TD Networks

Richard S. Sutton and Brian Tanner
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{sutton,btanner}@cs.ualberta.ca

Abstract

We introduce a generalization of temporal-difference (TD) learning to networks of interrelated predictions. Rather than relating a single prediction to itself at a later time, as in conventional TD methods, a TD network relates each prediction in a set of predictions to other predictions in the set at a later time. TD networks can represent and apply TD learning to a much wider class of predictions than has previously been possible. Using a random-walk example, we show that these networks can be used to learn to predict by a fixed interval, which is not possible with conventional TD methods. Secondly, we show that when actions are introduced, and the inter-prediction relationships made contingent on them, the usual learning-efficiency advantage of TD methods over Monte Carlo (supervised learning) methods becomes particularly pronounced. Thirdly, we demonstrate that TD networks can learn predictive state representations that enable exact solution of a non-Markov problem. A very broad range of inter-predictive temporal relationships can be expressed in these networks. Overall we argue that TD networks represent a substantial extension of the abilities of TD methods and bring us closer to the goal of representing world knowledge in entirely predictive, grounded terms.

Temporal-difference (TD) learning is widely used in reinforcement learning methods to learn moment-to-moment predictions of total future reward (value functions). In this setting, TD learning is often simpler and more data-efficient than other methods. But the idea of TD learning can be used more generally than it is in reinforcement learning. TD learning is a general method for learning predictions whenever multiple predictions are made of the same event over time, value functions being just one example. The most pertinent of the more general uses of TD learning have been in learning models of an environment or task domain (Dayan, 1993; Kaelbling, 1993; Sutton, 1995; Sutton, Precup & Singh, 1999). In these works, TD learning is used to predict future values of many observations or state variables of a dynamical system.

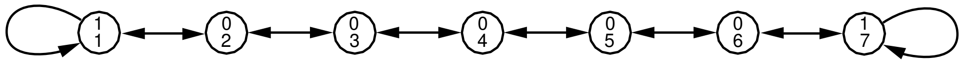
The essential idea of TD learning can be described as "learning a guess from a guess". In all previous work, the two guesses involved were predictions of the same quantity at two points in time, for example, of the discounted future reward at successive time steps. In this paper we explore a few of the possibilities that open up when the second guess is allowed to be different from the first.

To be more precise, we must make a distinction between the extensive definition of a prediction, expressing its desired relationship to measurable data, and its TD definition, expressing its desired relationship to other predictions. In reinforcement learning, for example, state values are extensively defined as an expectation of the discounted sum of future rewards, while they are *TD defined* as the solution to the Bellman equation (a relationship to the expectation of the value of successor states, plus the immediate reward). It's the same prediction, just defined or expressed in different ways. In past work with TD methods, the TD relationship was always among predictions with identical or very similar extensive semantics. In this paper we retain the TD idea of learning predictions based on others, but allow the predictions to have different extensive semantics.

1 The Learning-to-predict Problem

The problem we consider in this paper is a general one of learning to predict aspects of the interaction between a decision making agent and its environment. 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 $a_t \in \mathcal{A}$. Whereas $\mathcal{A}$ is an arbitrary discrete set, we assume without loss of generality that o_t can be represented as a vector of bits. The action and observation events occur in sequence, $a_1, o_1, a_2, o_2, a_3 \cdots$, with each event of course dependent only on those preceding it. This sequence will be called *experience*. We are interested in predicting not just each next observation but more general, action-conditional functions of future experience, as discussed in the next section.

In this paper we use a random-walk problem with seven states, with *left* and *right* actions available in every state:



The observation upon arriving in a state consists of a special bit that is 1 only at the two ends of the walk plus, in the first two of our three experiments, an explicit indication of the state number. This is a *continuing* task: reaching an end state does not end or interrupt experience. Although the sequence depends deterministically on action, we assume the actions are selected randomly with equal probability so the overall system can be viewed as a Markov chain.

The TD networks introduced in this paper can represent a wide variety of predictions, far more than can be represented by a conventional TD predictor. In this paper we take just a few steps toward more general predictions. In particular, we consider variations of the problem of prediction by a fixed interval. This is one of the simplest cases that cannot otherwise be handled by TD methods. For the seven-state random walk, we will predict the special observation bit some numbers of discrete steps in advance, first unconditionally and then conditioned on action sequences.

