TD(λ) Networks: Temporal-Difference Networks with Eligibility Traces

Brian Tanner BTANNER@CS.UALBERTA.CA
Richard S. Sutton SUTTON@CS.UALBERTA.CA
Reinforcement Learning and Artificial Intelligence Laboratory, University of Alberta, Canada

Abstract

Temporal-difference (TD) networks have been introduced as a formalism for expressing and learning grounded world knowledge in a predictive form (Sutton & Tanner, 2005). Like conventional TD(0) methods, the learning algorithm for TD networks uses 1-step backups to train prediction units about future events. In conventional TD learning, the TD(λ) algorithm is often used to do more general multi-step backups of future predictions. In our work, we introduce a generalization of the 1-step TD network specification that is based on the TD(λ) learning algorithm, creating TD(λ) networks. We present experimental results that show TD(λ) networks can learn solutions in more complex environments than TD networks. We also show that in problems that can be solved by TD networks, TD(λ) networks generally learn solutions much faster than their 1-step counterparts. Finally, we present an analysis of our algorithm that shows that the computational cost of TD(λ) networks is only slightly more than that of TD networks.

Temporal-difference (TD) networks are a formalism for expressing and learning grounded knowledge about dynamical systems (Sutton & Tanner, 2005). TD networks are one approach that can be used to learn a predictive representation of state (Littman et al., 2002; Jaeger, 1998; Rivest & Schapire, 1990). Predictive representations are one of several approaches to learning generative models of dynamical systems that can be used for planning and/or reinforcement learning. Predictive representations can be learned from data (Wolfe et al., 2005), and have been shown to have richer representational power than competing approaches like POMDPs and n^{th}-order Markov models (Singh et al., 2004).

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

Predictive representations represent the state of a dynamical system as a vector of predictions about future action–observation sequences. The hypothesis that important knowledge about the world can be represented strictly in terms of predictions of relationships between observable quantities is a novel idea that distinguishes predictive representations from other approaches. In *predictive state representations* (PSRs) introduced by Littman et al. (2002), each prediction is an estimate of the probability of some sequence of observations given a sequence of actions. TD networks generalize PSRs; in TD networks each prediction is an estimate of the probability or expected value of some function of future predictions and observations. The predictions are thought of as “answers” to a set of “questions” represented within the TD network.

The essential idea of TD learning can be described as learning a guess from a guess. Before TD networks, the two guesses involved were predictions of the same quantity at two points in time, for example, of the discounted future reward at successive time steps. TD networks generalize TD methods by allowing the second guess to be different from the first. The current TD network learning algorithm uses 1-step backups; the target for a prediction comes from the subsequent time step. In conventional TD learning, the TD(λ) algorithm is often used to do more general, n-step backups. Rather than a single future prediction, n-step backups use a weighted average of future predictions as a target for learning (Sutton, 1988). TD(λ) is really a spectrum of algorithms, controlled by the continuous valued parameter $\lambda \in [0, 1]$. TD(λ=0) uses the earliest possible prediction as the target for learning while TD(λ=1) uses the latest possible prediction as the target. For other values of λ, the targets for learning are distributed among all of the predictions along the way.

A worthwhile contribution to the work on predictive representations would be a least-squares algorithm for learning TD networks similar to LSTD (Boyan, 2002). We feel that TD networks are still not well understood and that the TD(λ) algorithm still remains of widespread interest, leaving LSTD networks an interesting possibility for future work.

In this work, we present an extension to the TD network learning algorithm that uses n-step backups of future predictions and observations. We call our approach TD(λ) networks. In this work we show that TD(λ) networks generalize the previous approach; that the TD network specification is identical to a TD(λ=0) network. From this point forward, we refer to both 1-step and n-step TD networks simply as TD networks. When the distinction is important, we will refer to the previous specification as TD(0) networks and our new specification as TD(λ) networks.

In Section 1 we review the TD(0) networks specification. We present and discuss the new learning algorithm in Sections 2 and 3 respectively. In Section 4 we compare the performance of TD(λ) networks for various values of λ, including TD(λ=0) networks. In Section 5 we have included a detailed cost analysis of our implementation the algorithm. Finally, we conclude and discuss our results in Section 6.

1. TD Networks

We present a brief review of the details of TD networks here, for further details please consult the original TD networks paper (Sutton & Tanner, 2005).

