The backwards version of this algorithm produced exactly the same estimates as TD(1/n), confirming the analysis of Section 3, and thus only one learning curve is shown for these two algorithms. Similarly, TD(1) with a step size of $\alpha = \frac{1}{n_s}$ performed identically to MC, and only one learning curve is shown. The backwards version of this algorithm should also perform identically to MC, but this was not explicitly tested.

The conventional TD algorithms, TD(0), TD(1), and BTD(0), are all biased, and so their predictions at each time depend on how they were initialized. In this case we initialized the predictions all to 0.5. This is in fact a pretty good initial prediction for this task, and for the first few trials these methods performed better than the unbiased methods. However, within 5–10 trials all the unbiased algorithms caught up and then performed much better. The biased algorithms appear to actually get worse for the first 50 trials or so, but this is an artifact of our performance measure. If we include all states, not just those in S, then this temporary increase disappears. The most interesting results on this task are the relative performance of the unbiased algorithms. On this task, TD(n/n) and TDC perform better than MC. TDC even approaches the performance of CE near the end of the run. TD(1/n) initially performs better than MC, but then loses its advantage after about 300 trials. It is not clear which of these two does better in the long run.

The *random acyclic* tasks were constructed as follows. There were 16 states in total, numbered from 1 to 16. All trajectories started in state 1 and progressed always to higher numbered states. States 15 and 16 were terminal states. Ending in 15 yielded an outcome of 0, whereas ending in 16 yielded an outcome of 1. The possible successors of each nonterminal state were selected randomly from all the higher numbered states. Up to 3 possible successors were selected in this way, and were then given random probabilities by uniformly partitioning the unit interval. After a task was constructed it was checked for ergodicity (to make sure all states could be reached with nonzero probability). If it was not ergodic, it was discarded and a new task was constructed. Only the 5 unbiased algorithms were run on this task, each for 100 runs (with a different randomly generated task) of 200 trials each. Figure 3 shows the RMSE after every trial. On this task, TD(n/n) and TDC perform better than MC, but TD(1/n) performs worse.

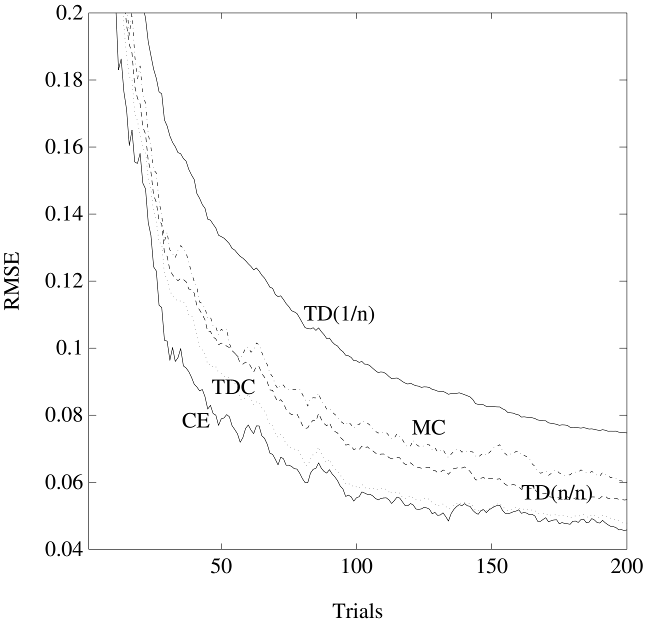

Figure 3: Results on Random Acyclic Task.

6 Conclusions

What general conclusions can we draw from these results? The new algorithms do indeed perform strikingly better than conventional, biased TD methods such as TD(λ) on these problems, and much better than the naive idea of simply using $\alpha = 1/n_s$. In many cases the new algorithms perform better than conventional Monte Carlo methods, but the only algorithms that do this consistently are TD(n/n) and TDC. These algorithms may be of limited practical interest because they are computationally expensive, at least $O(N^2)$, where N is the number of states.

Perhaps the biggest question about the new methods is how they might be extended beyond acyclic tasks. We have been looking into this extensively, but have not yet found completely satisfactory generalizations to the cyclic case. In particular, the TDC algorithm probably has no cyclic analog.

These results shed some light on longstanding questions about temporal-difference learning. One is the tradeoff between bias and variance. Temporal-difference learning is sometimes thought of as inherently biased, as if its sometimes speedy convergence was at the cost of introducing bias. Here we have exhibited several temporal-difference learning methods that are unbiased, and which still converge faster, at least in some cases, than conventional Monte Carlo learning methods.

The second general issue that these results shed light on is the choice of the trace parameter, λ. The excellent performance of the TDC and TD(n/n) algorithms strongly suggests that λ should be chosen at each time step approximately equal to the transition probability of the immediately preceding transition. In many applications this can probably be used as a rule of thumb to provide guidance in selecting a fixed λ for TD(λ).

Acknowledgments

The authors are grateful to Mike Duff, Ron Williams, Shaijan Mahamud, and Andy Barto for discussions and ideas which improved our understanding of step-size and trace-length issues. Satinder P. Singh was supported by grants to Michael I. Jordan (Brain and Cognitive Sciences, MIT) from the McDonnell-Pew Foundation, from ATR Human Information Processing Research Laboratories, and from Siemens Corporation.

References

Barto, A., Duff, M. (1994) Monte Carlo Matrix Inversion and Reinforcement Learning. *Advances in Neural Information Processing Systems 6.*

Dayan, P. (1992) The Convergence of TD(λ) for General λ. *Machine Learning 8*, 341–362.

Klopf, A.H. (1982) *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*, Washington DC: Hemisphere/Harper & Row.

Sutton, R.S. (1988) Learning to Predict by the Methods of Temporal Differences. *Machine Learning 3*, 9–44.

Watkins, C.J.C.H. (1989) *Learning From Delayed Rewards.* Cambridge University PhD Thesis (Psychology).