2 TD Networks

A *TD network* is a network of nodes, each representing a single scalar prediction. The nodes are interconnected by links representing the TD relationships among the predictions and to the observations and actions. These links determine the extensive semantics of each prediction—its desired or target relationship to the data. They represent *what* we seek to predict about the data as opposed to *how* we try to predict it. We think of these links as determining a set of *questions* being asked about the data, and accordingly we call them the *question network*. A separate set of interconnections determine the actual computational process—the updating of the predictions at each node from their previous values and the current action and observation. We think of this process as providing the *answers* to the questions, and accordingly we call them the *answer network*. The question network provides targets for a learning process shaping the answer network and does not otherwise affect the behavior of the TD network. It is natural to consider changing the question network, but in this paper we take it as fixed and given.

Figure 1a shows a suggestive example of a question network. The three squares across the top represent three observation bits. The node labeled 1 is directly connected to the first observation bit and represents a prediction that that bit will be 1 on the next time step. The node labeled 2 is similarly a prediction of the expected value of node 1 on the next step. Thus the extensive definition of Node 2's prediction is the probability that the first observation bit will be 1 two time steps from now. Node 3 similarly predicts the first observation bit three time steps in the future. Node 4 is a conventional TD prediction, in this case of the future discounted sum of the second observation bit, with discount parameter γ. Its target is the familiar TD target, the data bit plus the node's own prediction on the next time step. Nodes 5 and 6 predict the probability of the third observation bit being 1 *if* particular actions *a* or *b* are taken respectively. Node 7 is a prediction of the average of the first observation bit and Node 4's prediction, both on the next step. This is the first case where it is not easy to see or state the extensive semantics of the prediction in terms of the data. Node 8 predicts another average, this time of nodes 4 and 5, and the question it asks is even harder to express extensively. One could continue in this way, adding more and more nodes whose extensive definitions are difficult to express but which would nevertheless be completely defined as long as these local TD relationships are clear. The thinner links shown entering some nodes are meant to be a suggestion of the entirely separate answer network determining the actual computation (as opposed to these goals) of the network. In this paper we consider only simple question networks such as the left column of Figure 1a and of the action-conditional tree form shown in Figure 1b.

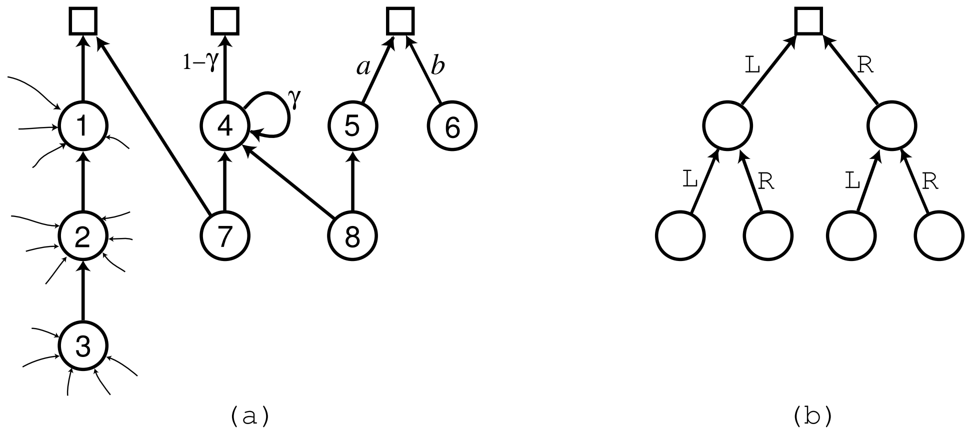


Figure 1: The question networks of two TD networks. (a) a question network discussed in the text, and (b) a depth-2 fully-action-conditional question network used in Experiments 2 and 3. Observation bits are represented as squares across the top while actual nodes of the TD network, corresponding each to a separate prediction, are below. The thick lines represent the question network and the thin lines in (a) represent the answer network. Only a small portion of the answer network is shown.