TD networks learn to predict future observations that will be generated by their environment (a dynamical system). At each of a series of discrete time steps t, the environment generates an observation $o_t \in \mathcal{O}$, and the agent takes an action $a_t \in \mathcal{A}$. The action and observation events occur in sequence, $a_{t-1}, o_t, a_t, o_{t+1}, a_{t+1}, o_{t+2}$, with each event potentially dependent only on those preceding it in the sequence. This sequence will be called *experience*. In this work, we will use the TD networks to predict action-conditional functions of future experience in partially-observable environments. Throughout this paper we consider the observation to be a single bit, either 0 or 1. In general, with more observations, o_t can be represented as a vector of bits.

A TD network is a network of nodes each of which represents a single scalar prediction. Each node has links to other nodes and/or the observation from the environment. These links represent the targets for learning — what the node should predict. Each of these nodes is some question about the system, and accordingly this set of nodes and links is called the *question network*.

Figure 1 is an example of the type of question network we use in this paper. In this example, the question being asked by the node labeled y^1 at time t is "If the next action is $a1$, what is the probability that the next observation o_{t+1} will be 1?" Similarly, the question asked by the node labeled y^4 at time t is "If the next action is $a2$, what is the expected value of the prediction from node y^1 at time $t+1$?"

Each node in a TD network can be thought of as a function approximator. The inputs to each node is defined by a set of interconnections called the *answer network*. The prediction made by each node is a function of these input features. These predictions provide the answers to the questions asked by the question network.

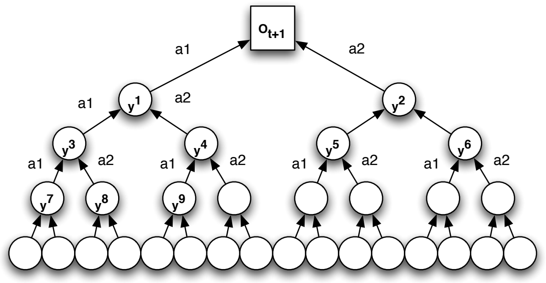

Figure 1. Symmetric action-conditional question network. The network forms a symmetric tree, with a branching factor equal $|A|$. This example has depth $d = 4$. Some of the labels have been left out of this diagram for clarity, each of these nodes should have a label y^i and each is conditioned on some action.

Formally, the prediction for node i at time step t is denoted as $y^i_t \in [0, 1]$, $i = 1, \ldots, n$. The column vector of predictions $\mathbf{y}_t = (y^1_t, \ldots, y^n_t)^T$ is updated by:

$$\mathbf{y}_t = \boldsymbol{\sigma}(\mathbf{W}_t \mathbf{x}_t) \quad (1)$$

where $\mathbf{x}_t \in \Re^m$ is a feature vector, $\mathbf{W}_t$ is a $n \times m$ matrix of modifiable weights, and σ is the S-shaped logistic function $\sigma(s) = \frac{1}{1+e^{-s}}$.

The feature vector is a function of the preceding action, observation, and node values.

$$\mathbf{x}_t = \mathbf{x}(a_{t-1}, o_t, \mathbf{y}_{t-1}) \in \Re^m \quad (2)$$

In our experiments, $\mathbf{x}_t$ has one component for each unique combination of the current observation and previous action (one of which is 1, the rest 0), and n continuous valued components equal to the predictions $\mathbf{y}_{t-1}$.

From this description, the operation of the question and answer network may not be completely clear. At a high level, the operation of a TD network can be explained in five steps:

1. Choose an action a_{t-1} and receive an observation from the environment o_t

2. Calculate the input vector $\mathbf{x}_t$ as a function of the previous predictions $\mathbf{y}_{t-1}$, the action just taken a_{t-1}, and the new observation o_t
3. Create the new predictions $\mathbf{y}_t = \boldsymbol{\sigma}(\mathbf{W}_t \mathbf{x}_t)$
4. Calculate the targets $\mathbf{z}_{t-1}$ for the previous predictions $\mathbf{y}_{t-1}$ using $\mathbf{y}_t$ and the observation o_t according to the question network's links and the action conditions
5. Update the weights W according to $(\mathbf{z}_{t-1} - \mathbf{y}_{t-1})$

We turn our attention back to the question network. In the most general case, the definition of a question is very loose; questions can be any function of future predictions or observations. In this paper we consider the special case of question network in which each node i has a single target, either some other prediction or the observation at the next time step. We call this type of question network a *single-target* question network. This special case includes all question networks that have been considered in previous work. We refer to the target of node i as the "parent" of i or $p(i)$. The later parents of node i: $\{p(p(i)), p(p(p(i))), ...\}$ are written in the short form $\{p^2(i), p^3(i), ...\}$.