More formally and generally, let $y_t^i \in [0,1]$, $i = 1, \ldots, n$ denote the prediction of the ith node at time step t. The column vector of predictions $\mathbf{y}_t = (y_t^1, \ldots, y_t^n)^T$ is updated according to a vector-valued function $\mathbf{u}$ with modifiable parameter $\mathbf{W}$:

$$\mathbf{y}_t = \mathbf{u}(\mathbf{y}_{t-1}, a_t, o_t, \mathbf{W}_t). \tag{1}$$

The function $\mathbf{u}$ corresponds to the answer network, with $\mathbf{W}$ being the weights on its links. Before detailing that process, we turn to the question network, the defining TD relationships between nodes. The TD target z_t^i for y_t^i is an arbitrary function z^i of the successive predictions and observations. In vector form we have[1]

$$\mathbf{z}_t = \mathbf{z}(o_{t+1}, \tilde{\mathbf{y}}_{t+1}) \in \Re^n, \tag{2}$$

where $\tilde{\mathbf{y}}_{t+1}$ is just like $\mathbf{y}_{t+1}$, as in (1), except calculated with the *old* weights before they are updated on the basis of $\mathbf{z}_t$:

$$\tilde{\mathbf{y}}_{t+1} = \mathbf{u}(\mathbf{y}_t, a_{t+1}, o_{t+1}, \mathbf{W}_t) \in \Re^n. \tag{3}$$

(This temporal subtlety also arises in conventional TD learning.) For example, for the nodes in Figure 1a we have $z_t^1 = o_{t+1}^1$, $z_t^2 = y_{t+1}^1$, $z_t^3 = y_{t+1}^2$, $z_t^4 = o_{t+1}^2 + \gamma y_{t+1}^4$, $z_t^5 = \frac{1}{2}o_{t+1}^3 + \frac{1}{2}y_{t+1}^4$, and $z_t^6 = \frac{1}{2}y_{t+1}^3 + \frac{1}{2}y_{t+1}^4$. These target functions z^i are only part of specifying the question network. The other part has to do with making them potentially conditional on action and observation. For example, one might want to predict "if I step right I will observe a 1". To arrange for such semantics we introduce a new vector $\boldsymbol{\beta}_t$ whose components β_t^i indicate the extent to which y_t^i is held responsible for matching z_t^i, thus making the ith prediction conditional on β_t^i. β_t^i is determined as an arbitrary function β^i of a_t and o_t. In vector form we have:

$$\boldsymbol{\beta}_t = \boldsymbol{\beta}(a_{t+1}, o_{t+1}, \tilde{\mathbf{y}}_{t+1}) \in [0,1]^n. \tag{4}$$

Equations (1–4) correspond to the *question* network. Let us now turn to defining $\mathbf{u}$, the update function for $\mathbf{y}_t$ mentioned earlier and which corresponds to the *answer* network. In general $\mathbf{u}$ is an arbitrary function approximator, but for concreteness we define it to be of a linear form

$$\mathbf{y}_t = \boldsymbol{\sigma}(\mathbf{W}_t \mathbf{x}_t) \tag{5}$$

where $\mathbf{x}_t \in \Re^m$ is a feature vector, $\mathbf{W}_t$ is a $n \times m$ matrix, and σ is the n-vector form of the identity function (Experiments 1 and 2) or the S-shaped logistic function $\sigma(s) = \frac{1}{1+e^{-s}}$ (Experiment 3). The feature vector is an arbitrary function of the preceding action, observation, and node values:

$$\mathbf{x}_t = \mathbf{x}(a_t, o_t, \mathbf{y}_{t-1}) \in \Re^m. \tag{6}$$

The learning algorithm for each component w_t^{ij} of $\mathbf{W}_t$ is

$$\Delta w_t^{ij} = \alpha(z_t^i - y_t^i)\beta_t^i \frac{\partial y_t^i}{\partial w_t^{ij}}, \tag{7}$$

where α is a step-size parameter. The timing details may be clarified by writing the sequence of quantities in the order in which they are computed:

$$\mathbf{y}_{t-1}\; a_t\; o_t\; \mathbf{x}_t\; \tilde{\mathbf{y}}_t\; \mathbf{z}_{t-1}\; \boldsymbol{\beta}_{t-1}\; \mathbf{W}_t\; \mathbf{y}_t\; a_{t+1}\; o_{t+1}\; \mathbf{x}_{t+1}\; \tilde{\mathbf{y}}_{t+1}\; \mathbf{z}_t\; \boldsymbol{\beta}_t\; \mathbf{W}_{t+1}\; \mathbf{y}_{t+1}. \tag{8}$$

Finally, the target in the extensive sense for $\mathbf{y}_t$ is

$$\mathbf{y}_t^* = E_{t,\pi}\left\{(1 - \boldsymbol{\beta}_t) \cdot \mathbf{y}_t^* + \boldsymbol{\beta}_t \cdot \mathbf{z}(o_{t+1}, \mathbf{y}_{t+1}^*)\right\}, \tag{9}$$

where $\cdot$ represents component-wise multiplication and π is the policy being followed, which is assumed fixed.

[1]In general, $\mathbf{z}$ is a function of all the future predictions and observations, but in this paper we treat only the one-step case.

3 Experiment 1: n-step Unconditional Prediction

In this experiment we sought to predict the observation bit precisely n steps in advance, for $n = 1$, 2, 5, 10, and 25. In order to predict n steps in advance, of course, we also have to predict $n - 1$ steps in advance, $n - 2$ steps in advance, etc., all the way down to predicting one step ahead. This is specified by a TD network consisting of a single chain of predictions like the left column of Figure 1a, but of length 25 rather than 3. We constructed random-walk experience sequences by starting at the center state and taking random actions for some number of time steps, the *sequence length*. We constructed 100 sequences of each length 50, 100, 150, and 200.

We then applied a TD network and a corresponding Monte Carlo method to this data. The Monte Carlo method learned the same predictions, but learned them by comparing them to the actual outcomes in the sequence (instead of z_t^i in (7)). This involved significant additional complexity to store the predictions until their corresponding targets were available. Both algorithms used feature vectors of 7 binary components, one for each of the seven states, all of which were zero except for the one corresponding to the current state. Both algorithms formed their predictions linearly ($\sigma(\cdot)$ was the identity) and unconditionally ($\beta_t^i = 1 \ \forall i, t$).

In an initial set of experiments, both algorithms were applied online with a variety of values for their step-size parameter α. Under these conditions we did not find that either algorithm was clearly better in terms of the mean square error in their predictions over the data sets. We found a clearer result when both algorithms were trained using batch updating, in which weight changes are collected "on the side" over an experience sequence and then made all at once at the end, and the whole process is repeated until convergence. Under batch updating, convergence is to the same predictions regardless of initial conditions or α value (as long as α is sufficiently small), which greatly simplifies comparison of algorithms. The predictions learned under batch updating are also the same as would be computed by least squares algorithms such as LSTD(λ) (Bradtke & Barto, 1996; Boyan, 2000; Lagoudakis & Parr, 2003).

The root-mean-square error (RMSE) in the predictions of each algorithm are shown in Table 1. For the 1-step predictions, the two methods perform identically, of course, but for longer predictions there is a significant difference. The RMSE of the Monte Carlo method consistently increases with prediction length whereas for the TD network it decreases with prediction length. The largest standard error in any of the numbers shown in the table is 0.008, so almost all of the differences shown are statistically significant. TD methods appear to have a significant data-efficiency advantage over non-TD methods in this prediction-by-n context (and this task) just as they do in conventional multi-step prediction problems (Sutton, 1988).

Sequence	1-step	2-step		5-step		10-step		25-step	
length	MC/TD	MC	TD	MC	TD	MC	TD	MC	TD
50	0.205	0.219	0.172	0.234	0.159	0.249	0.139	0.297	0.129
100	0.124	0.133	0.100	0.160	0.098	0.168	0.079	0.187	0.068
150	0.089	0.103	0.073	0.121	0.076	0.130	0.063	0.153	0.054
200	0.076	0.084	0.060	0.109	0.065	0.112	0.056	0.118	0.049

Table 1: RMSE of Monte-Carlo and TD-network predictions of various lengths and with various amounts of training data on the random-walk example.