In Figure 1 the parent of Node 9 is Node 4, $p(9) = 4$. The parent of Node 4 is Node 1 ($p(4) = 1$), so $p(p(9)) = p^2(9) = 4$. The third parent of Node 9, $p(p(p(9))) = p^3(9) = o$, the observation bit.

Formally, the target for node i is z^i:

$$z_t^i = \begin{cases} y_{t+1}^{p(i)} \text{ or } o_{t+1} & \text{if } c_t^i = 1 \\ y_t^i & \text{if } c_t^i = 0 \end{cases} \tag{3}$$

where a_t is the next action, and $c_t^i \in \{0, 1\}$ corresponds to whether the action condition for y^i was met at time t.

Each component w_t^{ij} of $\mathbf{W}_t$ is updated by the learning rule:

$$w_{t+1}^{ij} - w_t^{ij} = \alpha(z_t^i - y_t^i)\frac{\partial y_t^i}{\partial w_t^{ij}} \tag{4}$$

where α is a step-size parameter. Notice that if $c_t^i = 0$, then the error $(y_t^i - y_t^i)$ will be zero.

In this paper we analyze this learning rule directly so it is useful to simplify it as much as possible.

Using the prediction update equation from Equation 1, the exact weight update rule is:

$$\Delta w_t^{ij} = \alpha(z_t^i - y_t^i) y_t^i (1 - y_t^i) x_t^j \tag{5}$$

2. TD(λ) Networks

The target function described in Equation 3 is only correct for single-step TD(0) updates. Each prediction made at time t has a TD target that may become available at time $t+1$. This TD target may itself be a prediction of some other value that will be available at time $t+2$. In this case it is possible to "unroll" the first prediction, to ask a question at time t about an event at time $t+2$. A question can be unrolled step by step until it is asking a question about the data, the observation bit. Each prediction made at time t is indirectly predicting several events at different moments in time and therefore has a different target for each moment.

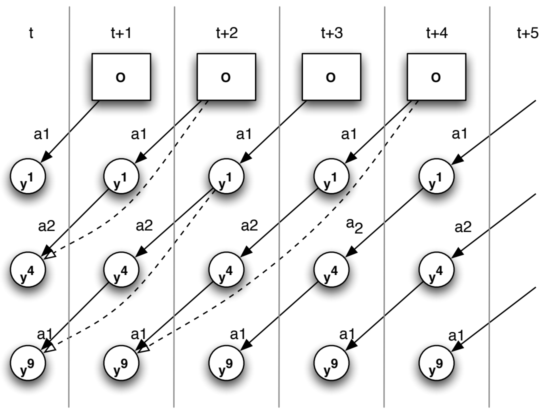

Figure 2. Extended target flow diagram for nodes {1,4,9} of question network in Figure 1. The links in this diagram show the flow of target values back toward the original predictions. The solid links are the 1-step TD targets for these predictions. The dashed links are a sample of the unrolled multi-step targets. We have left out the action-condition labeling on the dashed links to reduce the clutter.

Using the parent function $p(i)$, the relationship between targets follows the structure of the question network. The first target for y_t^i comes directly from the 1-step TD relationship in the question network, and is simply z_t^i. The second target is recursively defined, it is the next target of the parent of node i, $z_{t+1}^{p(i)}$. Following this process, we can derive a k step sequence of targets for y_t^i: $z_t^i, z_{t+1}^{p(i)}, z_{t+2}^{p^2(i)}, ...z_{t+k-1}^{p^{k-1}(i)}$, where $p^k(i)$ is the observation bit.

In order to keep notation a simple as we can, for the following section we will consider a single node i and a single starting time step t. Under these conditions, the first target of a prediction can be written as $z(0)$. The subsequent targets are $z(1), z(2), z(3), ..., z(k-1)$.

To make this concrete, consider the prediction node 9 in Figures 1 and 2. In this case, $z(0) = y_{t+1}^4$, $z(1) = y_{t+2}^1$, and $z(2) = o_{t+3}$. Although each target in this sequence is a prediction of the same event o_{t+3}, each was calculated using different information. Our intuition is that the targets generated later in time may sometimes be more accurate than the earlier targets. Some combination of the targets from this sequence may then be better than any particular single target in the sequence.

While the conditions of the predictions match the experience of the agent, this sequence of targets is available and valid. If the agent's experience diverges from the conditions of the question network, no further updates are performed. If multiple targets become available, we can imagine using any (or all) of these targets for learning. As with TD(λ), we propose an exponentially weighted average of these targets. The multi-step weighted target for prediction y_t^i is denoted as v_t^i, where:

$$v_t^i = ((1-\lambda)\sum_{n=0}^{k-1}\lambda^n z(n)) + \lambda^k z(k-1) \qquad (6)$$

The one-step target is given the largest weight $(1-\lambda)$, the two-step target is given $(1-\lambda)\lambda$, the three-step target is given $(1-\lambda)\lambda^2$, etc. The last item in this sequence will receive all of the remaining weight (λ^k).

Ideally, we would like to use the blended target v_t^i in an update rule such as:

$$\Delta w_t^{ij} = \alpha(v_t^i - y_t^i)y_t^i(1-y_t^i)x^j \qquad (7)$$

We desire an online, incremental algorithm where the value of v_t^i will not be available at time t. We can use some standard (and novel) tricks to implement this learning rule incrementally using a variation of eligibility traces as with TD(λ). Pseudo-code for our TD(λ) networks learning algorithm is in Figure 3. The weight update rule in this algorithm achieves the behavior of Equation 7 using incremental updates of successive targets.

Because the predictions made within TD networks are of different events, implementing eligibility traces is not as simple as with conventional TD(λ). Each prediction y_t^i needs its own eligibility trace. This makes our algorithm slightly more complicated than traditional TD(λ).

3. TD(λ) Algorithm Discussion

In a nutshell, our algorithm keeps a record of predictions and whether the conditions of the unrolled definition of those predictions are consistent with later experience of the agent. At each time step, new targets become available for past predictions. By combining the temporal-difference $\mathbf{y}_t - \mathbf{y}_{t-1}$ with historic information about the inputs to the answer network ($\mathbf{x}_{t-k}$), the weight vector $\mathbf{W}$ is updated (scaled by λ^{t-k-1}) towards the new target to improve the past prediction $\mathbf{y}_{t-k}$.

This algorithm has some interesting properties, controlled by the particular value of λ that is used.

```
$Traces \leftarrow \{\}$
**for** $t = 0$ to $T$ **do**
  $newTraces \leftarrow \{\}$
  $a \leftarrow chooseAction()$
  $o \leftarrow getObservation(a)$
  $x_t \leftarrow x(a, o, y_{t-1})$
  $y_t \leftarrow \boldsymbol{\sigma}(\mathbf{W}\mathbf{x}_t)$
  **for** $(i,k) \in Traces$ **do**
    **if** $checkCondition(p^{t-k-1}(i), a) == TRUE$ **then**
      **if** $p^{t-k}(i) \neq observation$ **then**
        $z \leftarrow y_{t-1}[p^{t-k}(i)]$
      **else**
        $z \leftarrow o$
      **end if**
      $p \leftarrow y_{t-1}[p^{t-k-1}(i)]$
      **for** $w^j \in W[i]$ **do**
        $w^j += \alpha(z-p)p(1-p)x_k^j\lambda^{t-k-1}$
      **end for**
      **if** $p^{t-k}(i) \neq observation$ **then**
        $newTraces \leftarrow newTraces \cup (i,k)$
      **end if**
    **end if**
  **end for**
  **for** $i \in y$ **do**
    $newTraces \leftarrow newTraces \cup (i,t)$
  **end for**
  $Traces \leftarrow newTraces$
**end for**
```

Figure 3. Pseudo-code for TD(λ) learning algorithm. The algorithm uses a boolean function $checkCondition(i, a)$ which will return true if the action a is consistent with the action condition of node i, and false otherwise.

If $\lambda = 0$, the first target of a prediction will get weight $0^0 = 1$, meaning that this first target will get the full weight of the update. For subsequent targets, $0^{t-k-1} = 0$ resulting in the update having no effect. This behavior is exactly the same as the previous 1-step TD network learning algorithm.

If $\lambda = 1$, each target available gets the full weight of the update, because $1^x = 1$. Each subsequent temporal-difference effectively overwrites the update made by the previous target. The net effect is that the last available target receives the full weight of the update and the intermediate targets receive no weight. If the prediction's conditions match exactly with the stream of experience, all of the weight will go to the grounded, unrolled target. If the stream of experience diverges from the conditions of the prediction, the weight of the update will go to some intermediate TD target. This behavior is analogous to a Monte Carlo style of update, with one important difference. In a Monte Carlo approach, an update would only occur if the conditions of the completely unrolled definition of a prediction were met. This would mean that nodes at deeper depths would be exponentially less likely to receive updates, because the exact sequence of the conditions would be less likely to occur. With TD(1), these predictions will always receive an update as long as their first condition is met, using more of the available data.