4 Experiment 2: Action-conditional Prediction

The advantage of TD methods should be greater for predictions that apply only when the experience sequence unfolds in a particular way, such as when a particular sequence of actions are made. In a second experiment we sought to learn n-step-ahead predictions conditional on action selections. The question network for learning all 2-step-ahead predictions is shown in Figure 1b. The upper two nodes predict the observation bit conditional on taking a left action (L) or a right action (R). The lower four nodes correspond to the two-step predictions, e.g., the second lower node is the prediction of what the observation bit will be if an L action is taken followed by an R action. These predictions are the same as the *e-tests* used in some of the work on predictive state representations (PSRs) (Littman, Sutton & Singh, 2002; Ruddery & Singh, 2003).

In this experiment we used a question network like that in 1b except of depth four, consisting of 30 (2+4+8+16) nodes. The βs were set to condition on a single action selection as suggested by the figure. The feature vectors were as in the previous experiment. Now that we are conditioning on action, the problem is deterministic and α can be set uniformly to 1. A Monte Carlo prediction can be learned only when its corresponding action sequence occurs in its entirety, but then it is complete and accurate in one step. The TD network, on the other hand, can learn from incomplete sequences but must propagate them back one-level at a time. First the one-step predictions must be learned, then the two-step predictions from them, and so on. The results for online and batch training are shown in Tables 2 and 3.

As anticipated, the TD network learns much faster than Monte Carlo with both online and batch updating. Because the TD network learns its n step predictions based on its $n-1$ step predictions, it has a clear advantage for this task. Once the TD Network has seen each action in each state, it can quickly learn any prediction 2, 10, or 1000 steps in the future. Monte Carlo, on the other hand, must sample actual sequences, so each exact action sequence must be observed.

	1-Step	2-Step		3-Step		4-Step	
Time Steps	MC/TD	MC	TD	MC	TD	MC	TD
100	0.153	0.222	0.182	0.253	0.195	0.285	0.185
200	0.019	0.092	0.044	0.142	0.054	0.196	0.062
300	0.000	0.040	0.000	0.089	0.013	0.139	0.017
400	0.000	0.019	0.000	0.055	0.000	0.093	0.000
500	0.000	0.019	0.000	0.038	0.000	0.062	0.000

Table 2: Online performance. RMS error of 1, 2, 3, and 4 step action-conditional predictions of Monte-Carlo and TD-network methods at 100-step checkpoints.

Sequence length	MC	TD
50	53.48%	17.21%
100	30.81%	4.50%
150	19.26%	1.57%
200	11.69%	0.14%

Table 3: Batch performance. Percentage of incorrect action-conditional predictions of batch-updating Monte-Carlo and TD-network methods, after various amounts of data, on the random-walk task. All differences are highly statistically significant.

5 Experiment 3: Learning a Predictive State Representation

Experiments 1 and 2 showed advantages for TD learning methods in Markov problems. The feature vectors in both experiments provided complete information about the nominal state of the random walk. In Experiment 3, on the other hand, we applied TD networks to a non-Markov version of the random-walk example, in particular, in which only the special observation bit was visible and not the state number. In this case it is not possible to make accurate predictions solely based on the current action and observation; the previous time step's predictions must be used as well. As in the previous experiment, we sought to learn n-step predictions using action-conditional question networks of depths 2, 3, and 4. The feature vector consisted of three components: a constant 1, four binary features to represent which *pair* of action and observation bit occurred, and n more features corresponding to the components of $\mathbf{y}_t$. The features vectors were thus of length $m = 11, 19,$ and 35 for the three depths. In this experiment $\sigma(\cdot)$ was the S-shaped logistic function.

We constructed sequences of 250,000 time steps and presented them to each of the three networks, then repeated everything for 50 different runs. The initial weights $\mathbf{W}_0$ and predictions $\mathbf{y}_0$ were both 0. We measured the RMSE of all predictions made by the network (computed from knowledge of the task) and also the empirical RMSE of the one-step prediction for the action taken on each time step. We found that in all cases the errors approached zero over time, showing that the problem was completely solved. Figure 2 shows some representative learning curves for the depth-2 and depth-4 TD networks.