Finally, if an intermediate value of λ is used, then the weight of the updates are divided among the targets that become available. The remainder of the weight will always be assigned to the final available target.

4. The effect of λ in TD networks

There are certain partially-observable environments for which a TD network solution exists, but the TD(0) learning algorithm cannot find it (Tanner & Sutton, 2005). The recursive nature of TD networks allow the occurrence of information flow dependencies between the question and answer networks. When the prediction y_t^i critically depends on an input feature in $\mathbf{x}_t$ that corresponds to y_{t-1}^j, and the target z_{t-1}^j depends on y_t^i, a "chicken or egg" type dilemma can occur. The TD(0) solution to this problem that we have previously proposed is to augment the input vector $\mathbf{x}$ by including recent actions and observations in addition to the immediately previous action and observation. This recent history allows the TD(0) learning algorithm to solve problems that could not be solved without history. History also allows the TD(0) learning algorithm to solve existing problems faster than before.

Our hypothesis is that for some values of $\lambda > 0$, the TD(λ) network learning algorithm can solve this information flow dependency problem without adding additional information to the input vector.

These two distinct approaches are both trying to solve the same general problem, allowing TD networks to model more complex dynamical systems. Although we do not report results here, we expect that these two techniques can be combined to solve even more difficult problems.

In this work we report the results of comparing our TD(λ) networks to TD(0) networks in three domains. It isn't clear exactly what is the best metric to compare one TD network learning algorithm to another; we will report the average error of the answer network vs. amount of data to learn that model. This will illustrate both the speed of learning and the relative error of the models that are learned. Average error of the answer network at time t is defined as the average root mean-square error (RMSE) of all of the nodes in the network. To evaluate the success of our algorithm, the error of an individual prediction is found by asking an oracle what the unrolled answer to each question would be if the node's action sequence were performed from the current time. This oracle is not used for anything other than evaluation. In each experiment, a variety of values were used for the step size parameters α, and the results presented are for whichever value of α performed best. In general, if any value of α could solve the problem, then all values of α that we tried $\{.5, .25, .125, .0625\}$ were able to solve the problem. Lowering α increased the amount of data required to learn a solution of the same quality. In each of these experiments, the initial weights in the answer network were set uniformly, $w_{ij} = \frac{1}{|\mathbf{X}|}$. Each environment (discussed further below) is started in the state where $o_t = 1$.

The first experimental results (Figure 5) that we present are for the 6-state cycle world in Figure 4 (Tanner & Sutton, 2005). The question network used for this experiment was a chain of 5 predictions like that one in Figure 4. In previous work, we have reported that this problem cannot be solved with TD(0) networks unless the input vector $\mathbf{x}$ is expanded to include recent history. We tested TD(λ) networks on this problem for a variety of values for λ.

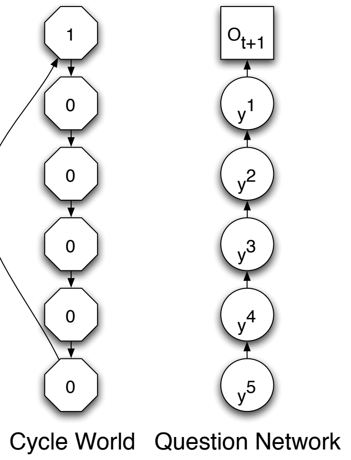

Figure 4. A counterexample for TD(0) network learning. On the left is a representation of the cycle world. This environment has six states that are cycled through deterministically. On the right is the associated question network. There are no actions in this world.

For all values of $\lambda > 0$, the TD(λ) network learning algorithm is able to find a solution. As λ increases, the amount of data required for learning decreases. A good model (RMSE $< .05$) is found with $\lambda = 1$ in under 5 000 steps. To learn an equivalent model, $\lambda = .75$ requires 7 000 steps, $\lambda = .5$ requires 32 000 steps, and $\lambda = .25$ requires 189 000 steps.

Our second experimental domain is the n-state ring world (Tanner & Sutton, 2005). The general structure of the ring world is shown in Figure 6. This domain is more complex than the cycle world because it has multiple actions. The

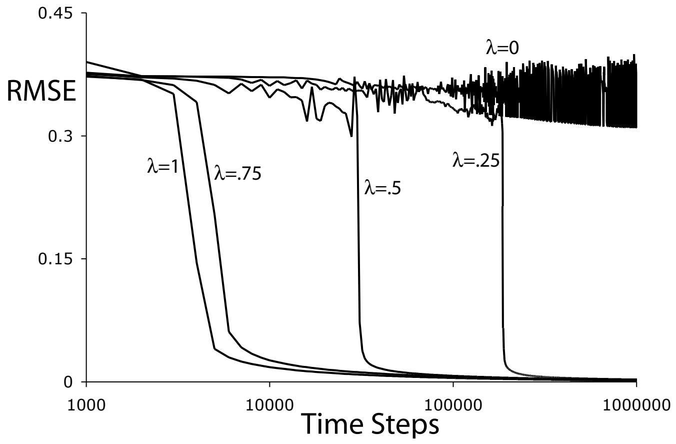

Figure 5. Learning curves of our learning algorithm for various values of λ on the 6-state cycle world. This chart represents the average RMSE over all of the nodes in the TD(λ) network as the amount of data is increased. Each data point in this graph is the average error of the network over 500 time steps. Note that the x-axis (amount of data) is an exponential scale. The cycle world is completely deterministic, so these results are for a single training run.

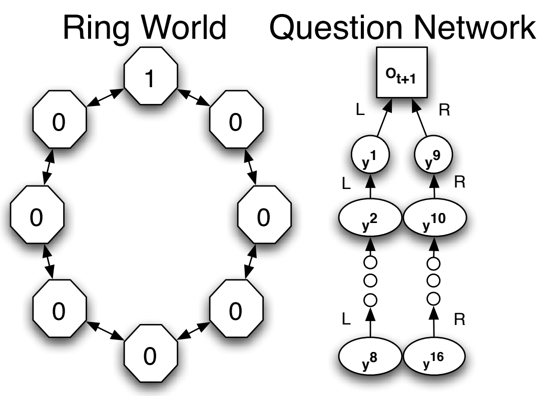

Figure 6. 8-state version of the ring world. On the left is a representation of the ring world. One of the states has an observation bit of 1, all of the others are 0. There are two actions in this world, one that moves the agent clockwise (call it ‘right’ or just R) and one that moves the agent counter-clockwise (’left’ or L). The question network on the right side of this figure is a sparse action conditional network that can represent a solution to this world. This question network has 8 levels, at each level there is a question about action L and a question about action R.

actions used to generate experience for our experiments are chosen randomly. The results from testing our algorithm for various value of λ on the 5-node and 8-node versions of the ring world are shown in Figures 7 and 8 respectively.

It is important to experiment with the 5-state ring world to investigate the effect of λ on a problem that can be solved with TD($\lambda = 0$). For all of the $\lambda > 0$ values that were used, RMSE $< .05$ was achieved in under 10 000 time steps. As before, increasing λ reduced the amount of data that was required to reach a the same error level. In the extreme TD(0) case, the model will not reach RMSE $< .05$ until over 150 000 time steps have passed.

The number of nodes in fully symmetric question networks rises exponentially with the depth of the network, making it quite costly to make longer predictions. Through experimentation, we have found a smaller question network that can represent the ring world. This question network is shown in Figure 6, and scales linearly with the number of states in the ring. In this experiment, TD(0) could not find a solution to the 8-state ring world in any of the configurations that we tried. Although not shown, we continued the experiment and even after ten million steps the TD(0) networks did not improve. TD($\lambda > 0$) was able to solve this problem for all values of λ that we tried. Again, increasing λ decreases the amount of data required by the algorithm.

Although not presented in detail here, we have seen similar improvements with TD(λ) vs TD(0) on other problems such as the partially-observable random walk world and the float-reset problem (Sutton & Tanner, 2005; Littman et al., 2002).

Complexity of the cycle and ring worlds

The domains we test in this work behave deterministically, but have extreme state aliasing. It is conceivable that these environments are trivial, and that our success is not encouraging. To test this theory, we have attempted to learn POMDP models of these three environments using the EM (Baum-Welch) algorithm.[1]

	RMSE (amount of data)	
Problem	EM	TD(1)
6 State Cycle	.313 (10 000)	.05 (5 000)
5 State Ring	.37 (10 000)	.05 (5 000)
8 State Ring	.28 (250 000)	.05 (125 000)

Table 1. RMSE of EM and TD(1) algorithm on the 6-state cycle, 5-state ring, and 8-state ring worlds. Error is calculated by comparing the learned POMDP observation predictions to the true probabilities over a 1000 step test sequence.

For each domain, we provided EM with the correct number of nominal states and ran 100 trials of 20 iterations each. In each case we provided the EM algorithm with more data than the TD(1) learning algorithm required. In Table 1 we compare the minimum RMSE error achieved of any of the 100 trials with the EM algorithm to our TD(1) results. The

[1]Code graciously provided for us by James et al. who modified code from Murphy (Wolfe et al., 2005; Murphy, 2004).