In ongoing experiments on other non-Markov problems we have found that TD networks do not always provide such complete solutions. Other problems have required more than one step of history information (the one-step-preceding action and observation). Other problems seem to require more history, though much less than would be required using history information alone. Our results as a whole suggests that TD networks may provide an effective alternative learning algorithm for PSRs. The only previous learning algorithm for PSRs is the myopic gradient algorithm developed by Singh et al. (2003). That algorithm was also found to be effective on some tasks but not on others. More work is needed to assess the range of effectiveness and learning rate of the two methods, and to explore their combination with history methods.

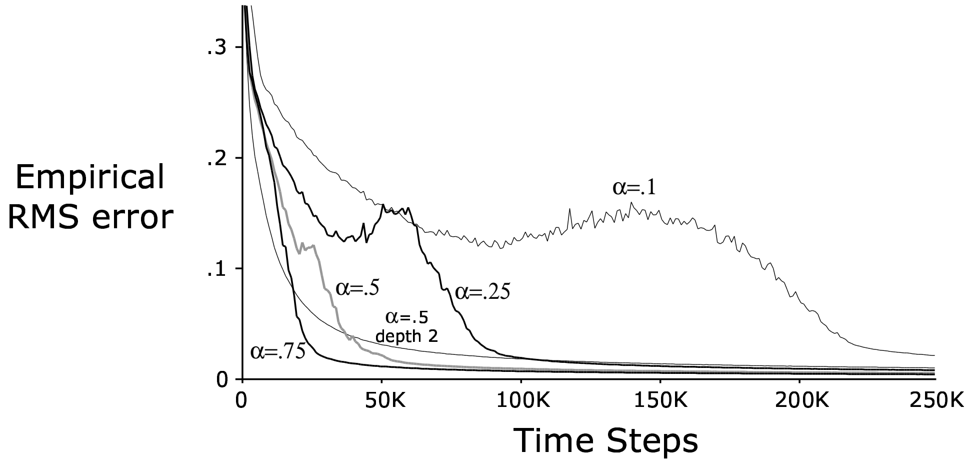


Figure 2: Prediction performance on the non-Markov random walk of of depth-4 TD networks (and one depth-2 network) with various step-size parameters, averaged over 50 runs and 1000 time-step bins

6 Conclusion

TD networks suggest a much larger set of possibilities for learning to predict, and in this paper we have begun exploring just the first few. Our results with the Markov random walk indicate that even in a fully observable setting there may remain significant advantages to TD methods when learning TD-defined predictions such as appear in TD networks. The action-conditional results show that TD methods can learn dramatically faster than other methods. TD networks allow the expression of many new kinds of predictions whose extensive semantics is not immediately clear, but which are ultimately fully grounded in data. We expect it will be fruitful to further explore the expressive potential of TD-defined predictions.

Our final experiment suggests that TD networks may provide a good alternate learning rule for PSRs. Our work here has focused on learning a variety of predictions that may be of interest. We feel that the ability to represent and express in TD form a wide variety of predictions is valuable even in the Markov case. But since we are learning predictions, it is natural to consider using them as state, as in PSRs. Our experiments suggest that this is a promising direction and that TD learning algorithms may have advantages over previous methods. We also note that TD networks may be an interesting way to generate new predictions, by connecting additional nodes with local connections to the TD network, and thus for addressing the discovery problem in PSRs.

Acknowledgments

The authors gratefully acknowledge the ideas and encouragement they have received in this work from Satinder Singh, Doina Precup, Michael Littman, Tao Wang, and all the members of the RLAI.net group.

References

Boyan, J. A. (2000). Technical update: Least-squares temporal difference learning. *Machine Learning*.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1/2/3):33–57.

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624.

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 167–173. Morgan Kaufmann, San Mateo, CA.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research* 4(Dec):1107–1149.

Littman, M. L., Sutton, R. S. and Singh, S. (2002). Predictive representations of state. In *Advances In Neural Information Processing Systems 14*.

Rudary, M. R., Singh, S. (2004). A nonlinear predictive state representation. In *Advances in Neural Information Processing Systems 16*.

Singh, S., Littman, M. L., Jong, N. K., Pardoe, D. and Stone, P. (2003) Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning*.

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

Sutton, R. S. (1995). TD models: Modeling the world at a mixture of time scales. In A. Prieditis and S. Russell (eds.), *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 531–539. Morgan Kaufmann, San Francisco.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 181–211.