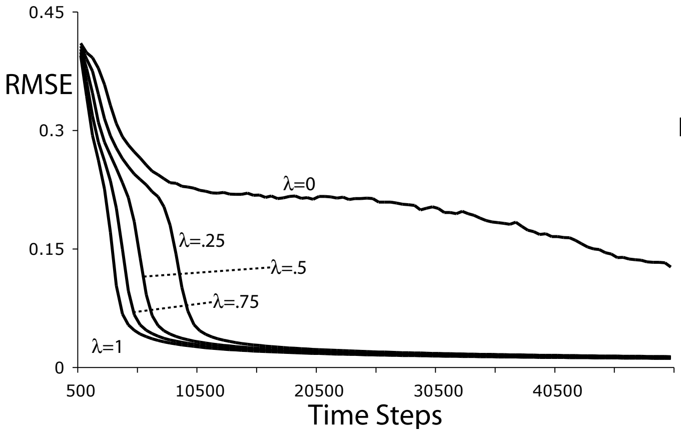

Figure 7. Learning curves of our algorithm for various values of λ on the 5-state ring world. This chart represents the average RMSE over all of the nodes in the TD(λ) network. The question network used is of the form seen in Figure 1, a full, symmetric, action-conditional question network with depth $d = 3$. Each data point in this graph is the average error of the network over 500 time steps. These results are the average of 50 trials.

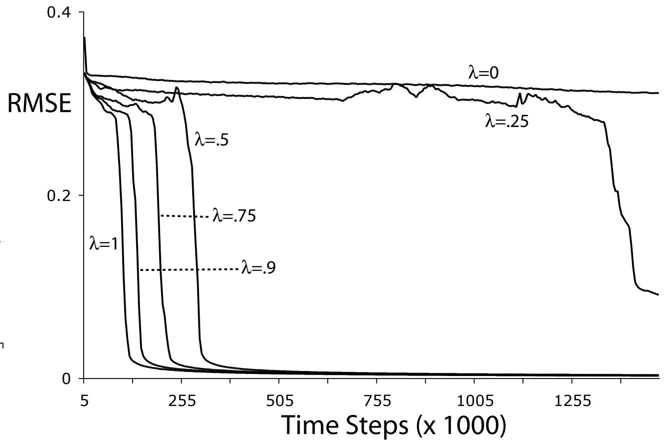

Figure 8. Learning curves of our algorithm for various values of λ on the 8-state ring world. The question network used is of the form in Figure 6 with depth $d = 8$. This chart represents the average RMSE over all of the nodes in the TD(λ) network. Each data point in this graph is the average error of the network over 5000 time steps. These results are the average of 50 trials.

TD network model does considerably better than the model learned with EM in all cases. The results strongly suggest that the cycle and ring worlds we have used are not trivial. Note that very little time was spent tuning the parameters of the EM algorithm; these results are not meant in any way to suggest TD networks are superior or inferior to learning POMDPs with EM.

5. The Cost of λ

The benefits that are gained by using TD(λ) over TD(0) must come at a cost. In our algorithm, memory and computation resource usage grows at approximately the same rate, we will refer to them collectively as "cost". In this section, we will consider two different families of question network that hilight the additional cost of TD(λ) networks over TD(0) networks.

First, consider an unconditioned question network like the one shown in Figure 4, but of an arbitrary depth, d. We will call this the "chain" question network. Second, consider an action-conditional question network like that in Figure 1, but with arbitrary branching factor (number of actions) b and depth d. We call this the "tree" question network.

The largest computational cost of either algorithm is the number of weight updates that are performed at each time step. Other book-keeping costs are negligible and are not included in our analysis.

With an unconditional "chain" question network, the number of updates for TD(λ) at each step will be $\frac{d(d+1)}{2}$ where d is the depth of the chain. In a TD(0) network of the same depth, the number of updates at each step is d. The ratio of the cost of TD(λ) over TD(0) gives us a measure of the factor of additional cost of TD(λ). This additional cost factor is:

$$\frac{Computation(TD(\lambda))}{Computation(TD(0))} = \frac{d(d+1)}{2d} = \frac{d+1}{2}$$

This is an upper bound on the additional cost that will be incurred by TD(λ) networks for any *single-target* question network. Note that this is a degenerate case, where the length of the longest question is equal to the number of nodes in the network, and every prediction always has a target. In practice, we will ask a variety of action-conditional questions of different lengths, and they will not always have an answer. For this reason, our primary interest is not the "chain" network, it is the "tree" question network. We have derived that the factor of additional work in this case is:

$$\frac{Computation(TD(\lambda))}{Computation(TD(0))} = \frac{b}{b-1} - \frac{d}{b^d - 1}$$

where b is the number of actions and d is the depth of the question network. This equation holds as long as $b > 1$ and $d >= 1$. When $b = 2$, this ratio quickly rises from 1.0 at $d = 1$ to 2.0 as the depth of the network increases. In the other dimension, as b increases, this ratio quickly goes to $1.0 + \epsilon$, a negligible amount of extra work.

In future applications of TD(λ) networks, we expect that a variety of question networks will be used. The topology of these networks (on average) will fall somewhere between the question networks that we have discussed above. In practice, when the question network is not a full tree, the number of node updates per step will depend on the policy being followed. For example, in some of our ring world experiments, we used a network with $d = 8$ which had only 16 nodes. In this case, our formulae above predict that for $d = 8$, the number of node updates will be 16 for a symmetric tree and 36 for a chain network. In our experience the average number of node updates per step for this question network is 14.5, less than either of our estimates.

We propose one modification to our algorithm that would mitigate the additional cost of TD(λ) (if required) in degenerate cases. In our algorithm, the target v_t^i is based on the full parental hierarchy of the prediction, right up to the observation bit. One simple change to the algorithm could cut off these traces after they have used some bounded number of targets. This "bounded lookahead" parameter would allow the number of updates per step to be tuned between d and $\frac{d(d+1)}{2}$ easily to suit any particular situation.

6. Conclusions

In all of the experiments we have run, TD(λ) networks with $\lambda > 0$ have learned faster than with $\lambda = 0$. This is strong evidence that our generalized TD(λ) network learning algorithm is an improvement over the existing TD(0) learning algorithm. TD(λ) networks have also solved problems that were not solvable with the TD(0) learning algorithm. We believe that these problems are solvable because the multi-step backups of the TD(λ) learning algorithm help eliminate information flow dependencies between the question and answer networks. The cost of the TD(λ) network learning algorithm is less than twice that of the TD(0) algorithm for the types of questions that are important to represent a model of a controlled dynamical system (action-conditional questions). For some other question networks (the chain network) the additional cost is larger, but with simple techniques such as adding a bounded lookahead parameter we can easily control this cost.

In the conventional TD(λ) learning algorithm, no single value of λ is always best. It is surprising that our experiments suggest $\lambda = 1$ is better than any other value for λ. This disparity may be related to the class of problems that we are using, partially-observable (non-Markov) environments. In reinforcement learning, the value of λ can be thought of as a parameter to specify a mixture of TD and Monte Carlo backups. TD is more data efficient and requires less computation while Monte Carlo is more robust in non-Markov environments. In our previous work we reported that TD(0) backups were better than Monte Carlo backups for TD network learning in certain Markov environments (Sutton & Tanner, 2005). In our TD(λ) work, we have only considered partially-observable environments, a situation that favors Monte Carlo or TD(1) backups. It is intuitive that in this case, learning favors higher values of λ.

Acknowledgments

The authors gratefully acknowledge the ideas and encouragement they have received in this work from Satinder Singh, Doina Precup, Michael Littman, Mark Ring, Eddie Rafols, David Silver, Peter McCracken and all the members of the rlai.net group. Special thanks to Britton Wolfe, Michael James and Satinder Singh for sharing their EM code for POMDP learning. This work was supported in part by NSERC and iCORE.

References

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 233–246.

Jaeger, H. (1998). *Discrete-time, discrete-valued observable operator models: a tutorial* (Technical Report). German National Research Center for Information Technology.

Littman, M., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.

Murphy, K. (2004). Hidden markov model (hmm) toolkit for matlab. http://www.cs.ubc.ca/ murphyk/software/hmm/hmm.html.

Rivest, R. L., & Schapire, R. E. (1990). A new approach to unsupervised learning in deterministic environments. *Machine learning: an artificial intelligence approach volume III*, 670–684.

Singh, S., James, M. R., & Rudary, M. R. (2004). Predictive state representations: A new theory for modeling dynamical systems. *Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference* (pp. 512–519).

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

Sutton, R. S., & Tanner, B. (2005). Temporal-difference networks. *Advances in Neural Information Processing Systems 17* (pp. 1377–1384). Cambridge, MA: MIT Press.

Tanner, B., & Sutton, R. S. (2005). Temporal-difference networks with history. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*.

Wolfe, B., James, M. R., & Singh, S. (2005). Learning predictive state representations in dynamical systems without reset. *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*.